# 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

## A standards-based, task-specifc telepresence application 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

> **A standards-based, task-specifc telepresence application 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

**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

**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

**ML** machine learning

**NPM** Node Package Manager

**OCI** Open Container Initiative

**OS** operating system

**PWA** progressive web application

**RFC** request for comments

**RTC** real-time communication

**SDK** software development kit

**SFU** selective forwarding unit

**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

**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. Concepts

## 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 that its development's biggest challenge is 'achieving that sense of "being there."' (Minsky, 1980). Minsky argued from a standpoint concerned mostly with robotic manipulators that performed labour either mediated over a distance or enhanced both the operator's abilities and safety.

The current spectrum of Telepresence is much more diverse. While there are applications of remote robotic control in industry, telemedicine and the military, the most common instance has become the teleconferencing application relaying video and audio streams, as well as allowing chat and collaborative whiteboards.

In this study, the term telepresence is used to explicitly describe a form of virtual or augmented reality that allows multiple people to experience a form of presence and immersion.

## 2.2. Motion capture

The positional tracking of specific key points on a moving body over time is referred to as motion capture.

Motion capture technology is often used in 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 or OptiTrack, which use visual markers to track movement in space and require a studio environment to be deployed. Another markerless optical system is Captury Live, 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 or the Perception Neuron sensor kit. These operate over the radio and are independent of the lighting conditions but tend to produce less accurate data or are subject to interference.

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' (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 with an integrated processing engine or the Orbbec Femto Bolt. 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 or BlazePose 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 sonification of movement data is used in health and therapeutic research to offer an acoustic interface to experience dynamics in movement properties. This can be used for rehabilitation and stabilising movement practice or as a guiding signal within an exercise.

The basic principle for movement sonification is the same as for any data sonification. 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). Still, it can also be achieved using indirect logical constraints (e.g. if multiple thresholds are crossed, a single signal is triggered).

## 2.4. Web standards

The idea behind web standards is meant 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 the interaction 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. There are several organisations that 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).

### 2.4.1. WebRTC

The standard for real-time communication in browsers was initially proposed and mainly developed by Google. It is an official standard since 2019(?). 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 communication, but also as a selective forwarding unit (SFU) enabling one-to-many or many-to-many communication setups.

### 2.4.2. WebSockets

A transmission protocol that was standardised as request for comments (RFC) 6455 by the IETF in 2011 (Fette & Melnikov, 2011). It allows full-duplex communication between client and server, running on the same ports and transport layer 7 as the half-duplex HyperText transmission protocol (HTTP) protocol, thus being compatible with existing web infrastructure.

### 2.4.3. WebAudio

This 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 that is commonly used in context of games or virtual reality experiences is the possiblity of placing sound sources in virtual soundstages that are then rendered as ambisonics for psychoacoustics in headphones.

### 2.4.4. WebXR

The various virtual and augmented reality devices available are made accessible via the WebXR API. The browser manages the communication with the headset and 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.

### 2.4.5. WebBluetooth

This rather simple API provides access to the computer's Bluetooth functionality. It allows connecting custom Bluetooth senders like Arduinos or other embedded devices with sensors or other do-it-yourself (DIY) electronics sending and receiving messages to and from the browser.

## 2.5. Application design paradigms

### 2.5.1. Single page applications

The idea of a 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.

### 2.5.2. Progressive web applications

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.

### 2.5.3. Real-time web applications

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.

## 2.6. Application deployment

### 2.6.1. Containerisation

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, 2024b) It was popularised through the release of the Docker Engine, 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 subsequently received Docker's container runtime and format as a donation, which was released as runC version 1.0 in 2020 (Linux Foundation, 2024). 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)

### 2.6.2. Container orchestration

'Container orchestration automates the provisioning, deployment, networking, scaling, availability, and lifecycle management of containers.' (IBM, 2024a) The concept first gained popularity as Docker Swarm, which is a functionality of the Docker software, but its most successful instance so far is as the software package Kubernetes, 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.

# 3. Tools

## 3.1. Web development languages

Table 3.1.: Ranking among the most used languages on GitHub (Daigle & GitHub, 2023)

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

### 3.1.1. JavaScript

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, 2024).

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).

### 3.1.2. TypeScript

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.

## 3.2. Native application development

### 3.2.1. Node

Released initially by developer Ryan Dahl in 2009, 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.

### 3.2.2. Python

Guido van Rossum created the Python programming language in 1990 (Python Software Foundation, 2024). 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.

### 3.2.3. C/C++

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 opens-source hardware platforms, 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.

### 3.2.4. 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 the majority of contemporary electronics, it rose to wider awareness through its popularity in the 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 be the 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 line of Arduino development boards and its software ecosystem went on to become the most popular framework for experimenting with open-source hardware and microcontrollers outside of the field of electronic engineering, while there are other successful projects like Adafruit Industries, SparkFun, RaspberryPI and many more.

## 3.3. 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.

[1]Greif and Burel, 2023b

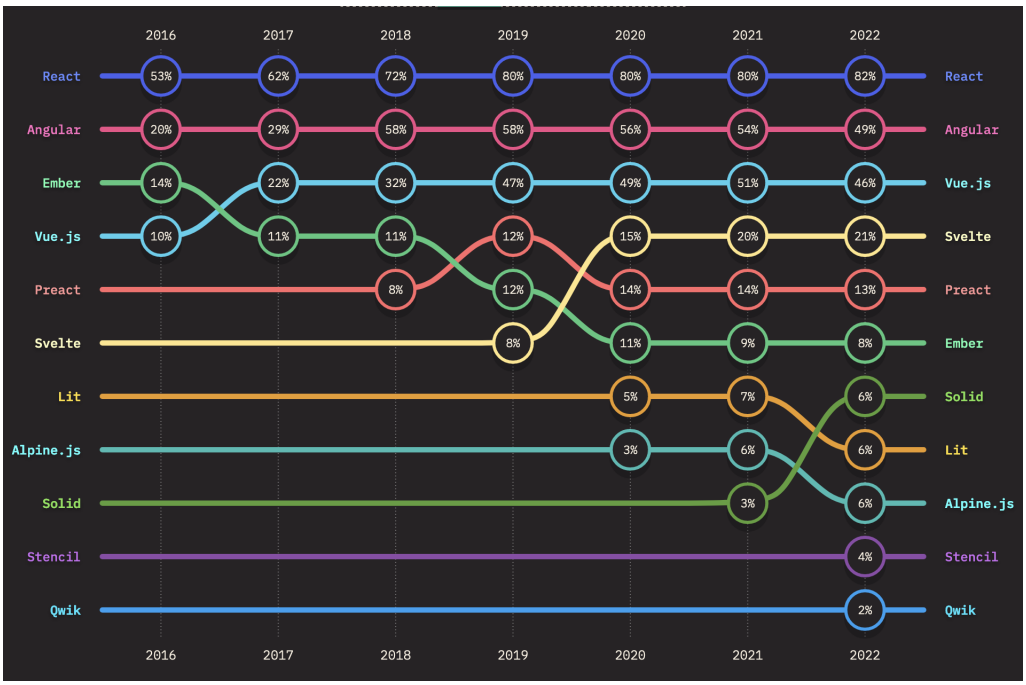


Figure 3.1.: State of JS: Most used frontend frameworks in 2022[1]

### 3.3.1. React

React (https://react.dev/), 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 (Overflow, 2023) and the State Of JS (Greif & Burel, 2023b) 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 (https://nextjs.org/) for the web and ReactNative (https://reactnative.dev/) 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.

### 3.3.2. Vue

Vue (https://vuejs.org/) 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 build 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 (https://quasar.dev/) and Nuxt (https://nuxt.com/) that enable even more sophisticated workflows for application development and deployment.

### 3.3.3. Angular

Angular was initially released by Google in 2010 as AngularJS and officially discontinued in 2022 (https://angularjs.org/). A completely overhauled and currently used version 2 was released in 2016 and maintained by Google. It is different from React and Vue in that it is a complete framework that contains 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.

## 3.4. Backend libraries

### 3.4.1. Express

The Express JS framework provides the basic functionality to create web servers, including routing and middleware functionality. TJ Holowaychuk developed and sold it to StrongLoop, 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, 2023a). 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 are built on top of it that add more opinionated structures or extensions.

### 3.4.2. Koa

Billed as a successor to Express, Koa 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.

## 3.5. Backend frameworks

Other frameworks and a more stringent and structured application structure might be more desirable for complex applications. There are numerous JS frameworks, some based on Express or Koa, and others 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 3.2.: State of JS survey: Most used backend frameworks (Greif & Burel, 2023a)

| Framework | % of question respondents |
|---|---|
| Nest | 30.2 |
| Feathers | 8.8 |
| Meteor | 2.7 |

### 3.5.1. Nest

Nest 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 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, 2023a).

### 3.5.2. Feathers

This 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 also abstracted and can be deployed using different WebSockets

libraries. It also provides a CLI to generate application code that can be written 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).

### 3.5.3. Meteor

Meteor focuses explicitly on real-time applications using WebSockets. The framework is a bit of an outlier in that 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 to debate regarding its ease of use versus its ‘growing pains’ and related trouble with wide adoption (doppp & forum users, 2019).

## 3.6. Virtual and augmented reality

### 3.6.1. THREE.js

This 3D graphics framework with a large community with over 1800 contributors has been around since it was publicly released by Ricardo Cabello in 2010. It features an extensive toolset for graphics generation, rendering and effects and relies on the WebGL standard to allow performant rendering via local graphics hardware.

### 3.6.2. A-Frame

Based on THREE.js, this framework allows the developer to create 3D scenes by composing custom HTML elements that provide geometric primitives, lights, cameras, etc. This way, it has a low entry barrier for people coming from web development with limited scripting experience. It explicitly focuses on mixed reality applications, implements the WebXR standard and controls for various headsets and controllers.

### 3.6.3. Resonance

Resonance was developed by Google-based Omnitone, another Google project 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 any WebAudio-compliant audio framework.

### 3.6.4. Howler.js

Howler is a more complete audio framework that builds on the WebAudio API and provides easier access to audio functionality. It offers spatial audio as a plugin, but at the time of writing, only supports connecting live audio sources through a yet unmerged pull request on GitHub (see: https://github.com/goldfire/howler.js/pull/1634).

## 3.7. WebRTC

Quite a number of 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 explicitly focus on this type of topology.

Table 3.3.: WebRTC servers ranked by stars received on GitHub

| WebRTC Server | Stars (k) | Year of initial commit |
|---|---|---|
| Janus Gateway | 7.6 | 2014 |
| LiveKit | 6.4 | 2020 |
| Mediasoup | 5.7 | 2014 |
| Jitsi Videobridge | 2.8 | 2013 |

### 3.7.1. LiveKit

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, 2024). 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.

### 3.7.2. Mediasoup

Mediasoup can be placed at an opposite end of the spectrum regarding high-level versus low-level frameworks. 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 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.

## 3.8. Databases

### 3.8.1. MongoDB

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. Its focus on storing documents of any kind can become problematic since it can lead to inconsistent data very easily if appropriate care isn't applied in the application development (it supports schema validation, but that is not mandatory). However, this loose schematic handling can be beneficial if the application data can't be adequately modelled from the get-go and is subject to more frequent changes.

### 3.8.2. PostgreSQL

This very widely used database 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.

# 4. Methodology

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

## 4.1. Reference implementation

### 4.1.1. Choice of concepts and tools

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 range presented in chapter 2 and chapter 3.

### 4.1.2. Application development and deployment

The application is implemented in its entirety, documented and packaged. Appropriate test coverage is provided, and the overall time spent is logged in timesheets and categorised by the general work areas. The application's server components are deployed to university

hardware and made available over the internet. The client application is then run on various consumer computer systems.

## 4.2. Quantitative analysis

### 4.2.1. Performance testing

The application's performance is only 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 signal processing works as expected through unit testing and simple testing tasks performed on the application. A practical test using actual performers and dance interaction is beyond the scope of this feasibility study.

### 4.2.2. Time spent

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.

## 4.3. Qualitative analysis

### 4.3.1. Code quality

The code quality is mainly assessed in terms of volume (lines of code), complexity (number of classes and functions, cognitive complexity) and stylistic coherence.

### 4.3.2. Critical reflection

A critical 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 critically evaluate the feasibility and discuss the benefits and drawbacks of establishing a task-specific application from scratch.

# 5. Implementation

## 5.1. Architecture

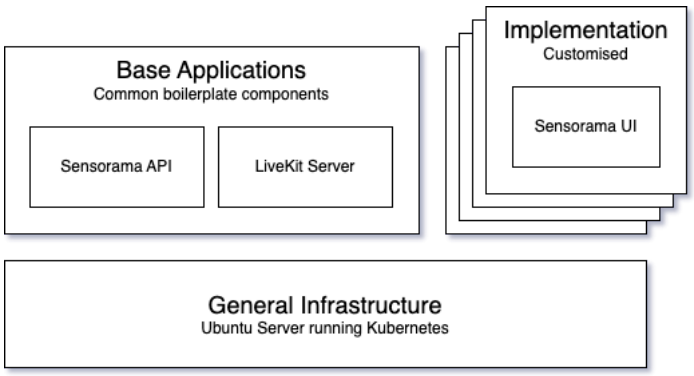


Figure 5.1.: The main components comprising the application architecture

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.

## 5.2. Application infrastructure

While all the frameworks represented in Figure 3.1 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.

## 5.3. 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.

### 5.3.1. Application partitioning

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.

### 5.3.2. Coding style

While the primarily favoured coding paradigm is object-oriented programming, this is only strictly applied to the core functionality. 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. This might be considered bad practice in a streamlined development environment, but it serves the purpose of a modular and somewhat unstable 'single-use' application environment.

### 5.3.3. Testing

Only the core functionality deemed stable and reusable is unit-tested to provide a solid base functionality. The general user interface and data producers are considered transient because they serve a singular use case. These application components should be hackable and replaceable, so they are not tested in the scope of this study. However, more stable and general tools and extensions that warrant a unit testing setup could still be developed in the future.

## 5.4. Application components

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

### 5.4.1. 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.

### 5.4.2. 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.

### 5.4.3. Native utilities

Three different native utilities are additionally implemented.

#### General data producer

This component 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 pointcloud extraction, Open3D 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. Python also allows for easy statistical data analysis using NumPy, which is used for movement quality extraction.

### Captury data producer

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, this producer component has to be implemented separately and uses a C++-based WebSockets server streaming the library's received data.

### Head-tracker

As there was no easily accessible and platform-independent head-tracking equipment, 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.

## 5.5. 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.

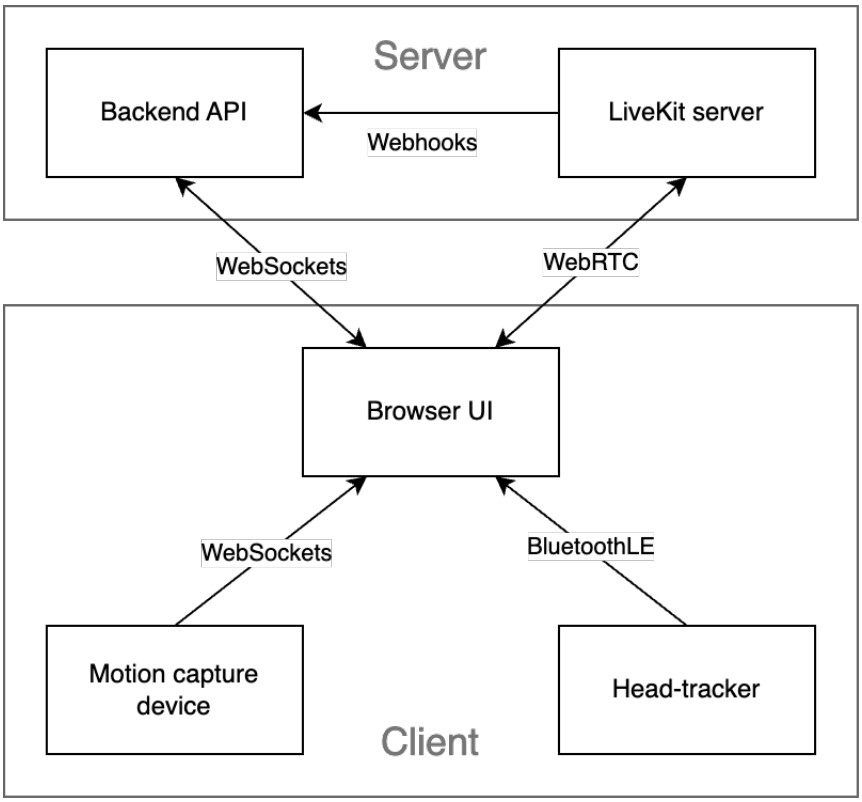


Figure 5.2.: Messaging flow between the application's components

## 5.6. 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.

### 5.6.1. API data models

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.

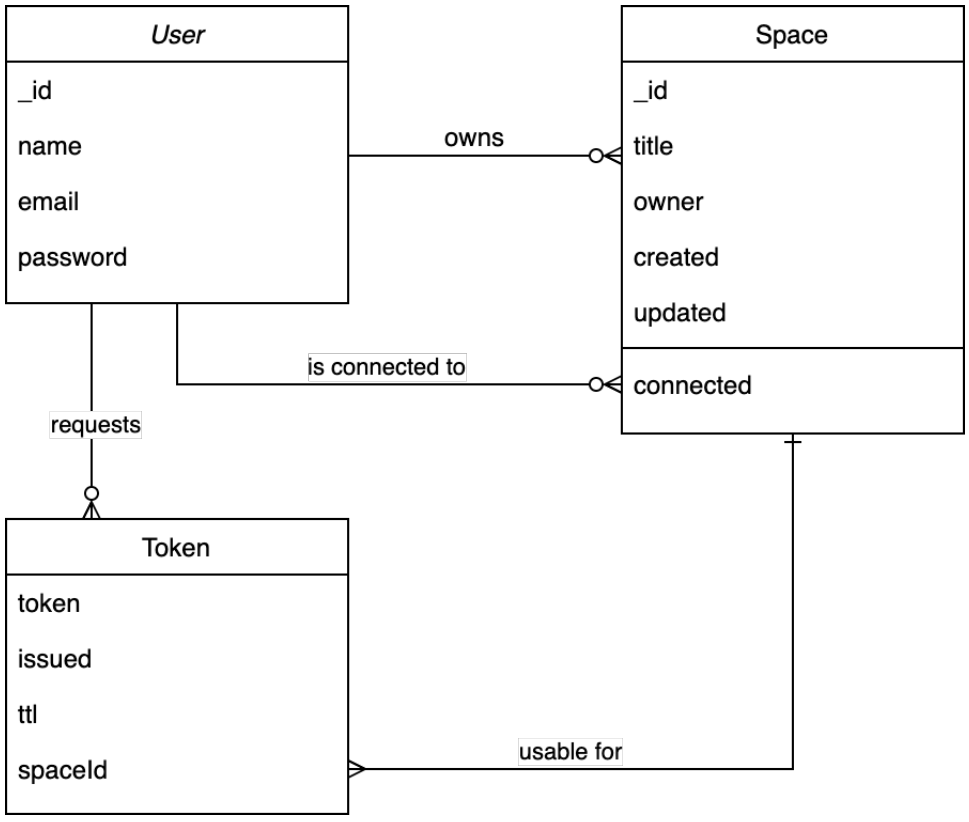


Figure 5.3.: Basic data model used in API server

### 5.6.2. Message format

The data messages are not encoded as JavaScript Object Notation (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. 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 5.4.: The basic message structure for transmitting numeric sensor readings

### 5.6.3. Message types

The message types are stored as JSON. This 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.

```
{
  "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 5.1.: Example pose message schema

# 6. Conclusion

## 6.1. Evaluation results

## 6.2. Critical reflection

## 6.3. Outlook

# Bibliography

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/

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/

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/

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/

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

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

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/

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: 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. (2023b, 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/

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

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

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

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

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

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

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

Overflow, S. (2023, 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

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

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

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

*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/

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

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

# Appendix Listing

# A. Appendix