Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Studiengang Medieninformatik

Bachelorarbeit

von

Thomas **Stangl**

Entwurf und Implementierung einer Webanwendung zur einfachen Fernsteuerung von Digitalmischpulten

Design and Implementation of a Web Application for Simple Remote Control of Digital Mixing Consoles

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Studiengang Medieninformatik

Bachelorarbeit

von

Thomas **Stangl**

Entwurf und Implementierung einer Webanwendung zur einfachen Fernsteuerung von Digitalmischpulten

Design and Implementation of a Web Application for Simple Remote Control of Digital Mixing Consoles

Bearbeitungszeitraum: von 12. Oktober 2021
bis 11. März 2022

1. Prüfer: Prof. Dr.-Ing. Christoph Neumann

2. Prüfer: Prof. Dr. rer. nat. Kurt Hoffmann

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Ostbayerische Technische Hochschule
Amberg-Weiden

Bestätigung gemäß § 12 APO

Name und Vorname
der Studentin/des Studenten: **Stangl, Thomas**

Studiengang: **Medieninformatik**

Ich bestätige, dass ich die Bachelorarbeit mit dem Titel:

Entwurf und Implementierung einer Webanwendung zur einfachen Fernsteuerung von Digitalmischpulten

selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Datum: 11. März 2022

Unterschrift:

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Ostbayerische Technische Hochschule
Amberg-Weiden

Bachelorarbeit Zusammenfassung

Studentin/Student (Name, Vorname): **Stangl, Thomas**
Studiengang: Medieninformatik
Aufgabensteller, Professor: Prof. Dr.-Ing. Christoph Neumann
Ausgabedatum: 12. Oktober 2021 Abgabedatum: 11. März 2022

Titel:

Entwurf und Implementierung einer Webanwendung zur einfachen Fernsteuerung von Digitalmischpulten

---

Zusammenfassung:

In dieser Arbeit wird eine Webanwendung namens „SimpleVolumeControl“ entworfen und implementiert, mit der Digitalmischpulte auf einfache Art und Weise auch von nicht fachkundigen Personen ferngesteuert werden können. Ein mögliches Einsatzszenario ist die Hintergrundbeschallung von Gebäuden, wobei für jeden Raum sowohl die Gesamtlautstärke als auch die jeweiligen Einzelsignale geregelt werden können. Das zentrale Hardware-Element von SimpleVolumeControl ist ein Raspberry Pi, das für jede Installation jeweils in ein bestehendes lokales Netz integriert wird, um dort die Webanwendung bereitzustellen und die Kommunikation mit dem Mischpult zu übernehmen. Das Frontend basiert auf dem Framework Preact, einer leichtgewichtigen Alternative zu React, in Kombination mit Next.js. Im Backend wird Node.js in Kombination mit Express.js verwendet. Als Programmiersprache kommt jeweils TypeScript zum Einsatz. Die Programmierschnittstelle zur Kommunikation zwischen Frontend und Backend basiert direkt auf dem WebSocket-Protokoll, um einen bidirektionalen Nachrichtenaustausch mit effizienter Bandbreitennutzung zu ermöglichen. Die Benutzeroberfläche ist auf die notwendigen Bestandteile reduziert und orientiert sich bei der Gestaltung der einzelnen Elemente an allgemein bekannten Beispielen, um eine intuitive Nutzung zu gewährleisten.

Schlüsselwörter: WebSocket, Preact, Open Sound Control, Node.js, Raspberry Pi

Title:

Design and Implementation of a Web Application for Simple Remote Control of Digital Mixing Consoles

Abstract:

For this thesis, a web application called “SimpleVolumeControl” is designed and implemented. It allows to remotely control digital mixing consoles in a simple manner that is also accessible to untrained persons. A possible use case are sound systems for background music in buildings. For each room, the main volume and the individual sound signals can be controlled. The central hardware element of each installation of SimpleVolumeControl is a Raspberry Pi. It is integrated into existing local networks in order to provide the web application and handle the communication with the mixing console. The frontend is based on the Preact framework, a lightweight alternative to React, in combination with Next.js. The backend uses Node.js in combination with Express.js. For both layers, TypeScript is used as a programming language. The interface for the communication between frontend and backend is directly based on the WebSocket protocol in order to provide bidirectional messaging capabilities with efficient bandwidth usage. The user interface is reduced to the essential parts and uses commonly known user interface elements as a basis for the design. This allows for an intuitive usage.

Keywords: WebSocket, Preact, Open Sound Control, Node.js, Raspberry Pi

Inhaltsverzeichnis

Kapitel 1

Einführung

In diesem Kapitel wird einführend das zu bearbeitende Problem dargestellt, außerdem werden die Forschungsfragen definiert. Des Weiteren werden die Ausgangssituation und das Vorgehen beschrieben.

1.1 Motivation

Mischpulte werden in der Tontechnik dazu verwendet, verschiedene Audio-Signale zu bearbeiten, zu mischen und diese Mischungen an den Ausgängen wieder auszugeben [1, S. 339–341]. In den vergangenen Jahren haben Digitalmischpulte stark an Bedeutung gewonnen und verdrängen analoge Mischpulte zunehmend. Durch die Verfügbarkeit relativ kostengünstiger Digitalmischpulte, beispielsweise dem Behringer X32, erstreckt sich diese Entwicklung auch auf niedrigpreisige Marktsegmente [2].

Oftmals bieten Digitalmischpulte einen deutlich größeren Funktionsumfang als vergleichbare Analogmischpulte, sodass auch die Bedienung komplexer ist. Viele Digitalmischpulte lassen sich über ein IP-Netz fernsteuern.

Es soll nun eine Anwendung erstellt werden, mit der es auch nicht fachkundigen Personen möglich ist, zumindest die grundlegenden Funktionen von Digitalmischpulten über eine Fernsteuerung zu nutzen. Ein beispielhaftes Nutzungsszenario ist die Hintergrundbeschallung in einem Hotel. Dort sollen auch Personen ohne Vorkenntnisse in der Tontechnik die Lautstärke der verschiedenen Ein- und Ausgänge des Mischpults regeln können.

Im Rahmen dieser Arbeit soll dabei vor allem folgenden Fragen nachgegangen werden:

- Welche Auswirkungen haben die Besonderheiten der Mischpult-Schnittstelle auf den Entwurf und die Implementierung der Webanwendung?
- Wie kann die Anwendung dahingehend optimiert werden, dass eine Verwendung auch bei schlechter Netzwerkverbindung möglich ist?

- Was muss berücksichtigt werden, damit die Webanwendung auf leistungsschwachen Servern, beispielsweise Raspberry-Pi-Geräten, in lokalen Netzen betrieben werden kann?
- Wie kann eine Benutzerschnittstelle gestaltet werden, welche auch für nicht fachkundige Personen leicht verständlich und intuitiv bedienbar ist?

1.2 Ausgangssituation

In den Jahren 2018/2019 habe ich bereits eine erste Version einer solchen Anwendung unter dem Namen „SimpleVolumeControl“ entwickelt. In Abbildung 1.1 werden die wesentlichen Bestandteile der Benutzeroberfläche gezeigt. Seit Juni 2019 befindet sich SimpleVolumeControl in einem Hotel im Produktiveinsatz zur Steuerung der Lautstärke der Hintergrundbeschallung. Dabei sind bisher keine Ausfälle oder ähnliche Probleme aufgetreten.

Dennoch ist eine Neuentwicklung nötig, da die erste Version nicht nach den Maßstäben ingenieurmäßiger Softwareentwicklung erstellt wurde und dementsprechend schlecht wartbar und erweiterbar ist. Außerdem sind die damals verwendeten Technologien und Bibliotheken teilweise veraltet.

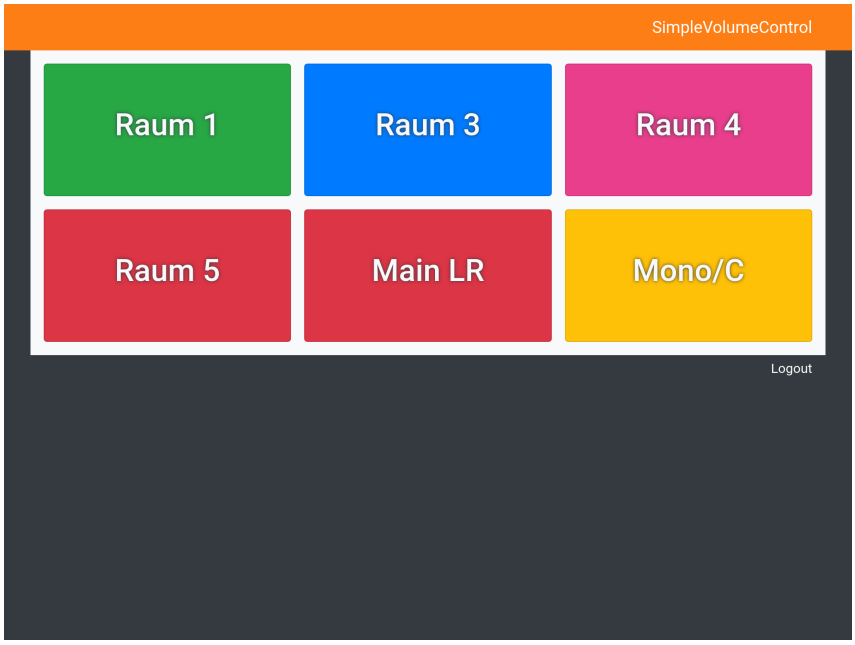
1.3 Vorgehen

Zunächst werden die Anforderungen analysiert. Nach der Identifikation und Beschreibung von Anwendungsfällen und Systemoperationen werden verwandte Anwendungen und wissenschaftliche Arbeiten betrachtet.

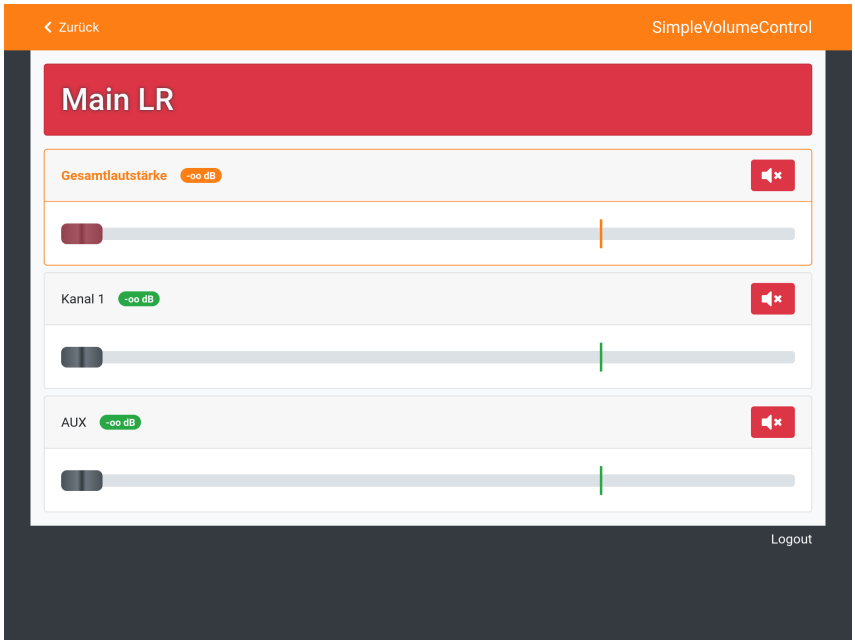
Daraufhin werden die Funktionalitäten des Mischpults und wichtige Begrifflichkeiten sowie die Benutzerinteraktionen erklärt. Danach folgt die Auswahl geeigneter Technologien und Bibliotheken für die Implementierung von Frontend, Backend und der Netzwerkschnittstelle. Im Anschluss daran wird die Architektur und der Entwurf der Anwendung auf verschiedenen Ebenen beschrieben.

Danach beginnt die Implementierung der Anwendung, wobei zwingend benötigte Funktionalitäten Vorrang vor optionalen Funktionalitäten haben. Interessante Aspekte mit Bezug zur Implementierung werden in einem eigenen Kapitel dargestellt.

Im Anschluss werden einige Ideen zum Deployment der Anwendung entwickelt, wobei aber ein vollständiges Deployment-Konzept nicht zum Aufgabenbereich der vorliegenden Arbeit gehört. Schließlich werden das Vorgehen und die Ergebnisse evaluiert und zusammengefasst.



(a) Übersicht über alle Räume



(b) Ansicht eines einzelnen Raums mit Lautstärkereglern

Abbildung 1.1: Bildschirmfotos aus der ersten Version von SimpleVolumeControl

Kapitel 2

Anforderungsanalyse

Im Rahmen dieser Anforderungsanalyse liegt der Fokus zunächst auf Festinstallationen, beispielsweise zur Steuerung der Hintergrundbeschallung in Gebäuden. Eine Erweiterung auf andere Anwendungsbereiche, zum Beispiel der Einsatz bei Veranstaltungen, soll zukünftig möglich sein. Die Umsetzung einer solchen Erweiterung wird vorerst nicht weiter verfolgt.

Im Folgenden werden die Anforderungen aus Sicht des Endanwenders und des technischen Betreuers beschrieben. Die Nummerierung der Anforderungen bezieht sich auf Tabelle 2.1.

2.1 Anforderungen aus Endanwendersicht

Als Endanwender sind in diesem Kontext diejenigen Personen zu sehen, welche das fest installierte System im täglichen Betrieb nutzen. Dies können beispielsweise der Betreiber eines Hotels und von ihm beauftragte Angestellte sein.

Die beiden Hauptanforderungen sind die Möglichkeit zum Regeln der Gesamtlautstärke einzelner Räume (Anforderung Nr. 1) sowie das Regeln der Lautstärke der einzelnen Eingangs-Audiosignale in den jeweiligen Räumen (Anforderung Nr. 2). Dabei muss jeweils auch eine Möglichkeit zur Stummschaltung inbegriffen sein.

Aus technischer und organisatorischer Sicht ist es erforderlich, dass keinerlei Installation auf den Client-Geräten notwendig ist (Anforderung Nr. 13). Die Anwendung muss über ein bestehendes lokales Netz bereitgestellt werden (Anforderung Nr. 14). Ziel ist also, dass die Steuerungsoberfläche mit einem Webbrowser im lokalen Netz aufgerufen werden kann.

Ferner ist ein Schutz gegen versehentliche Verwendung gefordert (Anforderung Nr. 3). Dies kann beispielsweise in Form einer einfachen Kennwortabfrage erfolgen. Dadurch soll erreicht werden, dass nur ausgewählte Mitarbeiter Änderungen vornehmen können. Damit ist ausdrücklich kein Schutz gegen böswillige Angreifer gemeint. Die

Webanwendung wird nur in zugangsbeschränkten lokalen Netzen bereitgestellt, von denen angenommen wird, dass nur vertrauenswürdige Akteure Zugang haben.

Die Anwendung muss für Endanwender intuitiv verständlich sein, das heißt, die Einarbeitungszeit darf maximal 15 Minuten betragen (Anforderung Nr. 11). Ferner muss das System für den Dauerbetrieb ausgelegt sein, wobei ein automatischer Neustart pro Tag durchgeführt werden darf (Anforderung Nr. 15). Um die Hardwarekosten zu minimieren, muss die Anwendung für den Betrieb auf leistungsschwachen lokalen Servern, beispielsweise einem Raspberry Pi 3 Model B, ausgelegt sein (Anforderung Nr. 16). Es ist davon auszugehen, dass die Anwendung auch in Bereichen verwendet werden soll, wo die WLAN-Verbindung eher schlecht ist. Deshalb muss eine Optimierung dahingehend stattfinden, dass trotz schlechter Netzwerkverbindung eine Nutzung der Anwendung möglich ist (Anforderung Nr. 17).

Der Lautstärkepegel der Eingangs- und Ausgangssignale soll visualisiert werden (Anforderung Nr. 4). Dies dient der Vergewisserung, dass auch tatsächlich Eingangssignale anliegen bzw. Audiosignale ausgegeben werden, was insbesondere dann hilfreich ist, wenn sich der Anwender außerhalb der Hörreichweite der jeweiligen Lautsprecher befindet.

Die Benutzeroberfläche soll übersetzbar gestaltet sein, neben der Primärsprache Deutsch kommt dabei vor allem auch Englisch als Alternative in Frage (Anforderung Nr. 12).

Optional können weitere Steuerungsparameter hinzugefügt werden, beispielsweise um Einstellungen am Equalizer vorzunehmen (Anforderung Nr. 5). Voraussetzung dafür ist, dass die Bedienung der Grundfunktionen trotzdem noch intuitiv und ohne besondere Fachkenntnisse möglich ist.

Vor dem Hintergrund, dass SimpleVolumeControl zunächst nur für die Steuerung der Hintergrundbeschallung in Gebäuden verwendet wird, ist eine Lautstärkeregelung der Eingangssignale auf globaler Ebene nicht gefordert.

2.2 Anforderungen aus Sicht des technischen Betreuers

Der technische Betreuer ist diejenige Person, welche die Anwendung installiert und wartet. Der technische Betreuer besitzt Fachkenntnisse im Bereich der Veranstaltungstechnik sowie Grundkenntnisse im Bereich der Netzwerktechnik.

Zwingend benötigt wird eine Unterstützung für alle Mischpulte aus der Behringer-X32/Midas-M32-Familie (Anforderung Nr. 6). Diese beinhaltet folgende Modelle:

- Behringer X32
- Behringer X32 Compact
- Behringer X32 Producer
- Behringer X32 Rack
- Midas M32
- Midas M32 Live
- Midas M32R
- Midas M32R Live
- Midas M32C

Eine direkte Verwendung des Mischpults muss weiterhin möglich bleiben (Anforderung Nr. 7). Dies ist beispielsweise dann relevant, wenn bei besonderen Anlässen das Mischpult von Fachpersonal bedient werden soll.

Es soll möglich sein, die Zuweisung der Eingangskanäle an die verschiedenen Räume und das Setzen weiterer Konfigurationsparameter über eine graphische Benutzeroberfläche innerhalb der Anwendung vorzunehmen (Anforderung Nr. 8). Dabei ist zwingend ein Schutz gegen versehentliche Konfiguration einzubauen, damit nicht fachkundiges Personal keine unerwünschten Einstellungen vornimmt. Auch hier ist wieder ausdrücklich keine Schutzmaßnahme gegen böswillige Angreifer gemeint; ein Bestätigungsdialog oder eine Passwortabfrage werden als ausreichend erachtet. Optional kann diese Konfigurationsansicht dahingehend erweitert werden, dass die Netzwerkkonfiguration direkt aus der Anwendung heraus möglich ist (Anforderung Nr. 9). Dies bietet den Vorteil, dass nach der Konfektionierung des Steuerungssystems die Anwendung nicht mehr aktiv verlassen werden muss.

Optional ist auch eine Unterstützung für andere Mischpulte möglich (Anforderung Nr. 10).

2.3 Zusammenfassung der Anforderungen

In der Tabelle 2.1 sind die oben beschriebenen Anforderungen anhand einiger von Balzert [3, S. 479f] beschriebenen Attribute zusammengefasst.

Nr.	Kurzbezeichnung	Wichtigkeit[1]	Stakeholder	Aufwand[2]
Funktionale Anforderungen				
1	Regelung der Raumlautstärken	essenziell	Endanwender	mittel
2	Regelung der Einzelsignale pro Raum	essenziell	Endanwender	mittel
3	Zugangsschutz (z.B. Kennwort)	essenziell	Endanwender	niedrig
4	Pegelanzeige	bedingt	Endanwender	niedrig
5	Weitere Audio-Einstellungen	optional	Endanwender	hoch
6	Unterstützung für X32/M32-Mischpulte	essenziell	Technischer Betreuer	mittel
7	Direkte Bedienung des Mischpults	essenziell	Technischer Betreuer	gering
8	GUI zur Konfiguration	bedingt	Technischer Betreuer	mittel
9	GUI für Netzwerkkonfiguration	optional	Technischer Betreuer	hoch
10	Unterstützung für weitere Mischpulte	optional	Technischer Betreuer	niedrig bis hoch[3]
Benutzbarkeit				
11	Einarbeitungszeit höchstens 15 min	essenziell	Endanwender	—
12	Übersetzbarkeit	bedingt	Endanwender	—
Portabilität				
13	Keine Installation auf Client-Geräten	essenziell	Endanwender	—
14	Bereitstellung über bestehendes lokales Netz	essenziell	Endanwender	—
Zuverlässigkeit				
15	Dauerbetrieb mit einem Neustart pro Tag	essenziell	Endanwender	—
Effizienz				
16	Lauffähig auf Raspberry Pi	essenziell	Endanwender	—
17	Verwendbar bei schlechter Netzwerkverbindung	essenziell	Endanwender	—

Tabelle 2.1: Zusammenfassende Darstellung der Anforderungen.

[1] vgl. Wichtigkeit bei Balzert [3, S. 475]
[2] geschätzt; „niedrig“ bedeutet hier weniger als 3 Personentage, „mittel“ weniger als 10 Personentage, „hoch“ mindestens 10 Personentage
[3] abhängig vom konkreten Mischpultmodell

2.4 Anwendungsfälle und Systemoperationen

Basierend auf diesen Anforderungen können folgende Anwendungsfälle identifiziert werden:

Anwendungsfall 1: Lautstärken anpassen

Anwendungsfall 2: Lautstärkepegel überwachen

Anwendungsfall 3: Mischpult direkt nutzen

Anwendungsfall 4: Anwendung konfigurieren (Raumkonfiguration und weitere Parameter)

Anwendungsfall 5: System (inkl. Netzwerk) einrichten

Anwendungsfall 6: Weitere Audio-Parameter steuern

Eine detaillierte Darstellung der Anwendungsfälle findet sich in Anhang A. Dabei wird bei jedem Anwendungsfall als Beschreibung eine User Story formuliert.

Aus diesen Anwendungsfällen lassen sich die Systemoperationen ableiten. Systemoperationen behandeln Nachrichten, die von außen an das System herangetragen werden [4, S. 176, S. 183f]. Intern wird durch die Systemoperation meist die Ausführung weiterer Operationen veranlasst. Im vorliegenden Fall lassen sich folgende Systemoperationen erkennen, wobei nur die bereits vollständig ausformulierten Anwendungsfälle (siehe Anhang A) betrachtet wurden:

Systemoperation 1: Kennwort eingeben

Systemoperation 2: Übersicht über alle Räume anfordern

Systemoperation 3: Raum auswählen

Systemoperation 4: Lautstärke ändern

Systemoperation 5: Stummschaltung ändern

Systemoperation 6: Sitzung beenden

Kapitel 3

Verwandte Arbeiten

Es existieren bereits verwandte Anwendungen, welche ähnliche Problemstellungen zu lösen versuchen. Allerdings erfüllt keine der im Folgenden vorgestellten Anwendungen alle Anforderungen, sodass die Neuentwicklung von SimpleVolumeControl notwendig ist. Anschließend werden einige wissenschaftliche Arbeiten vorgestellt, deren Inhalt für die Entwicklung von SimpleVolumeControl relevant erscheint.

3.1 Verwandte Anwendungen

Es gibt einige Anwendungen und Systeme, welche die oben beschriebenen Anforderungen zumindest teilweise erfüllen. Im Folgenden werden einige Beispiele näher beleuchtet, wobei kein Anspruch auf Vollständigkeit erhoben wird. Die Auswahl erfolgte mit dem Ziel, eine möglichst große Vielfalt unterschiedlicher Konzepte darzustellen.

3.1.1 Anwendung „Mixing Station“

Die Anwendung „Mixing Station“[1] zeichnet sich durch die Unterstützung verschiedener Mischpulttypen aus. Sie ermöglicht die Fernsteuerung sämtlicher Funktionen des Mischpults und ergänzt teilweise auch zusätzliche Funktionalitäten. „Mixing Station“ richtet sich also an eher professionelle Anwender und ist für PC und mobile Geräte verfügbar.

3.1.2 Anwendung „X32-Edit“

„X32-Edit“[2] ist die offizielle Fernsteuerungsanwendung von Behringer für die Mischpulte der X32-Familie. Dementsprechend ist diese Anwendung auf Mischpulte der Behringer-X32/Midas-M32-Familie begrenzt. Auch hier ist eine Steuerung sämtlicher Mischpult-Funktionen möglich. Die Zielgruppe umfasst ebenfalls eher professionelle

[1] `https://dev-core.org/mixing-station/`
[2] `https://www.behringer.com/product.html?modelCode=P0ASF`

Anwender, welche Kenntnisse in der Bedienung des Mischpults haben. „X32-Edit“ ist als PC-Anwendung für Linux, Windows und macOS verfügbar.

3.1.3 Arbeiten von Patrick-Gilles Maillot

Patrick-Gilles Maillot bietet auf seiner Internetseite[3] diverse Hilfswerkzeuge für Mischpulte aus der Behringer-X32-Familie an. Dabei geht es um die Ergänzung zusätzlicher Funktionen, vor allem im Hinblick auf die Kommunikation zwischen PC und Mischpult. Diese Anwendungen sind dem Anschein nach an Entwickler und Computer-affine Tontechniker gerichtet.

Besonders relevant ist die Dokumentation der OSC-Schnittstelle für Mischpulte der Behringer-X32-Familie, welche Patrick-Gilles Maillot zum Herunterladen anbietet.

3.1.4 Verschiedene OSC-Anwendungen

Es gibt diverse Anwendungen, beispielsweise „OSC Controller“[4] oder „TouchOSC“[5], welche eine flexibel konfigurierbare Oberfläche für OSC bieten. Sie sind sehr allgemein gehalten und nicht speziell für die Fernsteuerung von Mischpulten vorgesehen. Derartige Anwendungen sind vor allem für Android und iOS-Geräte verfügbar, aber auch für PC.

3.1.5 Spezialsysteme, beispielsweise Bose ControlSpace

Für Hintergrundbeschallungen gibt es auch Spezialsysteme mit zugehörigen Steuerungsanwendungen, beispielsweise Bose ControlSpace[6]. Um diese Anwendungen nutzen zu können, muss das System auf der jeweiligen Spezialhardware basieren, was meist auch eine relativ aufwändige Installation einschließt.

3.2 Verwandte wissenschaftliche Arbeiten

Lokale Raspberry-Pi-Geräte als Webserver für die Steuerung anderer Hardware werden bereits seit einiger Zeit eingesetzt, insbesondere im Smart-Home-Umfeld. So beschreiben beispielsweise Kehagias und Nini 2015 in ihrem Aufsatz „Home Automation Based on an Android and a Web Application Using Raspberry Pi“ ein System, bei dem eine Webanwendung sowie eine Android-Anwendung zum Ein- und Ausschalten von Geräten im Smart-Home-Umfeld verwendet wird. Dabei wird das Raspberry Pi als Webserver und für die eigentliche Steuerung der Geräte genutzt. Allerdings werden dabei die Geräte direkt über Relais, welche über die GPIO-Pins angesteuert werden, ein- und ausgeschaltet, sodass keine Netzwerkprotokolle für die Kommunikation zwischen Raspberry Pi und den Smart-Home-Geräten zum Einsatz kommen [5].

[3] `https://sites.google.com/site/patrickmaillot/x32`

[4] `https://play.google.com/store/apps/details?id=com.ffsmultimedia.osccontroller`

[5] `https://hexler.net/touchosc`

[6] `https://pro.bose.com/de_de/products/spaces/hospitality.html`

Verschiedene Arbeiten untersuchen Möglichkeiten, Benutzeroberflächen zum Mischen von Audiosignalen intuitiver und leichter verständlich zu gestalten. Carrascal und Jordà schlagen 2011 im Artikel „Multitouch Interface for Audio Mixing“ vor, die verschiedenen Eingänge auf einer Fläche zu arrangieren. Damit kann die Lautstärke und insbesondere auch die räumliche Anordnung in einem Stereo- oder Surround-System gesteuert werden. Um das Konzept verschiedener Tonmischungen aus den Eingangssignalen, beispielsweise für die Beschallung des Publikums, die Bühnenmonitore[7] und eine Aufnahme, besser verständlich zu machen, wird anstelle des üblicherweise verwendeten Begriffs *Mix Bus* die Metapher „Bühne“ (im englischen Original „*stage*“) eingeführt [6]. Insbesondere dieser letzte Aspekt ist auch für SimpleVolumeControl relevant, da hier als Metapher für verschiedene Tonmischungen der Begriff „Raum“ genutzt wird.

Dewey und Wakefield vergleichen verschiedene Arten von Benutzeroberflächen für Audio-Mischer. Auch dort sind insbesondere bei Oberflächen, die auf der Bühnen-Metapher aufsetzen, und bei Icon-basierten Oberflächen Vorteile erkennbar [7]. Als weniger vorteilhaft erweist sich eine von Cartwright, Pardo und Reiss untersuchte Benutzeroberfläche, bei der die einzelnen Kanäle auf einer Fläche angeordnet werden können, wobei eine Achse der Laustärke und die zweite Achse den Equalizer-Einstellungen entspricht [8].

In mehreren Veröffentlichungen wird das WebSocket-Protokoll diskutiert, vor allem auch in Bezug auf effiziente Bandbreitennutzung. Srinivasan, Scharnagl und Schilling untersuchen WebSockets im Umfeld der Fernsteuerung von Robotern, auch bei geringen Bandbreiten. Sie vergleichen WebSockets und traditionelle Implementierungen auf Basis von HTTP. In ihren Messungen ist erkennbar, dass WebSockets in diesem Anwendungsfall einfachem HTTP überlegen sind; insbesondere ist ein deutlich geringerer Overhead bei WebSockets im Vergleich zum Anfrage-Antwort-Mechanismus bei HTTP erkennbar, was die benötigte Bandbreite deutlich reduziert. Sie kommen zu dem Schluss, dass WebSockets für diesen Einsatzzweck geeignet sind [9]. Diese Erkenntnisse sind auf die Webanwendung zur Fernsteuerung von Mischpulten übertragbar, da auch hier eine bidirektionale Verbindung zum Austausch der Steuerungsinformationen benötigt wird, welche selbst bei geringer Bandbreite und einem hohen Aufkommen an Steuerungsbefehlen zuverlässig funktionieren soll.

Mun, Dinh und Kwon bestätigen ebenfalls, dass WebSocket ein geeignetes Protokoll für Internet-Of-Things-Anwendungen ist. Bei ihren Messungen der Übertragungsdauer sowie des Energieverbrauchs erzielen WebSockets ähnliche Ergebnisse wie reine TCP-Verbindungen [10].

[7] Als Bühnenmonitor bezeichnet man einen Lautsprecher, der nicht ins Publikum, sondern auf einen Bereich der Bühne gerichtet ist. Dabei wird eine für den jeweiligen Musiker oder Redner optimierte Tonmischung verwendet, die sich deutlich von der Hauptmischung unterscheiden kann.

Kapitel 4

Fachkonzept

Ziel von SimpleVolumeControl ist es, die Bedienung von Mischpulten auch für nicht fachkundige Personen zu ermöglichen. Um jedoch ein tiefergehendes Verständnis der Funktionsweise von SimpleVolumeControl zu erlangen, werden in diesem Kapitel zunächst die relevanten Begrifflichkeiten und Funktionsweisen von Mischpulten erklärt. Basierend darauf wird erläutert, wie diese Konzepte in SimpleVolumeControl dargestellt werden. Nach einer exemplarischen Beschreibung der Benutzerinteraktionen folgt im letzten Abschnitt die Spezifikation der Benutzeroberfläche.

4.1 Funktionalität des Mischpults und Begrifflichkeiten

SimpleVolumeControl ist eine Anwendung zur Fernsteuerung von Mischpulten. Für ein besseres Verständnis ist es deshalb notwendig, zunächst die Funktionalitäten des Mischpults, auf denen SimpleVolumeControl aufbaut, zu betrachten und dabei auch wichtige Begriffe zu klären.

Die folgenden Ausführungen beziehen sich auf Mischpulte der Behringer-X32-Familie, jedoch finden sich bei den meisten Digitalmischpulten ähnliche Mechanismen, teilweise mit etwas anderen Bezeichnungen. Die Informationen sind stark vereinfacht und auf die für SimpleVolumeControl relevanten Aspekte reduziert.

Jedes Mischpult hat eine festgelegte Anzahl an Eingängen (englisch *inputs*). Diese Eingänge können auch als Kanäle (englisch *channels*) bezeichnet werden[1]. An einen Eingang kann beispielsweise ein Mikrofon, ein CD-Spieler oder Ähnliches angeschlossen werden. Diese Audiosignale können dann weiterverarbeitet werden, beispielsweise durch Verwendung eines Equalizers oder anderer Effekte.

Als letzter Schritt in der Verarbeitung jedes Eingangssignals kann geregelt werden, mit welcher Lautstärke das jeweilige Signal auf den Hauptausgang gemischt wird und ob der Eingang stummgeschaltet sein soll. Stummgeschaltete Eingänge werden

[1] Je nach Mischpult können die Begriffe „Eingang“, „Kanal“ und „Hilfskanal“ (englisch *auxiliary channel*) etwas anderes bedeuten. Für SimpleVolumeControl sind diese Unterschiede jedoch irrelevant, weshalb nicht näher darauf eingegangen wird.

nicht auf den Hauptausgang gemischt. Für diesen Hauptausgang gibt es verschiedene Bezeichnungen, beispielsweise „Main LR“.

Dieser Hauptausgang ist jedoch nicht die einzige Tonmischung, vielmehr gibt es bei den meisten Digitalmischpulten mehrere sogenannte „Mix Buses“ oder kurz „Mixes“ bzw. „Buses“. Dabei handelt es sich um voneinander unabhängige Mischungen, das heißt es kann für jeden Kanal festgelegt werden, ob und wie viel er zu jedem einzelnen Mix Bus beiträgt. In anderen Worten: Jedem Paar aus Eingang und Mischung kann sowohl ein Lautstärkewert als auch ein Stummschaltungsstatus zugeordnet werden. Diese Informationen können anschaulich als Matrix dargestellt werden, wie in Tabelle 4.1 erkennbar ist. Die Darstellung ist auf insgesamt sieben Ausgänge und acht Eingänge limitiert; in der Realität sind eher mindestens 16 Ausgänge und mindestens 32 Eingänge zu erwarten.

	Bus 1	Bus 2	Bus 3	Bus 4	Bus 5	Bus 6	Main
Eingang 1	OFF 68 %	ON 95 %	OFF 47 %	OFF 27 %	ON 82 %	ON 52 %	ON 75 %
Eingang 2	ON 7 %	ON 53 %	OFF 1 %	ON 58 %	ON 51 %	ON 58 %	OFF 83 %
Eingang 3	ON 60 %	OFF 100 %	ON 68 %	ON 77 %	ON 77 %	ON 47 %	ON 67 %
Eingang 4	ON 30 %	ON 2 %	OFF 96 %	OFF 38 %	OFF 3 %	OFF 28 %	OFF 100 %
Eingang 5	ON 69 %	ON 53 %	OFF 93 %	ON 38 %	ON 45 %	ON 38 %	OFF 88 %
Eingang 6	OFF 37 %	OFF 36 %	ON 67 %	ON 5 %	ON 6 %	OFF 29 %	ON 28 %
Eingang 7	ON 52 %	OFF 33 %	OFF 4 %	OFF 86 %	OFF 65 %	OFF 57 %	ON 29 %
Eingang 8	OFF 25 %	ON 83 %	OFF 33 %	ON 89 %	OFF 82 %	OFF 42 %	ON 88 %

Tabelle 4.1: Beispielhafte Matrix-Darstellung der Mix-Bus-Konfiguration am Mischpult. „ON“ und „OFF“ geben an, ob der Kanal stummgeschaltet ist. Der Prozentwert gibt die Lautstärke an, mit der der entsprechende Eingang zum jeweiligen Mix beiträgt.

Bei Veranstaltungen wird üblicherweise die Hauptmischung für die Beschallung des Publikums verwendet. Die übrigen Mischungen werden beispielsweise für Bühnenmonitore[2], für Tonaufnahmen oder für eine Übertragung verwendet. Bei der Hintergrundbeschallung mehrerer Räume hingegen ist es denkbar, dass die Hauptmischung nicht genutzt wird, stattdessen wird für jeden Raum ein eigener Mix Bus genutzt, sodass eine individuelle Beschallung für jeden einzelnen Raum möglich ist.

[2]Als Bühnenmonitor bezeichnet man einen Lautsprecher, der nicht ins Publikum, sondern auf einen Bereich der Bühne gerichtet ist. Dabei wird eine für den jeweiligen Musiker oder Redner optimierte Tonmischung verwendet, die sich deutlich von der Hauptmischung unterscheiden kann.

Jedem Eingang und jedem Mix kann auf dem Mischpult eine Beschriftung und eine Farbe zugeordnet werden. Bei den meisten Mischpulten der Behringer-X32-Familie ist über jedem Schieberegler ein kleiner Bildschirm eingebaut, welcher die jeweiligen Informationen anzeigt.

Da diese Fachbegriffe für Endanwender nicht unbedingt intuitiv verständlich sind, werden – ähnlich wie bei Carrascal und Jordà [6] – Metaphern eingeführt. Dadurch, dass in der vorliegenden Arbeit der Fokus auf der Hintergrundbeschallung liegt, wird als Metapher für eine Tonmischung – also einen Mix Bus – der Begriff „Raum“ verwendet. Dies liegt nahe, da für jeden Raum eine eigene Mischung zur Beschallung verwendet wird. Es liegt also eine 1-zu-1-Beziehung zwischen Raum und Tonmischung vor.

In den meisten Fällen entspricht die Anzahl der auf dem Mischpult verfügbaren Mix Buses nicht exakt der tatsächlich vorhandenen Anzahl an beschallten Räumen, das heißt es bleiben üblicherweise mehrere Mix Buses ungenutzt. Ebenso übersteigt die Anzahl an verfügbaren Eingängen meist die Anzahl der tatsächlich genutzten Eingangssignale. Weiterhin sind nicht alle anliegenden Eingangssignale für jeden Raum relevant, beispielsweise wird in einem Party-Raum niemals die Entspannungsmusik gebraucht, die für die Beschallung des Spa-Bereichs gedacht ist.

Um die Bedienung für den Endanwender möglichst einfach zu gestalten, ist es deshalb nötig, in SimpleVolumeControl eine Raumkonfiguration anzulegen. Dort wird hinterlegt, welche Mischungen überhaupt für die Raumbeschallung genutzt werden und welche Eingänge für die jeweiligen Räume relevant sind. Eine beispielhafte Raumkonfiguration könnte also wie in Abbildung 4.1 dargestellt aussehen. Eine zu den Mischungen und Eingängen gehörende Beschreibung wird *nicht* in der Raumkonfiguration abgelegt. Stattdessen werden die jeweilige Beschriftung und Farbe, welche im Mischpult hinterlegt sind, verwendet.

```
Mix 1: Eingang 1, Eingang 2, Eingang 5, Eingang 6
Mix 2: Eingang 1, Eingang 2, Eingang 3, Eingang 4
Mix 3: Eingang 1, Eingang 2
Mix 4: Eingang 5, Eingang 6, Eingang 7
```

Abbildung 4.1: Eine beispielhafte Raumkonfiguration für SimpleVolumeControl.

Wird diese Raumkonfiguration aus Abbildung 4.1 auf die in Tabelle 4.1 gezeigte Mix-Bus-Matrix angewendet, so ergibt sich die in Tabelle 4.2 dargestellte Matrix. Dabei können ausschließlich die nicht schraffierten Kombinationen über SimpleVolume-Control gesteuert werden, während die schraffierten Felder nur über das Mischpult angesteuert werden können. Der technische Betreuer muss sicherstellen, dass die grau schraffierten Kombinationen auf dem Mischpult stummgeschaltet werden, wenn sie nicht in der Tonmischung hörbar sein sollen.

Mit dem Begriff Raumübersicht ist bei SimpleVolumeControl eine Liste aller Räume (Tonmischungen) gemeint, die in der Raumkonfiguration hinterlegt sind. Im Beispiel aus Abbildung 4.1 wären dies Mix 1, Mix 2, Mix 3 und Mix 4. Mit „Detailansicht eines

	Bus 1	Bus 2	Bus 3	Bus 4	Bus 5	Bus 6	Main
Eingang 1	OFF 68 %	ON 95 %	OFF 47 %	OFF 27 %	ON 82 %	ON 52 %	ON 75 %
Eingang 2	ON 7 %	ON 53 %	OFF 1 %	ON 58 %	ON 51 %	ON 58 %	OFF 83 %
Eingang 3	ON 60 %	OFF 100 %	ON 68 %	ON 77 %	ON 77 %	ON 47 %	ON 67 %
Eingang 4	ON 30 %	ON 2 %	OFF 96 %	OFF 38 %	OFF 3 %	OFF 28 %	OFF 100 %
Eingang 5	ON 69 %	ON 53 %	OFF 93 %	ON 38 %	ON 45 %	ON 38 %	OFF 88 %
Eingang 6	OFF 37 %	OFF 36 %	ON 67 %	ON 5 %	ON 6 %	OFF 29 %	ON 28 %
Eingang 7	ON 52 %	OFF 33 %	OFF 4 %	OFF 86 %	OFF 65 %	OFF 57 %	ON 29 %
Eingang 8	OFF 25 %	ON 83 %	OFF 33 %	ON 89 %	OFF 82 %	OFF 42 %	ON 88 %

Tabelle 4.2: Beispielhafte Matrix-Darstellung der Mix-Bus-Konfiguration unter Berücksichtigung der Raumkonfiguration. Grau schraffierte Felder können nicht über SimpleVolumeControl gesteuert werden, sondern lediglich über das Mischpult direkt.

Raumes“ sind alle für den jeweiligen Raum konfigurierten Eingänge und zusätzlich die Hauptregler für den gesamten Raum gemeint.

Es ist zu beachten, dass die Raummetapher nur dort verwendet wird, wo es für den Endanwender relevant ist. In der zugrunde liegenden Technik von SimpleVolumeControl werden weiterhin die Begriffe aus der Mischpulttechnik verwendet. Dies bietet den Vorteil, dass SimpleVolumeControl zukünftig relativ einfach um andere Anwendungsbereiche, für die die Raummetapher nicht geeignet ist, erweitert werden kann.

4.2 Benutzerinteraktionen

Die wichtigsten Benutzerinteraktionen lassen sich am Anwendungsfall 1 „Lautstärken anpassen“ erkennen. Dieser Anwendungsfall behandelt die Kernfunktionalität des Systems. Die einzelnen Interaktionen sind im normalen Ablauf und in den Ablaufvarianten detailliert aufgelistet, weshalb nun an dieser Stelle die Beschreibung von Anwendungsfall 1 folgt. Die übrigen Anwendungsfälle finden sich in Anhang A.

Anwendungsfall 1: **Lautstärken anpassen**

Priorität: hoch

Beschreibung: Als Person, welche für die Hintergrundbeschallung zuständig ist, (z.B. Hotelbetreiber, Personal, ...) möchte ich die Laustärken gezielt anpassen können, um eine optimale Beschallung zu erreichen.

Vorbedingung: Das Steuerungssystem und das Mischpult sind eingeschaltet und vollständig eingerichtet. Der Benutzer hat die Startseite der Anwendung im Internetbrowser geöffnet.

Nachbedingung: Die gewünschte Anpassung wurde an das Mischpult übertragen.

Normaler Ablauf:

1. Das System fragt nach dem Kennwort zur Anmeldung.
2. Der Benutzer meldet sich durch Eingabe des Kennworts an.
3. Das System zeigt eine Übersicht zur Auswahl der Räume.
4. Der Benutzer wählt denjenigen Raum aus, für welchen er Anpassungen vornehmen möchte.
5. Das System zeigt die Regler für die Einzelsignale und die Gesamtlautstärke des Raums.
6. Der Benutzer nimmt die gewünschte Anpassung vor.
7. Das System sendet die Anpassung an das Mischpult und zeigt die neuen Werte an.
8. Der Benutzer meldet sich ab.
9. Das System beendet die Sitzung.
10. Ende.

Ablaufvarianten:

1a Der Benutzer ist bereits angemeldet.

1. Das System fragt nicht nach dem Kennwort zur Anmeldung.
2. Weiter bei 3 im normalen Ablauf.

6a Der Benutzer möchte die Gesamtlautstärke des Raumes ändern.

1. Der Benutzer gibt die gewünschte Lautstärke am Gesamtlautstärkeregler ein.
2. Weiter bei 7 im normalen Ablauf.

6b Der Benutzer möchte den ganzen Raum stummschalten.

1. Der Benutzer aktiviert die Stummschaltung beim Gesamtlautstärkeregler.
2. Weiter bei 7 im normalen Ablauf.

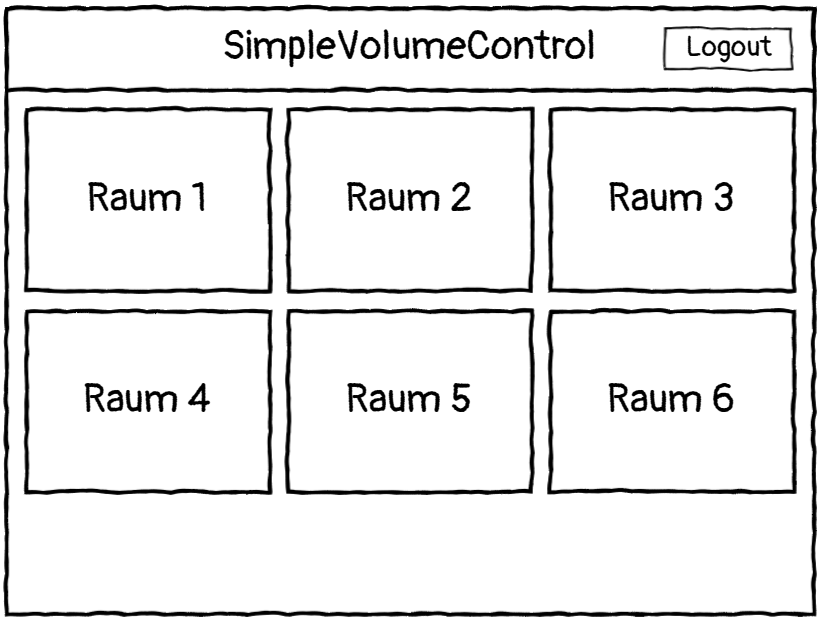
6c	Der Benutzer möchte die Stummschaltung des ganzen Raumes aufheben. 1. Der Benutzer deaktiviert die Stummschaltung beim Gesamtlautstärkeregler. 2. Weiter bei 7 im normalen Ablauf.
6d	Der Benutzer möchte die Lautstärke eines Einzelsignals für den Raum ändern. 1. Der Benutzer gibt die gewünschte Lautstärke am Lautstärkeregler für das jeweilige Einzelsignal ein. 2. Weiter bei 7 im normalen Ablauf.
6e	Der Benutzer möchte ein Einzelsignal für den Raum stummschalten. 1. Der Benutzer aktiviert die Stummschaltung beim Lautstärkeregler für das jeweilige Einzelsignal. 2. Weiter bei 7 im normalen Ablauf.
6f	Der Benutzer möchte die Stummschaltung eines Einzelsignals für den Raum aufheben. 1. Der Benutzer deaktiviert die Stummschaltung beim Lautstärkeregler für das jeweilige Einzelsignal. 2. Weiter bei 7 im normalen Ablauf.
8a	Der Benutzer möchte eine weitere Anpassung im selben Raum vornehmen. 1. Der Benutzer meldet sich nicht ab. 2. Weiter bei 5 im normalen Ablauf.
8b	Der Benutzer möchte eine weitere Anpassung in einem anderen Raum vornehmen. 1. Der Benutzer meldet sich nicht ab. 2. Der Benutzer wechselt zurück zur Raumübersicht. 3. Weiter bei 3 im normalen Ablauf.

4.3 Benutzeroberfläche

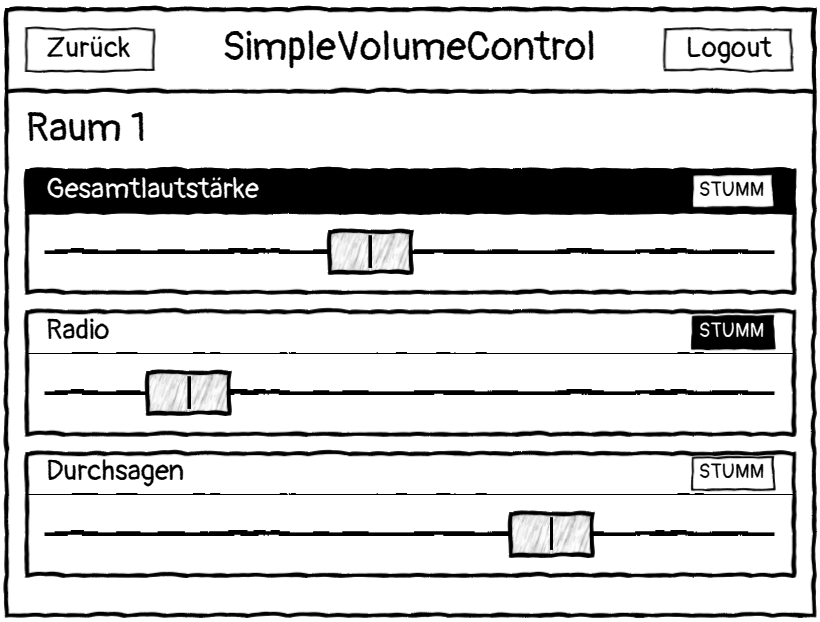
In Abbildung 4.2 ist eine Benutzeroberflächenspezifikation in Form von Wireframes dargestellt. Nachdem der Benutzer das Kennwort eingegeben hat, erscheint zunächst eine Ansicht zur Auswahl des gewünschten Raumes. Durch einen Klick auf eine entsprechende Kachel wird die Detailansicht des jeweiligen Raumes aufgerufen, in welcher der Benutzer sowohl die Gesamtlautstärke als auch die einzelnen Eingänge steuern kann. Eine Rückkehr zur Übersicht ist mittels einer Schaltfläche möglich.

Es ist zu beachten, dass diese Benutzeroberfläche nicht nur vom Endanwender, sondern auch vom technischen Betreuer verwendet wird. Wie bereits in Abschnitt 2.2 angedeutet, handelt es sich beim technischen Betreuer in der Regel um einen Veranstaltungstechniker und nicht um einen IT-Dienstleister. Ein denkbares Szenario wäre deshalb, dass der technische Betreuer bei Änderungen an der Beschallungsstruktur SimpleVolumeControl verwendet, um die neuen Lautstärkewerte zu setzen. Die für die Zukunft angedachte graphische Konfigurationsansicht soll also auch Bestandteil derselben Benutzeroberfläche sein, nur hinter einem Warnhinweis verborgen. Dies hilft dem technischen Betreuer, weil er dann nur mit *einer* Anwendung interagieren muss.

Da aktuell noch keine graphische Benutzeroberfläche für die Konfiguration vorhanden ist, müssen Anpassungen vorerst über einen Texteditor direkt in der Konfigurationsdatei vorgenommen werden. Auch für die Netzwerkkonfiguration gibt es zunächst und auf absehbare Zeit keine graphische Oberfläche, sodass hierfür die Systemverwaltungsprogramme des Betriebssystems verwendet werden müssen. Bei der Netzwerkkonfiguration ist dies auch nicht besonders problematisch, weil diese üblicherweise nach der Installation nur noch äußerst selten angepasst werden muss.



(a) Übersicht über alle Räume



(b) Ansicht eines einzelnen Raums mit Lautstärkereglern

Abbildung 4.2: Wireframe-Darstellung der Benutzeroberfläche.

Kapitel 5

Technologieauswahl

Bevor mit der eigentlichen Implementierung begonnen werden kann, ist es notwendig, die dafür zu verwendenden Technologien auszuwählen. Teilweise beeinflussen die gewählten Technologien auch die Architektur und den Entwurf der Anwendung, weshalb eine Auswahl bereits an dieser Stelle erfolgt.

5.1 Frontend

Für die Entwicklung des Frontends moderner Webanwendungen werden oftmals JavaScript-Frameworks verwendet. Zu den meist genutzten Frameworks zählen dabei Angular, React und Vue [11, S. 6], [12, S. 40–43]. Alle drei genannten Projekte sind als Open-Source-Software veröffentlicht [11, S. 54].

Bei Angular handelt es sich um ein komponentenbasiertes Framework. Das bedeutet, dass Komponenten als Bausteine der Weboberfläche verwendet werden. Diese Komponenten bestehen jeweils aus einer HTML-Datei als Vorlage für die Struktur, einer CSS-Datei für das Styling und einer TypeScript-Datei zur Steuerung der Logik [11, S. 17]. Angular wird von Google unterstützt und entwickelt [11, S. 24]. Nach anfänglichen Problemen kann Angular mittlerweile als hinreichend stabil erachtet werden [11, S. 22–24]. Als Nachteil von Angular kann die relativ steile Lernkurve angesehen werden [11, S. 25f].

Auch bei React bilden Komponenten die Grundbausteine der Anwendung. Es gibt zwei Möglichkeiten der Definition von Komponenten: *Function Components* und *Class Components* [11, S. 30f]. Während erstere einfacher zu verwenden sind [13], waren in früheren Versionen letztere notwendig, um auf alle Funktionalitäten zurückgreifen zu können [11, S. 30f]. Seit der Einführung der *Hook*-Funktionalität in Version 16.8 können alle Funktionalitäten auch in *Function Components* verwendet werden [14]. Bei der Verwendung von React bietet sich die Möglichkeit, JavaScript-Code mittels JSX um HTML-Schnipsel zu erweitern [11, S. 33]. React wird von Facebook entwickelt [11, S. 28] und kann seit vielen Jahren als hinreichend stabil für den Produktiveinsatz

angesehen werden [11, S. 36]. React zeichnet sich durch eine gut überschaubare API sowie durch hohe Flexibilität aus [11, S.36–38].

Ebenso wie bei Angular und React handelt es sich bei Vue.js (*Kurzform:* Vue) ebenfalls um ein komponentenbasiertes Framework. Eine Besonderheit sind die sogenannten „Single File Components“. Dabei werden alle Bestandteile einer Komponente, also die Template-Struktur, das eigentliche Skript sowie die Style-Informationen, in getrennten Abschnitten in einer Datei untergebracht [11, S. 43f]. Im Gegensatz zu den anderen beiden vorgestellten Frameworks wird Vue nicht von einer großen Firma entwickelt, sondern von unabhängigen Entwicklern. Trotzdem kann Vue als relativ stabil erachtet werden [11, S. 48f]. Ein großer Vorteil von Vue ist die flache Lernkurve [11, S. 50].

Wohlgethan [11, S. 63f] kommt zu dem Schluss, dass keines der drei Frameworks den jeweils anderen offensichtlich überlegen ist. Die Unterschiede liegen vielmehr in Detailfragen, die von der jeweiligen Anwendung abhängen.

Für SimpleVolumeControl ist insbesondere React geeignet, da es im Vergleich zu Vue mehr Stabilität und Flexibilität bietet. Die im Vergleich zu Angular weniger steile Lernkurve ermöglicht außerdem ein besseres Verständnis der Implementierung.

Da eine Anforderung an SimpleVolumeControl ist, Netzwerkressourcen möglichst effizient zu nutzen, bietet es sich an, anstelle von React die leichtgewichtigere Alternative Preact zu verwenden [12, S. 43]. Preact zeichnet sich durch eine geringere Größe aus, bietet aber bei Verwendung einer Kompatibilitätsbibliothek nahezu vollständige Kompatibilität zu React. Die Einsparungen sind unter anderem darauf zurückzuführen, dass für die Ereignisbehandlung auf die eingebaute Funktionalität von Internetbrowsern zurückgegriffen wird, anstatt es wie bei React neu zu implementieren [15]. Ein Nachteil von Preact ist die verhältnismäßig geringe Verbreitung [12, S. 41–43] und die damit einhergehende Gefahr, dass Preact nicht dauerhaft weiterentwickelt und gepflegt wird. Dieses Risiko scheint aber im vorliegenden Fall akzeptabel zu sein, da durch die nahezu vollständige Kompatibilität zu React auch eine Umstellung von SimpleVolumeControl auf React mit überschaubarem Aufwand möglich ist.

Next.js ist ein Framework, welches oftmals als Ergänzung zu React verwendet wird. Es beinhaltet eine Routing-Funktionalität zur Implementierung verschiedener Seiten. Mit Next.js ist es möglich, die React-Komponenten teilweise bereits serverseitig zu rendern, was unter anderem zu schnelleren Ladezeiten führt. Außerdem teilt Next.js den kompilierten Code in optimierte Blöcke auf, sodass nur der für eine Seite benötigte Code geladen werden muss und gleichzeitig die Anzahl der Netzwerkanfragen niedrig bleibt [16, S. 93f]. Insbesondere aufgrund der Vorteile bei der Netzwerknutzung wird Next.js auch für SimpleVolumeControl verwendet.

Als Programmiersprache wird nicht reines JavaScript genutzt, sondern TypeScript. TypeScript erweitert JavaScript um ein Modulsystem, Klassen, Schnittstellen und statische Typisierung [17].

5.2 Backend

Die Implementierung eines Backends für eine Webanwendung ist prinzipiell in vielen Programmiersprachen möglich. Häufig wird die Programmiersprache PHP in Verbindung mit Apache oder nginx als Webserver verwendet [18].

Eine Alternative dazu ist die Verwendung von Node.js, welches es erlaubt, JavaScript auch serverseitig zu verwenden. Damit wird es möglich, Frontend und Backend in derselben Programmiersprache zu implementieren [18].

Dies bietet einen großen Vorteil für SimpleVolumeControl, da so nicht nur eine bessere Integration des Frontends in den Webserver, sondern auch die gemeinsame Nutzung bestimmter Quellcode-Teile ermöglicht wird.

Wie die von Chaniotis, Kyriakou und Tselikas durchgeführten Untersuchungen zeigen, ist Node.js auch relativ effizient im Hinblick auf die Ressourcennutzung, was eine Verwendung auch auf leistungsschwachen Raspberry-Pi-Servern erlaubt. Der festgestellte Nachteil, dass Node.js nicht besonders effizient beim Ausliefern statischer Dateien ist, fällt bei SimpleVolumeControl nicht allzu sehr ins Gewicht, da statische Dateien nur verhältnismäßig selten ausgeliefert werden müssen. Falls sich dies später jedoch trotzdem als Problem erweisen sollte, kann immer noch auf die Verwendung von nginx für das Ausliefern statischer Dateien zurückgegriffen werden [18].

Wenn Node.js als Webserver verwendet wird, ist es üblich, als Ergänzung das Framework Express.js zu nutzen [19, S. 51]. Express.js nimmt die eingehenden HTTP-Anfragen entgegen und stellt dem Entwickler eine Schnittstelle zur Verfügung, über die Anfragen zielgenau verarbeitet und beantwortet werden können. Express.js kann durch sogenannte „Middleware“ flexibel erweitert werden [19, S. 52f]. Auch bei SimpleVolumeControl wird Node.js in Verbindung mit Express.js verwendet.

Ebenso wie das Frontend wird auch das Backend mit der Programmiersprache TypeScript umgesetzt.

5.3 Programmierschnittstelle

Bei der Programmierschnittstelle von SimpleVolumeControl handelt es sich um den Kommunikationskanal, der verwendet wird, um Nachrichten zwischen Frontend und Backend auszutauschen. Dieser soll bidirektional sein, sodass es sowohl für das Backend als auch für das Frontend möglich ist, das Senden von Nachrichten zu initiieren. Auch wenn aufgrund der geringen Nutzerzahl pro Installation oftmals nur ein geringes Nachrichtenaufkommen zu erwarten ist, so ist doch während der aktiven Verwendung punktuell mit einem stark erhöhten Nachrichtenaufkommen zu rechnen. Wenn beispielsweise Schieberegler bewegt werden, wird der jeweils aktuelle Zustand mehrmals pro Sekunde an das Backend übertragen, welches wiederum ebenfalls mehrmals pro Sekunde mit den aktuellen Mischpultdaten antwortet. Es muss berücksichtigt werden, dass dies auch bei einer schlechten Netzwerkverbindung noch möglichst stabil funktionieren soll.

5.3.1 WebSockets

Es wird also eine API-Lösung benötigt, die mit einem hohen Nachrichtenaufkommen und bidirektionalen Nachrichten umgehen kann und dabei eine effiziente Bandbreitennutzung gewährleistet. Das bereits in Abschnitt 3.2 vorgestellte WebSocket-Protokoll erfüllt diese Anforderungen [9], [10]. Außerdem kann es durch Verwendung der Erweiterung „express-ws“[1] sehr einfach in Express.js integriert werden.

Das WebSocket-Protokoll wurde entwickelt, um effiziente bidirektionale Kommunikation zwischen Webbrowser und Webserver zu ermöglichen [9]. Traditionell müssen bei HTTP die Anfragen vom Client an den Server gestellt werden, bevor dieser sie beantworten kann. Es ist also für den Server nicht möglich, selbstständig Nachrichten zu initiieren [9]. Um diese Beschränkung zu umgehen, wurden Behelfslösungen wie *Polling* und *Long Polling* entwickelt. Beim Polling stellt der Client in regelmäßigen Abständen Anfragen an den Server. Der Server beantwortet diese Anfragen sofort und kann bei Bedarf die gewünschte Nachricht in der Antwort mitsenden. Beim Long Polling wird dieses Vorgehen insofern abgeändert, als der Server die Anfrage nicht sofort beantwortet, sondern wartet, bis eine Nachricht zum Senden vorliegt oder eine Zeitüberschreitung auftritt. Der Vorteil ist hierbei, dass die Anzahl an HTTP-Anfragen reduziert wird. Insbesondere bei einem hohen Nachrichtenaufkommen wird dieser Vorteil wieder zunichtegemacht, da weiterhin für jede Nachricht eine eigene HTTP-Verbindung benötigt wird [9].

Um eine WebSocket-Verbindung aufzubauen, sendet der Client eine HTTP-Anfrage mit dem Header „Upgrade: websocket“ an den Server. Der Server sendet eine HTTP-Antwort, mit der der Verbindungsaufbau entweder bestätigt oder abgelehnt wird. Dieser Nachrichtenaustausch wird als „Opening Handshake“ bezeichnet. Nach dem Handshake wird auf das WebSocket-Protokoll gewechselt, das heißt, die Nachrichten werden nicht mehr basierend auf HTTP ausgetauscht. Vielmehr baut das WebSocket-Protokoll direkt auf TCP auf [9].

Auch wenn die Anforderungen an die Programmierschnittstelle von SimpleVolumeControl etwas anders sind als bei den meisten Webanwendungen, sollen trotzdem in den nächsten beiden Abschnitten häufig bei modernen Webanwendungen genutzte API-Technologien betrachtet werden.

[1] `https://github.com/HenningM/express-ws`

5.3.2 Representational State Transfer (REST)

Representational State Transfer (REST) wird in der Dissertation von Fielding aus dem Jahr 2000 vorgestellt. Er leitet diese Architektur aus einer gegebenen Menge an Bedingungen (englisch *constraints*) ab. Diese lauten [20]:

1. Client-Server-Architektur
2. Zustandslose Kommunikation
3. Unterstützung für Caching
4. Einheitliche Schnittstelle zwischen Komponenten
5. Mehrschichtige Systeme
6. *Optional*: Code on demand

Praktisch umgesetzt werden REST-Schnittstellen üblicherweise mit HTTP, indem Endpunkte definiert werden, über die auf Ressourcen zugegriffen werden kann [21].

Während einige der Prinzipien von REST auch auf WebSockets angewandt werden können, scheint eine Anwendung von REST in seiner Gesamtheit auf WebSockets eher unüblich.

5.3.3 GraphQL

GraphQL ist eine Abfragesprache, die von Facebook entwickelt und 2015 als Open-Source-Software veröffentlicht wurde. GraphQL wurde als Alternative zu auf REST basierenden API-Architekturen entwickelt. Bei Anfragen muss die Struktur der benötigten Daten mit angegeben werden. Durch die feingranulare Kontrolle über die abgefragten Daten kann verhindert werden, dass gar nicht benötigte Daten übertragen werden [21]. Da bei GraphQL alle Daten über einen einzigen Endpunkt erreichbar sind – gleichsam einer Datenbank – ist es ebenso möglich, die Anzahl der HTTP-Anfragen zu reduzieren, da alle benötigten Daten an einer zentralen Stelle abgefragt werden können und nicht aus mehreren Endpunkten kombiniert werden müssen [22].

Es gibt auch Bibliotheken[2], die die Verwendung von GraphQL in Kombination mit WebSockets ermöglichen.

Für SimpleVolumeControl wäre somit eine Verwendung von GraphQL zwar denkbar, würde allerdings auch Nachteile mit sich bringen. So würde sich beispielsweise die Größe der Netzwerkanfragen erhöhen, da die Struktur der abgefragten Daten mit angegeben werden muss. Außerdem würde die Verwendung zusätzlicher Bibliotheken nötig werden, was den Wartungsaufwand, beispielsweise in Bezug auf Aktualisierungen, erhöhen würde. Zusammengefasst überwiegen aktuell also die Nachteile. Falls jedoch zukünftig die Komplexität der Netzwerkschnittstelle von SimpleVolumeControl stark zunehmen würde (was derzeit nicht geplant ist), würde sich dieses Verhältnis wahrscheinlich umkehren und die Verwendung von GraphQL rechtfertigen.

[2] z.B. `https://github.com/enisdenjo/graphql-ws`

Kapitel 6

Architektur

Nachdem nun die Technologien ausgewählt sind, kann die Architektur der Anwendung entworfen werden. Zunächst wird dargestellt, wie sich SimpleVolumeControl in ein bestehendes Gesamtsystem einfügt, bevor die Grobarchitektur von SimpleVolumeControl selbst beschrieben wird. Anschließend wird die Architektur und der Entwurf von Backend, Frontend und der Netzwerkschnittstelle detailliert erläutert.

6.1 Technisches Konzept

Das zentrale Element des SimpleVolumeControl-Systems ist ein Raspberry-Pi-Einplatinenrechner. Jede Installation von SimpleVolumeControl benötigt ein eigenes Raspberry Pi. Pro Installation kann genau ein Digitalmischpult angesteuert werden. Das Raspberry Pi ist an das lokale Netzwerk angeschlossen und stellt dort die Webanwendung zur Steuerung des Mischpults bereit. Über eine zusätzliche Netzwerkschnittstelle am Raspberry Pi wird eine Verbindung zum Digitalmischpult hergestellt. Somit kann das Raspberry Pi mit dem Mischpult kommunizieren und die über die Weboberfläche getätigten Eingaben an das Mischpult weitergeben. Ebenso kann das Raspberry Pi alle Änderungen am Mischpult überwachen und so bei Bedarf aktualisierte Daten an die verbundenen Web-Clients senden. Um die Webanwendung auch ohne ein dediziertes Client-Gerät nutzen zu können und die Einbindung in das Netzwerk zu vereinfachen, wird an das Raspberry Pi ein kleiner berührungsempfindlicher Bildschirm angeschlossen. Alle Audioeingabegeräte, also beispielsweise Mikrophone, Radios und sonstige Abspielgeräte, sind direkt mit dem Mischpult verbunden, ebenso die Ausgabegeräte, also Endstufen[1] bzw. Lautsprecher. Das Raspberry-Pi-Gerät mit SimpleVolumeControl verarbeitet also keinerlei Audio-Material, sondern lediglich Steuerungsbefehle und Metainformationen.

[1] Endstufen sind Leistungsverstärker, die das Audiosignal für passive Lautsprecher (ohne Stromanschluss) anpassen. Sie nehmen eine Impedanzanpassung vor und sorgen für eine Stromverstärkung [1, S. 299]. Bei aktiven Lautsprechern (mit Stromanschluss) ist eine Endstufe in den Lautsprecher integriert [1, S. 294f].

In Abbildung 6.1 ist dieses Konzept graphisch dargestellt. Es ist zu erkennen, dass die Installation des SimpleVolumeControl-Systems keine größeren Eingriffe an einer bestehenden Infrastruktur erfordert. Sowohl bei der Tontechnik als auch beim lokalen Rechnernetz sind keine Anpassungen nötig; es wird lediglich ein zusätzliches Gerät zwischen Mischpult und lokalem Netzwerk platziert.

Alternativ zu diesem beschriebenen Konzept kann auf die zusätzliche Netzwerkschnittstelle am Raspberry Pi verzichtet werden, indem das Mischpult direkt mit dem bestehenden lokalen Netzwerk verbunden wird. Wichtig ist nur, dass eine Netzwerk-Route existiert, über die das Mischpult erreicht werden kann.

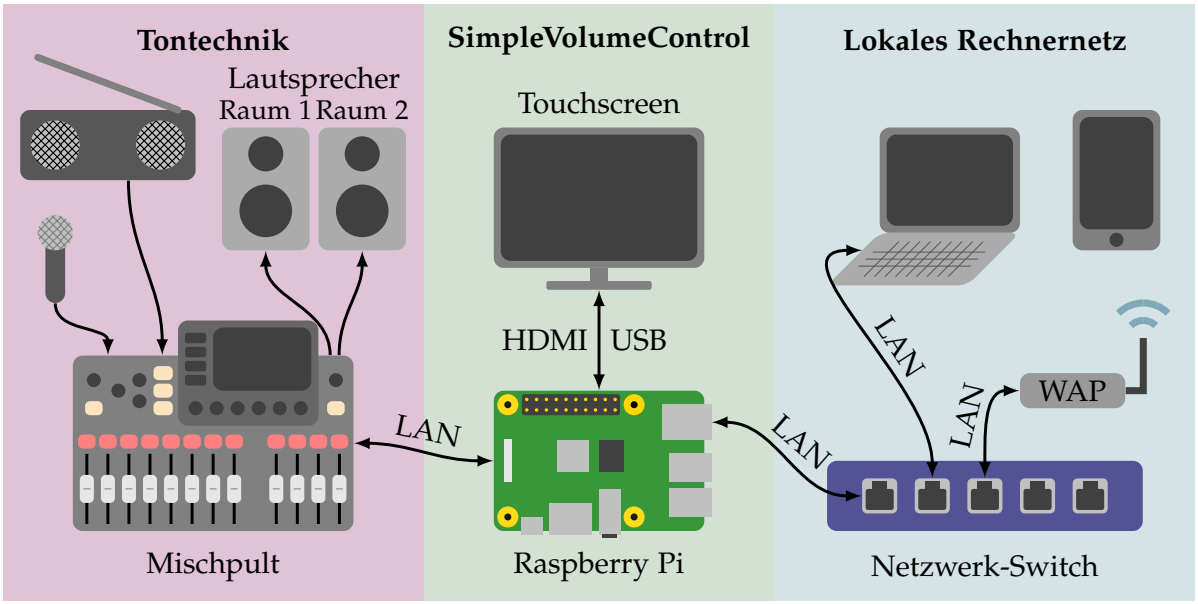


Abbildung 6.1: Übersicht über das technische Konzept von SimpleVolumeControl.

6.2 Grobarchitektur

Bei der Umsetzung dieses Konzepts wird insofern von der üblichen Dreischichtarchitektur[2] in der Webentwicklung abgewichen, als es keine Persistenzschicht im traditionellen Sinn gibt.

SimpleVolumeControl selbst ist nur für die Speicherung der Konfigurationsdaten zuständig. Dadurch, dass diese nur eine geringe Größe haben und auch nur äußerst selten angepasst werden[3], wird hierfür keine Datenbank oder Ähnliches benötigt. Stattdessen genügt es, die Konfiguration als JSON-Datei abzulegen. Aufgrund der

[2]Die Dreischichtarchitektur wurde 1992 von John J. Donovan entwickelt. Bei den drei Schichten handelt es sich um die Präsentationsschicht, die Businessschicht und die Persistenzschicht [23, Kapitel 14.2]. Eine relativ frühe Beschreibung der Dreischichtarchitektur findet sich auch bei Aarsten, Brugali und Menga [24].

[3]Vorerst gibt es nicht einmal die Möglichkeit, die Konfiguration über eine graphische Benutzeroberfläche zu bearbeiten. Die Konfigurationsdatei muss stattdessen manuell editiert werden. Für zukünftige Versionen ist allerdings ein graphischer Konfigurationseditor geplant.

Tatsache, dass die Konfiguration so selten bearbeitet werden wird, kann auf spezielle Sicherungsmaßnahmen verzichtet werden. Im Zweifelsfall ist der Benutzer, also typischerweise der technische Betreuer, dafür zuständig, die korrekte Umsetzung der Konfigurationsanpassung zu überprüfen.

Die eigentlichen Nutzdaten, die sich auch häufig ändern, sind die Lautstärke- und Stummschaltungsdaten. Dabei hat das Mischpult die Autorität über diese Daten, sie werden also auf dem Mischpult gespeichert und von SimpleVolumeControl selbst nicht persistent gespeichert. SimpleVolumeControl kann diese Daten über das OSC-Protokoll auf Basis von UDP-Sockets abfragen oder ändern. Dadurch, dass mit dem Mischpult nur über UDP kommuniziert werden kann, ist es notwendig, sämtliche Kommunikation über eine Server-Anwendung abzuwickeln. Clientseitig im Browser ist keine Kommunikation mittels UDP möglich.

Die clientseitig dargestellte Webseite kommuniziert über eine Schnittstelle mit dem Server und kann somit indirekt auf die Daten zugreifen und Änderungen veranlassen. Diese Grobarchitektur ist in Abbildung 6.2 dargestellt.

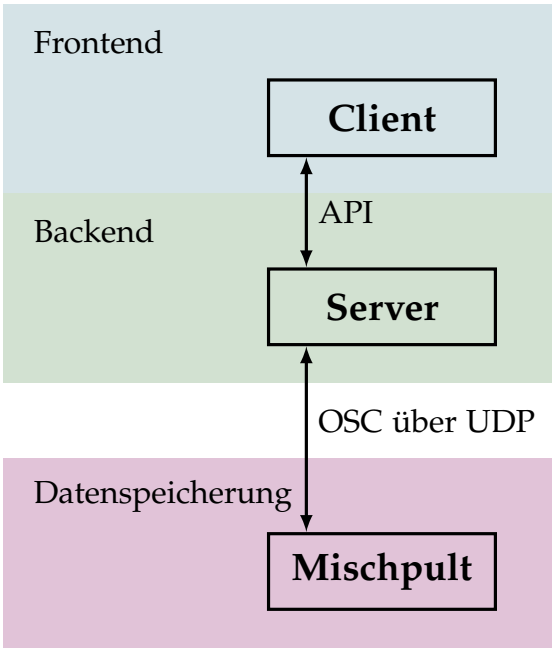


Abbildung 6.2: Die Grobarchitektur von SimpleVolumeControl.

Das Mischpult, auf dem die wesentlichen Nutzdaten persistiert werden, ist kein Bestandteil von SimpleVolumeControl im engeren Sinne. Im Folgenden wird nun eine Architektur für die beiden Bestandteile von SimpleVolumeControl, nämlich Frontend und Backend, sowie für die gemeinsame Programmierschnittstelle entwickelt.

6.3 Architektur des Backends

Um die Architektur des Backends zu entwickeln, werden zunächst wesentliche Konzepte der Anwendung identifiziert. Dazu zählen insbesondere das Mischpult im Allgemeinen (`Mixer`), das Behringer X32 im Konkreten (`BehringerX32`), die Daten von Eingängen und Ausgängen (`InputData` und `MixData`), die Konfiguration (`Config`) und die Zuweisung von Eingängen zu einem Ausgang (`MixAssignment`).

Zusätzlich wird die Klasse „Anwendung" (`App`) eingeführt, um eine klar definierte Schnittstelle zwischen Modellbereich und Webserver zu schaffen. Somit fungiert `App` als Facade Controller nach Larman [4, S. 302-314]. Dadurch braucht derjenige Teil der Webserver-Anwendung, der die Anfragen behandelt, kein Detailwissen über die einzelnen Klassen. Stattdessen ruft er bei entsprechenden Anfragen die zugehörigen Systemoperationen am Facade Controller auf.

Facade Controller dürfen nicht mit den Controllern aus dem MVC-Entwurfsmuster verwechselt werden. Erstere gehören zum Modell-Bereich und sind – unabhängig von der konkreten Benutzerschnittstelle – für die Verarbeitung der Systemoperationen verantwortlich, während MVC-Controller zur Benutzerschnittstelle gehören und dort beispielsweise für die Behandlung der Interaktionen zuständig sind [4, S. 307f].

Larman gibt zu bedenken, dass insbesondere bei der Verwendung eines Facade Controllers die Gefahr besteht, diesen zu überfrachten [4, S. 311]. Im vorliegenden Fall allerdings ist davon auszugehen, dass eine solche Situation nicht eintritt, da es zum einen lediglich eine geringe Anzahl an Systemoperationen gibt und zum anderen nur sehr wenig Funktionalität im Facade Controller selbst vorhanden ist. Die meisten Operationen werden an die jeweils zuständigen Objekte delegiert.

Auch empfiehlt Larman, spätestens beim Entwurf auf eine eigene Klasse `System` zu verzichten und die Funktionalität stattdessen in einer Klasse unterzubringen, die im Modell ohnehin schon existiert und die gewissermaßen das ganze System oder die Schnittstelle zum ganzen System repräsentiert [4, S. 303]. Im vorliegenden Fall ist dies nicht möglich, da es im Modell keine derartige Klasse gibt. Deshalb wurde behelfsmäßig die Klasse `App` eingeführt, welche die ganze Anwendung repräsentiert und als Schnittstelle zum Modellbereich fungiert.

Ein Problem dieser Bezeichnung „App" ist, dass insbesondere bei größeren Systemen mit mehreren Subsystemen eine Zuordnung schwierig werden kann. Im vorliegenden Fall wäre eine Benennung als „SimpleVolumeControl" oder „SimpleVolumeControlFacade" also präziser. Jedoch würde durch diese lange Bezeichnung der Code im Bereich der Schnittstelle zwischen Webserver und Modellbereich stark aufgebläht, da diese Klasse häufig referenziert wird. Insofern wurde – insbesondere auch vor dem Hintergrund, dass eine Unterteilung in mehrere Subsysteme in absehbarer Zeit nicht geplant ist – auf die kürzere Bezeichnung zurückgegriffen. Falls das System weiter wächst und diese Bezeichnung zu Verwirrung führt, kann eine Umbenennung dank der eingebauten Refactoring-Funktionalitäten moderner integrierter Entwicklungsumgebungen problemlos durchgeführt werden.

`App` repräsentiert tatsächlich die *ganze* Anwendung, also Frontend und Backend, im Modellbereich. Das Frontend mit der Benutzerschnittstelle wird dadurch abgebildet, dass `App` für die Behandlung der Systemereignisse, welche den Benutzerinteraktionen entsprechen, zuständig ist. Das Backend wird abgebildet, indem `App` die anderen Klassen des Modellbereichs, welche beispielsweise für die Kommunikation mit dem Mischpult zuständig sind, verwaltet.

Bei einer oberflächlichen Betrachtung kämen womöglich `Mixer` und `Config` jeweils als Facade Controller für die vermeintlichen Subsysteme „Lautstärkesteuerung durch den Endanwender“ respektive „Konfiguration durch den technischen Betreuer“ infrage. Tatsächlich erweist sich diese Idee jedoch aus mehreren Gründen als ungeeignet. Zunächst einmal ist `Mixer` im System nicht auf derjenigen Seite angesiedelt, welche vom Frontend die Systemereignisse erhält, sondern befindet sich im Gegenteil dort, wo die Schnittstelle zum realen Mischpult ist. Eine Verwendung von `Mixer` als Facade Controller würde also zu einer unverhältnismäßigen Erhöhung der Kopplung führen, da es zusätzlich eng mit dem Webserver verbunden werden müsste.

Weiterhin ist zu beachten, dass die Lebensdauer von Objekten der Klassen `Mixer` und `Config` beschränkt ist. Wenn beispielsweise die Konfiguration über die geplante Benutzeroberfläche geändert wird, ist es durchaus denkbar, dass ein komplett neues `Config`-Objekt erstellt wird. Ebenso wird ein neues `Mixer`-Objekt erstellt, wenn sich der Typ des konfigurierten Mischpults ändert. Während eine derartige begrenzte Lebensdauer kein zwingendes Ausschlusskriterium für die Verwendung als Facade Controller darstellt, so würde dies trotzdem einige Nachteile mit sich bringen. Es wäre dann nämlich möglich, dass Referenzen auf die jeweiligen Objekte veraltet sind, wenn nicht bei jedem Zugriff darauf geachtet wird, dass die gerade aktuellen Referenzen beschafft werden. Außerdem wäre weiterhin eine übergeordnete Klasse – beispielsweise `App` – nötig, die `Mixer` und `Config` verwaltet und über die Referenzen auf beide Objekte beschafft werden können. Das führt jedoch zu einer Verletzung des General Responsibility Assignment Software Pattern or Principle (GRASP) „Protected Variations“, genauer gesagt des Prinzips „Don’t talk to strangers“. Konkret bedeutet dies, dass die Instabilität der Schnittstelle des Modellbereichs zunimmt, da die (fragile) innere Objektstruktur nun auch nach außen hin relevant wird [4, S. 430–432].

Zudem würde es durch eine Delegierung der Controller-Aufgaben an `Mixer` erst recht zu einer Überfrachtung kommen, da Mixer bereits für die Kommunikation mit dem Mischpult zuständig ist.

Schließlich ist festzuhalten, dass eine derartige Trennung in Subsysteme nicht möglich ist. Erstens sind `Mixer` und `Config` von Informationen abhängig, die zum jeweils anderen Subsystem gehören. `Mixer` benötigt beispielsweise die IP-Adresse aus der Konfiguration und `Config` muss auf die Liste der verfügbaren Eingangs- und Mix-Bezeichnungen zugreifen. Zweitens ist eine Trennung auch aus Benutzersicht nicht sinnvoll, wie in Abschnitt 4.3 dargelegt wurde.

Vielmehr bietet sich `App` gerade deshalb als Fassade an, weil sie für die Verwaltung der anderen Objekte zuständig ist und damit gewissermaßen den Lebenszyklus der Anwendung steuert. Falls wider Erwarten SimpleVolumeControl in Zukunft so stark

wächst, dass `App` überfrachtet wird, bietet es sich an, auf die ebenfalls von Larman vorgeschlagenen Use Case Controller auszuweichen. Dabei wird für jeden Use Case ein eigener Controller erstellt [4, S. 303, S. 307]. Momentan und auf absehbare Zeit erscheint eine solche Trennung nach Use Cases für SimpleVolumeControl allerdings als übertrieben.

Die Klasse `App` wird als Singleton (vgl. [25, S. 127–134]) angelegt. Dies bietet sich an, da es logischerweise nur eine einzelne Instanz dieser Klasse, welche das gesamte System repräsentiert, geben kann. Ein zusätzlicher Vorteil ist, dass über die statische Methode `getInstance()` von überall her ohne größeren Aufwand eine Referenz auf diese eine Instanz beschafft werden kann.

In Abbildung 6.3 und Abbildung 6.4 sind die wesentlichen Bestandteile des Entwurfs des Modellbereichs als UML-Diagramm dargestellt. Es sind dort auch Operationen aufgeführt, deren Notwendigkeit nicht direkt aus der Anforderungsanalyse (siehe Kapitel 2) ersichtlich wird, sondern sich aus weiterführenden Überlegungen im Zuge der Implementierung ergibt.

Die Rolle von `App` als Fassade zum gesamten Modellbereich geht aus der Abbildung hervor. Die Systemoperation 1 „Kennwort eingeben“ wird von `checkPassword` abgedeckt. Eine Übersicht über alle Räume (Systemoperation 2) ist mit `getMixes` möglich; eine Raum-Einzelansicht (Systemoperation 3) mittels `getInputs`[4]. Unterstützend kann für eine Pegelanzeige `getMetersString` hinzugezogen werden. Zur Veränderung der Lautstärke oder des Stummschaltung-Status (Systemoperationen 4 und 5) können `setLevel` bzw. `setMute` genutzt werden. Das Beenden der Sitzung erfolgt durch Setzen eines leeren Kennworts, sodass die Systemoperation 6 entfällt und durch Systemoperation 1 ersetzt werden kann[5]. Zusätzlich gibt es Operationen zum Laden und Speichern der Konfigurationsdatei, zum Abonnieren von Änderungen am Mischpult sowie Hilfsoperationen, um Daten zu einzelnen Mischungen oder Eingängen abzufragen.

Die eigentliche Funktionalität hinter all diesen Operationen ist dabei in die beiden Klassen `Config` und `Mixer` ausgelagert. Die Klasse `Config` repräsentiert dabei die Konfiguration der Anwendung; es finden sich dort Informationen über das verwendete Mischpult-Modell, die IP-Adresse des Mischpults, das Kennwort sowie als Liste von `MixAssignment`-Objekten die Konfiguration der Räume/Mischungen und den ihnen jeweils zugeordneten Eingängen.

Dabei entspricht im Beispiel aus Abschnitt 4.1 ein einzelnes `MixAssignment`-Objekt jeweils einer Zeile aus Abbildung 4.1. Man könnte annehmen, dass die Liste von `MixAssignment`-Objekten auch durch einen Schlüssel-Wert-Speicher mit den Mix-Bezeichnungen als Schlüssel und der Liste an zugeordneten Eingängen als Wert ersetzt werden könnte. Dies ist allerdings nicht zutreffend, da in einem solchen Schlüssel-Wert-Speicher die Einträge ungeordnet vorliegen. Die Reihenfolge der Räume soll aber konfigurierbar sein, deshalb ist eine Liste von `MixAssignment`-Objekten nötig.

[4] Die Bezeichnungen ergeben sich aus Abschnitt 4.1. Wenn das System aus Benutzersicht beschrieben wird, wird die Raummetapher verwendet, andernfalls wird auf die allgemeineren Begriffe Mischung/Mix und Eingang/Input zurückgegriffen.

[5] Weitere Informationen dazu finden sich in Abschnitt 7.2.

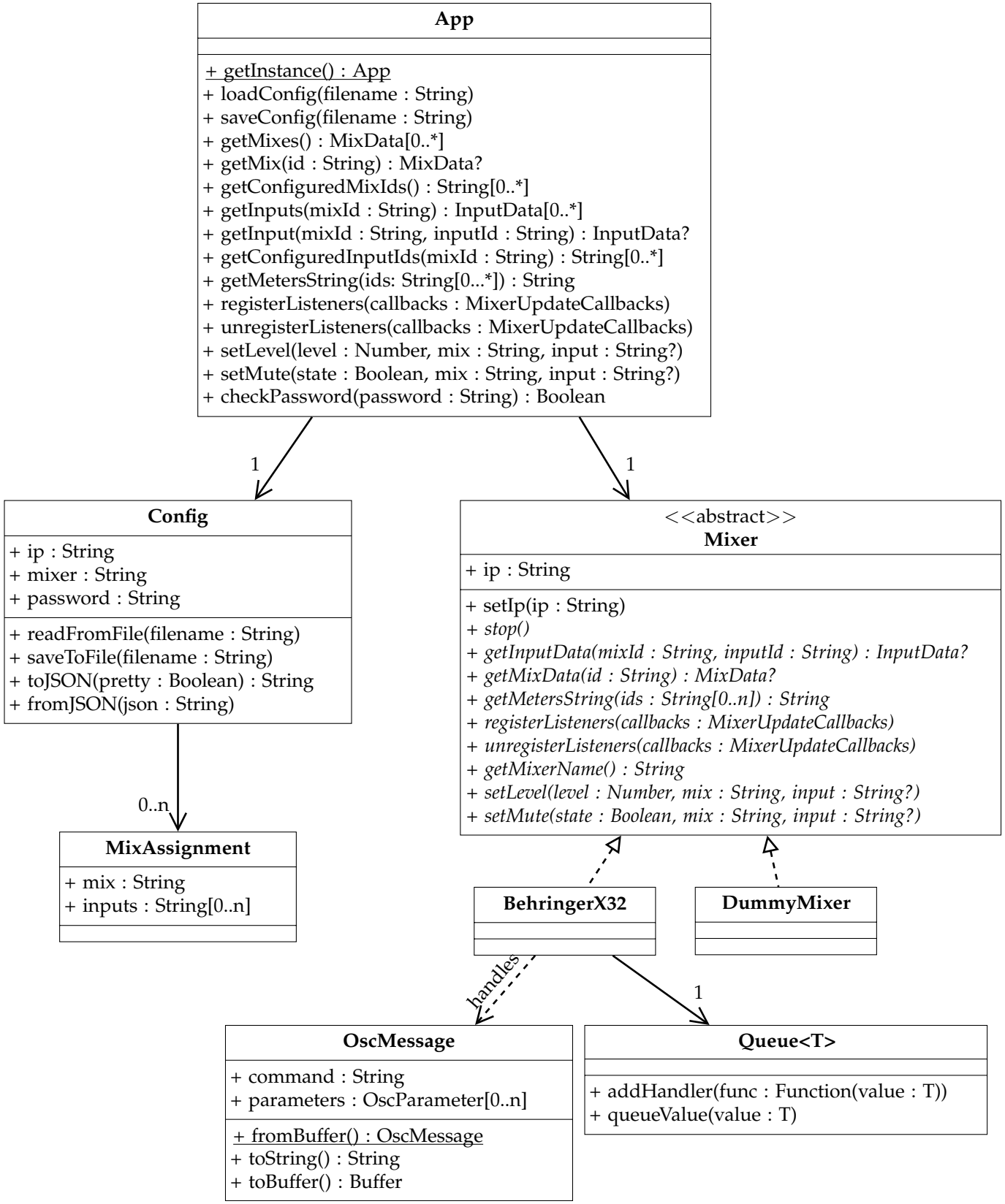


Abbildung 6.3: Die finale Fassung des Modellbereichs in der Darstellung als UML-Klassendiagramm. Eingezeichnet sind hier nur die wichtigsten Klassen mit ihren öffentlichen Eigenschaften und Operationen. Aus Platzgründen wird auch auf die Darstellung geerbter bzw. überschriebener Operationen verzichtet.

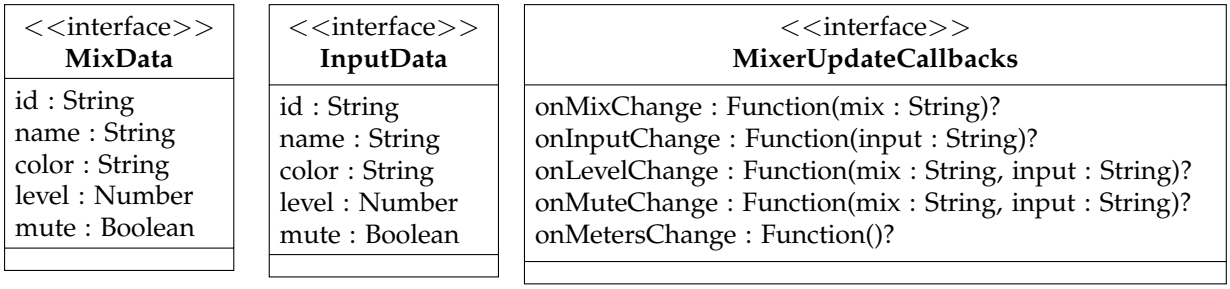


Abbildung 6.4: Die übrigen Schnittstellen im Modellbereich als UML-Klassendiagramm.

`Config` stellt Operationen bereit, mittels denen die Konfiguration als JSON-String exportiert werden kann. Ebenso gibt es eine entsprechende Funktion zum Import sowie zur Validierung der JSON-Inhalte. Unter Zuhilfenahme dieser Operationen kann die jeweils aktuelle Konfiguration in einer Datei persistiert werden. Das Speichern und Einlesen der Konfiguration wird dabei auch von `App` gesteuert.

Bei `Mixer` handelt es sich um eine abstrakte Klasse. Dies ist nötig, weil es keine einheitliche Schnittstelle zur Kommunikation mit Digitalmischpulten gibt. Stattdessen gibt es je nach Mischpulttyp unterschiedliche Schnittstellen mit verschiedenen zugrundeliegenden Protokollen. Auch beim OSC-Protokoll handelt es sich nicht um eine universell gültige Schnittstelle zur Kommunikation mit Digitalmischpulten, sondern eher um ein allgemein gehaltenes Protokoll mit unterschiedlichen Umsetzungen je nach Anwendung.

`Mixer` hat momentan zwei Konkretisierungen: `BehringerX32` und `DummyMixer`. Während bei letzterem keine tatsächliche Funktionalität vorhanden ist und diese Klasse vielmehr für Tests und als Platzhalter geeignet ist, handelt es sich bei `BehrigerX32` um die zentrale Klasse für die Kommunikation mit dem physischen Mischpult. Um die Kommunikation mit dem Mischpult abzuwickeln kommen zwei weitere Klassen zum Einsatz: `OscMessage`, welche eine einzelne Nachricht des OSC-Protokolls repräsentiert, und `Queue`, welche als Warteschlange mit einem garantierten zeitlichen Abstand zwischen zwei Elementen verwendet wird. Genauere Einzelheiten zur Kommunikation zwischen SimpleVolumeControl und dem Mischpult finden sich in Abschnitt 7.3.

Bei `MixData` und `InputData` ist zu beachten, dass keine Objekte dauerhaft vorgehalten werden, die diese Schnittstellen implementieren. Stattdessen werden bei Bedarf kurzfristig entsprechende Objekte erstellt. Beide Schnittstellen werden dabei hauptsächlich als Datentyp verwendet, um Informationen über den aktuellen Zustand einer Mischung bzw. eines einzelnen Eingangs für eine Mischung zwischen den verschiedenen Komponenten der Anwendung zu transportieren. Auch wenn die Signaturen identisch sind, werden dennoch beide Schnittstellen getrennt, da es sich auf fachlicher Ebene um unterschiedliche Konzepte handelt. Ferner muss berücksichtigt werden, dass `InputData` nicht nur vom jeweiligen Eingang abhängig ist, sondern auch von der jeweiligen Mischung. Dies ergibt sich aus der in Abschnitt 4.1 dargelegten Tatsache, dass Lautstärke- und Stummschaltedaten einem Paar aus Eingang und Mischung zugeordnet werden.

`MixerUpdateCallbacks` definiert eine Schnittstelle, um bei den verschiedenen Änderungen am Mischpult benachrichtigt zu werden. Derartige Callbacks können über den Facade Controller `App` registriert und entfernt werden. Diese Operationen werden an den jeweils aktuell verwendeten `Mixer` weitergegeben, der wiederum im Normalfall eine Liste der aktuell registrierten Callbacks vorhält.[6]

6.4 Architektur des Frontends

Aus der Verwendung von React als Framework für die Implementierung des Frontends (siehe Abschnitt 5.1) ergibt sich automatisch, dass das Frontend nach den Prinzipien der komponentenbasierten Entwicklung umgesetzt wird. Die Anwendung ist dabei aus einer hierarchischen Anordnung von Komponenten zusammengesetzt, wobei diese Komponenten die nötigen Informationen zur Präsentation sowie die Logik enthalten [26]. Es ist empfohlen, die einzelnen Komponenten klein und auf eine einzige Aufgabe fokussiert zu halten [27].

Wie bereits in Abschnitt 5.1 beschrieben, gibt es in React zwei verschiedene Möglichkeiten, Komponenten zu definieren: *Function Components* und *Class Components*. In aktuellen Versionen sind alle wesentlichen Funktionalitäten in beiden Varianten verfügbar. Es wird empfohlen, für neue Projekte vorrangig Function Components zu verwenden, da diese besser optimiert werden können und darüber hinaus leichter verständlich sind [14]. Aus diesem Grund werden für SimpleVolumeControl Function Components genutzt.

Bei React wird oftmals zwischen sogenannten *Presentational Components* und *Container Components* unterschieden. Der Unterschied ist, dass in Container Components Zustände gespeichert werden, während Presentational Components nur statisch Informationen anzeigen können [16, S. 57f]. Einige Quellen empfehlen eine relativ strikte Trennung von Presentational und Container Components (z.B. [28]), wohingegen andere diese Unterscheidung aufgegeben haben und stattdessen *Hooks* empfehlen, um Logik auszulagern [29]. SimpleVolumeControl folgt letzterer Empfehlung, das heißt, es werden Hooks verwendet, in die komplexere Anwendungslogik ausgelagert wird.

Hooks sind spezielle Funktionen, die es unter anderem erlauben, auch innerhalb von Function Components auf den Zustand zuzugreifen oder Seiteneffekte auszuführen. Beim Einsatz von Hooks müssen spezielle Regeln beachtet werden. Neben einigen vordefinierten Hooks ist auch die Verwendung selbst definierter Hooks möglich [16, S. 64–66]. Bei SimpleVolumeControl wird die Logik zur Verwendung der API in mehrere hierarchisch organisierte Hooks ausgelagert. Die Grundlage bildet der Hook `useWebSocket` aus der Bibliothek „react-use-websocket“[7]. Darauf aufbauend ist der selbstdefinierte Hook `useAuthenticatedWebSocket` für die Anmeldung mit dem Kennwort zuständig. Schließlich übernehmen die beiden Hooks `useMix` und `useMixes` die Verarbeitung der Daten für die Raumübersicht respektive die Detailansicht.

[6] Eine Ausnahme stellt `DummyMixer` dar: Da dort Änderungen keinen Effekt haben, brauchen auch die Callbacks nicht gespeichert zu werden.

[7] `https://github.com/robtaussig/react-use-websocket`

Um neben komponentengebundenen Zuständen auch global Zustände speichern zu können, kommt die Bibliothek „Recoil“[8] zum Einsatz. Damit kann das eingegebene Kennwort über die Komponenten-Grenzen hinweg verfügbar gemacht werden.

Indem Next.js verwendet wird, können die verschiedenen Seiten über Komponenten im Verzeichnis `pages` definiert werden. Dabei dienen diese Komponenten als Wurzelelement der jeweiligen Seite. Aktuell gibt es folgende Seiten:

- `index`: Raumübersicht (Startseite)
- `login`: Anmeldung (Kennworteingabe)
- `mix/<Mix Id>`: Detailansicht für einen Raum (mit `<Mix Id>` als variablem Parameter)

6.5 Entwurf der Programmierschnittstelle

Wie in Abschnitt 5.3 dargelegt, wird die Programmierschnittstelle direkt auf Basis des WebSocket-Protokolls umgesetzt. Dabei wird pro Seite (vgl. Abschnitt 6.4) ein Endpunkt bereitgestellt, über den die Verbindung aufgebaut werden kann.

Nachrichten können sowohl vom Client, als auch vom Server initiiert werden, sobald die WebSocket-Verbindung aufgebaut wurde. Es gibt verschiedene Arten von Nachrichten. Um diese zu unterscheiden, werden eindeutige Bezeichner für den Nachrichtentyp, welche im Folgenden „API-Code“ genannt werden, eingeführt. Diese bestehen nur aus Großbuchstaben und enden mit einem Doppelpunkt[9]. Sie werden den eigentlichen Nutzdaten vorangestellt. Somit ergibt sich für jede Nachricht der in Abbildung 6.5 dargestellte Aufbau.

Abbildung 6.5: Der Aufbau einer einzelnen Nachricht.

Jede Nachricht kann somit als String übermittelt werden. Die Nutzdaten, die Übermittlungsrichtung sowie der Auslöser der Nachricht unterscheiden sich dabei je nach Nachrichtentyp.

Sobald die WebSocket-Verbindung aufgebaut wurde, übermittelt der Client an den Server eine **AUTH**-Nachricht, die als Nutzlast den Hash des eingegebenen Kennworts enthält. Somit soll eine Authentifizierung ermöglicht werden.

Passt der übermittelte Hash nicht zum am Server hinterlegten Kennwort, so antwortet der Server mit einer **DEAUTH**-Nachricht. Die Nutzdaten dieser Nachricht können ignoriert werden. Erhält der Client eine solche Nachricht, so ist er angehalten, die Verbindung zu schließen und ein korrektes Passwort zu beschaffen.

[8] `https://recoiljs.org/`

[9] Der Doppelpunkt dient dazu, Typbezeichner und Nutzdaten zu trennen. Aus Gründen der besseren Lesbarkeit wird er im Folgenden teilweise weggelassen.

Wird eine Verbindung zum API-Endpunkt für die Raumübersicht geöffnet, so sendet der Server unverzüglich eine Nachricht mit dem API-Code **MIXES**. Diese Nachricht enthält eine als JSON-String serialisierte Liste von `MixData`-Objekten, die den konfigurierten Räumen entsprechen. Eine solche Nachricht wird erneut gesendet, wenn Änderungen an den konfigurierten Räumen auftreten.

Die übrigen Nachrichtentypen treten nur beim Endpunkt für die Detailansicht auf. Wird dort eine Verbindung geöffnet, so wird eine **MIX**-Nachricht gesendet, welche ein als JSON-String serialisiertes Objekt enthält. Dieses Objekt beinhaltet im Feld `inputs` eine Liste an `InputData`-Objekten, welche die dem jeweiligen Raum zugeordneten Eingänge repräsentieren. Im Feld `mix` ist das `MixData`-Objekt für den Raum selbst hinterlegt. Auch hier wird eine solche Nachricht erneut gesendet, wenn sich Änderungen am Raum selbst oder an den konfigurierten Eingängen ergeben.

Handelt es sich bei den Änderungen jedoch nur um eine Änderung der Lautstärke oder des Stummschaltungsstatus, so wird stattdessen eine **LEVEL**- bzw. **MUTE**-Nachricht gesendet. Diese Nachrichten bestehen aus einem zweiteiligen String, wobei beide Teile durch einen Schrägstrich getrennt werden. Der erste Teil ist der eindeutige Bezeichner für den betreffenden Eingang. Falls der erste Teil ein leerer String ist, so bezieht sich die Änderung auf den Mix selbst. Der zweite Teil beinhaltet den neuen Wert. Bei Lautstärkeänderungen handelt es sich dabei um Zahlenwerte zwischen null und eins, dargestellt als String mit einem Punkt als Dezimaltrenner. Bei Änderungen an der Stummschaltung werden die beiden Werte „true“ und „false“ als String verwendet.

Mehrmals pro Sekunde sendet der Server eine Nachricht mit dem API-Code **METERS** an die verbundenen Clients. Die Nutzdaten sind dabei in Base64 codiert. Damit kann pro Zeichen ein Wert im Bereich zwischen 0 und 63 transportiert werden. Der Wert des ersten Zeichens entspricht dem Pegelwert der Mischung insgesamt. Die übrigen Werte werden jeweils den Eingängen in der konfigurierten Reihenfolge zugeordnet.

Möchte der Client eine Änderung der Lautstärke oder der Stummschaltung veranlassen, so sendet er ebenfalls eine Nachricht vom Typ **LEVEL** oder **MUTE** mit dem oben beschriebenen Format. Der vollständige Entwurf der Programmierschnittstelle ist in Abbildung 6.6 und Abbildung 6.7 dargestellt.

Abbildung 6.6: Der Schnittstellenentwurf für den Endpunkt `/mixes` (mit Beispiel-Daten).

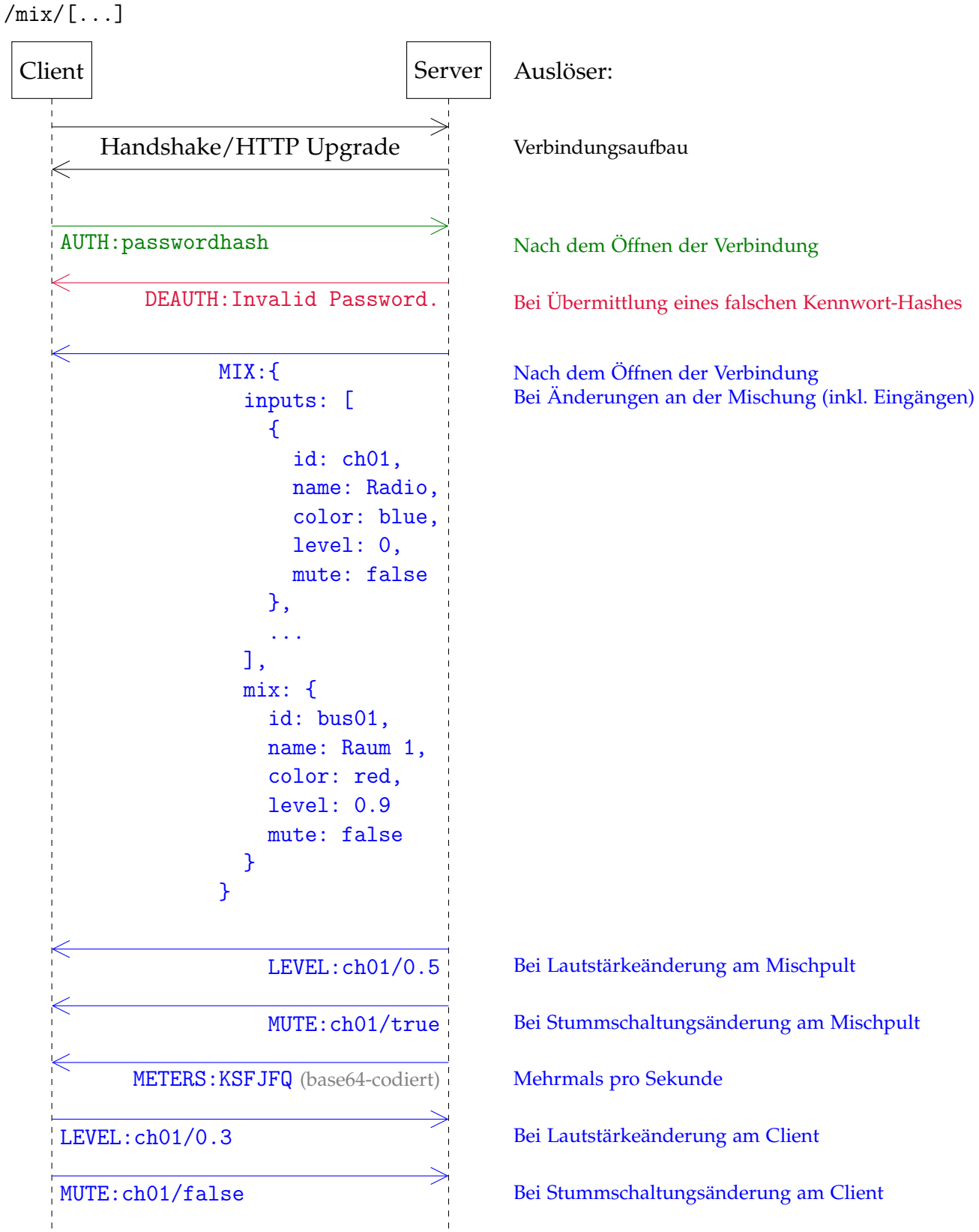


Abbildung 6.7: Der Schnittstellenentwurf für den Endpunkt `/mix/[...]` (mit Beispiel-Daten).

Kapitel 7

Technische Umsetzung

Dieses Kapitel beschäftigt sich mit der konkreten technischen Umsetzung der im vorherigen Kapitel allgemein beschriebenen Architektur. Im Folgenden werden einige interessante Aspekte, die sich aus den besonderen Anforderungen an SimpleVolumeControl ergeben, herausgegriffen und detailliert beschrieben.

7.1 Struktur des Backends

Das Backend besteht aus zwei Bestandteilen, dem eigentlichen Webserver und dem in Abschnitt 6.3 beschriebenen Modellbereich. Der Webserver übernimmt dabei sowohl die Auslieferung des Frontends als auch die Bereitstellung der API.

Es wäre zwar durchaus denkbar, diese beiden Bereiche zu trennen, das heißt, einen Webserver einzurichten, der nur das Frontend ausliefert, und einen Webserver, der nur die API bereitstellt. Bei SimpleVolumeControl allerdings würde eine solche Trennung kaum Vorteile bringen, da pro Installation sowieso nur ein einziges Gerät als Server zur Verfügung steht und auch nur mit wenigen Benutzern pro Installation zu rechnen ist. Somit gibt es keine Möglichkeit und Notwendigkeit, die Anwendung zwecks Skalierbarkeit auf mehrere Server zu verteilen. Stattdessen ergibt sich bei einem solchen Webserver, der die Programmierschnittstelle und die Auslieferung des Frontends kombiniert, der Vorteil, dass Code-Teile ohne großen Aufwand gemeinsam genutzt werden können. Außerdem wird der Startvorgang und die Verwaltung des Projekts erleichtert, da keine Abhängigkeiten zwischen beiden Bereichen gesondert betrachtet werden müssen.

Beim Start des Backends werden zunächst die Objekte des Modellbereichs initialisiert, indem eine Instanz der Klasse `App` erzeugt wird. Im Anschluss daran wird die Konfiguration eingelesen und validiert. Wie bereits in Abschnitt 5.2 beschrieben, wird Express.js mit einer WebSocket-Erweiterung als Framework für die Bereitstellung der Webanwendung verwendet. Alle Unterpfade von `/api` werden zwecks Bereitstellung der Programmierschnittstelle an einen eigenen Express.js-Router delegiert. Alle an-

deren Pfade werden von Next.js behandelt, damit das Frontend ausgeliefert werden kann.

Bei der Behandlung der WebSocket-Upgrades kommt es im Entwicklungsmodus zu einem Konflikt: Sowohl Next.js als auch die Express.js-WebSocket-Erweiterung versuchen das Upgrade durchzuführen. Dementsprechend muss die Behandlungsroutine für die WebSocket-Upgrades angepasst werden, damit die Express.js-Erweiterung bei denjenigen Pfaden, die Next.js zugeordnet sind, kein Upgrade durchführt.

Der Express.js-Router, der die API-Anfragen entgegennimmt, sorgt dafür, dass das Empfangen und Senden von WebSocket-Nachrichten entsprechend dem in Abschnitt 6.5 vorgestellten Entwurf abläuft. Es werden die Daten aus eingehenden Nachrichten ausgelesen und durch den Aufruf von Systemoperationen an den Modellbereich übergeben. Zum Senden von Nachrichten werden die benötigten Daten aus dem Modellbereich zusammengeführt und in eine geeignete Form gebracht. Der Router kommuniziert dabei ausschließlich mit der einen Instanz der Klasse `App`.

7.2 Schutz gegen versehentliche Verwendung

Üblicherweise verwendet man bei Webanwendungen HTTPS, also HTTP über TLS, um eine sichere Kommunikation zu ermöglichen, indem der Server authentisiert wird [30, S. 6]. Schwenk erklärt im Buch *Sicherheit und Kryptographie im Internet*, dass „[z]ur Authentifizierung des Servers [...] Zertifikate aus einer Public-Key-Infrastruktur (PKI) eingesetzt [werden]“ [31, S. 207]. Bei einer hierarchischen PKI sind die Zertifikate in Bäumen angeordnet. Nur die Wurzelzertifikate müssen explizit als vertrauenswürdig markiert werden. Üblicherweise enthalten Betriebssysteme und Webbrowser eine Liste mit bereits als vertrauenswürdig markierten Wurzelzertifikaten. Indem die Signaturen der Zertifikate niedrigerer Ebenen mittels Zertifikaten der nächsthöheren Ebene im Baum überprüft werden kann, kann das Vertrauen aus dem jeweiligen Wurzelzertifikat abgeleitet werden [31, S. 71f].

Für SimpleVolumeControl ergibt sich aus dieser Tatsache das Problem, dass Zertifikate, deren Vertrauenswürdigkeit sich aus den allgemein bekannten Wurzelzertifikaten ableitet, nur für öffentlich erreichbare Domain Names ausgestellt werden.

Somit bleiben für SimpleVolumeControl zwei Möglichkeiten: Ein ansonsten öffentlich erreichbarer Domain Name wird im lokalen Netz dem SimpleVolumeControl-Server zugeordnet. Dafür ist aber eine Anpassung am bestehenden lokalen Netz nötig (Router/DNS-Server), was gemäß den Anforderungen nicht erwünscht ist. Alternativ kann ein Zertifikat verwendet werden, dessen Vertrauenswürdigkeit nicht durch die allgemein bekannten Wurzelzertifikate überprüft werden kann. Wird die Anwendung dann allerdings in einem Webbrowser aufgerufen, so erscheint dann eine entsprechende Warnmeldung. Um dieses Problem zu beheben, könnte das eigene Zertifikat auf allen Client-Geräten installiert werden. Dies verletzt jedoch die Anforderung, dass keine Installation auf Client-Geräten durchgeführt werden darf. Ein Akzeptieren der Warnmeldung scheidet auch aus, da eine solche Warnmeldung zu Unverständnis und Unsicherheit beim Endanwender führen kann.

Es kann also festgehalten werden, dass eine Verwendung von HTTPS nicht möglich ist, ohne die Anforderungen zu verletzen. Somit muss auf die Verwendung von unsicherem HTTP zurückgegriffen werden. Damit ist es nicht mehr möglich, einen zuverlässigen Zugriffsschutz einzubauen, da Angreifer sämtliche Kommunikation zwischen Client und Server mitlesen und auch manipulieren können und damit mannigfaltige Möglichkeiten zur Umgehung der Schutzziele haben.

Diese Risiken sind im vorliegenden Fall aber akzeptabel, da SimpleVolumeControl nur in abgeschotteten Netzen betrieben wird, zu denen nur vertrauenswürdige Personen Zugang haben. Im Beispiel mit der Hintergrundbeschallung in einem Hotel würde es sich also um ein internes Netzwerk handeln, auf das nur die Angestellten Zugriff haben. Für Gäste des Hotels wird ein separates Gästenetz bereitgestellt.

Die Kennwortabfrage dient also im Wesentlichen als Schutz gegen versehentliche Verwendung. Konkret bedeutet dies, dass zwar Personen mit genügend technischen Kenntnissen und krimineller Energie den Zugangsschutz umgehen können, trotzdem werden aber beispielsweise Angestellte, die nicht in die Verwendung von SimpleVolumeControl eingewiesen wurden, davon abgehalten, die Anwendung aus Versehen, aus Interesse oder ähnlichen Gründen zu benützen.

Für den Fall, dass es trotzdem einen Angreifer im geschützten Netz gibt, wird das Kennwort bereits clientseitig gehasht; es tritt also gewissermaßen der Hash des Kennworts an die Stelle des Kennworts. Damit soll sichergestellt werden, dass zumindest ein nur lesender Angreifer nicht in Besitz des Originalkennworts kommt. So kann zwar der Angreifer sich mittels des Hashs gegenüber SimpleVolumeControl authentifizieren, er kann aber nicht direkt auf das eigentliche Kennwort schließen, was insbesondere dann wichtig ist, wenn – entgegen allgemeiner Empfehlungen – dasselbe Kennwort auch bei anderen Diensten verwendet wird.

Zum Hashen des Kennworts kommt der Algorithmus SHA-256 zum Einsatz. Prinzipiell wird SHA-256 vom Bundesamt für Sicherheit in der Informationstechnik als sichere Hashfunktion empfohlen [32, S. 41f]. Für das Hashen von Passwörtern im Speziellen wird jedoch der Algorithmus Argon2 empfohlen [32, S. ii, S. 75]. Allerdings ist die einzige Möglichkeit, Argon2 clientseitig im Browser zu verwenden, ein WebAssembly-Modul zu laden[1]. Dies würde aber zu einem unverhältnismäßig hohen Aufwand führen, insbesondere vor dem Hintergrund, dass das gleiche Kennwort sowieso nicht für mehrere Dienste genutzt werden sollte.

Die Implementierung der Anmelde-Funktionalität sieht wie folgt aus: Beim Laden der Webseite wird überprüft, ob ein Kennwort-Hash im LocalStorage, einem sitzungsübergreifenden lokalen Speicher im Webbrowser, hinterlegt ist. Ist dies der Fall, so wird dieser Hash verwendet, andernfalls der Hash des leeren Strings. Sobald eine Websocket-Verbindung aufgebaut wurde, sendet der Client eine AUTH-Nachricht (siehe Abschnitt 6.5) mit dem Kennwort-Hash an den Server.

[1] Folgende Bibliothek stellt das WebAssembly-Modul bereit: `https://github.com/antelle/argon2-browser`

Dort wird überprüft, ob der übermittelte Hash mit dem Hash des am Server hinterlegten Kennworts übereinstimmt. Ist dies der Fall, so wird die WebSocket-Verbindung serverseitig als authentifiziert markiert und künftige vom Client kommende Steuerbefehle werden akzeptiert. Andernfalls antwortet der Server mit einer DEAUTH-Nachricht, welche dem Client darüber informiert, dass das Kennwort falsch ist. Der Client beendet daraufhin die WebSocket-Verbindung und leitet zur Anmeldungsseite weiter. Dort kann der Benutzer ein Kennwort eingeben, von welchem der Hash im LocalStorage abgelegt wird, und wird nach Eingabe des Kennworts zur Raumübersicht weitergeleitet. Auf der Seite der Raumübersicht wird eine WebSocket-Verbindung aufgebaut, sodass dort das Kennwort dann wieder mit dem beschriebenen Mechanismus geprüft wird.

Wenn der Benutzer angemeldet ist, wird eine Schaltfläche zum Abmelden angezeigt. Dahinter verbirgt sich jedoch keine übliche Logout-Funktionalität. Vielmehr wird dadurch die Eingabe eines leeren Strings als Kennwort simuliert, was in der Regel dazu führt, dass das Kennwort falsch ist und der Benutzer zur Anmeldungsseite weitergeleitet wird.

Sofern serverseitig ein leerer String als Kennwort gesetzt wird, ist es somit möglich, die Anmelde-Funktionalität gänzlich vor dem Benutzer zu verbergen. Zu diesem Zweck wird die Schaltfläche zum Abmelden nicht angezeigt, wenn clientseitig ein leeres Kennwort gesetzt ist.

Für die Zukunft ist es denkbar, dass HTTPS zumindest als Option angeboten wird, sodass auch eine Authentifizierung, die tatsächlich Schutz bietet, umgesetzt werden kann.

7.3 Kommunikation mit dem Mischpult

Die Kommunikation mit dem einzigen bisher unterstützten Mischpulttyp, den Mischpulten der Behringer-X32-Familie, läuft über das OSC-Protokoll. Das OSC-Protokoll setzt auf UDP auf. Da UDP-Verbindungen unzuverlässig sind, muss davon ausgegangen werden, dass teilweise Nachrichten verloren gehen [33, S. 12]. Dem wird entgegengewirkt, indem SimpleVolumeControl in regelmäßigen Abständen die aktuellen Daten des Mischpults abfragt und so rollierend eine Aktualisierung der Daten und damit einhergehend eine Synchronisation der Daten erreicht wird.

Im Folgenden wird das OSC-Protokoll, so wie es bei den Mischpulten der Behringer-X32-Familie verwendet wird, beschrieben. Andere Implementierungen verwenden möglicherweise andere Dialekte von OSC. Ein `NULL`-Byte wird im Folgenden durch einen Mittelpunkt (·) dargestellt. Als Grundlage dieser Ausführungen dient das Dokument „*Unofficial X32/M32 OSC Remote Protocol*“ von Patrick-Gilles Maillot [34]. Es handelt sich dabei um eine weitgehend vollständige Beschreibung der OSC-Schnittstelle des Behringer X32.

OSC-Nachrichten setzen sich aus drei aneinandergereihten Bestandteilen zusammen: Die Nachricht beginnt mit der sogenannten „OSC-Adresse“. Diese Adresse besteht

aus ASCII-Zeichen und beginnt üblicherweise mit einem Schrägstrich[2]. Sie ordnet die Nachricht einer bestimmten Funktionalität zu. Ein Beispiel für eine derartige Adresse lautet: `/ch/01/mix/01/fader`. Die Länge der Adresse in Bytes muss ein Vielfaches von vier sein. Damit die benötigte Anzahl erreicht werden kann, werden `NULL`-Bytes zum Auffüllen verwendet. Falls die Länge der Adresse bereits durch vier teilbar ist, werden vier `NULL`-Bytes angehängt.

Es folgen die Parametertypen. Dieser Bestandteil beginnt immer mit einem Komma und gibt die Anzahl und Datentypen der Parameter an. Es gibt vier Datentypen:

- **Ganzzahl**, Abkürzung: `i`
- **Gleitkommazahl**, Abkürzung: `f`
- **String**, Abkürzung: `s`
- **Binary Large Object (BLOB)**, Abkürzung: `b`

Direkt im Anschluss an das Komma folgen die Abkürzungen zur Identifikation der Datentypen der Parameter in der Reihenfolge des Auftretens der Parameter. Auch dieser Bestandteil muss mit ein bis vier `NULL`-Bytes aufgefüllt werden, damit die Länge ein Vielfaches von vier beträgt.

Schließlich folgen die Werte der Parameter entsprechend der vorher spezifizierten Datentypen. Eine Ganzzahl oder Gleitkommazahl wird durch jeweils vier Byte, also 32 Bit, im Big-Endian-Format dargestellt. Bei Gleitkommazahlen wird dabei nur der Wertebereich zwischen null und eins, jeweils inklusive, genutzt. Ein String wird als Abfolge von ASCII-Zeichen dargestellt, wobei wieder ein bis vier `NULL`-Bytes nach dem bekannten Schema angehängt werden. Bei Parametern vom Typ BLOB geben die ersten vier Bytes die Länge der folgenden Binärdaten an, bevor diese selbst folgen. Zwischen den einzelnen Parametern finden sich keine Trennzeichen, abgesehen von den `NULL`-Bytes, mit denen jeder String-Parameter terminiert wird.

Beispiele für OSC-Nachrichten finden sich in Abbildung 7.1. In der oberen Zeile jeder Einzelabbildung ist dabei das menschenlesbare Format zu sehen, in der Zeile darunter die Hexadezimalwerte der entsprechenden Bytes.

Das Mischpult lauscht auf Port 10023 nach eingehenden UDP-Paketen [34, S. 6]. Es gibt bestimmte Befehle, auf die das Mischpult nicht direkt, sondern zeitverzögert und mehrfach antwortet. Ein Beispiel dafür ist `/xremote`. Wenn das Raspberry Pi diesen als OSC-Nachricht an das Mischpult sendet, so sendet das Mischpult sämtliche stattfindende Änderungen als OSC-Nachricht an den Socket, von dem die ursprüngliche Nachricht ausging.

Dies bedeutet in Konsequenz, dass für jede zu sendende Nachricht auch eine Information zum Verbindungsabbau hinterlegt werden muss. Bei „normalen“ OSC-Nachrichten, auf die sofort eine einmalige Antwort erfolgt, kann die Verbindung nach Erhalt dieser

[2]Entgegen dem Standard, der einen führenden Schrägstrich vorschreibt, sendet das Mischpult an bestimmten Stellen OSC-Nachrichten, bei denen dieser fehlt. Stattdessen wird die Nachricht zusätzlich durch einen Zeilenumbruch (`\n`) terminiert [34, S. 53].

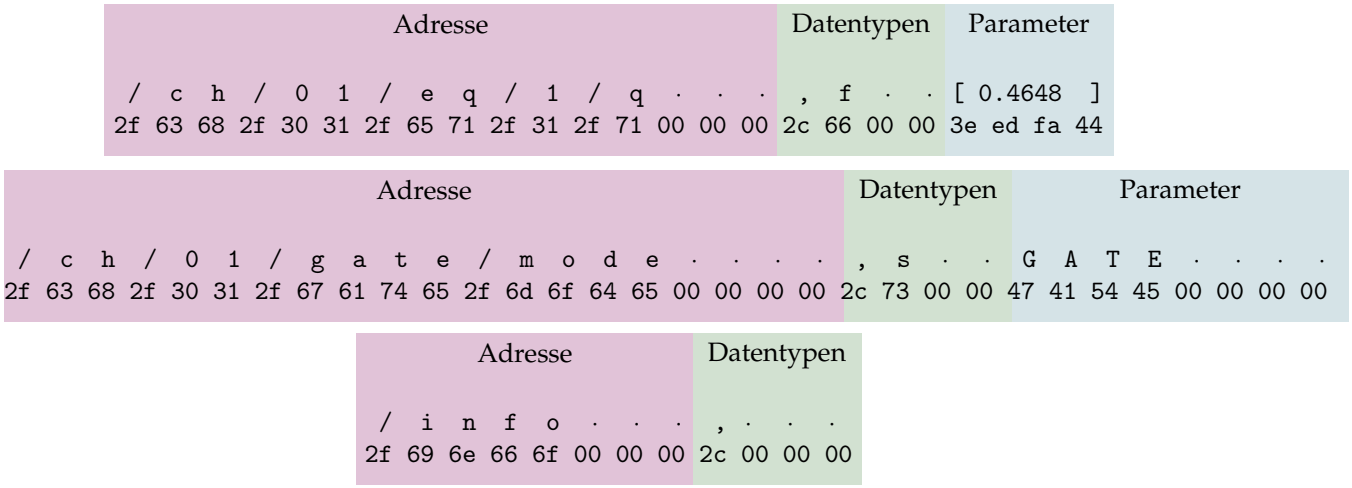


Abbildung 7.1: Beispiele für OSC-Nachrichten. Eigene Darstellung basierend auf [34, S. 11f] (mit Anpassungen)

Antwort direkt geschlossen werden. Handelt es sich aber um verzögerte Antworten, so muss das Schließen der Verbindung zeitgesteuert erfolgen.

Durch eigene Experimente konnte festgestellt werden, dass das Mischpult einen zeitlichen Mindestabstand von etwa 2 ms zwischen zwei aufeinanderfolgenden OSC-Nachrichten braucht, um einen stabilen Betrieb zu gewährleisten. Um die Einhaltung dieses Zeitabstands zu garantieren, wird die Klasse `Queue` verwendet.

Die Behandlung der OSC-Nachrichten – also Parsing und Konstruktion – erfolgt mittels der selbst implementierten Klasse `OscMessage`. Es gäbe zwar Bibliotheken für JavaScript, die eine Unterstützung für das OSC-Protokoll implementieren[3]. Allerdings kommen diese hier nicht zum Einsatz, da diese meist deutlich mehr Funktionalität enthalten als benötigt und durch eine eigene Implementierung auch flexibler auf die Eigenheiten der OSC-Implementierung des Behringer X32 eingegangen werden kann.

7.4 Auswirkung der Verwendung von Preact

Die Verwendung von Preact hat bisher auf die Entwicklung der Anwendung keinen negativen Einfluss. An den meisten Stellen ist die Verwendung von Preact vollkommen transparent, der Quellcode kann exakt so wie für React geschrieben werden. Lediglich beim Einsatz von Test-Bibliotheken ist speziell auf die Kompatibilität mit Preact zu achten, allerdings können auch hier problemlos geeignete Bibliotheken gefunden werden.

Eine zwischenzeitlich aufgetretene Inkompatibilität mit Next.js[4], welche jedoch durch die manuelle Installation eines zusätzlichen Pakets als Abhängigkeit umgangen werden konnte, ist auch behoben.

[3] z.B. `https://github.com/colinbdclark/osc.js`
[4] vgl. `https://github.com/vercel/next.js/issues/31240`

Durch Vergleichsmessungen, bei denen Preact testweise durch React ersetzt wird, kann festgestellt werden, dass die Gesamtgröße der Anwendung, die vom Browser geladen wird, durch die Verwendung von Preact auf etwa 75 % der Größe der React-Variante schrumpft. In absoluten Zahlen bedeutet dies bei einer Komprimierung mit gzip eine Einsparung von etwa 33 kB. Die Tatsache, dass solche Vergleiche ohne großen Aufwand durchgeführt werden können, zeigt auch, dass es ohne Weiteres möglich ist, zurück zu React zu wechseln, falls zukünftig Probleme mit Preact auftreten sollten.

7.5 Graphische Gestaltung des Frontends

Für die graphische Gestaltung der Benutzeroberfläche wird das Framework TailwindCSS[5] verwendet. TailwindCSS nimmt Optimierungen vor, damit nur diejenigen Bestandteile, die auch tatsächlich genutzt werden, im Produktivbetrieb ausgeliefert werden [35].

Um auf vorgefertigte Gestaltungen für Komponenten zurückgreifen zu können, wird DaisyUI[6] genutzt.

Allgemein bleibt festzuhalten, dass der Fokus bei der Gestaltung der Benutzeroberfläche auf der intuitiven Benutzbarkeit liegt. Die eigentliche graphische Gestaltung dagegen ist kein Schwerpunkt dieser Arbeit, es soll lediglich eine gut verständliche und optisch zumindest halbwegs ansprechende Oberfläche erstellt werden.

7.6 Überlegungen zur graphischen Benutzeroberfläche

Aufgrund der Tatsache, dass die Endanwender üblicherweise über keine Fachkenntnisse im Bereich der Tontechnik verfügen, muss die Benutzeroberfläche, wie in Abschnitt 2.1 dargelegt, einfach aufgebaut und intuitiv verständlich sein. Dies wird dadurch erreicht, dass entsprechend der Empfehlung von Preim und Dachselt auf Fachterminologie verzichtet und stattdessen die „Sprache des Benutzers“ verwendet wird [36, S. 15]. Im vorliegenden Fall handelt es sich dabei um die Raummetapher. Damit wird das für den Endanwender unbekannte Konzept der Mix Buses (vgl. Abschnitt 4.1) auf das bekannte Konzept der Beschallung mehrerer Räume übertragen. Die Verwendung bekannter Konzepte als Metaphern folgt auch einer Empfehlung von Preim und Dachselt [36, S. 105].

Für die Gestaltung der Anmeldemaske sind keine besonderen Vorkehrungen nötig. Das Konzept eines Passworteingabefeldes mit einer zugehörigen Bestätigungsschaltfläche kann aufgrund der weiten Verbreitung als bekannt vorausgesetzt werden. Für die Übersichtsseite zur Auswahl eines Raums wird auf eine Kachelansicht zurückgegriffen. Auch hier kann das Prinzip einer Kachelansicht zur Auswahl einer Detailansicht als bekannt angesehen werden.

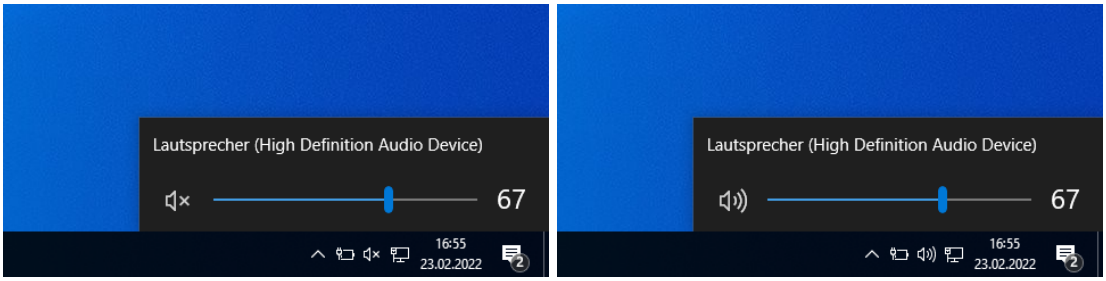
[5] `https://tailwindcss.com/`
[6] `https://daisyui.com/`

Für die Gestaltung der Lautstärkeregler in der Raum-Detailansicht wird auf die Regler zur Steuerung der Systemlautstärke in den graphischen Oberflächen gängiger Betriebssysteme zurückgegriffen. Es handelt sich um einen Schieberegler, um die Lautstärke in einem Bereich zwischen 0 % und 100 % einzustellen. Wie auch bei Preim und Dachselt beispielhaft dargestellt, wird zusätzlich zum Schieberegler der aktuell eingestellte Wert als Zahl angezeigt [36, S. 394]. Auch der Wertebereich ist wieder auf die Erfahrungen der Endanwender angepasst, da die entsprechenden Regler am Mischpult eigentlich einen Wertebereich von $-\infty$ db bis $+10$ db haben.

Der Hauptlautstärkeregler für den ganzen Raum wird durch eine farbige Hinterlegung hervorgehoben, sodass seine besondere Bedeutung erkennbar wird.

Der Stummschalteknopf birgt eine besondere Verwechslungsgefahr: Im Allgemeinen ist es bei Schaltflächen gemeinhin üblich, den Zielzustand als Beschriftung der Schaltfläche zu verwenden. Im Fall von Stummschalteknöpfen jedoch hat sich in den Benutzeroberflächen üblicher Betriebssysteme eine symbolische Darstellung des gegenwärtigen Zustands durchgesetzt. Das heißt, wenn aktuell eine Stummschaltung vorliegt, wird ein deaktivierter Lautsprecher als Symbol angezeigt und wenn keine Stummschaltung vorliegt, wird ein tönender Lautsprecher dargestellt. SimpleVolumeControl folgt dieser Praxis. Zusätzlich wird, in Anlehnung an die reale Mischpult-Hardware, der virtuelle Stummschalteknopf während einer aktiven Stummschaltung rot hinterlegt. Die physischen Stummschalteknöpfe am Mischpult leuchten ebenfalls rot, wenn eine Stummschaltung vorliegt.

In Abbildung 7.2 sind zum Vergleich Bildschirmfotos des Lautstärkereglers von Windows 10 mit und ohne Stummschaltung zu sehen. Bildschirmfotos von SimpleVolumeControl sind in Anhang B abgebildet.



(a) Lautstärkeregler mit aktivierter Stummschaltung

(b) Lautstärkeregler ohne aktivierter Stummschaltung

Abbildung 7.2: Bildschirmfotos des Lautstärkereglers von Windows 10.

Kapitel 8

Ideen zum Deployment

Eine vollständige Lösung zum Deployment von SimpleVolumeControl ist nicht Bestandteil dieser Arbeit. In den folgenden Abschnitten werden jedoch bereits einige Ideen aufgezeigt, welche die Grundlage für ein späteres Deployment-Konzept bilden können.

8.1 Nutzung von CI/CD

Für den Produktivbetrieb ist das Ziel eine hohe Stabilität sowie weitgehend automatisierte Aktualisierungen, inklusive Deployments. Um dieses Ziel zu erreichen, wird Continuous Integration/Continuous Deployment (CI/CD) angewendet.

Unter Continuous Integration versteht man eine Sammlung von Praktiken, die darauf abzielen, sämtliche Änderungen zügig in den Hauptzweig der Software zu integrieren. Dabei sind insbesondere eine hohe Testabdeckung und ein hoher Automatisierungsgrad hilfreich [37]. Fowler beschreibt dabei folgende Praktiken konkret [37]:

- Verwendung eines Repositoriums zur Verwaltung aller benötigten Quellen der Anwendung
- Automatisieren des Build-Vorgangs
- Automatisierte Tests mit hoher Abdeckungsrate
- Tägliches Integrieren der Änderungen in den Hauptzweig
- Bauen und Testen des Hauptzweigs auf einem Integrations-System bei jedem Commit
- Sofortiges Beheben fehlgeschlagener Builds
- Kurze Build-Zeiten für schnelle Rückmeldungen
- Testen in einer der Produktivumgebung möglichst ähnlichen Umgebung
- Einfaches Bereitstellen der aktuellsten ausführbaren Software-Version

- Einfache Einsehbarkeit des aktuellen Zustands
- Automatisiertes Deployment

Continuous Delivery bezeichnet eine Praktik, gemäß der die Software jederzeit für den Produktivbetrieb ausgerollt werden kann [38]. Mittels einer Pipeline können bei jedem Commit im Haupt-Repositorium automatisiert eine fest definierte Abfolge von Build- und Test-Schritten ausgeführt werden [37].

Die technische Umsetzung von CI/CD erfolgt bei SimpleVolumeControl mittels Gitlab CI[1]. Gitlab CI ist dabei so konfiguriert, dass bei jedem Commit auf jedem Zweig des Quellcode-Repositoriums eine Pipeline ausgeführt wird. Für jede Änderung wird die Pipeline also zunächst im jeweiligen Zweig ausgeführt, bevor nach dem Einpflegen der Änderung in den Hauptzweig die Pipeline erneut auf ebendiesen ausgeführt wird.

In der Pipeline erfolgt aktuell nach dem Klonen des Quellcode-Repositoriums eine Installation der Abhängigkeiten. Im Anschluss daran wird die Einhaltung der festgelegten Codierrichtlinien mittels der Werkzeuge „eslint“[2] und „prettier“[3] überprüft. Derartige Werkzeuge zur statischen Codeanalyse, welche im Englischen auch als „Linter“ bezeichnet werden, werden in Ergänzung zu Softwaretests eingesetzt und helfen unter anderem beim schnellen Aufdecken von Fehlern und der Einhaltung eines einheitlichen Stils im Code; sie tragen auch dazu bei, dass der Code gut verständlich bleibt [39].

Danach werden die Unit Tests ausgeführt. Mit diesen kann überprüft werden, ob die einzelnen Bestandteile der Anwendung wie erwartet funktionieren [16, S. 153]. Bei SimpleVolumeControl wird das Test-Framework „jest“[4] verwendet.

Aktuell beträgt die Testabdeckungsrate bezogen auf die für Unit Tests geeigneten Bereiche von SimpleVolumeControl zwischen 50 und 60 %. Als letzter Schritt in der Pipeline erfolgt ein Produktiv-Build von Frontend und Backend.

Um den fortlaufenden Wartungsaufwand für SimpleVolumeControl möglichst gering zu halten, sollen die Abhängigkeiten weitgehend automatisiert aktualisiert werden. Zu diesem Zweck wird ein Werkzeug namens „Renovate“[5] verwendet. Dabei läuft stündlich eine spezielle Pipeline, welche nach verfügbaren Aktualisierungen sucht und entsprechende *Merge Requests* (Änderungsvorschläge) für den Hauptzweig von SimpleVolumeControl stellt. Dies veranlasst die Ausführung der oben beschriebenen CI/CD-Pipeline. Wird diese fehlerfrei durchlaufen und handelt es sich bei der Änderung um keinen Major-Versionssprung[6], so wird die Aktualisierung automatisch in den Hauptzweig eingepflegt.

Wenn die Pipeline zukünftig um ein automatisches Deployment auf die Geräte im Produktiveinsatz erweitert werden soll, so muss sichergestellt werden, dass die Aktua-

[1] `https://docs.gitlab.com/ee/ci/`
[2] `https://eslint.org/`
[3] `https://prettier.io/`
[4] `https://jestjs.io/`
[5] `https://docs.renovatebot.com/`
[6] weiterführende Informationen zu *Semantic Versioning*: `https://semver.org/lang/de/`

lisierungen tatsächlich fehlerfrei funktionieren und keine unerwünschten Nebeneffekte mit sich bringen. Dazu soll zukünftig zum einen die Testabdeckungsrate bei den Unit Tests erhöht werden; zum anderen sollen aber auch zusätzliche Test-Arten eingeführt werden, da sich einige Bereiche der Anwendung nur schlecht mit Unit Tests abdecken lassen.

Infrage kommen dafür insbesondere sogenannte „Visual Regression Tests“ und „End-to-End-Tests“. Erstere operieren – im Gegensatz zu den Snapshot-Tests, welche eine Untergruppe der Unit Tests darstellen – nicht auf Basis des Markup-Codes, sondern auf Basis von Bilddateien. Von jeder zu testenden Frontend-Komponente wird ein Referenzbild gespeichert, das die Komponente so zeigt, wie sie auch im Browser dargestellt wird. Werden nun die Tests ausgeführt, so wird überprüft, ob der aktuelle Stand immer noch exakt mit dem Referenzbild übereinstimmt [40, S. 268]. Visual Regression Tests können beispielsweise mit dem Werkzeug „Loki“[7] umgesetzt werden.

End-to-End-Tests erlauben es, Benutzerinteraktionen zu simulieren und das Verhalten der Anwendung zu überprüfen [41]. Ein bekanntes Werkzeug, mit dem End-to-End-Tests umgesetzt werden können, ist Cypress[8]. Um dabei aussagekräftige Ergebnisse zu erhalten, ist es notwendig, auch die Kommunikation mit dem Mischpult in die End-to-End-Tests mit einzubeziehen. Da es sich aber aufgrund des hohen Aufwandes nicht lohnt, ein reales Mischpult dauerhaft für die Verwendung in der Pipeline bereitzuhalten, kann stattdessen auf „X32 emulator“[9], eine Software von Patrick-Gilles Maillot, zurückgegriffen werden. Diese Anwendung stellt eine dem Behringer X32 sehr ähnliche OSC-Schnittstelle bereit.

8.2 Debian-Paket-Repositorium

Für das eigentliche Deployment von SimpleVolumeControl gibt es mehrere Möglichkeiten. Bereits bei der ersten Version von SimpleVolumeControl wird auf ein eigenes Debian-Paket-Repositorium zur Verteilung der Anwendung zurückgegriffen. Ein solches Vorgehen ist auch für die Neuentwicklung von SimpleVolumeControl denkbar.

Dazu wird auf den Raspberry-Pi-Geräten das Betriebssystem „Raspberry Pi OS“[10] installiert. Dieses ist für Raspberry Pi optimiert und basiert auf Debian; es verwendet auch dasselbe Paketverwaltungssystem.

Die Idee ist nun, SimpleVolumeControl als Debian-Paket bereitzustellen. In einem solchen Paket sind neben den zu installierenden Dateien und den Metadaten auch Skripte enthalten, die beispielsweise bei der Installation oder bei Upgrades ausgeführt werden [42]. Dieses Paket kann dann mittels eines eigens eingerichteten Paket-Repositoriums bereitgestellt werden. Ein Debian-Paket-Repositorium kann beispiels-

[7] `https://loki.js.org/`
[8] `https://www.cypress.io/`
[9] `https://sites.google.com/site/patrickmaillot/x32`
[10] `https://www.raspberrypi.com/software/`

weise mittels der Software „aptly“[11] auf einem öffentlich erreichbaren Web-Server eingerichtet werden.

Zur Konfektionierung ist es also nötig, Raspberry Pi OS auf dem Einplatinenrechner zu installieren, das SimpleVolumeControl-Paket-Repositorium als Paketquelle hinzuzufügen, SimpleVolumeControl daraus zu installieren und automatische Paket-Updates zu aktivieren.

Damit kann das ganze System aktuell gehalten werden und bei Aktualisierungen von SimpleVolumeControl ist es nur nötig, die neue Version im Paket-Repositorium zu veröffentlichen; die eigentlichen Updates auf den Raspberry-Pi-Geräten im Produktivbetrieb erfolgen dann automatisch.

Ein Nachteil dieser Strategie ist, dass die Lebensdauer jeder Hauptversion von Raspberry Pi OS üblicherweise auf etwa zwei Jahre (zuzüglich eventueller verlängerter Unterstützung) begrenzt ist. Eine Aktualisierung auf die nächste Hauptversion geht oftmals mit einem erheblichen Aufwand einher, sodass eine vollständige Neuinstallation des Systems sinnvoller sein kann. In beiden Fällen ist also alle zwei Jahre eine Wartung für jedes einzelne Gerät nötig. Dies ist insbesondere dann problematisch, wenn die Anzahl an Systemen im Produktivbetrieb wächst.

8.3 Balena

Ein alternativer Ansatz wäre der Einsatz von Balena[12]. Balena erlaubt es, eine Vielzahl von Linux-Geräten orchestriert zu verwalten. Dazu wird auf den Geräten ein spezielles Betriebssystem, balenaOS, installiert. Es erlaubt, Anwendungen in Docker-Containern auszuführen, wobei das Betriebssystem auf die essentiellen Bestandteile reduziert ist [43].

Bei der Verwendung von Balena ist es notwendig, die einzelnen Geräte bei einer zentralen – im Fall der Open-Source-Variante selbst gehosteten – Verwaltungsinstanz zu registrieren [44]. Dies stellt im Vergleich zu obigen Debian-basierten Vorschlag einen Nachteil dar, da dort keine Registrierung der Geräte bei einer zentralen Instanz nötig ist. Alle entsprechend konfigurierten Geräte können bei der Debian-Variante – unabhängig davon, wer die konkrete SimpleVolumeControl-Instanz als Betreiber bereitstellt – auf das gleiche Repositorium zugreifen. Bei der Balena-Variante dagegen muss jeder Betreiber von SimpleVolumeControl eine eigene Balena-Verwaltungs-Instanz einrichten und verwenden.

Ein großer Vorteil von Balena ist, dass das Betriebssystem aus der Ferne auch bei Haupt-Versionssprüngen ohne größeren Aufwand aktualisiert werden kann; wobei auch eine Rollback-Funktionalität für den Fehlerfall bereitgestellt wird [45].

Schließlich wäre zu prüfen, ob SimpleVolumeControl in einer Docker-Container-Umgebung überhaupt stabil läuft.

[11] `https://www.aptly.info/`
[12] `https://www.balena.io/open/`

Kapitel 9

Evaluation

In diesem Kapitel soll zum einen Rückschau auf die bisherige Arbeit gehalten werden, zum anderen aber auch ein Ausblick auf mögliche zukünftige Arbeiten gegeben werden.

9.1 Diskussion

Zunächst wird versucht, die eingangs gestellten Forschungsfragen zu beantworten. Anschließend wird der aktuelle Stand von SimpleVolumeControl mit den Anforderungen aus Kapitel 2 abgeglichen.

9.1.1 Forschungsfragen

Im Rahmen dieser Arbeit sollte beobachtet werden, welche Auswirkungen die Besonderheiten der Mischpult-Schnittstelle auf den Entwurf und die Implementierung der Webanwendung haben. Dadurch, dass die Kommunikation mit dem Mischpult auf Basis von UDP erfolgt, war es nötig, einen Webserver als Vermittler zwischen Webclient und Mischpult einzusetzen. Bei der Netzwerk-API wurde ein auf effiziente Bandbreitennutzung optimierter Entwurf basierend auf WebSockets verwendet, da die Steuerungsinformationen bidirektional ausgetauscht werden, wobei auch damit zu rechnen ist, dass eine Vielzahl von Nachrichten in kurzer Zeit auftritt. Beim Entwurf der Anwendung wurde berücksichtigt, dass es zukünftig möglich sein soll, Unterstützung für weitere Mischpulttypen mit möglicherweise anderen Kommunikationsprotokollen hinzuzufügen.

Außerdem sollte untersucht werden, wie die Anwendung dahingehend optimiert werden kann, dass eine Verwendung auch bei schlechter Netzwerkverbindung möglich ist. Dazu wurden neben der bereits erwähnten WebSocket-API, welche auf effiziente Bandbreitennutzung optimiert ist, auch darauf geachtet, die Seitengröße beim initialen Laden klein zu halten. Dafür wurde anstelle von React das deutlich kleinere Preact verwendet. Außerdem wird durch die Nutzung von Next.js der zu ladende JavaScript-Code optimal auf mehrere Dateien aufgeteilt.

Im Hinblick auf die leistungsschwachen Server waren keine besonderen Vorkehrungen nötig, da auch nur mit einer geringen Nutzerzahl pro Instanz zu rechnen ist. Es musste lediglich darauf geachtet werden, dass keine Programme mit übermäßigem Arbeitsspeicherbedarf zum Einsatz kommen und keine Speicherlecks auftreten.

Um die Benutzeroberfläche auch für nicht fachkundige Personen leicht verständlich und intuitiv bedienbar zu gestalten, wurden die schwer verständlichen Konzepte aus der Tontechnik hinter der Raummetapher verborgen. Für die Gestaltung der Elemente der Benutzeroberfläche wurden für den Endbenutzer bekannte Benutzeroberflächenelemente, beispielsweise der Windows-Lautstärkeregler, als Vorlage genommen. Schließlich wurde darauf geachtet, die Benutzeroberfläche auf das Wesentliche zu beschränken.

9.1.2 Anforderungen

Alle essenziellen Anforderungen konnten im Prototyp umgesetzt werden. Es ist möglich, die Gesamtlautstärke eines jeden Raums zu regeln, ebenso die jeweiligen Eingangssignale. Zum Schutz gegen versehentliche Verwendung gibt es eine Kennwortabfrage. Es werden alle Mischpulte aus der Behringer-X32/Midas-M32-Familie unterstützt, wobei eine direkte Bedienung des Mischpults weiterhin möglich bleibt.

Durch die einfache und intuitive Gestaltung der Benutzeroberfläche kann davon ausgegangen werden, dass die geforderte Einarbeitungszeit von maximal 15 Minuten eingehalten werden kann. Es fanden zwar noch keine diesbezüglichen Tests mit Endanwendern statt, jedoch kann auf Erfahrungswerte aus der ersten Version von SimpleVolumeControl zurückgegriffen werden, da dort die Benutzeroberfläche sehr ähnlich aufgebaut war. Dort konnte die 15-Minuten-Grenze problemlos eingehalten werden.

Auch die Anforderung, dass keine Installation auf den Client-Geräten nötig sein darf, wurde berücksichtigt. Die Bereitstellung erfolgt wie gefordert über ein bestehendes lokales Netz. Bezüglich der Zuverlässigkeit wurde die Zielmarke des Dauerbetriebs mit einem Neustart pro Tag angestrebt. Ob dieses Ziel auch langfristig erreicht werden kann, konnte bisher nicht überprüft werden. Es sind jedoch keine Anzeichen erkennbar, die Zweifel an der Erreichung dieses Ziels begründen können.

Die Anwendung ist auf Raspberry-Pi-Geräten lauffähig und aufgrund der effizienten Bandbreitennutzung der API auch bei schlechter Netzwerkverbindung nutzbar.

Von den bedingten Anforderungen konnte bisher nur die Pegelanzeige umgesetzt werden. Die übrigen bedingten sowie alle optionalen Anforderungen konnten für diesen Prototyp nicht berücksichtigt werden.

9.2 Ausblick

Nachdem nun im Rahmen dieser Arbeit ein Prototyp erstellt wurde, der alle wesentlichen Anforderungen erfüllt, ist es nun in naher Zukunft nötig, eine Deployment-Strategie auszuarbeiten. Ideen dazu wurden in Kapitel 8 vorgestellt.

Danach können die weiteren bedingten und optionalen Anforderungen umgesetzt werden, wobei insbesondere ein graphischer Konfigurationseditor wichtig wäre. Weiterhin kann zukünftig eine Unterstützung für weitere Mischpulttypen hinzugefügt werden.

Sobald die Anwendung produktiv verwendet wird, kann dies beobachtet und bewertet werden. Dabei ist insbesondere die Stabilität und Zuverlässigkeit der Anwendung von Interesse. Ebenso kann analysiert werden, wie gut die Endanwender mit der Benutzeroberfläche zurechtkommen. Diesbezügliche Rückmeldungen der Endanwender können für die weitere Entwicklung berücksichtigt werden.

Kapitel 10

Zusammenfassung

In dieser Arbeit wurde eine Webanwendung names „SimpleVolumeControl“ entworfen und implementiert, mit der Digitalmischpulte auf einfache Art und Weise auch von nicht fachkundigen Personen ferngesteuert werden können. Dabei sollte auch erforscht werden, welche Auswirkungen die Besonderheiten der Mischpult-Schnittstelle haben, wie die Anwendung auch für die Verwendung bei schlechter Netzwerkverbindung optimiert werden kann, was im Hinblick auf die Verwendung eines lokalen, leistungsschwachen Servers pro Installation berücksichtigt werden muss und wie die Benutzerschnittstelle möglichst intuitiv gestaltet werden kann.

Zunächst wurde in Kapitel 1 eine Einführung in das Projekt gegeben. Es wurde erläutert, dass eine neue Version einer Webanwendung zur Fernsteuerung von Digitalmischpulten, beispielsweise im Kontext von Hintergrundbeschallungen in Hotels, entwickelt werden soll.

In Kapitel 2 wurden die Anforderungen der Endanwender und der technischen Betreuer analysiert. So ist es beispielsweise nötig, die Gesamtlautstärke von Räumen und die jeweiligen Einzelsignale regeln zu können. Es müssen alle Mischpulte der Behringer-X32/Midas-M32-Familie unterstützt werden, wobei auch eine direkte Bedienung des Mischpults weiterhin möglich bleiben muss. Um Installationen auf Client-Geräten zu vermeiden, muss die Webanwendung mittels im Dauerbetrieb laufender Raspberry-Pi-Geräte über ein bestehendes lokales Netz bereitgestellt werden. Die Anwendung muss intuitiv verwendbar sein, einen Schutz gegen versehentliche Verwendung bieten und auch bei schlechter Netzwerkverbindung nutzbar sein. Aus diesen Anforderungen werden Anwendungsfälle und Systemoperationen abgeleitet.

In Kapitel 3 wurden verwandte Arbeiten vorgestellt. Zunächst wurden verschiedene verwandte Anwendungen behandelt. Diese erlauben es zum Beispiel, alle Mischpultfunktionen fernzusteuern, spezielle Anwendungsfälle – insbesondere im Hinblick auf die Kommunikation zwischen PC und Mischpult – abzudecken, Geräte mit Unterstützung für das OSC-Protokoll ganz allgemein fernzusteuern oder die Hintergrundbeschallung mittels Spezialsystemen umzusetzen. Anschließend wurden wissenschaftliche Arbeiten zu mehreren Themengebieten vorgestellt. Dabei handelt es sich um Arbeiten, die die Verwendung von Raspberry-Pi-Geräten als Webserver

zur Steuerung anderer Hardware, verschiedene neuartige Möglichkeiten zur Gestaltung interaktiver Oberflächen zum Mischen von Audiosignalen oder die Nutzung des WebSocket-Protokolls behandeln.

Das Fachkonzept in Kapitel 4 beginnt mit einer Erklärung der Funktionalitäten des Mischpults und den entsprechenden Begriffen. Dort wurde auch die Raummetapher eingeführt, die das Konzept der *Mix Buses* besser verständlich machen sollte. Anschließend wurden die Benutzerinteraktionen anhand des Anwendungsfalls 1: „Lautstärken anpassen“ dargestellt. Zuletzt wurde die geplante Benutzeroberfläche mittels Wireframes skizziert.

In Kapitel 5 wurden die Auswahl der Technologien für Frontend, Backend und Netzwerkschnittstelle behandelt. Das Frontend wurde mit Preact, einer leichtgewichtigen Alternative zu React, in Kombination mit Next.js umgesetzt. Das Backend basiert auf Node.js und dem Framework Express.js. Für beide Bereiche wurde TypeScript als Programmiersprache gewählt. Die Netzwerkschnittstelle basiert direkt auf WebSockets, wobei auf ansonsten häufig verwendete Technologien wie REST oder GraphQL verzichtet wurde.

Die Architektur wurde in Kapitel 6 erläutert, wobei zunächst im technischen Konzept dargestellt wurde, wie SimpleVolumeControl in ein bestehendes System integriert werden kann. Anschließend wurde die Grobarchitektur von SimpleVolumeControl vorgestellt. Es handelt sich um eine Webanwendung bestehend aus den zwei Schichten Frontend und Backend. Die Datenhaltung der wesentlichen Nutzdaten übernimmt das Mischpult. Im weiteren Verlauf wurde die Architektur des Backends, des Frontends und der Netzwerkschnittstelle detailliert aufgezeigt.

Interessante Aspekte aus dem Bereich der technischen Umsetzung wurden in Kapitel 7 vorgestellt. Zunächst wurde die Struktur des Backends beleuchtet. Im Backend sind sowohl die Auslieferung des Frontends als auch die Bereitstellung der API inklusive Modellbereich in einer Anwendung zusammengefasst. Im nächsten Abschnitt wurde das Sicherheitskonzept und der Schutz gegen versehentliche Verwendung thematisiert. Aufgrund der Rahmenbedingungen kann die Webanwendung selbst nicht so gestaltet werden, dass die üblichen Schutzziele gewährleistet werden. Stattdessen wird die Anwendung nur in einem als sicher zu betrachtendem Netzwerk verwendet, zu dem nur vertrauenswürdige Personen Zugang haben. Anschließend wurde die Kommunikation mit dem Behringer-X32-Mischpult über die OSC-Schnittstelle erklärt. Daraufhin wurden die Auswirkungen der Verwendung von Preact thematisiert; es kann festgehalten werden, dass die Verwendung von Preact anstelle von React kaum negative Auswirkungen hatte. Vielmehr konnte eine Reduzierung der Gesamtgröße der vom Browser zu ladenden Javascript-Daten auf 75 % festgestellt werden. Nachdem kurz die graphische Gestaltung des Frontends thematisiert wurde, wurden die Überlegungen zur graphischen Benutzeroberfläche dargestellt. Um die Anwendung möglichst intuitiv verwendbar zu machen, wurde auf für den Endanwender bekannte Konzepte und Elemente zurückgegriffen und die Oberfläche insgesamt auf die notwendigen Bestandteile reduziert.

In Kapitel 8 wurden Ideen zum Deployment behandelt. Dabei wurde zunächst auf die Verwendung von CI/CD eingegangen, bevor zwei Alternativen für das eigentliche Deployment vorgestellt wurden. Bei der ersten Variante wird SimpleVolumeControl über ein eigenes Debian-Paket-Repositorium bereitgestellt, während bei der zweiten Variante Balena verwendet wird, um alle SimpleVolumeControl-Geräte zu verwalten.

Schließlich wurde in Kapitel 9 auf die eingangs gestellten Forschungsfragen zurückgegriffen und deren Beantwortung versucht. Auch wurde festgestellt, dass alle essenziellen Anforderungen umgesetzt werden konnten.

Literatur

[1] T. Görne, *Tontechnik*. München: Hanser, 2011.

[2] D. Proctor, „Analysis: Mixer consoles market growth fuelled by low-end“, *Installation*, R. Lane, Hrsg., 16. Aug. 2018. Adresse: `https://www.installation-international.com/business/analysis-mixer-consoles-market-growth-fuelled-by-low-end` (besucht am 27. 01. 2022).

[3] H. Balzert, *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*. Heidelberg: Spektrum Akademischer Verlag, 2009.

[4] C. Larman, *Applying UML and Patterns*. Upper Saddle River, New Jersey: Prentice Hall, 2004.

[5] D. Kehagias und D. Nini, „Home Automation Based on an Android and a Web Application Using Raspberry Pi“, *American Journal of Mobile Systems, Applications and Services*, Jg. 1, S. 174–181, Dez. 2015.

[6] J. P. Carrascal und S. Jordà, „Multitouch Interface for Audio Mixing“, in *Proc. of New Interfaces for Musical Expression*, Oslo, 2011.

[7] C. Dewey und J. P. Wakefield, „Novel designs for the audio mixing interface based on data visualisation first principles“, in *140th International AES Convention*, Paris, Mai 2016.

[8] M. Cartwright, B. Pardo und J. Reiss, „MIXPLORATION: Rethinking the Audio Mixer Interface“, in *Proceedings of the 19th International Conference on Intelligent User Interfaces*, Haifa, Israel: Association for Computing Machinery, Feb. 2014, S. 365–370.

[9] L. Srinivasan, J. Scharnagl und K. Schilling, „Analysis of WebSockets as the New Age Protocol for Remote Robot Tele-operation“, in *IFAC Proceedings Volumes*, 3rd IFAC Symposium on Telematics Applications, Bd. 46, Seoul, Nov. 2013, S. 83–88.

[10] D.-H. Mun, M. L. Dinh und Y.-W. Kwon, „An Assessment of Internet of Things Protocols for Resource-Constrained Applications“, in *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, Bd. 1, Atlanta, Georgia, Juni 2016, S. 555–560.

[11] E. Wohlgethan, „Supporting Web Development Decisions by Comparing Three Major JavaScript Frameworks: Angular, React and Vue.js“, Bachelorthesis, Hochschule für Angewandte Wissenschaften Hamburg, 2018. Adresse: `https://reposit.haw-hamburg.de/handle/20.500.12738/8417`.

[12] M. Levlin, „DOM benchmark comparison of the front-end JavaScript frameworks React, Angular, Vue, and Svelte“, Masterthesis, Åbo Akademi University, Turku, Finnland, 2020. Adresse: `https://www.doria.fi/handle/10024/177433`.

[13] React Docs, *Components and Props*, 27. Jan. 2022. Adresse: https://reactjs.org/docs/components-and-props.html.
[14] React Docs, *Introducing Hooks*, 26. Juli 2021. Adresse: https://reactjs.org/docs/hooks-intro.html.
[15] Preact. „Preact's Goals". (31. Aug. 2021), Adresse: https://preactjs.com/about/project-goals.
[16] M. Thakkar, *Building React Apps with Server-Side Rendering*. Berkeley, Kalifornien: Apress, 2020.
[17] G. Bierman, M. Abadi und M. Torgersen, „Understanding TypeScript", in *ECOOP 2014 – Object-Oriented Programming*, R. Jones, Hrsg., Uppsala, Schweden: Springer, 2014, S. 257–281.
[18] I. K. Chaniotis, K.-I. D. Kyriakou und N. D. Tselikas, „Is Node.js a viable option for building modern web applications? A performance evaluation study", *Computing*, Jg. 97, Nr. 10, S. 1023–1044, März 2014.
[19] A. Mardan, *Practical Node.js*. Berkeley, Kalifornien: Apress, 2018.
[20] R. T. Fielding, „REST: Architectural Styles and the Design of Network-based Software Architectures", Dissertation, University of California, Irvine, 2000.
[21] G. Brito und M. T. Valente, „REST vs GraphQL: A Controlled Experiment", in *2020 IEEE International Conference on Software Architecture (ICSA)*, Salvador, Brasilien, Feb. 2020, S. 81–91.
[22] G. Brito, T. Mombach und M. T. Valente, „Migrating to GraphQL: A Practical Assessment", in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Hangzhou, China, Feb. 2019.
[23] J. R. Vacca, *Cloud Computing Security: Foundations and Challenges*, 2. Aufl. Boca Raton, Florida: CRC Press, 2021.
[24] A. Aarsten, D. Brugali und G. Menga, „Patterns for Three-Tier Client/Server Applications", in *Proceedings of the PLoP 1996 Conference*, Monticello, Illinois, Dez. 1996.
[25] E. Gamma, R. Helm, R. E. Johnson und J. Vlissides, *Design Patterns*. Addison-Wesley, 1994.
[26] H. Tamer, D. van den Bongard und F. Beck, „Visually Analyzing the Structure and Code Quality of Component-based Web Applications", in *2021 Working Conference on Software Visualization (VISSOFT)*, Luxemburg, 2021, S. 160–164.
[27] React Docs, *Thinking in React*, 13. Okt. 2021. Adresse: https://reactjs.org/docs/thinking-in-react.html.
[28] B. Westfall. „Leveling Up With React: Container Components". (12. Apr. 2017), Adresse: https://css-tricks.com/learning-react-container-components/.
[29] D. Abramov. „Presentational and Container Components". (17. Feb. 2019), Adresse: https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0.
[30] Bundesamt für Sicherheit in der Informationstechnik, „Kryptographische Verfahren: Verwendung von Transport Layer Security (TLS), BSI TR-02102-2", Technische Richtlinie, 11. Feb. 2022. Adresse: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102-2.pdf?__blob=publicationFile&v=2.

[31] J. Schwenk, *Sicherheit und Kryptographie im Internet*. Wiesbaden: Springer Vieweg, 2020.

[32] Bundesamt für Sicherheit in der Informationstechnik, „Kryptographische Verfahren: Empfehlungen und Schlüssellängen, BSI TR-02102-1“, Technische Richtlinie, 24. März 2021. Adresse: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile&v=2.

[33] L. Eggert und G. Fairhurst, *Unicast UDP Usage Guidelines for Application Designers*, RFC 5405, Nov. 2008. Adresse: https://datatracker.ietf.org/doc/html/rfc5405.

[34] P.-G. Maillot, *Unofficial X32/M32 OSC Remote Protocol, OSC protocol implementation for the X32/M32 Digital Mixing Console families*, 8. Jan. 2022. Adresse: https://sites.google.com/site/patrickmaillot/x32.

[35] TailwindCSS Docs, *Optimizing for Production*, 17. Jan. 2022. Adresse: https://tailwindcss.com/docs/optimizing-for-production.

[36] B. Preim und R. Dachselt, *Interaktive Systeme, Band 1: Grundlagen, Graphical User Interfaces, Informationsvisualisierung*. Berlin, Heidelberg: Springer, 2010.

[37] M. Fowler. „Continuous Integration“. (1. Mai 2006), Adresse: https://martinfowler.com/articles/continuousIntegration.html.

[38] M. Fowler. „Continuous Delivery“. (30. Mai 2013), Adresse: https://martinfowler.com/bliki/ContinuousDelivery.html.

[39] K. F. Tómasdóttir, M. Aniche und A. van Deursen, „Why and How JavaScript Developers Use Linters“, in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Urbana-Champaign, Illinois, 2017, S. 578–589.

[40] H. Tanno, Y. Adachi, Y. Yoshimura, K. Natsukawa und H. Iwasaki, „Region-based Detection of Essential Differences in Image-based Visual Regression Testing“, *Journal of Information Processing*, Jg. 28, S. 268–278, 2020.

[41] S. Bose. „End To End Testing: A Detailed Guide“, BrowserStack. (19. Mai 2021), Adresse: https://www.browserstack.com/guide/end-to-end-testing.

[42] L. Nussbaum. „Debian Packaging Tutorial“. (3. Nov. 2021), Adresse: https://www.debian.org/doc/manuals/packaging-tutorial/packaging-tutorial.en.pdf.

[43] Balena, *balenaOS, Run Docker containers on embedded IoT devices*. Adresse: https://www.balena.io/os/ (besucht am 03. 03. 2022).

[44] Balena, *OpenBalena Getting Started Guide*. Adresse: https://www.balena.io/open/docs/getting-started/ (besucht am 03. 03. 2022).

[45] Balena Docs, *Update process details*, 16. Nov. 2021. Adresse: https://www.balena.io/docs/reference/OS/updates/update-process/.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

API Application Programming Interface. 21, 23, 24, 27, 33–35, 38, 39, 50, 51, 54

ASCII American Standard Code for Information Interchange. 42

BLOB Binary Large Object. 42

CI/CD Continuous Integration/Continuous Deployment. vii, 46, 47, 55

CSS Cascading Style Sheets. 20

DNS Domain Name System. 39

GPIO General Purpose Input/Output. 10

GRASP General Responsibility Assignment Software Pattern or Principle. 29

GUI Graphical User Interface, englisch für graphische Benutzeroberfläche. 7

HDMI High-Definition Multimedia Interface. 26

HTML Hypertext Markup Language. 20

HTTP Hypertext Transfer Protocol. 11, 22–24, 39, 40

HTTPS Hypertext Transfer Protocol Secure. 39–41

JSON JavaScript Object Notation. 26

JSX JavaScript Syntax Extension *oder* JavaScript XML, eine aus dem Umfeld des React-Frameworks stammende Erweiterung von JavaScript um HTML-ähnliche Elemente. 20

LAN Local Area Network. 26

MVC Model View Controller (Architektur-/Entwurfsmuster). 28

OSC Open Sound Control. vi, 10, 27, 32, 41–43, 48, 53, 54, 59

PC Personal Computer. 9, 10, 53

PHP PHP: Hypertext Preprocessor. 22

PKI Public-Key-Infrastruktur. 39

REST Representational State Transfer. vi, 24, 54

SHA Secure Hash Algorithm. 40

TCP Transmission Control Protocol. 11, 23

TLS Transport Layer Security. 39

UDP User Datagram Protocol. 27, 41, 42, 50

UML Unified Modeling Language. 30–32, 59

USB Universal Serial Bus. 26

WAP Wireless Access Point. 26

WLAN Wireless Local Area Network. 5

Anhang A

Anwendungsfälle

Anwendungsfall 1: **Lautstärken anpassen**

Siehe Abschnitt 4.2

Anwendungsfall 2: **Lautstärkepegel überwachen**

Priorität: mittel

Beschreibung: Als Person, welche für die Hintergrundbeschallung zuständig ist, (z.B. Hotelbetreiber, Personal, ...) möchte ich die tatsächlichen Pegel der verschiedenen Signale sehen können, um zu überwachen, dass wirklich Audiosignale verarbeitet und ausgegeben werden.

Vorbedingung: Das Steuerungssystem und das Mischpult sind eingeschaltet und fertig eingerichtet. Der Benutzer hat die Startseite der Anwendung im Internetbrowser geöffnet.

Nachbedingung: *Keine*

Normaler Ablauf:

1. Das System fragt nach dem Kennwort zur Anmeldung.
2. Der Benutzer meldet sich durch Eingabe des Kennworts an.
3. Das System zeigt eine Übersicht zur Auswahl der Räume.
4. Der Benutzer wählt denjenigen Raum aus, welchen er überwachen möchte.
5. Das System zeigt den Gesamtpegel des Raumes sowie die Pegel der Einzelsignale an.
6. Das System aktualisiert die Pegelanzeigen regelmäßig.
7. Der Benutzer meldet sich ab.
8. Das System beendet die Sitzung.
9. Ende.

Ablaufvarianten:

1a Der Benutzer ist bereits angemeldet.

1. Das System fragt nicht nach dem Kennwort zur Anmeldung.
2. Weiter bei 3 im normalen Ablauf.

7a Der Benutzer möchte einen anderen Raum überwachen.

1. Der Benutzer meldet sich nicht ab.
2. Der Benutzer wechselt zurück zur Raumübersicht.
3. Weiter bei 3 im normalen Ablauf.

Anwendungsfall 3: **Mischpult direkt nutzen**

Priorität: hoch

Beschreibung: Als Veranstaltungstechniker möchte ich weiterhin das Mischpult direkt oder über die offizielle Steuerungsanwendung des Herstellers nutzen können, um Zugriff auf alle Einstellungsmöglichkeiten zu haben.

Vorbedingung: Das Mischpult ist eingeschaltet. Falls die offizielle Steuerungsanwendung des Herstellers verwendet werden soll, so ist diese bereits über ein Netzwerk mit dem Mischpult verbunden.

Nachbedingung: Das System übernimmt die Änderungen, welche direkt am Mischpult oder über die offizielle Steuerungsanwendung des Herstellers getätigt wurden.

Normaler Ablauf:

1. Der Benutzer nimmt Änderungen direkt am Mischpult vor.
2. Das System übernimmt relevante Änderungen an den Parametern sofort.
3. Das System übernimmt in regelmäßigen Abständen die Parameter des Mischpults, um möglicherweise verpasste Aktualisierungen auszugleichen.
4. Ende.

Ablaufvarianten:

1a Der Benutzer nimmt Änderungen über die offizielle Steuerungsanwendung der Herstellers vor.
1. Der Benutzer nimmt Änderungen über die offizielle Steuerungsanwendung der Herstellers vor.
2. Weiter bei 2 im normalen Ablauf.

2a Das System ist ausgeschaltet.
1. Das System übernimmt die Parameter vom Mischpult beim nächsten Start.
2. Ende.

2b Das System ist ausgeschaltet. Beim nächsten Start des Systems ist aber das Mischpult ausgeschaltet.
1. Das System wird gestartet.
2. Das System versucht, die Parameter vom Mischpult zu übernehmen.
3. Die Übernahme der Parameter schlägt fehl, da das Mischpult nicht eingeschaltet ist.
4. Das System versucht regelmäßig, die Parameter vom Mischpult zu übernehmen.
5. Ende.

Die folgenden Anwendungsfälle mit niedrigerer Priorität werden nicht mehr vollständig beschrieben.

Anwendungsfall 4: **Anwendung konfigurieren (Raumkonfiguration und weitere Parameter)**

Priorität: mittel

Beschreibung: Als technischer Betreuer möchte ich die Anwendung konfigurieren, um sie den individuellen Gegebenheiten anzupassen.

Anwendungsfall 5: **System (inkl. Netzwerk) einrichten**

Priorität: niedrig

Beschreibung: Als technischer Betreuer möchte ich die ganze Einrichtung aus der Anwendung heraus durchführen (inkl. Netzwerkkonfiguration), um das Setup zu erleichtern.

Anwendungsfall 6: **Weitere Audio-Parameter steuern**

Priorität: niedrig

Beschreibung: Als Person, welche für die Beschallung zuständig ist und Grundkenntnisse in der Tontechnik besitzt, möchte ich die Paramter für Equalizer, Gates, Kompressoren etc. anpassen, um mehr Kontrolle über den Klang zu haben.

Anhang B

Bildschirmfotos der Benutzeroberfläche

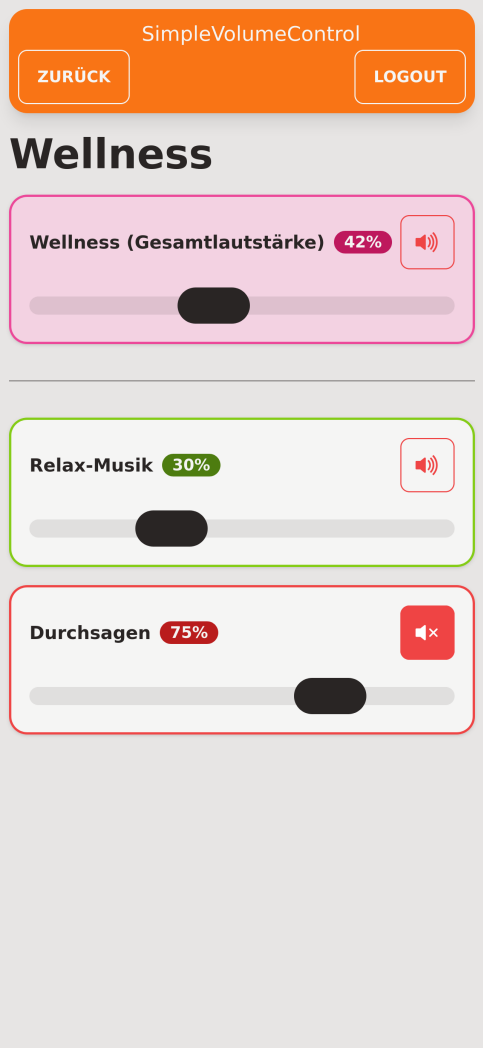
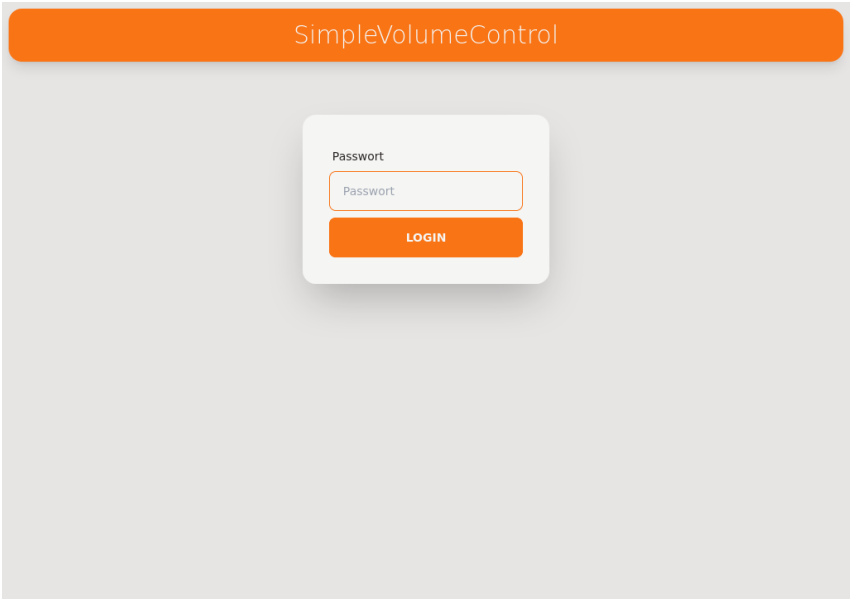
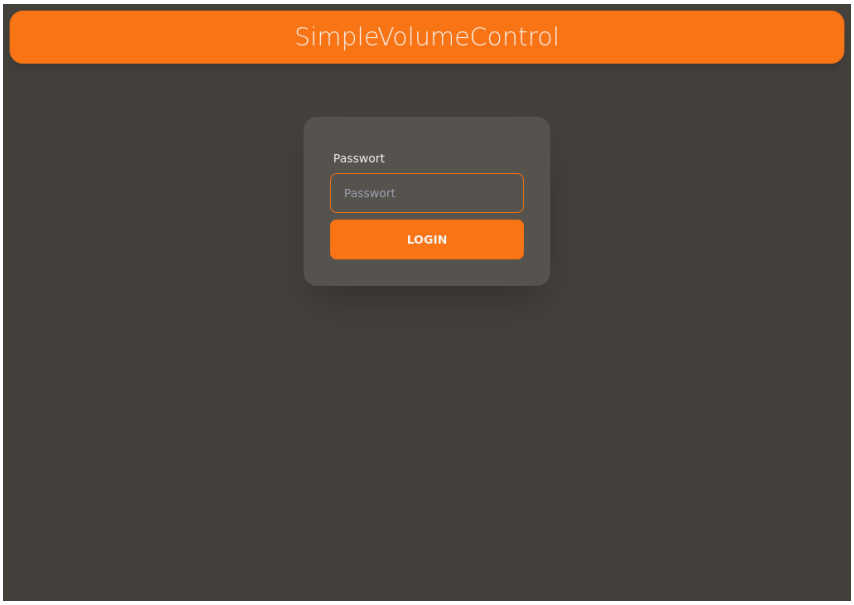


Abbildung B.1: Raum-Detailansicht mit hellem Farbschema auf einem Mobilgerät.

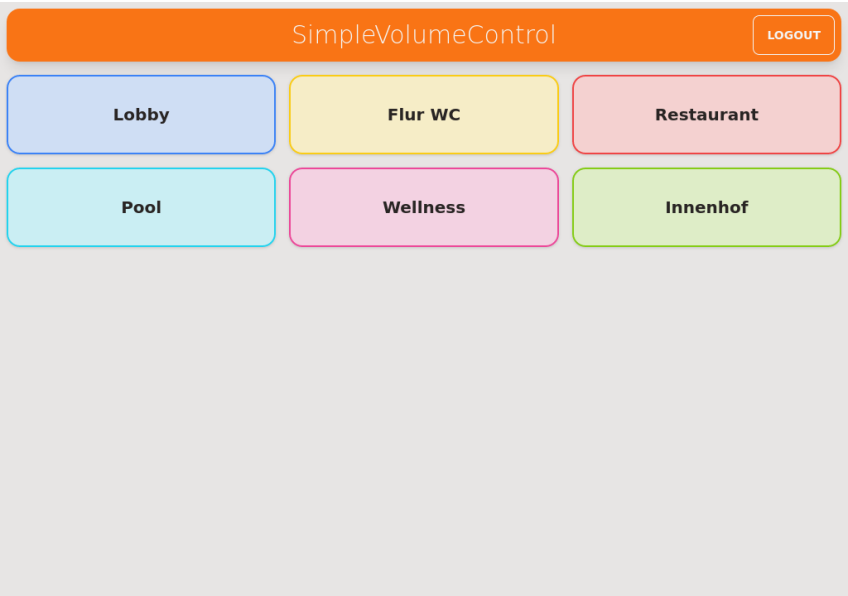


(a) Helles Farbschema.



(b) Dunkles Farbschema.

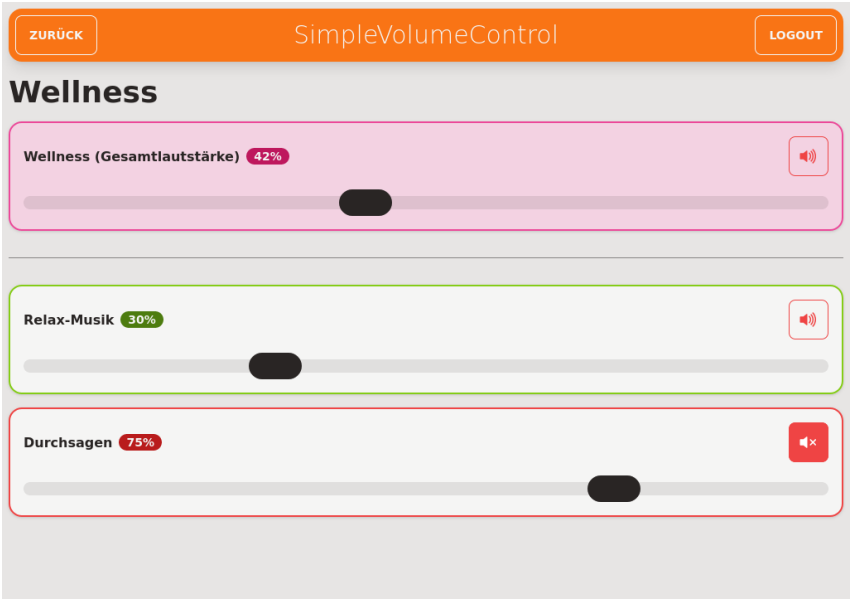
Abbildung B.2: Anmeldeseite mit hellem und dunklem Farbschema.



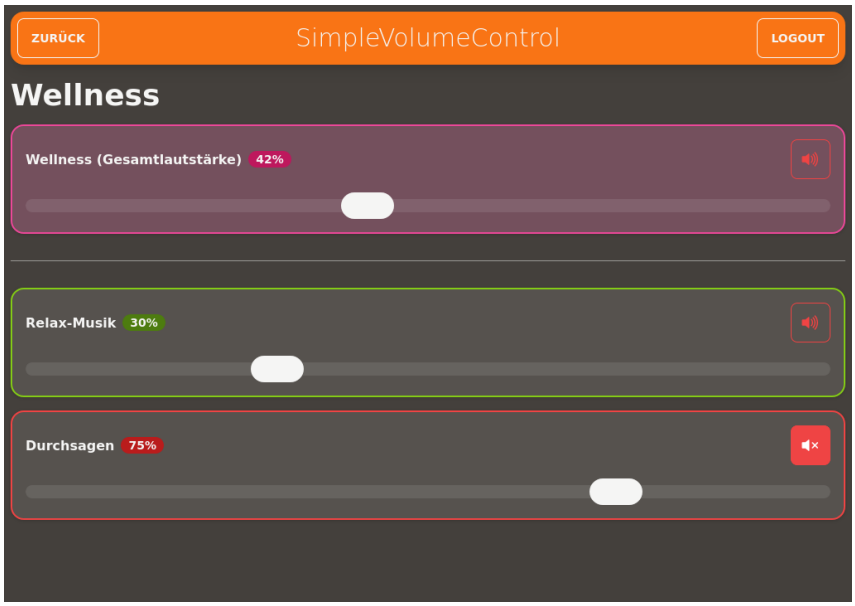
(a) Helles Farbschema.

(b) Dunkles Farbschema.

Abbildung B.3: Übersichtsseite mit hellem und dunklem Farbschema.



(a) Helles Farbschema.



(b) Dunkles Farbschema.

Abbildung B.4: Raum-Detailansicht mit hellem und dunklem Farbschema.