Multi-Room Audio Adapter

Nico Lang, Philipp Immler

Februar 2025

1 Projekt

1.1 Projektteam

Nico Lang
Wirtschaftsingenieure/Betriebsinformatik
6651 Häselgehr AT
Nico.Lang@hak-reutte.ac.at

Philipp Immler
Wirtschaftsingenieure/Betriebsinformatik
6682 Vils AT
Philipp.Immler@hak-reutte.ac.at

1.2 Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen Hilfsmittel als die angegebenen benützt habe. Die Stellen, die anderen Werken (gilt ebenso für Werke aus elektronischen Datenbanken oder aus dem Internet) wörtlich oder sinngemäß entnommen sind, habe ich unter Angabe der Quelle und Einhaltung der Regeln wissenschaftlichen Zitierens kenntlich gemacht. Diese Versicherung umfasst auch in der Arbeit verwendete bildliche Darstellungen, Tabellen, Skizzen und Zeichnungen. Für die Erstellung der Arbeit habe ich auch Hilfsmittel generativer KI-Tools (ChatGPT 3.5) zu folgendem Zweck verwendet: Inspiration und grobe Informationsbeschaffung. Auch Übersetzer (DeepL) wurden zur Hilfe genommen. Die verwendeten Hilfsmittel wurden vollständig und wahrheitsgetreu inkl. Produktversion und Prompt ausgewiesen.

. . . . . .
Ort, Datum

. . . . . .
Nico Lang

. . . . . .
Philipp Immler

1.3 Abstract Deutsch

Die vorliegende Diplomarbeit beschäftigt sich mit der Entwicklung eines Multi-Room Audio Adapters. Ein einzelner Adapter besitzt dabei die Funktion, einen Audiostream aus dem Internet auf einen Line-Output auszugeben. So lassen sich die Adapter beispielsweise mit aktiven Lautsprechern per Klinkenkabel verbinden, um so Musik in voneinander getrennten Räumen abzuspielen. Welche Musik in welchem Raum spielt, ist frei konfigurierbar. Dabei erfolgt die Konfiguration per Smartphone-App. Das Endergebnis der Diplomarbeit sind somit der Adapter selbst (physischer Prototyp) und die dazugehörige Smartphone-App.

Die Diplomarbeit lässt sich grob in die drei Teile Planung, Entwicklung und Testen/Fehlerbehebung gliedern.

1.4 Abstract English

This diploma thesis deals with the development of a multi-room audio adapter. A single adapter has the function of outputting an audio stream from the Internet to a line output. For example, the adapters can be connected to active speakers via a jack cable in order to play music in separate rooms. Which music plays in which room is freely configurable. The configuration is done via smartphone app. The final result of the thesis is the adapter itself (physical prototype) and the corresponding smartphone app.

The thesis can be roughly divided into three parts: planning, development and testing/troubleshooting.

1.5 Danksagung

Wir bedanken uns bei den betreuenden Lehrpersonen Dipl. Ing. Dr. Peter L. Steger und Dipl. Päd. Johannes Köll für die kompetente Unterstützung.

Inhaltsverzeichnis

2 Einleitung

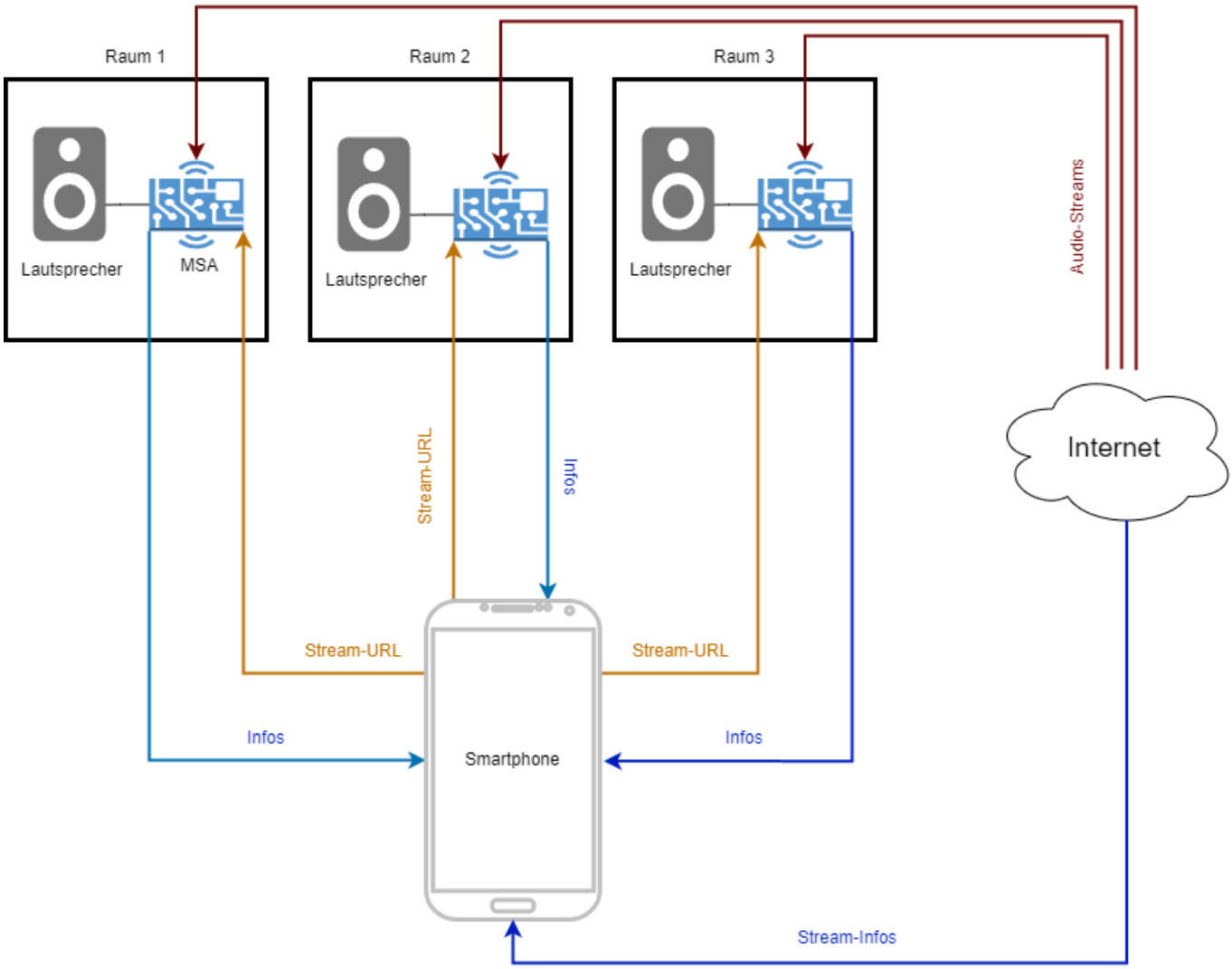

Abbildung 1: Skizze Gerätekette

Hier ist eine vereinfachte Darstellung der Gerätekette und wie sich der Multi-Room Audio Adapter in diese eingliedern lässt.
Am unteren Bildrand ist eine Wolke mit Information dargestellt. Sie symbolisiert das Internet mit dessen Inhalt. Darüber ein Smartphone, welches mit dem Internet kommuniziert und sich so Informationen wie Stream-URLs holt und sie an den Adapter weitergibt. Der Adapter selbst (hier als blaue Platine dargestellt) holt sich den Stream selbst (also praktisch ein Audiosignal) und stellt ihn als Klinkenausgang zur Verfügung. Man kann diesen dann beispielsweise mit einem Lautsprecher verbinden und Musik hören.

2.1 Einleitung Hardware

Das Ziel des Hardware-Teils für folgende Diplomarbeit war, einen sinnvollen internen Aufbau des Geräts zu erzielen, die am besten geeigneten Hardware-Komponenten zu finden, das System bzw. die einzelnen Komponenten zusammenzusetzen und zu testen. Dieser Teil der Diplomarbeit wird von Nico Lang übernommen. Zudem beschäftigt sich dieser Teil mit dem Gehäuse des Geräts und bestimmt die technischen Anforderungen (Schnittstellen), die der Adapter letztendlich haben soll. Bei der Planung soll zudem darauf geachtet werden, möglichst viele Kosten einzusparen, ohne dabei die Faktoren der Sicherheit und Qualität zu vergessen.

2.2 Einleitung Software

Das Ziel des Software-Teils für folgende Diplomarbeit war, einerseits die Software des Adapters, andererseits die Software der Smartphoneapp zu entwickeln. Dieser Teil der Diplomarbeit wurde von Philipp Immler übernommen. Die Software des Adapters wurde mit der Programmiersprache C++ codiert. Die Software der Smartphoneapp wurde mit JavaScript codiert. Bei der Programmierung wurden zahlreiche Bibliotheken und Frameworks verwendet. Dies hat den Vorteil, dass diese schon vorgefertigten Code für bestimmte Probleme bieten und deshalb nicht alles von Grund auf neu entwickeln werden muss.

3 Planung

3.1 Festlegung Funktionsweise

Beim Festlegen einer grundlegenden Funktionsweise des Adapters stellen sich vor allem folgende Fragen:

3.1.1 Was soll das System können?

Das Hauptziel ist, dass das System in verschiedenen, voneinander getrennten Räumlichkeiten bestimmte Audiosignale auf einen Line-Ausgang abspielen kann.

Line-Ausgang
Ein Line-Ausgang (Line-Out) ist eine Ausgangs-Schnittstelle für analoge Audiosignale, deren Ausgangsspannung immer grob dem Line-Pegel entspricht. Dieser „Line-Pegel beträgt etwa 0,5 Volt bis 1 Volt“.
Diese geringe Spannung reicht jedoch nicht, um das Audio-Signal direkt an einen Lautsprecher auszugeben. Es muss zuerst noch durch einen Verstärker verstärkt werden. Lautsprecher gibt es mit eingebautem Verstärker (aktive Lautsprecher), es gibt sie jedoch auch ohne internen Verstärker (passive Lautsprecher).

(vgl. `https://www.monacor.de/magazin/audio-pegel`)

Aktive vs. Passive Lautsprecher

Ein klassisches Beispiel für passive Lautsprecher sind herkömmliche Hi-Fi Stereoanlagen. Diese bestehen meistens aus einem oder mehreren Playern, einem Verstärker und zwei oder mehreren (Surround Sound also Raumklang) passiven Lautsprechern. Der Player liest das Signal (beispielsweise einer CD oder einer Schallplatte) und gibt den Line-Pegel über ein Kabel (im Hi-Fi Bereich meist Cinch oder 3,5mm Klinke) an den Verstärker weiter. Dieser Line-Pegel kann aber auch direkt aus einem TV-Gerät oder wie in unserem Fall aus einem Multi-Room Audio Adapter kommen. Der Verstärker verstärkt das Audiosignal nun von der geringen Spannung des Line-Pegels auf die für die Lautsprecher passende Spannung. Mit dem Lautstärkeregler am Verstärker kann man sich die Spannung (also Lautstärke) letztendlich noch auf persönliche Präferenzen anpassen.
Ein Beispiel für aktive Lautsprecher sind Bluetooth-Lautsprecher, deren Hauptziel es ist, möglichst kompakt und leicht transportierbar zu sein. Solche Bluetooth-Lautsprecher enthalten im Normalfall einen Akku, um auch unterwegs, ohne aktive Stromquelle, Musik hören zu können. Somit enthält das Gehäuse den Verstärker, die Lautsprecher, den Akku und sonstige Elektronik wie unter anderem ein Bluetooth-Modul. Hier fungiert meist ein herkömmliches Smartphone als Signalgeber, ob über Bluetooth oder 3,5mm Klinke bleibt dem/der Benutzer/-in überlassen.
Lenovo beschreibt Line-Ausgänge zum Beispiel folgendermaßen: „Der Line-Ausgang unterscheidet sich von anderen Audioausgängen wie z. B. Kopfhörerbuchsen, da er ein Signal mit festem Pegel liefert, das nicht von der Lautstärkeregelung Ihres Geräts beeinflusst wird. Er ist für den Anschluss an Geräte gedacht, die das Audiosignal verstärken oder weiterverarbeiten können.“
Man kann also daraus schließen, dass man das Line-Out Signal des Multi-Room Audio Adapter vor dem Lautsprecher noch verstärken muss. Wie genau, ist dem/der Endverbraucher/-in überlassen.

(vgl. `https://www.lenovo.com/at/de/glossary/line-out`)

Audioqualität

Zudem ist es wichtig, dass das System den Ton zuverlässig und möglichst flüssig überträgt und ausgibt.
„Das menschliche Ohr ist theoretisch in der Lage, Frequenzen von 20 Hz bis 20 kHz zu hören. Die Obergrenze von 20 kHz nimmt mit dem Alter ab. “
Es gibt drei wichtige Grundgrößen wenn es um Audioqualität geht: Samplerate, Auflösung und Bittiefe. Jan Baumann hat diese auf seiner Website sehr gut erklärt:

„Die **Samplerate** (Einheit Hz = Hertz) gibt an, wie oft in einer Sekunde der Audio-Pegel erfasst und gespeichert wird. Eine Angabe von 44.100 Hz (44,1 kHz) bedeutet, dass 44.100 Werte für eine Sekunde Musik gespeichert werden. Übliche Sample-Raten sind 44,1 kHz (Musik CD), 48 kHz (Film) und 96 kHz (Tonstudio). “

„Die **Auflösung** (Einheit Bit) gibt an, wie viel Speicher für so einen Sample-Wert genutzt wird. Zum Beispiel erlauben 16 Bit (2-hoch-16) eine Skala von 65.536 Werten für jeden einzelnen Sample-Wert. Wenn wir viel Speicher für einen Wert haben, können wir das Signal also mit mehr Genauigkeit

verarbeiten. Übliche Werte sind 16 Bit (Musik CD) oder 24 Bit bzw. 32 Bit im Studio.“

„Die **Bitrate** bzw. Datenrate (kBit/s) wird oft mit der Auflösung verwechselt. Sie steht für die “Bandbreite” der Audiodatei, also welche Datenmenge in einer Sekunde verarbeitet wird. Für unkomprimierte Formate wie WAV und AIFF berechnet man die Bitrate ganz einfach, indem man die drei Werte von oben multipliziert: “
Mehr zu Audioqualität aber im Teil Äuswahl interne Hardware/Digital-/Analogwandler"

(vgl. `https://www.axis.com/dam/public/ad/35/af/einf%C3%BChrung-in-das-thema-audio:-akustik-,-lautsprecher-und-audio-terminologie-de-DE-191125.pdf`)
(vgl. `https://www.baumannmusic.com/de/2012/sampleratehz-und-khz-aufloesung-bit-und-bitrate-kbits/`

Audio-Quellen
Grundsätzlich kann jeder beliebige Audio-Stream aus dem Internet verwendet werden. Das können beispielsweise Radiosender sein. Ein Beispiel für einen solchen Audio-Stream wäre der, des österreichischen Radio-Senders „OE3“:
`https://orf-live.ors-shoutcast.at/oe3-q2a`

Benutzerfreundlichkeit
Es wird zudem viel Wert auf Benutzerfreundlichkeit gelegt. Das bedeutet, dass sich der Adapter zum einen leicht einrichten lässt, aber auch, dass er sich (mithilfe der Smartphone-App) einfach bedienen lässt.

(vgl. (Fachbuch) `https://books.google.de/books?id=UI2INugaKwIC&pg=PA219#v=onepage`)

3.1.2 Was muss es nicht können?

Dieser Multi-Room Audio Adapter ist als Hi-Fi Produkt für den klassischen Durchschnittsbürger und/oder Musik-Liebhaber gedacht. Aufgrund dessen wurde die Bedienung sehr einfach und benutzerfreundlich, jedoch eindeutig nicht so präzise oder vielfältig einstellbar gestaltet wie es bei professionellem Audio-Equipment der Fall ist. Während der Laie das Produkt einfach anstecken und benutzen möchte, hätte ein Audio-Nerd beispielsweise gerne noch einen eingebauten Acht-Band Equalizer und vieles mehr. Das war jedoch nicht das Ziel dieser Diplomarbeit. Es ging eher darum, die Hauptfunktion, also Ton kabellos in Räume zu übertragen, und Einstellungsmöglichkeiten per App ohne großes Kopfzerbrechen zu ermöglichen.
Wie schon oben erwähnt liefert der Multi Room Sound-Adapter nur einen Audioausgang mit Line-Pegel und es ist ihm nicht möglich, dieses zu verstärken. Man kann also keine Kopfhörer mit einer Impedanz über 80 Ohm direkt an das Gerät anschließen (man kann theoretisch schon, aber der Ton wird sehr leise sein). Das Audiosignal muss also zuerst mit einem Verstärker verstärkt werden.

3.1.3 Wie könnte man es erweitern?

Unsere Variante des Multi-Room Audio Adapter zeichnet sich vor allem durch die beliebige Erweiterbarkeit aus. In der Theorie soll es ein einzelnes Modell, also den Adapter selbst geben. Mit jedem weiteren Adapter kann dementsprechend ein weiterer Lautsprecher oder ein Raum zugefügt werden. In Zukunft wäre es auch vorstellbar, dass man aus mehreren Adaptern Gruppen bilden kann, in denen die Adapter synchronisiert sind und somit der gleiche Audio-Stream auf mehreren Adaptern synchron läuft. Dies ist aber technisch sehr aufwendig, da die Latenz von WiFi ziemlich hoch ist.

SKIZZE: Grundriss mit Lautsprechern in allen Räumen und Gruppenbildung

3.2 Auswahl Hardwarekomponenten

Zur Auswahl der Hardwarekomponenten des Adapters wurde zu aller erst die externe Ausstattung des Adapters überlegt. Das bedeutet praktisch alles, mit dem ein Endverbraucher letztendlich zu tun hat. Dann kann der interne Teil, also die Technik dahinter, individuell auf die Anforderungen des externen Teils designet und entwickelt werden.

3.2.1 Auswahl externe Hardware

Der Adapter sollte ein möglichst kompakt konstruiertes und stabiles Gehäuse bekommen. An diesem ist ein einfacher Taster zur Interaktion angebracht. Mit dem Taster sind einige Funktionen des Adapters ermöglicht. Beispielsweise per Klick, Doppelklick oder kurzem Halten. Da sich die Aufgaben des Tasters selbst gering halten (Verbindungsvorgang, Ein- und Ausschalten, ...) wurde nur ein einziger Taster verwendet, um die Komplexität des Gesamtsystems zu senken. Die weitaus komplizierteren Funktionen wurden alle samt in der Smartphone-Applikation ermöglicht. Zusätzlich wurde eine RGB-Leuchtdiode zur Statusanzeige verbaut, um beispielsweise den aktuellen Verbindungsstatus zum Mobilgerät und zum Internet anzuzeigen.

Gehäuse
Das Gehäuse soll alle Komponenten auf möglichst kleinem Raum zusammenhalten, schützen und kühlen. Da sich Komponenten und möglicherweise auch das Design selbst laufend änderten, wurde dieses erst gegen Ende des Projekts finalisiert. Mehr dazu im Teil "Design Adaptergehäuse".

Taster
Für den Taster wurde ein herkömmlicher Tactile-Button mit verlängertem Taster in das Gehäuse geplant. Auf ihm klebt eine Art Aufsatz, auf den der/die Endverbraucher/-in letztendlich drückt, um ein gleichbleibendes optisches Design des Gehäuses zu ermöglichen. Mehr dazu im Teil "Design Adaptergehäuse"

LED (Light-Emitting Diode)
Als Statusanzeige wurde in diesem Fall eine herkömmliche RGB-LED verwendet. Mit dieser ist es theoretisch möglich, alle Farben des RGB-Spektrums (16,7 Mio) mit einer Komponente darzustellen.
Man muss sich jedoch im klaren darüber sein, für wie viel Spannung die benutzte LED gebaut ist. Dann kann man mit passenden Widerständen arbeiten, um die LED nicht aufgrund zu hoher Spannung zu beschädigen.

(vgl. `https://www.elektronik-kompendium.de/sites/praxis/bauteil_rgb-led.htm`)

3.2.2 Auswahl interne Hardware

Mikrocontroller (ESP32)
Als Herz des Systems wird ein ESP32-WROOM-32 Mikrocontroller mit angebauter Platine (DOIT ESP32 DEVKIT V1) verwendet. Der ESP32 ist ein weit verbreiteter Mikrocontroller. Das Konzept des Controllers sind GPIOs (general purpose input/output) also Pins die man mit Code mit bestimmten Funktionen beschreiben kann. Man kann die Arduino IDE mit C++ als Programmiersprache zum programmieren verwenden. Zudem verfügt er schon Onboard über einen Hybrid WIFI- und Bluetooth-Chip, wodurch externe Module vermieden, und somit Platz eingespart werden kann. Espressif selbst beschreibt den ESP32 als optimal für IoT-Anwendungen; auch wegen der hohen Energieeffizienz.
Hier ein paar Grundfakten des ESP32-WROOM-32 (der Controller selbst):
Der von uns benutzte ESP32 hat einen Dual-Core 32-bit Prozessor (2x Tensilica-LX6-Kernen). Der Mikrocontroller hat WLAN (802.11b/g/n) und kann ein eigenes WLAN Netzwerk (Access Point) erstellen (kleiner WebServer). Er unterstützt Bluetooth 4.0 (BLE/Bluetooth Smart) und Bluetooth Classic. Dies ist bei vielen SmartHome und IoT-Anwendungen nützlich. Er hat einen geringen Stromverbrauch von 50-70 mA (kleine Programme ohne WiFi). Der Deep-Sleep Modus macht einen Stromverbrauch von unter 0,1mA möglich. Die Anschaffungskosten fallen mit unter 10 Euro im Vergleich zu anderen Mikrocontrollern auch sehr niedrig aus.
Wie schon erwähnt ist der ESP32 mit der (uns vertrauten) Arduino IDE programmierbar und kann auch in industriellen Umgebungen (-40°C bis +125°C) betrieben werden.
Für den Multi Room Sound-Adapter kommt ein ESP32 Entwicklungsboard (DOIT ESP32 DEVKIT V1) zum Einsatz.
Hier eine Beschreibung von Dev-Kits auf digitalewelt.at: „Um alle Funktionen des ESP32 Moduls einfach nutzen zu können, gibt es sogenannte Entwicklungsboards. Diese sind nicht nur mit zusätzlichen Schaltungen für die Spannungsversorgung ausgestattet, sondern bieten uns auch die gängigen Anschlussmöglichkeiten für unsere externen Komponenten.“
Erwähnenswert ist auch, dass der ESP32 an manchen GPIOs intere Pullup- /Pulldown-Widerstände hat. „Wenn man einen Pin, an dem z.B. ein Sensor hängt, mit pinMode(5, INPUT) konfiguriert, hat der Pin keinen definierten Pegel. Aus diesem Grund kann er solange zwischen HIGH und LOW

schwanken, bis ihm ein Zustand zugeschrieben wird. “ ... „Dafür hat der ESP32 interne Pullup- / Pulldown-Widerstände, die zugeschaltet werden können. Diese können mit INPUT_PULLUP als Argument im Befehl pinMode() eingeschaltet werden. Bei INPUT_PULLUP wird der Pin als Eingang deklariert und wenn er nicht beschaltet ist wird dieser auf HIGH gesetzt. Bei INPUT_PULLDOWN hingegen wird der Eingang auf LOW gesetzt.“ Pullup- /Pulldown-Widerstände sind essenziell für einen sauber konfigurierten Tactile-Button.

(vgl. `https://www.espressif.com/en/products/socs/esp32`)
(vgl. `https://digitalewelt.at/esp32-grundlagen/`)
(vgl. `https://www.xplore-dna.net/mod/page/view.php?id=2741&forceview=1`

Digital-/Analogwandler (Audio-Modul)
Dieses Modul ist ausschlaggebend für eine gute Audioqualität. Das digitale Audiosignal vom ESP32 (Radiostream) muss nämlich für die Line-Out Buchse auf analog konvertiert werden. Dafür wird ein Modul mit dem PCM5102A („2VRMS DirectPath™, 112 dB Audio Stereo DAC mit 32-bit, 384 kHz-PCM-Schnittstelle“) DAC (Digital-Analog-Converter) verwendet. Auf diesem Modul befinden sich alle Komponenten sowie die Klinkenbuchse die für die Ausgabe des Audiosignals nötig sind.

(vgl. `https://www.ti.com/product/de-de/PCM5102A`)

Akku (Li-Ion)
Das Endprodukt soll mithilfe eines Akkus auch ohne Strom auskommen, dafür wird ein Lithium-Ionen-Akku mit 3,7 Volt verwendet. Akkus dieser Art zeichnen sich durch ihre hohe Energiedichte und, unter guten Umständen, hohe Lebensdauer aus. Man verwendet Li-Ionen-Akkus meist für (tragbare) Geräte in denen andere Akkus zu schwer oder zu groß wären. (vgl. `https://www.chemie.de/lexikon/Lithium-Ionen-Akkumulator.html`)

Natürlich haben Li-Ionen-Akkus auch gewisse Nachteile und bergen wie jeder andere Akku Gefahren. Beispiele dafür sind elektrische Überlastung, mechanische Beschädigung und thermische Überlastung:

Eine elektrische Überlast kann etliche Gründe haben, darunter:

- Verwendung eines falschen Ladegerätes
- Tiefenentladung
- Falsche Lagerbedingungen (z.B.: zu hohe Temperaturen)
  Zitat: „Hier kommt es zur Zersetzung der Elektrolytflüssigkeit und infolgedessen zur Bildung leicht brennbarer Gase. Wird anschließend versucht, die tiefentladenen Lithium-Ionen-Zellen wieder aufzuladen, kann die zugeführte Energie durch das Fehlen von Elektrolytflüssigkeit nicht mehr korrekt umgesetzt werden. Es kann zum Kurzschluss beziehungsweise zum Brand kommen.“ (vgl. `https://www.denios.de/services/denios-magazin/gefahren-im-umgang-mit-lithium-ionen-akkus`)

Eine mechanische Beschädigung jeglicher Art kann zu Kurzschlüssen im inneren der Zelle führen. Da unser Gerät nicht dafür gemacht ist, ständig in Bewegung zu sein, spielt dies keine zu große Rolle, es muss jedoch trotzdem ausreichend Schutz gegeben sein, was durch das Gehäuse sichergestellt wird.

Wie oben schon kurz erwähnt, muss großer Wert auf die richtige Lagerung/Kühlung des Akkus gelegt werden. Wird dieser zu heiß (etwa durch den Mikrocontroller oder sonstige Bauteile) oder durch äußere Einflüsse beschädigt kann es zum Brand kommen.

Man kann daraus schließen, dass jeder kleinste Fehler beispielsweise zu einem Brand oder sogar einer Explosion des Akkus führen kann. Es ist daher wichtig, den Akku mit absoluter Vorsicht zu handhaben. Ausreichend Tests (Betriebstemperatur, etc.), richtige Konfiguration des Ladereglers und die Auswahl des Akkus sind ausschlaggebend für die Sicherheit des Endverbrauchers und dessen Umfeld.

(vgl. `https://www.denios.de/services/denios-magazin/gefahren-im-umgang-mit-lithium-ionen-akkus`)

Laderegler

Für einen optimalen Ladeprozess und Schutz des Akkus wird ein spezieller Laderegler verwendet. Dieser regelt somit den Ladevorgang des Akkus und hört auf zu laden, sobald dieser voll ist. Der Aufbau dieses Ladereglers ist nicht kompliziert, es gibt jeweils einen Plus- und Minuspol für den Eingang (USB-C Buchse), den Akku (im englischen auch als Battery bezeichnet) und den Ausgang.

3.3 Auswahl Technologien

3.3.1 Protokolle

In diesem Kapitel geht es um die Recherche und Auswahl von Protokollen, die für den Austausch von Daten verwendet wurden.

Protokolle werden in der Informatik verwendet um eine standartisierte Kommunikation zwischen zwei oder mehreren Computern zu ermöglichen. Diese standartisierte Kommunikation wird durch das Festlegen von Standards und Normen erreicht. (vgl. *URLPI16* 2025)

HTTP

Das Hyper Text Transfer Protocol ist ein weitverbreitetes Protokoll im Web und wird größtenteils für die Kommunikation zwischen Browsern und Webservern eingesetzt. Dabei basiert das Protokoll auf sogenannten „Requests“ (auf Deutsch: Anfragen). Es gibt zahlreiche Anwendungen für HTTP. Wir nutzen es einerseits für die Kommunikation zwischen Smartphoneapp (Client) und Microcontroller (Server) mittels REST-API und andererseits zum Streamen der Audio-Files aus dem Internet, mittels HLS (siehe HLS). (vgl. *URLPI01* 2020)

HLS

Hier kommt der Text fürs HLS-Protokoll hin

mDNS

Das Multicast Domain Name Service - Protokoll hilft in kleineren, lokalen Netzwerken zur Namensauflösung. Im Web dominiert das bekanntere DNS (Domain Name Service) - Protokoll. Dieses dient dazu, die IP-Addresse hinter einer Domain zu ermitteln. Dabei gibt es einen DNS-Server, welcher eine Art Liste führt, in der aufgeführt ist, welcher Domainname zu welcher IP-Adresse gehört. Wenn man also im Browser eine URL eingibt, stellt der Client dem DNS-Server eine Anfrage. Dieser sendet anschließend die zugehörige IP-Adresse zurück, mit der sich der Client dann verbindet. Dieser Ansatz ist allerdings in kleineren, lokalen Netzwerken eher unpraktisch, da ein eigener Server benötigt wird. Als Alternative dazu dient das sogenannte mDNS -Protokoll, welches mit Multicast arbeitet und daher keinen DNS-Server benötigt. Beim Multicast sendet ein Teilnehmer eines Netzwerkes eine Nachricht an alle Teilnehmer im gleichen Netzwerk. Bei mDNS fragt also das Gerät, welches sich mit einem anderen Gerät verbinden will, jedes Gerät im Netzwerk ob der entsprechende Domain-Name zu diesem gehört. Wenn dies der Fall ist, meldet sich das zugehörige Gerät und somit lässt sich eine Verbindung mit diesem herstellen. (vgl. *Multicast DNS* 2022) In der Diplomarbeit wird mDNS verwendet, um eine Verbindung mit den MSA aufzubauen. Der Hostname eines MSA setzt sich dabei aus MSA gefolgt von den letzten 3 Byte der Mac-Adresse, getrennt durch einen Unterstrich, zusammen. Dies macht deshalb Sinn, weil somit jeder MSA einen eindeutigen Namen bekommt, da die letzten 3 Byte eines jeden Netzwerkgerätes weltweit eindeutig sind. Da in den meisten Netzwerken ein DHCP (Dynamic Host Configuration Protocol) - Server vorhanden ist, der die IP-Adressen automatisch vergibt, kann man die Adapter nicht anhand deren IP-Adresse eindeutig identifizieren, da sich diese ständig ändert. (vgl. *URLPI20* 2025) Mit mDNS ist dies allerdings möglich. Dabei ist jeder Adapter unter seinem Namen gefolgt von .local auffindbar. Ein Adapter mit dem Namen MSA_3C4D5E ist also im lokalen Netzwerk unter MSA_3C4D5E.local aufrufbar.

I2S

Das Inter IC Sound Protocol wird verwendet, um Stereo-Audio-Daten zwischen ICs (Integrated Circuits) auszutauschen. Es ist dabei zu unterscheiden von dem I2C (Inter-integrated circuit) - Protokoll, welches rein für den Datenaustausch zwischen ICs verwendet wird und somit nicht für Audiodaten spezialisiert ist. Für die Datenübertragung benötigt das I2S-Protokoll folgende Leitungen:

- Taktleitung
- Wortauswahl
- mindestens eine Datenleitung

Die Datenübertragung erfolgt seriell und synchron. Das heißt, dass die Daten nacheinander durch eine Leitung (die Datenleitung), in einem bestimmten Takt, welcher von der Taktleitung vorgegeben

wird, übertragen werden. Wenn Audiodaten im Stereo-Format (linker und rechter Kanal) übetragen werden, wählt die Wortauswahl-Leitung jeweils den Kanal aus, welcher übertragen werden soll. (vgl. *URLPI12* 2024)
In der Diplomarbeit wurde das I2S Protokoll verwendet, um die digitalen Stereo-Audio-Daten vom Microcontroller an den Digital-Analag-Wandler zu übertragen. Dabei werden die Buffer, die der Microcontroller vom Audio-Stream erhält, mittels I2S an den Digital-Analog-Wandler gesendet, welcher die digitalen Daten in analoge PCM (Pulse Code Modulation) Daten umwandelt, so dass diese dann anschließend auf der Lautsprecherbox ausgegeben werden können.

3.4 Auswahl Softwaretools

3.4.1 Einleitung

In diesem Kapitel geht es um die Recherche und Auswahl von geeigneten Softwaretools, welche für die App-Entwicklung, als auch für die Entwicklung der Software des Microcontrollers verwendet wurden. Im folgenden werden zusätzlich noch die Tools beschrieben, welche für das Schreiben der Diplomarbeit verwendet wurden:

LaTeX
"LaTeX ist ein hochwertiges Schriftsatzsystem, das Funktionen für die Erstellung technischer und wissenschaftlicher Dokumentationen enthält. LaTeX ist der De-facto-Standard für die Kommunikation und Veröffentlichung von wissenschaftlichen Dokumenten."(Übersetzung des englischen Originals von: https://www.latex-project.org/)
Wir haben uns für das Schreiben unserer Diplomarbeit in LaTeX entschieden, weil es sich sehr gut für das Schreiben von wissenschaftlichen Arbeiten, vor allem über technische Themen, eignet und wir somit schon damit vertraut sind, wenn wir es in der Zukunft, zum Beispiel im Studium benutzen müssen.

draw.io "draw.io ist eine kostenlose Online-Diagrammsoftware zur Erstellung von Flussdiagrammen, Prozessdiagrammen, Organigrammen, UML, ER und Netzwerkdiagrammen."(Übersetzung des englischen Originals von: https://app.diagrams.net/)
Wir haben alle Diagramme, welche in unserer Diplomarbeit zu sehen sind, in draw.io erstellt, weil es einfach zu handhaben ist und es eine große Auwahl an Diagrammtypen und Formen bereitstellt.

Visual Studio Code
"Visual Studio Code ist ein leichtgewichtiger, aber leistungsstarker Quellcode-Editor, der auf Ihrem Desktop läuft und für Windows, macOS und Linux verfügbar ist. Er bietet integrierte Unterstützung für JavaScript, TypeScript und Node.js und verfügt über ein umfangreiches Ökosystem von Erweiterungen für andere Sprachen und Laufzeiten (wie C++, C#, Java, Python, PHP, Go, .NET)."(Übersetzung des englischen Originals von: https://code.visualstudio.com/docs)
Wir haben Visual Studio Code als IDE für unsere Diplomarbeit gewählt, weil durch die unzähligen Erweiterungen viele verschieden Programmiersprachen und Bibliotheken unterstützt werden und

wir somit den gesamten Code, sowohl für den Microcontroller als auch für die Smartphoneapp, darin schreiben konnten.

GitHub
Git im Allgemeinen ist das meist verbreitete Versionskontrollsystem. Ein Versionskontrollsystem wird verwendet, um den Versionsverlauf einer Software zu erfassen. Dies hat den Vorteil, dass man bei auftretenden Fehlern jederzeit wieder auf eine ältere Version der Software umsteigen kann. Ein weiterer Vorteil beim Einsatz eines Versionskontrollsystems ist, dass mehrere Entwickler gleichzeitig an derselben Software arbeiten können. Git gehört zu den DVCS (Distributed Version Control Systems), also verteilten Versionskontrollsystemen. Das heißt, dass der Versionsverlauf der Software nicht zentral auf einem Server gespeichert ist, sondern auf den Rechnern der Benutzer, in einem sogenannten Repository. Dies bietet höhere Sicherheit, Performance und Flexibilität. (vgl. Atlassian 2025)
GitHub ist eine Software, die Git verwendet. Dabei ist Github cloud-basiert, das heißt es ist online zugänglich und man muss nicht eine eigene Software dafür auf dem Rechner installieren. Außerdem hat Github eine sehr benutzerfreundliche Oberfläche, was es auch für Einsteiger leichter verständlich macht. (vgl. *Was ist GitHub?* 2025) Wir verwenden GitHub für die Verwaltung unseres Codes und unserer Dokumente. Der Vorteil dabei ist, dass jedes Projektmitglied auf seinem lokalen PC an den Dokumenten arbeiten kann und die Änderungen dann per GitHub synchronisiert werden.

DeepL
Bei DeepL handelt es sich um einen Übersetzer, der KI einsetzt und somit einer der genausten Übersetzer weltweit ist. Wir haben DeepL in unserer Diplomarbeit verwendet, um englische Texte ins Deutsche zu übersetzen. Wir haben uns aufgrund der hohen Genauigkeit und weil dieser kostenlos ist für DeepL entschieden.

3.4.2 Bibliotheken Microcontroller

Im folgenden werden die Bibliotheken, welche für die Protgrammierung der Microcontroller-Software verwendet wurden, aufgezählt und beschrieben:

Arduino Core
(https://github.com/espressif/arduino-esp32)
Die Arduino-Core-Bibliothek wurde verwendet, um den ESP32 ähnlich wie ein Arduino-Board programmieren zu können. Es erleichtert dabei die Programmierung enorm. Vorallem dann, wenn man schon Vorerfahrung mit der Programmierung von Arduino-Boards hat. Ein weiterer Vorteil ist, dass diese Bibliothek bzw. dieses Framework bereits weitere nüztliche Bibliotheken beinhaltet, welche für die Programmierung benötigt werden.

WiFi
(https://github.com/espressif/arduino-esp32/tree/master/libraries/WiFi)

Die WiFi-Bibliothek ist eine offizielle Bibliothek von Arudino, welche in der Arduino-Core-Bibliothek inkludiert ist. Sie wird verwendet, um die Funktionen der eingebauten WiFi-Antenne des ESP32 zu nutzen. Der ESP32 kann dabei entweder als Access Point oder als Client fungieren. Wenn er als Access Point fungiert, stellt er ein eigenes WiFi-Netzwerk bereit, mit dem sich andere Geräte verbinden können und der ESP32 somit einen Host darstellt. Als Client kann er sich mit anderen WiFi-Netzwerken bzw. Access Points verbinden. In unserem Projekt fungiert der ESP32 sowohl als Access Point, als auch als Client. (vgl. *URLPI24* 2025)

ArduinoJson

(https://github.com/bblanchon/ArduinoJson)

Die ArduinoJson-Bibliothek wird verwendet, um Daten in das JSON-Format zu kodieren. JSON (Java Script Object Notation) ist ein Datenformat, welches für den einheitlichen Datenaustausch zwischen Server und Client verwendet wird. Dabei verwendet JSON sogenannte Schlüssel-Wert-Paare. Das heißt, ein Wert hat immer einen eindeutigen Schlüssel. In unserem Projekt wird die ArduinoJson-Bibliothek für den einheitlichen Datenaustausch zwischen Webserver (ESP32) und Client (Smartphone) verwendet. (vgl. *URLPI11* 2024)

WebServer

Die WebServer-Bibliothek wird verwendet, um einen Webserver auf dem ESP32 bereitzustellen. Dieser ist wichtig für die Funktion als REST-API und somit für den Datenaustausch zwischen ESP32 und Smartphoneapp. Der Webserver kann Anfragen von Clients entgegennehmen und diesen Antworten zurücksenden. Dabei gibt es vordefinierte Routen, welche aufrufbar sind. Diese Routen werden im Kapitel ... beschrieben.

ESP8266Audio

(https://github.com/earlephilhower/ESP8266Audio)

Die ESP8266Audio-Bibliothek wurde ursprünglich für das Vorgängermodell des ESP32, den ESP8266, programmiert. Es wurde allerdings mittlerweile von den Entwicklern angepasst, so dass man es auch für den ESP32 verwenden kann. Die Bibliothek stellt Funktionen zum Empfangen, Dekodieren und Ausgeben von Audio-Daten bereit. Wir haben diese Bibliothek verwendet, um MP3-Streams vom Internet zu empfangen, diese zu dekodieren und die dekodierten PCM-Signale über I2S an den DAC zu senden. Der ESP32 verfügt zwar bereits standartmäßig über Funktionen, mit deren Hilfe man Audiodaten mittels I2S übertragen kann, allerdings sind diese sehr komplex in der Verwendung und Konfiguration. Da die ESP8266Audio-Bibliothek bereits die perfekte Lösung für unsere Anforderungen bietet, wurde diese in unserer Diplomarbeit verwendet.

Preferences

(https://github.com/espressif/arduino-esp32/tree/master/libraries/Preferences) Die Preferences-Bibliothek wird verwendet, um Schreib- und Lesezugriffe auf den eingebauten EEPROM (Electrical Eresable Programmable Read Only Memory) durchzuführen. Dabei werden die im EEPROM gespeicherten

Werte mittels eindeutiger Schlüssel identifiziert.

3.4.3 Softwaretools Smartphoneapp

Im foldenden Teil werden die Softwaretools, welche für die Entwicklung der Smartphoneapp verwendet wurden, genauer beschrieben.

Node.js
Bei Node.js handelt es sich um eine JavaScript-Laufzeitumgebung, welche es ermöglicht JavaScript-Code lokal auf dem Rechner auszuführen. Normalerweise ist JavaScript eine Programmiersprache, welche für die Frontend-Entwicklung eingesetzt wird und nur im Browser läuft. Mit der Verwendung von Node.js ist es allerdings möglich, JavaScript auch im Backend, also lokal auf dem Rechner zu verwenden. Node.js stellt außerdem einen eigenen Paket-Manager namens npm (Node Package Manager) bereit, der es ermöglicht zahlreiche Open-Source-Pakete, für unterschiedlichste Anwendungen zu installieren. Somit können mithilfe von Node.js die Vorteile, welche JavaScript bietet, zur Entwicklung zahlreicher Apps verwendet werden. (vgl. *URLPI25* 2021) Wir haben Node.js für unsere Diplomarbeit verwendet, weil dieses für React Native und somit auch für Expo benötigt wird.

React Native
React Native ist ein Framework, welches die plattformübergreifende Entwicklung von Apps ermöglicht.Das heißt, man schreibt einen Code in JavaScript und kann aus diesem dann eine IOS-, Anrdoid- und fürs Web-App bauen. Um eine React Native App auszuführen, wird die Node.js-Laufzeitumgebung benötigt, welche im vorherigen Teil beschrieben wurde. React Native wurde erstmals 2015 von Meta (damals noch Facebook) als Open-Source-Projekt veröffentlicht. Seither wird es weiterhin von Meta gewartet bzw. weiterentwickelt und hat eine riesige Community, welche ständig neue Bibliotheken für das Framewok veröffentlicht. React Native basiert auf React, welches man bereits aus der Webentwicklung kennt. Der Vorteil von React im Gegensatz zur normalen Webentwicklung ist, dass man wiederverwendbare Komponenten bauen kann. Dies ist auch mit React Native möglich. In React Native stehen dabei einige Standardkomponenten zur Verfügung, welche dann jeweils in native Komponenten, passend für das jeweilige Betriebssystem, gerendert werden. Wir haben React Native für unsere Smartphoneapp verwendet, weil es sehr aufwendig gewesen wäre, für jede Plattform einen eigenen Code zu schreiben und dies außerdem sehr viel Wissen in unterschiedlichen Bereichen vorausgesetzt hätte. Außerdem konnten wir von der bereits gesammelten Erfahrung in der Entwicklung von JavaScript und React profitieren, was uns den Einstieg erleichterte. (vgl. *What Is React Native?* 2024)

Expo
Um das Entwickeln der Smartphoneapp noch einfacher bzw. effizienter zu gestalten, wurde das Expo Framework verwendet. Dieses Framework basiert wiederum auf React Native und stellt zusätzlich hilfreiche Bibliotheken und eine Projektstruktur zur Verfügung. Dies hat den Vorteil, dass

der grundlegende Aufbau einer App bereits vorgegeben wird und man somit bereits eine solide Basis hat auf der man weiterentwickeln kann. Expo bietet außerdem das sogenannte „File-based routing “, welches die Entwicklung der App-Navigation sehr erleichtert. Ein weiterer Vorteil von Expo ist, dass für das Testen der App die ExpoGo-App verwendet werden kann. Diese ermöglicht es, die App bereits bevor diese vollständig gebaut ist, auf dem Smartphone auszuführen. Dies erleichtert die Entwicklung um ein Vielfaches. (vgl. *URLPI08* 2024)

4 Entwicklung

4.1 Entwicklung Software Adapter

In diesem Kapitel wird der Übergang der Planung in die Entwicklung der Software des Microcontrollers, welcher im MSA verbaut ist beschrieben. Zur Entwicklung der Software des Microcontrollers wurde die IDE Visual Studio Code in Verbindung mit dem Framework PlatformIO verwendet. PlatformIO ist ein Framework, welches für die Entwicklung von Embedded Systems verwendet wird und somit einige hilfreiche Funktionen in Verbindung mit dem ESP32 bietet. (vgl. *URLPI26* 2025) Um die Entwicklung so effizient wie möglich zu gestalten, wurden die Bibliotheken, welche bereits im Kapitel ... beschrieben wurden, verwendet.

4.1.1 Anforderungen

Im folgenden wird beschrieben, welche Anforderungen an die Software des Adapters gestellt wurden:

Access Point
Ein Access Point ist ein Gerät, welches ein WLAN aufbaut. Dabei können sich WLAN-Clients mit diesem verbinden und somit untereinander Daten austauschen. (vgl. *Was ist ein Access Point?* 2024) Der Microcontroller soll im Konfigurationsmodus als Access Point arbeiten. Dabei wird dem/der Benutzer/in ermöglicht, sich mittels Smartphoneapp mit dem Netzwerk des Microcontrollers zu verbinden und somit die Anmeldedaten des gewünschten Netzwerkes zu senden. Somit kann sich dann der Microcontroller in weiterer Folge mit dem gewünschten WLAN verbinden. Die Daten sollen dabei als JSON-Datei mittels HTTP von der Smartphoneapp an den Microcontroller gesendet werden.

WLAN Client
Ein WLAN-Client ist ein Gerät, welche mit einem Access Point verbunden ist und somit einen Teilnehmer in einem WLAN darstellt. Der WLAN-Client kann dabei Daten mit anderen Clients im Netzwerk austauschen. Wenn der MSA vollständig konfiguriert ist bzw. die Anmeldedaten des WLANs hat, soll sich dieser als WLAN-Client mit dem WLAN verbinden. Um den Datenaustausch zwischen Smartphone und MSA zu ermöglichen, müssen sich beide Geräte im gleichen Netzwerk befinden. Damit der MSA in weiterer Folge Audio-Streams von Internetradios empfangen kann, muss zusätlich eine Verbindung zum Internet bestehen.

REST-API

Der Microcontroller soll eine REST-API bereitstellen, mithilfe der es möglich ist, per HTTP Daten mit diesem auszutauschen. Eine REST-API ist eine API, welche den Designprinzipien der REST-Architektur folgt. API steht dabei für „Application Programming Interface “. Eine API bietet im Allgemeinen eine Möglichkeit für den einheitlichen Datenaustausch zwischen Programmen. Eine REST-API (Representational State Transfer Application Programming Interface) ist dabei eine API, welche die von der REST-Architektur vorgegebenen Regeln befolgt. Sie dient zur Kommunikation zwischen Server und Client. Der Client kann mittels HTTP, Anfragen an den Server senden. Der Server liefert dann eine Antwort, welche Daten enthalten kann. (vgl. *Was ist eine REST-API?* 2024) Auf diese Weise kommuniziert die Smartphoneapp mit dem Microcontroller. Der Client kann dabei sogenannte Routen aufrufen. Die aufgerufene Route bestimmt die Aktion, welche auf dem Server ausgeführt wird. Es gibt dabei verschiedene Typen von HTTP-Requests. Jeder Typ ist für eine bestimmte Aktion verantwortlich. Die verwendeten Typen in unserem Programm sind: GET, POST und PUT. Ein GET-Request wird verwendet, um Daten vom Server abzufragen. Der POST-Request wird verwendet um neue Daten auf dem Server anzulegen. Mit einem PUT-Request werden Daten auf dem Server aktualisiert. (vgl. *URLPI13* 2024) Im folgenden werden die verfügbaren Routen der REST-API des MSA genauer beschrieben:

Route	**Anfragen-Typ**	**Funktion**
/getInfo	GET	Client erhält die Daten: Name, MAC-Addresse, Akkustand, Lautstärke und Stream-URL vom Microcontroller im JSON-Format.
/getAvailableNetworks	GET	Client erhält eine Liste, welche SSID und RSSI (Stärke) von für den MSA verfügbaren Netzwerken enthält, im JSON Format. Diese werden in der Smartphoneapp benötigt, um auszuwählen mit welchem Netzwerk sich der MSA verbinden soll.
/setConfigData	POST	Client sendet SSID und Passwort des Netzwerks, mit welchem sich der MSA verbinden soll, im JSON Format an den MSA. Dieser speichert dann die erhaltenen Daten in den EEPROM und startet neu.
/setStreamUrl	PUT	Client sendet die URL des Internetradios, von welchem der MSA einen Audio-Stream erhalten soll, im Textformat an den Microcontroller. Dieser startet dann den Empfang dieses Streams und Verarbeitet diesen bzw. gibt diesen in weiterer Folge auf dem Line-Out aus.
/setVolume	PUT	Client sendet die Lautstärke (zwischen 0% und 100%), an den MSA. Dieser setzt dann den Gain des Audio-Signals auf einen Wert zwischen 0 und 1;

Abbildung 2: REST-API Routen

Audio-Streaming

Der Microcontroller sollte in der Lage sein, Audio-Streams aus dem Internet zu empfangen, diese zu dekodieren und die dekodierten Daten als Signale auf dem Line-Out auszugeben. Die meisten Audio-Streams werden mit dem HLS-Protokoll, welches auf dem HTTP-Protokoll basiert, übertragen. Dabei wird das Audiofile in mehrere kleine Stücke zerteilt, welche dann per HTTP an den Client gesendet werden. Dieser fügt diese Stücke dann wieder zu einem Audiofile zusammen. In weiterer Folge wird das MP3-File von dem externen Digital-Analog-Wandler zu einem PCM(„Pulse Code Modulation“) - Signal übertragen, welches dann auf der Lautsprecherbox ausgegeben werden kann.

4.1.2 Klassen

Um die Modularität und somit die Wiederverwendbarkeit und Wartbarkeit des Codes zu gewährleisten, wurde in der Entwicklung der Software für den Adapter auf objektorientierte Programmierung gesetzt. Dies ist möglich, da für die Programmierung die Programmiersprache C++ verwendet wurde. Für einen Großteil der Klassen wurde außerdem das Singleton-Designpattern verwendet. Dies gewährleistet, dass immer nur eine Instanz der Klasse vorhanden ist, was Sinn macht, wenn von einer Klasse nicht mehrere Instanzen verfügbar sein sollen. (vgl. *URLPI14* 2024) Im folgenden Teil werden die erstellten Klassen genauer beschrieben. Das folgende UML-Klassendiagramm veranschaulicht die Beziehung der verschiedenen Klassen zueinander:

Mode

Im ENUM Mode werden die verschiedenen Modi der Software definiert. Ein ENUM ist wie eine Art Datentyp, der Konstanten definiert, welche man verwenden darf. (vgl. *URLPI15* 2024) In unserem Fall definiert der ENUM die drei Modi NORMAL, CONFIG und ERROR. Diese geben den Status des MSA an. Der Modus NORMAL signalisiert, dass sich der MSA im normalen Zustand befindet, dieser also voll funktionsfähig ist. Im Modus CONFIG fungiert der MSA als Access Point. Somit können sich Clients mit diesem verbinden und die Anmeldedaten des WLANs senden. Der Modus ERROR wird gesetzt, wenn ein Fehler im Programm auftritt. Dies ist zum Beispiel der Fall, wenn die Verbindung mit dem WLAN unterbrochen wird. Der aktuelle Modus wird mithilfe der Status-LED signalisiert, welche als nächstes genauer beschrieben wird.

StatusLED

Mithilfe der Klasse StatusLED wird die RGB-LED, welche am Microcontroller angeschlossen ist, gesteuert. Mit ihr wird der aktuelle Modus des MSA dem/der Benutzer/in angezeigt. Die nachstehende Tabelle beschreibt welche Farbe für welchen Modus steht:

Modus	Farbe
NORMAL	Grün
CONFIG	Blau
ERROR	Rot

Abbildung 3: Status-Farben

NetworkManager
Die Klasse NetworkManager beinhaltet Funktionen, welche das WiFi des ESP32 verwenden. Es ist hier möglich den Access Point - und den Client-Modus zu aktivieren, nach verfügbaren Netzwerken zu suchen, sowie die MAC-Addresse des Adapters zu lesen. Diese Funktionen werden alle mithilfe der WiFi-Bibliothek (siehe Kapitel ...) verwirklicht.

AudioManager
Die Klasse AudioManager ist für den Empfang des Internet-Audio-Streams, für das Dekodieren dieses Streams und für die Ausgabe des dekodierten Audio-Signals zuständig. Dafür wird primär die ESP8266Audio-Bibliothek (siehe Kapitel ...) verwendet.

Logger
Die Klasse Logger ist für die Verwaltung von Logs, welche von anderen Klassen geschrieben werden, zuständig. In ihr befindet sich ein Vektor, welcher Tuples, bestehend aus Log-Text und Log-Zeitpunkt, speichert, definiert. Diese Logs sind essentiell für die Fehlerbehebung, da man sehen kann welche Aktionen im Programm ausgeführt wurden und wo Fehler aufgetreten sind. Die Klasse bietet Funktionen zum Hinzufügen von Logs und zum Anzeigen bestehender Logs. Die Funktionen sind alle statisch, das heißt man muss kein Objekt dieser Klasse erzeugen, um die Funktionen auszuführen. Dies macht Sinn, weil es möglich sein soll, von jeder Klasse direkt auf die Log-Funktionen zuzugreifen.

ServerManager
Die Klasse ServerManager ist für das Verwalten des Webservers, welcher auf dem MSA läuft, zustängig. In ihr werden die Funktionen, die durch die Aufrufe der Routen der REST-API ausgeführt werden, definiert. Die Kernfunktionen der Klasse werden vorallem durch die WebServer-Bibliothek (siehe Kapitel ...) bereitgestellt.

MemoryManager
Die Klasse MemoryManager ist für das Schreiben in und das Lesen aus dem EEPROM des Microcontrollers zuständig. Der EEPROM ist ein nichtflüchtiger Speicher, das heißt er behält die Daten auch nach einem Spannungsverlust des Microcontrollers. Dies macht vorallem für Daten Sinn, welche nicht jedes Mal neu definiert werden, wie z.B. Anmeldedaten des WLANs. (vgl. Santos 2018) Um die Funktionen des EEPROMs des ESP32 zu nutzen, wurde die Preferences-Bibliothek (siehe Kapitel ...) verwendet. Diese ist so aufgebaut, dass Daten mit Schlüssel-Wert-Paaren gespeichert werden. Gespeicherte Variabeln werden also eindeutig durch einen Schlüssel identifiziert.

BatteryManager
Die Klasse BatteryManager regelt den Ladevorgang des ESP32 und liest den Akkustand aus.

4.1.3 Programmablauf

Da die Arduino-Core-Bibliothek verwendet wurde ist der Programmablauf gleich wie bei einem standardmäßigen Arduino-Programm. Ausgeführt wird die Main-Datei welche aus den Funktionen „setup “ und „loop “ besteht. Die Setup-Funktion wird beim Start des Microcontrollers einmalig ausgeführt. In ihr werden z.B. Variabeln initialisiert und Aus- bzw. Eingänge initialisiert. Anschließend wird die Loop-Funktion in einer Endlosschleife ausgeführt. Wenn also das Ende der Loop-Funktion erreicht wird, fängt diese wieder von vorne an. Im folgenden Teil wird der Ablauf innerhalb der einzelnen Funktionen genauer beschrieben.

Setup
Im Setup werden zuerst Konstanten, welche später im Programm benötigt werden, definiert. Auch werden am Anfang die entsprechenden Instanzen der Singleton-Klassen abgerufen. Anschließend wird der Name des Adapters definiert. Dieser setzt sich, wie bereits im Kapitel ... erwähnt, aus MSA und den letzten drei Byte der MAC-Adresse zusammen. Er ist somit für jeden Adapter eindeutig. Der Name wird später als SSID des Access Points bzw. als Hostname für den Webserver verwendet. Nachdem der Name gesetzt ist, wird mit der Klasse MemoryManager abgefragt ob WLAN-Anmeldedaten im EEPROM gespeichert sind. Sind diese vorhanden, so versucht der Microcontroller eine Verbindung mit dem WLAN herzustellen. Ist dies möglich wird der Modus auf Normal gesetzt. Schlägt die Verbindung mit dem WLAN, z.B. durch falsche Anmeldedaten, fehl, wird der Modus auf Error gesetzt.

Loop
Am Anfang der Loop-Funktion wird abgefragt ob der Modus auf Error gesetzt ist. Ist dies der Fall, wird nichts mehr ausgeführt, das heißt die Funktion springt wieder an den Startpunkt. Ist der Modus allerdings nicht auf Error gesetzt, wird überprüft ob der Knopf gedrückt ist. Ist dieser gedrückt, wird der Modus auf Config gesetzt. Anschließend wird überprüft ob der Modus auf Normal gesetzt ist. Ist dies der Fall wird überprüft ob immer noch eine Verbindung mit dem WLAN besteht. Dies geschieht mithilfe der Klasse NetworkManager. Wenn eine WLAN-Verbindung besteht, wird mithilfe der Klasse ServerManager überprüft ob der Webserver läuft. Wenn nicht, wird dieser gestartet. Ist der Modus allerdings nicht auf Normal gesetzt, also auf Config, so wird zuerst überprüft ob der Access Point - Modus gestartet ist, da dieser für die Verbindung zwischen Client (Smartphone) und Server (MSA) benötigt wird. Wenn nicht, wird dieser schließlich gestartet und das mDNS-Protokoll (siehe Kapitel ...) wird konfiguriert. Diese Prozesse werden wieder mithilfe der NetworkManager-Klasse durchgeführt. Anschließend wird ebenfalls der Zustand des Webservers überprüft. Am Schluss der Loop-Funktion wird noch die Loop-Funktion der Klasse Audio-Manager

ausgeführt. Diese regelt das Audio-Streaming.

4.1.4 Erweiterungsmöglichkeiten

In Zukunft sind noch weitere Funktionalitäten, welche außerhalb der Diplomarbeit implementiert werden, für die Software geplant. Im Hinblick auf die Sicherheit wäre es durchaus sinnvoll, den Webserver des Microcontrollers auf HTTPS umzustellen. Somit wäre eine verschlüsselte Kommunikation mit diesem möglich. Zusätzlich wäre es auch denkbar eine Funktion zu implementieren, die es ermöglicht mehrere Adapter zu Gruppen zu verbinden. In diesen Gruppen soll dann jeweils der gleiche Audio-Stream synchron empfangen und ausgegeben werden. Die Synchronisierung der Adapter bringt aber eine hohe Komplexizität mit sich, da man per WLAN-Verbindung mit sehr großen Latenzen zu rechnen hat. Ein Ansatz wäre zum Beispiel, dass jeder Microcontroller in einem bestimmten Intervall die Zeit von einem NTP (Network Time Protocoll) - Server abfragt und somit auf allen Microcontrollern die Zeit annähernd gleich ist. Dann könnte man die Buffer, welche vom Audio-Stream erhalten werden, mit einem Zeitstempel versehen und sie beim Erreichen dieser Zeit abspielen. Das Hauptproblem dieser Lösung wäre, dass die Zeitdifferenz zwischen den Microcontrollern vermutlich zu groß wäre und somit eine hörbare Latenz entsteht.

4.2 Entwicklung Smartphone-App

In diesem Kapitel wird der Übergang der Planung in die Entwicklung der Smartphone-App beschrieben.

4.2.1 Zielsetzung

Es wurde festgelegt, dass die Smartphoneapp folgende Anforderungen erfüllen soll:

Verwalten von Adaptern
Mit der App soll es einerseits möglich sein, neue Adapter zu konfiguieren, andererseits, bereits hinzugefügte Adapter zu verwalten. Dabei können von den Adaptern Daten, wie beispielsweise MAC-Adresse und Akkustand angezeigt werden. Außerdem soll regelmäßig überprüft werden, ob der Adapter vom Smartphone aus erreichbar ist.

Verwalten von Radiostationen
Mit der App soll es außerdem möglich sein, nach Audiostreams von Internetradios zu suchen, und diese in einer Favoritenliste zu verwalten.

Verwalten von Verbindungen
Letztendlich soll es mit der App möglich sein, Verbindungen zwischen Adaptern und Internetradio-Streams herzustellen und diese zu verwalten. Dabei soll der Stream gestoppt und fortgesetzt werden können und die Lautstärke der Audio-Ausgabe des MSA einstellbar sein.

4.2.2 Struktur

In diesem Kapitel wird der Aufbau bzw. die Struktur der Smartphone-App genauer beschrieben. Die folgende Grafik stellt dabei die grobe Struktur der App dar: **Navigation**
Für die Navigation zwischen den Seiten der App wurde einerseits die Tab-Navigation und andererseits die Stack-Navigation von Expo Router verwendet. Dabei wird grob zwischen dem Ordner auth und dem Ordner tabs unterschieden. Der Ordner auth enthält die Seiten, welche für die Benutzerauthentifizierung, also Login und Registrierung, notwendig sind. Der Ordner tabs hingegen enthält alle Seiten, welche für registrierte Benutzer sichtbar sind und somit die eigentliche App darstellen. Tabs ist dabei in die drei Tabs: Verbindungen, Adapter und Musik gegliedert. Zwischen diesen Tabs kann mithilfe der Tab-Bar, welche unten in der App sichtbar ist, gewechselt werden. In den einzelnen Tabs befinden sich jeweils mehrere Seiten. Zwischen diesen Seite wird mittels Stack-Navigator gewechselt. Der Stack-Navigator arbeitet dabei nach dem sogenannten Last In - First Out - Prinzip (LIFO). Das heißt, dass die Seite welche als letztes zum Stack (auf Deutsch: Stapel) hinzugefügt wurde, als erstes wieder von diesem entfernt wird. Somit wird die Navigations-Historie festgehalten und der/die Benutzer/in kann jederzeit wieder auf die vorherigen Seiten zurückgehen. (vgl. *URLPI28* 2025)

4.2.3 APIs

RadioBrowser-API
Die RadioBrowser-API ist eine open-source API welche eine sehr umfangreiche Sammlung von Internet-Radios bereitstellt. Dabei ist es einerseits möglich, mithilfe dieser API neue Radiostationen zu der Sammlung hinzuzufügen und andererseits, bestehenden Radiostationen nach bestimmten Suchkriterien zu suchen und die Daten derer abzufragen. (vgl. *URLPI21* 2025) Wir haben die RadioBrowser-API in unserer Smartphoneapp verwendet um nach Radiosendern zu suchen, und in weiterer Folge Stream-URLs dieser, an den MSA zu senden. Der MSA ruft in weiterer Folge den jeweiligen Audio-Stream auf und empfängt diesen.

Firebase
Firebase ist eine Entwicklungsplattform von Google, welche verschiedene Dienste, die hilfreich für die Entwicklung von Apps bzw. Web-Apps sind, beretstellt. Firebase stellt dabei ein Backend für diese Apps bereit. Damit werden Prozesse, wie z.B. Benutzerauthentifizierung, Speicherung von Benutzern und Speicherung von Daten nicht lokal auf dem Gerät, sondern in der Cloud ausgeführt (vgl. *URLPI22* 2025) Wir haben Firebase zur Benutzerverwaltung in unserer Smartphoneapp verwendet. Dabei benutzen wir einerseits den Firebase Authentication Dienst für die Benutzerauthentifizierung und andererseits den Cloud Firestore Dienst für das Speichern von Benutzerdaten. Wir haben uns für Firebase entschieden, da wir dadurch den Ressourcenverbrauch unserer App minimieren und der/die Benutzer/in dadurch von jedem Gerät aus Zugriff auf seine/ihre Daten hat und somit nicht auf ein bestimmtes Gerät angewiesen ist. Außerdem müssen Dienste, wie z.B. die Benutzerauthentifizierung nicht neu entwickelt werden, was die App sicherer macht.

4.2.4 Funktionen

In der Datei Utilities wurden Funktionen definiert, welche dabei helfen Daten abzurufen bzw. zu speichern. Dazu gehören Abfragen der RadioBrowser-API, Abfragen an den jeweiligen Adapter und Zugriffe in den Speicher. Speicheroperationen wurden mithilfe der in Expo integrierten Bibliothek „AsyncStorage “ durchgeführt. Abfragen mittels HTTP wurden mit der ebenfalls in Expo integrierten Bibliothek „Axios “ durchgeführt.

Asynchrone Funktionen
Alle der Funktionen in der Utilities-Datei wurden als asynchrone Funktionen deklariert. Asynchrone Funktionen werden immer dann verwendet, wenn eine Funktion ein Ergebnis nicht direkt beim Aufruf liefern kann, sondern dies noch eine gewisse Zeit in Anspruch nimmt. Diese Verhaltensweise wird realisiert, indem die Funktion ein sogenanntes Promise-Objekt zurückgibt. In diesem Objekt sind Callback-Funktionen deklariert, welche je nachdem ob das Ergebnis verfügbar ist oder ein Fehler auftritt ausgeführt werden. Es gibt dabei die Callback-Funktion resolve(ergebnis), welche ausgeführt wird, wenn die Funktion das Ergebnis hat und die Funktion reject(fehler), welche ausgeführt wird, wenn in der Funktion ein Fehler auftritt. Das praktische dabei ist, dass man beim Aufruf einer asynchronen Funktion, hinten eine .then(function(result)) Funktion anhängen kann, welche erst ausgeführt wird, wenn das Ergebnis verfügbar ist. Man kann aber auch mit await warten, bis das Ergebnis verfügbar wird. Mit .catch(function(error)) kann man Fehler abfangen welche eventuell in der Funktion auftreten.

Liste von verfügbaren Ländern/Sprachen abrufen
Mit der Funktion getCountries() ist es möglich alle verfügbaren Ländernamen der RadioBrowser-API abzufragen. Der GET-Request wird dabei mit Axios ausgeführt. Als Rückgabe liefert die Funktion einen Promise, welcher entweder ein String-Array oder null ist. Das String-Array ist gefüllt mit allen verfügbaren Ländernamen. Null wird dann zurückgelifert, wenn bei der Abfrage ein Fehler auftritt. Es gibt außerdem noch eine Funktion um alle Sprachen der RadioBrowser-API aufzurufen. Diese Funktion ist gleich aufgebaut, der einzige Unterschied liegt in der URL, welche abgefragt wird. Die Ländernamen und Sprachen werden später für die Filterung der Radiostationen benötigt.

Stationen abfragen
Die Funktion getStations(language, country) fragt alle Radiostationen, welche in der übergebenen Sprache bzw. dem übergebenen Land verfügbar sind, von der RadioBrowser-API ab und gibt diese dann als JSON-Objekt zurück. Wenn bei der Abfrage ein Fehler entsteht, wird null zurückgeliefert.

Liste von Elementen vom Speicher lesen
Die Funktion getFavouriteStations() fragt die Favoritenliste von Radiostationen, welche im Speicher gespeichert ist, ab. Diese wird als Array aus Station-Objekten zurückgegeben. Wenn keine Favoriten im Speicher sind, wird wiederum null zurückgegeben. Die Funktion zur Abfrage der Liste von

gespeicherten Adaptern heißt getAdapters() und ist gleich aufgebaut. Der einzige Unterschied liegt im Rückgabetyp. Hier wird nähmlich ein Array aus Adapter-Objekten zurückgegeben.

Element zum Speicher hinzufügen
Die Funktion addFavouriteStations(stationList) fügt eine Liste aus Stationen, welche als Array aus Station-Objekten übergeben wird, der gespeicherten Favoritenliste hinzu. Wenn noch keine Favoritenliste gespeichert ist, wird eine neue erstellt. Die Funktion zum Hinzufügen eines neuen Adapters heißt dementsprechend addAdapter(adapter) und erwartet als Parameter ein einzelnes Objekt der Klasse Adapter.

Element vom Speicher entfernen
Die Funktion removeFavouriteStation(uuid) entfernt die Station mit der übergebenen UUID von der gespeicherten Favoritenliste, wenn diese dort vorhanden ist. Die UUID wird bereits von der RadioBrowser-API mitgeliefert und hilft dabei, jede Station eindeutig zu identifizieren. Die Funktion zum Entfernen eines Adapters aus dem Speicher heißt removeAdapter() und erwartet die Mac-Addresse des zu entfernenden Adapters als Argument. Durch diese ist auch jeder Adapter eindeutig identifizierbar.

Liste im Speicher leeren
Die Funktion clearFavouriteStationList() wird die Favoritenliste im Speicher geleert, das heißt es sind keine Favoriten mehr gespeichert. Die Funktion zum leeren der Adapter-List heißt clearAdapterList() und löscht die Liste aller konfigurierten Adaptern aus dem Speicher.

4.2.5 Design

Beim Style des User Inferface der App wurde auf Übersichtlichkeit und wenig Komplexität gesetzt. Der/Die Benutzer/in soll sich in der App schnellstmöglich zurecht finden. Durch die gute Benutzbarkeit werden allerdings gleichzeitig die Konfigurationsmöglichkeiten eingeschränkt, was die Flexibilität der App senkt. Bei der Auswahl der Farben, welche in der App wurde darauf geachtet, dass diese die App modern und schlicht erscheinen lassen. Für die App wurde eine Darkmode implementiert, weil dieser in den meisten modernen Applikationen verwendet wird und dieser die App somit moderner aussehen lässt. **Farben**
Bei der Farbauswahl wurden als Hauptfarben eher dunklere Farben verwendet, um die App im Darkmode darzustellen. Im Kontrast zu den primär dunklen Farben wurden knallige, satte Farben im Torquise-Ton für die Steuerelemente, wie z.B. Buttons verwendet. Im folgenden ist eine Tabelle mit den verwendeten Farben und deren Farbcodes im Hex-Format abgebildet:

Bezeichnung	Farbcode (hex)
grey	# 2B2C28
lightTurquoise	# 7DE2D1
white	# FFFAFB
red	# d90b0b
lightGrey	# 3e403a

Abbildung 4: App-Farben

4.2.6 Struktur

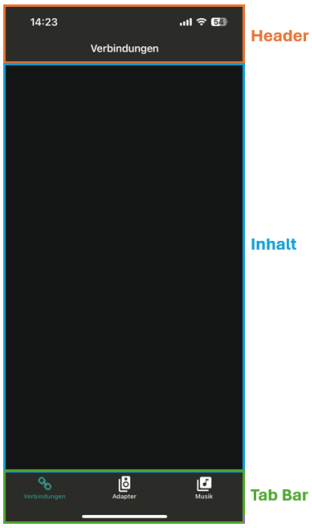

Abbildung 5: Aufbau Seite

Die Seiten der App sind alle gleich strukturiert. Ganz oben ist der Header, in welchem der Seitenname steht. Ganz unten ist die Tab Bar, welche die einzelnen Tabs anzeigt und auch sichtbar macht, in welchem Tab man sich aktuell befindet. Zwischen dem Header und der Tab Bar ist der jeweilige Inhalt der Seite. Die App ist in drei Tabs gegliedert. Im folgenden werden die einzelnen Tabs genauer beschrieben.

Verbindungen
Dieser Tab ist für die Verwaltung von Verbindungen zwischen Adaptern und Audioquellen zuständig. Hier kann konfiguriert werden, welcher Adapter welchen Stream empfangen soll. Es ist außerdem möglich, laufende Streams zu stoppen, deren Lautstärke zu ändern und zu löschen.

Adapter
Dieser Tab ist für die Verwaltung von Adaptern zuständig. Es ist hier möglich, gespeicherte Adapter anzusehen, zu löschen und neue Adapter hinzuzufügen. Der Tab besteht aus den Seiten index und addAdapter, welche im Kapitel Seiten noch genauer beschrieben werden.

Musik
Dieser Tab ist für die Verwaltung von Audio-Quellen zuständig. Vorerst werden hier als Quellen nur Internet-Radios verwendet. In Zukunft wäre es aber auch denkbar, Streaming-Dienste als Quellen zu implementieren. Mit jetztigem Stand ist es in diesem Tab möglich, neue Radiosender zu der Favoritenliste hinzuzufügen und die Favoritenliste anzusehen bzw. Favoriten zu löschen. Der Tab besteht aus den Seiten index und searchStations, welche im Kapitel Seiten noch genauer beschrieben werden.

4.2.7 Zustandsverwaltung

Um Zustände, welche in mehreren Teilen der App benötigt werden, global bereitzustellen, wird der Context von React verwendet. Context ermöglicht es Zustände global in der App bereitzustellen und diese in den einzelnen Komponenten aufzurufen. Dabei werden bei der Änderung eines Zustands die betroffenen Komponenten sofort aktualisiert. Um den Context zu bereitzustellen, wird ein sogenannter Context-Provider verwendet. Alle Komponenten welche in den Context-Provider eingebettet sind, können auf den Zustand von diesem zugreifen. (vgl. *URLPI23* 2025) In unserer Smartphoneapp gibt es folgende ContextProvider:

- **UserContext**

  Stellt das Benutzerobjekt bereit. Dieses beinhaltet wichtige Benutzerdaten, welche an mehreren Stellen der App verwendet werden. Das Benutzerobjekt wird von Firebase Authentication abgefragt.

- **AdapterContext**

  Stellt eine Liste von hinzugefügten Adaptern bereit. Diese Liste wird von Firebase Firestore abgefragt. Falls sich die in Firestore gespeicherte Liste ändert, wird die Liste, welche der AdapterContext bereitstellt, automatisch aktualisiert.

- **StationContext**

  Stellt eine Liste von hinzugefügten Radiostationen bereit. Diese Liste wird ebenfalls von Firebase Firestore abgefragt. Die hier gespeicherte Liste wird ebenfalls automatisch aktualisiert, falls sich die Liste in Firestore ändert.

4.2.8 Benutzerverwaltung

Die Benutzerdaten sind durch ein im UserContext gespeichertes Benutzerobjekt in der gesamten App global verfügbar. Das Objekt wird am Anfang der App, falls vorhanden, aus dem lokalen Gerätespeicher ausgelesen oder mittels Login von Firebase abgefragt. Der Status des Benutzerobjektes wird mittels eines EventHandlers verfolgt. Das heißt, wenn sich der Status des Benutzerobjektes,

z.B. durch einen Logout, ändert, wird Die Verwaltung von Benutzerdaten wurde primär mit Firebase (siehe APIs) verwirklicht. Beim Start der App wird abgefragt ob im lokalen Speicher ein Benutzer-Objekt gespeichert ist. Wenn dies der Fall ist, ist der Teil Tabs für den/die Benutzer/in erreichbar. Ist dies nicht der Fall, so wird der/die Benutzer/in zur Login-Seite weitergeleitet. Dort ist es möglich

4.2.9 Komponenten

In diesem Kapitel werden die selbst erstellen React-Komponenten genauer beschrieben.

AdapterItem
Die Komponente AdapterItem wird dazu verwendet, die Daten eines einzelnen Adapters abzufragen und diese anzuzeigen. Dabei muss man dieser Komponente ein Objekt der Klasse Adapter als Argument mitgeben. Beim Rendern wird direkt versucht eine Verbindung zum Webserver mit der IP-Adresse des übergebenen Adapters herzustellen. Ist dies erfolgreich, so wird ein GET-Request auf die /getInfo - Route des Adapters mittels Axios durchgeführt. Dabei werden die erhaltenen Informationen grafisch dargestellt. Zu diesen Daten zählt Lautstärke und Akkustand des Adapters. Ist der Adapter allerdings nicht erreichbar, werden die Daten nicht angezeigt. Gleichzeitig ändert sich die Hintergrundfarbe der Komponente zu einem helleren Grauton und es erscheint rechts eine durchgestrichene Wolke. Dies soll signalisieren, dass der Adapter nicht erreichbar ist. Diese Informations-Abfrage wird in einem Intervall von 5 Sekunden ausgeführt, um Änderungen in den Daten des Adapters bzw. der Verbindung des Adapters schnellstmöglich zu signalisieren. Dieses Intervall wird mit der JavaScript-Methode setInterval verwirklicht. Am Ende des renderns, wird das Interval noch geschlossen, um zu vermeiden, dass es mehrere Instanzen davon gibt.

AdapterList
Die Komponente AdapterList stellt eine Liste dar, in der alle bisher hinzugefügten Adaptern aufgelistet sind. Die Adapter werden dabei mithilfe der AdapterItem-Komponente dargestellt. Die Adapter werden mithilfe der Methode getAdapter() aus dem Speicher abgefragt. Wenn noch keine hinzugefügten Adapter vorhanden sind bzw. die Methode getAdapter() null zurückgibt, wird die Komponente ErrorScreen dargestellt.

AddToListButton
Die Komponente AddToListButton wurde verwendet, um ein drückbares Icon darzustellen, welches symbolisieren soll, ein weiteres Element zu einer Liste hinzuzufügen. Für die Umsetzung der Drück-Funktion wurde die Pressable-Komponente von React Native verwendet. Das Icon wurde von der Entypo-Bibliothek importiert.

BatteryIndicator
Die Komponente BatteryIndicator zeigt die Batterieladung in Prozent mit dazugehörigem Icon an.

Dabei erwartet sie als Argument die Batterieladung in Prozent. Je nachdem, in welchem Bereich diese Batterialadung liegt, wird ein entsprechendes Icon gerendert. Wenn für die Batterieladung ein negativer Wert (-1) übergeben wird, bedeutet dies, dass der Akku aktuell geladen wird. In diesem Fall, wird ein Lade-Icon dargestellt.

ConnectionItem

...

DeleteButton

Die Komponente DeleteButton stellt ein drückbares Icon dar, welches signalisieren soll, ein Element zu löschen. Dabei wurde die Drück-Funktion mit der Pressable-Komponente realisiert. Das Icon wurde von der FontAwesome-Bibliothek importiert.

ErrorScreen

Die Komponente ErrorScreen stellt eine Ansicht bereit, die signalisieren soll, dass ein Fehler aufgetreten ist. Dabei wird die Fehlermeldung als Text angezeigt und man hat die Möglichkeit auf einen Knopf zu drücken, welcher dann eine Aktion ausführt. Als Parameter werden der zu anzeigende Text, der Text des Knopfs und die Funktion, welche beim Druck auf den Knopf ausgeführt wird, übergeben. Der Fehlertext wird durch die Text-Komponente angezeigt und für den Knopf wurde eine Button-Komponnte verwendet.

FavouriteStationList

Die Komponente FavouriteStationList stellt eine Liste aus mehreren StationItem-Komponenten dar. Die Daten dafür, werden beim Rendern der Komponente mithilfe der Funktion getFavouriteStations() aus dem Speicher ausgelesen. Wenn noch keine Favoriten im Speicher sind, wird die ErrorScreen-Komponente gerendert. Dabei ist es möglich, bei längerem Drücken auf ein FavouriteStationItem dieses zu selektieren und in weiterer Folge aus der Liste zu löschen. Die Selektierung wird mit einem kurzen Klick auf ein FavouriteStationItem wieder aufgehoben. Beim Druck auf die AddToListButton-Komponente, wird zum Screen „Stationsearch “ navigiert. Beim Druck auf die DeleteButton-Komponente, wird die ausgewählte Komponente aus dem Speicher gelöscht.

LoadingScreen

Die Komponente LoadingScreen soll signalisieren, dass ein Vorgang durchgeführt wird und deshalb der Screen bzw. die Komponenten noch nicht angezeigt werden können. Dies kann zum Beispiel das Laden von Daten sein. Als Argument wird der Text übergeben, welcher in der Komponente angezeigt wird. Der Ladevorgang wird mit der ActivityIndicator-Komponente signalisiert.

NetworkItem

Die Komponente NetworkItem stellt Daten von einem Netzwerk dar. Dabei werden als Argumente

die SSID und RSSI des Netzwerks übergeben. Die SSID wird mit der Text-Komponente angezeigt. Abhängig von dem Wert der RSSI werden verschiedene Icons angezeigt. Diese Icons stellen die Stärke des Netzwerks dar. Das Argument selected gibt an, ob die Komponente ausgewählt wurde. Wenn dies der Fall ist, verändert sich die Hintergrundfarbe der Komponente.

Die Komponente StationItem stellt Name und Icon einer Radiostation dar. Die Radiostation wird dabei bei den Argumenten, als Objekt der Klasse Station übergeben. Das Argument selected gibt an, ob die Station ausgewählt ist. Wenn dies der Fall ist, verändert sich die Hintergrundfarbe der Komponente.

TextInputWindow
Die Komponente TextInputWindow ermöglicht es Text in einem Fenster, welches vor dem anderen Content gerendert wird, einzugeben. Dabei ist unter dem Text ein Knopt „Bestätigen “ und ein Knopt „Abbrechen “ verfügbar. Der Text, welcher ganz oben angezeigt wird, wird als Argument übergeben. Außerdem wird mit isPassword festgelegt, ob die Eingabe in das Textfeld sichtbar sein soll oder nicht und die übergebenen Funktionen onEnter bzw. onCancel bestimmen, was passiert, wenn man den „Abbrechen “- oder „Bestätigen “ Knopf drückt.

VolumeIndicator
Die Komponente VolumeIndicator zeigt eine Lautstärke in Prozent an, mit einem Lautstärke-Icon davor. Das Lautstärke-Icon wird von der Feather-Bibliothek importiert. Die Lautstärke in Prozent wird mit einer Text-Komponente angezeigt. Als Argument wird die Lautstärke in Prozent übergeben. Wenn diese kleiner als 0 ist, wird ein Icon angezeigt, welches symbolisiert, dass die Lautstärke stumm ist.

WifiItem
Die Komponente WiFiItem dient dazu, die Informationen eines WLAN-Netzwerkes darzustellen. Zu diesen Informationen zählt die SSID, welche sozusagen der Name des Netzwerks ist und die RSSI, welche die Signalstärke des Netzwerks angibt. Diese zwei Werte werden als Parameter der Komponente übergeben.

4.2.10 Seiten

In diesem Kapitel werden die einzelnen Seiten der App genauer beschrieben.

Startseite Verbindungen
Diese Seite ist die Startseite des Tabs Verbindungen. Auf ihr werden aktive Verbindungen zwischen Adaptern und Audio-Quellen angezeigt. Die Darstellunge wird mithilfe der Komponente ConnectionList verwirklicht. Es ist hier möglich neue Verbindungen zu erstellen, aktive Verbindungen zu trennen, die Lautstärke von aktiven Verbindungen zu ändern und aktive Verbindungen zu pausieren

bzw. fortzusetzen.

Verbindung hinzufügen
Diese Seite befindet sich im Tab Verbindungen. Hier ist es möglich neue Verbindungen hinzuzufügen. Dazu muss zuerst ein Adapter ausgewählt werden und in weiterer Folge eine Audio-Quelle, von der der Adapter den Stream empfangen soll. Dabei können nur verbundene Adapter, das heißt, Adapter die im Netz erreichbar sind, ausgewählt werden. Zum Anzeigen der Adapter wurde die Komponente AdapterList verwendet, zum Anzeigen der Stationen die Komponente Station List. Der Parameter editable wurde dabei bei beiden Stationen auf false gesetzt, da ein bearbeiten der Listen in dieser Ansicht nicht möglich sein soll. Wenn jeweils ein Adapter und eine Station ausgewählt ist, ist es möglich den Knopf unten zu drücken und die Verbindung wird erstellt.

Startseite Adapter
Diese Seite ist die Startseite des Tabs Adapter. Auf ihr werden hinzugefügte Adapter angezeigt. Dies geschieht mithilfe der Komponente AdapterList. Der Parameter editable der AdapterList hat den Wert true, da es hier möglich sein soll neue Adapter hinzuzufügen bzw. bestehende Adapter zu entfernen.

Startseite Musik
Diese Seite ist die Startseite des Tabs Musik. Auf ihr werden die Favoriten der Radiostationen mithilfe der Komponente StationList dargestellt. Der Parameter selectable der StationList-Komponente hat dabei den Wert true, da es auf dieser Seite möglich sein soll, neue Favoriten hinzuzufügen bzw. bestehende Favoriten zu entfernen. In Zukunft wäre denkbar, dass hier noch Streaming-Dienste implementiert werden, welche dann auch als Audio-Quelle benutzt werden können.

Stationen filtern
Diese Seite befindet sich im Tab Musik. Hier ist es möglich Land und Sprache auswählen, nach denen in weiterer Folge die Stationen auf der Seite Stationen auswählen gefiltert werden. Der Sprachen-Datensatz wird dabei mithilfe der Methode getLanguages() von Utilities abgerufen. Der Länder-Datensatz wird mit der Methode getCountries() von Utilities aufgerufen. Mit druck auf den Knopf wird die Seite Stationen auswählen aufgerufen und die Paramter werden an diese übergeben.

Stationen auswählen
Diese Seite befindet sich im Tab Musik. Hier ist es möglich Stationen, welche zu der Favoritenliste hinzugefügt werden sollen, auszuwählen. Als Parameter werden Ländername und Sprache übergeben. Dementsprechend werden die zu auswählenden Stationen nach diesen Parametern gefiltert. Das Abrufen der Stationen erfolgt mithilfe der Methode getStations() und die Darstellung mithilfe der StationList-Komponente. Nach klick auf den Haken rechts unten werden die Stationen mithilfe der Funktion addFavouriteStations() zur Favoritenliste hinzugefügt.

Favoriten-Auswahl
Auf dieser Seite wird eine Liste, mit allen Stationen, welche dem ausgewählten Land und der ausgewählten Sprache entsprechen, angezeigt. Hier kann man Radio-Stationen markieren und mit Klick auf den Haken diese zu seiner Favoritenliste hinzufügen.

4.3 Design Adaptergehäuse

Das Adaptergehäuse trägt einen wesentlichen Teil zur Sicherheit des Endverbrauchers sowie zur optimalen Funktionalität der Komponenten bei. Zudem soll es möglichst kompakt sein.

4.3.1 Grundsätzlicher Aufbau

Die Grundlage für den Prototyp bildet ein Kasten mit Deckel.
Das Gehäuse wurde mit frei schwebenden, jedoch an den Wänden befestigten Stützen ausgestattet, um den Mikrocontroller fest montieren zu können. Der Prototyp wurde zudem mit kleinen Zylindern auf den Stützen ausgestattet, um den Mikrocontroller mit seinen bereits Vorhandenen Aussparungen darauf platzieren zu können. Der Digital-/Analogwandler und der Akku-Laderegler finden auch auf solchen Stützen ihren Platz. Der Gedanke dahinter war, den Akku unter den Bauteilen zu platzieren. Mehr dazu im Teil "Wärmeableitung". Zudem wurden in einer Wand des Gehäuses Aussparungen für die Buchsen platziert. Die Aussparungen für die RGB-LED und den Taster wurden im Deckel platziert. Eine Art Abdeckung für den Taster selbst wird auf diesen geklebt um ein gleichbleibendes Erscheinungsbild des Gehäuses zu behalten. Der Deckel, der von oben auf das Gehäuse gedrückt wird, schließt dieses. Im Deckel ist zusätzlich ein Belüftungsgitter eingelassen.

4.3.2 Wärmeableitung

Wärmeableitung ist wichtig, da Mikrocontroller, wie alle anderen Prozessoren auch, bei intensivem Betrieb Hitze entwickeln. Laut einer eigens durchgeführten Messung wird der in diesem Fall eingesetzte ESP32 meistens nur rund 38°C warm. Bei hoher Rechenleistung sind jedoch auch höhere Temperaturen möglich. Der ESP32 hat laut Hersteller eine mögliche Betriebstemperatur (Umgebungstemperatur) von –40°C bis +125°C. Damit die entstandene Wärme gut ableiten kann und keine der Komponenten stark beeinflusst (vor allem den Akku, da dieser bei hohen Temperaturen eine Explosionsgefahr darstellt), wird der Mikrocontroller im Gehäuse oben, also auf Stützen angebracht. Die aufsteigende Wärme kann somit nach oben durch das dafür vorgesehene Gitter entweichen und staut sich somit nicht im Inneren des Gehäuses. Der Akku liegt dementsprechend unter dem Mikrocontroller und allen anderen Komponenten und wird durch die abstrahlende Hitze dementsprechend nur etwas wärmer als Raumtemperatur.

vgl. `https://www.espressif.com/en/products/socs/esp32`

4.3.3 Virtuelles 3D-Design

Um ein 3D-Modell des Prototypen zu zeichnen, wurde AUTODESK Fusion verwendet. Der Anbieter beschreibt seine Software folgendermaßen: Äutodesk Fusion verbindet Ihren gesamten Fertigungsprozess durch die Integration von CAD, CAM, CAE und PCB in einer einzigen Lösung, mit der Sie Ihre Ideen verwirklichen und praktisch alles fertigen können."
Wenn man schon früh beachtet, dass die Prototypen mit einem 3D-Drucker gefertigt werden, kann man schon das 3D-Design für eine gute Druckbarkeit auf 3D-Drucker anpassen. Damit ist hauptsächlich gemeint, überhängende Drucke, komplizierte Stützstrukturen oder ähnliches zu vermeiden.

Basis
Die Basis des Adapters bildet ein 71x58x27mm großer Kasten mit 2mm Wanddicke. Aus Erfahrung kann man sagen, dass diese Dicke bei 3D-Drucken stabil ist, während sie jedoch nicht zu klobig wirkt.
Die Stützen des Mikrocontroller beginnen auf 11mm Höhe und sind auf einer Längenseite 6x6x5mm und auf der anderen 6,9x6x5mm groß. Sie sind jeweils mit einer oder zwei Seiten an der Wand der Basis und somit überhängend. Dies wird im Teil "Drucken des Prototyps/Stützstruktur"noch wichtig.

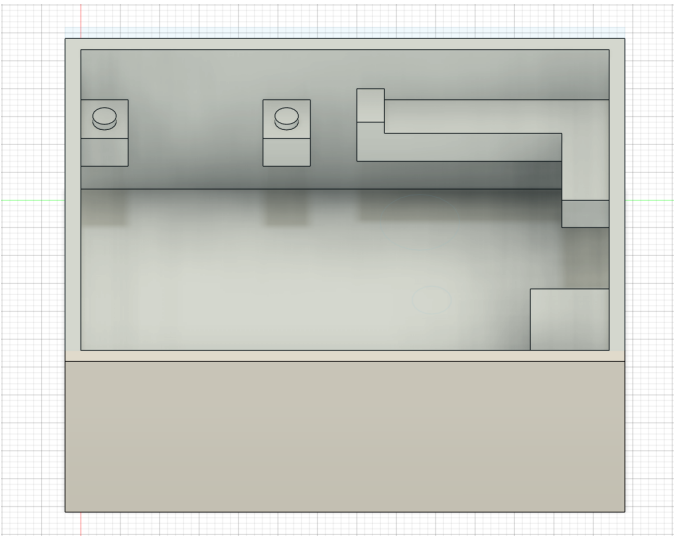

Abbildung 6: Abbildung der Stützen für die Komponenten

Auf den Stützen befinden sich jeweils Zylinder, die genau auf die Aussparungen des Mikrocontroller vermessen wurden. Die Zylinder haben einen Durchmesser von exakt 3mm. Der Abstand der Mittelpunkte dieser Zylinder war bei unserem Modell 47,10mm in der Länge und 23,10mm in der Breite. Auf einer Seite befindet sich zwischen Wand und Zylinder etwas mehr Platz, da der USB-C Port des Mikrocontrollers etwas über diesen herausragt. Deswegen auch der zuletzt erwähnte Ver-

satz der Stütze von 0,9mm.
Die Breite der Basis ist also genau auf die Länge von dem von uns benutzten Mikrocontroller zugeschnitten.
In den gegenüberliegenden Ecken der Basis befinden sich der Digital-/Analogwandler und der Laderegler. Die Maße des Digital-/Analogwandler sind 31,8x17,2mm. Die Maße des Laderegler sind 28x17,45mm. Beide liegen, wie der Mikrocontroller auch, auf überhängenden, 5mm hohen Stützen auf. Aufgrund der USB-C Buchse wird der Halt des Laderegler noch von einer 3,5mm breiten 2mm-Erhöhung am Ende der Stütze verstärkt. Die USB-C Buchse wurde passend zum Laderegler und der Norm entsprechend (8,34x2,56mm Größe) eingelassen. Der Digital-/Analogwandler wird aufgrund der 3,5mm Klinkenbuchse von einer herabstehenden Wand gestützt, die im Teil "Deckel"genauer beschrieben wird. Die Aussparung der AUX-Buchse wurde auch passend für den Digital-/Analogwandler platziert und hat einen Durchmesser von 6mm.

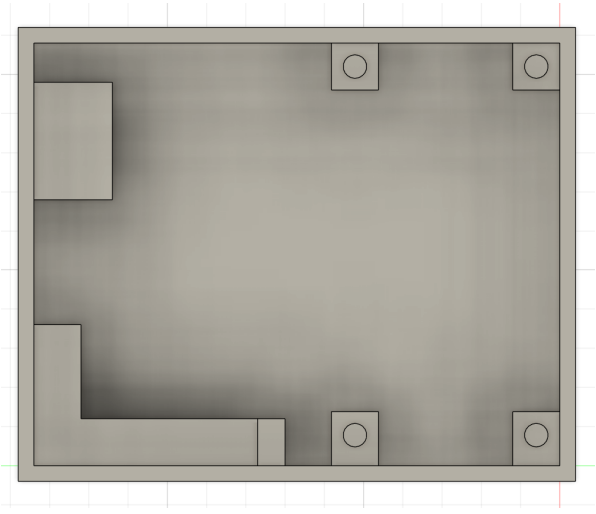

Abbildung 7: Draufsicht der Basis in AUTODESK Fusion

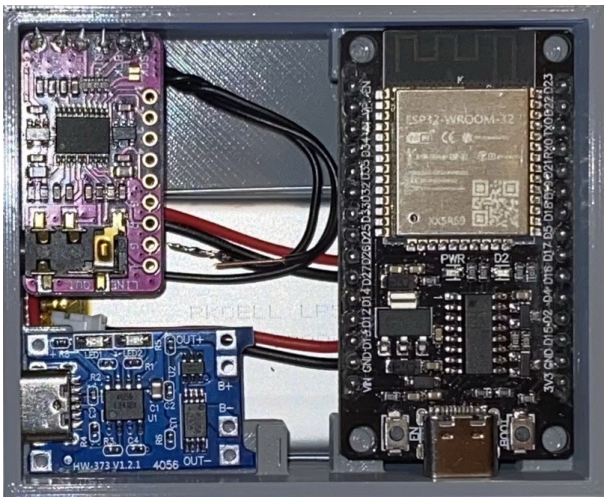

Abbildung 8: Bild der Basis mit den wichtigsten Komponenten an Ort und Stelle

Deckel

Der Deckel des Adapters ist grundsätzlich, wie die Wände der Basis, 71x58x2mm groß. Dieser hat jedoch eine zentrierte 67x54x1mm große Stufe. Mit dieser Stufe lässt sich der Deckel kleberlos auf den Adapter setzen und hält aufgrund der Eigenschaften des 3D-Drucks auch, zumindest für den Prototypen, fest genug. Wie schon erwähnt, wird der DAC durch eine herabstehende Wand zusätzlich gestützt. Die Maße dieser Wand sind 32x2x10mm. Es würde keinen Sinn machen, die Wand wie beim Laderegler von der Außenwand aus überhängend zu machen, da der DAC länger als der Laderegler ist und die Buchse sich eher mittig in der entsprechenden Außenwand der Basis befindet.
An der freien Seite der Stützwand (nicht die des DAC) befindet sich ein runder Schacht für die RGB-LED mit 5mm Durchmesser und eine Aussparung für den Aufsatz des Tasters mit 10,10mm Durchmesser. Der Taster mit den Grundmaßen 6x6mm wird durch eine Art U-Form aus Wänden im Gehäuse gehalten. Eine Wand davon bildet die gerade eben beschriebene Stützwand. Die untere Wand, auf der der Taster aufliegt, hat zudem zwei auf den Taster angepasste, 1mm große Aussparungen für die zwei Pole des Tasters.
Wenn man in der Draufsicht auf den Deckel schaut, ist dort wo sich der Prozessor selbst befindet, ein Gitter in den Deckel eingelassen. Dieses Gitter hat die Maße 19,9x21,6mm, was etwas größer als der Prozessor des ESP32 ist. Das Gitter besteht aus diagonalen Streben, die jeweils 1,5mm breit sowie 1,5mm weit voneinander entfernt sind. So entsteht eine einfache und stabile Möglichkeit, Wärme durch den Deckel abzuleiten.

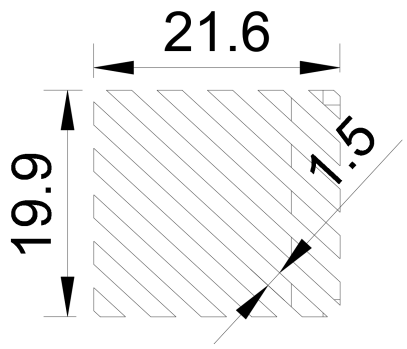

Abbildung 9: bemaßte Skizze des Belüftungsgitters in AUTODESK Fusion

Tasteraufsatz
Den Tasteraufsatz bildet ein Zylinder mit 9,5mm Durchmesser, auf dem sich ein weiterer Zylinder mit 5,5mm Durchmesser und zentrierter 3,5mm Aussparung für den Taster befindet.

vgl. `https://www.autodesk.com/de/products/fusion-360/overview`
vgl. `https://www.elektronik-kompendium.de/sites/com/2009021.htm`

4.4 Fertigung Adaptergehäuse

Das Material unseres Gehäuses wurde auf Kunststoff begrenzt. Für die Fertigung von Kunststoffgehäusen gibt es hauptsächlich diese Möglichkeiten (in diesem Fall kommt mangels Alternativen nur der 3D-Druck in Frage):

Spritzgießen
„Beim Spritzgießen wird der Kunststoff aus einem Plastifiziergerät (erwärmt den Kunststoff auf Schmelztemperatur) in einen Hohlraum (Formwerkzeug) gespritzt, in welchem er erst verdichtet wird und dann erkaltet."
Ein Vorteil für dieses Verfahren ist, dass auch komplizierte Formteile voll automatisiert sehr schnell in hohen Stückzahlen produziert werden können. Der große Nachteil sind jedoch die hohen Stückkosten für die Formwerkzeuge.

vgl. `https://www.chemie.de/lexikon/Kunststoffverarbeitung.html`

3D-Druck
Die zwei gängigsten 3D-Druck-Methoden sind Filament und Resin (Harz). Aufgrund des hohen Aufwands, den ein Resin-Drucker mit sich bringt, wurde für dieses Projekt die Methode mit Filament gewählt.
Beim 3D-Drucken durch Fused Deposition Modeling (Schichtschmelzverfahren) wird Kunststoff in Drahtform (Filament) (die häufigsten Dicken sind 2,85mm (allgemein als 3mm bezeichnet) und 1,75mm wobei die 1,75mm Version weltweit am häufigsten verbreitet ist) durch beheizte Düsen

geleitet und somit geschmolzen. Das nun weiche Filament wird in Schichten auf die Druckplatte aufgetragen und erhärtet kurz darauf. Durch dieses Schichten lassen sich präzise Körper aus Kunststoff bauen.

vgl. https://www.printer-care.de/de/drucker-ratgeber/wie-funktioniert-ein-3d-drucker
vgl. https://help.prusa3d.com/de/glossary/175-mm_134816

4.4.1 Drucken des Prototypen

Die Gehäuse-Prototypen wurden mit einem „PRUSA MK4S“ 3D-Drucker gefertigt. Alle FFF-Drucker von Prusa sind grundsätzlich für 1,75-Filament konfiguriert.

Druckeinstellungen
Die Temperatur der Build Plate lag bei uns Standartmäßig auf 60°C. Die Drucktemperatur, also die der Nozzle (Düse) lag bei etwa 185°C.
Das erste Layer wurde mit 0,15mm Dicke gedruckt, die restlichen mit 0,2mm Dicke.
Als Infill-Pattern wurde „Grid“ mit 20% Dichte und 4mm Line-Distance gewählt.
Das Drucken verlief mit den von uns gewählten Einstellungen reibungslos, jedoch an manchen Stellen etwas unsauber. Beispielsweise war das Ergebnis der Aussparung für den Taster im Deckel des Adapters so ungenau, dass der Durchmesser des Aufsatzes für den Taster um 0,5mm verkleinert werden musste. Sonstige Ungenauheiten stellten, zumindest abgesehen von der Optik, kein Problem dar.
Man darf auch nicht vergessen, dass gewisse Drucke (Gehäusedeckel und -basis) Stützstrukturen erfordern. Stützstrukturen sind Konstruktionen, die der Drucker zur Unterstützung stark überhängender oder freischwebender Strukturen druckt. Jedoch sind diese nur temporär. Da das Stützmaterial nicht so fest an der Druckfigur haftet, wie die Teile der Figur selbst, kann es nach dem Druck mit einer herkömmlichen Zange entfernt werden. Diese Stützstrukturen lassen sich in der Slicer-Software genau einstellen und konfigurieren. So kann man unter anderem sicherstellen, dass das Entfernen einfach möglich ist und die Stützstruktur die Figur selbst nicht all zu stark beeinflusst.

vgl. https://help.prusa3d.com/de/glossary/175-mm_134816

4.5 Zusammensetzen des Prototypen

Die Komponenten werden jeweils einzeln direkt mit dem Mikrocontroller verbunden. Bei diesem Prototyp lässt sich der Mikrocontroller nämlich vorerst als Platine betrachten, welcher zusätzliche Komponenten zugefügt werden.

4.5.1 Schaltplan

Die Verdrahtung zwischen den Komponenten und dem Mikrocontroller ist folgendermaßen gelöst:

Digital-/Analogwandler

BCK (Wandler) an D26 (Mikrocontroller)
LCK (Wandler) an D25 (Mikrocontroller)
DIN (Wandler) an D22 (Mikrocontroller)

Taster
1 (Taster) an 3V3 (Mikrocontroller)
2 (Taster) an D12 (Mikrocontroller)

RGB-LED
R (LED) an D15 (Mikrocontroller)
G (LED) an D2 (Mikrocontroller)
B (LED) an D4 (Mikrocontroller)
GND (LED) an GND (Mikrocontroller)

4.5.2 Verdrahten

Bei unserem Prototypen wurden alle Komponenten, der Pinbelegung entsprechend, mit dem Mikrocontroller verbunden. Dabei kamen Drahtkabel, Litzenkabel und Steckkabel (herkömmliche Jumper Kabel) zur Verwendung. Dies hatte hauptsächlich den Grund, dass zum Beispiel nur mit Drähten nicht alle Verbindungen optimal möglich gewesen wären. Das ist hauptsächlich der Löthaftung an einigen Komponenten geschuldet. Um keine Wackelkontakte, oder gar unterbrochene Verbindungen zu riskieren, wurde für jede Verbindung einzeln entschieden, welches der genannten Verfahren sich am besten eignet. So entstanden zum Beispiel Steckverbindungen, gelötete Drahtverbindungen oder Mischungen aus gelöteten Litze- und Steckverbindungen (jeweils am anderen Ende des Kabels).

Löten
„Das Löten ist das Verbinden von Metallteilen durch eine Metalllegierung (das Lot) unter Einfluss von Wärme/Hitze.“
Man unterscheidet grundsätzlich zwischen Weich- und Hartlöten. Ausschlaggebend dabei ist die Schmelztemperatur des Lots. So haben Weichlote eine Schmelztemperatur unter 450°C während Hartlote erst ab 450°C bis etwa 1100°C schmelzen. „Das Weichlot wird verwendet, wenn die Verbindung zweier Metalle dicht und Leitfähig sein soll und um die mechanische Belastbarkeit keine hohe Anforderung gestellt wird. “ Da die in diesem Projekt benutzten Bauteile jedoch keine höheren Temperaturen vertragen, war Weichlöten für dieses Projekt die einzige Wahl.
Um einen Lötvorgang aus eigener Erfahrung möglichst kurz zu beschreiben:
Man hat beispielsweise zwei Kabel die man verbinden möchte und die Verbindung sollte möglichst fest halten und gut leiten. Zuerst müssen die Enden der Kabel abisoliert werden. Es kann helfen, wenn man die Enden schon vor der eigentlichen Verbindung sozusagen etwas „verzinnt“. Nun richtet

man die Kabel zueinander so aus, wie sie später fixiert sein sollen. Man fährt mit dem Ende des Lots (umgangssprachlich Lötzinn) an die heiße Spitze des Lötkolben und schmilzt so etwas Lot auf die Verbindungsstelle (nicht zu viel, sonst erhält man dicke Tropfen; jedoch auch nicht zu wenig, da die Verbindung dann möglicherweise nicht ausreichend hält). Nach dem abkühlen macht es Sinn, die Lötverbindung durch leichtes rütteln oder ziehen zu überprüfen. Löten ist letztendlich aber ein Handwerk, das für saubere Ergebnisse Geduld und Übung vorraussetzt.

(vgl. `https://www.elektronik-kompendium.de/sites/grd/0705261.htm`)

4.5.3 Kleben

Damit alle Komponenten und Teile sicher im Gehäuse sitzen und nichts klappert oder sich gar bewegt, worunter die Lötverbindung leiden würde, müssen gewisse Komponenten angeklebt werden. Vorallem bei den Buchsen ist eine feste Verbindung wichtig, da diese bei jedem Ein- und Ausstecken großer Belastung ausgesetzt sind. Somit wurden der Laderegler und der Digital-/Analogwandler an den dafür gedruckten Stützen angeklebt. Zudem wurde der gedruckte Aufsatz in Gehäuseoptik für den Taster auf diesem befestigt. Für alle Klebverbindungen im Adapter wurden entweder herkömmliches Cyanacrylat (Superkleber) oder Schmelzklebstoff (Heißkleber) verwendet.

5 Testen und Fehlerbehebung

5.1 Testen des Gesamtsystems

5.1.1 Testen des Prototypen

Es gibt unzählbar viele Möglichkeiten einen Prototypen auf Herz und Nieren zu testen. Diese Diplomarbeit beschränkt sich jedoch auf einige wesentliche Aspekte wie Verarbeitung, Funktion und Useability.

Verarbeitung

Positives:
Die äußere Verarbeitung des Multi Room Sound-Adapters ist für einen Prototypen sehr gut ausgefallen. Fertig zusammengesetzt wirkt das Gerät stabil und wertig. Es hat etwas Gewicht und nichts klappert wenn man es schüttelt. Die Buchsen halten der für Buchsen gemäßen Belastung stand. Beide der Buchsen lassen sich problemlos benutzen, beim Ein- und Ausstecken gibt es keine Probleme. Der Taster des Adapters ist, zumindest unseren Erwartungen nach, besonders gut ausgefallen, er hat ein angenehmes Klickfeeling und lässt sich reibungslos betätigen. Die Statusanzeige funktioniert so wie sie soll.

Negatives:
Man sieht dem Gehäuse bei guter Beleuchtung eindeutig an, dass es 3D-gedruckt wurde. Manche Oberflächen wirken eher geriffelt als flach. Diese Oberflächeneigenschafen sind völlig normal für

3D-Drucker und beeinflussen die Funktionsweise des Gesamtsystems kaum. Bei einem tatsächlichen Vetrieb des Produkts müsste man sich jedoch noch einmal genau über Herstellungsmöglichkeiten für kleinere Gehäuse informieren oder das Gehäuse, wie auch bei der Auswahl des Fertigungsverfahrens schon kurz diskutiert, einfach Spritzgießen lassen. Ein Spritzguss hat bessere Oberflächeneigenschaften als ein 3D-Druck.

(vgl. `https://2d-spritzguss.de/3d-druck-spritzguss`)

Funktion
Der Adapter funktioniert und erfüllt seine Funktionen ohne Fehler oder Komplikationen.

Usability
Die Anwendung des Adapter ist grundsätzlich angenehm. Es gilt jedoch zu bedenken, dass uns sowohl Software als auch Hardware schon bekannt sind. Wie sich die Usability verändert, wenn der Adapter von jemandem benutzt wird, der ihn noch nie gesehen hat, kann man nur schlecht sagen. Dafür wären mehrere Produkttests nötig. Falls ein Vetrieb des Produkts in betracht gezogen werden würde, wäre dies auf jeden Fall ein wichtiger Punkt.

5.2 Auftretende Fehler beheben

Es handelt sich hierbei nicht direkt um einen Fehler, jedoch ist uns wichtig, dass auch die Optik des Gehäuse so gut wie möglich ausfällt. Deshalb wurde viel mit verschiedenen Tricks und Methoden gearbeitet, die 3D-Drucke wertiger ausfallen zu lassen.
So war anfangs beispielsweise die Druckgenauigkeit in Millimeter nicht auf dem kleinst-möglichen Wert, was für Testdrucks aufgrund der wesentlich kürzeren Druckzeit jedoch völlig in Ordnung ist. Wenn alles passt, macht es aber definitiv Sinn eine lange Druckdauer in Kauf zu nehmen, um das für den vorliegenden 3D-Drucker genauste Ergebnis zu erhalten.
Eine Herausforderung war die Beschriftung des Deckels. Diese war beim ersten Testdruck teilweise unlesbar ungenau. Das der „Boden“ der Schrift auf Grund des druckabhängigen Überhanges (die größte Fläche also die Oberseite des Deckels muss praktisch auf der Druckplatte sein) nicht glatt wird, war uns klar. Jedoch hatte dieses Erscheinungsbild augenscheinlich nichts mit dem Überhang zu tun. Bei genauerem Betrachten fiel auf, dass das erste Layer fast wie gepatzt aussah. Ein weiterer Versuch mit simpler Schriftart und bei dem lediglich die Druckgenauigkeit des ersten Layers von 0.15 auf 0.1, die der restlichen Layer von 0.2 auf 0.15 kalibriert wurde, zeigte jedoch schon für einen 3D-Druck schöne Ergebnisse.
Eine weitere Möglichkeit zum glätten der schon zuvor beschriebenen riffelartigen Oberflächen ist sogenanntes Ironing. „Ironing beschreibt die einstellbare Funktion in einiger Slicer-Software. Ironing bietet dir die Möglichkeit, die letzte 3D-Druckschicht mit der Nozzle zu "bügeln". Die Nozzle deines 3D-Druckers bewegt sich ohne dabei zu extrudieren nochmals über die zuletzt gefertigte Schicht. Durch diesen Vorgang glättet sie die Objekt-Oberfläche. Du bekommst somit eine verbesserte und

glatte letzten Schicht.“ Beim PRUSA MK4S war diese Funktion sehr zufriedenstellend. Oberflächen werden, ohne den gesamten Druck zu beeinflussen, glatter.

(vgl. `https://www.3dprima.com/de/3dprima/tipps-tricks-begriffe-3d-druck#Ironing`)

6 Schluss

6.1 Konklusion Hardware

Ziel des Hardware-Teils in dieser Diplomarbeit war es, passende Komponenten zu finden und diese in einem selbst designten Gehäuse zu einem Prototypen des Multi-Room Audio Adapter zusammenzusetzen. Dies wurde, wie geplant, umgesetzt. Dabei kamen grundlegende Methoden wie 3D-Design, 3D-Druck, Löten, Online-Recherche und weitere zum Einsatz. Grundsätzlich lief alles wie geplant, es ergaben sich jedoch öfters größere Herausforderungen dort, wo man eigentlich keine erwartet. Für diese fand sich aber immer eine passende Lösung.

6.2 Konklusion Software

7 Einzelnachweise

7.1 Literaturverzeichnis

Literatur

Atlassian (2025). *URLPI17*. de. URL: https://www.atlassian.com/de/git/tutorials/what-is-git (besucht am 06. 01. 2025).

Santos, Sara (Nov. 2018). *URLPI27*. en-US. URL: https://randomnerdtutorials.com/esp32-flash-memory/ (besucht am 29. 01. 2025).

URLPI01 (Mai 2020). de. URL: https://www.ionos.de/digitalguide/hosting/hosting-technik/http-request-erklaert/ (besucht am 23. 09. 2024).

Was ist eine REST-API? (2024). *URLPI04*. de. URL: https://www.redhat.com/de/topics/api/what-is-a-rest-api (besucht am 23. 09. 2024).

What Is React Native? (2024). *URLPI07*. en. URL: https://www.netguru.com/glossary/react-native (besucht am 30. 10. 2024).

URLPI08 (2024). en. URL: https://docs.expo.dev/get-started/introduction/ (besucht am 30. 10. 2024).

Was ist ein Access Point? (2024). *URLPI10*. de. Publication Title: techbold. URL: https://www.techbold.at/lexikon-eintrag/access-point (besucht am 16. 12. 2024).

URLPI11 (2024). de. URL: https://www.oracle.com/at/database/what-is-json/ (besucht am 16. 12. 2024).

URLPI12 (2024). URL: https://www.mikrocontroller.net/articles/I2S (besucht am 16. 12. 2024).

URLPI13 (2024). URL: https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/HTTP-methods (besucht am 17. 12. 2024).

URLPI14 (2024). de. Section: Design Pattern. URL: https://www.geeksforgeeks.org/singleton-design-pattern/ (besucht am 17. 12. 2024).

URLPI15 (2024). URL: https://en.cppreference.com/w/cpp/language/enum (besucht am 17. 12. 2024).

URLPI16 (2025). de. URL: https://www.informatik-verstehen.de/lexikon/protokoll/ (besucht am 05. 01. 2025).

Was ist GitHub? (2025). *URLPI18*. de. URL: https://kinsta.com/de/wissensdatenbank/was-ist-github/ (besucht am 06. 01. 2025).

Multicast DNS (Feb. 2022). *URLPI19*. de. URL: https://www.ionos.at/digitalguide/server/knowhow/multicast-dns/ (besucht am 06. 01. 2025).

URLPI20 (2025). de. URL: https://zid.univie.ac.at/dhcp/ (besucht am 06. 01. 2025).

URLPI21 (2025). URL: https://www.radio-browser.info/ (besucht am 28. 01. 2025).

URLPI22 (2025). de. URL: https://www.it-intouch.de/glossar/firebase/ (besucht am 28. 01. 2025).

URLPI23 (2025). en. URL: https://de.legacy.reactjs.org/docs/context.html (besucht am 28. 01. 2025).

URLPI24 (2025). URL: https://docs.espressif.com/projects/arduino-esp32/en/latest/api/wifi.html (besucht am 29. 01. 2025).

URLPI25 (Sep. 2021). de. URL: https://www.mittwald.de/blog/hosting/was-ist-eigentlich-node-js (besucht am 29. 01. 2025).

URLPI26 (2025). URL: https://docs.platformio.org/en/latest/what-is-platformio.html (besucht am 29. 01. 2025).

URLPI28 (2025). en. URL: https://docs.expo.dev/router/introduction/ (besucht am 29. 01. 2025).

7.2 Abbildungsverzeichnis

Abbildungsverzeichnis

7.3 Anhang

7.3.1 Code Microcontroller

7.3.2 Code Smartphoneapp