THM | CAMPUS GIESSEN | MNI
TECHNISCHE HOCHSCHULE MITTELHESSEN
Mathematik, Naturwissenschaften und Informatik

Bachelorthesis

Auswahl und Integration eines Chatsystems für digitale Lehre

zur Erlangung des akademischen Grades

Bachelor of Science

eingereicht im Fachbereich Mathematik, Naturwissenschaften und Informatik an der Technischen Hochschule Mittelhessen

von

Max Stephan

13. Januar 2022

Referent: Prof. Dr. Frank Kammer

Korreferent: Prof. Thomas Friedl

Erklärung der Selbstständigkeit

Hiermit versichere ich, die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die Zitate deutlich kenntlich gemacht zu haben.

Gießen, den 13. Januar 2022

Zusammenfassung

Messenger sind mittlerweile im alltäglichen Bereich eine der Hauptkommunikationsmittel geworden. Durch die voranschreitende Digitalisierung werden auch im Bereich der Lehre immer mehr Veranstaltungen online durchgeführt oder durch digitale Systeme begleitet. Dabei spielt auch die Kommunikation der verschiedenen Teilnehmer eine große Rolle, welche ebenfalls mithilfe von Messenger durchgeführt wird.
Es ist zwar möglich, auf externe Chatsysteme zurückzugreifen, diese haben allerdings teilweise Datenschutzprobleme und bieten keine gute Integration mit den vorhandenen Lehrsystemen. Demnach stellen sich die Fragen, welches der verschiedenen Chatsysteme eignet sich für die Integration in ein System für die digitale Lehre, welche Funktionen werden benötigt und wie ist eine Integration möglich.

Inhaltsverzeichnis

1 Einführung

1.1 Motivation

Da die Digitalisierung in immer mehr Bereichen des Lebens voranschreitet, gerade auch begleitet durch die Covid-19-Pandemie, werden besonders im Bereich der Lehre immer häufiger Softwarelösungen für die Unterstützung der digitalen Lehre gesucht. Hierbei spielt auch die Kommunikation eine wichtige Rolle. Im alltäglichen Bereich sind Messenger (Chatsysteme) durch die starke Verbreitung von Smartphones eines der Hauptkommunikationsmittel geworden und haben die *SMS* großteils verdrängt [vgl. Bun21, S.4]. In der digitalen Lehre sieht dies ähnlich aus. So werden zum Beispiel im Rahmen von Lehrveranstaltungen Chatsysteme eingesetzt, um Fragen von Studierenden außerhalb der Lehrveranstaltungen zu beantworten. Dabei werden häufig Messenger wie zum Beispiel *WhatsApp*, *Discord* oder *Signal* verwendet, da deren Funktionsumfang im Gegensatz zu den in Lernplattformen integrierten Messenger (z.B. *Ilias* [ILI22] oder *Moodle* [Moo22a]) umfangreicher ist. Die Verwendung der genannten Messenger bringt jedoch einige Probleme mit sich. So sind die meisten dieser genutzten Messenger nicht datenschutzkonform. In einer Handreichung für Lehrkräfte, welche vom hessischen Beauftragten für Datenschutz und Informationsfreiheit 2015 herausgegeben wurde, wird die Nutzung von Messengern wie z.B. WhatsApp für die schulische Kommunikation nicht empfohlen [vgl. Hes15]. Das Kultusministerium von Baden-Württemberg verbietet sogar die Verwendung von sozialen Netzwerken für die Kommunikation zwischen Schülern und Lehrkräften sowie zwischen Lehrkräften untereinander[vgl. Min]. Zusätzlich zu den Bedenken bezüglich des Datenschutzes müssen durch die Nutzung der genannten Messenger mehrere Plattformen genutzt werden, um den gesamten gewünschten Funktionsumfang für die digitale Lehre zu bieten.

Bei der Entwicklung der Lernplattform *Audimax* (Abschnitt 1.2) stand das Vereinen aller Funktionen in einer Anwendung im Vordergrund. Dabei ist das Ziel dieser Lernplattform, mehrere Dienste in einem System zu vereinen und dem Nutzer das Gefühl eines einzigen geschlossenen Systems zu bieten.

1.2 Audimax

Durch die Covid-19-Pandemie wurde es notwendig, Lehrveranstaltungen online abzuhalten. Hierbei gab es allerdings einige Probleme: Einerseits wurden Programme verwendet, welche sich nach kurzer Zeit als nicht *datenschutzkonform* herausstellen, wofür dann Alternativen gesucht werden mussten. Andererseits wurden für unterschiedliche Aufgaben viele verschiedene Systeme verwendet, was die Verwaltung der Plattformen und Nutzer erschwert. Des Weiteren ist dies auch für die Studenten/Schüler schwieriger, da diese sich an mehrere Oberflächen gewöhnen müssen.

Die zugrundeliegende Idee von Audimax ist es, verschiedenste Dienste in einem einzigen Programm zu vereinen und somit dem Benutzer alle benötigten Funktionen bezüglich der Online-Lehre auf einer Plattform zur Verfügung zu stellen. Dies bietet den Vorteil, dass der Benutzer sich lediglich innerhalb einer Software aufhalten muss und nicht für das Nutzen weiterer Funktionen die Plattform wechseln muss. Bei der Entwicklung wurde zunächst der Fokus darauf gelegt, dass es möglich ist, Videokonferenzen abzuhalten, Dateien zu teilen und Termine zu verwalten. Später wurde, um eine nicht datenschutzkonforme Plattform zu ersetzen, die Möglichkeit, Pinboards zu erstellen, hinzugefügt. Dabei wurde das Pinboard nicht nur neben den anderen Funktionen zu Audimax hinzugefügt, sondern mit diesen verknüpft. Somit ist es zum Beispiel möglich, Dateien auf dem Pinboard zu teilen oder Termine anzuzeigen.

Für die grundlegende Organisation von Veranstaltungen/Klassen gibt es sogenannte *Kurse*. Zu diesen kann man beliebig viele Benutzer mit verschiedenen Berechtigungen hinzufügen. Die *Kurse* enthalten wiederum sogenannte *Kanäle*. Diese Kanäle repräsentieren hierbei die verschiedenen Audimax-Funktionen.

Zu Beginn der Arbeit (Oktober 2021) gab es in Audimax Kanäle für die *Dateiverwaltung*, *Konferenzen (BigBlueButton)*, *Kalender* und *Pinboards*. Innerhalb eines Kurses können beliebig viele Kanäle mit den genannten Funktionen erstellt werden. Dies ermöglicht es den Nutzern zum Beispiel, verschiedene Konferenzen, Pinboards und Chats für unterschiedliche Anwendungsfälle zu erstellen oder auch Konferenzen parallel abzuhalten. Zusätzlich können in einem Kurs auch Gruppen erstellt werden, welche wiederum beliebig viele Kanäle beinhalten können. Da innerhalb der Gruppen die Nutzer der Kurse unabhängig verwaltet werden können, bietet diese Struktur Lehrkräften und Lernenden die Möglichkeit, gruppenspezifische Arbeiten durchzuführen.

1.3 Ziele dieser Arbeit

Ziel dieser Arbeit ist es, herauszufinden, welches Chatsystem am besten für die Integration in ein Programm für die digitale Lehre verwendet werden kann. Zusätzlich soll für das gefundene System auch ein Konzept zur Integration erstellt werden und ein Teil der Umsetzung davon aufgezeigt werden. Um herauszufinden, welche Funktionen dieses Chatsystem benötigt, soll in vorhandenen Chatsystemen geschaut werden, welche Funktionen diese bieten. Auch soll die Frage, warum ein Programm für die digitale Lehre überhaupt ein Chatsystems benötigt, beantwortet werden. Bei der Integration soll des Weiteren darauf geachtet werden, dass dies so einfach wie möglich für den Nutzer geschieht, damit dieser das Gefühl eines geschlossenen Systems nicht verliert.

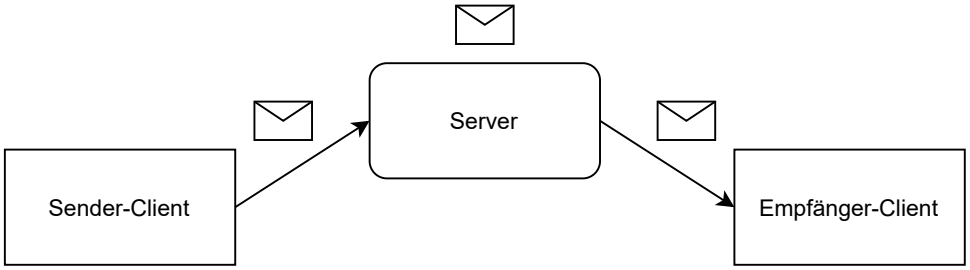
1.4 Vorgehensweise

Zuerst folgt eine grundlegende Erklärung, was ein Messenger (Chatsystem) ist, und eine Vorstellung aller in dieser Arbeit erwähnten Messenger (Kapitel 2). Hiernach folgt eine grundlegende Erklärung aller für die Arbeit benötigten Technologien (Kapitel 3). Daraufhin folgt eine Bestandsaufnahme, warum ein Chatsystem überhaupt in einer Lernplattform benötigt wird (Kapitel 4). Anschließend folgt eine Zusammenstellung, welche Features ein Chatsystem im Kontext der digitale Lehre benötigt. Dies geschieht, indem sich Funktionen von bekannten Messengern angeschaut werden und geschaut wird, wie nützlich diese für die digitale Lehre sind. Basierend auf dem vorherigen Kapitel folgt eine Konzeptionierung für das Chatsystem (Kapitel 5). Diese startet mit dem Vergleich und der Auswahl des Chatsystems und resultiert in einem Konzept zur Integration des ausgewählten Chatsystems. Im 6. Kapitel geht es dann um die konkrete Umsetzung des im vorherigen Kapitel beschriebenen Konzepts. Dieses enthält einige Beispiele sowie die aufgetretenen Probleme und deren Lösung.

2 Chatsysteme

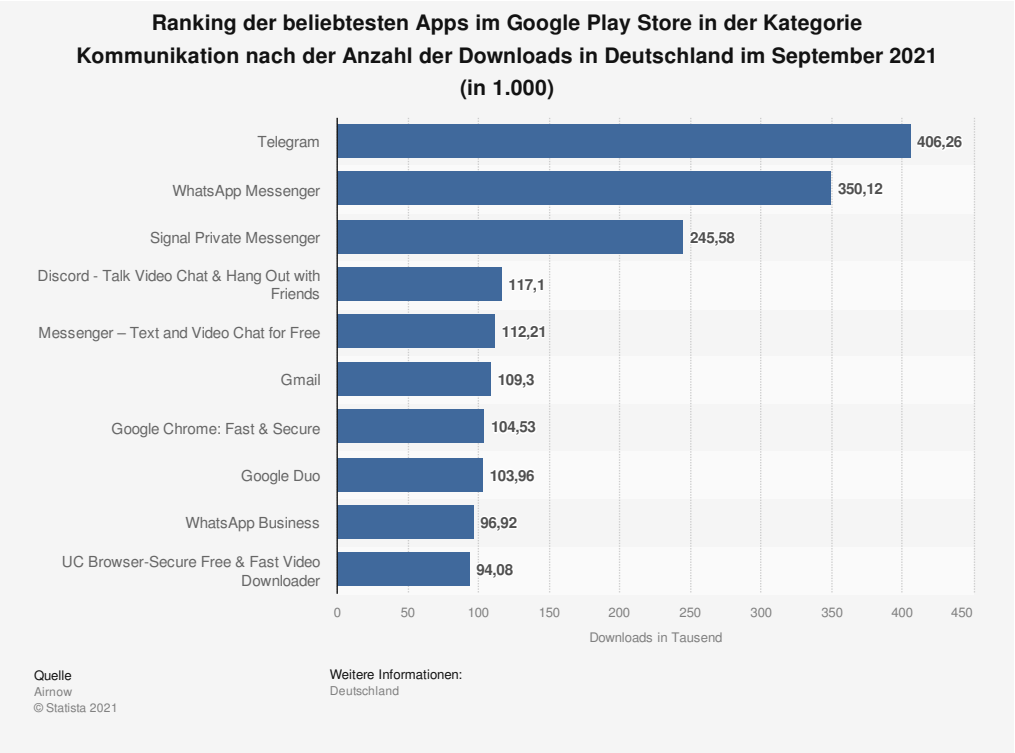
Um Messenger (Chatsysteme) besser zu verstehen, folgt in diesem Kapitel zuerst eine Erklärung, wie diese funktionieren. Danach folgt eine Vorstellung einiger Messenger, welche im Laufe dieser Arbeit erwähnt werden.

Grundlegend findet in einem Messenger sämtliche Kommunikation mithilfe eines Clients statt. Dieser Client sendet Nachrichten an den Server des Betreibers, welcher diese sobald möglich an den Client des gewünschten Empfängers weiterleitet. Hierbei ist der Nutzer meist nicht auf einen Client beschränkt, sondern kann die Nachrichten an mehreren Clients (wie zum Beispiel über eine Smartphone-App und einen Internetbrowser) abrufen. Des Weiteren ist es auch möglich, Nachrichten an mehrere Nutzer zu versenden. Wenn der Server die Nachricht so lange aufbewahrt bis diese zugestellt werden kann und somit die verschiedenen Nutzer nicht gleichzeitig online sein müssen, handelt es sich hierbei um eine *asynchrone* Kommunikation. [vgl. Bun21, S.6]



Listing 1: Grundlegende Funktionsweise eines Messengers [vgl. Bun21, S.6]

Doch welche Messenger sind relevant und können als Referenz oder auch zur Integration verwendet werden?
Es werden die Messenger betrachtet, welche die größte Relevanz in Deutschland besitzen. Laut *Statista* wurden im September 2021 die Messenger *Telegram* [Tel22], *WhatsApp* [Wha22], *Signal* [Sig22] und *Discord* [Dis22] am meisten im *Google Play Store* heruntergeladenen [vgl. Air21].



Listing 2: Meist heruntergeladene Messenger [Air21]

Der Messenger *Threema* [Thr22] gilt als sicherere Alternative zu den meist heruntergeladenen Messengern, weshalb dieser auch mit in die Liste aufgenommen wird. Um auch Chatsysteme aus dem Arbeits- und Lehrbereichen in die Betrachtung einzubeziehen, werden die beiden Chatsysteme *Slack* [Sla22] und *Microsoft Teams* [Mic22] betrachtet, da diese dort weitverbreitet sind. Damit neben *Signal* noch weitere *Open-Source* (Abschnitt 3.5) Programme vertreten sind, werden auch noch *Nextcloud Talk* [Nex22b] und *Matrix* [Mat22] betrachtet.

2.1 Telegram

Telegram ist ein kostenloser Messenger, der es ermöglicht, Textnachrichten, Bilder, Videos und Dateien zu versenden. Dies ist entweder zwischen zwei Personen oder Gruppen von bis zu 200.000 Personen möglich. Hierbei legt Telegram laut eigener Angabe den Schwerpunkt auf Geschwindigkeit und Sicherheit. Die versendeten Nachrichten können auf allen Geräten (Handy, Tablets oder Computer) gleichzeitig abgerufen und versendet werden. Telegram wurde 2013 von Pavel Durov und Nikolai Durov in St. Petersburg gegründet. Aufgrund von lokalen IT-Vorschriften, die Telegram nicht erfüllen wollte, hat Telegram den Sitz aus St. Petersburg nach Dubai verlegt. [vgl. Tel22]

2.2 WhatsApp

WhatsApp ist ein kostenloser Messenger, welcher 2009 von Jan Koum und Brian Acton gegründet wurde und seit 2014 zu Facebook (Meta) gehört. Er ermöglicht es, Textnachrichten, Dateien, Bilder uvm. zwischen zwei Personen oder Gruppen zu versenden. Für die Nutzung von WhatsApp ist zwingend ein Smartphone notwendig, über das Nachrichten empfangen und versendet werden. Ein Abrufen/Versenden der Nachrichten ist auch über einen Computer möglich, hierbei werden diese allerdings nur zwischen dem Computer und dem Smartphone synchronisiert. [vgl. Wha22]

2.3 Signal

Signal wird von der gemeinnützigen Signal Foundation entwickelt und ist ein kostenloser Messenger, welcher durch Spenden finanziert wird. Das Ziel des Messengers ist es, eine datenschutzkonforme und sichere Kommunikation zu ermöglichen. Signal kann auf *Android*, *iOS* und dem Desktop verwendet werden, benötigt für die Registrierung allerdings eine Telefonnummer. Wie bei vielen anderen Messengern ist die Kommunikation in Privat- oder Gruppenchats möglich. [vgl. Sig22]

2.4 Discord

Discord wurde im Jahr 2015 von Jason Citron und Stan Vishnevskiy als Plattform zur Kommunikation während des Spielens von Online-Spielen veröffentlicht. Mittlerweile wird Discord allerdings nicht mehr nur für das Online-Spielen, sondern für jegliche Gruppen- und Einzelkommunikation verwendet. Discord funktioniert plattformunabhängig und kann auf allen gängigen Geräten verwendet werden. [vgl. Dis22]

2.5 Threema

Threema ist ein Messenger, der seit 2014 in der Schweiz entwickelt wird. Der Hauptfokus des Messengers ist eine sichere und datenschutzkonforme Kommunikation. Wie bei den meisten anderen Messengern ist die Kommunikation zwischen zwei Nutzern oder in Gruppen möglich. Für die Nutzung von Threema ist ein Smartphone notwendig. Im Gegensatz zu vielen anderen Messengern ist die Angabe einer Telefonnummer allerdings optional. [vgl. Thr22]

2.6 Slack

Slack ist ein Messenger, welcher hauptsächlich für die Businesskommunikation ausgelegt ist. Das Versenden und Abrufen von Nachrichten ist auf allen gängigen Geräten möglich. Die Nutzer werden in sogenannten *Workspaces* verwaltet. Innerhalb dieser *Workspaces* können die Nutzer untereinander oder in sogenannten *Channels* kommunizieren. [vgl. Sla22]

2.7 Microsoft Teams

Das Programm Microsoft Teams wurde im Jahr 2017 von Microsoft als Plattform für Kommunikation und Zusammenarbeit veröffentlicht. Es ist für die Verwendung in Unternehmen, Bildungseinrichtungen oder auch für den Privatgebrauch gedacht und ist Teil von *Microsoft 365 (Office 365)*. Um zum Beispiel Chatnachrichten zu organisieren, werden sogenannte *Channels* erstellt, die das Versenden von Nachrichten an alle Mitglieder dieses *Channels* ermöglichen. Diese *Channels* werden in sogenannten *Teams* verwaltet, welche zum Beispiel eine Abteilung in einer Firma darstellen können. [vgl. Mic22]

2.8 Matrix

„Matrix ist ein offener Standard für interoperable, dezentralisierte Echtzeitkommunikation über IP. Es kann für Instant Messaging, VoIP/WebRTC-Signalisierung, Internet of Things-Kommunikation oder überall dort eingesetzt werden, wo eine Standard-HTTP-API für die Veröffentlichung und das Abonnieren von Daten bei gleichzeitiger Verfolgung des Gesprächsverlaufs benötigt wird.“ [Matc]

Dieser Standard wird von der gemeinnützigen *Matrix.org Foundation* entwickelt und verwaltet. Hierbei definiert die *Matrix.org Foundation* den Standard und stellt Tools (wie z.B. SDKs) zur Verwendung bereit. Zusätzlich entwickelt die *Marix.org Foundation* noch Referenz-Implementierungen, welche die vollständige Verwendung des Matrix-Protokolls ermöglichen. [vgl. Matc].

2.9 Nextcloud Talk

Für das Chatten in der Kollaborations-Plattform *Nextcloud* [Nex22a] gibt es die Erweiterung Nextcloud Talk. *Nextcloud* ist eine Kollaborations-Plattform, die es ermöglicht,

Dateien, Bilder, Kalender uvm. zu teilen, um die Online-Zusammenarbeit zu vereinfachen [vgl. Nex22a]. Nextcloud Talk wird, wie Nextcloud selbst, von der *Nextcloud GmbH* entwickelt. Im Gegensatz zu den anderen Messengern, mit der Ausnahme von Matrix, wird Nextcloud Talk nicht direkt als Service angeboten, sondern ist dafür gedacht, auf eigenen Servern ausgeführt und verwendet zu werden. Nextcloud Talk kann auf allen gängigen Geräten verwendet werden und ermöglicht hierbei Privat- und Gruppenkommunikation. [vgl. Nex22b]

3 Grundlagen

3.1 Ende-zu-Ende-Verschlüsselung (E2E)

Um die von Messenger (Chatsystemen) gesendeten Daten gegen unerwünschtes Mitlesen zu schützen, gibt es zwei Arten der Verschlüsselung. Einerseits eine sogenannte *Transportverschlüsselung*, die die übertragenen Inhalte vor Außenstehenden schützt. Der Serverbetreiber kann hierbei allerdings jegliche Inhalte einsehen. Um diese auch vor dem Serverbetreiber zu schützen, ist eine zusätzliche Verschlüsselung der Inhalte zwischen den beiden Clients notwendig, dies ist mit einer *Ende-zu-Ende-Verschlüsselung* (E2E) möglich. Generell ist eine E2E-Verschlüsselung völlig ausreichend, eine zusätzliche Transportverschlüsselung ist heutzutage jedoch Standard und bringt keinen großen Mehraufwand. [vgl. Bun21, 3.1]

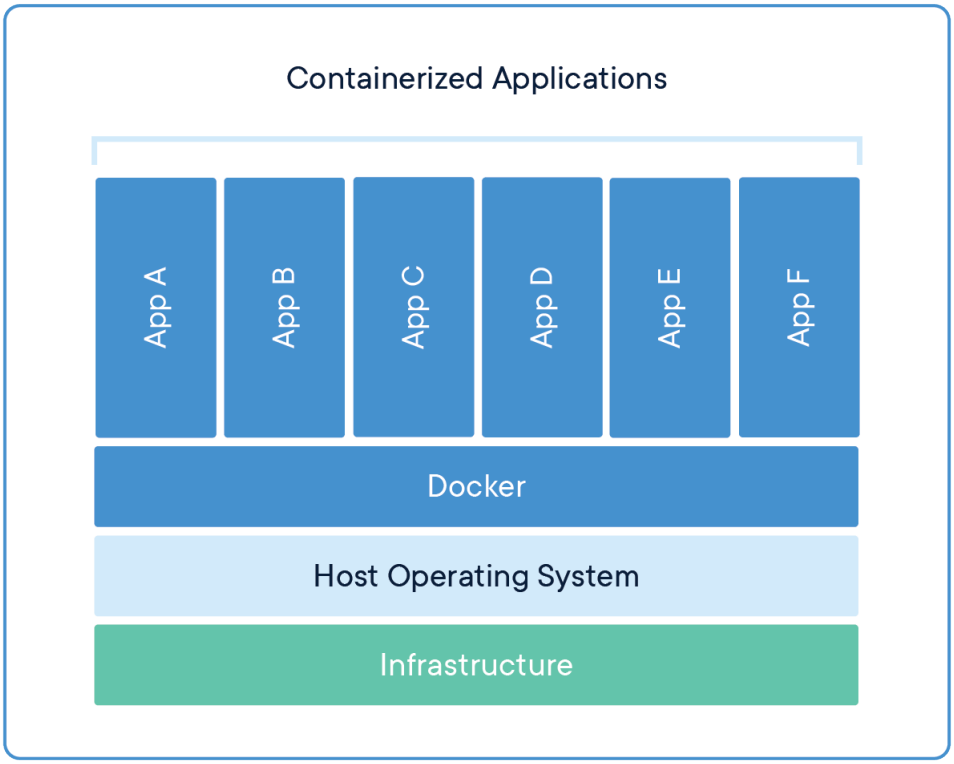
3.2 Software Development Kit (SDK)

Ein Software Development Kit (SDK) ist ein Baukasten für Programmierer, welches für die Applikationsentwicklung Hilfsprogramme und Bibliotheken bereitstellt [vgl. PF10, S.796]. Zum Beispiel stellt eine Website eine SDK bereit, die es ermöglicht, mit deren Schnittstellen zu kommunizieren. Außerdem gibt es einige SDKs für die Entwicklung von plattformspezifischen Programmen wie zum Beispiel Android, IOS oder Windows Apps.

3.3 Docker und Docker Compose

Docker [Doc22] ist eine Plattform für das Entwickeln, Bauen und Ausführen von portablen verteilten Anwendungen. Hierbei ermöglicht es Docker, alle benötigten Abhängigkeiten, um ein Programm auszuführen, in eine einzelne Einheit, ein *Docker image*, zu packen. Dieses *Docker image* enthält dann alle Konfigurationen, *Bibliotheken* und Betriebssystem-Tools, welche das Programm zur Ausführung benötigt. Die Besonderheit an *Docker* ist es, dass die *Docker images* nicht mit einem eigenen virtuellen Betriebs-

system ausgeführt werden, sondern Teile des Betriebssystems verwenden, auf dem sie ausgeführt werden (Listing 3). Hierbei wird das *Docker image* in einer isolierten Umgebung, einem *Docker Container*, ausgeführt, welcher sein eigenes Dateisystem und eigene Umgebungsvariablen besitzt. [vgl. Voh16, S.1]



Listing 3: Anwendungen in Docker Containern [Doc22]

Viele der heutigen Anwendungen bestehen aus mehreren Teilen, die für das Ausführen der Anwendung benötigt werden. Zum Beispiel benötigt eine einfache Website schon einen *Web-Server* und eine Datenbank. Um diese Art von Anwendung einfacher ausführen zu können gibt es die *Docker*-Weiterentwicklung *Docker Compose*. *Docker Compose* ermöglicht es, mehrere *Docker images* mit deren Konfigurationen und Beziehungen untereinander in einer Datei zu verwalten. Diese können dann mithilfe eines *Befehls* gestartet und gestoppt werden. [vgl. Ibr21, S.2].

3.4 Unicode

Grundlegend wird jegliche Information in einem Computer durch eine Zahl repräsentiert. Um nun also Buchstaben und andere Zeichen abzubilden, wird die sogenannte *Zeichenkodierung (character encoding)* verwendet, die jedem Zeichen eine Zahl zuordnet. Zu Beginn gab es viele verschiedene Systeme, welche diese Zuordnungen definierten.

Allerdings gab es kein System, welches alle Sprachen abdecken konnte. Selbst für Englisch gab es kein System, dass alle allgemein verwendeten Buchstaben, Satzzeichen und technischen Symbole abdeckte. Weitere Probleme waren die Unterstützung von piktografischen Sprachen (wie zum Beispiel Japanisch) und die Überschneidungen von Zahlen, wobei verschiedene Systeme die gleiche Zahl für ein anderes Zeichen verwenden. Um diese Probleme zu lösen, wurde der *Unicode-Standard* entwickelt. [vgl. Uni22]

Der *Unicode-Standard* definiert ein einheitliches Verfahren zur Kodierung von geschriebenen Zeichen und Text. Hierfür wird jedem Zeichen eine Zahl *(code point)* und ein Name zugewiesen. Der *Unicode-Standard* wird von allen modernen Betriebssystemen und Programmiersprachen unterstützt und bildet heutzutage die Grundlage für das Internet und globale Software. Die im September 2021 erschienene Version 14 des Standards enthält 144.697 Zeichen. Hierbei sind nicht nur die Zeichen aller modernen Sprachen enthalten, sondern auch einige historische. [vgl. Uni21, S.1,S.2]

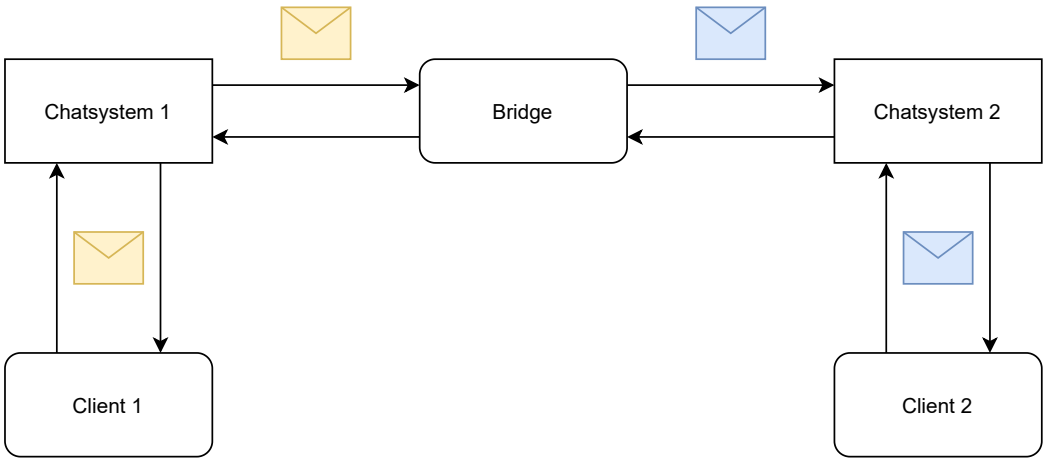
3.5 Open-Source

Der Begriff *Open-Source* beschreibt Software, deren Verwendung kostenlos möglich ist und welche einen frei verfügbaren Source-Code besitzt [vgl. Kee19, S.1]. Um als *Open-Soruce* Software zu gelten reicht es allerdings nicht, den Source-Code einfach zu veröffentlichen, sondern es müssen einige Kriterien der *Open Source Initiative* erfüllt werden [vgl. Ope22]. Dabei geht es größtenteils um die *Lizenz* des veröffentlichen Source-Codes, die zum Beispiel ermöglichen muss, dass die Software als Bestandteil eines Software-Pakets verkauft oder weitergegeben werden darf [vgl. Ope22]. Viele der weitverbreiteten Programme oder Teile dieser sind *Open-Source*, wie zum Beispiel das *Linux-Betriebssystem*, welches im Server, aber auch im Client-Bereich weit verbreitet ist.

3.6 Bridging

Eine Kommunikation zwischen verschiedenen Messengern ist meist nicht einfach möglich, da diese verschiedene Technologien und Spezifikationen verwenden. Um die Kommunikation über Messenger hinweg zu ermöglicht, ist *Interoperabilität* notwendig. Als *Interoperabilität* wird die Zusammenarbeit von zwei oder mehreren Software-Komponenten beschrieben, die trotz Unterschieden in der Sprache, dem Interface und der Ausführungsplattform möglich ist [vgl. Weg96, S.285]. Für die *Interoperabilität* gibt es zwei Möglichkeiten. Entweder die Standardisierung der Interfaces oder das Verwenden einer Bridge, welche die verschiedenen Interfaces gegenseitig aufeinander abbildet [vgl. Weg96,

S.286]. Die Standardisierung der Interfaces ist bei den verschiedenen Messengern nur schwer möglich, da sich hier viele verschiedene Organisationen einigen müssten und einige möglicherweise kein einheitliches Interface umsetzen möchten. Somit bleibt nur die Verwendung einer *Bridge*, welche die Kommunikation zwischen den Messengern übernimmt. Dabei „übersetzt“ die *Bridge* die Nachrichten des einen Chatsystems in das Format des anderen und übernimmt die Synchronisation der Nachrichten (Listing 4). Diese Art von *Interperabilität* bei Messengern wird als *Bridging* bezeichnet.



Listing 4: Bridging

4 Chatsysteme in der digitalen Lehre

4.1 Einsatzmöglichkeiten eines Chatsystems

Diese Arbeit beschäftigt sich mit der Integration eines Chatsystems zur Verwendung in der digitalen Lehre, doch warum wird ein Chatsystem überhaupt in der digitalen Lehre benötigt?

Da Messenger (Chatsysteme) im alltäglichen Bereich durch die starke Verbreitung von Smartphones eines der Hauptkommunikationsmittel geworden sind [vgl. Bun21, S.4], werden diese in allen Bereichen des Lebens eingesetzt, unter anderem auch von Studenten/Schülern zum Austausch und zur Besprechung von Lehrinhalten. Hierbei haben viele der verwendeten Messenger allerdings Probleme mit dem Datenschutz und die Nutzung derer im Bereich der Lehre wurde auch von einigen zuständigen Ministerien nicht empfohlen oder untersagt [vgl. Min][vgl. Hes15]. Somit kann ein Chatsystem, welches in ein System für die digitale Lehre eingebunden ist, diese Messenger für den Lehrbereich ersetzen und eine datenschutzkonforme Alternative bieten.

Des Weiteren ist es auch möglich, den Chat für den eigentlichen Lehrprozess zu verwenden. Hierbei ist es zum Beispiel möglich, Aufgaben (Dateien) über den Chat zu verteilen, welche dann von Schülern/Studenten über diesen beantwortet werden können [vgl. Rah20, S.77]. Ein weiterer Verwendungszweck für ein Chatsystems ist zum Beispiel das Durchführen von *literature circles* (Literaturkreise), wie eine Studie [Ivi20] der *Faculty of Humanities and Social Sciences in Osijek* zeigt.

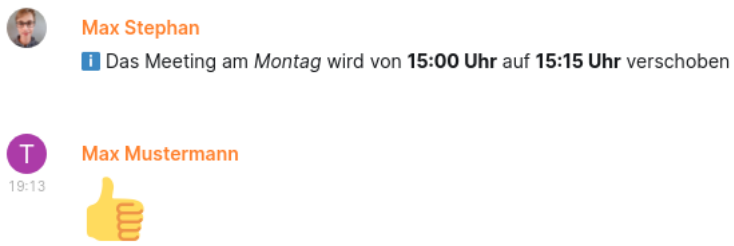
Doch welche Funktionen benötigt ein Chat im Kontext der digitalen Lehre?

4.2 Funktionsanforderungen

Um festzulegen, welche Funktionen ein Chatsystem benötigt, ist es sinnvoll, zu sehen, welche Funktionen weitverbreitete Chatsysteme anbieten und was hiervon auch im Kontext der digitalen Lehre hilfreich ist. Um die Integration zu vereinfachen und die Entwicklungszeit zu verringern, ist es des Weiteren wichtig, dass nur Funktionen implementiert werden, die später auch wirklich verwendet werden.

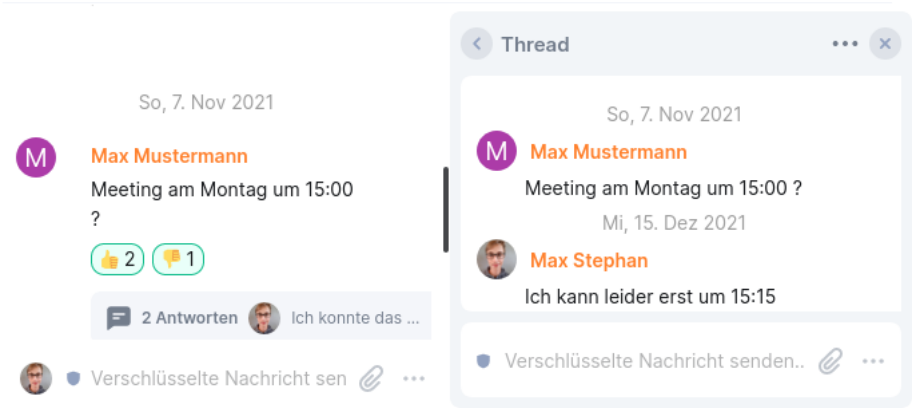
Die grundlegende Funktion eines Chatsystems ist die Möglichkeit, Textnachrichten zu versenden. Hierbei erlauben die meisten Chatsysteme das Versenden aller *Unicode* Zeichen, was es ermöglicht, auch sogenannte *Emojis* (Listing 5) zu versenden. Da *Emojis* auf normalen Tastaturen nicht zu finden sind, bieten die Chatsysteme meist einen sogenannten *Emoji-Picker* an, welcher eine Liste aller *Emojis* bereitstellt, um diese auszuwählen.

Ein Feature, welches die meisten Messenger unterstützen, ist die einfache Formatierung von Texten. Damit ist es möglich, mit bestimmten Zeichen die Formatierung des Textes zu ändern (Listing 5). Die meisten Messenger bieten hierbei nur eine einfache Formatierung an, wie das Fett- oder Kursiv-Schreiben von Texten. Es gibt auch Messenger, die noch weitgehendere Formatierungsmöglichkeiten anbieten, wie zum Beispiel *Discord*. Hierbei wird es ermöglicht, Text-Abschnitte als Code-Blöcke zu markieren, um diese mithilfe von *Syntax-Highlighting* besser lesbar zu machen. Die Formatierung von Texten kann auch im Bereich der digitalen Lehre praktisch sein, um zum Beispiel Informationen in einer Nachricht hervorzuheben.



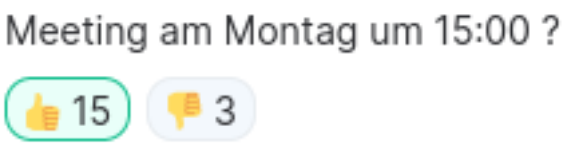
Listing 5: Einfache Formatierung von Texten

Um die Unterhaltungen in einem Chat besser zu organisieren, gibt es in einigen Messengern die Möglichkeit, auf Nachrichten zu antworten. Diese Nachrichten enthalten dann eine Vorschau auf die ursprüngliche Nachricht und eine Möglichkeit, zu dieser Nachricht zu springen. Antworten ermöglichen es, den Kontext für eine Nachricht herzustellen, ohne die vorherige Nachricht zu kopieren. Aber auch diese Funktion kann bei größeren Chatgruppen nicht ausreichend sein, um mehrere Unterhaltungen gleichzeitig zu führen. Damit dies möglich wird, bieten einige Messenger wie zum Beispiel *Discord* oder *Slack* zusätzlich die Möglichkeit sogenannten *Threads* zu erstellen. *Threads* ermöglichen es, nicht im normalen Chat auf eine Nachricht zu antworten, sondern hierfür eine eigene Unterhaltung zu starten, welche in einem Unterfenster angezeigt wird (Listing 6). Diese Funktionen können im E-Learning-Bereich gerade für größere Gruppen hilfreich sein, um die Unterhaltungen besser zu organisieren und Diskussionen besser zu strukturieren.



Listing 6: Threads

Für das schnelle Antworten mit *Emojis* auf Nachrichten, die Bewertung von Nachrichten oder um einfache Abstimmungen durchzuführen, bieten die Messenger *Signal* und *Discord* sogenannte *Reaktionen* an. Es ist hiermit möglich, mit *Emojis* auf Nachrichten zu *reagieren* (Listing 7). Diese Funktion kann dabei helfen, überflüssige Nachrichten zu verhindern, da zum Beispiel eine Zustimmung oder Ablehnung für einen in der Nachricht vorgeschlagenen Termin nicht mit einer neuen Nachricht gezeigt werden muss (Listing 7). Dies wiederum führt zu einem übersichtlicheren Chatverlauf, was verhindert, dass mögliche wichtige Nachrichten übersehen werden.



Listing 7: Reaktionen

Um wichtige Nachrichten noch besser hervorzuheben, bieten *Discord* und *Slack* die Möglichkeit, Nachrichten *anzuheften*. Dabei werden diese Nachrichten in einem gesonderten Bereich angezeigt, um diese schnell wiederzufinden (Listing 8). Diese Funktion ist praktisch, um häufig gestellte Fragen oder wichtige Ankündigungen hervorzuheben damit diese einfach wiedergefunden werden können.

Listing 8: Angeheftete Nachrichten

Ein weiteres praktisches Feature für das E-Learning sind Umfragen, die zum Beispiel von *Telegram* oder *Threema* unterstützt werden. Diese können eingesetzt werden, um Entscheidungen zu treffen oder um Aufgaben zu bearbeiten.

Das Teilen von Bildern oder Dateien wird auch von nahezu alle Messenger-Diensten unterstützt. Bei einem Chatsystem, welches in ein E-Learning System integriert ist, gibt es für dieses Feature noch einige Zusatzanforderungen. Da ein E-Learning System möglicherweise schon über die Möglichkeit verfügt, Dateien hochzuladen und diese zum Beispiel als Aufgaben zur Verfügung zu stellen, sollte dieses mit dem Chat integriert werden. Dabei sollte es möglich sein, die schon vorhandenen Dateien in dem Chat zu teilen ohne diese erneut hochzuladen. Auch sollte es möglich sein, auf diese Dateien oder Aufgaben zu verweisen, um Fragen oder Anmerkungen direkt im Chat zu stellen.

Aus den in diesem Abschnitt beschriebenen Funktionen ergibt sich nun eine Liste von Funktionen, die das gewünschte Chatsystem unterstützen sollte:

- Emojis
- Einfache Text-Formatierung
- Antworten
- Threads
- Reaktionen
- Angeheftete Nachrichten
- Umfragen
- Versenden von Bildern oder Dateien

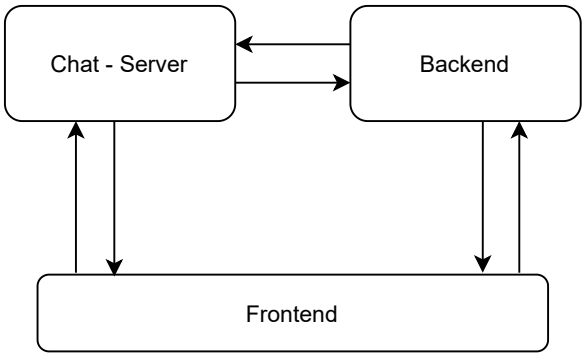
4.3 Chatsysteme in bekannten Lernplattformen

Wie zuvor in Abschnitt 4.1 beschrieben, besitzt ein Chat einige Einsatzmöglichkeiten im Bereich der digitalen Lehre, doch wie wird dieser in bekannten Lernplattformen unterstützt? Die bekannten Lernplattformen *Moodle*, die laut *Moodle an Hochschulen e.V.* an 180 deutschen Hochschulen eingesetzt wird [vgl. Moo22b] und *Ilias*, welches zum Beispiel an der *Justus-Liebig-Universität Gießen* oder der *Philipps-Universität Marburg* verwendet wird, bieten die Möglichkeit, Chat-Nachrichten zu versenden. Hierbei bieten diese allerdings nur einen Chat mit wenigen Funktionen an. Dabei ist es nur möglich, einfache Textnachrichten und *Emojis* zu versenden. Der Verglich mit diesen Systemen bietet sich nicht an, da der gewünschte Funktionsumfang des in dieser Arbeit entwickelten Chatsystems höher ausfällt.

5 Konzept

Damit für die Integration einer Chat-Funktion in Audimax ein möglichst geringer Aufwand benötigt wird, soll ein vorhandenes Chatsystem integriert und verwendet werden. Diese Integration soll keine Auswirkungen auf die Benutzerfreundlichkeit des Systems haben und der Benutzer sollte das Gefühl eines einzigen Systems haben.

Dies im Backend zu erreichen, ist ohne Probleme möglich. Da der Benutzer nicht mitbekommt, mit welchem Server er kommuniziert, können hier einfach mehrere Server ausgeführt werden, die verschiedenste Aufgaben erledigen. Somit ist es möglich, den Server eines vorhandenen Chatsystems zu verwenden (Listing 9). Der *Chat-Server* und das vorhandene Backend können hierbei auch untereinander kommunizieren, um Informationen wie zum Beispiel die Benutzer-Informationen zu synchronisieren.



Listing 9: Integration des Chatsystems

Im Frontend ist es nicht so leicht möglich, ein vorhandenes Chat-Frontend zu integrieren, ohne dass der Benutzer dies bemerkt. Dabei gibt es mehrere Probleme: Einerseits können Doppelungen im Source-Code dafür sorgen, dass die Ladezeiten erhöht werden, andererseits gibt es möglicherweise Überscheidungen in der Oberfläche oder es wird nur ein Teil der vorhandenen Oberfläche benötigt. Ein weiteres Problem hierbei ist es, dass die Designs der beiden Systeme möglicherweise nicht übereinstimmen, was dazu führt, dass der Benutzer einen klaren Unterschied erkennt.

Um diese Probleme zu lösen, wäre es zwar möglich, aus dem vorhandenen Source-Code des Chatsystem Teile zu kopieren und diese zu verwenden. Dabei gibt es allerdings

auch mehrere Probleme: Einerseits muss die *Lizenz* des Chatsystems dies erlauben und andererseits ist es hierbei sehr schwer, Updates des vorhandenen Systems zu übernehmen.

Um die gewünschte Integration mit den geringsten Problemen durchzuführen, ist es also am besten, das Frontend selbst und mit der Technik und im Style des vorhandenen Systems zu schreiben. Doch welches Chatsystem ist am besten für solch eine Integration geeignet?

5.1 Vergleich der Chatsysteme

In diesem Abschnitt folgt nun ein Vergleich der Chatsysteme. Im Abschnitt 5.2 folgt dann auf Basis dieses Vergleiches eine Entscheidung, welches Chatsystem am besten für eine Integration geeignet ist.

5.1.1 Funktionen

In diesem Abschnitt erfolgt ein Vergleich, wie gut die Chatsysteme die in Abschnitt 4.2 definierten Chatfunktionalitäten umsetzen.

Die grundlegende Funktion, Nachrichten mit jeglichen *Unicode* Zeichen und somit auch *Emojis* zu versenden, beherrschen alle Chatsysteme. Die Möglichkeit, auf Nachrichten zu antworten und Dateien zu versenden, wird auch von allen Systemen unterstützt.

Threads und *angeheftete Nachrichten* werden nur von *Discord*, *Slack*, *Microsoft Teams* und *Matrix* unterstützt. Zusätzlich unterstützt *Telegram* auch noch *angeheftete Nachrichten*.

Reaktionen werden von allen Chatsystemen außer *WhatsApp* und *Nextcloud Talk* unterstützt. In *Nextcloud Talk* sind diese allerdings eine geplante Funktion, die im Jahr 2022 umgesetzt werden soll [vgl. Ehr19]. In *Threema* werden *Reaktionen* zwar unterstützt, diese sind allerdings eingeschränkt. Bei den anderen Chatsystemen ist es möglich mit mehreren *Emojis* auf eine Nachricht zu reagieren, in *Threema* ist es nur möglich, eine Nachricht zu bestätigen (*Daumen hoch*) oder diese abzulehnen (*Daumen runter*).

Umfragen werden vollständig von *Telegram*, *Threema* und *Microsoft Teams* unterstützt. In Matrix gibt es aktuell (Stand 31.12.2021) nur einen *Proposal* (Vorschlag) für die Aufnahme von Umfragen in den Matrix-Standard. Dies bedeutet zwar, dass es möglich ist, Umfragen zu implementieren, wie es die Referenz-Implementierung *Element* schon macht, allerdings ist es möglich, dass der Standard sich hierfür noch häufiger ändert und somit die Implementierung angepasst werden muss. In Nextcloud Talk gibt es

von Haus aus keine Umfragen, es gibt allerdings eine Erweiterung (*talk_simple_poll*), die Umfragen ermöglicht. Diese Umfragen haben allerdings keine Schaltflächen zum anklicken, sondern die Abstimmung ist nur über Chat-Nachrichten möglich. Dies macht die Umfragen in Nextcloud Talk weniger intuitiv, was vor allem Nutzern mit einem weniger ausgeprägten technischen Verständnis Schwierigkeiten bereiten kann. In *Slack* ist es möglich, die Funktionalität dessen durch sogenannte *Apps* zu erweitern. Hierbei gibt es auch mehrere *Apps*, die das Erstellen von Umfragen ermöglichen.

Funktionen aus Abschnitt 4.2	Telegram	WhatsApp	Signal	Discord	Threema
Emojis	✓	✓	✓	✓	✓
Einfache Text-Formatierung	✓	✓	✗	✓	✓
antworten	✓	✓	✓	✓	✓
Threads	✗	✗	✗	✓	✗
Reaktionen	✓	✗	✓	✓	✓
Angeheftete Nachrichten	✓	✗	✗	✓	✗
Umfragen	✓	✗	✗	✗	✓
Bildern oder Dateien	✓	✓	✓	✓	✓

Listing 10: Funktionsvergleich der Chatsysteme (Teil 1)

Funktionen aus Abschnitt 4.2	Slack	Microsoft Teams	Matrix	Nextcloud Talk
Emojis	✓	✓	✓	✓
Einfache Text-Formatierung	✓	✓	✓	✗
antworten	✓	✓	✓	✓
Threads	✓	✓	✓	✗
Reaktionen	✓	✓	✓	✗
Angeheftete Nachrichten	✓	✓	✓	✗
Umfragen	✓	✓	✓	✓
Bildern oder Dateien	✓	✓	✓	✓

Listing 11: Funktionsvergleich der Chatsysteme (Teil 2)

5.1.2 Integrationsmöglichkeiten

Um das Chatsystem in das vorhandene System zu integrieren, ist es notwendig, mit diesem zu kommunizieren. Deshalb ist es wichtig, welche Schnittstellen das Chatsystem bereitstellt. Des Weiteren ist es für die maximale Datenkontrolle und der damit verbundenen Datensicherheit notwendig, dass es möglich ist, den Service selbst zu hosten, also nicht auf vorhandene Server des Anbieters zurückgreifen zu müssen.

Die Chatsysteme *Telegram*, *WhatsApp*, *Discord*, *Slack*, *Microsoft Teams* und *Threema* bieten keinerlei Integrationsmöglichkeiten, da es nicht möglich ist, diese selbst zu hosten.

Dies liegt daran, dass deren Source-Code nicht öffentlich oder nur teilweise öffentlich verfügbar ist und es somit nicht gewollt und auch nicht möglich ist, diese selbst zu hosten.

Bei dem Chatsystem *Signal* ist es zwar möglich, eigene Server zu hosten, es gibt allerdings ein anderes Problem bei der Integration. In *Signal* muss jeder Benutzer eine Telefonnummer besitzen, da die grundlegende Identifikation der Benutzer darüber funktioniert. Da es in *Audimax* nicht gefordert sein soll, eine Telefonnummer anzugeben, und dies in Zukunft nicht geplant ist, führt dies zu dem Problem, dass eine Einbindung von *Signal* nicht oder nur schwer möglich ist.

Bei Nextcloud Talk und Matrix ist es möglich, eigene Server zu hosten. Außerdem verfügen diese über eine Rest-Schnittstelle, über die jegliche Funktionen gesteuert werden können. Hierbei stellt Matrix auch *SDKs* (Abschnitt 3.2) in verschiedensten Sprachen bereit, welche die Kommunikation vereinfachen und zum Beispiel die Synchronisation der Nachrichten übernehmen können. Nextcloud Talk bietet darüber hinaus die Möglichkeit, jegliche Funktionen über eine *PHP-API* auszuführen. Mit anderen Nextcloud-Apps ist es möglich, diese aufzurufen und somit diese Funktionen auch „Nextcloud intern“ zu verwenden.

Da es nur bei *Nextcloud Talk* und *Matrix* möglich ist, diese in dem gewünschten Maße zu integrieren, werden nur diese beiden Chatsysteme im restlichen Vergleich berücksichtigt. Im Abschnitt 5.2 werden dann allerdings nochmals alle Chatsysteme aufgegriffen, um zu sehen, wie gut das ausgewählte Chatsystem im Punkt der Funktionalität gegenüber den anderen Systemen abschneidet.

5.1.3 Dokumentation

Bei der Integration eines Chatsystems spielt eine gute Dokumentation eine wichtige Rolle, da diese die Integration stark erleichtern kann.

Beide Chatsysteme besitzen eine API-Dokumentation, also eine Dokumentation der Schnittstellen zum Backend. Die Dokumentation von Matrix ist hierbei jedoch wesentlich ausführlicher. Diese enthält nicht nur die Auflistung der Schnittstellen mit Parametern, welche auch die Dokumentation von Nextcloud Talk enthält, sondern auch noch ausführliche Erklärungstexte, wie diese Schnittstellen zu verwenden sind. Darüber hinaus enthält die Matrix-Dokumentation auch noch Anleitungen für die Implementierung bestimmter Funktionen.

Eine Installations- und Konfigurations-Anleitung ist bei Nextcloud Talk und auch bei den Matrixreferenz-Implementierungen (zum Beispiel *Synapse*) vorhanden. Die Matrix-Anleitungen sind ausführlicher und enthalten auch Optionen für verschiedenste

Betriebssysteme. Das heißt allerdings nicht unbedingt, dass die Anleitungen von Nextcloud Talk schlechter sind. Der Unterschied entsteht dadurch, dass Nextcloud Talk durch die Integration in Nextcloud leichter zu installieren ist und somit keine so ausführlichen Anleitungen benötigt.

5.1.4 Erweiterbarkeit

Damit das Chatsystem gut in ein vorhandenes System integriert werden kann, ist es auch wichtig, zu wissen, wie einfach dieses erweitert werden kann, da für die Integration möglicherweise Teile des vorhandenen Chatsystems erweitert werden müssen.

In Matrix ist es möglich, *Events* zu versenden, welche einen eigens definierten *Event-Typen* haben (Events werden im Abschnitt 5.3 näher erklärt). Dies ermöglicht es, zusätzliche Informationen in Chat-Räumen zu speichern oder Nachrichten mit einem eigenen Layout zu versenden. Hierdurch ist es zum Beispiel möglich, eine Nachricht zu versenden, die eine Referenz auf eine Aufgabe enthält, um hierzu eine Frage zu stellen. In Nextcloud Talk ist so etwas nicht möglich. Hier ist es nur möglich, die *Aktionen*, welche für Nachrichten angeboten werden, wie zum Beispiel das Antworten auf eine Nachricht, zu erweitern. Diese Funktion ist nicht dafür gedacht, um die Chat-Funktionalität zu erweitern, sondern nur um mit dem Inhalt der Chat-Nachricht etwas zu erstellen (z.B. Kalender-Events).

Die Matrix Referenz-Implementierung *Synapse* bietet außerdem noch die Möglichkeit, die Funktionalität des *Synapse-Servers* mit *Modulen (Modules)* zu erweitern. Diese ermöglichen es zum Beispiel, *Spamfilter* einzurichten, externe Benutzer zu authentifizieren oder gesendeten Dateien auf einem externen Cloud-Server zu speichern.

5.1.5 Sicherheit

Ein weiteres wichtiges Kriterium ist die Sicherheit der Chat-Nachrichten. Hierbei ist es wichtig, welche Verschlüsselungsmöglichkeiten die Chatsysteme unterstützen.

Beide Chatsysteme bieten eine Transportverschlüsselung über *HTTPS/TLS* an. Diese verschlüsselt den Datenverkehr zwischen *Client* und *Server* und sorgt dafür, dass die versendeten Nachrichten nicht von dritten gelesen werden können.

Matrix unterstützt zusätzliche noch eine *Ende-zu-Ende-Verschlüsselung* (E2E). Diese sorgt dafür, dass nur der Sender und die Empfänger die Nachricht lesen können (Abschnitt 3.1). Nextcloud Talk unterstützt aktuell keine weiteren Verschlüsselungen, eine *E2E* und/oder eine serverseitige Verschlüsselung ist allerdings geplant [vgl. J0W19].

5.1.6 Weiterentwicklung

Da nach der fertiggestellten Integration eines Chatsystems ein Wechsel auf ein anderes Chatsystem nur schwer möglich ist, ist es wichtig, ein Chatsystem auszuwählen, welches stetig weiterentwickelt und gewartet wird. Da beide Chatsysteme Open-Source sind und ihren Source-Code auf *Github* bereitstellen, ist es möglich, anhand der *Github Insights* zu sehen, wie aktiv diese Systeme im letzten Monat weiterentwickelt wurden.

Im Nextcloud Talk Source-Code wurden im letzten Monat von 9 Personen 102 *commits (Änderungen)* vorgenommen [vgl. Gitc]. An der Matrix-Spezifikation wurden in dem gleichen Zeitraum von 17 Personen 14 *commits (Änderungen)* vorgenommen [vgl. Gita]. Hierbei sind die Zahlen der beiden Chatsysteme nicht vergleichbar, da die Statistik bei Nextcloud Talk die Entwicklung des gesamten Source-Codes einschließt und bei Matrix nur die Entwicklung der Spezifikation. Einen besseren Vergleichswert bietet hierbei die Statistik der Matrix Referenz-Implementierung *Synapse*, an welcher im letzten Monat 25 Personen 137 *commits (Änderungen)* vorgenommen haben [vgl. Gitb]. An beiden Chatsystemen wird also aktiv weiterentwickelt und es ist zu erwarten, dass diese auch in Zukunft mit *updates* und *bugfixes (Fehlerbehebungen)* versorgt werden.

5.1.7 Bridging

Damit ein Wechsel von einem alten Chatsystem oder auch die Verwendung von mehreren Chatsystemen gleichzeitig möglich ist, um zum Beispiel eine Übergangsphase zu bieten, ist es praktisch, wenn das Chatsystem die Kommunikation mit anderen Chatsystemen ermöglicht.

Matrix und Nextcloud Talk ermöglichen dies über *Bridges* (Abschnitt 3.6). In Matrix gibt es *Bridges* für die Kommunikation mit 29 verschiedenen Programmen [vgl. Matb]. Nextcloud Talk verwendet für die Umsetzung von *Bridges* das Programm *matterbridge* [42w], welches die Kommunikation mit 25 verschiedenen Programmen unterstützt [vgl. 42w]. Des Weiteren ist es bei beiden Systemen möglich, eigene *Bridges* zu entwickeln, falls eine *Bridge* zu einem nicht unterstützten Programm benötigt wird.

5.2 Auswahl des Chatsystems

Doch welches der beiden Chatsysteme ist nun besser für die Integration geeignet?

In dem Punkt der Weiterentwicklung unterscheiden sich die beiden Chatsysteme nicht viel, da beide aktiv weiterentwickelt werden. In Sachen Dokumentation ist Matrix gegenüber Nextcloud Talk etwas überlegen, da hier auch noch Anleitungen und weiterführende

Informationen bereitgestellt werden. Bei den Kriterien Sicherheit, Erweiterbarkeit, Integrationsmöglichkeiten, Bridging und Funktionen ist Matrix auch Nextcloud Talk überlegen. Matrix bietet wesentlich mehr der gewünschten Funktionen, mehr Möglichkeiten zur Erweiterung und Ende-zu-Ende-Verschlüsselung der Nachrichten. Des Weiteren sind die bereitgestellten SDKs für die Integration sehr hilfreich, um die Entwicklung des eigenen Frontends zu vereinfachen, was den Zeitaufwand stark reduziert. Somit ist Matrix das bessere Chatsystem für die gewünschte Integration.

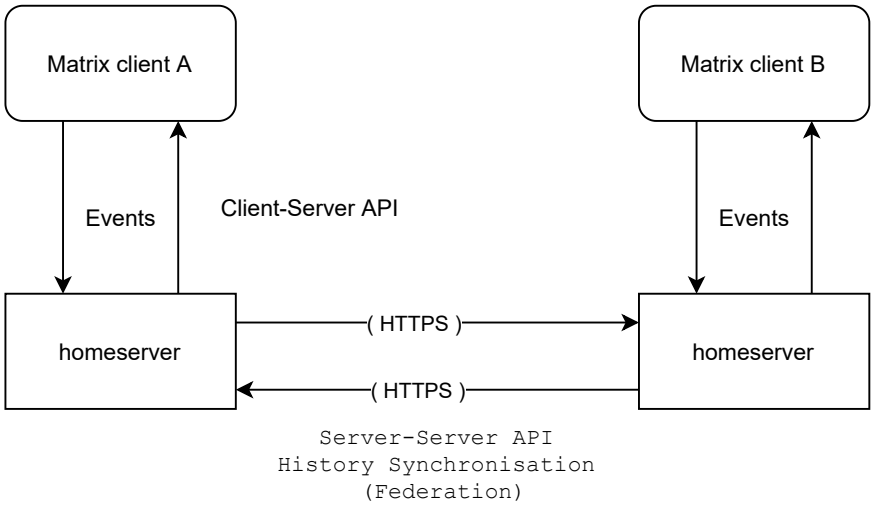
Abschließend sei noch bemerkt, dass in Sachen Funktionalität Matrix den anderen Chatsystemen (Abschnitt 5.1.2) in nichts nach steht, da es alle der gewünschten Funktionen unterstützt.

5.3 Konzept zur Integration

Zunächst folgt eine Erklärung der Funktionsweise von Matrix und danach das Konzept zu Integration.

5.3.1 Matrix Architektur

Grundlegend wird jegliche Interaktion in Matrix mithilfe von sogenannten *Events* durchgeführt. Diese werden zwischen Client und Server versendet und repräsentieren zum Beispiel eine Nachricht oder eine Einstellung. Hierbei kommuniziert ein Client über die *Client-Server-Api* mit einem Matrix-Server, welcher als *homeserver* bezeichnet wird. Um festzulegen, wer die versendeten *Events* erhält, werden diese im Kontext von virtuellen Räumen gesendet. Dabei kann ein Nutzer (Client) in mehreren Räumen sein, die wiederum mehrere Nutzer beinhalten können. Die Nutzer in einem Raum müssen nicht den gleichen *homeserver* verwenden, sondern können auf mehrere Server verteilt sein. Hierzu synchronisieren die *homeserver Events* mittels der *Server-Server API* (Listing 12). [vgl. Mat21a, Kap. 3]

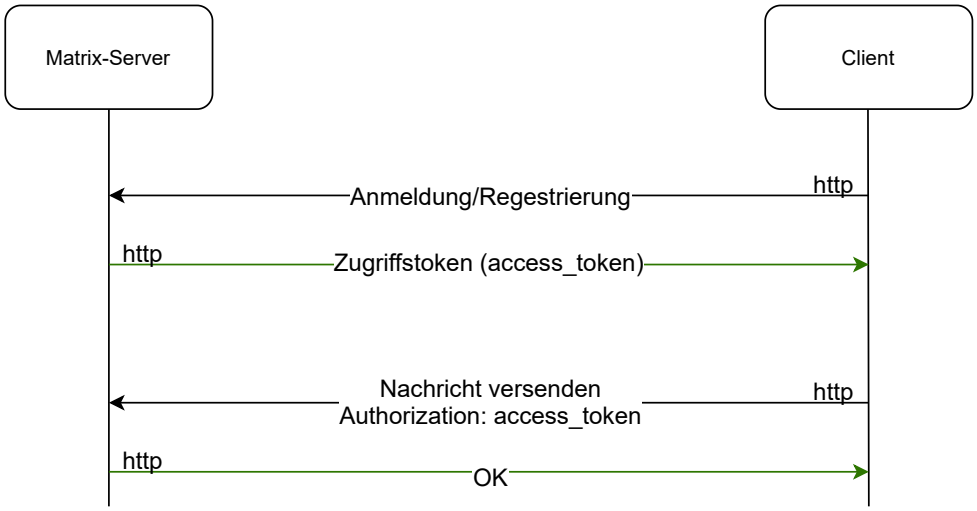


Listing 12: Matrix Architektur [Mat21a, Kap. 3]

In Matrix gibt es zwei Arten von *Events*, die in einem Raum gesendet werden können. Einerseits die *State Events*, welche die Einstellungen eines Raumes, wie zum Beispiel das Raum-Thema, speichern und verändern und andererseits *Message Events*, welche die Kommunikation wie zum Beispiel eine Textnachricht abbilden. Dabei besitzt jedes *Event* einen *Type (Art)*, der die Funktion des *Events* darstellt. Das *Event* zum Versenden einer Nachricht hat zum Beispiel den Type *m.room.message*. Hierbei sind die *Events* nicht auf die vorhandenen Typen beschränkt und es ist auch möglich, eigene zu definieren. [vgl. Mat21c, Kap. 7.1]

In Matrix bestehen die Benutzer-IDs aus den zwei Teilen *localpart* und (Server-) *domain*, welche im Format *@localpart:domain* zusammengesetzt werden [vgl. Mat21b, Kap. 4.2.1]. Der *localpart* identifiziert hierbei den Benutzer eindeutig auf dem *homeserver* und die *domain* wird verwendet, um gleiche IDs auf verschiedenen Matrix-Servern zu vermeiden.

Bei den meisten Anfragen an den *Matrix-Server* ist es notwendig, dass der Nutzer sich authentifiziert, um zu bestätigen, dass dieser die gewünschte Aktion durchführen kann. Dies geschieht in *Matrix* mithilfe eines *access_tokens* (Zugriffstoken) der bei jeder Anfrage an den Server mitgesendet werden muss. Der *Client* bekommt diesen Token normalerweise durch die Anmeldung oder bei der Registrierung am *Matrix-Server* (Liting 13). [vgl. Mat21c, Kap. 4]

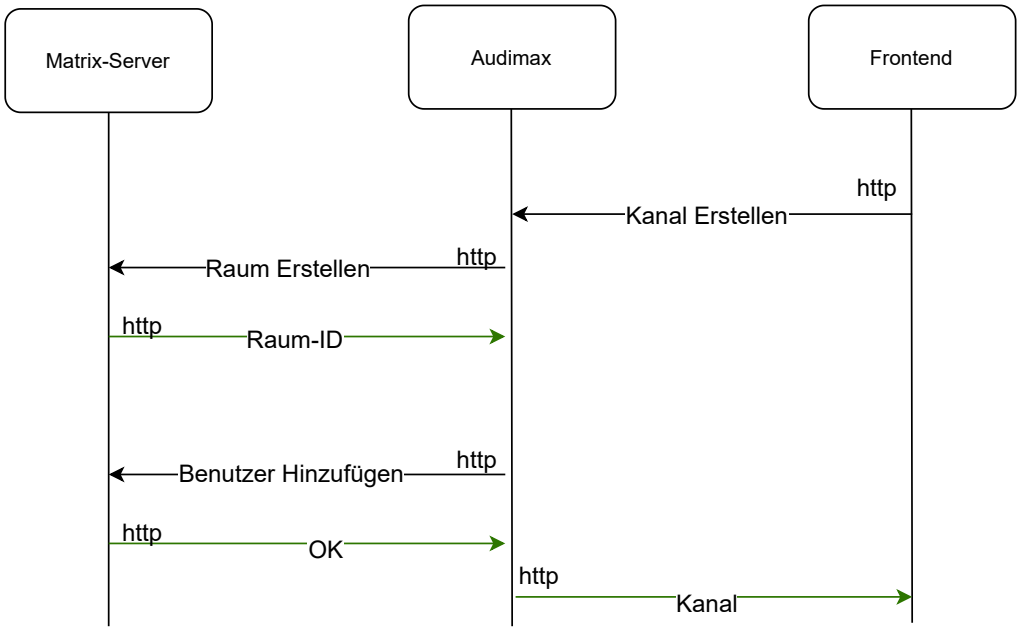


Listing 13: Authentifikation in Matrix [vgl. Mat21c, Kap. 4]

5.3.2 Backend-Integration

Wie zu Beginn dieses Kapitels erläutert, soll ein vorhandener Matrix-Server für die Backend-Integration verwendet werden (Listing 9). Als Matrix-Server wird die Referenz-Implementierung *Synapse* verwendet, da dieser Server aktuell (Dezember 2021) die meisten Funktionen und Stabilität bietet.

Damit das *Audimax-Backend* Räume erstellen, Nutzer zu diesem hinzufügen und Rechte der Nutzer verändern kann, muss dieses mit dem Matrix-Server kommunizieren. Diese geschieht mithilfe von *http-Requests* über die *Client-Server-API* [Mat21c] des Matrix-Servers. Diese *API* ermöglicht es dem Audimax-Backend, Räume zu erstellen und diese zu verändern (Listing 14). Zwar ist es auch möglich, über diese *API* Benutzer zu Räumen hinzuzufügen, allerdings wird hierbei der Nutzer nur eingeladen. Dies führt zu dem Problem, dass die Nutzer für jeden Chat-Kanal eine Einladung annehmen müssten, um den Chat verwenden zu können. Um dieses Problem zu umgehen, bietet *Synapse* eine *Admin-API* [Mata] an, die es ermöglicht, Nutzer direkt zu einem Raum hinzuzufügen. Diese beiden *APIs* ermöglichen es, alle notwendigen Aktionen durchzuführen und das Chatsystem im Backend vollständig zu integrieren. Jede Anfrage an die *Client-Server-API* oder die *Admin-API* benötigt einen Nutzer, mit dem die Anfragen durchgeführt werden. Daher benötigt das *Audimax-Backend*, um diese *APIs* zu verwenden, einen *Synapse-Nutzer*. Dieser sollte einmalig angelegt werden und dessen Daten im Backend zur Anmeldung hinterlegt werden.



Listing 14: Chat-Kanal erstellen

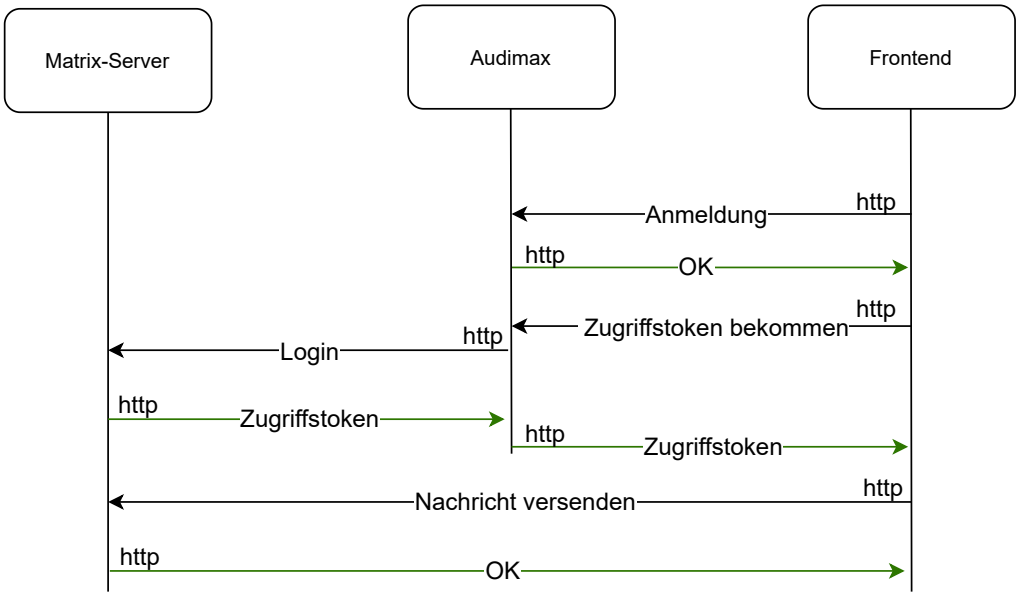
Nicht nur das *Audimax-Backend* benötigt einen *Synapse-Nutzer*, sondern jeder *Audimax-Nutzer* muss *Synpase* bekannt sein, damit dieser zum Beispiel Nachrichten versenden kann. Dabei gibt es zwei Probleme zu lösen: Einerseits müssen die Benutzer zwischen dem *Synapse-Server* und dem *Audimax-Backend* synchronisiert werden und andererseits muss es für die Nutzer möglich sein, sich bei *Synapse* zu authentifizieren.

Um diese Probleme zu lösen, sollen die Module (Modules) des *Synapse-Servers* verwendet werden, da diese es zum Beispiel ermöglichen, die Authentifikationsmöglichkeiten von *Synapse* zu erweitern und somit auch externe Benutzer zu authentifizieren. Bei dieser Lösung gibt es allerdings noch ein weiteres Problem: Da das *Audimax-Backend* und der *Synapse-Server* verschiedene Methoden zur Authentifikation der Benutzer verwenden, müsste sich der Nutzer bei beiden Services anmelden, um sich zu authentifizieren. Zwar wäre es möglich, dies im Frontend durchzuführen, ohne dass der Nutzer mitbekommt, dass er sich bei zwei Servern anmeldet, dies kann allerdings zu mehreren Fehlern führen:

Damit die Nutzer sich nicht bei jedem Öffnen von Audimax anmelden müssen, wird ein *Token* gespeichert, mit dem sich der Benutzer autorisieren kann. Durch das Integrieren des Chatsystems müssten nun also zwei Tokens gespeichert werden, einer für *Audimax* und ein anderer für den *Matrix-Server*. Dies kann zu dem Problem führen, dass es einerseits möglich ist, dass nur ein Token gelöscht/gespeichert wurde oder aber auch die beiden Tokens unterschiedlichen Benutzern gehören. Ein weiteres Problem bei einer solchen doppelten Anmeldung besteht darin, dass *Matrix* nicht die gleichen Login-Methoden wie das *Audimax-Backend* unterstützt. *Audimax* unterstützt bei der Anmeldung ein Verfahren namens *Zwei-Faktor-Authentisierung*, dies ermöglicht es, dass der Benutzer

bei der Anmeldung nicht nur sein Passwort eingeben muss, sondern seine Identität noch mit einem zweiten Faktor bestätigen muss. Dies ist zum Beispiel mit einer Smartphone-App, welche mithilfe des *TOTP (Time-based One-time Password)* [M'R11] Verfahrens Bestätigungscodes generiert, möglich. Zwar sind die meisten dieser Probleme mit etwas Aufwand lösbar, es gibt allerdings noch eine einfachere und bessere Lösung:

Um die Doppelanmeldung zu vermeiden, sollte es für einen angemeldeten Audimax-Nutzer möglich sein, bei dem *Audimax-Backend* einen *access_token* (Zugriffstoken) für den *Matrix-Server* anzufragen und sich somit bei beiden Services anzumelden (Listing 15). Damit das *Audimax-Backend* einen solchen Token generieren kann, benötigt es eine Möglichkeit, um Token für jegliche Benutzer bei dem *Matrix-Server* anzufragen. Hierzu gibt es das Modul *Shared Secret Authenticator* [Dev], welches es ermöglicht, sich mithilfe eines geteilten Secrets anzumelden und somit einen *access_token* (Zugriffstoken) zu generieren.



Listing 15: Anmeldung am Matrix-Server [Mat21a, Kap. 3]

5.3.3 Frontend-Integration

Um die Frontend-Integration zu vereinfachen, soll eine der bereitgestellten *SDKs* verwendent werden. Da das *Audimax-Frontend* in TypeScript geschrieben ist, wird die *Matrix Javascript SDK* [Mate] verwendet. Diese übernimmt jegliche Kommunikation mit dem Matrix-Server und sorgt somit auch dafür, dass neue *Events* immer in Echtzeit ankommen.

6 Realisierung

Die Umsetzung des Konzeptes unterteilt sich in drei Aufgabenbereiche. Zuerst muss der Matrix-Server (*Synapse*) aufgesetzt und konfiguriert werden. Hiernach geht es um die Integration in das vorhandene System, dabei ist es wichtig, dass die vorhandenen Benutzer sowie *Kurse/Kanäle* mit dem Matrix-Server synchronisiert werden. Zuletzt geht es um die Kommunikation zwischen dem Frontend und dem Matrix-Server unter Verwendung der *Matrix Javascript SDK*. Bei der Realisierung wurde die Funktion der *Ende-zu-Ende-Verschlüsselung* von Chat-Nachrichten aus Zeitgründen nicht umgesetzt. Um diese zu unterstützen, ist es daher möglich, dass einige Änderungen an den folgenden Code-Abschnitten oder auch an einigen Konzepten notwendig sind.

6.1 Konfiguration

Um den *Synapse* Server aufzusetzen wurde die offizielle Beispiel *docker-compose*-Datei [Matg] verwendet. Um *Synapse* konfigurieren zu können, muss zuerst eine Konfigurations-Datei erstellt werden, dies ist mit dem Befehl in Listing 16 möglich. Hierbei muss der Servername, welcher normalerweise der Domain des Matrix-Servers entspricht, angegeben werden. Zusätzlich kann noch die Option *SYNAPSE_REPORT_STATS* angegeben werden, welche bestimmt, ob anonyme Nutzungsdaten an die Entwickler gesendet werden sollen [vgl. Matg].

```
docker-compose run --rm -e
 ↪  SYNAPSE_SERVER_NAME=my.matrix.host -e
 ↪  SYNAPSE_REPORT_STATS=yes synapse generate
```

Listing 16: Synapse Konfigurationsdatei erstellen [Matg]

Um die anfallenden Daten wie zum Beispiel Chat-Nachrichten zu speichern, verwendet *Synapse* eine Datenbank. In der generierten Konfigurations-Datei wird hierbei standardmäßig eine *SQLite* Datenbank verwendet. Allerdings empfiehlt *Synapse* diese durch eine *PostgreSQL* Datenbank auszutauschen, da diese eine wesentlich bessere Performance bietet und auch auf einem anderen Server ausgeführt werden kann [vgl. Matd]. Da

die verwendete *docker-compose*-Datei schon eine *PostgreSQL* Datenbank enthält, muss, um diese zu verwenden, nur die Konfigurations-Datei angepasst werden. Hierzu muss nur unter dem Punkt *database* die alte Konfiguration gelöscht werden und durch die für Postgres ersetzt werden (Listing 17). Hierbei ist es wichtig, dass die Einträge *password* und *user* mit den Werten in der *docker-compose*-Datei übereinstimmen. Das in dem Beispiel verwendete Passwort sollte auch durch ein sicheres Passwort ersetzt werden.

```
database:
  name: psycopg2
  txn_limit: 10000
  args:
    user: synapse_user
    password: changeme
    database: synapse
    host: localhost
    port: 5432
    cp_min: 5
    cp_max: 10
```

Listing 17: Postgress Datenbank verwenden [Matg]

6.2 Authentifikation

Um den in Abschnitt 5.3.2 beschriebenen Nutzer für das *Audimax-Backend* zu erstellen, bietet *Synapse* die Möglichkeit, dies über die Kommandozeile durchzuführen (Listing 18). Hierbei ist der Befehl darauf ausgelegt, dass *Synapse* mit der zuvor beschriebenen *docker-compose*-Datei ausgeführt wird. Falls *Synapse* ohne *docker-compose* ausgeführt wird, wird nur der Befehl in Zeile 2 benötigt. Nach dem Ausführen des Befehls ist es noch notwendig, den Benutzernamen und das Passwort einzugeben. Des Weiteren ist es möglich, dem erstellten Benutzer Admin-Rechte zu geben, was notwendig ist, um die *Synapse-Admin-API* zu verwenden.

```
1 docker-compose run synapse /bin/bash \
2   register_new_matrix_user -c data/homeserver.yaml
    ↪  http://synapse:8448
```

Listing 18: Synapse Benutzer erstellen

Um die Benutzer zwischen *Audimax* und *Synapse* zu synchronisieren, wird, wie im Konzept beschrieben, ein Modul verwendet. Hierfür wird das Modul *REST Password provider* [ma1] verwendet. Dies ermöglicht es, dass *Synapse* sich zur Überprüfung der Benutzerdaten an eine externe Rest-Schnittstelle wendet. Dies ist eine gute Lösung, da

es einfach möglich ist, Audimax um so eine Schnittstelle zu erweitern. Hierbei kann Audimax einfach überprüfen, ob es den Benutzer mit diesem Password gibt (Listing 19). Falls sich nun ein *Audimax-Nutzer* anmeldet, welcher *Synapse* nicht bekannt ist, wird dieser automatisch angelegt.

```
public function checkCredentials(array $user):
 ↪  JSONResponse
{
  $isValid = $this->userService->login($user);

  return new JSONResponse(['auth' => ['success' =>
   ↪  $isValid]]);
}
```

Listing 19: Authentifikation des Benutzers

Hierbei ist allerdings ein Problem aufgetreten: Die erlaubten Zeichen für die IDs der Benutzer von Matrix und Nextcloud (bzw. dem Audimax Backend) sind unterschiedlich (Listing 20), was zu dem Problem führt, dass Nextcloud-Benutzer mit Großbuchstaben oder einem At-Zeichen in der Benutzer-ID nicht in Matrix angelegt werden können.

Matrix-ID *localpart* [Mat21b, Kap. 4.2.1]	Nextcloud Benutzer-ID [Nex]
Kleinbuchstaben (a-z)	Kleinbuchstaben (a-z)
-	Großbuchstaben (A-Z)
Zahlen (0-9)	Zahlen (0-9)
Punkte (.)	Punkte (.)
Unterstriche (_)	Unterstriche (_)
Gleichzeichen (=)	Gleichzeichen (=)
Bindestriche (-)	Bindestriche (-)
Schrägstrich (/)	-
-	At-Zeichen (@)

Listing 20: Vergleich der erlaubten Zeichen in den Benutzer-IDs

Um dieses Problem zu lösen, gab es zwei Optionen. Entweder das Anwenden von Standard-Methoden, um die Benutzer-IDs auf Zahlenstrings abzubildenden (zum Beispiel mit sogenannten *Hash-Methoden*) oder das Generieren eigener Matrix-IDs. Diese Matrix-IDs sollen aus Teilen der ursprünglichen Benutzer-IDs generiert werden und falls notwendig mit Zufallszahlen erweitert werden. Doch welche der beiden Optionen ist besser für den Anwendungsfall geeignet?

Zahlenstrings haben den Nachteil, dass diese lange Zeichenketten sind, die für Benutzer nur schwer unterscheidbar sind. Da es vorkommen kann, dass ein Benutzer diese IDs sieht, haben die generierten Matrix-IDs den Vorteil, dass diese auch für den Benutzer gut unterscheidbar sind. Diese sind zum Beispiel hilfreich, wenn *Bridging* (Abschnitt 3.6) verwendet wird. Ein weiterer Vorteil der Matrix-IDs ist es, dass diese im Normalfall wesentlich kürzer sind als die genierierten Zahlenstrings, was das Eingeben dieser vereinfacht. Aufgrund dieser Vorteile wurde sich für die Erstellung von eigenen Matrix-IDs entschieden.

Bei der Erstellung der eigenen Matrix-IDs werden die ersten 4 Konsonanten der Nextcloud Benutzer-ID genommen und weitere drei zufällige Zahlen angehangen. Falls diese noch nicht ausreichen, um eindeutig zu sein, werden noch weitere Zahlen hinzugefügt (Listing 21).

```
$matrixUid = getLowercaseLetters($uid, 4);

$exist = true;
while ($exist) {
    $matrixUid = addRandomNumbers($matrixUid);
    $found = $this->userOptionDAO->getUserId('matrixUid',
    ↪  $matrixUid);
    $exist = null !== $found;
}
```

Listing 21: Generierung der Matrix-ID

6.3 Automatisches Erstellen von Räumen

Um die Kurse und Kanäle zu synchronisieren, wurden die Rest-Schnittstellen des Matrix-Servers (*Synapse*) verwendet. Um somit bei der Erstellung eines Chat-Kanals einen entsprechenden Matrix-Chat zu erstellen, muss der *createRoom* Endpunkt der *Client-Server API* aufgerufen werden (Listing 22). Bei der Erstellung eines Raumes können über ein *preset* (Voreinstellung) verschiedenste Standardeinstellungen wie z.B. die Beitrittsregeln für den Raum festgelegt werden [vgl. Mat21c, Kap. 8.1]. Da es nicht möglich sein soll, diesen Raum ohne Einladung zu betreten, wurde das *preset private_chat* verwendet.

```php
public function createRoom(string $roomName, mixed
 ↪  $initialState): string
{
    $bodyJson = json_encode(['name' => $roomName, 'preset'
     ↪  => 'private_chat', 'initial_state' =>
     ↪  initialState]);
    $res =
     ↪  $this->client->post($this->getMatrixUrl('/createRoom'),
     ↪  ['body' => $bodyJson, 'headers' =>
     ↪  $this->getSystemRequestHeader()]);

    $result = json_decode($res->getBody(),
     ↪  true)['room_id'];

    return $result;
}
```

Listing 22: Erstellung eines Chatraumes

Um im Frontend den Chatraum eindeutig dem Kurs/Kanal zuordnen zu können, wurden *Custom State Events* (Abschnitt 5.3.1) verwendet. Bei der Erstellung eines Raumes ist es mit dem *initialState* möglich, verschiedenste Events zu definieren, die beim Erstellen direkt gesendet werden sollen (Listing 22). Somit ist es möglich, direkt bei der Raumerstellung ein Event zu senden, welches dem Frontend mitteilt, zu welchem Kurs/Kanal dieser Chatraum gehört (Listing 23).

```json
{
  "content": {
    "courseId": 1,
    "channelId": 1,
  },
  "state_key": "",
  "type": "de.dida.audimax.course",
}
```

Listing 23: Kurs/Kanal Event

Nachdem der Raum erstellt worden ist, müssen die Kurs/Kanal-Teilnehmer noch dem Raum hinzugefügt werden. Damit Audimax die Kurs/Kanal-Teilnehmer mit dem Chatraum synchronisieren kann, wird die *Synapse-Admin API* verwendet, die es dem Backend ermöglicht, andere Benutzer ohne Einladung einem Raum hinzuzufügen (Listing 24).

```php
public function joinRoom(string $uid, string $roomId)
{
    $matrixUid = $this->getMatrixUid($uid);
    $bodyJson = json_encode(['user_id' => $matrixUid]);
    $this->client->post(
      $this->getSynapseUrl('/join/'.$roomId),
      ['headers' => $this->getSystemRequestHeader(),
       ↪  'body' => $bodyJson]
    );
}
```

Listing 24: Raum beitreten

Die weiteren benötigten Aktionen wie z.B. das Löschen von Räumen, das Setzen der Nutzerberechtigungen usw., wurden alle nach dem gleichen Schema umgesetzt und fragen alle verschieden Endpunkte der *Matrix-Client-Server API* oder der *Synapse-Admin API* an.

6.4 Verwendung der SDK

Für die Entwicklung des Frontends wird in dieser Arbeit nur die grundlegende Verwendung der Matrix Javascript SDK gezeigt, da die restliche Entwicklung wie zum Beispiel das Anzeigen der Nachrichten sehr spezifisch für die verwendete Technologien und die Designanforderungen sind.

Um in der Matrix Javascript SDK mit dem Matrix-Server zu kommunizieren, muss zuerst ein Client erstellt werden (Listing 25). Um den Benutzer zu authentifizieren, muss der in Abschnitt 12 erklärte *access_token* (Zugriffstoken) sowie die Nutzer-ID mitgegeben werden. Wenn ein Client erstellt wurde, kann diesem zwar Anfragen an den Matrix-Server senden, allerdings synchronisiert dieser noch nicht alle eingehenden Events (wie z.B. das Ankommen einer neuen Chat-Nachricht). Um die automatische Synchronisation zu aktivieren, wird diese mit dem Aufruf von *startClient* gestartet (Listing 25), hierbei kann auch angegeben werden, wie viele Events initial geladen werden sollen.

```
import * as sdk from "matrix-js-sdk";
const client = sdk.createClient({
  baseUrl: "https://matrix.org",
  accessToken: "token",
  userId: "userId"
});

await client.startClient({initialSyncLimit: 10});
```

Listing 25: Matrix-js-Sdk: Client erstellen [vgl. Matf]

Um eine Nachricht zu versenden, stellt die SDK ebenfalls eine Funktion bereit. Dieser wird die Raum-ID und die zu versendende Nachricht übergeben (Listing 26).

```
const content = {
  "body": "message text",
  "msgtype": "m.text"
};

client.sendEvent("roomId", "m.room.message", content, "",
 ↪  (err, res) => {
  console.log(err);
});
```

Listing 26: Matrix-js-Sdk: Nachrichten senden [Matf]

Um auf verschiedenste Events des Matrix-Servers zu reagieren, bietet der Client die Möglichkeit, *Event-Listener* für diese Events zu definieren. Hierbei kann angegeben werden, bei welchem Event eine bestimmte Funktion ausgeführt werden soll. So ist es zum Beispiel möglich, darauf zu reagieren, wenn es eine neue Nachricht in einem Raum gibt, um diese dann dem Benutzer anzuzeigen (Listing 27).

```
client.on("Room.timeline", function(event, /*...*/) {
  if (event.getType() !== "m.room.message") {
    return; // only use messages
  }
  console.log(event.event.content.body);
});
```

Listing 27: Matrix-js-Sdk: Nachrichten anzeigen [Matf]

Um die in Abschnitt 6.2 gesetzten Kurs/Kanal-IDs zu verwenden, muss der aktuelle Status des Raumes abgefragt werden (Listing 28). Mit der in Listing 28 beschriebenen Methode ist es nicht nur möglich, die Kurs/Kanal-IDs abzufragen, sondern auch alle an-

deren *State Events* des Raumes. Hierfür muss nur der *Event-Type de.dida.audimax.course* durch den gewünschten *Event-Type* ausgetauscht werden.

```
const room = client.getRoom("roomId");

room
.currentState.getStateEvents("de.dida.audimax.course")[0]
.getContent();
```

Listing 28: Matrix-js-Sdk: Kurs/Kanal-ID abfragen

7 Fazit

Chatsysteme bieten die Möglichkeit der *asynchronen* Online-Kommunikation von mehreren Clients. Mittlerweile sind Chatsysteme weit verbreitet und werden in vielen Bereichen des Alltags eingesetzt. Im Bereich der digitalen Lehre gibt es auch einige Anwendungsmöglichkeiten von Chatsystemen wie zum Beispiel der Datenaustausch zwischen den Lehrkräften und den Studenten/Schülern oder die Verwendung im Lehrprozess. Hierbei gibt es einige nützliche Funktionen wie zum Beispiel *Antworten*, *Threads* und *Reaktionen*, welche die Nutzung eines Chatsystems verbessern können.

Bei der Integration eines Chatsystems in ein System zur digitalen Lehre ist die Backend-Integration einfach möglich, da es hier einfach ist, den Server des Chatsystems auszuführen und mit diesem zu kommunizieren. Die Frontend-Integration ist aufwendiger, da es hier nicht einfach möglich ist, das vorhandene Chatsystem zu verwenden. Das Frontend muss mit der Technik und im Style des vorhandenen Systems selbst entwickelt werden. Aus den in Kapitel 2 vorgestellten Chatsystemen ist Matrix am besten für eine Integration in ein System für die digitale Lehre geeignet, da es alle gewünschten Funktionen bietet, die beste Dokumentation besitzt, Ende-zu-Ende-Verschlüsselung unterstützt, die meiste Erweiterbarkeit ermöglicht und aktiv weiterentwickelt wird.

In Matrix erfolgt jegliche Interaktion mithilfe von sogenannten *Events*, die zwischen dem Client und seinem *homeserver* synchronisiert werden. Die Kommunikation der Nutzer erfolgt im Kontext von virtuellen Räumen. Dabei kann jeder Nutzer in mehreren Räumen sein, die wiederum mehrere Nutzer beinhalten können. Hierbei ist es auch möglich, dass die Nutzer verschiedene *homeserver* verwenden, welche die *Events* dann synchronisieren. Um die Nutzer in Matrix zu authentifizieren, werden sogenannte *access_tokens* (Zugriffstoken) verwendet.

Für die Integration des Matrix-Servers wird *Synapse* verwendet. Hierbei werden alle benötigten Daten aus dem System für die digitale Lehre mithilfe der *Client-Server-API* und der *Admin-API* synchronisiert. Damit sich die Benutzer bei *Synapse* authentifizieren können, werden Module (Modules) verwendet, die es ermöglichen, die Authentifikationsmethoden von Synapse zu erweitern, um so die Benutzer aus dem vorhandenen System zu verwenden. Damit der Nutzer sich nicht zweimal anmelden muss, soll es möglich sein, dass der Nutzer den *access_token* (Zugriffstoken) für den *Synapse-Server* bei dem vorhandenen Backend abfragen kann, um somit mit einer Anmeldung Zugang zu beiden

Systemen zu bekommen. Für die Frontend-Integration wird die *Matrix Javascript SDK* verwendet, welche jegliche Kommunikation mit dem Matrix-Server übernimmt.

Grundlegend ist eine Integration von Matrix in ein vorhandenes Chatsystem gut möglich. Dabei ist es auch möglich, das Gefühl eines einzigen geschlossenen Systems beizubehalten, um so dem Nutzer eine möglichst einheitliche Oberfläche zu bieten. Da Matrix sehr viele Funktionen unterstützt, kann das System funktionsmäßig mit anderen Systemen mithalten oder diese teilweise übertreffen. Da aus Zeitgründen eine Implementierung der Ende-zu-Ende-Verschlüsselung nicht möglich war, sollte diese für eine bessere Sicherheit der Nutzerdaten noch umgesetzt werden. Zusätzlich sollte in Zukunft die Verwendung weiterer Matrix Funktionen überlegt werden. Hierbei wäre es zum Beispiel über *Federation* möglich mit anderen Matrix-Servern zu kommunizieren, um so die Kommunikation zwischen verschiedenen Bildungseinrichtungen zu ermöglichen. Um die Migration von einem anderen Chatsystem einfacher zu ermöglichen, sollte auch die Verwendung von *Bridges* in Betracht gezogen werden.

Literaturverzeichnis

[42w] 42wim: matterbridge, URL https://github.com/42wim/matterbridge/, Abrufdatum: 05.01.2022

[Air21] Airnow: *Ranking der beliebtesten Apps im Google Play Store in der Kategorie Kommunikation nach der Anzahl der Downloads in Deutschland im September 2021*, Statista (2021), URL https://de.statista.com/statistik/daten/studie/688712/umfrage/beliebteste-kommunikations-apps-im-google-play-store-nach-downloads-in-deutschland/, Abrufdatum: 15.12.2021

[Bun21] Bundesamt für Sicherheit in der Informationstechnik: Moderne Messenger – heute verschlüsselt, morgen interoperabel? (2021), URL https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/DVS-Berichte/messenger.html, Abrufdatum: 06.12.2021

[Dev] Devture Ltd: Shared Secret Authenticator password provider module for Matrix Synapse, URL https://github.com/devture/matrix-synapse-shared-secret-auth, Abrufdatum: 28.12.2021

[Dis22] Discord: Discord (2022), URL https://discord.com/, Abrufdatum: 07.01.2022

[Doc22] Docker, Inc.: Docker (2022), URL https://www.docker.com/, Abrufdatum: 10.01.2022

[Ehr19] Ehrke, Georg: Reactions to chat messages (2019), URL https://github.com/nextcloud/spreed/issues/1920, Abrufdatum: 20.12.2021

[Gita] Github Insights: Pulse matrix-org/matrix-doc: November 20, 2021 – December 20, 2021, URL https://github.com/matrix-org/matrix-doc/pulse/monthly, Abrufdatum: 20.12.2021

[Gitb] Github Insights: Pulse matrix-org/synapse: November 20, 2021 – December 20, 2021, URL https://github.com/matrix-org/synapse/pulse/monthly, Abrufdatum: 20.12.2021

[Gitc] Github Insights: Pulse nextcloud/spreed: November 20, 2021 – December 20, 2021, URL https://github.com/nextcloud/spreed/pulse/monthly, Abrufdatum: 20.12.2021

[Hes15] Hessisches Kultusministerium: Handreichung für Lehrkräfte zum Umgang mit Sozialen Netzwerken in hessischen Schulen (2015), URL https://datenschutz.hessen.de/sites/datenschutz.hessen.de/files/content-downloads/Handreichung%20Soziale%20Netzwerke%20-%20Stand%20Februar%202015_0.pdf, Abrufdatum: 16.12.2021

[Ibr21] Ibrahim, Md Hasan; Sayagh, Mohammed und Hassan, Ahmed E.: A study of how Docker Compose is used to compose multi-component systems. *Empirical Software Engineering* (2021), Bd. 26(6): S. 1–27

[ILI22] ILIAS open source e-Learning e.V.: ILIAS (2022), URL https://www.ilias.de/, Abrufdatum: 07.01.2022

[Ivi20] Ivic, Vlatka und Sostaric, Blazenka: Benefits and drawbacks on literature circles in Moodle chat for the students of English as a foreign language. *ALAR: Action Learning and Action Research Journal* (2020), Bd. 26(1): S. 133–158

[J0W19] J0WI: Allow server/client-side encryption of chats (2019), URL https://github.com/nextcloud/spreed/issues/1437, Abrufdatum: 20.12.2021

[Kee19] Kees, Alexandra und Markowski, Dominic Raimon: *Open Source Enterprise Software : Grundlagen, Praxistauglichkeit und Marktübersicht quelloffener Unternehmenssoftware*, Springer Fachmedien Wiesbaden, Wiesbaden, 2. Aufl. (2019)

[ma1] ma1uta: Synapse REST Password provider, URL https://github.com/ma1uta/matrix-synapse-rest-password-provider, Abrufdatum: 28.12.2021

[Mata] Matrix.org Foundation: The Admin API - Synapse, URL https://matrix-org.github.io/synapse/develop/usage/administration/admin_api/index.html, Abrufdatum: 23.12.2021

[Matb] Matrix.org Foundation: Bridges, URL https://matrix.org/bridges/, Abrufdatum: 05.01.2022

[Matc] Matrix.org Foundation: Frequently Asked Questions, URL https://matrix.org/faq/#what-is-matrix%3F, Abrufdatum: 16.12.2021

[Matd] Matrix.org Foundation: Instalation: Using PostgreSQL, URL https://matrix-org.github.io/synapse/develop/setup/installation.html#using-postgresql, Abrufdatum: 29.12.2021

[Mate] Matrix.org Foundation: Matrix Javascript SDK, URL https://github.com/matrix-org/matrix-js-sdk, Abrufdatum: 28.12.2021

[Matf] Matrix.org Foundation: Matrix Javascript SDK, URL https://github.com/matrix-org/matrix-js-sdk, Abrufdatum: 05.11.2021

[Matg] Matrix.org Foundation: Synapse Docker, URL https://github.com/matrix-org/synapse/tree/develop/contrib/docker, Abrufdatum: 24.11.2021

[Mat21a] Matrix.org Foundation: Matrix Specification version v1.1 (2021), URL https://spec.matrix.org/v1.1/, Abrufdatum: 21.12.2021

[Mat21b] Matrix.org Foundation: Matrix Specification version v1.1: Appendices (2021), URL https://spec.matrix.org/v1.1/appendices/, Abrufdatum: 21.12.2021

[Mat21c] Matrix.org Foundation: Matrix Specification version v1.1: Client-Server API (2021), URL https://spec.matrix.org/v1.1/client-server-api, Abrufdatum: 21.12.2021

[Mat22] Matrix.org Foundation: Matrix (2022), URL https://matrix.org/, Abrufdatum: 07.01.2022

[Mic22] Microsoft Corporation: Microsoft Teams (2022), URL https://www.microsoft.com/de-de/microsoft-teams, Abrufdatum: 07.01.2022

[Min] Ministerium für Kultus, Jugend und Sport Baden-Württemberg: Der Einsatz von „Sozialen Netzwerken“ an Schulen, URL https://it.kultus-bw.de/,Lde_DE/Startseite/IT-Sicherheit/soziale-netzwerke, Abrufdatum: 16.12.2021

[Moo22a] Moodle: Moodle (2022), URL https://moodle.org, Abrufdatum: 07.01.2022

[Moo22b] Moodle an Hochschulen e.V.: Moodle an Hochschulen (2022), URL https://moodle-an-hochschulen.de/, Abrufdatum: 12.01.2022

[M'R11] M'Raïhi, David; Machani, Salah; Pei, Mingliang und Rydell, Johan: TOTP: Time-Based One-Time Password Algorithm. *RFC* (2011), Bd. 6238

[Nex] Nextcloud gmbh: Creating a new user, URL https://docs.nextcloud.com/server/latest/admin_manual/configuration_user/user_configuration.html?highlight=username#creating-a-new-user, Abrufdatum: 21.11.2021

[Nex22a] Nextcloud GmbH: Nextcloud (2022), URL https://nextcloud.com/, Abrufdatum: 07.01.2022

[Nex22b] Nextcloud GmbH: Nextcloud Talk (2022), URL https://nextcloud.com/talk/, Abrufdatum: 07.01.2022

[Ope22] Open Source Initiative: The Open Source Definition (2022), URL https://opensource.org/osd, Abrufdatum: 10.01.2022

[PF10] Peter Fischer, Peter Hofer: *Lexikon der Informatik*, Springer Heidelberg Dordrecht London New York, 15. Aufl. (2010)

[Rah20] Rahayu, Dwi: Students' E-Learning Experience through a Synchronous Zoom Web Conference System. *Journal of ELT Research* (2020), Bd. 5(1)

[Sig22] Signal: Signal (2022), URL https://signal.org, Abrufdatum: 07.01.2022

[Sla22] Slack Technologies, LLC: Slack (2022), URL https://slack.com, Abrufdatum: 07.01.2022

[Tel22] Telegram: Telegram Messenger (2022), URL https://telegram.org/, Abrufdatum: 07.01.2022

[Thr22] Threema GmbH: Threema (2022), URL https://threema.ch, Abrufdatum: 07.01.2022

[Uni21] Unicode Consortium: *The Unicode®Standard Version 14.0 – Core Specification*, Unicode Consortium (2021)

[Uni22] Unicode Consortium: Overview - Unicode (2022), URL https://home.unicode.org/basic-info/overview/, Abrufdatum: 10.01.2022

[Voh16] Vohra, Deepak: *Pro Docker*, Apress, Berkeley, CA, 1. Aufl. (2016)

[Weg96] Wegner, Peter: Interoperability. *ACM Comput. Surv.* (1996), Bd. 28(1): S. 285–287

[Wha22] WhatsApp LLC: WhatsApp (2022), URL https://www.whatsapp.com, Abrufdatum: 07.01.2022

Listingsverzeichnis

.