



AudiTim Dokumentation

SE-Projekt
des Studiengangs
Allgemeine Informatik

an der Dualen Hochschule Baden-Württemberg, Campus Heidenheim

Diese Dokumentation entstand im Rahmen des Software-Engineering-Projekts an der Dualen Hochschule Baden-Württemberg. Sie beschreibt die Konzeption und Umsetzung von AudiTim, einem verteilten System zur akustischen Raumüberwachung, dessen Ziel die Erfassung, Speicherung und Visualisierung von Lautstärkedaten in einem Vorlesungsraum ist.

Bearbeitungszeitraum

Gruppe, Kurs

Gruppenmitglieder

10 Wochen

5, INF2023AI

Aaron Reiber, Jan-David Oberländer,
Luca Müller, Moritz Flaig

Inhaltsverzeichnis

1	Einleitung	1
1.1	Projektüberblick	1
1.2	Ziele und Motivation	1
1.3	MVP und Qualitätsmerkmale	2
2	SFMEA-Analyse	4
2.1	Analyse	4
2.2	Auswertung	6
3	Hardwarekomponenten	8
3.1	Mikrofonmodule	8
3.2	ESP32	10
3.3	Umsetzung	10
4	Softwarekomponenten	11
4.1	Entwicklungsumgebungen	11
4.2	Kommunikationsmethode	11
4.3	Frontend	12
4.4	Backend	17
4.5	MQTT Subscriber	19
5	Befestigung der Hardware	21
5.1	Platzungskonzept	21
5.2	Deckenhalterung	22
5.3	ESP32-Halterung	22
5.4	Sensor-Halterung	24
5.5	3D-Raummodellierung	24
6	Algorithmus zur Schallquellenlokalisierung	27
6.1	Grundprinzipien	27
6.2	Betrachtete Verfahren	27
6.3	Vergleich der Verfahren	33
6.4	Fazit	33

7	API-Kommunikation	35
7.1	Problemstellung	35
7.2	Datenbasis	35
7.3	Optimierungsoptionen	36
7.4	Vergleich der Ansätze	37
	Literaturverzeichnis	i

1 Einleitung

Im Rahmen des Software-Engineering-Projekts wird in einem Team ein System zur akustischen Raumüberwachung entwickelt. Ziel des Projekts ist die kontinuierliche Erfassung und spätere Analyse der Lautstärkeverteilung in einem typischen Vorlesungsraum über den Zeitraum mehrerer Tage hinweg.

1.1 Projektüberblick

Die Umsetzung erfolgt nach agilen Prinzipien unter Anwendung von Scrum, wobei die jeweiligen Sprints von Freitag bis Freitag festgelegt wurden.

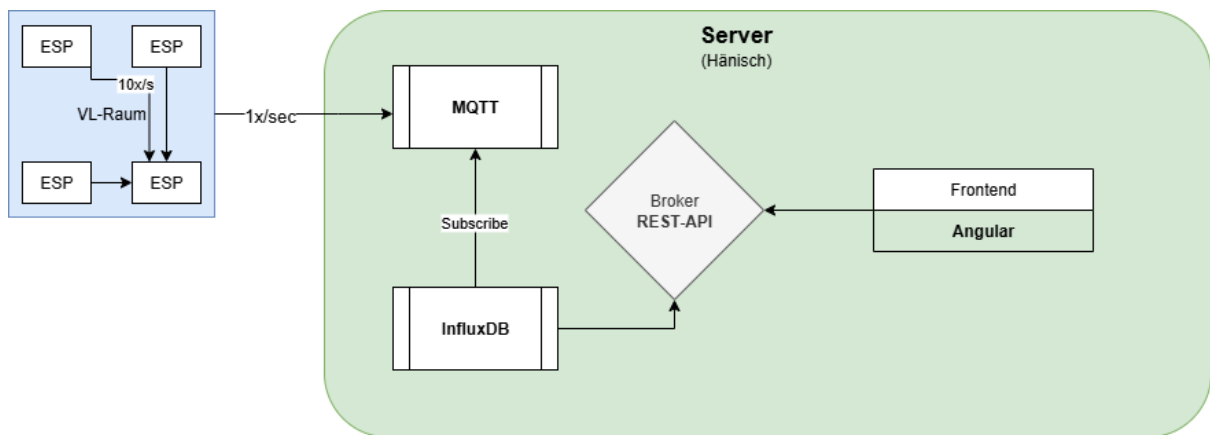
Die technische Architektur sieht den Einsatz von vier Mikrocontroller-Einheiten (ESP32) vor, die jeweils mit einem Mikrofon oder einem Dezibel-Messsensor ausgestattet werden. Die Geräte werden in den vier Ecken des Vorlesungsraums positioniert, um eine möglichst flächendeckende Abdeckung des Raumes zu gewährleisten. Über ein Kommunikationsprotokoll tauschen sich die ESPs untereinander aus und senden die gesammelten Messdaten gemeinsam an ein zentrales Backend.

1.2 Ziele und Motivation

Ziel ist es, ein skalierbares und robustes System zu entwickeln, das Lautstärkepegel über einen längeren Zeitraum erfassen kann. Die gesammelten Daten sollen im Backend gespeichert und im Nachgang analysiert werden. Hierbei sind verschiedene Darstellungsformen wie etwa Heatmaps oder zeitbasierte Diagramme denkbar, um Muster oder Auffälligkeiten im akustischen Verhalten des Raumes sichtbar zu machen.

1.3 MVP und Qualitätsmerkmale

Im Rahmen des MVPs soll ein verteiltes System zur akustischen Raumüberwachung auf Basis von vier Mikrocontrollern der ESP-Serie entwickelt werden. Jeder ESP wird über ein extern angeschlossenes Mikrofon verfügen, mit dem kontinuierlich Lautstärke-daten erfasst werden. Drei der Module sollen ihre Messwerte mithilfe des ESP-NOW-Protokolls an eine zentrale Einheit übermitteln, die zusätzlich eigene Daten erfasst. Die aggregierten Informationen aller vier Einheiten werden in festen Intervallen über das MQTT-Protokoll an einen Server gesendet und dort zeitsynchron in einer InfluxDB gespeichert. Über eine REST-API sollen die Daten einem Angular-basierten Web-Frontend zur Visualisierung der akustischen Messwerte zur Verfügung gestellt werden.



Auf Grundlage des MVPs sollen die folgenden Qualitätsmerkmale sichergestellt werden:

- **Echtheit & Qualität der Daten:** Nur qualitative Daten z.B. durch manuelle Überprüfung und Kalibrierung der Mikrofone ermöglichen realistische Messwerte und damit eine sinnvolle Analyse der akustischen Verhältnisse im Raum.
- **Accountability der Daten:** Es muss nachvollziehbar sein, welche Sensoren zu welchem Zeitpunkt welche Daten erfasst haben. Diese Anforderung soll durch das Einführen einer eindeutigen ID für jeden Sensor, die mit jedem Value mitgeschickt wird, umgesetzt werden.
- **Verfügbarkeit:** Da die Übertragungswege mit ESP-NOW und MQTT unterbrochen werden könnten ist es wichtig die Verfügbarkeit von Daten sicherzustellen indem ein möglicher Ausfall schnellstmöglich erkannt wird um diesen dann

beheben zu können. Dies soll sichergestellt werden durch die Einführung einer Notification, die ausgelöst wird beim Ausfall eines Gerätes, was final über einen eigenen Discord-Server bereitgestellt werden soll.

2 SFMEA-Analyse

2.1 Analyse

Systemfunktion	Fehlermöglichkeit	Fehlerursache	Fehlerfolge	B	A	E	RPZ	Maßnahmen
Dezibel erfassen	Falsche oder keine Messwerte	Mikrofon defekt, Sensor ungeeignet	Ungenaue oder keine Daten	8	6	3	144	Robuste Mikros, Kalibrierung, Ersatzsensor bereit
ESP kommuniziert nicht mit anderen	Keine oder fehlerhafte Datenübertragung	WLAN-Probleme, falsches Protokoll	Daten gehen verloren, keine Heatmap	7	5	4	140	Netzwerk prüfen, Fallback, Fehler-Logs
Daten werden nicht ans Backend gesendet	ESP schickt keine Daten	Stromausfall, Code-crash, Timeout	Datenlücken, unvollständige Analyse	7	5	3	105	Watchdog, Logging, häufigere Syncs
GitHub-Zugang fehlerhaft	Kein Zugriff oder Merge-Konflikte	Rechte falsch, keine Git-Strategie	Team kann nicht arbeiten	6	4	2	48	Git-Workflow, Rechte regeln

Systemfunktion	Fehlermöglichkeit	Fehlerursache	Fehlerfolge	B	A	E	RPZ	Maßnahmen
Backend speichert keine Daten	Fehlerhafte Speicherung / Absturz	Server voll, Codefehler	Datenverlust, Ausfall	9	4	4	144	Monitoring, Logging, Backups, Testlauf
Daten werden falsch ausgewertet	Heatmap falsch / Werte inkonsistent	Algorithmusfehler, unvollständige Daten	Falsche Rückschlüsse	6	4	3	72	Testdaten prüfen, Algo validieren, Visualisierung testen
ESPs messen nicht synchron	Unterschiedliche Messzeitpunkte	Kein Sync, NTP fehlt	Daten nicht vergleichbar	7	5	4	140	NTP-Sync, Zeitstempel ergänzen

2.2 Auswertung

- **Dezibel erfassen:** Während der tatsächlichen Arbeit mit den Sensoren/Mikrofonen trat ein solcher Fehler selten auf, dennoch wurde zur Absicherung beim Ausfall der Sensoren eine Discord-Notifikation hinzugefügt. Deshalb ist die RPZ von 144 grundsätzlich zu hoch.
- **ESP kommuniziert nicht mit anderen:** Auch hierbei hilft die bereits genannte Discord-Notifikation. Zusätzlich wurden mehrere Datenübertragungs-Möglichkeiten implementiert und getestet, wobei sich ESP-Now mit einem zusätzlichen Edge-Device, ohne Sensorfunktion, als die beste Option zeigte. Hierdurch wurde das Risiko bestmöglich minimiert.
- **Daten werden nicht ans Backend gesendet:** Wenn Daten nicht am Backend angekommen sind, lag dies meistens an Verbindungsproblemen zwischen den einzelnen Komponenten. Stromausfälle oder Codecrashes waren hierbei quasi nie die Ursache.
- **GitHub-Zugang fehlerhaft:** Mit dem Github-Zugang entstanden keine Probleme, jedoch kam es gelegentlich zu Merge-Konflikten, sowie zu kleineren Problemen bei der Zusammenführen der anfangs getrennten Repositories. Daher ist die klein ausfallende RPZ gerechtfertigt.
- **Backend speichert keine Daten:** Die hoch angesetzte RPZ wurde durch die Serverprobleme mit dem "HänischServer und die allgemeine Datenspeicherstruktur des Projekts bestätigt.
- **Daten werden falsch ausgewertet:** Durch die anfängliche Annahme, dass es vorgefertigte Algorithmen/Bibliotheken für unseren Usecase gäbe, wurde die RPZ zu niedrig angesetzt. Da es diese nicht gab, musste der Algorithmus selbst entwickelt werden, was zu einer nicht erwarteten Komplexität führte. Daher ist die RPZ bei mindestens 140 anzusetzen.
- **ESPs messen nicht synchron:** Durch das gebündelte Senden der Daten über das Edge-Device und das vereinheitlichen der Daten innerhalb einer Sekunde, entfällt das zwingend genaue Synchronisieren der Messdaten.

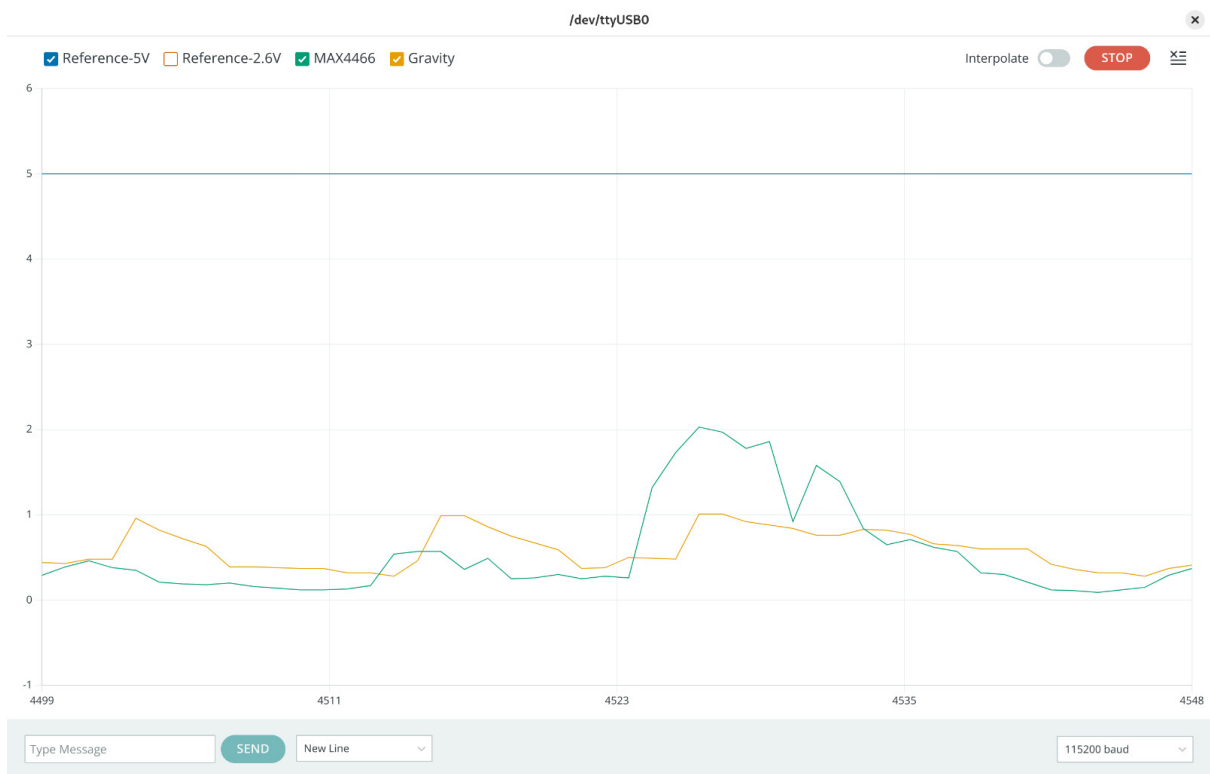
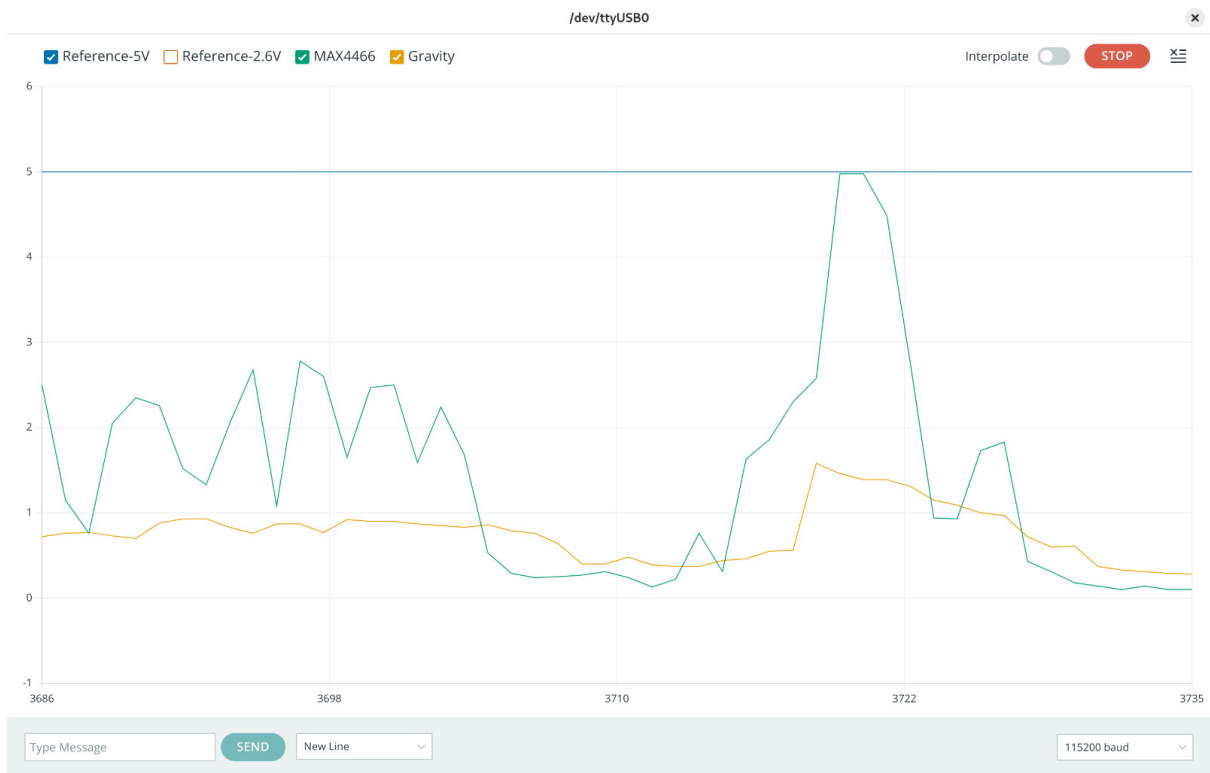
- **Nicht beachtete Störquellen:** Ein unerwartetes Problem stellen Störsignale dar, etwa durch das gleichzeitige Senden des Edge-Devices während eigener Sensoraufnahmen, oder durch "laute" Netzteile anderer Gruppen. Dieses Problem wurde erst spät erkannt und hat sehr viel Zeit in Anspruch genommen, weshalb eine RPZ von 140 angebracht wäre.

3 Hardwarekomponenten

3.1 Mikrofonmodule

Zur Erfassung des Lautstärkepegels wurden drei unterschiedliche Mikrofonmodule in Betracht gezogen. Der erste Sensor ist ein kostengünstiges Modul "GY-MAX4466"(ca. 4€), das ursprünglich als Klatschsensor konzipiert wurde. Der zweite Sensor ist das *Sound Level Meter V2.0* von DFRobot (ca. 40€), das über zusätzliche Hardware zur Signalverarbeitung verfügt und den Schalldruckpegel bereits gefiltert und bereinigt im dBA-Format ausgibt. Der dritte in Betracht gezogene Sensor ist ein handelsübliches DB-Messgerät (ca. 20 €), das jedoch nicht für die Integration in das Projekt geeignet ist, da es keine digitale Schnittstelle bietet und somit nicht direkt mit dem ESP32 kommunizieren kann. Dieser Sensor wird daher zur Überprüfung der Ergebnisse verwendet, ist aber nicht für die direkte Integration in das Projekt vorgesehen.

Da die absolute Einheit (dBA) für die Auswertung im Rahmen eines 3D-Diagramms nicht zwingend erforderlich ist, wurde der Fokus auf die Vergleichbarkeit der analogen Ausgangssignale gelegt. Beide Sensoren wurden gleichzeitig an einem ESP32 betrieben und ihre Ausgangssignale mithilfe des *Serial Plotter* in Echtzeit visualisiert. Dabei zeigte sich, dass die Kurvenverläufe bei normalen Sprachgeräuschen nahezu identisch sind. Lediglich bei impulsartigen Geräuschen mit sehr geringer Distanz (z. B. Klatschen oder Klopfen) erzeugt das günstigere Mikrofon deutlich stärkere Peaks, während diese durch den DFRobot-Sensor wirksam herausgefiltert werden.



Hier sind die Sensorwerte des günstigeren Klatschsensors (grün), sowie die des Teureren DbA sensors (gelb) dargestellt. Die In Blau dargestellte 5v Referenzlinie stellt den Maximalwert da, welcher von den Sensoren zurückgegeben werden kann.

Da die in den Vergleichsdiagrammen dargestellten Abweichungen bei größerem Abstand deutlich abnehmen und im Anwendungsfall als vernachlässigbar eingestuft werden können, wurde entschieden, den günstigeren Sensor zu verwenden. Um mögliche Reststörungen dennoch weiter zu reduzieren, ist geplant, eine mechanische Dämpfung oder Abschirmung am Sensor zu testen.

Insgesamt konnte festgestellt werden, dass der günstigere Sensor trotz fehlender dBA-Ausgabe für die angestrebte Visualisierung ausreichend präzise Ergebnisse liefert und somit für den weiteren Projektverlauf verwendet wird.

3.2 ESP32

Die ESP-Module bieten im Vergleich zu klassischen Arduinos eine integrierte WLAN-Funktionalität und unterstützen direkt das ESP-NOW-Protokoll. Dadurch entfällt zusätzlicher Hardware- oder Softwareaufwand für die Netzwerkkommunikation. In Kombination mit ESP-NOW ergibt sich somit eine kompakte, performante und zuverlässige Lösung.

3.3 Umsetzung

Nach der Entscheidung für die finalen Hardwarekomponenten wurde dies erfolgreich nach Plan umgesetzt. Da jedoch das IOT-eigene Netzwerk nicht bis zum letzten ESP im Raum reichte, wurden mehrere Möglichkeiten getestet (WLAN-Repeater, lokaler Access-Point auf Edge-ESP, ESP-Now Brücke). Final wurde sich für einen fünften ESP entschieden, der nur als Edge-Device fungiert (zuvor war das Edge-Device als Sensor genutzt worden) und die Daten der anderen vier ESPs sammelt und an den MQTT-Server sendet. Damit waren keine Störgeräusche mehr in den Daten zu finden und die Verbindung war stabil.

4 Softwarekomponenten

4.1 Entwicklungsumgebungen

Zu Beginn des Projekts wurden zwei gängige Entwicklungsumgebungen für die Programmierung von Mikrocontrollern evaluiert: die klassische *Arduino IDE*, welche auch von den betreuenden Dozierenden empfohlen wurde, sowie die moderne, zunehmend verbreitete *PlatformIO*-Erweiterung für Visual Studio Code. Im Projektteam wird plattformübergreifend mit Linux, Windows 10 und Windows 11 gearbeitet. Beide Entwicklungsumgebungen sind grundsätzlich mit allen eingesetzten Betriebssystemen kompatibel. Zur Prüfung der technischen Umsetzbarkeit wurden zwei separate Repositories eingerichtet, in denen jeweils einfache Testprojekte zur Ansteuerung eines ESP32 implementiert wurden. Beide IDEs konnten den Mikrocontroller erfolgreich ansprechen. PlatformIO bietet durch seine deklarative Verwaltung von Bibliotheken und Abhängigkeiten Vorteile in Bezug auf Reproduzierbarkeit und Build-Konsistenz. In der Praxis kam es jedoch auf einem System im Team zu anhaltenden technischen Problemen mit PlatformIO, die nicht zufriedenstellend gelöst werden konnten. Um eine einheitliche und zuverlässige Entwicklungsgrundlage für alle Teammitglieder sicherzustellen, fiel die Entscheidung letztlich zugunsten der *Arduino IDE*. Diese ermöglicht eine barrierefreie Mitarbeit aller Beteiligten, auch wenn sie im Funktionsumfang nicht ganz an PlatformIO heranreicht.

4.2 Kommunikationsmethode

ESP-NOW wurde ausgewählt, da es im Vergleich zu anderen Protokollen einen deutlich geringeren Overhead besitzt und dadurch eine schnellere und effizientere Datenübertragung ermöglicht. Die Struktur ist weniger komplex, was die Implementierung vereinfacht und die Fehleranfälligkeit reduziert. Dadurch wird das System insgesamt

robuster und zuverlässiger: ESP-Now ist ein Kommunikationsprotokoll von Espressif, das speziell für die ESP32- und ESP8266-Chips entwickelt wurde. Es verwendet die 2,4 Ghz Frequenz, welche eigentlich für WiFi Kommunikation verwendet wird. Im Gegensatz zu anderen Protokollen sind die Nachrichten auf Schicht 2 des ISO-OSI-Modells. Keine IP-Adressen sondern nur MAC-Adressen. Die ESPs müssen sich dafür im gleichen Kanal befinden. Die 2,4 Ghz Frequenz hat grundsätzlich nur 3 Nutzbare Kanäle. Der zu verwendene Kanal muss von dem ÄI401"Wlan Netz übernommen werden. Es ist mit relativ wenig Code möglich eine große Menge an Daten in sehr Kurzen Zeitabschnitten zu senden. Theoretisch ist eine Datenübertragung von 1 bis 2 Mbps möglich. Die Nachrichtenlänge ist auf 250 Byte pro Paket beschränkt. Da unsere Nachrichten nur wenige Bit lang sind, haben wir hier keine Probleme. Das Protokoll hat keine festgelegten Sender/Empfänger Rollen. Jeder ESP kann senden und empfangen. In unserem Beispiel senden aber 3 ESPs und ein Edge-Device empfängt alle Daten. Es wird 10 mal pro Sekunde der Lautstärkepegel erfasst und gesendet.

4.3 Frontend

4.3.1 Technologieauswahl

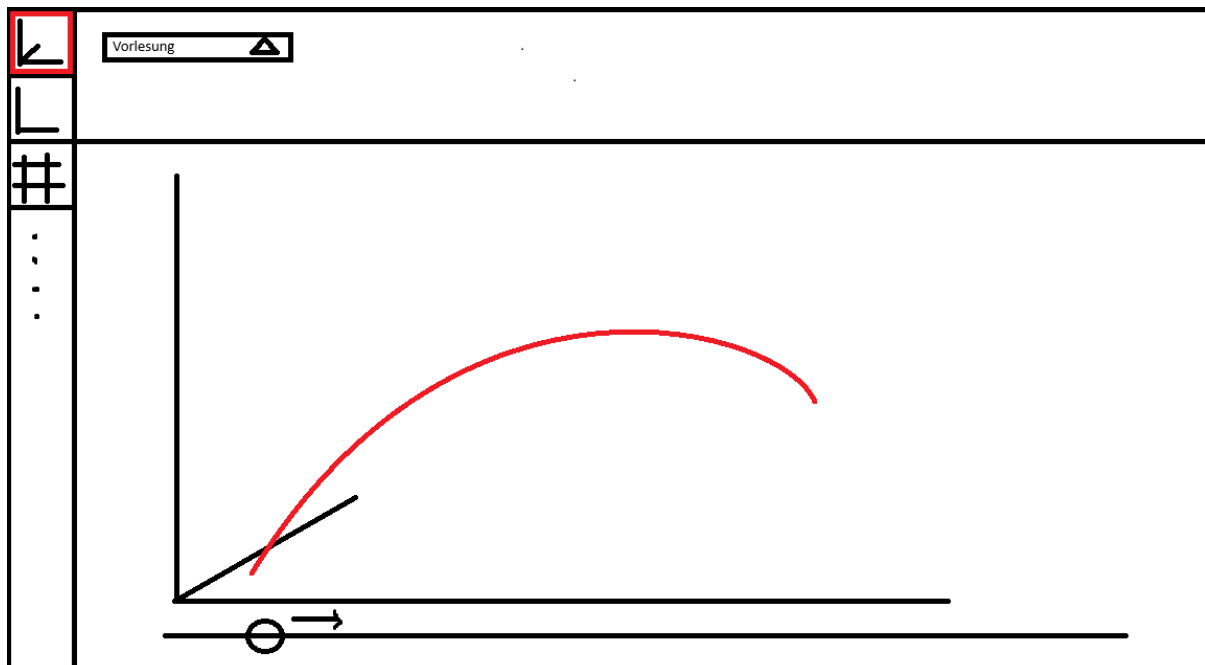
Für die Umsetzung des Frontends wurde das Framework *Angular* gewählt. Diese Entscheidung basiert auf technischen Faktoren und persönlichen Präferenzen. Angular bietet eine klare Struktur durch komponentenbasiertes Design und ist für die moderne Webentwicklung gut geeignet. Es erleichtert die Wiederverwendbarkeit von eigenen aber auch externen UI-Elementen und unterstützt eine saubere Trennung von Logik und Darstellung.

Im Vergleich dazu ist *React* ein von Meta entwickeltes UI-Framework. *Angular* und *React* haben beide eine komponentenbasierte Architektur. Diese Komponenten sind innerhalb anderer Komponenten wiederverwendbar und somit endlos wiederverwertbar. Beide sind Open Source. Daher haben React und Angular große Entwicklergemeinschaften, die die Ressourcen regelmäßig erweitern. Angular ist ein vollwertiges Framework, während React eine JavaScript-Bibliothek ist. Deshalb muss React mit einem Framework gepaart werden, um eine schnelle, schöne und kompatible Benutzeroberfläche zu erstellen. (Powell, 2023)

Da außerdem im Team bereits umfangreiche Erfahrung mit diesem Framework vorhanden war, fiel die Entscheidung zugunsten von Angular. Das Frontend läuft in einem eigenen Docker-Container und kommuniziert, über ein internes Netzwerk, mit dem separat laufenden Backend-Container. So bleibt die Architektur modular, skalierbar und einfach wartbar.

4.3.2 Umsetzungsplan

Während der Entwicklung wurde ein einfaches Frontend zu Testzwecken angelegt. Deshalb soll nun ein Plan entwickelt werden, wie die Benutzeroberfläche aussehen soll und welche Funktionen sie bieten soll:

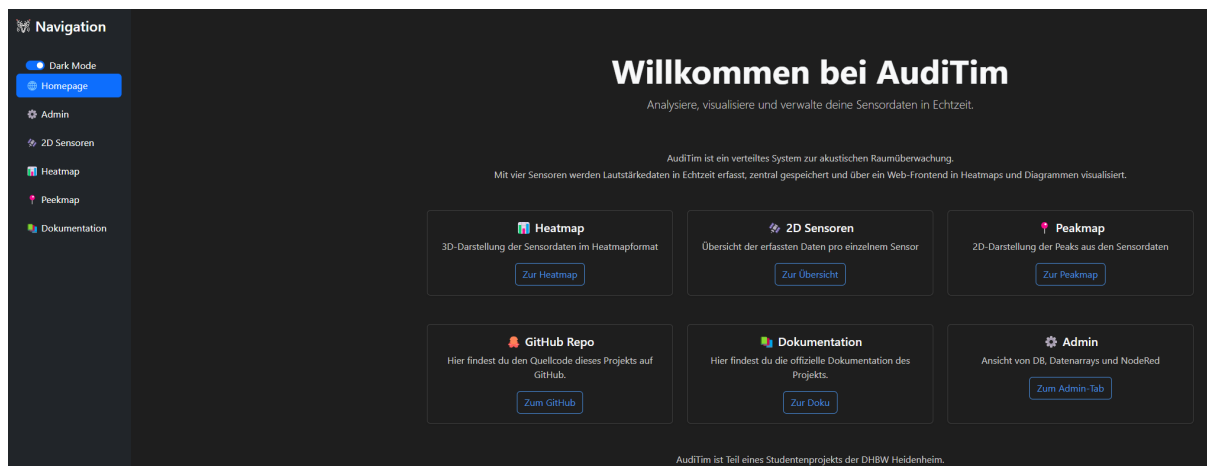


In dieser Skizze wurden die Grundfunktionen der Benutzeroberfläche festgelegt und die Anordnung der einzelnen Elemente skizziert. Über eine Navbar an der linken Seite kann zwischen den einzelnen Seiten gewechselt werden. Auf der Hauptseite wird eine Heatmap angezeigt, die die Lautstärkeverteilung im Vorlesungsraum räumlich darstellt, wobei mit einem Dropdown-Menü der anzuzueigende Zeitraum ausgewählt werden kann. Durch den ausgewählten Zeitraum kann anschließend mit einem Schieberegler die Zeit ausgewählt werden, für die die Lautstärke angezeigt werden soll. Im nächsten Nav-Item sollen die einzelnen 2D-Grafen der jeweiligen Sensoren angezeigt werden.

Das dritte Nav-Item soll einen tabellarischen Überblick über die gesammelten Daten geben, die in der Datenbank gespeichert sind. Die Nvabar bietet Raum für zukünftige Erweiterungen.

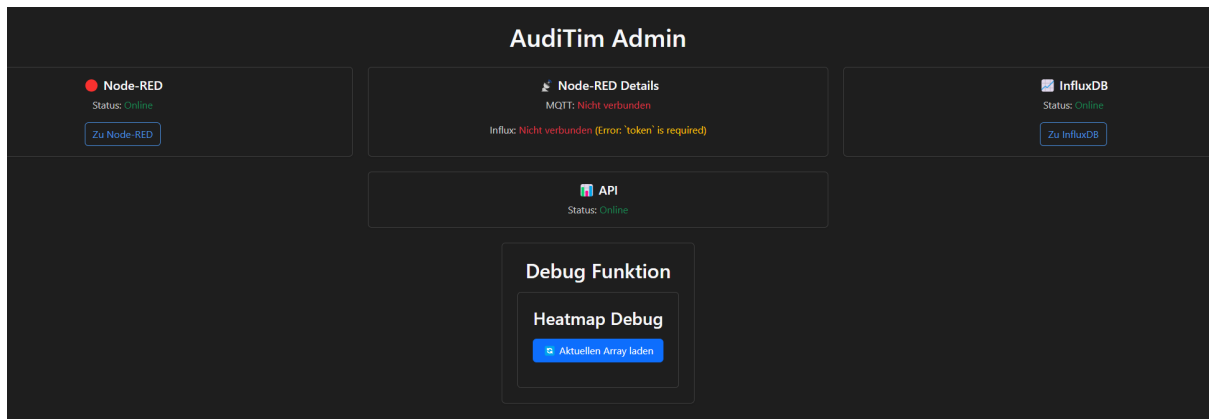
4.3.3 Umsetzung

Auf Grundlage dieser Skizze wurde das Frontend und UI erfolgreich umgesetzt. So hat man nun eine übersichtliche Homepage, die auf alle Elemente verweist, welche auch in der interaktiven Navbar zu finden sind. Als kleines AddOn wurde unter anderem hierfür ein "AudiTim" Logo entworfen, was den ikonischen Hirsch mit dem vernetzten Geweih und den Audiowellen verknüpft, sowie eine Light- und Darkmode Funktionalität implementiert.

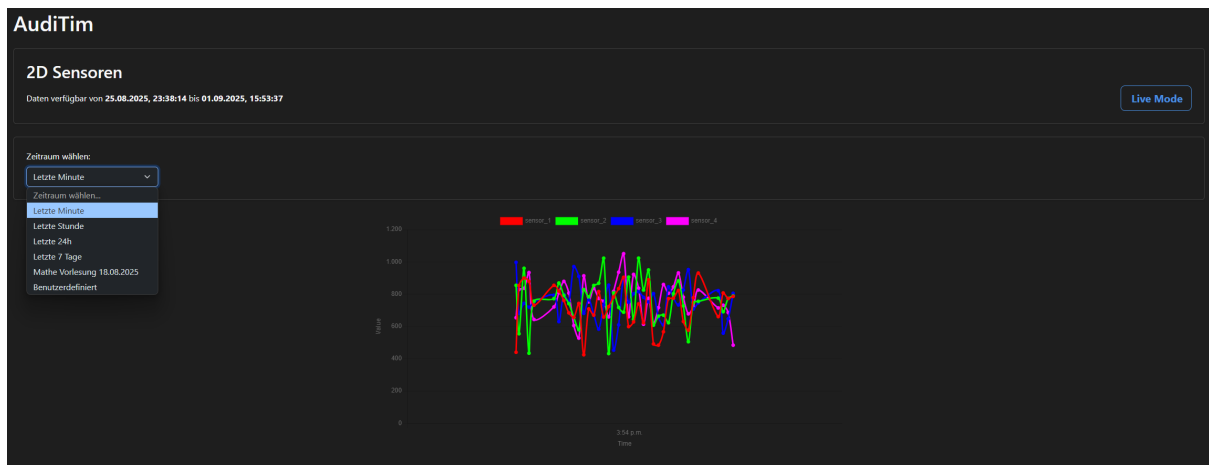


Auch im Screenshot zu sehen, sind die schlussendlich umgesetzten Seiten; Heatmap, 2D-Sensoren, Peakmap, Github Repo, Dokumentation und Admin. Für diese Seiten wurden die in der Skizze angedachten Funktionen erfolgreich umgesetzt und flexibel während der Entwicklung angepasst und erweitert.

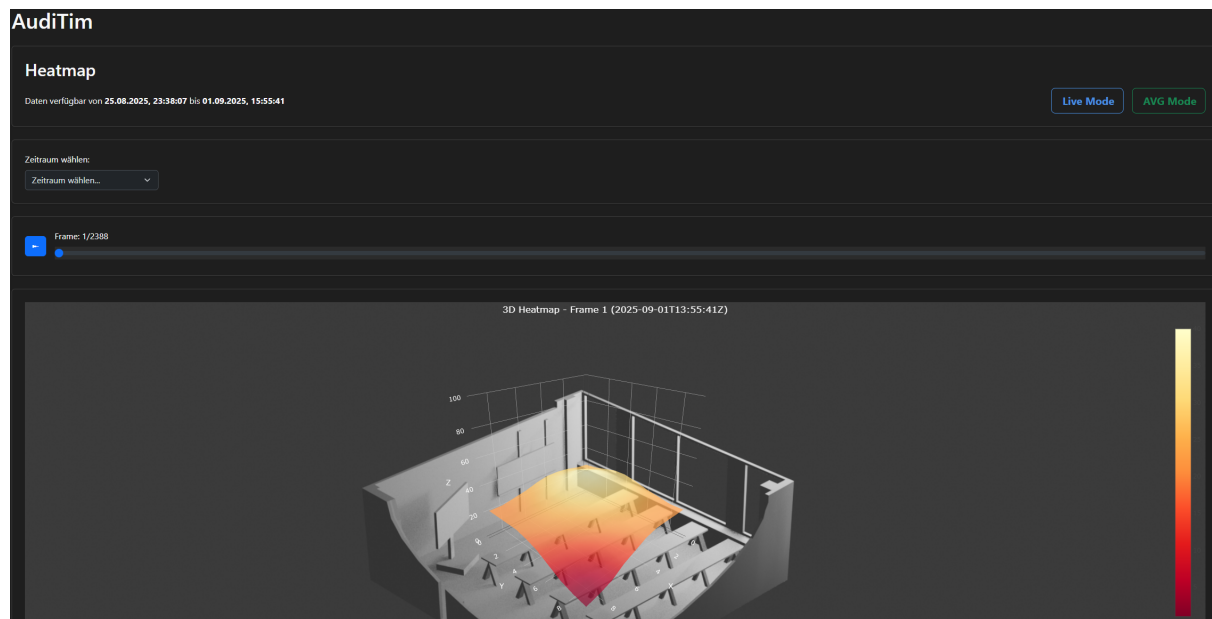
So zeigt die Admin-Seite nun eine Übersicht über die verbundenen Container und Elemente sowie deren Status an:



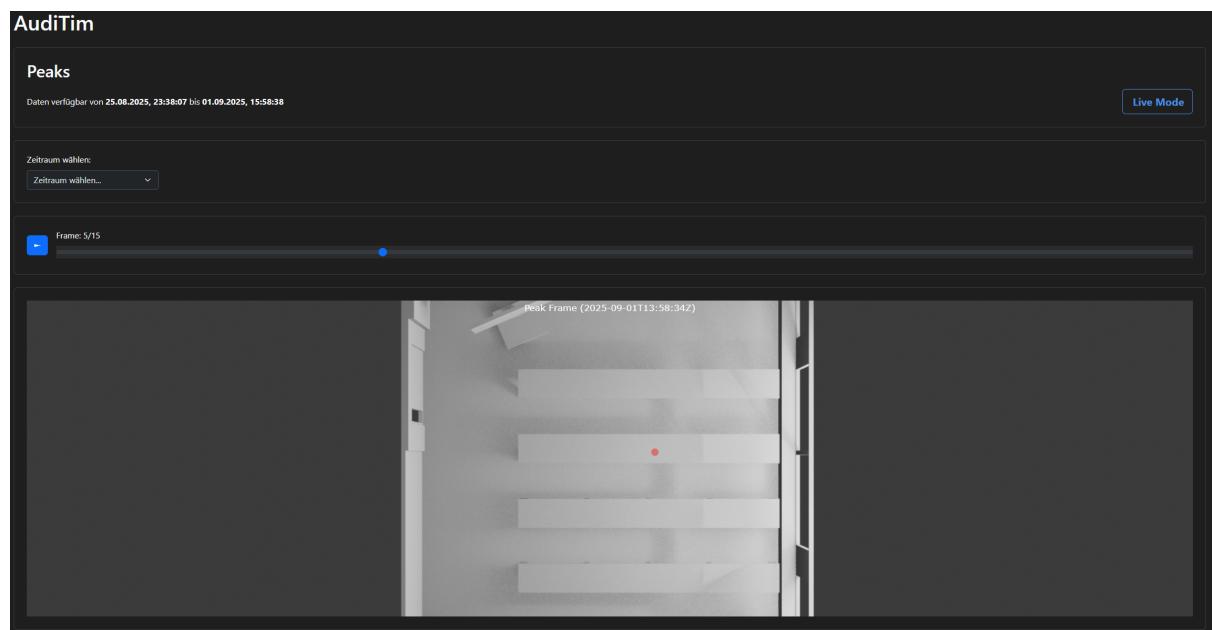
Die 2D-Sensoren-Seite zeigt die einzelnen Sensorwerte in übersichtlichen Liniendiagrammen an, wobei man zwischen einem ausgewählten Zeitraum und einem Live-Modus wechseln kann:



Auch die Heatmap-Seite zeigt die Auswahl zwischen einem ausgewählten Zeitraum und einem Live-Modus, sowie einen Average-Modus, in dem die durchschnittliche Lautstärke über den ausgewählten Zeitraum angezeigt wird. Mit einem Schieberegler kann die Zeit innerhalb des ausgewählten Zeitraums angepasst werden und gegebenenfalls eine Animation gestartet werden, die die Lautstärkeverteilung über den Zeitraum hinweg anzeigt. Außerdem wurde das in "4.5 3D-Raummodellierung" beschriebene 3D-Modell des Raumes als Hintergrundbild für die Heatmap verwendet, um eine realistische Darstellung zu ermöglichen:



Dieses 3D-Modell wurde auch für die Peakmap-Seite verwendet, auf der die Spitzenwerte der Sensoren über den ausgewählten Zeitraum angezeigt werden:



Und zu guter Letzt zeigt die Dokumentation-Seite diese Dokumentation, die sie hier gerade lesen, in einem eingebetteten PDF-Viewer an.

4.3.4 Testing

Für die Umsetzung des automatisierten Testings wurde Playwright eingesetzt. Die Auswahl dieses Frameworks erfolgte nach einer vergleichenden Analyse. Dabei wurde Playwright von einem Teammitglied im Rahmen einer Projektarbeit mit einem ähnlichen Anwendungsfall ausführlich evaluiert und als am geeignetsten bewertet. Ausschlaggebend hierfür waren insbesondere die breite Browserunterstützung, die integrierten Funktionen für End-to-End-Tests sowie die hohe Stabilität und Geschwindigkeit der Testausführung.

Durch den Einsatz von Playwright konnte die Konsistenz der Benutzeroberfläche (UI) zuverlässig sichergestellt werden. Die automatisierten Tests ermöglichen es, Interaktionen und Darstellungsvarianten reproduzierbar zu prüfen und so frühzeitig Abweichungen oder Fehler zu erkennen. Auf diese Weise trägt das Testing wesentlich zur Qualitätssicherung und Nachhaltigkeit der entwickelten Lösung bei.

4.4 Backend

Das Backend wurde mit Node.js und dem minimalistischen Framework Express.js umgesetzt. Express ist das am weitesten verbreitete Node.js-Framework für Webserver und APIs. Es bietet eine ausgereifte Middleware-Architektur, die sehr flexibel und modular ist. Im Vergleich zu ähnlich minimalistischen Frameworks wie Koa, das von den gleichen Entwicklern stammt, bietet Express mehr out-of-the-box Funktionalitäten und eine größere Community mit umfangreichen Plugins und Support. (Community, 2025; Appventurez, 2025)

Koa hingegen ist moderner und basiert stärker auf ES6 Features wie `async/await`, was den Code oft sauberer macht. Allerdings ist Koa weniger „batteries included“ und erfordert mehr Initialaufwand und zusätzliche Libraries, um Funktionen bereitzustellen, die Express standardmäßig mitbringt. Für dieses Projekt, welches einen schnellen Einstieg, umfangreiche Dokumentation und stabile Middleware braucht, ist Express daher die pragmatischere Wahl. Zudem sind Gruppenmitglieder bereits mit Express vertraut, was die Einarbeitungszeit verkürzt und die Produktivität steigert.

Die Anbindung an eine InfluxDB erfolgt über die offizielle Client-Bibliothek. Neben dem Abrufen historischer Sensorwerte werden auch Funktionen zum Schreiben von

Dummy-Daten bereitgestellt. Diese werden, solange die Verbindung zu den wirklichen Sensorwerten noch nicht hergestellt ist, zum Testen verwendet.

Für die Speicherung der Sensordaten wurde InfluxDB gewählt. Eine Zeitreihen-Datenbank, die speziell für viele konsekutive Datenströme optimiert ist. InfluxDB bietet eine simple API und gutes visuelles UI, ohne dass viel Konfiguration nötig ist. Im Vergleich zu einer relationalen Datenbank wie TimescaleDB, die auf PostgreSQL basiert, ist InfluxDB deutlich einfacher zu handhaben und benötigt weniger Overhead für die Konfiguration. Um eine TimescaleDB sicher zum laufen zu bringen, sind komplexe Zugriff-Skripte benötigt, um die Benutzer zu verwalten. InfluxDB hingegen arbeitet mit einem einfachen Benutzer- und Rollenkonzept. Auch für die API-Authentifizierung ist InfluxDB mithilfe von Tokens und Secrets deutlich einfacher zu konfigurieren. Der Rationale Aspekt einer traditionellen ist sicherlich in vieler Hinsicht nützlich, wird aber in diesem Projekt nicht benötigt. Eine nahtlosen Integration in das Node.js-Backend ist mit der offiziellen InfluxDB-Bibliothek kein Problem. InfluxDB wurde uns initial von den "Verteilten SystemeDozenten ausdrücklich empfohlen, da wir viele Zeitwerte speichern und analysieren wollen.

Berechnung des Speicherplatzbedarfs für InfluxDB

Zur Abschätzung des benötigten Speicherplatzes in InfluxDB betrachten wir die anfallenden Sensordaten pro Stunde.

Rohdaten Pro Sekunde werden **10 Messungen** von **4 Sensoren** erfasst, wobei jeder Wert als `double` (64 Bit = 8 Byte) gespeichert wird. Damit ergibt sich:

$$10 \times 4 \times 8 \text{ Byte} \times 60 \times 60 = 1,152,000 \text{ Byte} \approx 1.1 \text{ MB/h}$$

Base64-kodierte Daten Wird der Algorithmus auf die Sensordaten angewendet, erzeugt er daraus ein 10x10-Array mit Werten im Bereich von 0 bis 255. Dieses Array wird anschließend kodiert. Durch die Base64-Kodierung entsteht dabei ein Overhead von etwa 36%. Für die Datenmenge ergibt sich pro Sekunde:

$$800 \text{ Bit} = 100 \text{ Byte} \Rightarrow \text{mit Overhead} \approx 136 \text{ Byte}$$

Somit pro Stunde:

$$136 \text{ Byte} \times 60 \times 60 = 489,600 \text{ Byte} \approx 470 \text{ KB/h}$$

Fazit Für die Speicherung in InfluxDB müssen daher pro Stunde etwa

$$1.1 \text{ MB (Rohdaten)} + 0.47 \text{ MB (Base64)} \approx 1.6 \text{ MB/h}$$

eingepplant werden.

Das Backend ist ebenfalls in einem eigenen Docker-Container gekapselt. Die Kommunikation mit dem Frontend erfolgt über definierte HTTP-Endpunkte im Container-Netzwerk. Diese Trennung verbessert die Wartbarkeit und erlaubt eine flexible Skalierung beider Komponenten unabhängig voneinander.

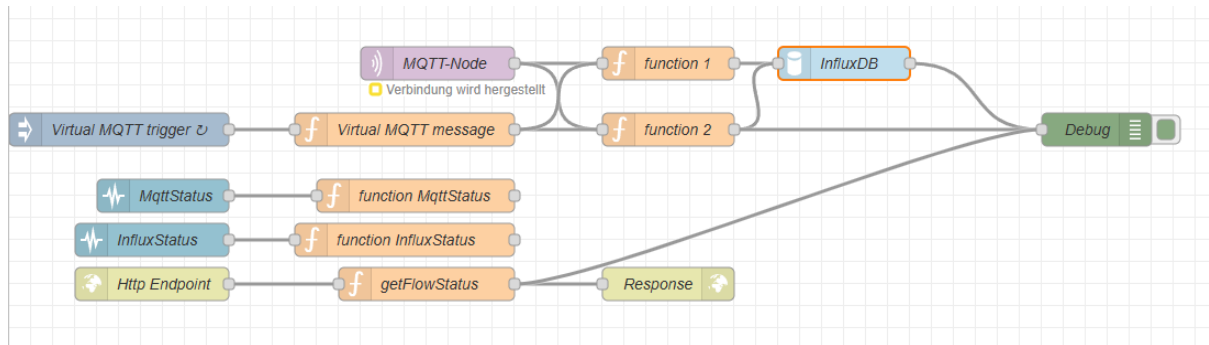
4.5 MQTT Subscriber

Um den Datenfluss zu vervollständigen fehlt eine Brücke zwischen dem MQTT Server und der InfluxDB. Hierfür wird ein sogenannter MQTT-Subscriber benötigt, der die Daten vom MQTT-Broker abonniert und in die InfluxDB schreibt. Technisch gesehen kann dies mit einer Vielzahl von Frameworks / Sprachen / Tools umgesetzt werden. Zum Beispiel mit Python, Node.js, Golang oder auch Java. Wir haben uns für Node-RED entschieden, da es eine einfache und schnelle Möglichkeit bietet, um Daten von MQTT zu InfluxDB zu übertragen. Node-RED ist eine visuelle Programmierungsumgebung, die auf Node.js basiert und es ermöglicht, Datenströme einfach zu verarbeiten und zu transformieren. Besonders das Debugging und die Visualisierung der Datenströme ist in Node-RED sehr einfach und intuitiv. (Node-RED, 2025)

Im Vergleich zu z.B. einem Python-Skript ist Node-RED deutlich einfacher zu handhaben. Natürlich ist Node-RED nicht so performant oder bietet dieselbe Flexibilität und Kontrolle wie ein selbstgeschriebenes Skript, ist aber dafür deutlich einfacher zu konfigurieren und zu warten. Besonders in unserem spezifischen Fall, wo wenig Kontrolle über den schlussendlichen Server besteht, ist Node-RED eine gute Wahl. Eine automatische Ausführung von Skripten ist beinahe unmöglich, da keine Kontrolle über Sudo-Level Systemdiensten besteht.

4.5.1 NodeRED Umsetzung

Node-RED stellte sich als eine gute Entscheidung heraus, um den Flow zwischen MQTT und InfluxDB zu realisieren, und aber auch um bspw. Statusanfragen für die Admin-Seite zu beantworten. Das einzige Manko war das ständige Erneuern der Tokens für die InfluxDB, da Node-RED keine Möglichkeit bietet, Umgebungsvariablen zu verwenden. Schlussendlich entstand ein gut-funktionierender Flow:



5 Befestigung der Hardware

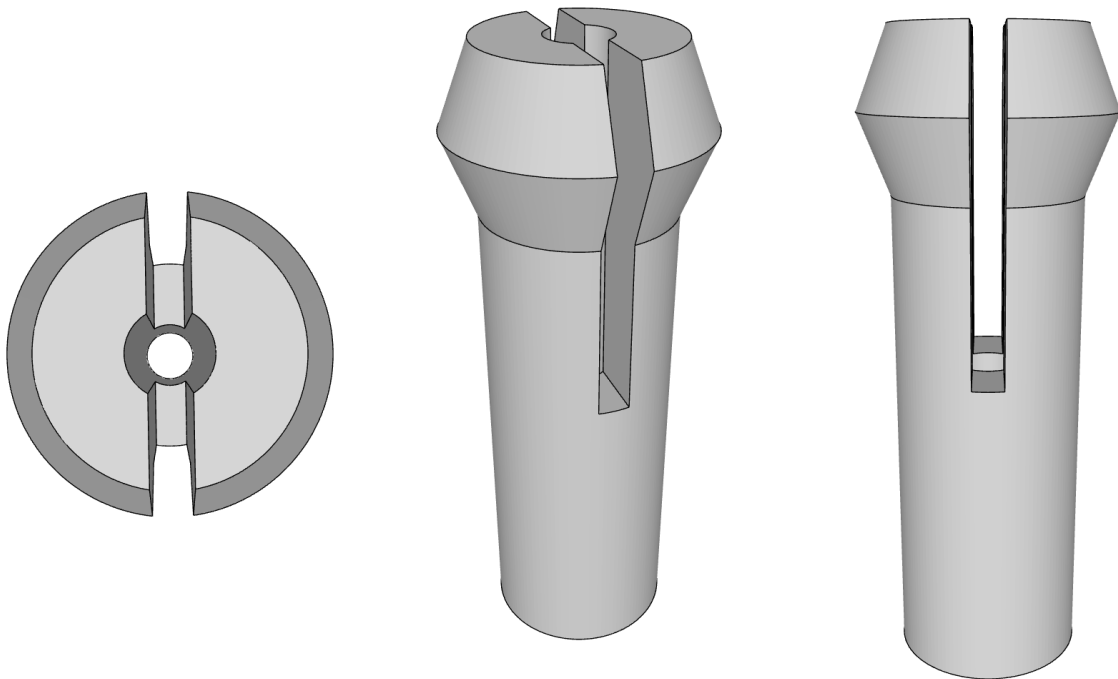
Um die Hardware des AudiTim-Systems sicher und stabil zu befestigen, sind verschiedene Montagemöglichkeiten verfügbar. Ein Großes Problem bei vorgefertigten Halterungen ist jedoch, dass diese oft an die Wand geschraubt oder anderweitig befestigt werden müssen. Daher fiel für unser Projekt die Wahl auf eine selbst designte Halterung, welche in einem 3D-Drucker hergestellt werden kann. Dies garantiert, dass keine Schäden an der Hardware oder am Raum, in welchem die Hardware montiert wird, entstehen.

5.1 Platzungskonzept

Aus akustischer Sicht liefern Messungen die zuverlässigsten Ergebnisse, wenn die Sensoren auf Kopfhöhe im Sitzen (1,2-1,5 m Höhe) angebracht sind und mindestens 1 Meter Abstand zu Wänden und der Schallquelle haben. Dieses Vorgehen minimiert frühe Reflexionen, stehende Wellen und lokale Pegelabweichungen - Faktoren, die die Messgenauigkeit maßgeblich beeinträchtigen (Oltheten, 2019). Leider ist diese ideale Positionierung in unserem Fall praktisch nicht umsetzbar. So birgt Montage am Boden, Tischen oder gesonderte Ständererhöhte Gefahr von Beschädigungen, Verschiebungen und Manipulation. An der Decke jedoch befinden sich bereits vorhandene Löcher; diese können genutzt werden, um die Sensoren und den Mikrocontroller sicher zu befestigen. Auch den einen Meter Abstand zu Wänden und Schallquellen kann so eingehalten werden. Dieser Kompromiss ermöglicht eine schützende, wartungsarme und vergleichbare Positionierung der Sensoren. So bleibt der Messaufbau reproduzierbar, sicher und funktional, auch wenn die akustisch ideale Lösung nicht vollständig umsetzbar ist.

5.2 Deckenhalterung

Um die Hardware an der Decke zu befestigen, wurde eine spezielle Halterung entworfen, die eine sichere Montage ermöglicht. Diese Halterung nutzt die in den Decken vorhandenen "Löcher" für die Montage und wurde als eine Art Pin konzipiert.

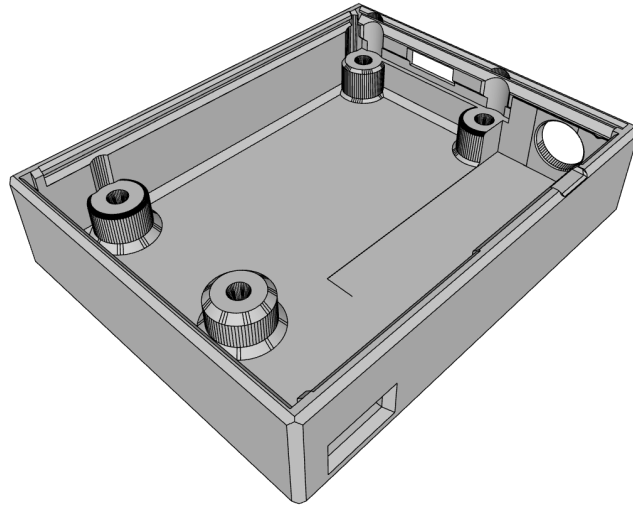


Der in den Bildern Dargestellte Pin kann durch den Spalt in der Mitte in die Öffnung der Decke gesteckt werden. Durch die konische Form der Spitze des Pins wird garantiert, dass dieser auch bei der Demontage des Projekts keine Schäden hinterlässt. Das runde Loch, welches sich durch den Pin zieht, dient dazu, den Pin mit einer Schraube an der Hardware zu befestigen.

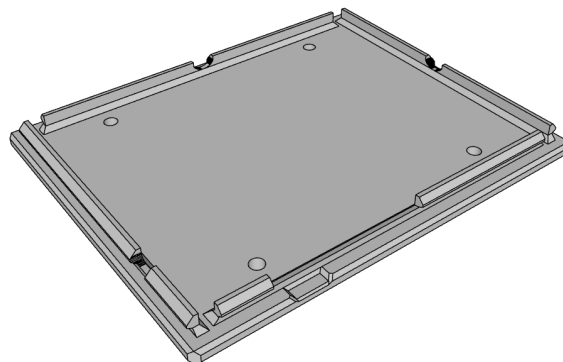
5.3 ESP32-Halterung

Der ESP benötigt eine Box, welche sicher an der Decke mit den Pins befestigt werden kann. Bei dem Design der Box wurde sich an einigen Vorlagen orientiert, um eine

optimale Passform zu gewährleisten. Die Box ist so gestaltet, dass sie den ESP32 sicher hält und gleichzeitig genug Platz für die Verkabelung bietet.

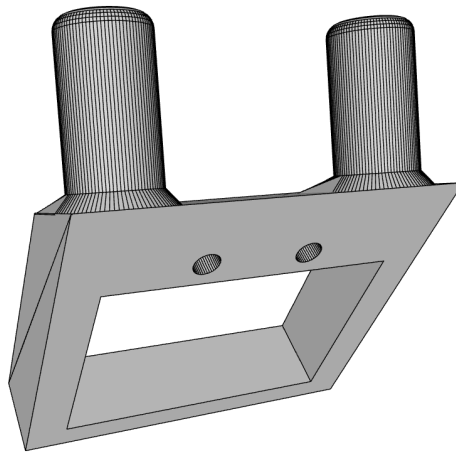


Die angesprochenen Löcher der Pins kommen in der Decke der Box zum Einsatz, um die Box sicher an der Decke zu befestigen. Diese werden durch die Löcher in der Decke mit Schrauben von innen befestigt.



5.4 Sensor-Halterung

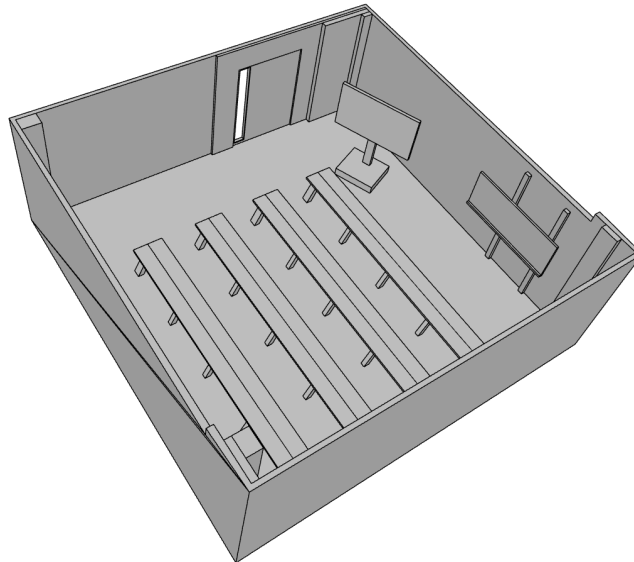
Die Sensorhalterung wurde so entworfen dass sie platz für einen möglichen Dämpfer bietet, welcher um das Mikrofon befestigt werden kann. Auch diese Halterung wird mit den Pins an der Decke befestigt, jedoch wird sie nicht geschraubt sondern direkt mit den Pins gedrückt. Der Sensor wird dann einfach mit zwei schrauben an der Halterung befestigt.



5.5 3D-Raummodellierung

Um eine grafische Übersicht der Montage der ESPs und Sensoren zu erhalten, wurde ein 3D-Modell des Raumes erstellt. Außerdem soll dieses später als Grundlage für die Simulation der Schallausbreitung als Hintergrund für die Heatmap im Frontend dienen. Dafür wurde vorerst der Vorlesungsraum ausgemessen und in einer technischen Zeichnung festgehalten:

Diese Zeichnung dient als Grundlage für die 3D-Modellierung des Raumes, welche in Blender durchgeführt wurde:



Hinweis zur Nutzung der 3D-Modelle Die im Projekt erstellten Blender-Dateien und Halterungen können direkt für den 3D-Druck verwendet werden. Zum Export der STL-Dateien ist im Blender-Menü der Pfad `audiTim/Blender/Printable` zu wählen.

6 Algorithmus zur Schallquellenlokalisierung

In diesem Kapitel werden verschiedene Ansätze zur Lokalisierung einer Schallquelle im Raum beschrieben. Die Verfahren unterscheiden sich in ihrer physikalischen Genauigkeit, im Rechenaufwand sowie in der praktischen Umsetzbarkeit. Es werden sowohl rein mathematische Interpolationsverfahren als auch modellbasierte und lernbasierte Methoden betrachtet.

6.1 Grundprinzipien

Ein präziser, aber aktuell nicht umsetzbarer Ansatz wäre die *Time Difference of Arrival* (TDOA) in Kombination mit Beamforming. Dabei wird die Zeitdifferenz der Schallausbreitung zwischen mehreren Mikrofonen gemessen. Über Verfahren wie Trilateration oder GCC-PHAT lassen sich daraus Einfallswinkel und Quelle berechnen. Da die vorhandene Hardware jedoch keine ausreichend genaue Zeitmessung erlaubt, konnte dieser Ansatz bislang nicht praktisch umgesetzt werden.

6.2 Betrachtete Verfahren

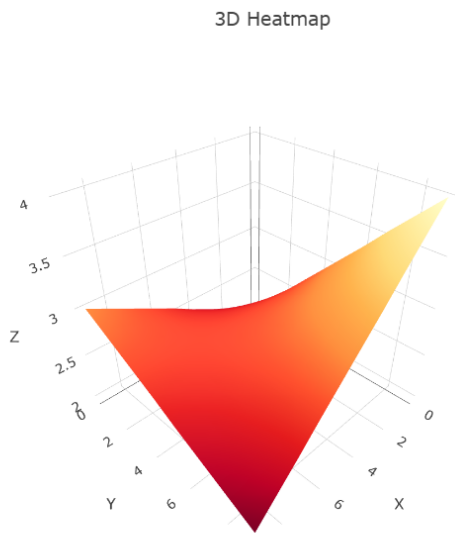
Bilineare Interpolation: Aus den vier Eckpunkten eines Messrasters wird durch zweifache lineare Interpolation eine flache Ebene gebildet. Dies liefert eine gleichmäßige, symmetrische Heatmap, berücksichtigt jedoch keinerlei physikalische Eigenschaften der Schallausbreitung. Damit eignet sich die Methode eher zur Visualisierung als zur tatsächlichen Lokalisierung.

```

1 for (let row = 0; row < 10; row++) {
2   const y = row / 9;
3   const rowData = [];
4   for (let col = 0; col < 10; col++) {
5     const x = col / 9;
6     const V =
7       sensorValues.d1 * (1 - x) * (1 - y) +
8       sensorValues.d2 * x * (1 - y) +
9       sensorValues.d3 * (1 - x) * y +
10      sensorValues.d4 * x * y;
11     rowData.push(Number(V.toFixed(2)));
12   }
13   grid.push(rowData);
14 }

```

Listing 6.1: Bilineare Interpolation



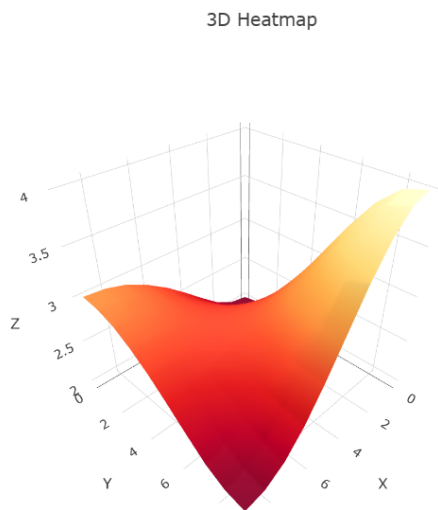
Inverse Distance Weighting (IDW): Hierbei wird der Einfluss eines Sensors umgekehrt proportional zur Distanz zur betrachteten Position gewichtet, häufig mit einem Distanzexponenten von $p = 2$ (inverse quadratische Gewichtung). Das Ergebnis ist eine weiche Heatmap, deren Maximum meist in der Nähe der lautesten Sensoren liegt. Bei mehreren Schallquellen oder komplexer Raumakustik verschwimmt jedoch das Ergebnis.

```

1 const power = 2; // Distanzexponent
2 for (let row = 0; row < 10; row++) {
3   const y = row / 9;
4   const rowData = [];
5   for (let col = 0; col < 10; col++) {
6     const x = col / 9;
7     let numerator = 0, denominator = 0;
8     for (const [key, { x: sx, y: sy }] of Object.entries(
9       sensorPositions)) {
10      const value = sensorValues[key];
11      const dx = x - sx;
12      const dy = y - sy;
13      const distance = Math.sqrt(dx * dx + dy * dy) || 0.0001;
14      const weight = 1 / Math.pow(distance, power);
15      numerator += value * weight;
16      denominator += weight;
17    }
18    rowData.push(Number((numerator / denominator).toFixed(2)));
19  }
20  grid.push(rowData);
21 }

```

Listing 6.2: Inverse Distance Weighting



Physikalisches Modell – Rückwärtsprojektion: Für jeden Punkt im Raum wird der erwartete Pegel an jedem Mikrofon berechnet:

$$L = L_0 - 20 \cdot \log_{10}(d)$$

mit d als Abstand zwischen Quelle und Mikrofon. Anschließend werden die berechneten Werte mit den real gemessenen Pegeln verglichen und die Abweichungen gespeichert. Eine Heatmap zeigt die Positionen mit den geringsten Fehlern als wahrscheinlichste Quelle an.

Maximum Likelihood Estimation (MLE): Dieser statistische Ansatz modelliert die Wahrscheinlichkeit, dass eine Quelle an Position (x, y) die gemessenen Pegel s_i erzeugt:

$$P = \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left(-\frac{(s_i - (L_0 - 20 \log_{10} d_i))^2}{2\sigma^2}\right)$$

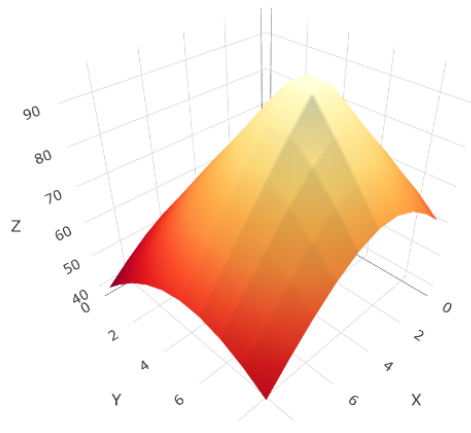
Der Punkt mit der höchsten Wahrscheinlichkeit wird als Quellenposition angenommen. Das Verfahren ist robust gegenüber Rauschen, erfordert jedoch ein präzises Modell für Messfehler.

Weighted Centroid Localization (WCL): Jeder Sensorwert wirkt wie eine „Gravitationskraft“, die die geschätzte Quelle in seine Richtung zieht. Durch Berechnung des gewichteten Mittelpunkts aller Sensoren lässt sich eine einzelne (x, y) -Position bestimmen, die den Schwerpunkt der Schallstärke darstellt. Gut geeignet, wenn ein dominanter Bereich im Raum deutlich lauter ist.

```
1 power = 2;
2 for (const [key, { x, y }] of Object.entries(sensorPositions)) {
3   const value = sensorValues[key];
4   const weight = Math.pow(value, power);
5   weightedX += x * weight;
6   weightedY += y * weight;
7   weightSum += weight;
8 }
```

Listing 6.3: Weighted Centroid Localization (WCL)

3D Heatmap



Machine Learning: Durch Training eines Modells (z. B. KNN, Random Forest, MLP) mit bekannten Quellenpositionen und zugehörigen Pegelwerten kann die Quellenposition direkt aus den Messwerten vorhergesagt werden. Bei guter Trainingsbasis liefert dies sehr schnelle und präzise Ergebnisse. Allerdings sind umfangreiche und repräsentative Trainingsdaten erforderlich.

Kombinierter IDW-WCL-Ansatz: Zur Verbesserung der Lokalisierungsgenauigkeit wird zunächst eine IDW-Heatmap berechnet. Anschließend wird der durch WCL ermittelte Schwerpunkt durch einen zusätzlichen Peak (z. B. Gauß-Blob) verstärkt und in die Heatmap eingeblendet. Dadurch lassen sich weiche Verteilungen mit einem klaren Maximum kombinieren.

```

1   for (let row = 0; row < gridSize; row++) {
2       const y = row / (gridSize - 1);
3       const rowData = [];
4
5       for (let col = 0; col < gridSize; col++) {
6           const x = col / (gridSize - 1);
7           let numerator = 0;
8           let denominator = 0;
9
10          for (const point of interpolationPoints) {
11              const dx = x - point.x;
12              const dy = y - point.y;
13              const distance = Math.sqrt(dx * dx + dy * dy) || epsilon;

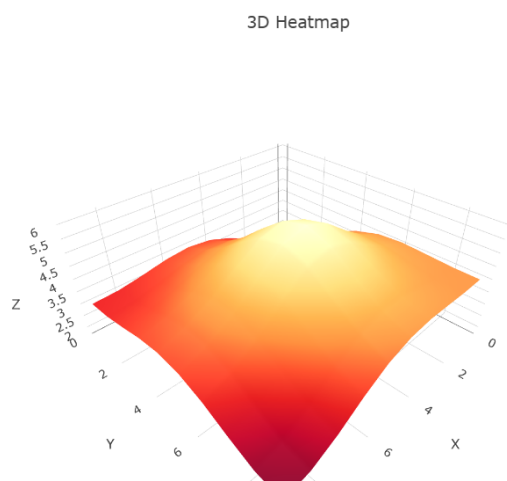
```

```

14     const weight = 1 / Math.pow(distance, idwFlatteningPower);
15     numerator += point.value * weight;
16     denominator += weight;
17 }
18
19 const interpolatedValue = numerator / denominator;
20 rowData.push(Number(interpolatedValue.toFixed(2)));
21 }
22
23 grid.push(rowData);
24 }

```

Listing 6.4: Kombiniertes IDW-WCL-Ansatz



6.3 Vergleich der Verfahren

Methode	Typ	Vorteile	Nachteile	Quelle bestimmbar?
Bilineare Interpolation	Interpolation	Schnell, symmetrische Heatmap	Kein physikalischer Bezug	Ungefähr
IDW	Interpolation	Einfach, intuitiv, glatte Heatmap	Kein klares Maximum	Ungefähr
Physikalisches Modell	Modellbasiert	Realitätsnah	Rechenintensiv, Annahmen nötig	Ja (Heatmap)
MLE	Wahrscheinlichkeit	Robust, statistisch fundiert	Modell für Fehler nötig	Ja (Wahrscheinlichkeit)
WCL	Schätzwert	Einfach, schneller Indikator	Ungenau bei asymmetrischen Verteilungen	Ja (Koordinate)
Machine Learning	Lernbasiert	Sehr schnell nach Training	Trainingsdaten nötig	Ja (xy direkt)

6.4 Fazit

Aus den Tests ergeben sich folgende Tendenzen:

- **Machine Learning** liefert bei guter Umsetzung die besten Ergebnisse, ist jedoch mit hohem Initialaufwand verbunden.
- **WCL** eignet sich gut, um den ungefähren Lautstärkenschwerpunkt im Raum zu bestimmen.
- **IDW** ist hilfreich, um Unterschiede in der Schallverteilung („hinten lauter als vorne“) darzustellen.
- **MLE** bietet hohes Potenzial für präzise Ortung, ist jedoch rechenintensiv und komplex in der Parametrisierung.

Eine Kombination aus IDW und WCL hat sich als vielversprechend erwiesen, um eine aussagekräftige Heatmap mit einem klar erkennbaren Maximum zu erzeugen. Zudem ist diese Methode relativ einfach zu implementieren und benötigt keine aufwändige Kalibrierung. Die Heatmap wird mit aktuellen Daten getestet, somit kann der Algorithmus durch die Gewichtung von IDW und WCL angepasst werden.

7 API-Kommunikation

In diesem Kapitel wird die Kommunikation zwischen Frontend und Backend beschrieben, insbesondere die Übertragung von Heatmap-Daten. Es werden verschiedene Ansätze zur Datenübertragung und -kompression vorgestellt, um die Effizienz und Performance der Anwendung zu optimieren.

7.1 Problemstellung

Wenn für jede Slider-Position eine API-Abfrage durchgeführt wird:

- Bei einer Frequenz von 10 Hz (alle 100 ms ein Datenpaket) ergeben sich 36,000 Zeitpunkte pro Sensor und somit 144,000 Werte pro Stunde.
- Beispielrechnung: $36,000 \times 10 \times 10 = 3,6$ Millionen Werte für ein 10×10 -Array.

7.2 Datenbasis

- **Heatmap-Größe:** 10×10 Werte (100 Felder pro Heatmap)
- **Datentyp:** Float16 (2 Byte pro Wert)
- **Frequenz:** 10 Hz (alle 100 ms ein Frame)
- **Beispiel:** 1 Stunde = 3,600 Sekunden = 36,000 Frames

Speicherbedarf (unkomprimiert):

$$100 \cdot 2 \text{ Bytes} = 200 \text{ Bytes/Frame}$$

$$200 \cdot 36,000 = 7,200,000 \text{ Bytes} \approx 7.2 \text{ MB}$$

7.3 Optimierungsoptionen

1. Downsampling (Mittelwert pro Sekunde)

3,600 Frames/h $\rightarrow 3,600 \times 100$ Werte = 0.72 MB. Vorteil: geringe Datenmenge, einfaches JSON; Nachteil: Peaks nicht sichtbar.

2. Textkompression (Gzip/Brotli)

Reduktion auf ca. 15–25 % $\rightarrow \approx 1.8$ MB.

3. Binärformate (MessagePack, Protobuf, FlatBuffers)

30–40 % Ersparnis gegenüber JSON $\rightarrow \approx 5.0$ MB.

4. Binär + Kompression

Kombination aus beidem $\rightarrow \approx 1.2$ MB.

5. Delta-Encoding (nur Änderungen)

Angenommen: jeder 3. Wert ändert sich $\rightarrow \approx 3.4$ MB.

6. Delta-Encoding + Kompression

Weitere Reduktion auf ≈ 1.0 – 1.2 MB.

7. Pre-Exportierte Dateien

Backend exportiert pro Stunde eine Datei $\rightarrow \approx 7.2$ MB.

8. Pre-Export + Kompression

Speicherbedarf: ≈ 1.2 MB/h; sehr performant und cachebar.

7.4 Vergleich der Ansätze

Beschreibung	Speicher/h	Vorteile	Nachteile
Mittelwert pro Sekunde (Downsampling)	≈ 0.72 MB	Sehr leicht, JSON möglich, wenig Traffic	Daten stark geglättet, Peaks nicht sichtbar
JSON + Gzip/Brotli	≈ 1.8 MB	Einfach umsetzbar, keine großen Änderungen	Höhere Lesezeit, trotzdem noch Textformat
Binärformat	≈ 5.0 MB	Schnell zu dekodieren, weniger Overhead	Komplexere Integration im Frontend
Binär + Kompression	≈ 1.2 MB	Gute Mischung aus Effizienz & Geschwindigkeit	Backend-Implementierung aufwändiger
Delta-Encoding	≈ 3.5 MB	Ideal bei geringen Änderungen	Schlecht bei stark variablen Daten
Delta + Kompression	1.0–1.2 MB	Sehr gute Kompression	Schwer zu debuggen, komplex
Pre-Export (roh)	≈ 7.2 MB	Kein Runtime-Rechenaufwand, einfaches Caching	Hoher Speicherbedarf
Pre-Export + Kompression	≈ 1.2 MB	Sehr performant, cachebar, ideal für Slider	Dateiverwaltung nötig

Fazit

Aufgrund der hohen Komplexität und der großen Datenmengen ist es sinnvoll, die Heatmap-Arrays in einer separaten Datenbasis zu speichern. Das Frontend erhält die Daten in vorverarbeiteter Form durch Downsampling, um die Übertragungsmenge zu reduzieren und die Darstellungsgeschwindigkeit zu erhöhen. Zusätzlich wird eine Komprimierung der Daten in Betracht gezogen, um den Speicher- und Bandbreitenbedarf weiter zu minimieren, ohne die wesentliche Aussagekraft der Werte zu verlieren.

Literaturverzeichnis

Appventurez (2025), 'Node.js frameworks comparison: Express.js, koa.js, and more'.

Accessed: 2025-08-01.

URL: <https://www.appventurez.com/blog/node-js-framework>

Community, B. (2025), 'Koa.js vs express.js: Which node.js framework to choose?'.

Accessed: 2025-08-01.

URL: <https://betterstack.com/community/guides/scaling-nodejs/koa.js-vs-express.js>

Node-RED (2025), 'Node-red dokumentation'. Accessed: 2025-08-14.

URL: <https://nodered.org/docs/>

Oltheten, W. (2019), 'The 7 steps of ideal mic placement'. Accessed: 2025-07-31.

URL: <https://sonicscoop.com/7-steps-for-ideal-mic-placement/>

Powell, Z. (2023), 'Angular vs. react: Ein detaillierter vergleich'. Accessed: 2025-08-14.

URL: <https://kinsta.com/de/blog/angular-vs-react/>