

**Hochschule Flensburg**

# **BACHELOR – THESIS**

Thema: Entwicklung einer Webapplikation zum Thema Literatur mit dem Fokus auf Chat- und Sprachfunktionalität

von: Jonas Leonhard

Matrikel-Nr.: 611179

Studiengang: Medieninformatik

Betreuer/in und

Erstbewerter/in: Dipl. VK Tobias Hiep

Zweitbewerter/in: Dipl.-Designer Uwe Zimmermann

Ausgabedatum: 08.06.2021

Abgabedatum: 08.08.2021

# Inhaltsverzeichnis

1 - Einleitung	6
1.1 - Überblick über die Arbeit	6
1.2 - Motivation	7
1.3 - Ziel des Projekts	8
1.4 - Alleinstellungsmerkmale	9
1.5 - Relevanz und Aktualität des Projekts	10
1.6 - Marktanalyse	11
2 - Projektanforderungen	11
2.1 - User-Stories	11
2.2 - Funktionale Anforderungen	14
2.2.1 - Diskussionsräume	15
2.2.2 - Reporting, Content-Moderation und Hassrede	16
2.2.3 - Rechte und Take-Down-Notices	17
2.2.4 - Blockieren und melden	17

2.3 - Grenzen des Prototyps	17
3 - Technische Herausforderungen	18
3.1 - Pflege	18
3.2 - Verbindungsmenge und Skalierbarkeit	19
3.3 - Live	19
3.4 - Speicher	19
3.5 - Missbrauch	20
3.6 - Dynamischer und statischer Content	20
3.7 - Benutzer*innen	21
4 - Architektur	21
5 - Systembestandteile	22
5.1 - Node.js	22
5.2 - REST und GraphQL API's	23
5.3 - Next.js und React	24
5.4 - Postgres	26
5.5 - Strapi CMS	27

5.6 - Express.js	28
5.7 - WebRTC	28
5.7.1 - Mesh-Architektur	30
5.7.2 - MCU - Multipoint-Conferencing-Unit	32
5.7.3 - SFU - Selective-Forwarding-Unit	34
5.7.4 - Simulcast- und Scalable-Video-Coding	35
5.7.5 - Horizontale Skalierung	36
5.8 - WebSockets	38
5.9 - Docker	40
5.10 - SFU Voice server - Janus	41
5.11 - User-Authentication - Jason-Web-Tokens	42
5.12 Ant Design	43
6 - Technische Implementierung	44
6.1 - Postgres-Service	45
6.2 - Auth Service	45
6.3 - CMS-Service	46

6.4 - Socket-Service	46
6.5 - WebRTC-Service	47
6.6 - Frontend-Service	48
7 - Systembeschreibung	50
8 - Das Projekt starten, bauen und deployen	52
9 - Fazit	53
10 - Quellenverzeichnis	54
11 - Liste der verwendeten Abkürzungen	60

# 1 - Einleitung

## 1.1 - Überblick über die Arbeit

Bei dieser Arbeit handelt es sich um die Dokumentation und theoretische Einordnung der Erstellung eines Prototypen für eine Web-App. Die Applikation soll es Benutzer\*innen ermöglichen, sich interaktiv und digital über das Thema Literatur auszutauschen. Der Fokus der Arbeit liegt auf den zentralen Chat- und Audiofunktionalitäten, da diese den interaktiven Kern der Plattform darstellen. Zuerst werde ich meine Motivation und den Prozess der Ideenfindung beschreiben. Anschließend formuliere ich das Ziel des Projektes und arbeite anhand dessen die Alleinstellungsmerkmale meiner Idee heraus und begründe, warum diese relevant und aktuell in das Zeitgeschehen einzuordnen ist. Im Anschluss führe ich eine Marktanalyse durch und gehe auf ähnliche Produkte und deren Abgrenzung zu meiner Idee ein. Um aus meiner Grundidee konkrete Features abzuleiten, erstelle ich im darauffolgenden Kapitel User-Stories und beschreibe aus der Sicht von Benutzer\*innen den Mehrwert von Features der Plattform.

Mit Hilfe der User-Stories leite ich anschließend funktionale Anforderungen an die Plattform ab, die für das fertige Produkt entwickelt sein müssten und erstelle jeweils technische Herausforderungen für die Entwicklung solcher Funktionalitäten. Dabei verweise mit einem Lösungsansatz darauf, wie ich diese in meinem Prototypen gelöst habe oder lösen würde und gebe einen Überblick über die Architektur und Systembestandteile der Plattform.

Ich gehe anschließend darauf ein, wie und welche dieser Systembestandteile in den Prototypen als Services implementiert wurden,

dokumentiere diese einzeln und gehe mit einem Schaubild darauf ein, wie sie zusammenhängen.

Abschließend erläutere ich, wie man den Prototypen für die Weiterentwicklung bauen und auf einem Server deployen müsste. Im abschließenden Fazit gehe ich noch einmal auf den gesamte Arbeitsprozess ein und reflektiere diesen.

## **1.2 - Motivation**

Auf die Idee, im Rahmen meiner Bachelorarbeit eine digitale Literatur-Applikation zu gestalten, kam ich erstmals bei dem Besuch einer Büchervorstellung und Lesung in einer Hamburger Buchhandlung. Ich fragte mich, ob es möglich wäre, Events dieser Art auch auf einer Online-Plattform stattfinden zu lassen. Außerdem interessierte mich, wie man diese Plattform interaktiv und ansprechend gestalten könnte, ohne dabei auf die besondere und persönliche Atmosphäre zu verzichten.

Eine solche Plattform könnte einen ortsunabhängigen und flexiblen digitalen Bereich schaffen, in dem es Literatur-Liebhaber\*innen ermöglicht werden würde, sich über ihre Leidenschaft auszutauschen. Bei meinem Lesungsbesuch war mir noch nicht bewusst, wie aktuell dieser Gedanke durch die Corona-Pandemie einige Monate später werden würde. Eine digitale Austauschplattform bietet nicht nur den Vorteil, trotz räumlicher Trennung durch Lockdown, Reisen oder Ähnliches, in persönlichen Kontakt zu treten und sich mit Gleichgesinnten auszutauschen, sondern stellt zusätzlich einen weitgehend barrierefreien Raum dar, der Menschen mit eingeschränkter Mobilität oder Einschränkungen beim Sehen oder Hören (mit Hilfe der Dualität von Chat und Audio) inklusiv mit einbindet. Allein in Deutschland haben 7,5 Millionen Menschen eine anerkannte Schwerbehinderung (Aktion Mensch, o.D.). Darunter befinden sich ca. 80.000 Gehörlose. Darüber hinaus gibt es weitere Menschen, die zwar nicht als schwerbehindert eingestuft werden,

aber trotzdem auf verschiedene Weise körperlich eingeschränkt sind, wie zum Beispiel 16 Millionen schwerhörige Menschen in Deutschland. (Merker, 2021).

Diese Menschen sind überdurchschnittlich oft intensive Nutzer\*innen von Online-Plattformen und -Inhalten, werden aber gleichzeitig selten aktiv im Web-Design mitgedacht (Aktion Mensch, o.D.). Im Rahmen meiner Möglichkeiten möchte ich mit dieser Anwendung auch Nutzer\*innen mit unterschiedlichen Einschränkungen Zugang zu mehr digitaler Teilhabe verschaffen.

Im Gegensatz zu anderen Literatur-Plattformen, auf die ich zu einem späteren Zeitpunkt in dieser Arbeit genauer eingehen werde, möchte ich versuchen, den Austausch möglichst natürlich und benutzer\*innenfreundlich zu gestalten. Ich gehe für mein Projekt von einer Zielgruppe aus, die nicht zwangsläufig technikaffin ist und der ich eine reibungslose und organische Nutzung ermöglichen möchte. Bisher bin ich selbst nur als Benutzer mit Voice- und Chat-Applikationen wie *Whatsapp*, *Zoom* oder *Clubhouse* in Berührung gekommen und möchte mich mit dieser Arbeit erstmals an der Programmierung dieser Anwendungen und der Implementierung einzelner Features versuchen und mich in diese Richtung weiterbilden und ausprobieren.

### **1.3 - Ziel des Projekts**

Das Ziel für der finalen Webanwendung ist es, Aspekte interaktiver Plattformen mit dem thematischen Fokus klassischer Literaturforen zu kombinieren, bei denen der Fokus auf dem Bewerten und Kommunizieren liegt. In Bezug auf die Eigenschaft der Interaktivität habe ich mich unter anderem durch die App *Clubhouse* inspirieren lassen. *Clubhouse* ist eine Social-Media-App für iOS und Android, bei der Benutzer\*innen in Voice-Chaträumen miteinander interagieren können. Die Räume können kleine



Gruppen, aber auch mehrere Tausend Personen umfassen. Sowohl als Zuhörer\*in, als auch als selbst sprechende Person, kann an den Live-Events teilgenommen werden (Clubhouse Community Guidelines, 2021).

Auch wenn ich die Audioaspekte der App sehr spannend und innovativ finde, pflichte ich auch einigen der Kritikpunkte an der Anwendung bei und sehe Verbesserungspotential. Ein vermehrt geäußelter Kritikpunkt ist die mangelnde Barrierefreiheit der App, durch ihren reinen Audio-Fokus. Meine eigene Anwendung möchte ich deshalb mithilfe eines Live-Chats breiter aufstellen. Dabei ist zu verdeutlichen, dass ich mich lediglich durch diese Kritik und Perspektive auf Plattformen habe inspirieren lassen und mir bewusst ist, dass mein Prototyp nicht mit der fertig entwickelten App *Clubhouse* oder anderen etablierten Plattformen verglichen werden kann. Neben *Clubhouse* haben mich auch klassische Literaturseiten wie zum Beispiel [goodreads.com](https://www.goodreads.com) oder [lovelybooks.de](https://www.lovelybooks.de) durch ihren Aufbau und ihre Kategorien und Features inspiriert.

Welche Komponenten letztendlich die Architektur meiner Webanwendung bilden, werde ich im Abschnitt "Architektur" genauer erläutern.

## 1.4 - Alleinstellungsmerkmale

Bei meiner Recherche habe ich festgestellt, dass es bisher keine digitale Plattform gibt, die einen Austausch per Chat- und Audiofunktion ermöglicht und sich ausschließlich auf das Thema Literatur fokussiert. Zwar ist ein Austausch zu diesem Thema auch auf anderen Plattformen wie *Discord* oder *Clubhouse* durch private Communities möglich, dennoch ist meine Anwendung auf ein Themengebiet ausgerichtet und zieht vermutlich eine spezialisiertere und besonders interessierte Zielgruppe an, sodass sich Benutzer\*innen mit der Plattform identifizieren und eine verstärkte Bindung zu ihr aufbauen können. Durch die Entstehung persönlicher Gespräche, die über das rein fachliche hinausgehen,

verspreche ich mir außerdem die Entstehung eines persönlichen Gruppengefühls und die natürliche Vernetzung von Benutzer\*innen. Mit "natürlicher Vernetzung" meine ich in diesem Fall eine digitale Kommunikation, die der im realen Leben möglichst nahe kommt und ihren technischen Charakter in den Hintergrund stellt.

Meine Anwendung könnte eine Anlaufstelle für alle Bücherliebhaber\*innen darstellen, die sich zum Beispiel über ein konkretes Buch austauschen wollen, aber auch für alle, die sich inspirieren und überraschen lassen wollen und gerne neue Bücher entdecken.

## **1.5 - Relevanz und Aktualität des Projekts**

Gerade in Zeiten der Corona-Pandemie gewinnt die digitale Vernetzung immer mehr an Bedeutung. Von Homeoffice und Homeschooling über den Austausch mit Freund\*innen und Familie bis hin zum Ausleben persönlicher Hobbies in Form von Sportkursen, Home-Workouts, oder digitalen Spieleabenden. Die Gesellschaft ist mehr denn je auf digitale Strukturen angewiesen. Auch wenn diese den realen Kontakt nicht adäquat ersetzen können, bieten sie Möglichkeiten, gewohnte Aktivitäten trotzdem stattfinden zu lassen und sich individuell auszuleben. Auch meine Plattform sehe ich in diesem Bereich verortet, da sie Menschen die Möglichkeit bietet, sich im Internet über ihr Hobby auszutauschen und miteinander in Kontakt zu treten. Dabei versuche ich, das Nutzungserlebnis so persönlich wie möglich zu gestalten, um einen freien Austausch zu ermöglichen. Dieser vermittelt den Nutzer\*innen das Gefühl, sich tatsächlich in einem realen Raum miteinander zu befinden und gemeinsame Erlebnisse zu schaffen. Die Relevanz meines Projektes sehe ich also einerseits in der, durch die Pandemie, geschaffenen Situation und andererseits im Mangel an wirklich interaktiven Plattformen für eine

Zielgruppe, die sich über Literatur austauschen möchte und in einer sicheren Online-Umgebung in Kontakt treten möchte.

## 1.6 - Marktanalyse

Bei meiner Recherche bin ich auf diverse deutsche und internationale Webseiten gestoßen, die sich mit dem Thema Literatur beschäftigen. Diese lassen sich in zwei grundlegende Kategorien einteilen. Die erste Kategorie bilden klassische Webseiten großer oder kleiner Buchhandlungen wie zum Beispiel [Thalia.de](http://Thalia.de) oder [barnesandnoble.com](http://barnesandnoble.com), die zu Verkaufszwecken Buchinhalte zusammenfassen und in Kategorien unterteilen. Die zweite Kategorie bilden Online-Foren, die sich vor allem mit dem Bewerten von Büchern beschäftigen und im Wesentlichen aus Rezensionen von Benutzer\*innen und/oder Redakteur\*innen bestehen. Beispiele für diese Foren sind unter anderem [goodreads.com](http://goodreads.com) oder [lovelybooks.de](http://lovelybooks.de).

Abgesehen von dem thematischen Schwerpunkt der Literatur, gibt es bereits einige Plattformen, die Live-Chats und Live-Audio zur Verfügung stellen und somit meiner Anwendung ähneln. Über *Google Hangouts*, *Skype* oder *Discord* lassen sich zum Beispiel Gruppengespräche führen. Diese Anbieter zielen jedoch nicht auf spezialisierte Zielgruppen und deren Bedürfnisse ab, sondern bieten eine offene Struktur für unterschiedliche Vorhaben.

## 2 - Projektanforderungen

### 2.1 - User-Stories

Um aus meiner Idee konkrete Features für das Projekt abzuleiten, beschreibe ich mittels sogenannter User-Stories die informellen und allgemeinen Software-Features aus der Sicht von Benutzer\*innen der

Webapplikation. Der Zweck einer User-Story ist es, zu beschreiben, welcher persönliche Wert und welche Nutzungsmöglichkeiten hinter den einzelnen Funktionalitäten stecken. Dabei ist diese allerdings kein Feature im Sinne der Software-Entwicklung, sondern ein Endziel, welches einen an Benutzer\*innen orientierten Rahmen für die Entwicklung von Software-Features bereitstellen soll. Es soll deutlich werden "warum" und "was" für Benutzer\*innen gebaut wird und welcher Mehrwert dabei für sie erzeugt wird. Dabei wird in einfacher Sprache das gewünschte Ergebnis umschrieben (Rehkopft, o. D.). Die technische Herangehensweise sowie konkrete Anforderungen, werden zu einem späteren Zeitpunkt aus den User Stories abgeleitet.

Der Vorteil von User Stories ist vor allem, dass der Fokus im Entwicklungsprozess auf den Benutzer\*innen und ihrer späteren User-Experience liegt und nur Funktionalitäten implementiert werden, die einen klaren Mehrwert für diese schaffen. Darüber hinaus ermöglichen User-Stories das vereinfachte Einteilen von sinnvollen Arbeitspaketen für die jeweiligen Entwickler\*innen und ermöglichen es, den Aufwand eines Projekts im Voraus besser einschätzen und somit organisieren zu können

(Rehkopft, o. D.).

In der folgenden Tabelle sind alle für das Projekt relevanten User-Stories übergreifend dargestellt.

Story-Name:	Benutzer*innen Typ...	möchte/n...	so dass er/sie...
<b>Registrierung</b>	Benutzer*innen	möchten sich registrieren	um sich einzuloggen/ um einem Diskussionsraum beizutreten

<b>Story-Name:</b>	<b>Benutzer*innen Typ...</b>	<b>möchte/n...</b>	<b>so dass er/sie...</b>
<b>Login</b>	Registrierte Benutzer*innen	sich mit einem Account einloggen	um einem Diskussionsraum beizutreten
<b>Diskussionsraum</b>	Registrierte Benutzer*innen	einen Raum-Link teilen	um andere Benutzer*innen zu einer Diskussion einzuladen
<b>Diskussionsraum (Chat)</b>	Registrierte Benutzer*innen	eine Nachricht an den Raum senden	um mit anderen Benutzer*innen über Literatur zu diskutieren
<b>Diskussionsraum (Voice)</b>	Registrierte Benutzer*innen	im Raum sprechen	um mit anderen Benutzer*innen über Literatur zu diskutieren
<b>Diskussionraum (Video)</b>	Registrierte Benutzer*innen	möchten die Kamera ausschalten	um nicht von anderen Benutzer*innen im Raum gesehen zu werden
<b>Diskussionraum (Voice)</b>	Registrierte Benutzer*innen	möchten das Mikrofon ausschalten	um nicht von anderen Benutzer*innen im Raum gehört zu werden
<b>Einstellungen</b>	Registrierte Benutzer*innen	möchten den Account löschen	weil sie die Plattform nicht weiter nutzen möchten
<b>Einstellungen</b>	Registrierte Benutzer*innen	möchten ihren Benutzernamen ändern	weil der alte Name nicht mehr gefällt

<b>Story-Name:</b>	<b>Benutzer*innen Typ...</b>	<b>möchte/n...</b>	<b>so dass er/sie...</b>
<b>Homepage</b>	Benutzer*innen	möchten eine Liste an neuen literarischen Werken durchsuchen	um mehr Informationen über diese zu erhalten
<b>Homepage</b>	Benutzer*innen	möchten eine Liste an literarischen Werken durchsuchen	um einer Diskussion über diese beizutreten
<b>Literaturseite</b>	Benutzer*innen	möchten detaillierte Informationen über ein Buch erhalten	um mehr über ein Werk zu erfahren
<b>Literaturseite</b>	Benutzer*innen	möchten ein Buch kaufen	um es zu lesen
<b>Literaturseite</b>	Benutzer*innen	möchten sich einen Überblick über Rezensionen zu einem bestimmten Buch verschaffen	um sich eine Meinung zu bilden
<b>Literaturseite</b>	Benutzer*innen	möchten Meinungen über ein literarischen Werk teilen	um anderen Nutzer*innen Bücher zu empfehlen oder sich über diese auszutauschen

## 2.2 - Funktionale Anforderungen

Aus den User-Stories lässt sich eine Reihe von Funktionalitäten ableiten, die für die Applikation implementiert werden müssen, um alle Bedürfnisse der Benutzer\*innen zu erfüllen. Zunächst muss es Benutzer\*innen ermöglicht werden, sich ein Benutzerkonto in Form eines Accounts anzulegen und sich mit diesem in die Webapplikation einzuloggen. Die Benutzerdaten der Accounts müssen editierbar und jederzeit löschar sein.

Die Startseite soll es den Benutzer\*innen ermöglichen, einen Überblick über neu erschienene literarischen Werke zu erhalten. Jedes Werk in dieser Übersicht benötigt einen eigenen Bereich in Form einer Unterseite mit weiterführenden Informationen. Darüber hinaus soll die fertige App es ermöglichen, Meinungen in Form von Rezensionen zu teilen, Live-Diskussionen beizutreten und Bücher zu erwerben. Um den Live-Audio-Charakter der App zu betonen, wären auch Rezensionen in Form von kurzen Audio-Aufnahmen denkbar. Diese könnten den Austausch noch persönlicher gestalten. In den Live-Diskussionen soll mit Video, Audio und per Chat kommuniziert werden können.

---

### 2.2.1 - Diskussionsräume

In Bezug auf die Organisation innerhalb der Diskussionsräume ist es essenziell, eine Struktur zu schaffen, die bestimmt, wer wann sprechen darf. So bleiben Diskussionen möglichst übersichtlich und produktiv.

Aus diesem Grund sieht die Web-App drei Arten von Diskussionsräumen vor: offene Diskussionsräume, geschlossene Diskussionsräume und moderierte Diskussionsräume. Ein offener Diskussionsraum ist für alle Benutzer\*innen zugänglich. Einem geschlossenen Diskussionsraum kann nur beigetreten werden, sofern eine entsprechende Einladung vorliegt. Hier können Benutzer\*innen private Unterhaltungen abhalten. In einem moderierten Diskussionsraum können die Administrator\*innen des

Raumes Teilnehmer\*innen die Erlaubnis zum Sprechen erteilen, diese aber gegebenenfalls auch wieder entziehen.

So können auch Diskussionen mit vielen Teilnehmer\*innen möglichst übersichtlich geführt werden, ohne dass Chaos ausbricht. Durch ihr zusätzliches Maß an Kontrolle bieten geschlossene und moderierte Räume außerdem in der Regel Schutz vor Hassrede und unangemessenen Beiträgen. Weitere Schutzvorrichtungen der Web-App, die über diese Raumstrukturen hinausgehen, werden im nächsten Abschnitt genauer erläutert.

---

### 2.2.2 - Reporting, Content-Moderation und Hassrede

Um den eben genannten Problemen, die von Benutzer\*innen generierte Inhalte häufig mit sich bringen, entgegenzuwirken, benötigt die Plattform kontrollierende Mechanismen und Instanzen. Diese können zum Beispiel in Form eines Reportings oder einer Content-Moderation etabliert werden. Sie wirken unter anderem der Erstellung unangemessener Kommentare und der Verbreitung von Hassrede auf der Webseite entgegen und helfen somit dabei, Benutzer\*innen ein angenehmes Nutzungserlebnis zu ermöglichen, das nur relevanten und angemessenen Content miteinbezieht.

Einerseits muss es Administrator\*innen also möglich sein, Postings von Benutzer\*innen zu löschen, die nicht den Plattformrichtlinien entsprechen, andererseits muss es zusätzlich die Möglichkeit für Benutzer\*innen geben, auf die oben genannte Art von Beiträgen aufmerksam zu machen und diese an zuständige Administrator\*innen zu melden.



---

### 2.2.3 - Rechte und Take-Down-Notices

Benutzergenerierte Inhalte bringen außerdem eine Copyright-Problematik mit sich, da Benutzer\*innen Daten hochladen könnten, an denen sie keine Rechte besitzen. Damit die Plattform für die Eventualität eines Rechtsstreites abgesichert ist, müssen Benutzer\*innen ihr Einverständnis für das Abtreten ihrer Rechte an ihren Inhalten abgeben. Außerdem muss sichergestellt werden, dass Benutzer\*innen diese Rechte überhaupt besitzen und nicht die Werke Dritter auf der Plattform hochladen. Sollte dies dennoch passieren, muss den betroffenen Parteien ermöglicht werden, Take-Down-Notices (Anweisungen zum sofortigen Entfernen) zu erteilen und entsprechend auf Copyright-Verletzungen zu reagieren.

---

### 2.2.4 - Blockieren und melden

Cybermobbing und andere Arten der Belästigung oder Beleidigung sind mittlerweile ein zentrales und weitverbreitetes Problem auf Internetplattformen (Landeszentrale für politische Bildung Baden-Württemberg, 2020). Um ein solches Verhalten bestmöglich zu unterbinden, ist die Funktion des Blockierens von großer Bedeutung.

Deshalb muss es möglich sein, User zu blockieren oder zu melden, und ihre Inhalte für den blockierenden User nicht weiter anzuzeigen. Unangemessene Inhalte und gegebenenfalls rechtswidriges Verhalten soll darüber hinaus in einem Reporting festgehalten und somit dokumentiert werden.

## 2.3 - Grenzen des Prototyps

Im Rahmen meiner Abschlussarbeit war es mir für den Prototypen dieses Projekts nur möglich, die wichtigsten Features für die Webanwendung zu

implementieren. Das heißt, dass ich mit meinem Prototypen das Grundgerüst für die Diskussionsräume und den Rest der Plattform aufgestellt habe. Es können Inhalte im CMS gepflegt werden, Benutzer\*innen können Accounts anlegen und in Diskussionsräumen in Echtzeit diskutieren.

Die in den vorgehenden Abschnitten erwähnten Features müssten in den Prototypen der Anwendung implementiert werden, um die Plattform als fertiges Produkt oder Minimum-Viable-Product (MVP) auf den Markt bringen zu können. Ihre technische Integration in den Prototypen würde an dieser Stelle jedoch den Rahmen dieser Arbeit sprengen.

### **3 - Technische Herausforderungen**

Aus den abgeleiteten funktionalen Anforderungen der User-Stories entstehen die zentralen technischen Herausforderungen des Projektes, auf die ich in den Abschnitten "Architektur" und "Systembestandteile" mithilfe von Lösungsansätzen weiter eingehen werde.

#### **3.1 - Pflege**

Um eine Auswahl der neu erschienenen Bücher abbilden zu können, müsste die Plattform einer Redaktion ermöglichen, eigenständig neue Inhalte zu erstellen und hochzuladen. Um die notwendige Struktur für derartige Tätigkeiten zu schaffen, verfügt das Projekt über das Content-Management-System (CMS) Strapi. (Siehe Abschnitt "Strapi CMS"). Strapi ermöglicht es, in einem grafischen Web-User-Interface neue Inhalte zu erstellen und diese zu verwalten.

## **3.2 - Verbindungsmenge und Skalierbarkeit**

Entsprechend der Nachfrage, müsste ein digitaler Diskussionsraum in der Lage sein, mehrere Hundert bis mehrere Tausend Zuhörer\*innen und Sprecher\*innen zu unterstützen. Traditionelle Peer-to-Peer-Verbindungen mit WebRTC unterstützen allerdings maximal fünf gleichzeitig bestehende Verbindungen. Um dieses Limit zu erhöhen, verfügt die Anwendung über eine Server-Forwarding-Unit (Siehe Abschnitt "WebRTC"), an die Benutzer\*innen eines Diskussionsraums mittels WebSockets (Siehe Abschnitt " - WebSockets") verteilt werden.

## **3.3 - Live**

Da es sich bei den Diskussionsräumen nicht um statische Inhalte handelt, wird für deren Darstellung und Interaktivität Javascript benötigt, welches Live-Chat- und Audio-Funktionalitäten implementiert. Deshalb setze ich das Framework Next.js ein, welches eine Mischung aus statischen Seiten und nicht statischen Seiten ermöglicht (Siehe Abschnitt "Next.js und React"). Die Diskussionsräume benötigen Protokolle, die es dem Browser ermöglichen, Live-Daten mit anderen verbunden Benutzer\*innen im gleichen Raum auszutauschen. Diese implementiere ich mittels WebSocket und WebRTC Protokollen in Javascript. Die Interaktion des Chats findet mittels WebSockets (siehe Abschnitt "WebSockets") statt und das Live-Audio mittels WebRTC (siehe Abschnitt "WebRTC").

## **3.4 - Speicher**

Alle von den Redakteur\*innen und Benutzer\*innen generierten Inhalte, sowie die Daten der Nutzer\*innen oder andere anfallende Daten müssen persistent gespeichert werden und schnell abrufbar sein. Dazu benutze ich eine relationale Datenbank (siehe Abschnitt "Postgresql"). Von

Benutzer\*innen generierte Inhalte in Form von Audio-Rezensionen müssten außerdem möglichst vor dem Speichern auf dem Server komprimiert werden und von Administrator\*innen blockiert oder gelöscht werden können, sofern diese Inhalte den Richtlinien der Plattform widersprechen.

### **3.5 - Missbrauch**

Um jegliche Art des Missbrauchs im Zusammenhang mit der Applikation zu verhindern, sollte es Benutzer\*innen möglich sein, andere Benutzer\*innen zu blockieren oder zu melden, damit Administrator\*innen die Situation einschätzen und dementsprechend handeln können. Dazu müssten Richtlinien erstellt werden, die nachvollziehbar und auch durchsetzbar sind. Inhalte, die von Benutzer\*innen generiert werden, könnten außerdem vor dem Hochladen von Administrator\*innen geprüft und freigeschaltet werden, damit sichergestellt werden kann, dass diese nicht gegen die Richtlinien verstoßen.

### **3.6 - Dynamischer und statischer Content**

Aus technischer Sicht besteht die Webseite sowohl aus statischem Content (zum Beispiel den Literaturlisten), als auch aus dynamischem Content (zum Beispiel den Diskussionsräume). Um diese beiden Arten von Content miteinander zu verbinden, benutzt die Plattform Next.js. Next.js (Siehe Abschnitt "Next.js und React") ermöglicht es, dass Nicht-Diskussions-Seiten auch ohne Javascript aufgerufen werden können und interaktive Funktionalitäten bereitstellen, da sie statisch oder Server-seitig gerendert (SSR) werden. Für die Diskussionsräume initialisiert Next.js das React-Framework auf der Client-Seite und ermöglicht so das dynamische Rendering von Komponenten und deren Inhalten auf der Seite der Benutzer\*innen (CSR).

### **3.7 - Benutzer\*innen**

Damit die Plattform nachvollziehen kann, wer gerade redet oder schreibt, müssen Benutzer\*innen ein Konto anlegen und sich über dieses einloggen. Ich habe mich dazu entschieden Accounts mittels Jason Web Tokens zu implementieren (siehe Abschnitt "User Authentication"). Dies bedeutet, dass einige Inhalte nicht ohne einen aktiven Account funktionieren. Die App muss grundsätzlich zwar nutzbar sein, auch wenn ein/e Benutzer/in nicht eingeloggt ist, aber den Login-Status erkennen und gegebenenfalls darum bitten, sich für ein bestimmtes Feature einzuloggen.

## **4 - Architektur**

Um die Lösungsansätze für die technischen Probleme der Plattform zu implementieren, wird die Plattform auf Basis einer Microservices-Architektur entwickelt. Bei dieser ist die Seitenarchitektur in mehrere kleine Services aufgeteilt. Jeder Service läuft in einem eigenen Prozess und kommuniziert mit anderen Prozessen mittels Protokollen wie HTTP oder WebSockets (Anil et al., 2018). Die Services funktionieren dabei auch unabhängig voneinander, auch wenn sie API's der anderen Services implementieren können. Das heißt zum Beispiel, dass der Bestandteil AuthService aus diesem Projekt auch eigenständig auf einen eigenen Server deployed werden kann (Anil et al., 2018). Dadurch lassen sich die einzelnen Bestandteile der Anwendung, die mehr Rechenleistung oder Netzwerkbandbreite benötigen, unabhängig voneinander horizontal skalieren (Rungta, o. D.). Horizontale Skalierung beschreibt dabei das Hinzufügen von maschinellen Ressourcen, also von zusätzlichen Servern, die parallel zueinander laufen. Die vertikale Skalierung beschreibt das Hinzufügen einer Leistung zu einer bestehenden Maschine durch RAM

und CPU (LeMay, 2021). Die Microservices-Architektur bietet bessere Möglichkeiten, einzelne Bereiche des Systems horizontal zu skalieren (Azure Application Architecture Guide, 2019). Dadurch können beispielsweise die Voice-Server, bei denen ich die höchste parallele Nutzer\*innenanzahl erwarte, unabhängig von der restlichen Applikation skaliert werden und dadurch mehr Benutzer\*innen in einem Diskussionsraum unterstützt werden. Bei einer monolithischen Architektur hängen alle Bestandteile miteinander zusammen, wodurch sich die gesamte Architektur nur horizontal skalieren lässt.

## **5 - Systembestandteile**

Teil dieser Architektur sind verschiedene Systembestandteile, die in den Services implementiert werden (siehe Abschnitt "Technische Implementierung"), um auf deren Basis die verschiedenen technischen Herausforderungen der App zu lösen. Ich wähle die meisten aufgrund meiner positiven Erfahrungen mit ihnen aus und habe versucht, die gesamte Applikation in Javascript oder Typescript zu schreiben, um möglichst viel Code zwischen den Anwendungen teilen zu können.

### **5.1 - Node.js**

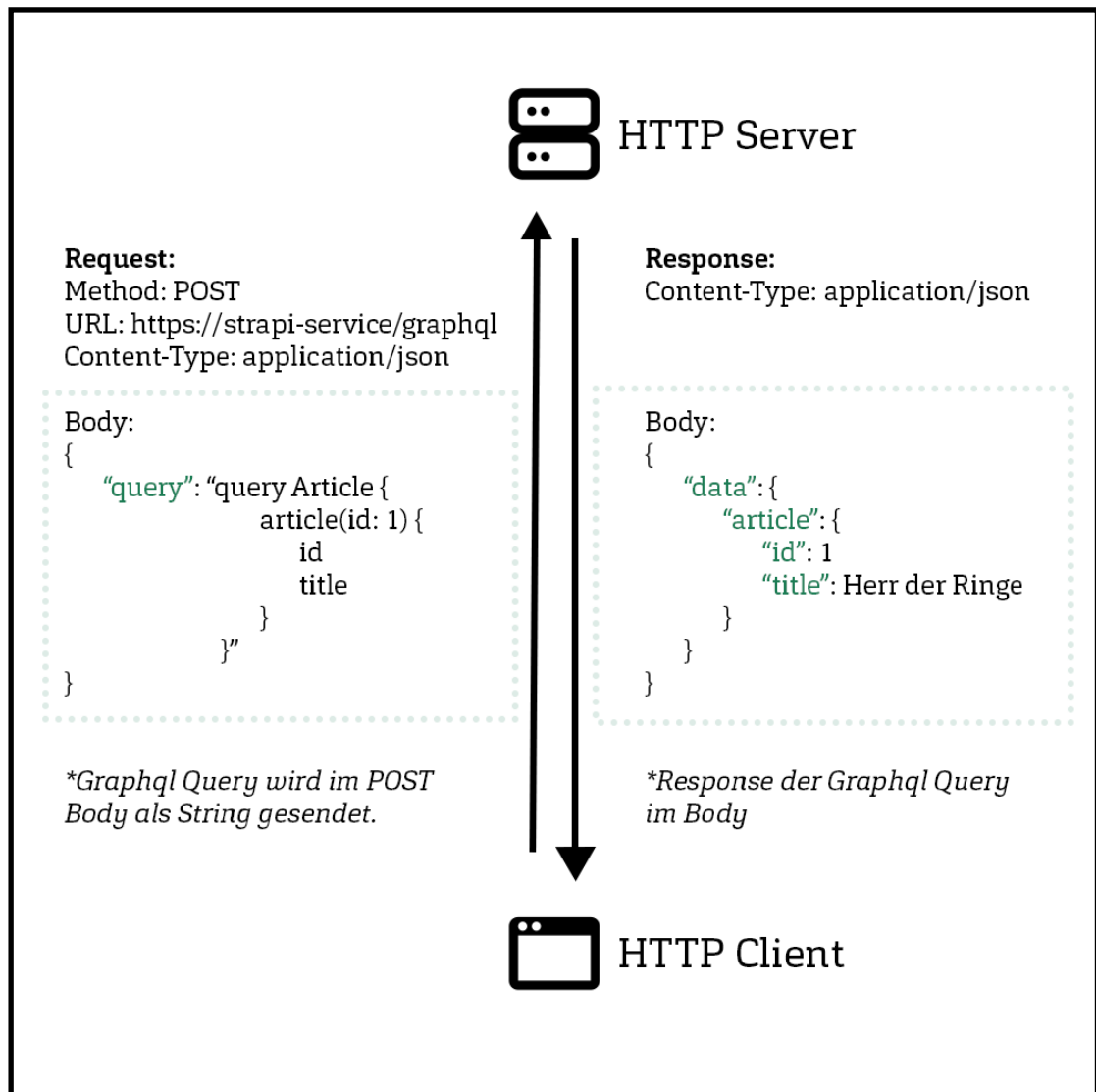
Node.js ist eine JavaScript-Runtime für das Backend. Diese basiert auf Chrome's Open-Source-V8-JavaScript-Engine und führt JavaScript-Code außerhalb des Browsers aus. Fast alle Anwendungen dieser Web-App benutzen Node.js, um ihren JavaScript-Code auf dem Server zu kompilieren.

## 5.2 - REST und GraphQL API's

REST beschreibt eine Representational-State-Transfer-API-Architektur, welche 2000 von Roy Fielding entwickelt wurde, um verschiedene Komponenten und Anwendungen einer Microservices-Architektur zu verbinden (Cloud Education, 2021). Ein Application-Programming-Interface (API) erlaubt es, eine Ressource einer anderen Anwendung aufzurufen oder mit dieser zu interagieren.

Eine REST API folgt dabei folgenden Grundprinzipien (Restfulapi, 2021). Sie besitzt eine Client-Server-Architektur, bei der Requests von einem Client an den Server mittels HTTP gestellt werden. Diese Anfragen sind zustandslos. Jede REST-Request funktioniert separat von Request zu Request und von Client zu Server. Die Schnittstelle ist einheitlich und sollte deshalb vom Client gecached werden können, sofern die Requests die gleichen Informationen enthalten würden und der Server die Anfrage als cacheable auszeichnet. REST-Requests werden durch ein einheitliches Interface beschrieben, welches die Funktionalität der Request beschreibt. Diese Ebenen-Architektur führt zu einer stabileren Basis, weil die einzelnen Services unabhängig voneinander funktionieren und nicht direkt mit einem anderen Service interagieren können.

GraphQL ist eine Query Syntax, die an einen Server mittels POST-REST-Endpunkten versandt wird und es ermöglicht, Daten als Query abzufragen und zu bearbeiten (Hasura, o. D.). Dabei werden die Daten direkt aus der Datenbank abgerufen, ohne dass ein REST-API-Endpunkt erzeugt werden muss. Das macht die Abfragen flexibler und weniger zeitintensiv in der Entwicklung.



(Abbildung 1: GraphQL Request durch REST API (Hasura, o. D.))

## 5.3 - Next.js und React

Next.js ist ein, von Vercel entwickeltes und auf React basierendes, Open-Source-Framework, welches mittels automatischer statischer Optimierung feststellt, ob eine Seite auf dem Server statisch vor- (Server Staticly Generated - SSG) oder bei einer Anfrage auf dem Server gerendert wird (Server Side Rendered - SSR) (Vercel, o. D.).

Auf diesen gerenderten Seiten und deren Daten initialisiert Next.js dann eine React-Anwendung im Browser (Client Side Rendered - CSR). Deshalb werden Next.js-Apps auch häufig als Hybrid-Applikationen beschrieben.



Denn sie enthalten sowohl komplett statisch generierte als auch serverseitig gerenderte Seiten, die beide mittels Hydration auf der Client-Seite volle Interaktivität erreichen, aber trotzdem ohne Javascript Inhalte anzeigen können (Next.js, o. D.).

Hydration beschreibt das Initialisieren einer React-Anwendung auf den bestehenden statisch generierten oder gerenderten HTML-Dateien, die an den Benutzer\*innen-Client geschickt werden (Gatsby, o. D.). Da sich nicht erst ein Javascript Framework im Browser initialisieren muss, um Inhalte anzuzeigen, sind Next.js-Anwendungen besonders schnell bis zum First-Contentful-Paint, also bis Inhalte der Seite angezeigt werden.












Da Webseiten-Crawler von Suchmaschinen außerdem statische Inhalte bevorzugen (Maldonado, 2020), haben Next.js-Anwendungen besonders gute SEO-Werte. Das heißt, dass sie besonders gut bei Suchmaschinen-Seiten-Rankings abschneiden.

Für die App bietet sich dieser Ansatz sehr gut an, denn die Applikation beinhaltet sowohl komplett statische Server-statically-generated Seiten, Server-Side-rendered Seiten, als auch Client-Side-rendered Seiten.

Komplett statische Seiten finden sich zum Beispiel in Kategorien, wie dem Impressum, welches keine sich dynamisch ändernden Daten besitzt und deshalb von Next.js als HTML statisch generiert werden kann. Die Startseite beinhaltet eine Literaturliste, welche bei jedem Aufruf der Seite aus dem Content-Management-System abgefragt werden muss und danach dynamisch gerendert wird. Die Diskussionsräume benötigen alle interaktiven Client-Side-rendered Features, die Javascript API's wie WebSockets und WebRTC bereitstellen, um Live-Inhalte zu ändern und würden ohne Javascript nicht funktionieren.

## 5.4 - Postgres

Als Datenbank benutze ich die relationale Datenbank Postgres. Postgres wurde 1986 von der University of California in Berkeley entwickelt und ist ein robustes Open-Source-Projekt, das von vielen großen Software-Unternehmen verwendet wird (Postgres, o. D.). Eine relationale Datenbank organisiert Datenstrukturen in Tabellen. Diese Tabellen können mittels Beziehungen gemeinsamer Daten miteinander verknüpft werden (IBM, 2019).

Column Name	#	Data type
 id	1	<a href="#">serial</a>
 email	2	<a href="#">varchar</a>
 password	3	<a href="#">varchar</a>
 displayName	4	<a href="#">varchar</a>
 providerId	5	<a href="#">varchar</a>
 provider	6	<a href="#">varchar</a>
 businessName	7	<a href="#">varchar</a>
 firstName	8	<a href="#">varchar</a>
 lastName	9	<a href="#">varchar</a>
 createdAt	10	<a href="#">timestampz</a>
 updatedAt	11	<a href="#">timestampz</a>

(Abbildung 1: Auth-Service User-Tabelle Datenstrukturen)

Eine Tabelle für einen User-Account des Auth-Services könnte dabei wie in Abbildung 1 beschrieben werden. Die Structured-Query-Language (SQL) ermöglicht es, verschiedene Tabellen und deren Abhängigkeiten einer laufenden Datenbank auszulesen und zu modifizieren.

	id	email	password	displayName	providerId	provider	businessName	fi
1	1	jonasleonha	\$2b\$10\$RUvcd	Jonas	0	local	[NULL]	[NU

(Abbildung 2: Auth-Service-User-Tabelle mit Daten für einen User-Account)

So speichert der Auth-Service einen User-Account in der User-Tabelle mittels SQL, indem er die benötigten Datenstrukturen in eine Tabelle einfügt (Abbildung 2). Postgres besitzt eine API-Integrationen mit allen gängigen Frameworks. Es unterstützt GraphQL-API's und ist deshalb universal einsetzbar.

## 5.5 - Strapi CMS

Strapi CMS ist ein Open-Source headless Content-Management-System, welches auf Javascript basiert. Ein traditionelles Content-Management-System speichert Seiteninhalte wie Texte, Bilder oder Videos und gibt diese als HTML zurück an den Browser. Die Inhalte lassen sich in einem grafischen User-Interface bearbeiten und veröffentlichen. Das hat den Vorteil, dass sich Inhalte einer Seite einfach und schnell anpassen lassen und auch nicht IT-affine Personen, wie zum Beispiel Redakteur\*innen Inhalte veröffentlichen können. Bei einem headless CMS ist dabei die Präsentationsschicht "Head" von den gespeicherten Inhalten "Body" getrennt. Die Inhalte des CMS werden für die Benutzer\*innen nicht als HTML und CSS gerendert und gesendet, sondern stattdessen an eine andere Anwendung mittels API's bereitgestellt und dort dargestellt (Wessling, o. D.). Strapi verwendet als Datenbank Postgres, erzeugt die Tabellen für gespeicherte Inhalte automatisch und stellt die Inhalte abschließend mittels API's bereit.

Für diese App bezieht beispielsweise die Next.js-Anwendung bei jeder Seitenanfrage die gepflegten Inhalte aus dem CMS mittels GraphQL-API's. Diese Inhalte werden dann im Next.js Frontend-Service für die Benutzer\*innen als HTML und CSS gerendert und bereitgestellt.

## **5.6 - Express.js**

Express.js ist ein minimalistisches auf Node.js basierendes Web-Framework für Javascript. Express ermöglicht es, Endpunkte für eine RESTful-API zu erzeugen und die Applikation dabei modular zu strukturieren. Dazu lassen sich Router-Module an Express anbinden, die nach verschiedenen URL-Parametern aufgerufen werden. Express ermöglicht es, jeden Teil der Request-Pipeline mittels externer Middleware-Module beliebig mit Funktionalitäten zu erweitern. Verfügbare Middlewares sind unter anderem Cookies, Sessions, Datenbanken, Benutzeranmeldungen, URL-Parameter, POST-Daten und Sicherheits-Header (MDN, 2021).

Der Authentication-Server und der WebSocket-Server der Microservices-Architektur sind unabhängige Express-Anwendungen, die mittels REST-API's oder anderen Protokollen mit anderen Services kommunizieren.

## **5.7 - WebRTC**

WebRTC ist ein Open-Source real-time Communications-Framework mit geringen Latenzen. Es ist Teil der HTML-5 Spezifikation und deshalb in allen Browsern als Javascript-API integriert (Levent-Levi, 2021). Alle modernen Plattformen unterstützen das Peer-To-Peer-Protokoll, mit dem auch Audio- und Video-Daten gesendet werden können (CanIUse, 2021). Peer-To-Peer bedeutet in diesem Fall, dass zwei Browser nach bestehender Verbindung keinen Server benötigen, um miteinander zu kommunizieren. Für den Verbindungsaufbau zweier Peer-To-Peer-Verbindungen benutzt WebRTC

das sogenannte Interactive Connectivity Establishment (ICE) Framework, welches alternative Kandidaten bereit stellt, sofern eine direkte Verbindung mittels Netzwerkadressen nicht funktioniert. Diese können beispielsweise fehlschlagen, wenn ein Netzwerk sich hinter einer Firewall oder einer Network-Address-Translation (NAT) befindet, welche verhindert, dass die direkte IP-Adresse des Clients weitergegeben wird (WebRTC.Ventures, 2019).

Diese ICE-Kandidaten beinhalten sowohl einen Session-Traversal-Utilities for NAT Server (STUN Server), als auch einen Traversal-Using-Relays around NAT Server (TURN Server) (WebRTC.org, 2020). Der STUN Server versucht die öffentliche IP-Adresse des Clients zu identifizieren, sofern diese durch ein NAT maskiert ist (Cloudflare, o. D.).

Der TURN-Server gibt die gesendeten Daten des Clients als Man-in-the-Middle-Server weiter, sofern die IP-Adressen des Clients nicht ermittelt werden können und somit eine direkte Verbindung nicht möglich ist (WebRTC.org, 2019).

Während des Verbindungsaufbaus zweier Clients benötigt eine WebRTC-Verbindung einen Server. Dabei sendet ein Client für den Austausch ein Session-Description-Protocol (SDP) von Client zu Server, der diese Daten an den anderen Client vermittelt. Nun besitzen beide Client's das Session-Description-Protocol des jeweils Anderen (MDN, 2021).

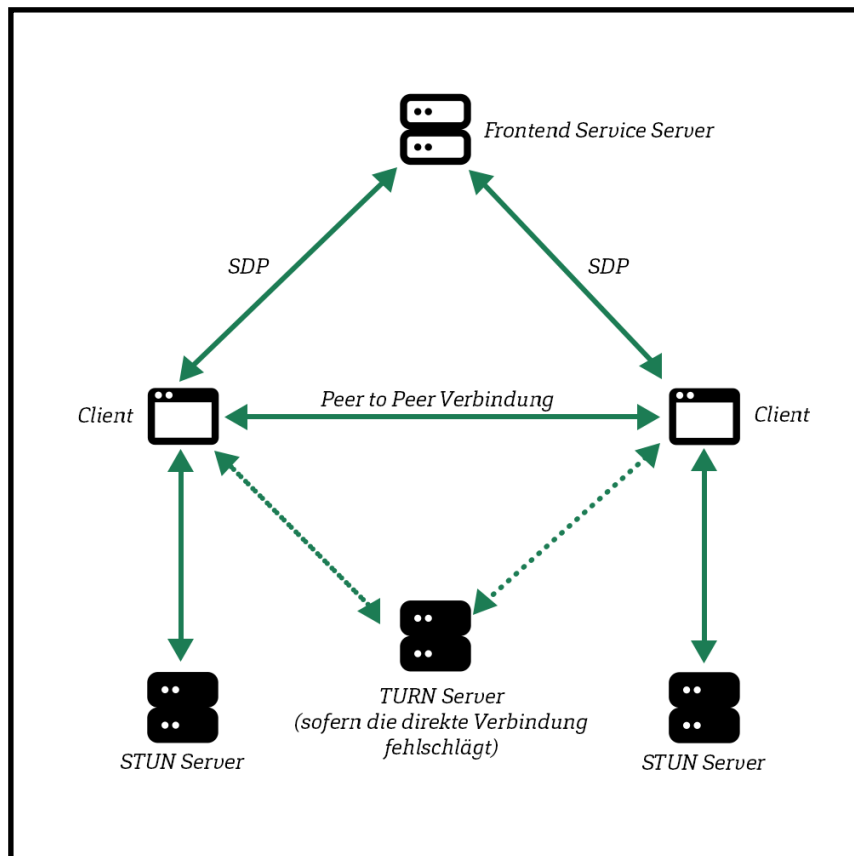
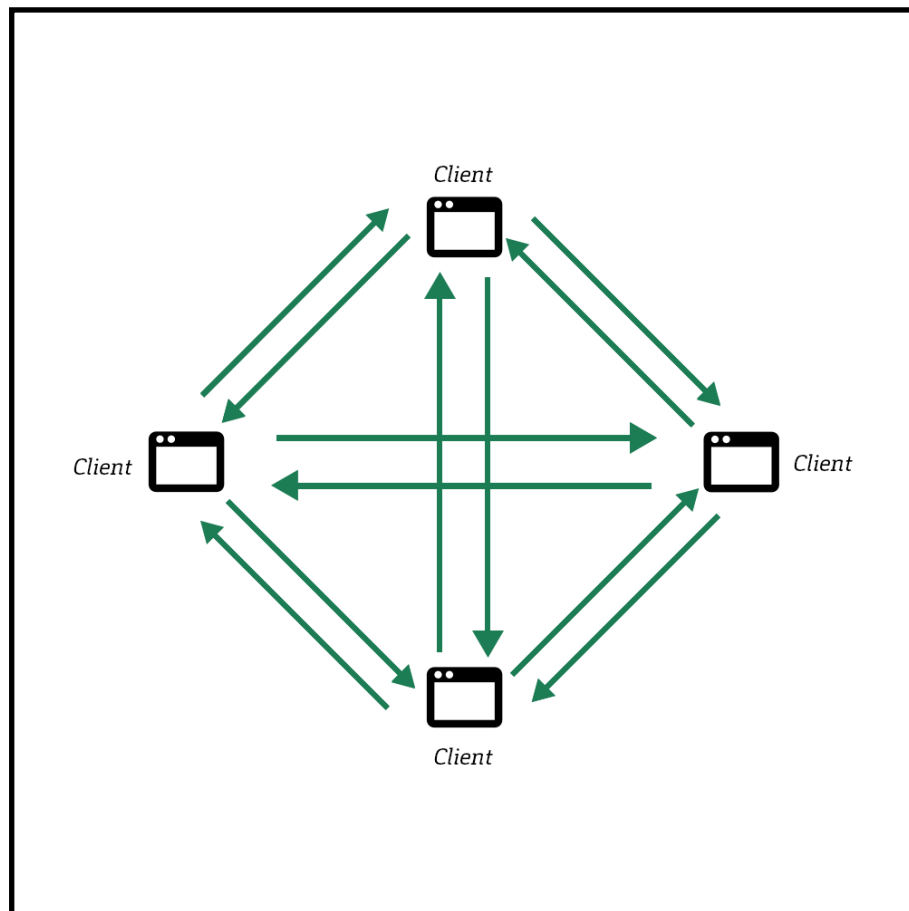


Abbildung 1: ICE Peer to Peer mit STUN und TURN Server (WebRTC.org, 2020).

Ein Session-Description-Protocol beinhaltet dabei neben ICE-Kandidaten, die als Fallback für die Verbindung benutzt werden sollen, andere Konfigurationsdaten für eine Peer-To-Peer-Verbindung. Diese beinhalten Netzwerkadressen und die Information, ob der Client nur Video- oder Audiodaten sendet, sowie andere verbindungsrelevante Meta Daten (Mukit, 2018).

### 5.7.1 - Mesh-Architektur

In der klassischen Mesh-Architektur verbinden sich alle Teilnehmer\*innen direkt mittels WebRTC-Audio und oder -Video mit den anderen Teilnehmer\*innen. Der Vorteil dieser Architektur ist, dass eine Applikation keine Server bereitstellen muss, da alle Teilnehmer\*innen untereinander Verbindungen miteinander aufbauen (Fosdem, 2018)].



(Abbildung 1: WebRTC Mesh Verbindungen ohne zentralen Server.)

Dadurch haben Teilnehmer\*innen eine möglichst geringe Latenz und es entstehen keine Sicherheits- und Privatsphäreprobleme durch einen Man-in-the-Middle-Server. Durch eine bilaterale Verbindungsmenge ( $N - 1$ ) pro Teilnehmer/in skaliert die gesamte Upload- und Download-Rate der Konferenz exponentiell mit  $n * (n-1)$  Verbindungen (Venema, 2021). Aus der Sicht einzelner Teilnehmer\*innen skaliert die Verbindungsmenge allerdings linear mit  $(n - 1)$  Verbindungen.  $N$  ist dabei die totale Anzahl an Konferenzteilnehmer\*innen. Mit zehn Teilnehmer\*innen muss jeder Client neun Verbindungen hochladen und die Daten der anderen Teilnehmer\*innen downloaden. Geht man von 1.5 Mbps Upload- und Download-Rate pro WebRTC-Verbindung aus, so würde bei unserem Beispiel eine Upload-Rate von 15Mbps benötigt werden (Venema, 2021).

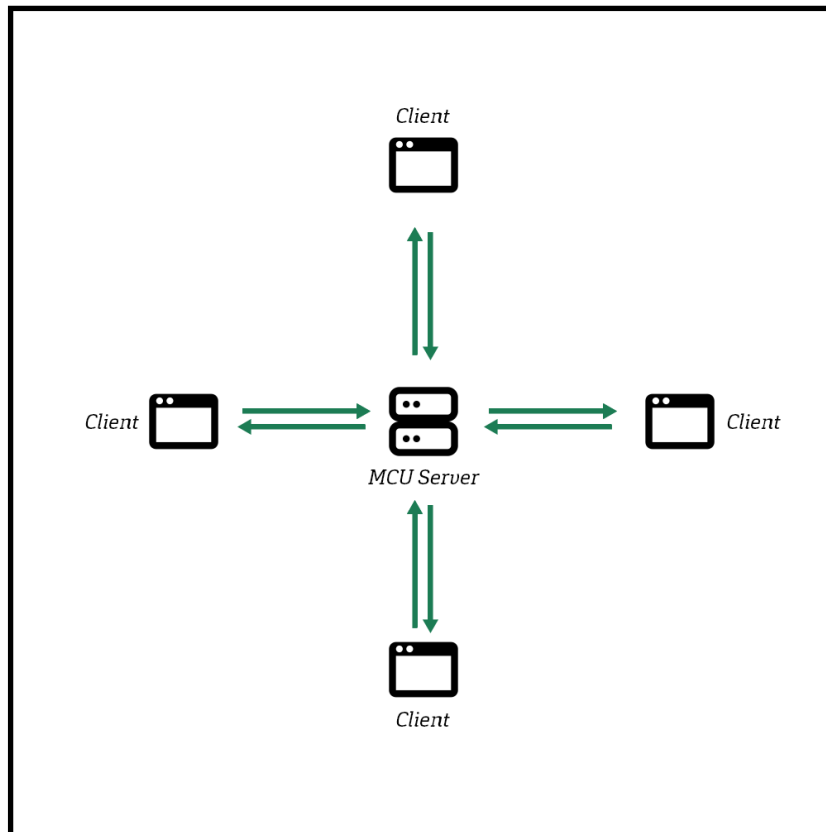
Ab 14 Teilnehmer\*innen würde also die durchschnittliche Upload-Rate in Deutschland, welche ca. 20Mbps beträgt, für die Konferenz nicht mehr ausreichen (Netzwelt, o. D.). Mit der durchschnittlichen Download-Rate könnten immer noch 40 Teilnehmer\*innen unterstützt werden. Zusätzlich muss jede aufgebaute Verbindung beibehalten, verschlüsselt, entschlüsselt sowie für Video und Audio encodiert und decodiert werden. Dadurch werden Client-CPU's stark belastet. Dabei ist vor allem das Encodieren der Video- und Audio-Daten besonders belastend für den Arbeitsspeicher und bereits für mehr als vier Teilnehmer\*innen nicht mehr ausreichend. Für Gruppenkonferenzen ab vier Teilnehmer\*innen müsste also eine andere Architektur gewählt werden.

---

#### 5.7.2 - MCU - Multipoint-Conferencing-Unit

Eine Multipoint-Conferencing-Unit (MCU) beschreibt einen zentralen WebRTC-Server, der die Audio- und Videostream-Daten aller Teilnehmer\*innen einer Konferenz zu einem Datenstream auf dem Server zusammenführt. Aus diesen Datenstreams erstellt die Multipoint-Conferencing-Unit dann einen Video- und Audio-Stream, der für alle Teilnehmer\*innen der Konferenz identisch ist (Sime, 2020).



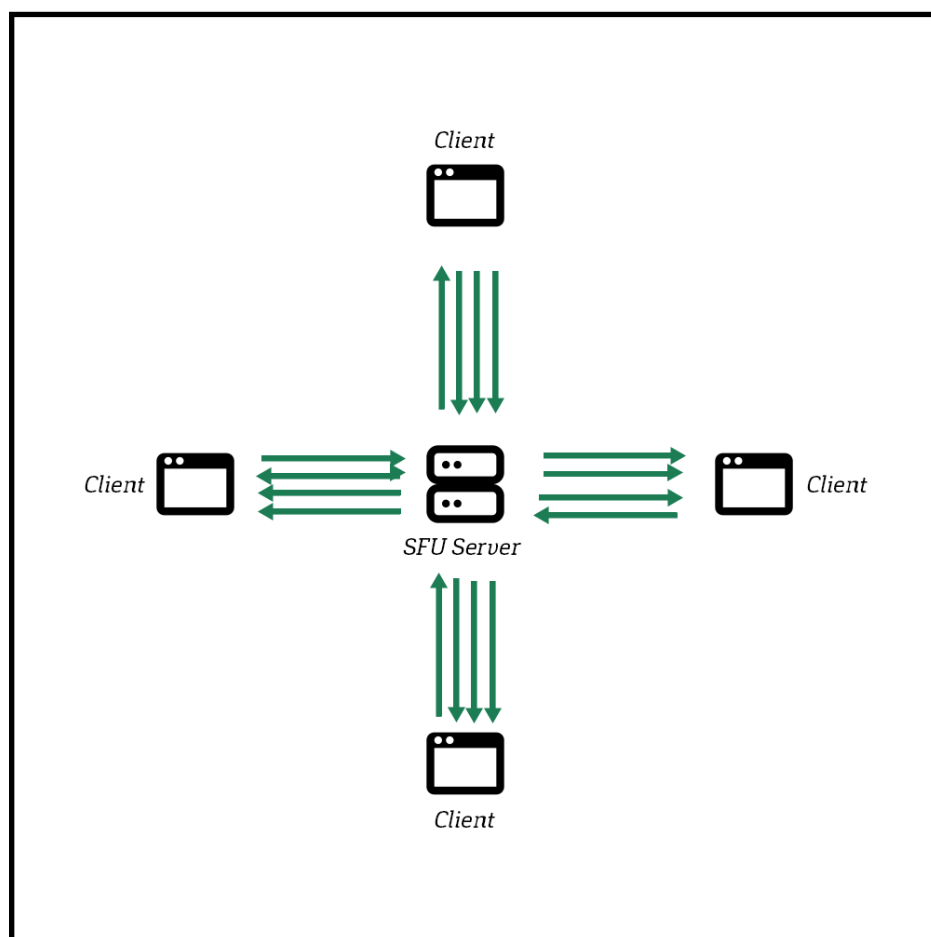


(Abbildung 1: Verbindungs Schema zwischen Clients und MCU (Sime, 2020))

Dadurch hat jeder Client nur eine Upload- und Download-Verbindungsmenge von (1), welche die Client-Systeme und Download-sowie Upload-Raten kaum belastet. Da alle Streams der Teilnehmer\*innen zusammengeführt werden, entfällt bei einer Multipoint Conferencing Unit die Möglichkeit zur variablen Anpassung der einzelnen Streams. Weder das Layout, noch die Lautstärke der einzelnen Streams lassen sich vom Client anpassen. Eine Multipoint-Conferencing-Unit ist durch die Datenverarbeitung der Streams auf dem Server deutlich belastender für die CPU des Servers, als die anderen Architekturen und erzeugt dadurch deutlich höhere Latenzen und Kosten für die Skalierung des Systems (Allen, 2021).

### 5.7.3 - SFU - Selective-Forwarding-Unit

Eine Selective-Forwarding-Unit (SFU) ist eine WebRTC-Architektur, bei der ein Teil der Stream-Verarbeitung auf einen SFU-Server verlagert wird. Teilnehmer\*innen verbinden sich mit der Selective-Forwarding-Unit, welche dann die Streams ohne weitere Verarbeitung an die anderen Teilnehmer\*innen weiterleitet (WebRTC Glossary, 2021). Dadurch wird die Latenz der Teilnehmer\*innen kaum beeinflusst und es kommt zu einer geringen CPU-Auslastung auf der Seite des Servers, da dieser nur die Verbindungen weiterleitet (Allen, 2021). Da ein Client immer noch den Stream aller Teilnehmer\*innen erhält, lässt sich das Stream-Layout außerdem noch immer dynamisch anpassen und einzelne Clients sich stummschalten.



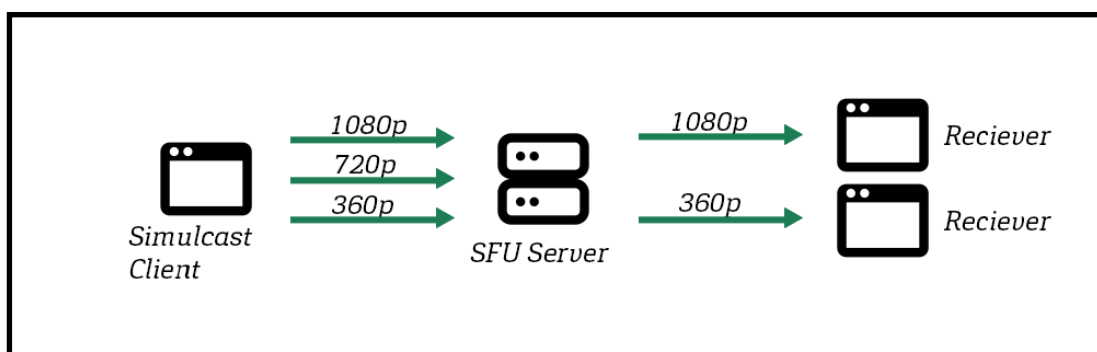
(Abbildung 2: Verbindungs Schema zwischen Clients und SFU)

Statt eines  $(N - 1)$  Uploads und  $(N - 1)$  Downloads besitzt eine Selective-Forwarding-Unit-Architektur nur einen (1) Upload und  $(N - 1)$  Downloads und skaliert dadurch immer noch mit  $(N - 1)$  Downloads linear. Durch den Wegfall der Upload-Streams des Clients entfällt allerdings die Hauptauslastung des CPU's, die multiple Upload-Encodierung des Streams (Allen, 2021). Deshalb ist diese Architektur vor allem auf die Download-Rate des Clients beschränkt (Allen, 2021).

---

#### 5.7.4 - Simulcast- und Scalable-Video-Coding

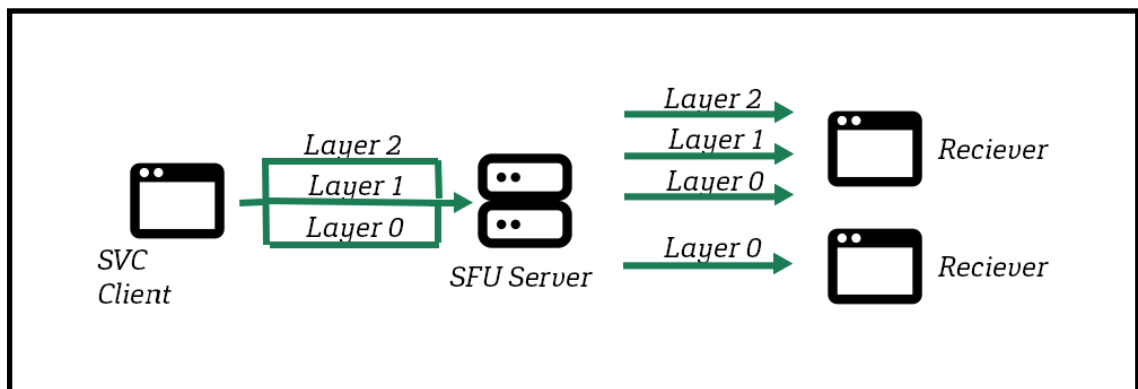
Um die Download-Streams der Selective-Forwarding-Unit für den einzelnen Download-Client zu optimieren werden vom Client mehrere Datenraten WebRTC-Stream-Qualitäten an die Selective-Forwarding-Unit geuploadet. Dieses Verfahren wird Simulcast genannt. Simulcast ermöglicht es, die Audio- und Video-Qualität entsprechend der bereitstehenden Download-Datenrate anzupassen, indem die Selective-Forwarding-Unit die Upload-Streams des Clients entsprechend der Internetverbindung eines Download-Clients weiterleitet (Levent-Levi, 2019)].



(Abbildung 1: Simulcast Upload und Downlaod mit einem SFU Server

(Levent-Levi, 2019))

Ein weiteres Verfahren für die Download-Optimierung von Selective-Forwarding-Units Videos ist der VP9 Codec. Dieser Scalable Video Coding (SVC) Codec ermöglicht es, ein Upload-Video in mehrere Ebenen aufzuteilen, welche dann von einem Client zu einem Video zurückgebaut werden können. Dadurch entfällt das multiple Encodieren eines einzelnen Video- und Audio-Streams in verschiedene Video-Qualitäten und der Client CPU wird noch weniger belastet. (Levent-Levi, 2019)].



(Abbildung 1: SVC Upload und Download mit einem SFU Server

(Levent-Levi, 2019))

---

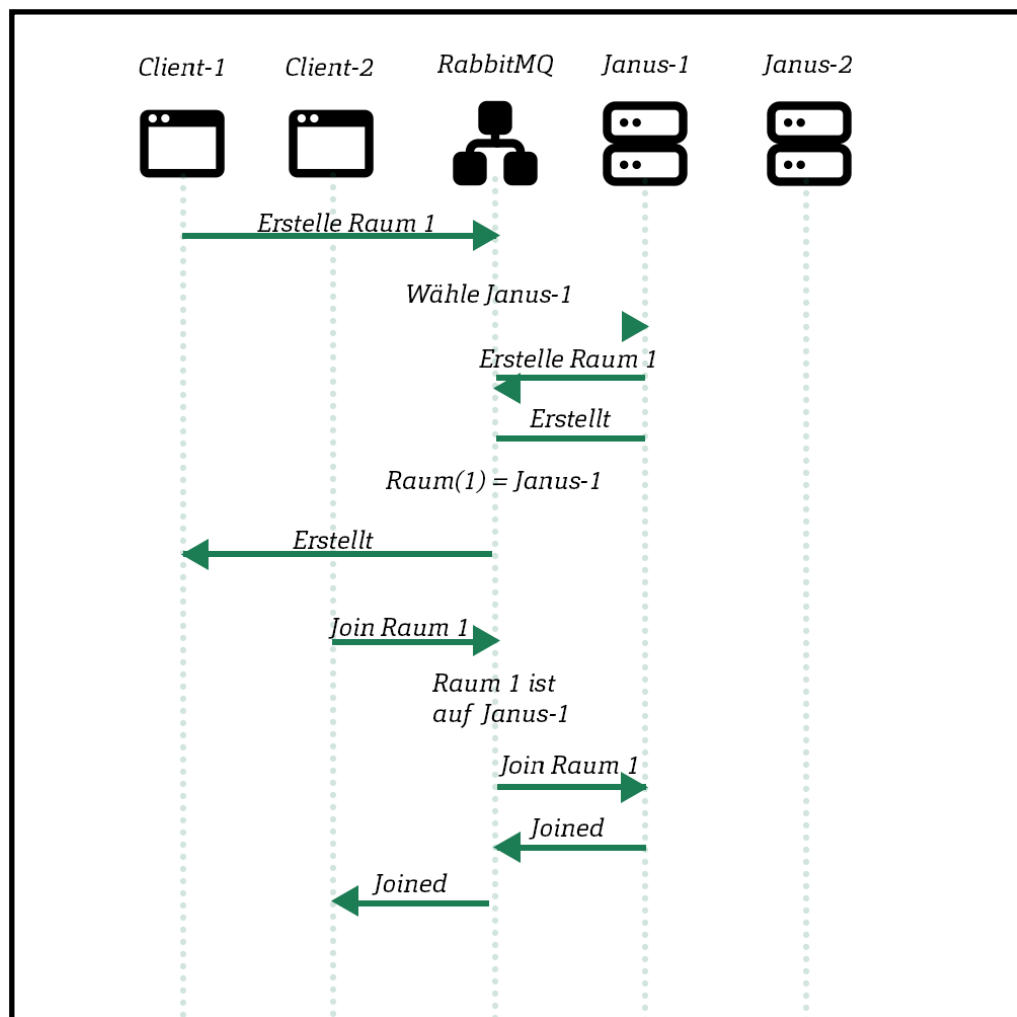
### 5.7.5 - Horizontale Skalierung

Moderne Konferenzsysteme wie *Google Meet* oder *Discord* müssen Hunderte oder Tausende simultane WebRTC-Konferenzteilnehmer\*innen unterstützen. Dazu müssen alle Systembestandteile horizontal skaliert und die Belastung auf alle Server aufgeteilt werden.

Da ein einzelner SFU- oder MCU-Server sehr große Verbindungsmengen nicht verarbeiten kann, wird eine extra Kommunikationsebene benötigt, die die Belastung einer großen Konferenz auf mehrere Server-Instanzen aufteilt.

Ein einfacher Ansatz wäre der Einsatz eines Message-Brokers wie RabbitMQ. Dieser bestimmt, welche Server-Instanz bei Erstellung einer

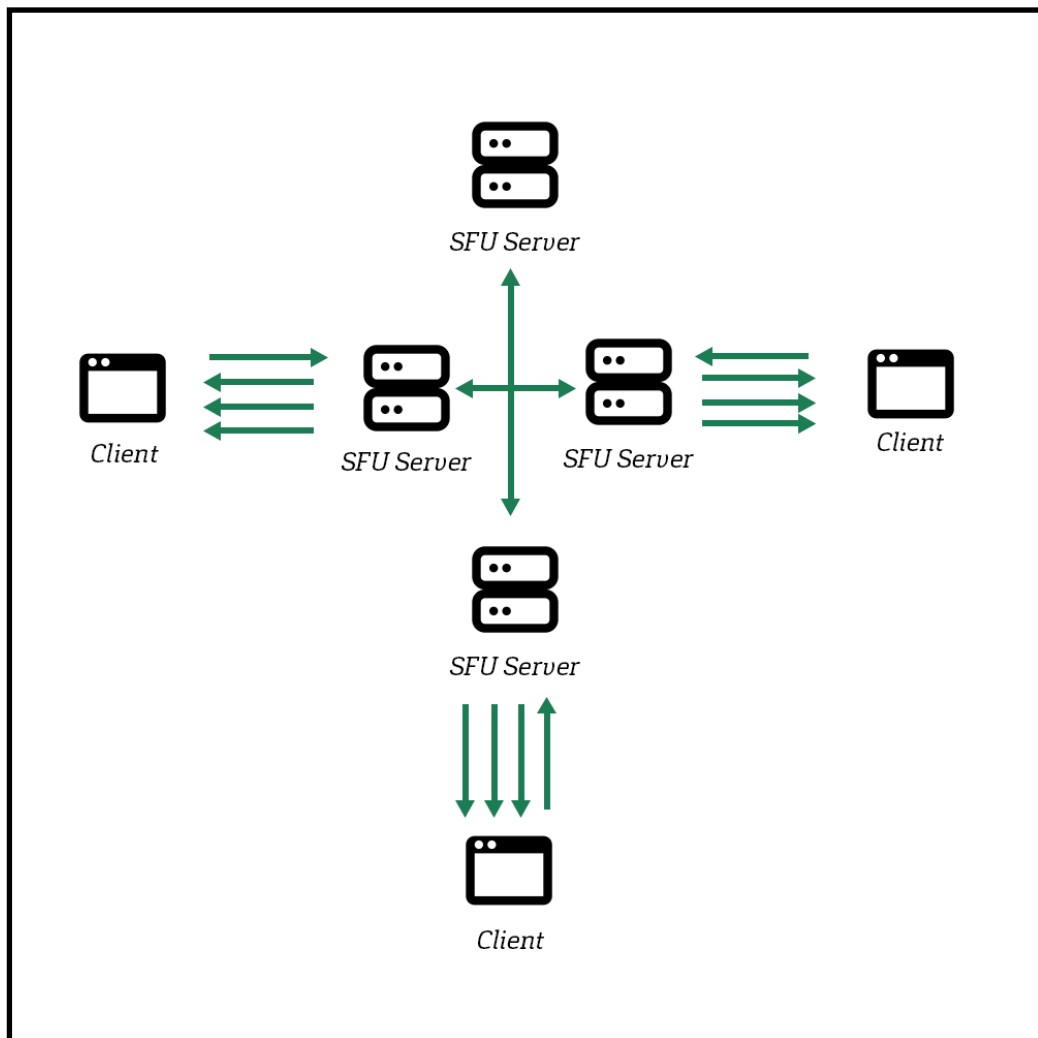
Konferenz am wenigsten ausgelastet ist und leitet den Client und alle folgenden Join-Room-Anfragen an die jeweilige Server-Instanz weiter (CommCon 2018).



(Abbildung 1: Message Broker (MRB) verteilt 'join-room' Anfragen an Janus Instanzen. (CommCon 2018))

Außerdem lassen sich mehrere SFU-Server cascadierend horizontal skalieren, sofern die maximale Kapazität eines Servers erreicht ist.

Dabei werden mehrere SFU-Server miteinander verbunden und Clients wird es erlaubt sich mit einem der SFU-Server zu verbinden, diesem den Upload-Stream zu übertragen und von einem komplett anderen SFU-Server zu downloaden. (Siehe Abbildung 2.) (SFHTM5, 2017).



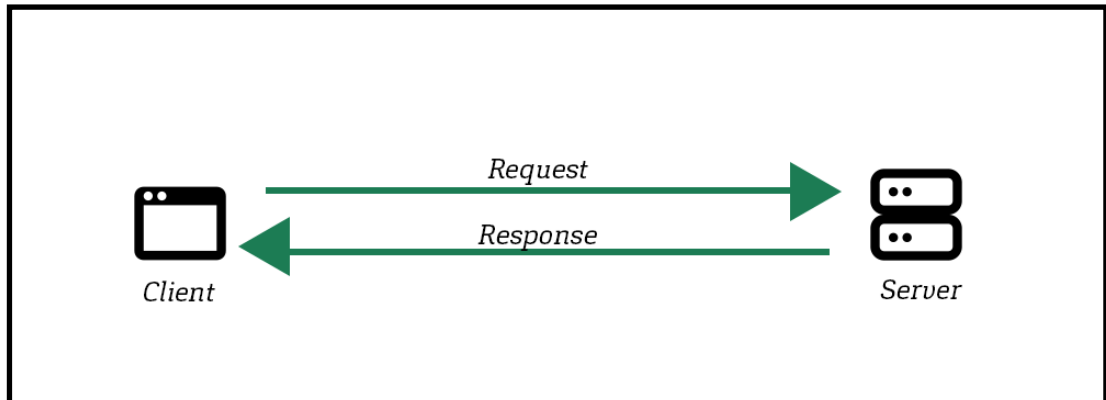
(Abbildung 2: Architektur von cascadierenden horizontal skalierten SFU-Servern (SFHTM5, 2017))

## 5.8 - WebSockets

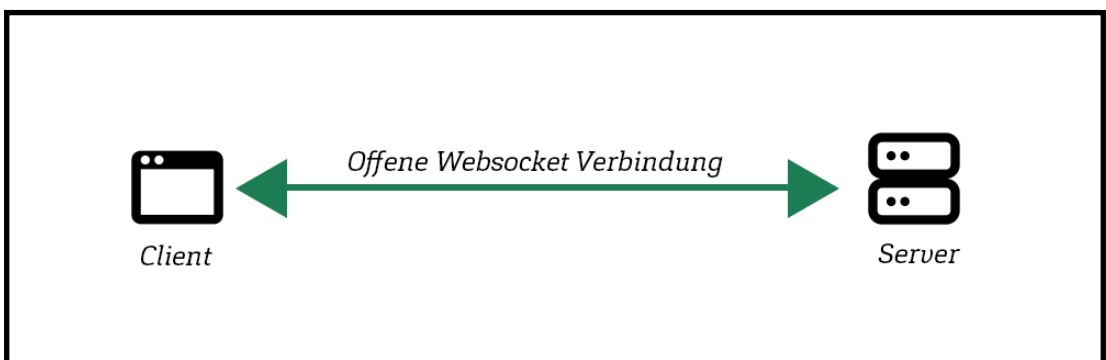
Das WebSocket-Protokoll ist ein Kommunikationsprotokoll, das mittels TCP-Verbindung beidseitige Kommunikationskanäle bereitstellt. Diese Zwei-Wege-Verbindungen zwischen zwei Endpunkten werden als Sockets bezeichnet und ermöglichen den simultanen Austausch zwischen einem Socket-Host-Server und einem Socket-Client (IONOS, 2020).

Dabei unterscheiden sich WebSockets von WebRTC dadurch, dass WebSockets immer einen Socket-Host-Server benötigen und WebRTC Peer-To-Peer-Verbindungen zwischen zwei Browser Clients ermöglicht (Levent-Levi, 2019). Dazu benötigt WebRTC die Möglichkeit, ein Session-Description-Protocol zwischen Client-Server und Client weiterzugeben. Als

Session-Description-Protocol-Server lässt sich gut ein Socket-Host-Server verwenden, der das Session-Description-Protocol der WebRTC-Clients austauscht.



(Abbildung 1: Eine HTTP-Verbindung verwendet ein Request-Response-Model und der Client muss zunächst eine Anfrage senden, bevor der Server antworten kann. (IONOS, 2020).)



(Abbildung 2: Eine offene WebSocket-Verbindung (IONOS, 2020))

In einer offenen Socket-Verbindung können, nach dem initialen Handshake und einem aktiven Verbinden, beidseitig Daten versandt werden, ohne dass ein Client vorher eine Anfrage senden muss. Es können sich außerdem mehrere Socket-Client's mit dem Socket-Host verbinden, welcher gesendete Nachrichten an alle Socket-Client's mittels Broadcast-Nachricht weiterleitet.

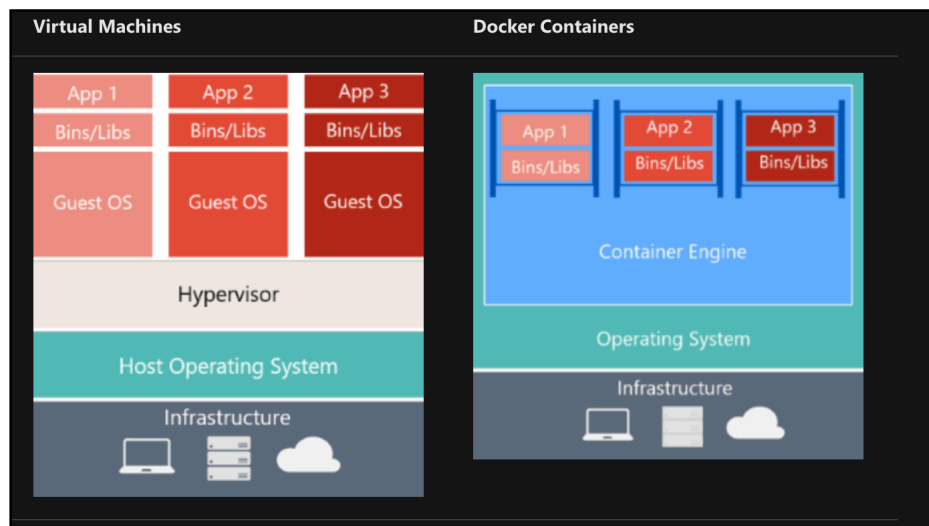
Besuchen Benutzer\*innen einen Diskussionsraum, erstellt der Frontend Service eine neue Socket-Verbindung mit dem Socket-Host-Server-Service. Zwischen Client und Server findet ein Connection-Handshake statt, bei dem sich der Client und der Server miteinander verbinden und einen Socket öffnen. In diesem Socket lassen sich jetzt live bilateral Daten austauschen.

## **5.9 - Docker**

Docker-Containerisierung ist eine Form der Virtualisierung des Betriebssystems. Die Docker-Plattform verpackt eine Anwendung und alle ihre Abhängigkeiten als Image, also quasi als Bauanleitung zum Nachstellen der Entwicklungsumgebung inklusive der Abhängigkeiten der Anwendung. Mittels dieser Bauanleitung kann Docker eine Container-Anwendung starten. Dieser Container ermöglicht es, die Applikation auf jedem Betriebssystem oder in jeder Entwicklungsumgebung laufen zu lassen (Microsoft, 2018).

Im Gegensatz zu einer klassischen virtuellen Maschine, enthalten die Container-Anwendungen neben den Abhängigkeiten kein vollständiges Gastbetriebssystem. Sie teilen den Betriebssystem-Kernel mit anderen Containern und laufen als isolierte Prozesse im Container-Engine-Userspace des Host-Betriebssystems. Dadurch lassen sich Docker-Container einfach horizontal skalieren, da sie nicht den Overhead eines eigenen Gastbetriebssystems mit sich führen und deutlich weniger Speicherplatz benötigen (Arora, 2019).





(Abbildung 1: Gegenüberstellung einer Virtuellen Maschine und einer Container Engine wie Docker (Microsoft, 2018))

Die gesamte *Microservices*-Architektur der App ist *dockerisiert*. Das bedeutet, dass jeder Service ein eigenständiger Container ist und mit einem Dockerfile gebaut werden kann.

Mittels Programmen zu Container-Orchestrierung wie Kubernetes sollte es möglich sein, die Anwendung horizontal auf mehrere Maschinen zu deployen und entsprechend der Anfragen von Benutzer\*Innen zu skalieren.

## 5.10 - SFU Voice server - Janus

Um die Teilnahme von einer möglichst großen Anzahl an Benutzer\*innen in Diskussionsräumen zu ermöglichen, implementiere ich den WebRTC-Server Janus. Janus ist ein Open-Source-C-WebRTC-Server, der funktional durch Plugins erweiterbar ist.

Mit API's, welche die Plugins bereitstellen, lassen sich der Janus-Server oder die WebRTC-Verbindungen manipulieren. Die von Janus bereitgestellte Javascript-Bibliothek Janus.js implementiert oder vereinfacht dabei die clientseitige Interaktion mit den Plugin-API's. Janus

stellt unter anderem ein, durch die WebSocket-API steuerbares, VideoRoom-Plugin bereit. Dieses ermöglicht es, Janus als Selective-Forwarding-Unit für Video- und Audio-Calls zu benutzen, die auf einen Raum beschränkt sind.

## 5.11 - User-Authentication - Jason-Web-Tokens

Damit Benutzer\*innen in Diskussionräumen identifiziert werden können, müssen sie sich in die Anwendung mit ihren persönlichen Daten einloggen. Die Auth-Service-Express-Anwendung ermöglicht das Registrieren und Einloggen von Benutzer\*innen mittels Cookies und Jason-Web-Tokens. Der Auth-Service stellt eine Authentifizierung-REST-API bereit. Unter /auth/login können sich Benutzer\*innen einloggen und erhalten als Response einen Jason-Web-Token-Cookie, der die User-Daten beinhaltet.

The image shows a web interface for decoding a JWT token. It is divided into two main sections: 'Encoded' and 'Decoded'.

**Encoded:** This section has a label 'PASTE A TOKEN HERE' and contains a long, multi-line string of base64-encoded characters, which is a JWT token.

**Decoded:** This section has a label 'EDIT THE PAYLOAD AND SECRET'. It displays the decoded components of the token:

- HEADER: ALGORITHM & TOKEN TYPE:** Shows a JSON object: `{ "alg": "HS256", "typ": "JWT" }`.
- PAYLOAD: DATA:** Shows a JSON object containing user information: `{ "data": { "id": 1, "email": "jonasleonhardf1@gmail.com", "password": "S2bS10$Jv.4kbCAYgD0jrCXLpxm07WuLshY40GVXL2qBS06UDuPZX FV9c2", "displayName": "chatuser123", "providerId": "0", "provider": "local", "businessName": null, "firstName": null, "lastName": null, "createdAt": "2021-06-17T11:01:01.892Z", "updatedAt": "2021-06-17T11:01:01.892Z" }, "iat": 1625941743, "exp": 1626546543 }`.
- VERIFY SIGNATURE:** Shows the formula for verifying the signature: `HMACHA256( base64UrlEncode(header) + ".", base64UrlEncode(payload), your-256-bit-secret )`. There is a checkbox labeled 'secret base64 encoded' which is currently unchecked.

(Abbildung 1: Gegenüberstellung von encodiertem JWT und decodiertem JWT (JWT.io, o. D.))

Der Jason-Web-Token-Standard ist eine Methode, um Informationsübertragung digital zu signieren. Zur Signierung verwendet man einen einfachen Schlüsselsatz oder einen open und private SSL-Key. Diese Informationen können danach auf ihre Vertrauenswürdigkeit mit dem einfachen Schlüsselsatz oder open SSL-Key verifiziert werden (JWT.io, o. D.).

Das Jason-Web-Token (JWT) besteht aus drei Bestandteilen, die mit einem Punkt getrennt sind und per Base64 encodiert sind (JWT.io, o. D.). Der Header beschreibt die Art des Signier-Algorithmus (Schaubild: rot). Die Payload beschreibt die Daten, die verifiziert werden sollen, in diesem Fall der/die Benutzer/in (Schaubild: pink). Der dritte Teil ist die Signatur (Schaubild: blau). Die Signatur wird aus dem Header und der Payload und einem einfachen Schlüsselsatz oder private SSL-Key erstellt und mit einem einfachem Schlüsselsatz oder open SSL-Key ebenfalls auf ihre Vertrauenswürdigkeit geprüft. Loggen sich Benutzer\*innen im Auth-Service ein, erstellt der Auth-Server ein Jason-Web-Token, welches die Nutzerdaten des jeweiligen Kontos beinhaltet.



	Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
http://localhost:3000	jwt	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkYXRhbnp7ImkljoxLjJibWpCbCI6I...	localhost	/	Tue, 17 Jan 2073 ...	559	false	false	Lax	Sun, 11 Jul 2021 1...

(Abbildung 2: Ein, als Cookie gespeichertes, encodiertes Jason-Web-Token)

Das Jason-Web-Token wird danach vom Auth-Server als Cookie zurück an den Browser geschickt. Dort wird nun bei jeder Anfrage das Jason-Web-Token als Cookie mitgesendet, wodurch Benutzer\*innen auf der Server-Seite mittels Schlüsselsatz oder SSL-Key verifiziert werden können.

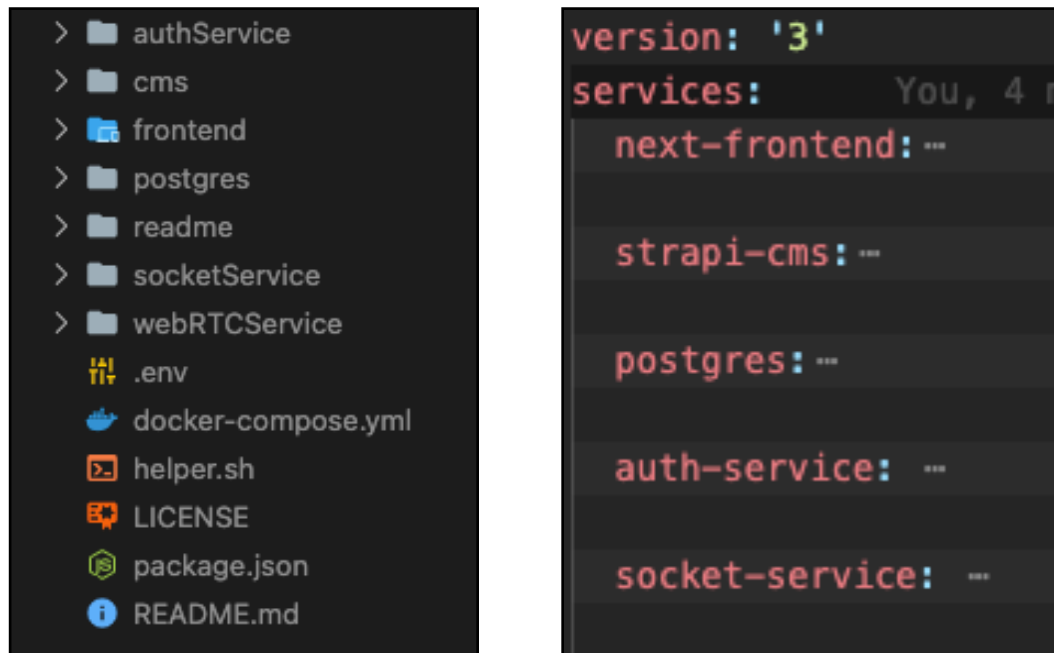
## 5.12 Ant Design

Für das User-Interface der Frontend-Applikation benutze ich Ant-Design (Antd). Antd ist ein Design-System und eine Open-Source-Komponenten-

Bibliothek. Es beinhaltet viele gängige Komponenten, wie Formulare, Alarmer, Modale oder Buttons und lässt sich mittels Less kompilieren. Less ist eine abwärts kompatible Spracherweiterung für CSS, die sich mittels Javascript zu CSS-Dateien kompilieren lässt. Die Antd-Less-Stylesheets beinhalten dabei eine Reihe an Design-Tokens, die sich vor dem Kompilieren zu CSS-Dateien anpassen lassen (Ant Design, o. D.). Dazu zählen zum Beispiel die Primärfarben, der Randradius oder die Randfarben der Komponenten. Das Ändern der Design-Tokens ermöglicht dabei den Style der App nach Belieben anzupassen. Die Ersparnis der Entwicklung von gängigen Komponenten ermöglicht es Entwickler\*innen außerdem, den Fokus auf die Kernfunktionen der App zu legen. Ein Design-System bietet dabei eine Vorlage für ein stimmiges User-Interface, welches bereits gängige Design-Schwierigkeiten wie Zugänglichkeit und bewährte Praktiken beinhaltet.

## **6 - Technische Implementierung**

Das Root-Verzeichnis der Applikation ist in Ordner aufgeteilt, welche jeweils einen Microservice bereitstellen. Jeder Microservice beinhaltet Dockerfile-Build-Instruktionen, um den Service als Docker-Container zu bauen. Mittels Docker-Compose lässt sich die gesamte Microservices-Architektur starten und die Applikation aufrufen.



(Abbildung 1 Links: Root-Ordner-Struktur der Applikation. Abbildung 2 Rechts: Docker-compose.yml Services im Überblick.)

## 6.1 - Postgres-Service

Der Postgres-Service baut einen Docker-Container, der eine PostgreSQL-Datenbank auf Port 5432 laufen lässt. Die Daten zur Authentifizierung dieser Datenbank lassen sich in der .env Root-Datei anpassen. Die anderen Microservices verbinden sich mit dieser Datenbank, um ihre Daten persistent zu speichern.

## 6.2 - Auth Service

Der AuthService ist ein Docker-Container mit einer Express-App, der eine REST-API zu User-Authentifizierung mittels Jason-Web-Token-Cookies bereitstellt. Der Server implementiert vier Endpunkte. /login, /register, /logout und /token. Der Endpunkt /login überprüft, ob ein User-Account mit den Daten des Request-Body bereits in der Postgres-Datenbank existiert. Sofern dies zutrifft, erstellt der Endpunkt ein valides Jason-Web-Token und

schickt dieses als Cookie zurück an den Browser. Der Endpunkt `/register` erstellt einen neuen User-Account und sendet den dazugehörigen Jason-Web-Token als Cookie zurück. `/logout` invalidiert das aktive Jason-Web-Token des Clients und loggt damit den Client aus der Applikation aus. `/token` validiert das, als Cookie gesendete, Jason-Web-Token und sendet diese Validität zurück. Sofern also ein anderer Service überprüfen möchte, ob ein überliefertes Jason-Web-Token valide ist und der Client eingeloggt ist, ruft der Service den `/token`-Endpunkt des AuthServices auf.

### 6.3 - CMS-Service

Der CMS-Service implementiert einen Docker-Container mit einem Strapi-Content-Management-System, welches sich auf `localhost:1337/admin` aufrufen lässt. Dort lassen sich die Literaturlisten der Applikation in einem grafischen User-Interface anpassen und bearbeiten.

### 6.4 - Socket-Service

Der Socket-Service besteht aus einer Express-App, die einen WebSocket-Server startet. Der WebSocket-Server implementiert eine WebSocket-API, die mittels JSON-Requests vom Typ `{ type: string, data: any }` gesteuert werden. So kann das Frontend beispielsweise einen neuen WebSocket-Raum eröffnen, indem das WebSocket

`{ type: 'join-room', data: { roomUuid } }` als Request sendet.

Mittels `{ type: 'message-room', data: msg }` sendet man eine Nachricht an alle Teilnehmer\*innen im selben Raum. Während der `'join-room'` Endpunkt aufgerufen wird, erstellt der Socket-Service mittels WebSocket-API der Janus-Selective-Forwarding-Unit einen neuen webRTC-Raum mit der selben ID und ermöglicht es so, dem Frontend eine WebRTC-Verbindung zu Janus aufzubauen.

## 6.5 - WebRTC-Service

Der WebRTC-Service enthält ein janus.sh Bash Script welches Janus installiert, baut und startet. Mittels 'bash janus.sh setup' installiert das Script alle notwendigen Dependencies von Janus auf dem System. Danach kann der Janus-Server mittels 'bash janus.sh build' gebaut und im Directory /opt/janus installiert werden. 'bash janus.sh start' startet den gebauten Janus-Server in /opt/janus. Das Janus-Video-Room-Plugin stellt bereits eine WebSocket-API bereit, die es ermöglicht, die WebRTC-Verbindungen jeweils einem Raum zuzuteilen. Zuerst muss ein WebSocket eine Session mit dem Server erzeugen und eine Session-ID erhalten:

```
{
  "janus": "create",
  "transaction": [REDACTED]
"socketService_establishSession"
}
```

Jetzt kann sich der WebSocket-Client mit dem Video-Room-Plugin verbinden und eine PluginHandle-ID erhalten:

```
{
  "janus": "attach",
  "session_id": wss.janus.sessionId,
  "plugin": "janus.plugin.videoroom",
  "transaction": [REDACTED]
"socketService_connectSessionToVideoPlugin"
}
```

Besteht eine Verbindung zum Video-Room-Plugin, kann die Video-Room-Plugin-API verwendet werden. Einen neuen Raum zu erstellen sieht dann so aus:

```
{
  "janus": "message",
  "session_id": wss.janus.sessionId,
  "handle_id": wss.janus.handleId,
```

```

        "transaction": `socketService_createRoom::${
{uuid}`,
        "body": {
            "admin_key":
process.env.JANUS_ADMIN_KEY,
            "request": "create",
            "permanent": false,
            "description": `Room for UUID::${{uuid}}`,
            "pin": uuid,
            "is_private": true
        }
    }
}

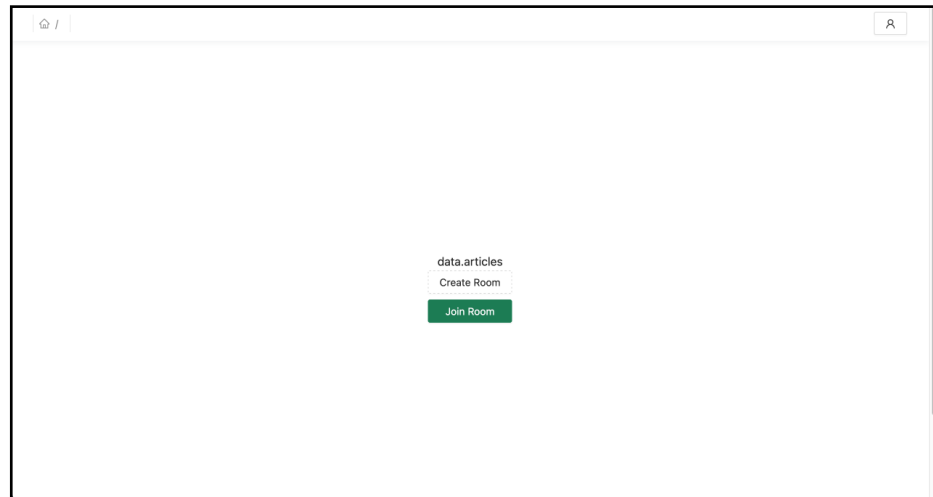
```

Dies erstellt den Raum und übermittelt eine Raum-ID zurück an den Socket-Service. Mittels dieser Raum-ID können sich Clients per WebRTC mit Janus verbinden und die Raum-Selective-Forwarding-Unit benutzen.

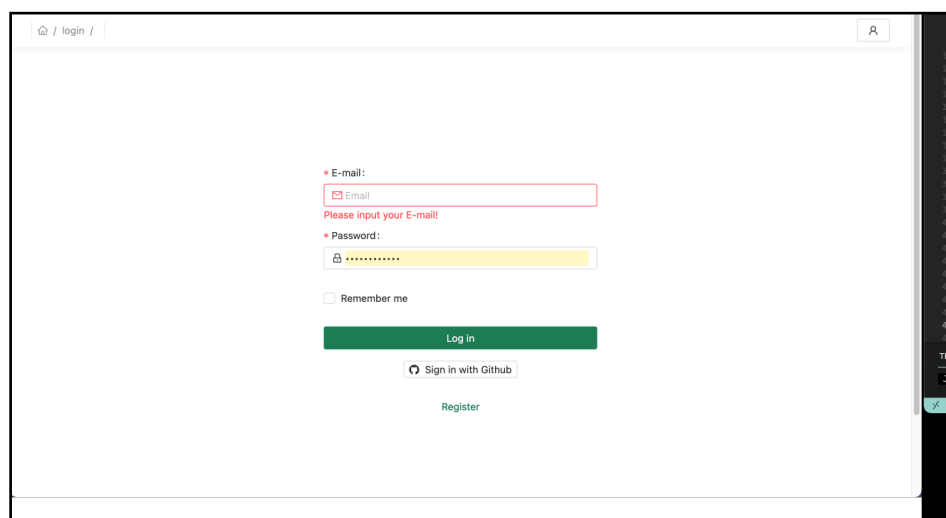
## 6.6 - Frontend-Service

Der Frontend-Service implementiert eine Next.js-Anwendung in einem Docker-Container, welcher auf Port 3000 startet und präsentiert die Seiten für Benutzer\*innen der Webapp. Das Frontend stellt die gepflegten Daten des Content-Management-Systems auf der Startseite mittels GraphQL-Queries dar. Es benutzt das Jason-Web-Token des Auth-Services für User-Accounts und es implementiert die Kommunikation eines Diskussionsraums mit dem Socket-Service per WebSockets und dem WebRTCService über WebRTC.

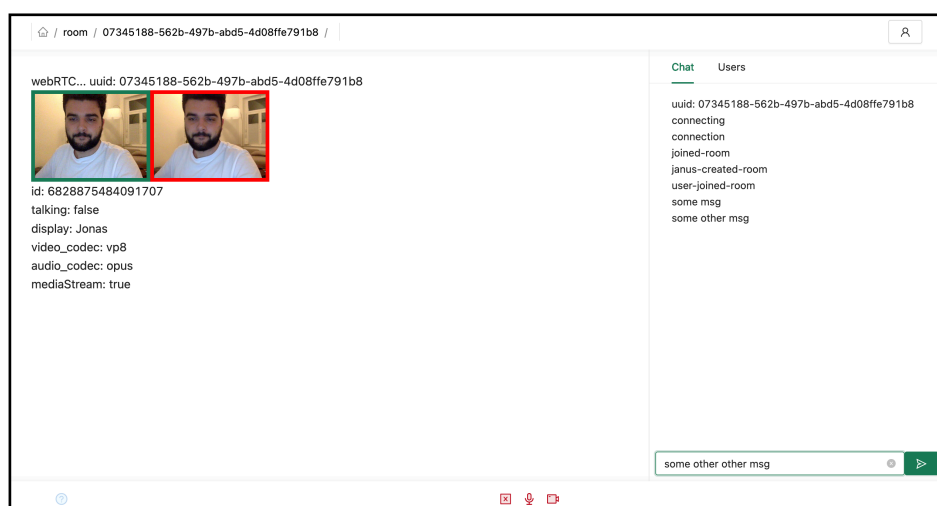




(Abbildung 1: Prototyp - Startseite des Frontends)



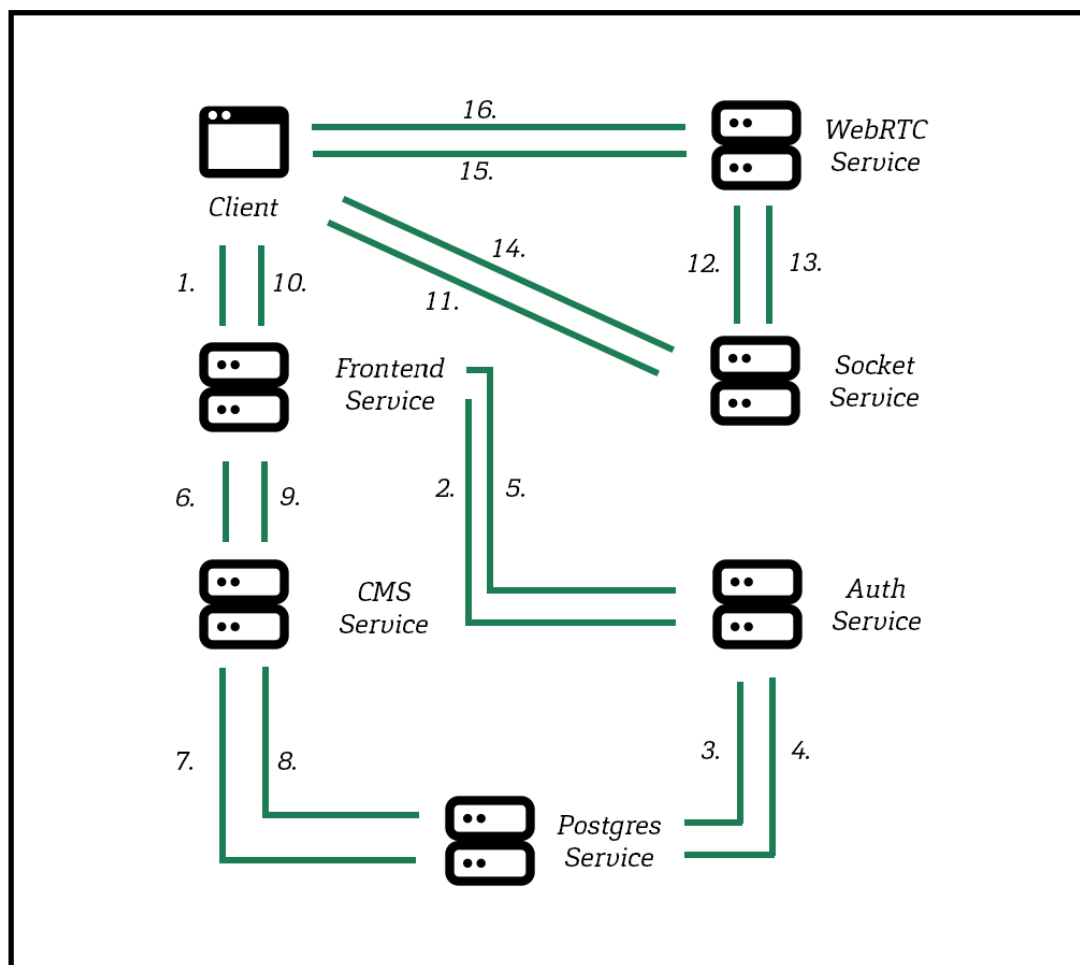
(Abbildung 2: Prototyp - Login-Screen)



(Abbildung 3: Prototyp - Ein Live-Diskussionsraum mit zwei Teilnehmer\*innen.)

## 7 - Systembeschreibung

Um einen Überblick über die Systembestandteile und deren Implementierung als und in Microservices zu schaffen, erläutere ich das gesamte System der Plattform in einem Schaubild ab dem Zeitpunkt, an dem ein exemplarischer Benutzer einem Diskussionsraum beiträgt, um einen besseren Überblick über das Zusammenspiel der Services zu geben.



(Abbildung 1: Ablauf und Zusammenspiel der Microservices bei Aufruf eines Diskussionsraums)

Der beispielhafte Benutzer geht auf die Webseite der Applikation. Ein Dns-Server resolved die Text-Url zu einer IP-Adresse und fragt diesen auf Port 80 an. Ein Nginx-Load-Balancer verteilt die Anfragen auf die

bestehenden Docker Services. In diesem Fall einen Frontend-Container in dem die Next.js-Applikation läuft (1). Next.js ermittelt aus dem Request, welche Seite aufgerufen werden soll und ob diese Seite Authentifizierung benötigt. Sofern der Benutzer einen Jason-Web-Token-Cookie in der Anfrage mitsendet, prüft das Frontend die Validität des Cookies durch die / token Route des Socket-Services (2). Der Socket-Service überprüft die Validität des Cookies und ob der Benutzer in der Postgres-Datenbank enthalten ist (3. und 4.) und sendet diese zurück an das Frontend (5.).

Sofern es sich bei der initialen Anfrage nicht um eine statische Seite handelt, fragt Next.js das StrapiCMS mittels GraphQL-Query nach den eingepflegten Inhalten der jeweiligen Seite (6.).

Strapi verbindet sich dafür mit dem Postgres-Service, einem Docker-Container, der eine Postgresql-Datenbank mit gepflegten Daten beinhaltet (7.). Postgres sendet die Daten an Strapi zurück (8.) und Strapi leitet diese Daten an die Next-Anwendung weiter (9.). Der Frontend-Service Next.js erhält die Inhalte und rendert diese mittels React als reines HTML und schickt diese mit Hinweisen zur React-Hydration zurück zum Browser (10.). Der Browser zeigt die HTML-Seite an und initialisiert die React-App mittels Hydration. Der Benutzer ist einer Diskussions-Seite beigetreten und die React-App baut eine WebSocket-Verbindung zum Socket-Service auf (11.). Nachdem der Connection-Handshake erfolgreich erfolgt ist, besteht eine bilaterale Verbindung zwischen dem Socket-Service und dem Browser. Jetzt sendet der Browser die Botschaft, dass er einem Diskussionsraum beitreten möchte, dazu sendet er 'join-room' und die Raum-ID an den Service. Der Socket-Service ordnet nun alle Benutzer\*innen mit der selben Raum-ID einander zu und schickt allen eine Benachrichtigung, dass ein/e Benutzer/in dem Raum beigetreten ist. Jetzt können Benutzer\*innen mittels WebSockets über Socket-Service-Broadcasting miteinander chatten. Während der Browser sich mit dem Socket-Service verbindet, eröffnet dieser außerdem einen Janus-Raum

mittels Video-Room-Plugin-API (12.). Dieses schickt die Janus Raum-ID zurück an den Socket-Service (13.). Die ID wird anschließend an den Browser in der "join-room" Anfrage weiterleitet (14.).

Dieser kann sich nun im Browser mit dem Video-Room-Plugin der Janus-Instanz verbinden und eine WebRTC-Verbindung aufbauen (15.).

Die bestehende WebRTC-Verbindung sendet ihre Audio- und Video-Streams an alle Teilnehmer\*innen des Raumes und das Frontend rendert die bestehenden Mediastreams in einem Videoelement (16.).

## **8 - Das Projekt starten, bauen und deployen**

Um das Projekt zu bauen und zu installieren, werden folgende Abhängigkeiten benötigt.

Der Package-Manager Yarn v1.22.5, Node version v14, Docker version 20.10.7.

Mittels 'yarn up' bauen alle Docker-Container und mittels 'yarn start' starten alle Docker-Container. Jeder Service besitzt zusätzlich eine Readme mit detaillierten Installationssanweisungen und Hinweisen zur Weiterentwicklung der Applikation.

Für ein Deployment auf einen Live-Server müssen unbedingt die Env Dateien der einzelnen Services angepasst werden. Mittels 'yarn env:show' werden alle Env-Dateien und -Werte angezeigt, die angepasst werden müssen. Dabei ist vor allem wichtig, dass alle Zugangsdaten für die Datenbank und das JWT\_SECRET keine öffentlich zugänglichen Werte benutzen, da sonst die Sicherheit der Applikation nicht gewährleistet werden kann.

Um den webRTCSERVICE zu starten, muss Janus auf einem Server installiert werden. Dazu muss 'sudo bash janus.sh setup && sudo bash janus.sh build && sudo bash janus.sh start' im webRTCSERVICE-

Verzeichnis ausgeführt werden. Außerdem muss der Server, auf dem die App deployed ist, Verbindungen über folgende Ports zulassen:

Port 3000, 1337, 5432, 15432, 4001, sowie UDP Verbindungen über Port 10000-60000.

## **9 - Fazit**

In dieser Arbeit habe ich die verschiedenen Bestandteile und Herangehensweisen für die Entwicklung einer Webapplikation zum Thema Literatur mit dem Fokus auf Voice- und Chatfunktionalität erläutert. Dabei bin ich auf die verschiedenen Technologien eingegangen, die in der Webapplikation und deren Entwicklung eine Rolle spielen und inwiefern diese die auftretenden Probleme lösen. Besonders die Entwicklung der komplexen Zusammenhänge der Microservices und die Implementierung einer Selective-Forwarding-Unit waren sehr anspruchsvoll und zeitintensiv. Auch wenn mein Prototyp noch lange keine fertige Plattform darstellt, die sich auf dem Markt durchsetzen würde, bin ich der Meinung, dass er thematisch eine Nische abdeckt und spannende Komponenten miteinander kombiniert. Wie uns allen durch die Auswirkungen der Corona-Pandemie bewusst geworden ist, ist es wichtig, funktionale und interaktive digitale Strukturen zu schaffen, durch die menschlicher Kontakt möglichst natürlich stattfinden kann. Neben Homeoffice oder Homeschooling sollten auch Freizeitveranstaltungen vermehrt online stattfinden können und für jeden leicht zugänglich sein.

## 10 - Quellenverzeichnis

Advanced Features: Automatic Static Optimization | Next.js. (o. D.). Vercel. <https://nextjs.org/docs/advanced-features/automatic-static-optimization> (abgerufen am 06.August 2021)

Allen, C. (2021, 20. Januar). 3 Key Approaches for Scaling WebRTC: SFU, MCU, and XDN. Red5 Pro. <https://www.red5pro.com/blog/3-key-approaches-for-scaling-webrtc-sfu-mcu-and-xdn/> (abgerufen am 06.August 2021)

Anil, N., Parente, J. & Wenzel, M. (2018, 20. September). Microservices architecture. Microsoft. <https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/microservices-architecture> (abgerufen am 06.August 2021)

Arora, S. (2019, 29. Oktober). Docker vs. Virtual Machines: Differences You Should. Cloud Academy. <https://cloudacademy.com/blog/docker-vs-virtual-machines-differences-you-should-know/> (abgerufen am 06.August 2021)

Barrierefreie Website. (o. D.). Aktion Mensch. <https://www.aktion-mensch.de/inklusion/barrierefreiheit/barrierefreie-website> (abgerufen am 06.August 2021)

Basic Features: Pages | Next.js. (o. D.). Next.Js. <https://nextjs.org/docs/basic-features/pages#pre-rendering> (abgerufen am 06.August 2021)

Cloud Education (2021, 6. April). REST APIs. IBM. <https://www.ibm.com/cloud/learn/rest-apis> (abgerufen am 06.August 2021)

Cloudflare. (o. D.). What Is My IP Address. <https://whatismyipaddress.com/nat> (abgerufen am 06.August 2021)

CommCon. (2018, 27. Juni). Scaling server-side WebRTC applications: the Janus challenge by Lorenzo Miniero [Video]. YouTube. <https://www.youtube.com/watch?v=zxRwELmyWU0> (abgerufen am 06.August 2021)

Community Guidelines. (2021, 5. April). Clubhouse. <https://www.notion.so/joinclubhouse/Community-Guidelines-461a6860abda41649e17c34dc1dd4b5f> (abgerufen am 06.August 2021)

Customize Theme - Ant Design. (o. D.). Ant Design. <https://ant.design/docs/react/customize-theme> (abgerufen am 06.August 2021)

DSL-Statistik: Die schnellsten Provider, Bundesländer und Orte. (o. D.). Netzwelt. <https://www.netzwelt.de/dsl-speedtest/statistik.html> (abgerufen am 06.August 2021)

Express/Node introduction - Learn web development | MDN. (2021, 27. April). MDN Web Docs. [https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express\\_Nodejs/Introduction](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction) (abgerufen am 06.August 2021)

FOSDEM. (2018, 8. März). Introducing mediasoup A WebRTC SFU for Node.js [Video]. YouTube. <https://www.youtube.com/watch?v=GhdFOZTWTw> (abgerufen am 06.August 2021)

Getting started with peer connections |. (2020, 31. Juli). Web RTC. <https://webrtc.org/getting-started/peer-connections> (abgerufen am 06.August 2021)

Hate Speech. (2020, März). Landeszentrale für politische Bildung Baden-Württemberg. <https://www.lpb-bw.de/hatespeech> (zuletzt aufgerufen am 06.08.2021)

I.C.Education (2019, 6. August). Relational Databases. IBM. <https://www.ibm.com/cloud/learn/relational-databases> (abgerufen am 06.August 2021)

JWT.IO. (o. D.). JSON Web Tokens - Jwt.Io. <https://jwt.io/> (abgerufen am 06.August 2021)

JWT.IO - JSON Web Tokens Introduction. (o. D.). JSON Web Tokens - Jwt.Io. <https://jwt.io/introduction> (abgerufen am 06.August 2021)

LeMay, R. (2021, 30. April). What is Software Scalability? Horizontal and Vertical Scaling - Cloud Let's GO. Cloud Let's GO. <https://cloudletsgo.com/2021/04/30/what-is-software-scalability-horizontal-and-vertical-scaling/> (abgerufen am 06.August 2021)

Levent-Levi, T. (2019, 15. April). WebRTC Multiparty Architectures. BlogGeek.Me. <https://bloggeek.me/webrtc-multiparty-architectures/> (abgerufen am 06.August 2021)

Levent-Levi, T. (2019, Mai 28). WebRTC vs WebSockets. BlogGeek.Me. <https://bloggeek.me/webrtc-vs-websockets/> (abgerufen am 06.August 2021)



Levent-Levi, T. (2021, 1. August). What is WebRTC and What is it Good For? BlogGeek.Me. <https://bloggeek.me/what-is-webrtc/> (abgerufen am 06.August 2021)

Maldonado, L. (2020, 7. August). How Next.js can help improve SEO. LogRocket Blog. <https://blog.logrocket.com/how-next-js-can-help-improve-seo/> (abgerufen am 06.August 2021)

Merker, H. (2021, 19. Februar). Wir Gehörlosen nehmen den Hörenden doch nichts weg. Zeit Online. <https://www.zeit.de/zustimmung?url=https%3A%2F%2Fwww.zeit.de%2Fdigital%2F2021-02%2Fclubhouse-julia-probst-gehoerlos-schwerhoerig-bloggerin> (abgerufen am 06.August 2021)

Microservices architecture style - Azure Application Architecture Guide. (2019, 30. Oktober). Microsoft. <https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices> (abgerufen am 06.August 2021)

Mukit, A. (2018, 12. April). WebRTC SDP (Session Description Protocol) Details. DEV Community. <https://dev.to/lucpattyn/webrtc-sdp-session-description-protocol-details--5593> (abgerufen am 06.August 2021)

PostgreSQL: About. (o.D.). The PostgreSQL Global Development Group. <https://www.postgresql.org/about/> (abgerufen am 06.August 2021)

Rehkopf, M. (o. D.). User Stories | Examples and Template. Atlassian Agile Coach. <https://www.atlassian.com/agile/project-management/user-stories> (abgerufen am 06.August 2021)

Rungta, K. (o. D.). Microservices Tutorial: What is, Architecture and Example. Guru 99. [https://www.guru99.com/microservices-tutorial.html#3+\(Microservices+vs.+Monolithic+Architecture\)](https://www.guru99.com/microservices-tutorial.html#3+(Microservices+vs.+Monolithic+Architecture)) (abgerufen am 06.August 2021)

SFHTML5. (2017, 28. Oktober). Scaling WebRTC Video Infrastructure [Video]. YouTube. <https://www.youtube.com/watch?v=o2SVegcONNk> (abgerufen am 06.August 2021)

SFU (Selective Forwarding Unit). (2021, 11. Juli). WebRTC Glossary. <https://webrtcglossary.com/sfu/> (abgerufen am 06.August 2021)

Sime, A. (2020, Dezember). WebRTC Media Servers – SFUs vs MCUs. WebRTC.Ventures. <https://webrtc.ventures/2020/12/webrtc-media-servers-sfus-vs-mcus/> (abgerufen am 06.August 2021)

TURN server |. (2019, 28. Mai). Web RTC. <https://webrtc.org/getting-started/turn-server> (abgerufen am 06.August 2021)

Understanding React Hydration. (o. D.). Gatsby. <https://www.gatsbyjs.com/docs/conceptual/react-hydration/> (abgerufen am 06.August 2021)

Venema, A. (2021, 20. Juli). How to Successfully Scale Your WebRTC Application in 2021. OTTVerse. <https://ottverse.com/how-to-scale-webrtc-application-in-2021/> (abgerufen am 06.August 2021)

„webrtc“ | Can I use. . . Support tables for HTML5, CSS3, etc. (2021, 6. Juli). Can I Use. <https://caniuse.com/?search=webrtc> (abgerufen am 06.August 2021)

WebRTC connectivity - Web APIs | MDN. (2021, 7. Juni). MDN Web Docs. [https://developer.mozilla.org/en-US/docs/Web/API/WebRTC\\_API/Connectivity](https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Connectivity) (abgerufen am 06.August 2021)

WebRTC.ventures. (2019, 6. Dezember). 1 4 ICE STUN and TURN [Video]. YouTube. <https://www.youtube.com/watch?v=4FkRf9utSc> (abgerufen am 06.August 2021)

Wessling, R. (o. D.). Headless CMS explained in 1 minute. Contentful. <https://www.contentful.com/r/knowledgebase/what-is-headless-cms/> (abgerufen am 06.August 2021)

What is Docker? (2018, 31. August). Microsoft. <https://docs.microsoft.com/en-us/dotnet/architecture/microservices/container-docker-introduction/docker-defined> (abgerufen am 06.August 2021)

What is GraphQL? | GraphQL Tutorial. (o. D.). Hasura. <https://hasura.io/learn/graphql/intro-graphql/what-is-graphql/> (abgerufen am 06.August 2021)

What is REST. (o. D.). REST API Tutorial. <https://restfulapi.net/> (abgerufen am 06.August 2021)

What is WebSocket? Explanation and examples. (2020, 7. August). IONOS Digitalguide. <https://www.ionos.com/digitalguide/websites/web-development/what-is-websocket/> (abgerufen am 06.August 2021)

# 11 - Liste der verwendeten Abkürzungen

***Antd***: Ant Design

***API***: Application Programming Interface

***CMS***: Content Management System

***CSR***: Client Side Rendering

***CSS***: Cascading Style Sheets

***CPU***: Central Processing Unit

***HTML***: Hypertext Markup Language

***HTTP***: Hypertext Transfer Protocol

***ICE***: Interactive Connectivity Establishment

***JWT***: Jason Web Token

***MVP***: Minimum Viable Product

***MCU***: Multipoint Conferencing Unit

***NAT***: Network Address Translation

***REST***: Representational State Transfer

***RAM***: Random Access Memory

***SSR***: Server Side Rendered

***SSG***: Server Statically Generated

***SEO***: Search Engine Optimisation

***STUN***: Session Traversal Utilities for NAT

***SQL***: Structured Query Language

***SDP***: Session Description Protocol

***SVG***: Scalable Video Coding

***SFU***: Selective Forwarding Unit

***TCP***: Transmission Control Protocol

***TURN***: Traversal Using Relays around NAT

***UDP***: User Datagram Protocol

***WebRTC***: Web Realtime Communication Protocol

## Eidesstattliche Erklärung

Hiermit gebe ich eine eidesstattliche Erklärung ab, dass ich die vorliegende Arbeit selbstständig verfasst und dabei keine anderen als die angegebenen Hilfsmittel benutzt habe. Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach Publikationen oder Vorträgen anderer Autoren entnommen sind, habe ich als solche kenntlich gemacht.

Die Arbeit wurde bisher weder gesamt noch in Teilen einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Hamburg, 07.08.2021



---

gez. Jonas Leonhard, 611179