Analyse von Message Oriented Middleware mit der Hilfe eines erstellten Prototypen

Semesterarbeit Wirtschaftsinformatik-Seminar 2021

Verfasser:	Sandro Caroppo, Finn Graf, Nico Kruse, Elias Züst
Modul:	WISE
Semester:	HS 2021
Modulverantwortlicher:	Prof. Dr. Christian Thiel
Klasse:	WINF19VZ

OST
Ostschweizer
Fachhochschule

Quellennachweis Titelblatt

Hirschi Informatik AG, [Foto]. (o.D.) Gefunden am 27.09.2021 unter https://www.hinfo.ch/%C3%BCber-uns/

Projektteam:

Sandro Caroppo Projektleiter

Finn Graf

Nico Kruse

Elias Züst

Projekt-Coach:

Hansruedi Tremp

Eingereicht am:

St. Gallen, 23. Dezember 2021

Vorwort

Im Rahmen des Wirtschaftsinformatik-Seminars an der Ostschweizer Fachhochschule St. Gallen erarbeitete die Projektegruppe ein Softwareprototyp, welcher mithilfe von aktueller «Message Oriented Middleware»-Technologie ein in dieser Arbeit definierter Use Case erfolgreich abzudecken vermag. Der Zweck der Semesterarbeit wird dadurch erreicht, indem das gelernte Wissen in den vergangenen Wirtschaftsinformatikmodulen praxisbezogen in einer realen Softwareumgebung angewendet wird. Das Hauptziel dieser Arbeit bestand darin, das Thema der «Message Oriented Middleware» wissenschaftlich zu untersuchen und daraus einen wirkungsvollen und nutzenstiftenden Prototypen zu entwickeln, der als Basis für weitere, skalierbare Softwareprodukte im selben Middleware-Bereich verwendet werden und auch für Unterrichtszwecke dienlich sein kann.

Theoretisch erlernte Inhalte konnte die Projektgruppe dank dieser Semesterarbeit in die Tat umsetzen. Die Middleware-Technologie war der Gruppe lediglich auf Modellierungsebene bekannt und stellte somit eine zusätzliche Herausforderung dar. Dies weckte die Wissenslust der Projektgruppe, wodurch wir viel über die theoretischen Inhalte und praktischen Abläufe der Middleware-Technologie, die in dieser Arbeit thematisiert wurden, lernen konnten. Das grosse Engagement der gesamten Gruppe führte schlussendlich zu einem Lösungsansatz, der vom Projekt-Coach als angemessen betrachtet wurde. Die Teilnehmer der Projektgruppe sind vom Praxisbezug begeistert und stolz auf das Resultat der Arbeit.

Es hat uns gefreut, die vorliegende Semesterarbeit in Zusammenarbeit mit IPM Dozent Hansruedi Tremp verfassen zu dürfen. Ganz besonders bedanken wir uns bei Herrn Tremp, der uns mit ausführlichen Quellen und Literatur während dem Erarbeiten der Arbeit unterstützte und schliesslich die Projektgruppe auf den richtigen Weg zum Abschluss der Semesterarbeit führte. Die offene, stets unkomplizierte und rasche Kommunikation haben wir sehr geschätzt. Wir bedanken uns zudem herzlich für das entgegengebrachte Vertrauen und Engagement. All dies ermöglichte eine angenehme und konstruktive Arbeitsumgebung, die eine erfolgreiche Semesterarbeit zur Folge hat.

St. Gallen, im Dezember 2021 Die Projektgruppe

Management Summary

Ausgangslage

Verteilte Softwaresysteme setzen zur Kommunikation immer mehr auf «Message Oriented Middleware». Als intermediäre Software stellen sie den Nachrichtenaustausch zwischen verschiedenen Systemen sicher. Die vorliegende Arbeit beschäftigt sich mit den Grundlagen von Message Oriented Middleware (MOM). Es werden verschiedene Messaging Models und auch Arten von Messaging Protokollen behandelt und aufgezeigt. Die theoretische Erarbeitung wird um eine umfassende Marktsituationsanalyse ergänzt. Die Recherche umfasst den aktuellen Stand der MOM mit Schwerpunkt auf den Message Brokern. In einem letzten Teil der Arbeit ist ein konkreter Use Case mithilfe des Prototyps umgesetzt worden. Für die Implementierung des Prototyps wurde die Software «RabbitMQ» verwendet. (Kap. 1)

Ziele

Die vorliegende Semesterarbeit ist erfolgreich bearbeitet, wenn folgende Ziele erfüllt werden (Kap. 1.2):

1.	Der Schlussbericht deckt technologische Aspekte von Message Oriented Middleware anhand bestehender Literatur in Breite und Tiefe ab. Der Einsatz von MOM wird ebenfalls aufgezeigt.
2.	Die Marktanalyse verschafft einen Überblick über bestehende MOM-Produkte und zeigt die besten Anbieter an.
3.	Der erstellte Prototyp ist lauffähig und deckt die Bedürfnisse des Anwendungsfalles ab.
4.	Der erstellte Prototyp lässt sich als Vorlage für grössere Anwendungen nutzen.

Methodik

In einem ersten Teil dieser Semesterarbeit wird eine Literaturrecherche zum Thema Message Oriented Middleware inklusive Marktforschung durchgeführt (Kap. 2) und in einem zweiten Teil wird Engineering-Arbeit in der Softwareentwicklung mit einem MOM-Broker geleistet (Kap. 3). Mithilfe der erforschten Literatur wird ein konkreter Anwendungsfall generiert, der mithilfe eines MOM-Brokers in einen funktionsfähigen Prototypen umgesetzt wird.

Ergebnisse

Literaturrecherche und Marktforschung

Die Literaturrecherche und die Marktforschung haben sich als schwierig erwiesen. Dies lag hauptsächlich daran, dass im Bereich der Message Oriented Middleware wenig aktuelle Literaturwerke vorhanden sind. Technologische Aspekte wurden daher mit Informationen abgedeckt, die direkt von MOM-Produkt Anbietern stammen. Bei der Marktanalyse wurden keine Zahlen und Statistiken zu Verkäufen oder Downloads gefunden. Somit bediente sich die Projektgruppe aus bestehenden Reviews und Bewertungen von drei IT-Communities. Schlussendlich lässt sich sagen, dass MOM-Systeme in der IT breitflächig angewendet werden, in der Literatur jedoch zu diesem Thema Nachholbedarf besteht.

Umsetzung des Prototyps

Ableitend von den gesammelten Erkenntnissen gelang es der Projektgruppe ein Prototyp mit integrierter Middleware zu implementieren. Als Anwendungsfall wird ein praxisnaher, aber hypothetisches Szenario aus der Bildung herangezogen. Es soll ein System zur Verfügung stehen, in der Lehrpersonen Hausaufgaben erfassen können, welche wiederum mit integriertem Chat von den Schülern und Schülerinnen eingesehen werden können. Der erstellte Prototyp deckt dabei der soeben beschriebene Anwendungsfall erfolgreich ab.

Der Vorteil von Middleware liegt nicht nur in der zusätzlichen Architekturschicht, sondern auch in der Skalierbarkeit. Somit besteht die Möglichkeit, der erstellte Prototyp auf grössere Anwendungsfälle zu skalieren und anzupassen.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

AMQP	Advanced Message Queuing Protocol
AMS	Advanced Message Security
API	Application Programming Interface
AWS	Amazon Web Services
BMW	Bayerische Motoren Werke AG
bzw.	beziehungsweise
CEP	Complex Event Processing
CSS	Cascading Style Sheets
EDA	Event-Driven-Architecture
EE	Enterprise Edition
FIFO	First In First Out
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTP	Hypertext Transfer Protocol Secure
IBM	International Business Machines Corporation
IMAP	Internet Message Access Protocol
IP	Internet Protocol
IT	Informationstechnologie
IoT	Internet of Things
JMS	Java Message Service
JSR	Java Specification Requests
MOM	Message Oriented Middleware
MQ	Message Queue
MQTT	Message Queuing Telemetry Transport
NASA	National Aeronautics and Space Administration

M2M	Machine-to-Machine
OAuth	Open-Authorization
ORB	Object Request Broker
POP3	Post Office Protocol Version 3
RPC	Remote Procedure Call
REST	Representational State Transfer
SASL	Simple Authentication and Security Layer
SMTP	Simple Mail Transfer Protocol
SNS	Simple Notification Service
SQS	Simple Queue Service
SSL	Secure Sockets Layer
STOMP	Simple (or Streaming) Text Oriented Messaging Protocol
TCP	Transmission Control Protocol
TLS	Transport Layer Security
WWW	World Wide Web

1 Auftragsdefinition

In der vorliegenden Semesterarbeit befasst sich die Arbeitsgruppe mit dem Thema der «Message Oriented Middleware», wobei die folgenden Unterkapitel die Ausgangslage, die Ziele und das Vorgehen erläutern.

1.1 Ausgangslage

Verteilte Softwaresysteme setzen zur Kommunikation immer mehr auf «Message Oriented Middleware», kurz MOM. Diese MOMs agieren als intermediäre Software bzw. als Kommunikations-Broker, um verschiedene Systeme oder Kommunikationsprotokolle zusammen zu verbinden und somit den asynchronen Nachrichtenaustausch zu ermöglichen. Obwohl MOMs in der Praxis unter anderem in vielen grossen Unternehmen wie Google, Amazon oder IBM Verwendung finden, so gibt es in der Literatur auf den ersten Blick relativ wenig aktuelle Informationen zu finden. Die vorliegende Arbeit soll dementsprechend ein Überblick über das Thema MOM zum Zweck haben und den dafür vorgesehenen Einsatz der Softwareprodukte aufzeigen.

1.2 Ziele

1.	Der Schlussbericht deckt technologische Aspekte von Message Oriented Middleware anhand bestehender Literatur in Breite und Tiefe ab. Der Einsatz von MOM wird ebenfalls aufgezeigt.
2.	Die Marktanalyse verschafft einen Überblick über bestehende MOM-Produkte und zeigt die besten Anbieter an.
3.	Der erstellte Prototyp ist lauffähig und deckt die Bedürfnisse des Anwendungsfalles ab.
4.	Der erstellte Prototyp lässt sich als Vorlage für grössere Anwendungen nutzen.

Tabelle 1: Definition der Ziele für die Semesterarbeit
Quelle: eigene Darstellung

1.3 Vorgehen

Die Vorgehensweise in dieser Semesterarbeit soll eine effiziente Zielerreichung ermöglichen. Grundsätzlich besteht diese aus zwei Hauptteilen: Einem Theorie- und einem Praxisteil respektive Kapitel 2 und Kapitel 3. Mithilfe von bestehender Literatur soll die Technologie der Message Oriented Middleware umschrieben und erklärt werden. Dabei werden mit Fokus auf die Technik die Definition, die Funktionen und Messaging Modelle wie auch Protokolle in

Betracht gezogen. Nicht zuletzt werden Sicherheitsaspekte und daraus folgende Vor- und Nachteile dargelegt. Schliesslich wird der Markt an MOMs einer konkreten Analyse unterzogen, indem verschiedene Message Broker anhand eines Rankings erläutert werden.

Im zweiten Teil der Arbeit wird anhand der Theorie aus Kapitel 2 ein konkreter Anwendungsfall generiert, der mithilfe eines MOM-Brokers in einen funktionsfähigen Prototyp umgesetzt wird. Das Kapitel 3 wird dabei mit einer Dokumentation und dem relevantesten Teil des zugehörigen Prototyp-Quellcode vervollständigt. Die unten abgebildete Abbildung 1 stellt den genauen Ablauf dar.

Abbildung 1: Vorgehen WISE-Projekt
Quelle: eigene Darstellung

2 Message Oriented Middleware

Wie bereits in Abbildung 1 ersichtlich, beinhaltet das Kapitel 2 eine Literaturrecherche zur Theorie von Message Oriented Middlewares. Neben einer Marktanalyse zu den meistverwendeten Message Brokern, werden die Eigenschaften von MOM erklärt, verschiedene Messaging Models analysiert und auch verschiedene Arten von Messaging Protocols erläutert.

2.1 Definition und Funktionen

Eine Middleware, wie unten in Abbildung 2 dargestellt, dient dazu, dass mehrere Applikationen auf verschiedensten Plattformen miteinander kommunizieren können. Durch die Middleware können Anwendungen auf verschiedensten Netzwerkelementen verteilt über eine API miteinander kommunizieren, ohne die Betriebsumgebung der verschiedensten Anwendungen zu kennen. Dadurch wird ein neues virtuelles System von den verbundenen Applikationen erstellt, das zuverlässig und sicher ist. Mit diesem System wird auch die Skalierbarkeit verbessert. (Oracle, 2010a)

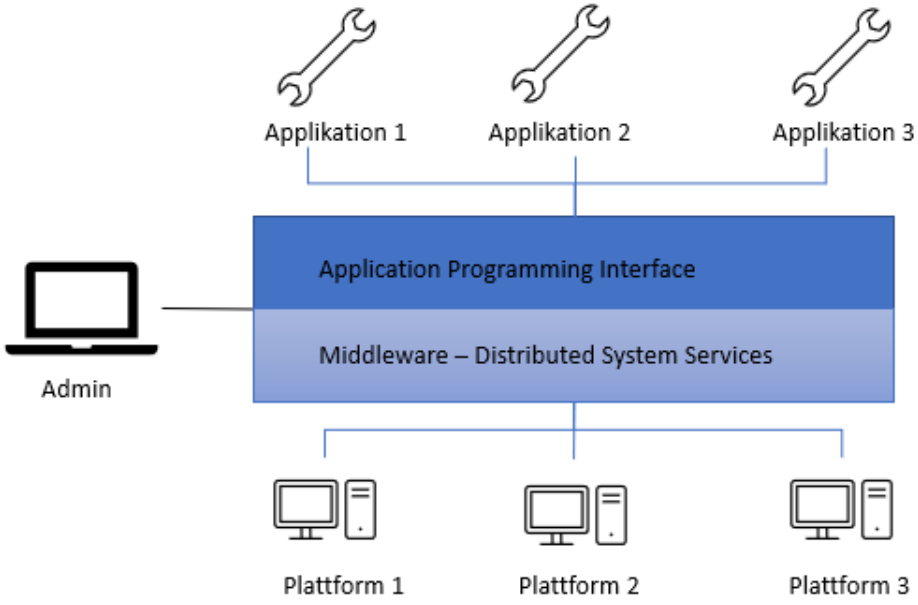


Abbildung 2: Visuelle Darstellung einer Middleware
Quelle: eigene Darstellung in Anlehnung an https://docs.oracle.com/cd/E19340-01/820-6424/aeraq/index.html

Die Middleware lassen sich in folgende Kategorien einteilen:

- Remote Procedure Call (RPC)-based middleware
- Object Request Broker (ORB)-based middleware
- Message oriented Middleware (MOM)-based middleware

Die RPC-based Middleware dient dazu, eine entfernte Applikation so aufzurufen, als wäre diese lokal verfügbar. Die Middleware implementiert einen Verknüpfungsmechanismus, der entfernte

Prozeduren ausfindig und diese für den Aufrufer transparent verfügbar macht. Grundsätzlich wurde diese Art von Middleware für Prozedurale Programmierung verwendet, was auch die objektbasierten Komponenten umfasst. Die ORB-based Middleware ermöglicht die Verteilung und gemeinsame Nutzung von Objekten einer Anwendung in heterogenen Netzen. Schliesslich bietet MOM-based Middleware in verteilten Anwendungen Funktionen für die Kommunikation und für den Austausch durch das Empfangen und Senden von Nachrichten an. (Oracle, 2010a)

All diese verschiedene Middleware erlauben das Verhalten von anderen Komponenten zu beeinflussen. Jedoch bestehen technische Unterschiede. Die RCP- und ORB-basierten Middleware sind eng gekoppelt zu den Komponenten, während die MOM-basierte Middleware eine losere Kopplung der Komponenten ermöglicht. Im folgenden Abschnitt wird nun explizit der Fokus auf die Message Oriented Middleware gelegt. (Oracle, 2010a)

Ein MOM-basiertes System, wie unten in Abbildung 3 dargestellt, ermöglicht die Kommunikationen durch den asynchronen Austausch von Nachrichten. MOM benutzt den Messaging Provider als Vermittler der Messaging Operationen. Grundsätzlich besteht ein MOM-basiertes System aus Clients, Nachrichten und dem Messaging Provider, wobei auch die API integriert sind. Der MOM-Anbieter verwendet verschiedene Architekturen für die Weiterleitung und Zustellung von Nachrichten. Er kann einen zentralisierten Nachrichtenserver verwenden oder die Weiterleitungs- und Zustellungsfunktionen auf jeden Client Rechner verteilen. (Oracle, 2010a)

Abbildung 3: MOM basiertes System
Quelle: eigene Darstellung in Anlehnung an https://docs.oracle.com/cd/E19340-01/820-6424/aeraq/index.html

Bei einem MOM-System tätigt ein Client einen API-Aufruf, um eine Nachricht an den Ort zu senden der vom Messaging Provider verwaltet wird. Durch den Aufruf wird der Service der Messaging Provider aktiviert. Dabei wird die Nachricht weitergeleitet und zugestellt. Nachdem die Nachricht gesendet wurde, kann der Client sich anderen Aufgaben widmen, da die Nachricht so lange aufbewahrt wird, bis ein empfangender Client die Nachricht aufruft bzw. konsumiert. Somit ermöglicht ein nachrichtenorientierte Architektur die lose Kopplung von Komponenten. (Oracle, 2010a)

Ein weiterer Vorteil eines Messaging Provider, der den Nachrichtenaustausch zwischen Clients vermittelt, besteht darin, dass durch das Hinzufügen eines administrativen Interface die Leistung überwacht und optimiert werden kann. Die Client-Anwendungen sind somit von allen Aufgaben außer dem Senden, Empfangen und Verarbeiten von Nachrichten entlastet. Es ist die Aufgabe des Administrators, der das MOM-System implementiert, Probleme wie Interoperabilität, Zuverlässigkeit, Sicherheit, Skalierbarkeit und Leistung zu lösen. (Oracle, 2010a)

Bisher wurden die Vorteile der Verbindung verteilter Komponenten über eine nachrichtenorientierte Middleware beschrieben. Es gibt aber auch Nachteile: Einer davon ergibt sich aus der losen Kopplung selbst. Bei einem synchronen Nachrichtensystem kehrt die aufrufende Funktion erst zurück, wenn die aufgerufene Funktion ihre Aufgabe beendet hat. In einem asynchronen System kann der aufrufende Client den Empfänger so lange Arbeit aufbürden, bis die erforderlichen Ressourcen für diese Arbeit erschöpft sind und die aufgerufene Komponente ausfallen. Natürlich können diese Zustände minimiert oder vermieden werden, indem die Leistung überwacht und der Nachrichtenfluss angepasst wird. Dies ist jedoch eine Aufgabe, die bei einem synchronen Nachrichtensystem nicht nötig ist. Es ist wichtig, die Vor- und Nachteile der einzelnen Systeme zu kennen. Jedes System ist für unterschiedliche Aufgaben geeignet. Manchmal müssen die beiden Systeme kombiniert werden, um das gewünschte Verhalten zu erreichen. (Oracle, 2010a)

2.2 Event-Driven-Architecture

Die gestiegenen Anforderungen in der Informationstechnologie in den letzten Jahren setzen betriebliche IT-Systeme voraus, die flexibel und agil eingesetzt werden können. Ableitend von der realen Welt nehmen Ereignisse eine entscheidende Rolle ein, weshalb neue Unternehmensanwendungen in Form von Event-Driven-Architecture (EDA) und Complex Event Processing (CEP) eingeführt wurden. EDA stellt ein ereignisorientierter Entwurfsstil dar, mit dem eine Organisation diverse Ereignisse oder Geschäftsmomente erkennen kann. CEP hingegen

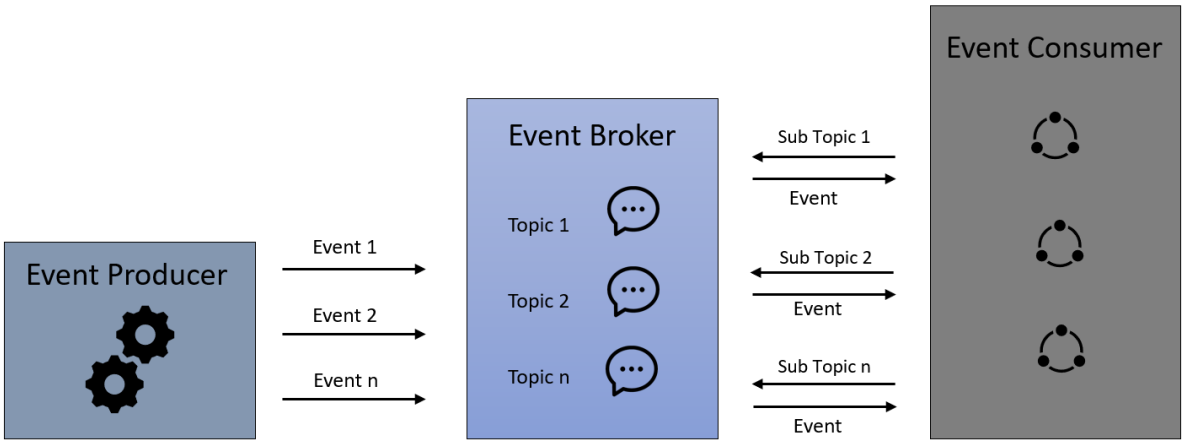


Abbildung 4: Aufbau einer EDA
Quelle: eigene Darstellung in Anlehnung an https://www.tibco.com/de/reference-center/what-is-event-driven-architecture

wird für die dynamische Verarbeitung von grossen Mengen an Ereignissen genutzt, damit nahezu in Echtzeit eine Reaktion des Systems erfolgen kann. Die Architektur läuft somit nicht rein zyklisch oder sequenziell ab, sondern wird durch diverse Ereignisse beeinflusst und dabei dynamisch verändert. Abbildung 4 demonstriert das Grundprinzip der EDA und umfasst drei Hauptkomponenten. (Bruns & Dunkel, 2010)

Die Event Producer sind dafür zuständig, dass diverse Ereignisbenachrichtigungen generiert und gesendet werden. Neben dem Produzenten gibt es den Konsumenten, in Abbildung 4 als Event Consumer dargestellt. Die Konsumenten empfangen die Ereignisse und können diese unmittelbar ausführen. Die Kommunikation erfolgt dabei asynchron, was dazu führt, dass sowohl Konsument als auch Produzent nicht aufeinander warten müssen, sondern zeitlich versetzt kommunizieren können. Der Event Consumer kann bei Bedarf auch wieder Event Producer werden, indem dieser eine Reaktion auf ein Ereignis, als eine neue Aktion, ausführt. In diesem Falle befasst man sich mit der reaktiven Programmierung. Dies ist ein Programmierparadigma, dass sich an Datenströmen (Data Streams) orientiert und es ermöglicht, diverse Änderungen in der Ausführungsumgebung in einer bestimmten Reihenfolge zu propagieren. Ein sehr bekanntes System mit reaktiver Programmierung ist das Microsoft Office Produkt Excel. Wird der Wert in einer Zelle geändert, dann passt sich der Wert innerhalb der Summenzelle automatisch an. Die Zelle, deren Wert geändert wurde, löst automatischen ein Event aus, den die Summenzelle empfängt und demzufolge eine Neukalkulation vornimmt. Der Event Broker bildet den dritten Komponenten in der EDA und kann optional sein. (Tibco, 2021)

Microsoft sieht die Verwendung von EDA bei Anwendungen mit mehreren Subsystemen, welche die gleichen Ereignisse verarbeiten müssen, oder bei Echtzeitverarbeitungen mit minimaler Verzögerung vor. Zudem kommen EDAs vermehrt gerade in IoT-Systemen zum Einsatz, denn EDA ermöglicht eine Entkopplung regelbasierter Dienste mit Mikrodiensten. Die Abwicklung von grossen Datenmengen wird dabei unterstützt. (Microsoft, 2021)

Neben EDA wird in der Literatur auch oftmals von Message Driven Architecture gesprochen. Tendenziell werden diese Begriffe als Synonyme wahrgenommen, da sie nur minimale Differenzen aufweisen. In nachrichtengesteuerten Systemen erzeugt Komponente A eine Nachricht, die an die Adresse von Komponente B geliefert werden muss. Die Nachricht von Komponente A wird gesendet. Die Kontrolle wird direkt zurückgegeben, anstatt zu warten, bis Komponente B die Bearbeitung der Nachricht abgeschlossen hat. In ereignisgesteuerten Systemen geben die Komponenten einen Ort bekannt, an dem sie ihre Ereignisse ausgeben. Somit sendet eine Komponente A diverse Ereignisse an einen bestimmten Ort, der diese veröffentlicht. Die Komponente A weiss dabei aber nicht, welche Komponenten die Ereignisse konsumieren. Manchmal können die Verbraucher die bereits konsumierten Ereignisse und die

noch ausstehenden Ereignisse verfolgen. Diese Unterscheidung kann jedoch schnell irritierend wirken. Vereinfacht ausgedrückt bezieht sich "Message Driven" auf einen Baustein eines Systems und "Event Driven" auf eine übergeordnete Eigenschaft eines Systems. Mit Message Driven Tools werden also ereignisgesteuerte Architekturen gebaut. (Lightbend, 2021)

2.3 Message Queuing und Messaging Models

Um die Funktionen und Anwendungsmöglichkeiten von MOM noch besser verstehen zu können, werden im Kapitel 2.3.2 und 2.3.3 die zwei gängigsten Nachrichtenmodelle genauer analysiert und deren Eigenschaften erklärt. Diese bilden den Schlüssel zum genaueren Verständnis über MOM. Beide Nachrichtenmodelle beruhen auf dem Austausch von Nachrichten über einen Kanal, der mehrheitlich Nachrichtenwarteschlange genannt wird.

Diese Nachrichtenwarteschlange, in der Literatur auch oftmals Queue genannt, ermöglicht es also den Anwendungen, miteinander zu kommunizieren. Ebenfalls werden vorübergehend die Nachrichten in der Queue gespeichert, wenn das Zielprogramm beschäftigt oder nicht verbunden ist. (Reich, ohne Datum)

Um die Nachrichtenmodelle einfacher zu verstehen, wird zuerst im Kapitel 2.3.1 das Grundprinzip von Message Queuing erklärt.

2.3.1 Message Queuing

Für die erfolgreiche Verwendung einer Messaging Queue braucht es zwei Hauptkomponenten: Eine Queue und eine Nachricht. Die Queue ist eine Reihe von Instanzen oder Arbeitsobjekten, die genau darauf warten, bearbeitet zu werden, wobei dies vom Anfang in einer vordefinierten Reihenfolge abgearbeitet wird. Unter einer Nachricht wird der Datentransport zwischen der sendenden und der empfangenden Anwendung verstanden. Ein Beispiel für eine Nachricht könnte beispielsweise eine Meldung sein, dass ein System anweist, mit der Bearbeitung einer bestimmten Aufgabe zu beginnen. Beide Komponenten zusammen bilden die Grundlage für eine asynchronen Kommunikation von Dienst zu Dienst, wie in Abbildung 5 verdeutlicht. (IBM, 2018)

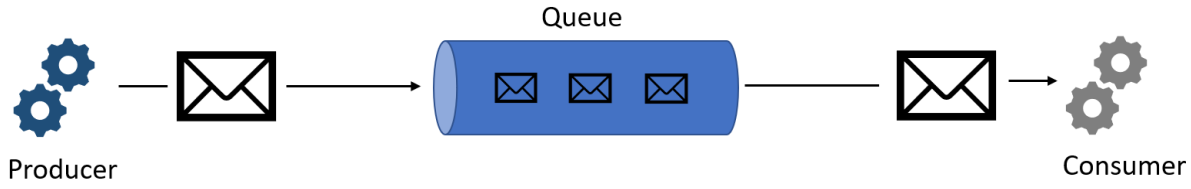


Abbildung 5: Message Queue
Quelle: eigene Darstellung in Anlehnung an https://medium.com/tech-sauce/introduction-to-message-queuing-4a7ab8968b59

Das Messaging Queuing ermöglicht somit eine zeitunabhängige Kommunikation, erhöht die Zuverlässigkeit und bietet die Möglichkeit einer besseren Skalierung. AWS nennt Messaging Queues als «eine elegante, einfache Möglichkeit, verteilte Systeme aufzugliedern – unabhängig davon, ob Sie monolithische Anwendungen, Microservices oder serverlose Architekturen einsetzen“ (Amazon, ohne Datum a).

2.3.2 Point-To-Point

Das Point-To-Point Nachrichtenmodell ermöglicht einen unkomplizierten asynchronen Austausch von Nachrichten zwischen verschiedenen Softwareeinheiten. In Abbildung 6 wird das Modell präziser dargestellt. Die grundlegende Architektur von Point-To-Point ist ziemlich einfach aufgebaut. Zum einem gibt es Client-Anwendungen, so genannte Producer (Client 1), die Nachrichten erstellen und an die Nachrichtenwarteschlange liefern. Eine andere Anwendung, ein so genannter Consumer (Client 2), stellt eine Verbindung zur Warteschlange her und holt die zu verarbeitenden Nachrichten ab. Das Point-To-Point Nachrichtenmodell stellt sicher, dass eine Nachricht nur von einer Verbraucheranwendung empfangen wird. (Oracle, 2010b)

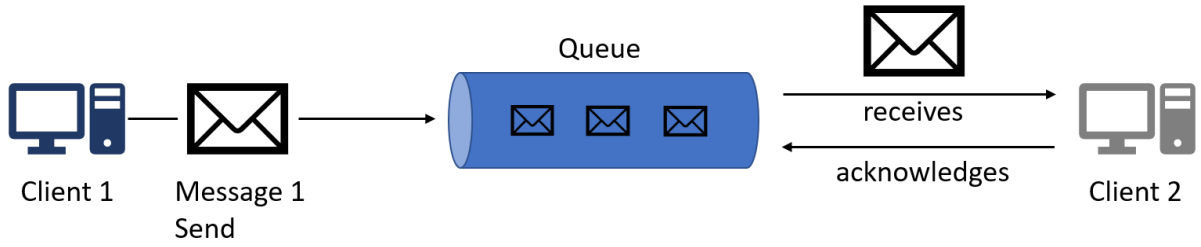


Abbildung 6: Point-To-Point Modell
Quelle: eigene Darstellung in Anlehnung an Reich, ohne Datum

2.3.3 Publish & Subscribe

Das Publish & Subscribe Nachrichtenmodell, auch Pub/Sub genannt, ist ein Mechanismus zur Verbreitung von Informationen zwischen Konsumenten und Produzenten. Dieser Austausch wird auch «One-To-Many» und «Many-To-Many» genannt. Pub/Sub ermöglicht es einzelnen Produzenten, eine Nachricht an einem Benutzer oder potenziell an beliebig viele Konsumenten zu senden. Die sendende, oder auch empfangende Anwendung muss dabei nichts über die Zielanwendung wissen. Clients können Nachrichten produzieren und diese in einem bestimmten Thema oder Kanal veröffentlichen. Der Kanal kann dann von Clients abonniert werden, welche die Nachrichten konsumieren möchten. Der Pub/Sub Dienst leitet die Nachrichten an die Verbraucher auf Grundlage der Themen weiter, die folglich abonniert

wurden. Ein Kunde kann sowohl Produzent als auch Konsument eines Themas / Kanals sein. Abbildung 7 zeigt den Aufbau eines Pub/Sub Messaging Models. (Google Cloud, 2021)

Pub/Sub-Systeme bieten den Abonnenten in der Regel eine Möglichkeit an, nach relevanten Nachrichten direkt zu filtern. Für die Filterung gibt es zwei gängige Formen: Themenbasierte Filterung oder inhaltsbasierte Filterung, die im Englischen auch «Topic-based» oder «Content-based» genannt werden. (ETH, ohne Datum) In einem Topic-based System werden die Nachrichten in «Themen» oder logischen Kanälen veröffentlicht. Der Herausgeber trägt nun die Verantwortlichkeit, die Themen zu definieren, die von den Abonnenten abonniert werden können. Die Abonnenten erhalten wiederum alle Nachrichten, die in einem Thema veröffentlicht werden. Der Ansatz von Content-based Systemen liegt bei der Filterung des Inhalts. Nachrichten werden an Abonnenten nur zugestellt, wenn die Attribute oder der Inhalt dieser Nachrichten den vordefinierten Einschränkungen entsprechen. Die Abonnenten selbst sind für die Klassifizierung der Nachrichten verantwortlich. Neben der Filterung gibt es auch die «Single» bzw. «Multi-Delivery» Funktionalität. In der Single-Delivery können Message Oriented Middlewares garantieren, dass eine Nachricht nur ein einziges Mal zugestellt wird. Zudem werden die Nachrichten exakt in der Reihenfolge empfangen, in der sie versendet wurden. In der Multi-Delivery Methode können mehrere Nachrichten generiert und versendet werden. (Google Cloud, 2021)

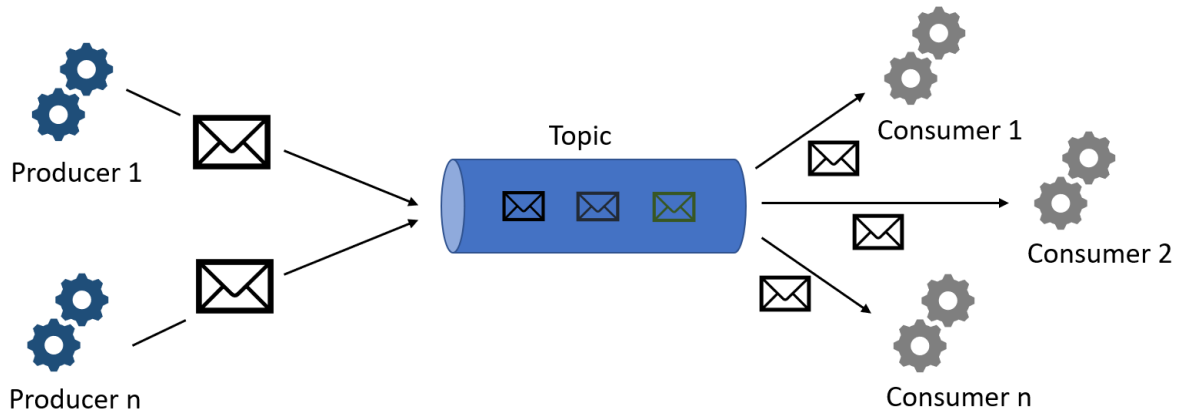


Abbildung 7: Pub/Sub
Quelle: eigene Darstellung in Anlehnung an https://dashbird.io/knowledge-base/well-architected/pub-sub-messaging/

2.4 Arten von Messaging Protocols

Das Kapitel 2.4 befasst sich mit den verschiedenen Arten von Messaging Protocols. Dabei wird lediglich ein kleiner Ausschnitt von möglichen Protokollen analysiert und deren Nutzen beschrieben. Gerade auch in den heutigen agilen Entwicklungen gibt es diverse Alternativen, die verwendet werden können.

2.4.1 AMQP

Advanced Message Queuing Protocol, abgekürzt AMQP, ist ein binäres Protokoll, das diverse Verfahren zum Senden und Empfangen von Nachrichten über ein Netzwerk definiert. AMQP wurde 2003 von John O'Hara in der Londoner Niederlassung von JPMorgan Chase entwickelt und ermöglicht eine verschlüsselte und kompatible Nachrichtenübertragung zwischen Unternehmen und Anwendungen. Mit der Verwendung von AMQP können beliebige AMQP-kompatible Client-Bibliotheken und beliebige AMQP-kompatiblen Broker verwendet werden. Zudem unterstützt es mehrheitlich alle Nachrichtenmodelle, einschliesslich Pub/Sub und Point-To-Point. (Sandoval, 2019)

AMQP ist zudem ein Protokoll der Anwendungsschicht, das Client-Anwendungen erlaubt, mit dem Server kommunizieren und interagieren zu können. Es kann aber auch eine High-Level-Architecture für Nachrichten-Broker darstellen. In der Praxis stellt AMQP eine Reihe von Methoden oder Operationen zur Verfügung, um AMQP aktiv zu verwenden. (Joahnsson, 2019)

2.4.2 MQTT

MQTT steht für «Message Queuing Telemetry Transport» und wurde erstmals von Andy Stanford-Clark und Arlen Nipper entwickelt. Es wird als eher einfaches und leichtes Messaging-Protokoll verstanden, das für eingeschränkte Geräte und Netzwerke mit geringer Bandbreite, hoher Latenz, oder unzuverlässigen Netzwerken entwickelt wurde. Demzufolge sieht sich MQTT selbst als Standard für IoT-Messaging, was die Kommunikation zwischen Maschinen, auch unter dem Begriff Machine-To-Maschine (M2M) bekannt, darstellt. (mqtt.org, 2020)

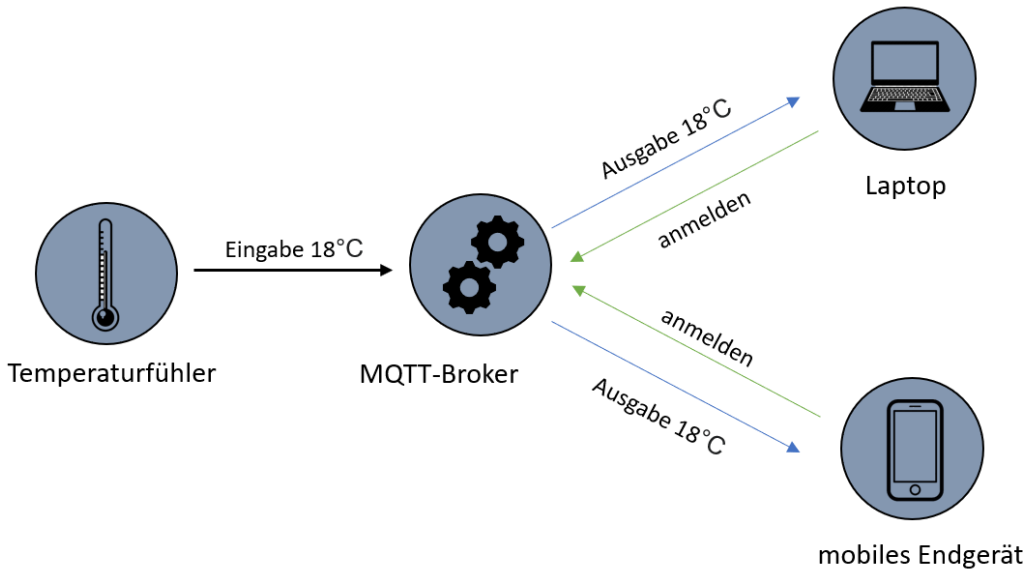


Abbildung 8: Beispiel für die Verwendung von MQTT
Quelle: eigene Darstellung in Anlehnung an mqtt.org

MQTT ermöglich zudem ein Nachrichtenaustausch zwischen Geräten und Clouds wird. Folglich wird die Übermittlung von Nachrichten an Gruppen von Geräten stark vereinfacht. MQTT setzt dabei auf das Pub/Sub Nachrichtenmodell. In Abbildung 8 wird ersichtlich, dass der MQTT Broker im Mittelpunkt jedes Pub/Sub Protokolls steht. Der Broker ist verantwortlich für das Senden, Empfangen und Filtern von Nachrichten. Daraus folgt, dass ein Broker die Authentifizierung und Autorisierung von Clients vornehmen muss, wobei die Sicherheit durch die Verschlüsselung mit «OAuth» gewährleistet ist. (mqtt.org, 2020)

MQTT setzt auf sogenannte MQTT-Topics, welche das Filtern von Nachrichten für den Broker vereinfachen. Jedes Topic besteht aus einer oder mehreren Themenebenen. Ein Client muss das gewünschte Thema nicht erstellen, bevor es veröffentlicht oder abonniert wird, denn der Broker akzeptiert jedes gültige Topic ohne vorherige Initialisierung. Im Vergleich zu gewöhnlichen Nachrichtenwarteschlangen sind MQTT-Topics bedeutend einfacher und flexibler. (mqtt.org, 2020)

2.4.3 Jakarta Messaging

Jakarta Messaging, früher Java Message Service (JMS) genannt, ist die erste Message-API für Unternehmen, die breite Unterstützung in der Industrie gefunden hat. Jakarta Messaging ist dabei Teil der Jakarta Enterprise Edition (Jakarta EE) und wurde von damaligen Unternehmen Sun Microsystems und mehreren anderen Unternehmen im Rahmen des Java Community Process als JSR 914 entwickelt. Die neuste Version 3.0 Jakarta Messaging kam von der Eclipse Foundation im Jahr 2019/2020 auf den Markt. Der Messaging-Standard wird somit neu vom Eclipse Jakarta Project geführt und ermöglicht es Anwendungskomponenten auf Basis der Java Platform EE Nachrichten zu erstellen, zu empfangen, zu senden und zu lesen. Die kommende Version 3.1 ist noch unter Development. Eine verteilte Kommunikation, die lose gekoppelt, zuverlässig und asynchron ist, wird ermöglicht. JMS definiert zudem ein gemeinsames Messaging-API für Unternehmen, das so konzipiert ist, dass eine breite Palette von Messaging-Produkten für Unternehmen unterstützt werden. JMS unterstützt dabei die beiden Messaging-Modelle Publish-Subscribe und Point-To-Point. (Jakarta EE, ohne Datum)

JMS gehört insbesondere zu den Message-Oriented Middleware-Systeme (MOM) und definiert eine Reihe von Semantiken und Schnittstellen, die es diversen Java-Anwendungen ermöglichen mit anderen Messaging-Implementierungen zu kommunizieren. Die Menge an Konzepte, die ein Java-Entwickler erlernen muss, werden durch JMS stark minimiert. Trotzdem wird die Portabilität von Messaging-Anwendungen maximiert. (Jakarta EE, ohne Datum)

2.4.4 STOMP

Simple (or Streaming) Text Oriented Messaging Protocol, abgekürzt STOMP, bietet ein interoperables Datenformat an. Dies bedeutet, dass STOMP-Clients mit jedem STOMP-Message-Broker kommunizieren können. Im Vergleich zu AMQP ist es ein weniger kompliziertes Protokoll und ähnelt eher am HTTP-Protokoll, der grundsätzlich im World Wide Web (WWW) eingesetzt wird. STOMP wird unter anderem von den Message Brokern «Apache ActiveMQ», «HornetQ» und «RabbitMQ» unterstützt. STOMP zählt als einfaches und leicht implementierendes Protokoll und eignet sich für einfache Message-Queuing-Anwendungen ohne komplexe Anforderungen. Um sich bei einem STOMP-Broker anzumelden, kann zum Beispiel «Telnet» verwendet werden. Da es sich bei Telnet um ein Internetprotokoll handelt, das heutzutage eher weniger verbreitet ist, so findet folglich STOMP auch seltener Verwendung. Dennoch eignet sich STOMP als ein einfaches textbasiertes Protokoll für nachrichtenorientierte Middleware. (ActiveMQ, 2021; CloudAMQP, 2021a)

2.4.5 Abgrenzung von SMPT, POP und IMAP

Nachdem einige verschiedene Messaging Protocols genauer analysiert wurden, geht es in diesem Kapitel um die Abgrenzung zu bekannten Protokollen wie SMTP (Simple Mail Transfer Protocol), IMAP (Internet Message Access Protocol) und POP3 (Post Office Protocol Version 3).

POP3 (Post Office Protocol): Der POP-Zugang erfolgt via Internet Protocol. Die Client-Anwendung greift auf die Mailbox zu, welche auf einem Mailserver stationiert ist. Das Protokoll unterstützt die Funktionen "Download" und "Löschen". Mittels POP3 kann sich ein Client verbinden, die Nachrichten herunterladen und dabei diese auf dem Server direkt löschen, wobei die Nachrichten dann lokal auf dem Client-Computer gespeichert werden. POP ist grundsätzlich dazu gedacht, nur temporär eine Verbindung mit dem Server aufzubauen. Sobald die Nachricht lokal gespeichert ist, wird die Verbindung nicht mehr gebraucht. Es besteht jedoch auch die Option, die Nachrichten auf dem Server gespeichert zu lassen. Dafür wurde jedoch IMAP entwickelt. (Dean, 2009, S. 519)

IMAP (Internet Message Access Protocol): IMAP ist das Standardprotokoll für E-Mail Services, welche über TCP/IP Schnittstellen kommunizieren. Die Mailbox kann mittels IMAP von mehreren Clients verwaltet werden, das heisst, dass von mehreren Geräten parallel, ob lokal oder mobil (also überall, wo eine Internetverbindung besteht) darauf zugegriffen werden kann. Die Nachrichten bleiben auf dem Server gespeichert, bis der Client diese direkt auf dem Server löscht. Folgende Unterschiede zu POP3 lassen sich ableiten:

- Die Verbindung bleibt bestehen, solange das Interface aktiviert ist.

- IMAP ermöglicht mehrere, simultane Verbindungen mit der Mailbox
- IMAP ermöglicht Statusvermerke für Nachrichten (gelesen, markiert, etc.)
- IMAP ermöglicht eine individuelle Ordnerstruktur
- Die Filterung ist mittels IMAP direkt auf dem Server möglich

Diese Vorteile machen die E-Mail-Verwaltung technisch gesehen komplexer, weshalb auf Back-End-Seite eine Persistenz aufgebaut werden muss. Serverleistungen müssen dabei durch den möglichen, simultanen Serverzugriff ebenfalls erhöht werden. (Mullet, 2000, S. 25)

SMTP (Simple Mail Transfer Protocol): SMTP ist ebenfalls ein Protokoll, das hauptsächlich für das Versenden von E-Mails eingesetzt wird, während für das Empfangen heutzutage typischerweise IMAP zum Einsatz kommt. SMTP unterstützt, abgesehen vom Senden der Nachrichten, lediglich eine Funktion zur Initialisierung einer Warteschlange der Nachrichten, die dann mittels IMAP oder POP3 heruntergeladen werden können. (Rhoton, J., 1999, Programmer's Guide to Internet Mail: SMTP, POP, IMAP and LDAP)

Während sich POP, IMAP und SMTP auf den Versand und Empfang von E-Mail-Nachrichten spezialisieren, dient AMQP dazu, jegliche Formate von Nachrichten zwischen Anwendungen auszutauschen. E-Mail-Services kommunizieren bilateral, das heisst es findet ein Nachrichtenaustausch zwischen einem Client (oder mehreren, siehe IMAP) und einem Server statt. Der Nachrichtenaustausch von Anwendungen mittels AMQP kann über mehrere Geräte und Server stattfinden, welche ebenfalls AMQP unterstützen. AMQP ermöglicht die Verwaltung von Warteschlangen, Publish/Subscribe-Funktionen und das Messaging über Cloud-Systeme. Ausserdem funktioniert das AMQP-Protokoll über diverse Programmiersprachen hinweg. (amqp.org, 2021)

2.5 Design-/ Entwicklungswerkzeuge

Verschiedene Design- und Entwicklungswerkzeuge unterstützen die Verwendung und Umsetzung von MOM in einem System. In diesem Kapitel werden «Postman» und «AsyncAPI» genauer beschrieben und untersucht, ob eine asynchrone Kommunikation unterstützt wird. Neben den beschriebenen Design- und Entwicklungswerkzeuge gibt es natürlich auch eine grosse Anzahl an Alternativen, die in dieser Arbeit nicht berücksichtigt werden.

2.5.1 Postman

Postman (siehe Abbildung 9) ist eines der beliebtesten Werkzeuge zum Testen von APIs (Application Programming Interfaces). Der Fokus liegt dabei auf das Testen von REST APIs auf HTTP-Basis (testautomatisierung.org, ohne Datum). Zudem dient der Postman nicht nur in der Nutzung und Ansteuerung von APIs, sondern auch zum Aufbau von APIs. Das Tool vereinfacht dabei auch jeder Schritt im API-Lebenszyklus. Zusätzlich bietet das Programm ein umfangreiches Verzeichnis von APIs, mithilfe es möglich ist, Spezifikationen, Dokumentationen und Test-Cases zu managen und zu speichern. Sicherheitstechnisch verfügt Postman über ein System, das Sicherheitsschwachstellen erkennen kann. Unterstützung bietet das Programm auch bei Arbeitsumgebungen für das Testen und Bauen von APIs und befähigt zudem zur Integration von APIs. (Postman, 2021a)

Abbildung 9: Logo von Postman
Quelle: https://www.postman.com/home

Die integrierte Applikation „Postman Sandbox“, eine JavaScript Ausführungsumgebung, ermöglicht eine asynchrone Kommunikation. Durch JavaScript lassen sich asynchrone Skripte im Hintergrund ausführen, wenn diese eine hohe Rechenleistung benötigen. Anstatt auf den Abschluss eines Aufrufs zu warten und die nächsten Anfragen zu blockieren, kann eine Callback-Funktion bestimmt werden. Diese sendet eine Benachrichtigung, wenn der zugrunde liegende Vorgang abgeschlossen ist. (Postman, 2021b)

2.5.2 AsyncAPI

AsyncAPI (siehe Abbildung 10) ist eine Open-Source Anwendung, welche die Verbesserung der ereignisgesteuerten Architekturen (EDA) anstrebt. Es gilt als Schwesterprojekt der OpenAPI-Initiative, die sich auf synchrone REST-Kommunikation konzentriert und von der Linux Foundation betreut wird. AsyncAPI wird einerseits für die Entwicklung von asynchronen APIs verwendet und bildet andererseits eine eigene Spezifikationssprache. Spezifikationssprachen werden entwickelt, um Konzepte wie Channels, Messages oder unterschiedliche Protokolle unterstützen zu können. Die AsyncAPI-Spezifikation bildet also die Grundlage für ein größeres und besseres Tooling-Ökosystem für EDAs. (AsyncAPI, ohne Datum a)

Abbildung 10: Logo von AsyncAPI
Quelle: https://www.asyncapi.com/

AsyncAPI kommuniziert auf der eigenen Webseite, dass die meisten Prozesse, die heute auf REST-APIs angewendet werden, auch auf ereignisgesteuerten oder asynchronen APIs anwendbar sind. (AsyncAPI, ohne Datum b)

Die AsyncAPI-Spezifikation setzt keine bestimmte Software-Topologien Architektur oder bestimmte Muster voraus. Folglich kann ein Message Broker, ein Server, oder jegliche andere Art von Computerprogrammen Daten senden und empfangen. Zudem unterstützt die Spezifikation eine Vereinheitlichung der Dokumentationsautomatisierung und der Codegenerierung sowie das Verwalten, Testen und Überwachen asynchroner APIs. Das Grundkonzept ist somit die Unterstützung des gesamten Entwicklungszyklus ereignisgesteuerter Architekturen. (AsyncAPI, ohne Datum c)

2.6 Security-Aspekte

In der heutigen Zeit, gerade auch aufgrund von der Datenschutzthematik, muss für das Senden und Empfangen von sensiblen Nachrichten genügend Sicherheit vorhanden sein. Dieses Kapitel beinhaltet die verschiedenen Security-Aspekte zu Message Oriented Middlewares.

2.6.1 Transport-Level Security

Message Oriented Middleware Systeme verwenden das AMQP-Protokoll für den Nachrichtenaustausch. Um sicherzustellen, dass die Nachrichten nicht abgehöhrt werden, unterstützen viele Message Broker, unter anderem RabbitMQ und IBM MQ, die Transport-Level Security (RabbitMQ, 2021a). Diese Transport-Level Security basiert auf die Transport Layer Security (TLS) bzw. Secure Sockets Layer (SSL), welche wiederum über das HTTP-Protokoll laufen. Um HTTP-Verbindungen nun zu sichern, kann die Transport-Level Security angewendet werden. Denn durch TLS bzw. SSL werden Sicherheitsanforderungen wie Authentication, Datensicherheit und Kryptografische Tokens unterstützt. Das AMQP-Protokoll macht nun von diesem Transport-Level Security ebenfalls Gebrauch, um die Nachrichten eines Queues bzw. einer Warteschlange eindeutig mit einer Key Repository zu Verschlüsseln. Damit wird sichergestellt, dass einzig die dafür vorgesehene Sender und Empfänger der Nachricht, eben diese Nachricht entschlüsseln und somit lesen können. (IBM, 2021a)

2.6.2 Simple Authentication and Security Layer (SASL)

Um sich im Netzwerk für verschiedene Dienstleistungen zu identifizieren, steht das Framework für «Simple Authentication and Security Layer» zur Verfügung. AMQP nutzte ebenfalls die SASL-Methoden, um eine Verbindung zu verifizieren und somit sicherzustellen, dass kein unbefugter Zugriff auf Message Queues erfolgt. Oft wird in Kombination mit SASL die Transport-Level Security verwendet, um Beispielsweise bei der PLAIN-Methode (unverschlüsselter Austausch von Daten; hier Name und Passwort) sicherzustellen, dass der nötige Sicherheitsmechanismus mittels TLS bereitgestellt wird. (IBM, 2021b; Neurodump, 2011)

2.6.3 Advanced Message Security

Verschiedene Message Broker verwenden meist auch individuelle oder ergänzende Sicherheitsmechanismen. Im Fall von IBM MQ beispielsweise, kommt beim Austausch von sensiblen Daten, namentlich finanzielle Transaktionsdaten oder personenbezogene Daten, die Advanced Message Security (AMS) zum Einsatz. Diese Technologie erlaubt die Datensignierung und Datenverschlüsselung auf der Nachrichtenebene. Dies stellt sicher, dass Daten auch in der Warteschlange geschützt sind und dabei nicht verändert werden können. Zusätzlich werden nicht autorisierte Nachrichten automatisch entfernt, bevor diese überhaupt verarbeitet werden. (IBM, 2021c)

2.7 Marktanalyse Message-Broker-Produkte

Das Kapitel 2.7 bildet mit einer Marktanalyse der Schluss des Theorieabschnitts über Message Oriented Middelware. Für die Analyse liegen keine wissenschaftlich untersuchten Statistiken (mit Kriterien wie Anzahl Downloads, Anzahl Lizenzen, etc.) vor, auf die sich die Projektgruppe stützen könnte. Die folgenden Erläuterungen werden also auf Reviews und Bewertungen seitens der MOM-Community basieren, bestehend sowohl aus Experten wie auch aus Open-Source-Mitgliedern. Die daraus entstehende Rangliste ist das Resultat einer Kombination aus verschiedenen Meinungen der Community. Anschliessend werden fünf Message-Broker vorgestellt, die gemäss Rangliste am besten abgeschnitten haben.

2.7.1 Top 10 Rangliste

Um eine geeignete Rangliste der besten MOM-Broker zu generieren, stützt sich die Projektgruppe auf einer Kombination der Kriterien «**Bekanntheit des MOM-Produkts**» und «**Subjektive Bewertung des MOM-Produkts**». Die Daten dazu stammen aus den Communities der folgenden Online-Plattformen:

- www.itcentralstation.com
- www.trustradius.com
- www.g2.com

Auf diesen Plattformen ist es möglich, verschiedene MOM-Produkte (wie auch viele weitere Software-Produkte) in einem persönlichen Review zu bewerten. Ein Beispiel zeigt hier Abbildung 11:

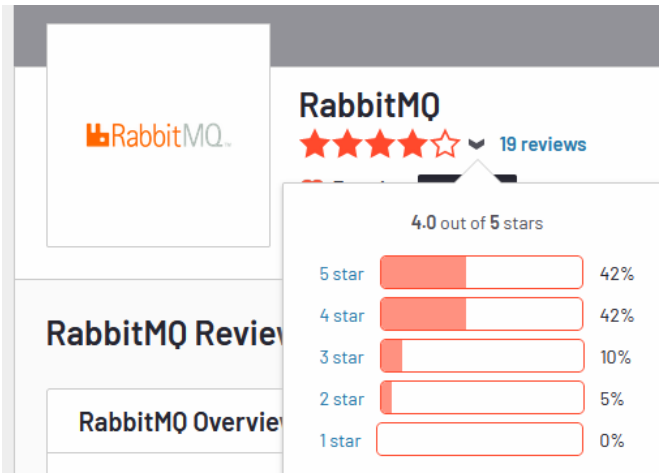


Abbildung 11: Bildschirmausschnitt eines Reviews auf www.g2.com
Quelle: https://www.g2.com/products/rabbitmq/reviews

Die Anzahl der Reviews kann somit als «Bekanntheit» übersetzt werden und wird mit der durchschnittlich erhaltenen, subjektiven Bewertung ergänzt, um einen Vergleich mit anderen Produkten innerhalb und ausserhalb der Online-Plattform, gemäss den soeben beschriebenen Kriterien, zu ermöglichen. Es werden dabei nur Produkte in Betracht gezogen, die auf allen genutzten Plattformen kombiniert eine Anzahl von mindestens fünf Reviews aufzeigen. Die restlichen Produkte werden am Ende dieses Unterkapitels erwähnt, jedoch nicht in die Rangliste aufgenommen.

Solche Bewertungen, wie sie auf vielen Online-Plattformen zu finden sind, beispielsweise auf Urlaubs- oder Detailhandelsinternetseiten, weisen ein Problem vor. Die durchschnittliche Bewertung basiert einzig auf die erhaltenen Reviews des Produktes. Diese durchschnittliche Bewertung lässt sich jedoch nicht nach Belieben mit anderen Produkten vergleichen, da nicht alle Produkte die gleiche Anzahl an Reviews aufweisen. Folgendes Beispiel, um das Konzept zu verdeutlichen:

Produkt A wurde insgesamt 50 Mal bewertet und weist eine durchschnittliche Bewertung von 4.5 von 5 auf. Produkt B wurde 1 Mal bewertet, weist jedoch eine durchschnittliche Bewertung von 5 auf. Welches Produkt ist nun dem anderen vorzuziehen?

Aus dem Bereich der Bayesschen Statistik lässt sich die Kennzahl «Bayesian Average» ableiten. Damit lässt sich anhand einer Schätzung eine genauere, approximierte Bewertung berechnen, welche die Anzahl und die Bewertung gemeinsam berücksichtigt. Die Formel dazu lautet: $(C\ x\ m) * \frac{(Anzahl\ Reviews * durchschnittliche\ Bewertung)}{C\ x\ Anzahl\ Reviews}$, *wobei*

C = Konfidenznummer *(Schätzung: benötigte Reviews, um konkrete Aussagekraft zu besitzen)*

m = Durchschnitt der Bewertung aller Produkte *(aus dem gleichen Bereich auf Online-Plattform)*

Wenn nun für C = 5 und für m = 3 gilt, so erhalten wird für obige Aufgabenstellung folgende Werte bzw. Bewertungen:

Produkt A = $(5\ x\ 3) * \frac{(50*4.5)}{5\ x\ 50} = \mathbf{4.36}$ / Produkt B = $(5\ x\ 3) * \frac{(1*5)}{5\ x\ 1} = \mathbf{3.3}$

Es wäre also Produkt A dem Produkt B vorzuziehen, da die erhaltene Werte den 50 Reviews eine stärkere Gewichtung verleihen. (Masurel, 2013; Algolia, 2021)

Dieses Verfahren, um das «Bayesian Average» zu erhalten, wurde für die Erstellung der Rangliste in diesem Unterkapitel angewendet (siehe Anhang X). Zuerst wurde jede durchschnittliche Bewertung pro Plattform mit diesem Verfahren adjustiert. Anschliessend wurde das Verfahren nochmals auf die durchschnittlichen Bewertungen aller drei Plattformen kombiniert angewendet, wobei die totale Anzahl ebenfalls aus allen drei Plattformen berücksichtigt wurden. Tabelle 2 zeigt die daraus resultierende Rangliste auf. Die Werte wurden auf drei Nachkommastellen genau gerundet, wobei eine Skala von 1 bis 10 verwendet wurde.

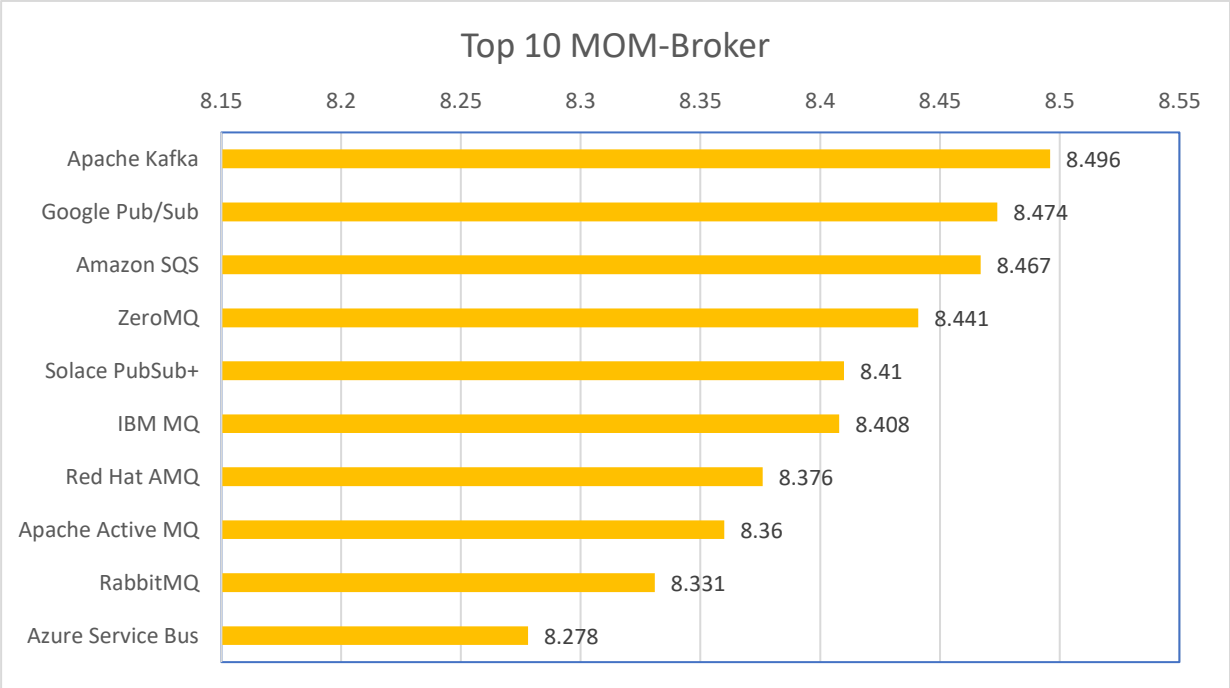


Tabelle 2: Top 10 Rangliste der MOM-Broker auf dem Markt
Quelle: eigene Darstellung

Die angezeigten Werte sind jedoch trügerisch, denn bei mehreren Produkten zeichneten sich bei einer oder zwei Plattformen „Missing Values“ (im Anhang X rot mit einer «0» markiert) heraus. Das heisst, dass auf der Plattform keine Reviews zur Verfügung standen. Dies trifft für folgende Produkte zu:

- Google Pub/Sub
- Solace PubSub+

- Apache ActiveMQ
- Red Hat AMQ
- TIBCO EMS
- Azure Service Bus (Reviews lediglich auf www.g2.com auffindbar, aber Beschreibung der Software auf www.trustradius.com vorhanden)
- Zero MQ (Reviews lediglich auf www.g2.com)

Die vorhandenen Bewertungen aus Tabelle 2 werden nach Ermessen der Projektgruppe mit einer Gewichtung adjustiert, um die Bekanntheit der MOM-Produkte besser darzustellen. Folgende Gewichtungen sind zum Einsatz gekommen:

- 1 = Auf allen drei Plattformen Reviews vorhanden
- 0.98 = Auf zwei Plattformen Reviews vorhanden
- 0.97 = Auf zwei Plattformen erwähnt, jedoch nur auf einer Reviews vorhanden
- 0.95 = Auf einer Plattform Reviews vorhanden

Die Rangliste, die daraus entsteht, lässt sich in Tabelle 3 auslesen.

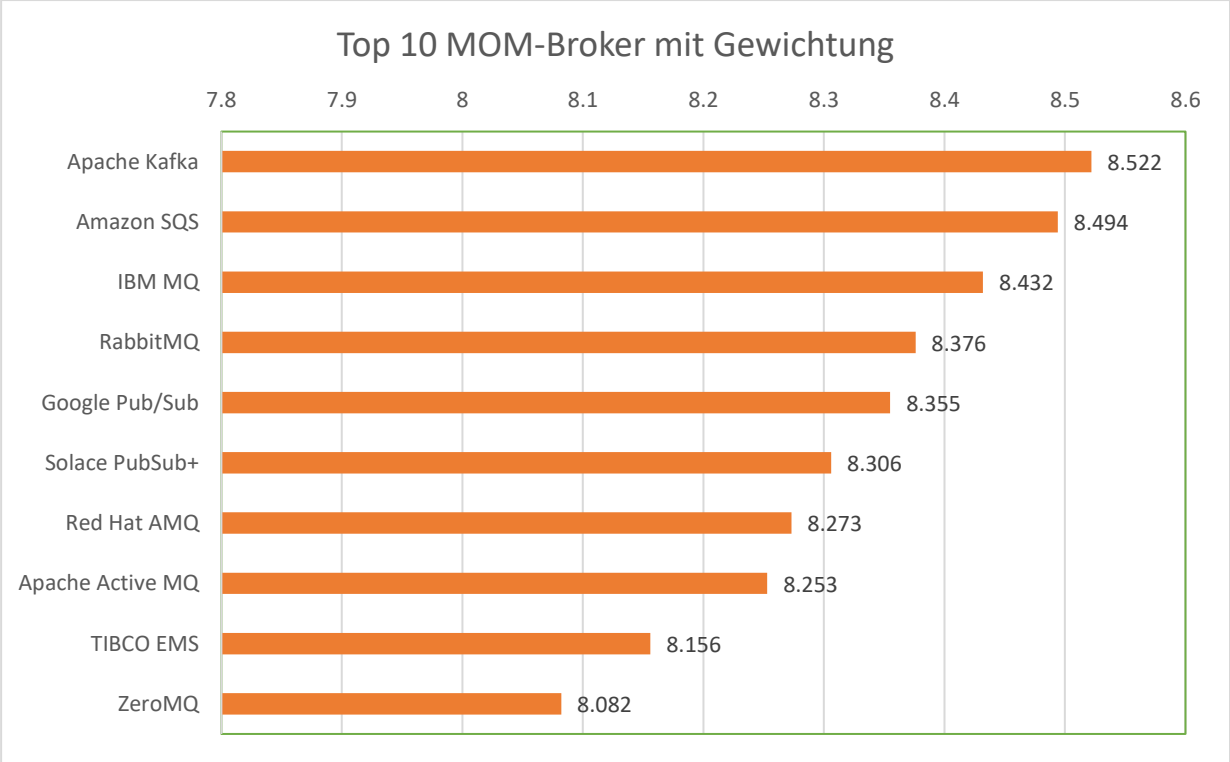


Tabelle 3: Top 10 Rangliste der MOM-Broker auf dem Markt mit Berücksichtigung der Gewichtung
Quelle: eigene Darstellung

In Tabelle 3 lässt sich die finale Rangliste der auf dem Markt vorhandenen MOM-Broker beobachten. Es wird ersichtlich, dass im Vergleich zu Tabelle 2 diejenigen Produkte, welche nicht auf allen Plattformen Reviews aufweisen, einige Positionen verloren haben. Mithilfe der

Gewichtung lässt sich jedoch, mit Berücksichtigung der wenigen, vorhandenen Daten, unter Umständen eine realitätsnähre Marktanalyse darstellen.

Es gibt noch viele weitere Produkte auf dem Markt zu finden, ob Open-Source oder mit einem Pricing, die für verschiedene Zwecke geeignet sind. Folgende MOM-Produkte gilt es für ein vollständiges Bild des Marktes noch zu erwähnen:

- Ably
- IronMQ
- RoboMQ
- Infrared360
- KubeMQ
- Alibaba Message Queue
- Apache RocketMQ
- HiveMQ

Die fünf besten MOM-Produkte gemäss Tabelle 3 werden in den Kapiteln 2.7.2 bis 2.7.6 nun näher erläutert. Obwohl es die Microsoft Azure Lösung nicht in die Top 5 geschafft hat, wird der Service ebenfalls in Kapitel 2.7.7 beschrieben, da Microsoft in der IT-Branche ein wichtiger Akteur ist.

2.7.2 Apache Kafka

Apache Kafka (siehe Abbildung 12) wurde 2008 von der Unternehmung LinkedIn entwickelt und im Jahr 2011 als hoch skalierbares Messaging-System an Apache übergeben. LinkedIn hat Apache in Java und Scala geschrieben. Scala ist eine objektorientierte Softwareprogrammiersprache, die objektorientierte Methoden mit funktionalen Programmierfähigkeiten kombiniert. Stand heute macht Apache als fehlertolerante Event-Streaming-Messaging-Plattform auf sich aufmerksam. (Apache Kafka, 2017)

Die Plattform ermöglicht es Konsumenten, verschiedene Themen zu abonnieren und Datenströme von Produzenten zu verarbeiten. Folglich können diese gespeichert und veröffentlicht werden. Ein Kafka-Thema kann dabei als Kategorie/Feed-Namen betrachtet werden. (Apache Kafka, 2017)

Der wesentliche Vorteil von Apache Kafka ist die Verwendung eines Streaming-Prozesses. Der Prozess ermöglicht die Verarbeitung von Daten in einem parallel verbundenen System. Somit können Anwendungen einen Datensatz ausführen, sobald dieser eintrifft, statt auf die Ausgabe des vorherigen Datensatzes zu warten. Demzufolge

Abbildung 12: Apache Kafka Logo
Quelle: https://www.indellient.com/blog/intro-to-apache-kafka-a-stream-processing-software-platform/

können innert Sekunden Millionen von Datensätze verarbeitet werden, was Apache Kafka als Message Broker sehr beliebt macht. (Apache Kafka, 2017)

2.7.3 Amazon SQS

Der internationale Grosskonzern Amazon bietet im Rahmen der eigenen Cloud-Computing-Lösung «Amazon Web Services», oder kurz AWS (siehe Abbildung 13), mehrere Produkte an, die für verschiedenste Aufgaben geeignet sind. In der Anwendungsintegration finden sich spezifisch folgende drei Produkte, die im Nachrichtenaustausch involviert sind:

- Amazon Simple Queue Service (Amazon SQS)
- Amazon Simple Notification Service (Amazon SNS)
- Amazon Message Queue (Amazon MQ)

Die Bewertung in Kapitel 2.7.1 bezieht sich auf Amazon SQS, weshalb der Fokus in diesem Text darauf gelegt ist. Jedoch sind die beiden anderen Produkte ebenfalls erwähnenswert, da sie zum gleichen Themenbereich bzw. Funktionsbereich gehören. Während Amazon SNS ein einfacher und vollständig verwaltetes Pub/Sub Messaging Service (für SMS, E-Mail und Push-Benachrichtigungen) ist, kann mit Amazon MQ die Verwaltung von anderen, Open-Source Message-Broker wie Apache Active MQ oder Rabbit MQ (siehe Kap. 2.7.5) vereinfacht werden. (Amazon, ohne Datum b; Amazon, ohne Datum c)

Amazon SQS ist ein vollständig verwalteter Message Queue-Service. Dies bedeutet, dass der Verwaltungsaufwand grösstenteils wegfällt, da AWS bereits als Infrastruktur dient. SQS ermöglicht eine Lose Kopplung, sowie das automatische und dynamische Skalieren von Microservices und Anwendungen. Ausserdem ist der Schutz von Daten durch die Serverseitige Verschlüsselung von einzelnen Nachrichtentexten gewährleistet. Amazon SWS bietet Standardwarteschlangen und FIFO-Wartschlangen an, die somit für unterschiedliche Anwendungsanforderungen Verwendung finden. Beide Warteschlangentypen ermöglichen einen hohen Durchsatz von Nachrichten, wobei beim FIFO-Prinzip die Reihenfolge der Nachrichten exakt eingehalten wird und bei der Standardwarteschlange ein nahezu unbegrenzter Durchsatz ermöglich wird. Weiter ist SQS für den Einsatz mit diversen AWS-Infrastruktur-Web-Services geeignet und kann somit erweitert werden. Das Pricing des Produkts erstreckt sich lediglich auf die individuell benötigten Module und enthält keine Mindestgebühren. Unternehmen wie EMS Driving Fuel IQ, die BMW-Gruppe und auch die NASA setzen auf Amazon Messaging Produkte. (Amazon, ohne Datum d)

Abbildung 13: Amazon Web Services Logo Quelle: https://commons.wikimedia.org/wiki/File:Amazon_Web_Services_Logo.svg

2.7.4 IBM MQ (MQSeries)

Auch eines der weltweit grössten Unternehmen für branchenspezifische Lösungen und Dienstleistungen im IT-Bereich, sowie Software und Hardware, hat 1992 auf dem Message Broker Markt Fuss gefasst. Die MQSeries von IBM (siehe Abbildung 14) ist eine plattformunabhängige Message Oriented Middleware und basiert auf dem Prinzip des Message Queuing. Früher agierte die Middleware noch unter dem Namen WebSphere MQ, heute wird sie als IBM MQ vertrieben. (IBM, ohne Datum)

Abbildung 14: IBM MQ Logo
Quelle: https://www.cleo.com/solutions/integration-connectors/message-queue-protocols

Wie auch Apache Kafka unterstützt IBM MQ fast alle gängigen Plattformen und bietet verschiedene Konfigurationsmöglichkeiten wie Pub7/Sub oder Remote / Distributed Queuing an. Für die Queues ist der Queue Manager zuständig. IBM bietet ferner die Möglichkeit einer Konfiguration an, die Clusters erstellt. Ein Cluster besteht dabei aus verschiedenen Queue Managern, auf denen die Queues so konfiguriert werden, dass ein Load Balancing stattfinden kann. Unter «Load Balancing» versteht sich die systematische oder effiziente Verteilung von Anfragen im Hintergrund auf verschiedenen Servern, ohne dass Nutzer dies bemerken. (IBM, 2020)

IBM MQ wird als sehr zuverlässiger und sicherer Message Broker wahrgenommen. Gerade seit der Version 6 sind die Security Konzepte kontinuierlich überarbeitet und verbessert worden. Ein Object Authority Manager sorgt für die Verteilung von Berechtigung auf den Queues. (IBM, ohne Datum)

2.7.5 RabbitMQ

RabbitMQ (siehe Abbildung 15) ist einen Open-Source Message Broker Software, die AMQP und, durch ein Plug-In-Architecture, auch STOMP unterstützt. Im Jahr 2010 wurde die Software von «SpringSource», eine Abteilung von VMware, erworben. Für die Entwicklung wurde die Programmiersprache «Erlang» verwendet. Die Veröffentlichung des Quellcodes läuft derweil über Mozilla Public License.

Abbildung 15: RabbitMQ Logo
Quelle: https://icon-icons.com/icon/rabbitmq-logo/170812

RabbitMQ ist in der Community aufgrund seiner schlanken Konstruktion und der Unterstützung von AMQP, MQTT, HTTPS, STOMP und WebSockets sehr beliebt. RabbitMQ arbeitet mit der Cloud-Nutzungs-Plattform CloudAMQP, die viele RabbitMQ Cluster auf der ganzen Welt verwaltet. CloudAMQP automatisiert die gesamte Einrichtung, den Betrieb und die Skalierung. Ein Control Panel bietet

verschiedene Tools zur Überwachung und Alarmierung. Es können benutzerdefinierte Alarme über E-Mail, Webhooks und andere externe Dienste eingerichtet werden. Leistungsprobleme werden somit automatisch erkennt, bevor sie das Unternehmen beeinträchtigen. Für Entwicklung und Tests bei kleiner Verwendung ist der Dienst auch kostenlos. CloudAMQP bietet einen 24/7 Support an. Für die Verwendung werden Cloud Plattformen wie Amazon AWS, Google Cloud Platform, Microsoft Azure oder Digital Ocean unterstützt. (RabbitMQ, 2021; CloudAMQP, 2021b)

2.7.6 Google Cloud Pub/Sub

Google Pub/Sub (siehe Abbildung 16) ist ein asynchroner Messaging-Dienst, der sich durch hohe Zuverlässigkeit und Skalierbarkeit auszeichnet. Der Messaging-Dienst ermöglicht diese Kommunikation mit einer Latenz von rund 100 Millisekunden. Pub/Sub wird verwendet, um Daten für Streaming und Datenintegrationspipelines aufzunehmen und zu verteilen. Zusätzlich dient es auch als nachrichtenorientierte Middleware für die Dienstintegration und als Warteschlange zur Parallelisierung von Aufgaben. Mit Pub/Sub ist es möglich, Systeme von Eventproduzenten (Publisher) und Eventkonsumenten (Subscriber) mit asynchroner Kommunikation zu erstellen. Die Publisher senden Events an den Pub/Sub-Dienst, unabhängig davon, wie diese Events weiterverarbeitet werden. Folglich werden diese Events durch Pub/Sub an alle Dienste weitergeleitet, meist Subscriber, welche auf die Events reagieren müssen. (Google Cloud, 2021)

Abbildung 16: Google Cloud Pub/Sub Logo
Quelle: https://blog.iron.io/google-cloud-pub-sub-vs-ironmq/

2.7.7 Microsoft Azure Service Bus

Ähnlich wie in Kapitel 2.7.3 bietet Microsoft im Sinne des hauseigenen Cloud-Systems «Microsoft Azure» diverse Produkte im Messaging-Bereich an. Die im Nachrichtenaustusch involvierten Services sind die folgenden:

- Azure Service Bus
- Azure Queue Storage
- Azure Event Grid
- Azure Event Hubs

Abbildung 17: Microsoft Azure Logo
Quelle: https://de.wikipedia.org/wiki/Datei:Microsoft_Azure_Logo.svg

Hierbei ist der «Azure Service Bus» der eigentliche Akteur, wenn es um Message Oriented Middleware geht. Dieser erlaubt das geordnete Messaging über verschiedene, entkoppelte Anwendungen mittels AMQP-Protokolls. Der Service kann dabei von den oben erwähnten Produkten ergänzt und unterstützt werden. Der «Azure Queue Storage» erlaubt das Skalieren

von Anwendungen entsprechend dem vorhandenen Datenverkehr, wobei der «Azure Event Grid» die Events, also die Ereigniszustellung steuert. Zusätzlich können im Bereich Internet of Things (IoT) mit dem «Azure Event Hubs» Telemetriedaten empfangen und verarbeitet werden. (Microsoft Azure, 2021a; Microsoft Azure, 2021b; Microsoft Azure, 2021c; Microsoft Azure, 2021d)

3 MOM-Prototyp

Im folgenden Kapitel werden ein Anwendungsfall für eine MOM-Anwendung, das Vorgehen der Projektgruppe bei der Entwicklung des Prototyps und die daraus entstehenden Ergebnisse dargestellt. Zusätzlich dient Kapitel 3.4 dazu, die wichtigsten Code-Abschnitte zu erläutern und verständlich zu erklären. Der komplette Quellcode ist hingegen im Anhang A - W ersichtlich. Schliesslich bietet Kapitel 3.5 eine Anleitung für die Installation des Prototyps auf einer Windows-Umgebung, damit der Prototyp von weiteren Benutzern verwendet und auch erweitert werden kann.

3.1 Anwendungsfall

Die Corona Pandemie hat viele Unternehmen, Institutionen und Vereine finanziell geschwächt. Neben den finanziellen Aspekten wurden auch administrative Prozesse erschwert. Deshalb wünschen sich Lehrpersonen vermehrt einen Service zur Kommunikation mit den Primarschülern, bei dem die Lehrenden die Hausaufgaben der eigenen Klasse zur Verfügung stellen können, wobei der asynchrone Austausch zwischen den Schülerinnen und Lehrpersonen gewährleistet wird. Dabei soll die Lehrperson Hausaufgaben in verschiedensten Fächer einrichten können. Ausserdem ist ein Chat enthalten, um die Kommunikation zwischen den Schülerinnen und Schülern sowie der Lehrperson zum jeweiligen Thema beziehungsweise zur jeweiligen Hausaufgabe zu ermöglichen. In Abbildung 18 sind die Skizzen des Services ersichtlich, die den Vorstellungen der Lehrpersonen entsprechen.

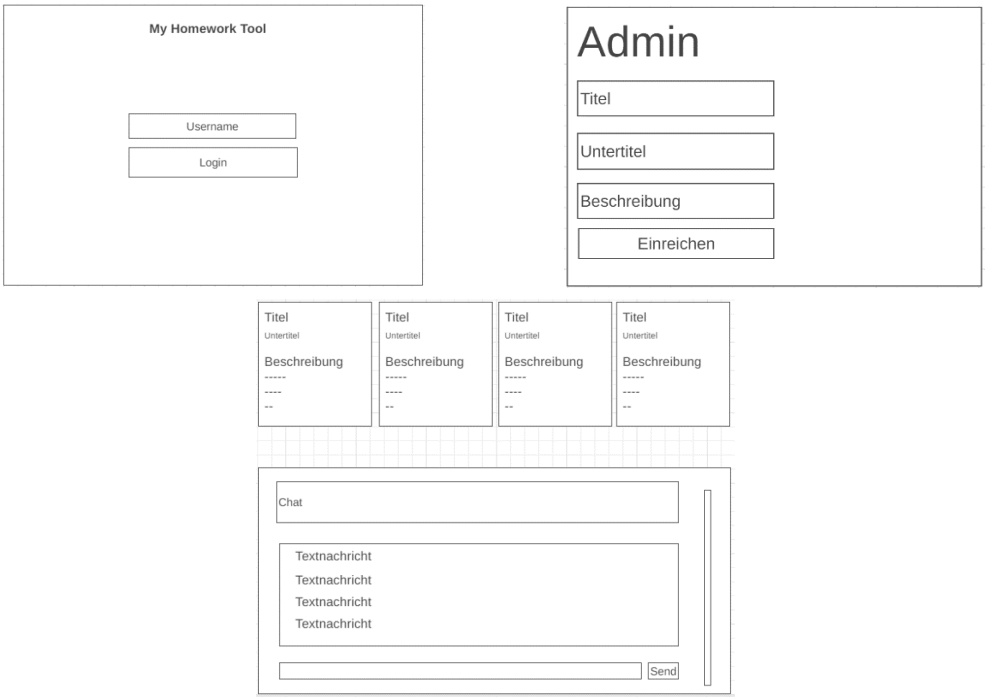


Abbildung 18: Skizze der Lehrpersonen für die Umsetzung der Anwendung
Quelle: eigene Darstellung mittels Onlinetool www.wireframe.cc

3.2 Vorgehen

Als erstes wurden verschiedene MOM Anbieter begutachtet und deren Produkte analysiert. Dabei wurde der Entschluss gefällt, die Lösung von RabbitMQ zu verwenden. Es handelt sich dabei um einen Open-Source Message Broker. Aufgrund der grossen Community können leicht Libraries und Lösungen gefunden werden. Auch Plattformen wie «Reddit», «Robinhood» und «T-Mobile» nutzen RabbitMQ. Dies zeigt, dass die Lösung in der Praxis oft Verwendung findet.

Für das Front-End setzt die Projektgruppe Angular ein. Angular ist ein TypeScript-basiertes Front-End-Webapplikationsframework, welches in der Entwicklungscommunity beliebt ist. Das Entwicklerteam für den Prototypen erhofft sich Synergien aus anderen Modulen nutzen zu können und ein besseres Verständnis des Entwicklungsframeworks zu erhalten. Das Front-End wird durch Bootstrap ergänzt, um ein simples, aber schönes User Interface zu erhalten. Bootsrap ist ein freies Front-End-CSS-Framework und enthält auf HTML und CSS basierende Gestaltungsvorlagen. Formulare, Buttons, Tabellen, Navigationsgestaltungselemente und viele weitere Elemente können verwendet werden. Zusätzlich werden weitere JavaScript Libraries für die Applikationslogik herbeigezogen. Im Back-End wird ein Node.JS Server verwendet. Da Angular auf JavaScript basiert, wird auch der Server mit dieser Sprache konfiguriert. Zur persistenten Speicherung verwendet die Projektgruppe MySQL, da sie hier erste Berührungspunkte in vergangenen Modulen erhielt. Abbildung 19 stellt die verwendete Tools bildlich dar.

Abbildung 19: verwendete Werkzeuge für die Entwicklung
Quelle: eigene Darstellung der Logos von verwendeten Entwicklungswerkzeuge

3.3 Modellierung des Prototyps

In diesem Unterkapitel werden drei verschiedene Diagramme erläutert, welche für ein besseres Verständnis des Prototyps modelliert wurden.

3.3.1 Use Case Diagramm

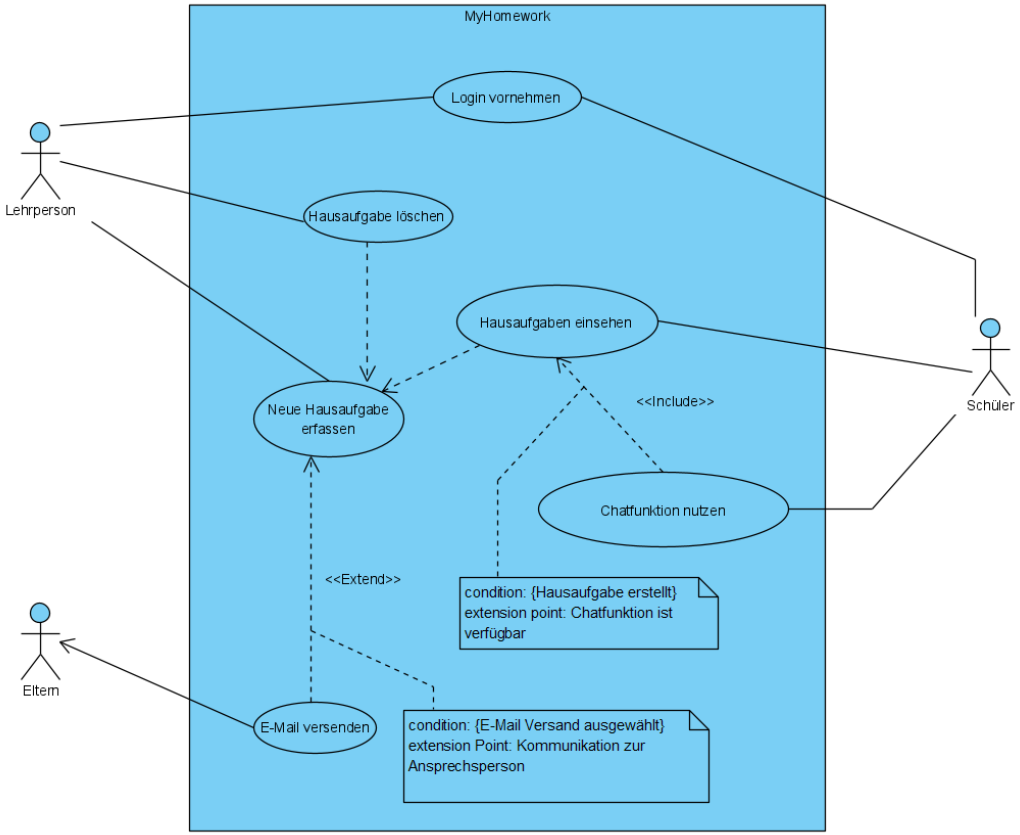


Abbildung 20: Use Case Diagramm des Prototyps
Quelle: eigene Darstellung

In Abbildung 20 ist der Anwendungsfall mittels Use Case Diagramm zu sehen. Es ist ersichtlich, dass dabei drei Akteure im Use Case mit dem System interagieren. Die Lehrperson, welche als Admin fungiert, kann sich im System einloggen, eine neue Hausaufgabe erstellen und erstellte Hausaufgaben wieder löschen. Die Schüler können sich ebenfalls mit einem Schüleraccount anmelden und dadurch die erstellten Hausaufgaben einsehen und die integrierte Chatfunktion nutzen. Schliesslich gibt es die Eltern, die lediglich eine Mail vom System erhalten, sofern die Lehrperson dies beim Erstellen der Hausaufgabe so vermerkt hat. Um eine Hausaufgabe zu löschen oder eine Hausaufgabe einzusehen, muss diese zuerst erstellt worden sein, weshalb in diesem Fall Abhängigkeiten bestehen.

3.3.2 Sequenzdiagramm

Abbildung 21 zeigt die Sequenz beim Erstellen einer Hausaufgabe an. Sobald bekannt ist, dass eine Hausaufgabe erfasst werden soll, kann sich die Lehrperson, hier der «Actor», beim System über den Webserver anmelden. Der Webserver übernimmt dabei die Authentifikation, wobei die Abfrage der Anmeldedaten bei der Datenbank durchgeführt wird. Sobald das Login erfolgreich ist, kann eine neue Hausaufgabe auf dem Webserver erfasst werden. Die Hausaufgabe wird dabei auf dem AMQP-Server in die Warteschlange eingefügt. Der Webserver konsumiert dann die Hausaufgabe, und fügt sie in die Datenbank ein. Wurde beim Erfassen ausgewählt, dass die Eltern ein Mail bekommen sollten, so wird die Nachricht vom Webserver via AMQP-Server an den Mailserver übermittelt, der dann via SMTP den E-Mail Versand übernimmt. Schliesslich findet ein Datenbankaufruf statt, welcher dann der Lehrperson die Erfassung der Hausaufgabe bestätigt.

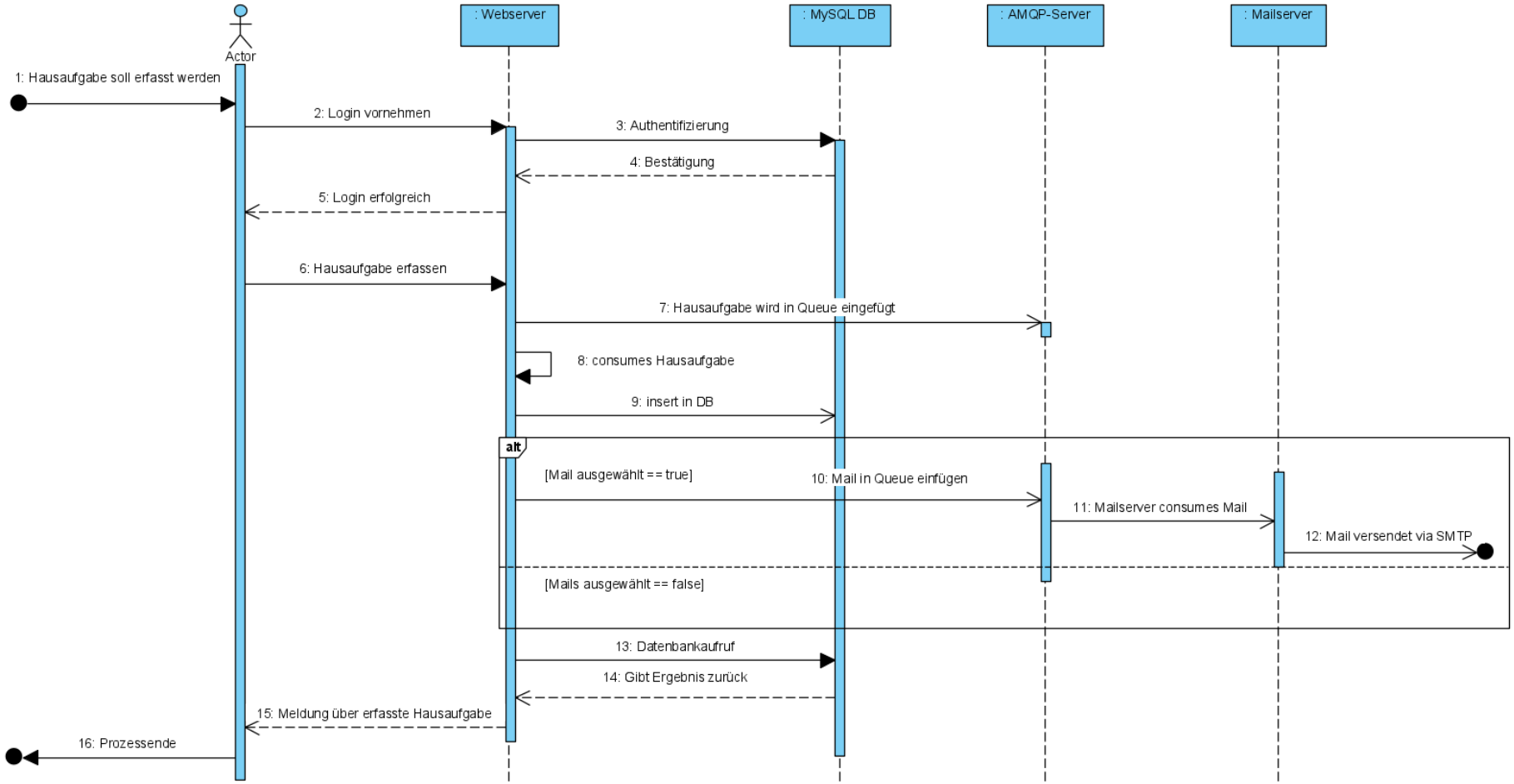


Abbildung 21: Sequenzdiagramm des Prototyps
Quelle: eigene Darstellung

3.3.3 UML Diagramm

Um ein besseres Verständnis der Webapplikation zu erhalten, ist in Abbildung 22 die Architektur des Prototyps mittels UML-Diagramm dargestellt. Es ist ersichtlich, dass der Browser mittels eines initialen HTTP-Request auf die Angular Komponenten zugreift. Ausserdem verbindet sich der Client über Socket.io mit dem NodeJS Server. Nach dem initialen Request findet die fortlaufende Kommunikation zwischen Client und Server über Socket.io statt. Socket.io stellt sicher, dass Daten vom Client an den Server gesendet werden, um eine bestimmte serverseitige Operation auszuführen. Es dient allerdings auch dazu, die Daten vom Server an die Clients zurückzusenden. Dies ist insbesondere bei der eingebauten Chat Funktion wichtig, damit alle verbundenen Clients in Echtzeit die Nachricht erhalten.

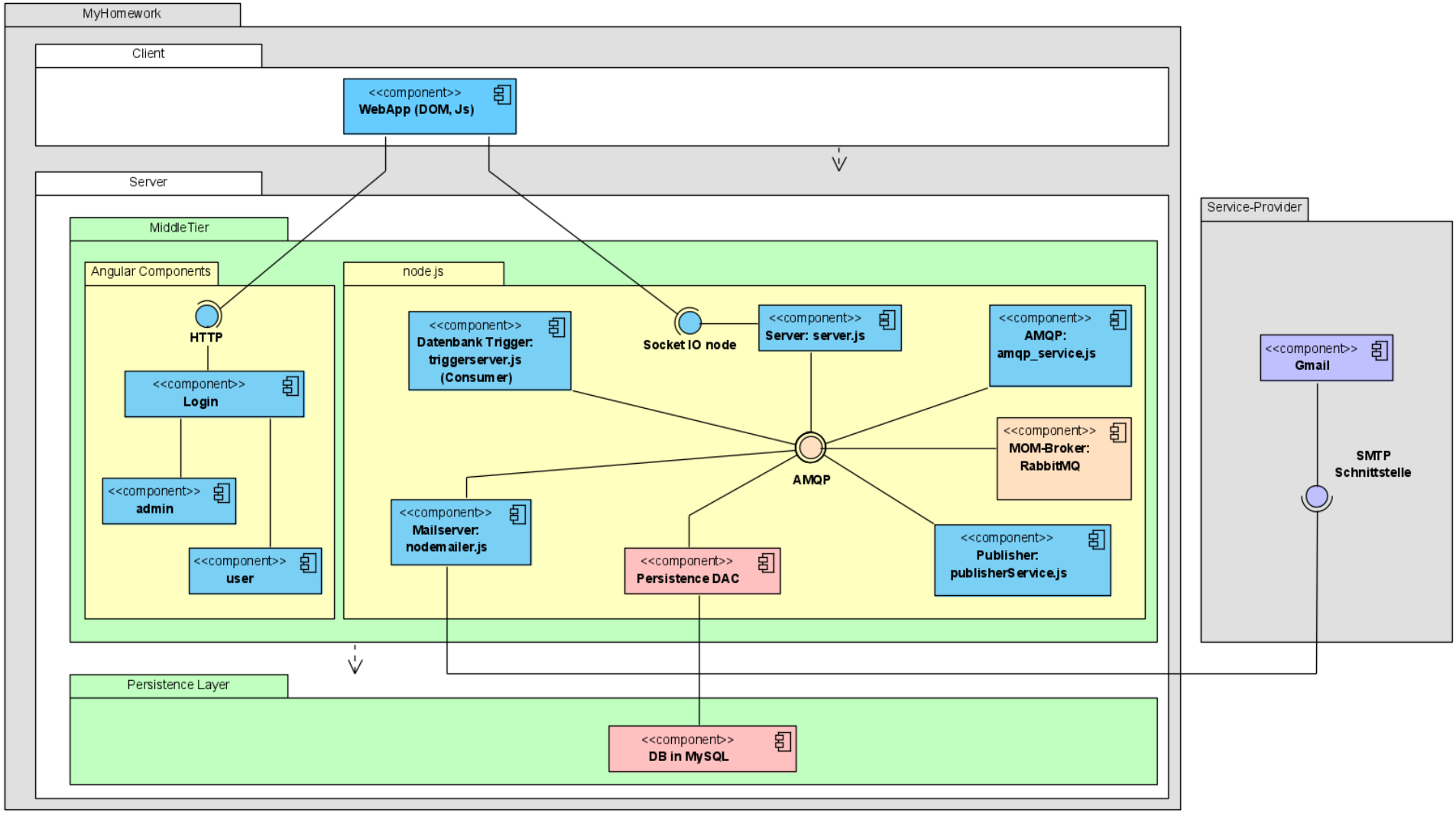


Abbildung 22: UML-Diagramm des Prototyps
Quelle: eigene Darstellung

3.4 Dokumentation

In diesem Unterkapitel soll der erstelle Code dokumentiert werden. Wichtig zu erwähnen ist, dass nicht auf alle Code-Abschnitte direkt eingegangen wird. Es werden einzelne wichtige Funktionen, sowie die Logik der Clients, der Server, der Mailserver, der Triggerserver und der Datenbankserver erläutert und aufgezeigt. Der vollständige Code ist im Anhang A - W zu finden.

In Tabelle 4 ist die Bedeutung der jeweiligen Server des MOM-Projekts aufgezeigt.

Server	Hauptserver, der die Webseite hostet und die Grundfunktionalität zur Verfügung stellt. (NodeJS Server)
Mailserver	Zuständig für das Versenden der E-Mails, sobald eine neue Hausaufgabe erfasst wird. (SMTP Server)
Triggerserver	Server, der nur dafür zuständig ist, einen Datenbankeintrag nach 7 Tagen zu löschen. (NodeJS Server)
Datenbankserver	Ermöglicht den Zugriff auf die MySQL-Datenbank. (MySQL Workbench)

Tabelle 4: Server
Quelle: eigene Darstellung

In den folgenden Unterkapiteln wird auf das Frontend und deren Komponenten sowie Services eingegangen. Auf die vom Angular Framework erstellten Standardfiles wird in der Dokumentation keinen Bezug genommen. Auch die CSS-Dateien werden nicht speziell erwähnt, da diese lediglich zum Design, aber nicht zur Funktionalität beigetragen haben.

3.4.1 Login Component

Die Abbildung 23 zeigt die HTML-Datei der «Login Component» auf. In dieser Datei wurden sowohl Bootstrap-Klassen (z.B. «col-lg-auto») verwendet. Die «Login Component» bietet die Möglichkeit, sich mittels zwei Input Feldern anzumelden. Ausserdem ist ein Button ersichtlich, der bei einem Klick die Funktion «login()» ausführt, welche in der TypeScript-Datei definiert ist.

```
Go to component
<header>
  <script src="../../../src/assets/custom.js"></script>
  <script src="../../../src/assets/send.js"></script>
  <script src="../../../src/assets/receive.js"></script>
  <script type="text/javascript" src="../sendRabbitMQ.js"></script>

</header>
<body>
  <div class="body">
    <div class="background">
      <div class="spacer500"></div>
        <div class="container">
          <div class="card text-center wow fadeInUp col-lg-auto col-md-auto col-sm-auto col-xs-auto">
            <div class="ownCardHeader wow fadeInUp">
              💬 Schülertool
            </div>
            <div class="card-body">
              <h5 class="card-title wow fadeInUp">Hey 👋, melde dich hier an:</h5>
              <div id="login" class="">
                <form>
                  <input class="InputStyle wow fadeInUp" placeholder="Username" type="text" id="usernameInput" autofocus="true">
                  <br>
                  <input class="InputStyle wow fadeInUp" placeholder="Passwort" type="password" id="passwordInput" autofocus="false">
                  <br>
                  <button type="button" class="ownButton wow fadeInUp" (click)="login()">Lets go</button>
                </form>
              </div>
            </div>
          <div class="ownCardFooter text-muted wow fadeInUp">
            Created by Elias Züst, Finn Graf, Nico Kruse & Sandro Caroppo
          </div>
        </div>
    </div>
  </div>
</div>
</body>
<footer>
</footer>
```

Abbildung 23: HTML-Code Login Component
Quelle: eigene Darstellung

Die «login()»-Funktion nimmt dabei den Input der beiden eingegebenen Felder entgegen, hashed das Passwort mit einem «sha512-Algorithmus», damit das Passwort nicht 1:1 in der Datenbank abgelegt wird und übergibt dieses dem WebSocket-Service, der die Daten über den «mysqldata»-Kanal an den Server übergibt. Der Server kann somit die Daten mit einer «listen()»-Funktion empfangen. Die Login-Funktion ist in Abbildung 24 dargestellt.

```
login() {
  // @ts-ignore: Object is possibly 'null'.
  var usernamemsg = document.getElementById('usernameInput').value;
  // @ts-ignore: Object is possibly 'null'.
  var passwordmsg = document.getElementById('passwordInput').value;
  this.webSocketService.emit('username', usernamemsg);

  var hashedpassword = sha512.sha512(passwordmsg);

  this.webSocketService.emit2('mysqldata', usernamemsg, hashedpassword);
  //this.router.navigate(['/chat']);
}
```

Abbildung 24: Typescriptcode Login-Funktion
Quelle: eigene Darstellung

Der WebSocket-Service ist ein individuell erstellter Service, der im «Constructor Socket.io» initialisiert ist. Anschliessend wurde eine «listen»-Funktion definiert, um die vom Server gesendeten Daten zu empfangen, sowie vier «Emit»-Funktionen, um die Daten an den Server zu übertragen. Die Zahl im Namen der Funktion gibt an, wie viele Parameter an den Server übergeben werden. Die Abbildung 25 zeigt die Implementation des WebSocket-Service auf.

```
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { io } from 'socket.io-client';

@Injectable({
  providedIn: 'root'
})
export class WebSocketService {
  socket: any;
  constructor() {
    this.socket = io();
  }

  listen(eventName: string){
    return new Observable((subscriber) =>{
      this.socket.on(eventName, (data: any) =>{
        subscriber.next(data);
      })
    });
  }

  emit(eventName: string, data: any){
    this.socket.emit(eventName, data);
  }

  emit2(eventName: string, data: any, data2: any){
    this.socket.emit(eventName, data, data2);
  }

  emit3(eventName: string, data: any, data2: any, messageSender: any){
    this.socket.emit(eventName, data, data2, messageSender);
  }

  emit6(eventName: string, fach: any, title: any, message: any, date: any, creation_date: any, parentsCC: any){
    this.socket.emit(eventName, fach, title, message, date, creation_date, parentsCC);
  }
}
```

Abbildung 25: Typescriptcode Socket.io
Quelle: eigene Darstellung

In der «Login Component» wird bei der Initialisierung diese «listen()»-Funktion verwendet, um Rückmeldungen des Servers zu empfangen. In Abbildung 26 sind dabei zwei Listen Funktionen definiert, die einmal auf dem Kanal «loginAuthorized» und das andere Mal auf dem «loginNotAuthorized» hören. Je nach Antwort des Servers, darf sich der User entweder einloggen oder er wird gebeten, das Passwort erneut einzutippen. Wird der Username «Admin» eingegeben, so landet man im Admin-Bereich. Alle weiteren User werden als Schüler eingeloggt.

```
ngOnInit(): void {
  this.webSocketService.listen('loginAuthorized').subscribe((data: any) => {
    var username = data;
    myUser = new User(username);
    Swal.fire({
      icon: 'success',
      title: 'Du hast dich als ' + data + ' erfolgreich eingeloggt!',
      showConfirmButton: false,
      timer: 1750
    })

    if(username == 'Finn'){
      this.router.navigate(['/admin']);
    }else{
    this.router.navigate(['/chat', username]);
    }
  });


  this.webSocketService.listen('loginNotAuthorized').subscribe((data: any) => {
    var username: string = data;
    Swal.fire({
      icon: 'error',
      title: 'Oops...',
      text: 'Dein Login für den User ' + data + " ist fehlgeschlagen",
      showCancelButton: false,
      timer: 1500
    })

  });
```

Abbildung 26: Typescriptcode Login
Quelle: eigene Darstellung

3.4.2 Admin Component

Die «Admin Component» bietet für die Lehrerinnen und Lehrer die Möglichkeit eine neue Hausaufgabe zu erfassen. Beim Erfassen der Hausaufgabe kann ausgewählt werden, ob die Nachricht per Mail an die Eltern versendet werden soll. Wenn dem so ist, wird mittels AMQP über den Mailserver eine Nachricht an die hinterlegte E-Mail-Adresse gesendet. Die Abbildung 27 zeigt den dazugehörigen HTML-Code auf.

```
<body>
  <div class="background">
    <div class="containerown">
      <div class="h-100 row align-items-center">
    <div class="titleHomework">
      Erfasse neue Hausaufgaben
    </div>
  </div>
    </div>


      <div class="h-100 row align-items-center">
        <div class="container ownMarginClass mt-4 ownClass">
          <div class="wow fadeInUp col-lg-auto col-md-auto col-sm-auto col-xs-auto">

      <div class="card">
        <select class="form-select" name="fach" id="fachselector">
          <option value="Mathe">Mathe 📘</option>
          <option value="Deutsch">Deutsch 🎨</option>
          <option value="Französisch">Französisch 📏</option>
          <option value="Englisch">Englisch ⚛️</option>
          <option value="Informatik">Informatik 👨‍💻</option>
        </select><br>
        <input type="text" id="title" class="InputStyle" placeholder="Titel"> <br>
        <textarea type="text" id="message" class="InputStyleMessage" placeholder="Nachricht"></textarea><br>
        <input type="text" onfocus="(this.type = 'date')" id="date" class="dateInput" placeholder="Abgabedatum eintragen"> <br>
        <div class="form-check">
        <input class="form-check-input" type="checkbox" id="parentsCC">
        <label for="parentsCC" class="form-check-label">E-Mail an Eltern senden</label></div><br>
        <button class="button" (click)="msgSend()"> Senden </button>
      </div>
```

Abbildung 27: HTML-Code Userseite
Quelle: eigene Darstellung

Ausserdem beinhaltet der Admin-Bereich ein Container für alle bereits erstellten Hausaufgaben. Der Lehrer kann dort mit dem «Löschen»-Button die erstellten Nachrichten löschen. Dazu wurde eine «Template-Component» erstellt, welche dynamisch eingefügt wird. Die Abbildung 28 zeigt den dazugehörigen HTML-Code auf.

```
<template #createdContainer>
</template>
```

Abbildung 28: HTML-Code Template
Quelle: eigene Darstellung

In der dazugehörigen TypeScript-Datei werden mittels der «msgSend()»-Funktion die eingegebenen Daten ausgelesen und über den WebSocket-Service an den Server übergeben. Die Abbildung 29 illustriert diesen Sachverhalt.

```
msgSend(){
  // @ts-ignore: Object is possibly 'null'.
  var fach = document.getElementById("fachselector").value;
  // @ts-ignore: Object is possibly 'null'.
  var title = document.getElementById('title').value;
  // @ts-ignore: Object is possibly 'null'.
  var message = document.getElementById('message').value;
    // @ts-ignore: Object is possibly 'null'.
    var date = document.getElementById('date').value;
      // @ts-ignore: Object is possibly 'null'.
  if(document.getElementById('parentsCC').checked == true){
    // @ts-ignore: Object is possibly 'null'.
  var parentsCC = "yes";
  }else{
    var parentsCC = "no";
  }

  var current_date = new Date()
  var c_date = current_date.toISOString().split('T')[0]

  var creation_date = c_date;

  this.webSocketService.emit6('newHomework', fach, title, message, date, creation_date, parentsCC);
```

Abbildung 29: Typescriptcode msgSend()-Funktion
Quelle: eigene Darstellung

Eine weitere wichtige Funktionalität ist der «Listener», der beim Einloggen als Admin alle bereits erstellen Hausaufgaben vom Server lädt. Hierzu hört der WebSocket-Service auf den Kanal «getData» und weist die erhaltenen Daten neuen Variablen zu. Anschliessend wird die Funktion «createComponent()» aufgerufen, um die bereits erstellten Hausaufgabeneinträge dynamisch in den Container einzufügen. Die Funktion des WebSocket-Service Listeners ist in Abbildung 30 ersichtlich. Die Umsetzung der dynamischen Darstellung wird in der Abbildung 31 aufgezeigt.

```
ngOnInit(): void {
  this.webSocketService.listen('getData').subscribe((data: any) => {


    var newMessage = JSON.parse(data);
    var id = newMessage['id'];
    var fach = newMessage['fach'];
    var title = newMessage['title'];
    var message = newMessage['message'];
    if(message.length > 27){
    var messageShort = message.substr(0, 27) + "...";
    } else{
    messageShort = message;
    }
    var date = newMessage['date'];
    var parentsCC = newMessage['parentsCC'];



    this.createComponent(fach, title, messageShort, date, parentsCC, message, id);
    var created = document.getElementsByTagName('app-created-homework-template').length;
    var i = 0;
    while (i < created) {
      var newdoc = document.getElementsByTagName('app-created-homework-template')[i];
      newdoc.setAttribute('class', 'col-auto');
      i++;
    }

  });
}
```

Abbildung 30: Typescriptcode Websocket-Service
Quelle: eigene Darstellung

```
createComponent(fach: any, title: any, message: any, date: any, parentsCC: any, messageLong: any, id: any) {
  // @ts-ignore: Object is possibly 'null'.
  var text = fach + title + message + date + parentsCC;

  if (this.array.includes(text) == false) {
    this.array.push(text);

    const factory = this.resolver.resolveComponentFactory(CreatedHomeworkTemplateComponent);
    // @ts-ignore: Object is possibly 'null'.
    const componentRef = this.entry.createComponent(factory, 0);

    componentRef.instance.fach = fach;
    componentRef.instance.title = title;
    componentRef.instance.message = message;
    componentRef.instance.date = date;
    componentRef.instance.parentsCC = parentsCC;
    componentRef.instance.id = id;
    componentRef.instance.messageLong = messageLong;

    this.newClass = document.getElementsByClassName('identifyClass')[0];
    this.newID = document.getElementsByTagName('app-created-homework-template')[0];
    this.newID.setAttribute('id', id);
```

```
switch (fach) {
  case "Mathe": {
    this.newClass.classList.add('cardOne');
    break;
  }
  case "Deutsch": {
    this.newClass.classList.add('cardTwo');
    break;
  }
  case "Französisch": {
    this.newClass.classList.add('cardThree');
    break;
  }

  case "Englisch": {
    this.newClass.classList.add('cardFour');
    break;
  }

  case "Informatik": {
    this.newClass.classList.add('cardFive');
    break;
  }
  default: {
    this.newClass.classList.add('cardThree')
    break;
  }
}
```

Abbildung 31: Typescriptcode createComponent
Quelle: eigene Darstellung

3.4.3 Created Homework Template Component

Diese Komponente dient als Vorlage für die dynamisch zu erzeugenden Hausaufgaben-Elemente, welche bereits erstellt wurden. Sie sollen die grafische Möglichkeit bieten, die erstellten Einträge wieder aus dem System zu löschen. Die Abbildung 32 zeigt den dazugehörigen HTML Code auf.

```
<div class="col-lg-auto col-md-auto col-sm-auto col-xs-auto mb-3 ml-3 wow fadeInUp">
    <div class="card hvr-grow-rotate identifyClass wow fadeInUp">
        <div class="card-body wow fadeInUp">
            <h5 class="card-title wow fadeInUp">{{fach}} </h5>
            <h6 class="card-subtitle mb-2 wow fadeInUp">{{title}}</h6>
            <p class="card-text wow fadeInUp">{{message}}</p>
            <button type="button" class="btn btn-danger" id="deleteButton{{id}}" (click)="deleteHomework(id)">🗑</button>
        </div>
    </div>
</div>
```

Abbildung 32: HTML-Code Homework
Quelle: eigene Darstellung

Die TypeScript Datei beinhaltet lediglich die Funktionalität, den Eintrag zu löschen. Dieser wird über die «deleteHomework()»-Funktion an den Server übermittelt. Dieser verarbeitet den Vorgang. Ist die Löschung erfolgreich, so erhält der Browser eine Rückmeldung und das Element mit der «el.remove()»-Funktion löschen. Dies ist in der Abbildung 33 dokumentiert.

```
ngOnInit(): void {

  this.webSocketService.listen('onDelete').subscribe((data: any) => {

    var el = document.getElementById(data);
    // @ts-ignore: Object is possibly 'null'.
    if(el != null){
    el.remove();
    }

  });

}

deleteHomework(id: any){
  this.webSocketService.emit('deleteID', id)
}
```

Abbildung 33: Typescriptcode deleteHomework
Quelle: eigene Darstellung

3.4.4 User Component

Die «User Component» ist dafür zuständig, den Schülerinnen und Schülern die erstellten Hausaufgaben anzuzeigen. Da die erstellten Hausaufgaben dynamisch eingefügt werden müssen, wurde auch hier eine Template Component erstellt, welche bei der Erstellung eines Eintrags Lehrperson, diese den Schülerinnen und Schülern anzeigt. Die Abbildung 34 zeigt den dazugehörigen HTML-Code auf.

```
<body>

    <div class="background wow fadeInUp">


        <div class="spacer100"></div>
        <div>
            <h1 class="titlewise wow fadeInUp">My Homework</h1>
        </div>
        <div class="h-20">
            <div class="container mt-4 ownClass">
                <div class="row align-items-center" id="dataID">
                    <template #messagecontainer>
                    </template>
                </div>
            </div>
        </div>


    </div>
</body>

<footer></footer>
```

Abbildung 34: HTML-Code User Component
Quelle: eigene Darstellung

Die dazugehörige TypeScript Datei dient dazu, die erstellten Daten zu empfangen. Mit der «createComponent()»-Funktion wird dynamisch pro erstelltem Dokument, welches in der Datenbank ist, eine Template-Komponente erzeugt. Die Abbildung 35 zeigt diesen Sachverhalt auf.

```
ngOnInit(): void {
  this.webSocketService.listen('clear').subscribe((data: any) => {
    this.entry.clear();
  });
  this.username  = this.router.snapshot.paramMap.get('username');
  this.webSocketService.listen('getData').subscribe((data: any) => {

    var newMessage = JSON.parse(data);
    var fach = newMessage['fach'];
    var title = newMessage['title'];
    var message = newMessage['message'];
    var id = newMessage['id'];
    if(message.length > 27){
    var messageShort = message.substr(0, 27) + "...";
    } else{
    messageShort = message;
    }
    var date = newMessage['date'];
    var parentsCC = newMessage['parentsCC'];

    this.createComponent(fach, title, messageShort, date, parentsCC, message, id);

    var created = document.getElementsByTagName('app-homework-template').length;
    var i = 0;
    while (i < created) {
      var newdoc = document.getElementsByTagName('app-homework-template')[i];
      newdoc.setAttribute('class', 'col-auto');
      i++;
    }

  });


}
```

Abbildung 35: Typescriptcode createComponent Variablen
Quelle: eigene Darstellung

Die «createComponent()»-Funktion ist dafür zuständig diese Komponenten dynamisch in das Document Object Model einzufügen und die jeweiligen Werte des Servers für die bestimmte Hausaufgaben zuzuweisen. Die Abbildung 36 zeigt dies auf.

```
createComponent(fach: any, title: any, message: any, date: any, parentsCC: any, messageLong: any, id: any) {
  // @ts-ignore: Object is possibly 'null'.
  //this.entry.clear();
  var text = fach + title + message + date + parentsCC;

  if (this.array.includes(text) == false) {
    this.array.push(text);
    const factory = this.resolver.resolveComponentFactory(HomeworkTemplateComponent);
    // @ts-ignore: Object is possibly 'null'.
    const componentRef = this.entry.createComponent(factory, 0);

    componentRef.instance.fach = fach;
    componentRef.instance.title = title;
    componentRef.instance.message = message;
    componentRef.instance.date = date;
    componentRef.instance.parentsCC = parentsCC;
    componentRef.instance.id = id;
    componentRef.instance.messageLong = messageLong;
    this.index++;

    this.newClass = document.getElementsByClassName('identifyClass')[0];
    this.newID = document.getElementsByTagName('app-homework-template')[0];
    this.newID.setAttribute('id', id);


    switch (fach) {
      case "Mathe": {
        this.newClass.classList.add('cardOne');
        break;
      }
      case "Deutsch": {
        this.newClass.classList.add('cardTwo');
        break;
      }
      case "Französisch": {
        this.newClass.classList.add('cardThree');
        break;
      }

      case "Englisch": {
        this.newClass.classList.add('cardFour');
        break;
      }

      case "Informatik": {
        this.newClass.classList.add('cardFive');
        break;
      }
      default: {
        this.newClass.classList.add('cardThree');
      break;
      }
    }
  } else {
  }
}
```

Abbildung 36: Typescript createComponent Switch
Quelle: eigene Darstellung

In den vier bisherigen Unterkapiteln sind somit alle benötigten Angular-Komponenten dokumentiert worden. In den folgenden Unterkapiteln wird auf die verwendeten Server und deren Funktionalitäten eingegangen.

3.4.5 Server.js

Diese Datei stellt den Hauptserver dar. Er ist für das Hosting der Webseite zuständig und ermöglicht es die Webseite unter «localhost:8082» aufzurufen. Ausserdem nimmt er die Socket-Anfragen des Clients entgegen und sendet Antworten an diesen zurück. Auch werden Verbindungen zum AMQP-Server aufgebaut, um bestimmte Nachrichten abzusetzen.

Die Abbildung 37 zeigt auf, wie der Server konfiguriert wurde. Es wurde ExpressJS verwendet und als Datei, welche den Server hosten soll, wurde in der zweitletzten Zeile «app.use(express.static('firstapp'));» geschrieben. Dies bedeutet, dass der Server den firstapp Ordner im Verzeichnis hosten wird. Ausserdem wurde socket.io auch serverseitig initialisiert, um die Anfragen des Clients entgegenzunehmen.

```
#!/usr/bin/env node
const express = require('express');
const app = express();
const http = require('http');
const server = http.createServer(app);
var persistanceService = require('./persistanceService')

io = require('socket.io')(server);


const bodyParser = require('body-parser');
const { EventEmitter } = require('stream');
const { checkLogin } = require('./persistanceService');
const amqpservice = require('./amqpservice');

app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());

app.use(express.static('firstapp'));


var emitter = new EventEmitter();

server.listen(8082, () => {
    console.log('Server listening on *:8082');
});
```

Abbildung 37: Javascriptcode Server.js
Quelle: eigene Darstellung

Eine weitere Funktionalität ist das Überprüfen der Nachrichten, welche bereits älter als 7 Tage sind, überhaupt gelöscht werden sollen. Dies wird jeweils jeden Tag um Mitternacht überprüft. Die Abbildung 38 zeigt diese Funktionalität auf.

```
/* JEDEN TAG um Mitternach überprüfen, ob Einträge älter als 7 Tage sind, wenn dem so ist, werden diese gelöscht!*/
resetAtMidnight();

function resetAtMidnight() {
    var now = new Date();
    var night = new Date(
        now.getFullYear(),
        now.getMonth(),
        now.getDate() + 1,  // nächster Tag
        0, 0, 0,// um 0.00 Uhr
    );
    var msToMidnight = night.getTime() - now.getTime();

    setTimeout(function() {
        var current_date = new Date()
        var c_date = current_date.toISOString().split('T')[0]
        amqpservice.sendTriggerViaAMQP();
    }, msToMidnight);
}
```

Abbildung 38: Javascriptcode Triggerfunktion
Quelle: eigene Darstellung

Eine weitere wichtige Aufgabe des server.js ist es, die Socket-Nachrichten der Clients zu empfangen. Die Abbildung 39 zeigt dabei den für diesen Zweck zuständigen Code auf. Bei dem ersten socket.on() hört der Server auf den Channel «data» und nimmt dabei 3 Parameter entgegen. Diese sind einerseits «data», welche die gesendete Nachricht enthält, andererseits die ID, damit zugeordnet werden kann, in welchem Chatraum die Nachricht versendet wurde und schliesslich auch der «messageSender», damit später der Name des Senders in der Nachricht angezeigt werden kann.

```
io.on('connection', function (socket) {
    console.log('Ein neuer Client hat sich verbunden:' + socket.id);
    // SOCKET LISTENER
    socket.on('data', function (data, id, messageSender) {
        send(data, 'hello'+id, id, messageSender);
    });

    socket.on('getMessageHistory', function(chatid){
        persistanceService.getMessagesFromDatabaseExported(chatid);
    })

    socket.on('deleteID', function(data){
        persistanceService.deleteFromDatabase(data);
    })
    socket.on('mysqldata', function (data, data2) {
        //Schaut ob Passwort übereinstimmt
        logIn();
        function logIn(){
            checkLogin(data, data2, function(wert){
                if(wert == true){
                    socket.username = data;
                    socket.emit('loginAuthorized', data);
                }else{
                    socket.emit('loginNotAuthorized', data);
                }
            });
        }

    });
```

Abbildung 39: Javascriptcode Socketio
Quelle: eigene Darstellung

Einen weiteren Socket-Listener gibt es für das Laden der «MessageHistory», für das Löschen eines Eintrages, der von den Lehrerinnen und Lehrern ausgelöst wird und für das Anmelden des Nutzers. Hierzu wird die Funktion «checkLogin()» aufgerufen, welche ein Callback-Funktion beinhaltet, um zu überprüfen, ob der Username und das Passwort übereinstimmen. Die «checkLogin()»-Funktion ist im PersistenceService definiert. Auch alle anderen «socket.on()»-Funktionen verweisen auf diesen Service. Die Funktionsweise des PersistenceService wird im Unterkapitel 3.4.6 ausgeführt.

Eine weitere Aufgabe, welche die server.js Datei übernimmt, ist es, die neuen Hausaufgaben sowie Nachrichten als Messages über AMQP zu versenden. Hierzu dienen jeweils die Funktionen «sendNewHomework()» sowie lediglich die «send()» Funktion, welche das Versenden von Nachrichten in den Chaträumen über AMQP ermöglicht. Diese Funktionalitäten werden direkt wieder vom selben Server aus der AMQP-Queue konsumiert. Dies soll aufzeigen, dass man die Warteschlangen grundsätzlich auch auf einem Server verwenden kann. In einem späteren Beispiel wird auch aufgezeigt, dass die Nachrichten über mehrere Server verteilt versendet bzw. konsumiert werden können. Die Abbildung 40 und 41 zeigen diese beiden Funktionalitäten auf.

```
function send(message, queuevar, idchat, messageSender) {
    var amqp = require('amqplib/callback_api');

    amqp.connect('amqp://localhost', function (error0, connection) {
        if (error0) {
            throw error0;
        }
        connection.createChannel(function (error1, channel) {
            if (error1) {
                throw error1;
            }

            var myMessageObject = {};
            myMessageObject['message'] = message;
            myMessageObject['messageSender'] = messageSender;
            var myMessageObjectString = JSON.stringify(myMessageObject);

            var queue = queuevar;
            var msg = message;

            channel.assertQueue(queue, {
                durable: true
            });
            channel.sendToQueue(queue, Buffer.from(myMessageObjectString));

            console.log(" [x] Nachricht gesendet %s", myMessageObjectString);

            receiveMSG(queuevar, idchat);

        });
        setTimeout(function() {
            connection.close();
        }, 500);
    });
};
```

Abbildung 40: Javascriptcode send-Funktion
Quelle: eigene Darstellung

```
var sendNewHomework = function (message, queuevar) {
    var amqp = require('amqplib/callback_api');

    amqp.connect('amqp://localhost', function (error0, connection) {
        if (error0) {
            throw error0;
        }
        connection.createChannel(function (error1, channel) {
            if (error1) {
                throw error1;
            }
            var queue = queuevar;
            var msg = message;
            channel.assertQueue(queue, {
                durable: true
            });
            channel.sendToQueue(queue, Buffer.from(msg));
            console.log(" [x] Nachricht gesendet %s", msg);
            receiveNewHomework();

        });
        setTimeout(function() {
            connection.close();
        }, 500);


    });
```

Abbildung 41: Javascriptcode sendNewHomework-Funktion
Quelle: eigene Darstellung

3.4.6 persistenceService.js

In diesem Service wird einmalig die Verbindung zur Datenbank aufgebaut. Der Service übernimmt ansonsten alle Funktionalitäten, um die Daten in die Datenbank hinzuzufügen, sie auszulesen oder zu löschen. Die Abbildung 42 zeigt die Verbindung zur MySQL-Datenbank auf.

```
1   var mysql = require('mysql');
2   var amqppublisher = require('./publisherService');
3   var amqpservice = require('./amqpservice')
4
5   var con = mysql.createConnection({
6       host: "localhost",
7       user: "root",
8       password: "root"
9   });
```

Abbildung 42: Javascriptcode Persistenceservice
Quelle: eigene Darstellung

Beispielsweise wird auf die «insertIntoDatabase()»-Funktion eingegangen, welche die Funktion übernimmt, neue Hausaufgaben in die MySQL-Datenbank einzutragen. Die Abbildung 43 zeigt diese Funktionalität auf. Weitere Funktionalitäten dieses Services sind, dass ältere Nachrichten aus der Datenbank gelöscht werden, neue Nachrichten aus den Chaträumen in die Datenbank hinzugefügt werden, der Login des Users überprüft wird oder die erstellten Hausaufgaben seitens der Lehrerinnen und Lehrer gelöscht wird.

```
insertIntoDatabase: function(fach, title, messageone, date, creation_date, ccint) {
    con.query("INSERT INTO homework(fach, title, message, date, created, parentsCC) VALUES('" + fach + "', '" + title + "',
        if (err) {
            return false;

        } else {
            console.log("MYSQL Eintrag erstellt!");
            if(ccint == 1){
                amqppublisher.sendEmail('finn.graf@ost.ch', fach, title, messageone, date);
            }
            getDataFromDatabase();
        }
    });
},
```

Abbildung 43: Javascriptcode InsertIntoDatabase
Quelle: eigene Darstellung

3.4.7 amqpService.js

In diesem Service werden weitere Funktionalitäten von AMQP definiert. Es ermöglicht, dass die Nachrichten wiederum über AMQP konsumiert werden und an alle Clients zurückgesendet werden können. In der Abbildung 44 ist dies ersichtlich.

```
channel.consume(queue, function (msg) {

    io.emit('test'+idchat, msg.content.toString());
```

Abbildung 44: Javascriptcode consumemsg
Quelle: eigene Darstellung

Auch übernimmt der Service die Funktion, die bei einem Trigger, der das Löschen eines Eintrages auslöst, diese Nachricht in die AMQP Warteschlange setzt. Die Abbildung 45 zeigt diese Funktion auf.

```
function sendTriggerViaAMQP(){
    var amqp = require('amqplib/callback_api');

    amqp.connect('amqp://localhost', function (error0, connection) {
        if (error0) {
            throw error0;
        }
        connection.createChannel(function (error1, channel) {
            if (error1) {
                throw error1;
            }

            var queue = "TriggerQueue";
            var msg = "Item found, that should be deleted.";

            channel.assertQueue(queue, {
                durable: true
            });
            channel.sendToQueue(queue, Buffer.from(msg));


        });
        setTimeout(function() {
            connection.close();
        }, 500);
    });
```

Abbildung 45: Javascriptcode sendTriggerViaAMQP
Quelle: eigene Darstellung

3.4.8 publisherService.js

Dieser Service übernimmt die Funktionalität, dass eine zu versendende E-Mail in die AMQP Message Queue gelegt wird. Anschliessend kann diese Nachricht auf einem anderen Server konsumiert und das E-Mail versendet werden. Die Abbildung 46 zeigt die Funktionalität, die dafür zuständig ist, die Nachricht in die AMQP Warteschlange zu legen.

```
sendEmail: function(receiver, subject, title, message, date){
    amqplib.connect('amqp://localhost', (err, connection) => {
        if (err) {
            console.error(err.stack);
            return process.exit(1);
        }

        // Create channel
        connection.createChannel((err, channel) => {
            if (err) {
                console.error(err.stack);
                return process.exit(1);
            }

            queue = "MailAnEltern";

            // Ensure queue for messages
            channel.assertQueue(queue, {
                // Ensure that the queue is not deleted when server restarts
                durable: true
            }, err => {
                if (err) {
                    console.error(err.stack);
                    return process.exit(1);
                }

                // Create a function to send objects to the queue
                // Javascript object is converted to JSON and then into a Buffer
                let sender = (content, next) => {
                    let sent = channel.sendToQueue(queue, Buffer.from(JSON.stringify(content)), {
                        // Store queued elements on disk
                        persistent: true,
                        contentType: 'application/json'
                    });
                    if (sent) {
                        return next();
                    } else {
                        channel.once('drain', () => next());
                    }
                };

                let sendNext = () => {

                    console.log('E-Mail in RabbitMQ gelegt');
                    // Close connection to AMQP server
                    // We need to call channel.close first, otherwise pending
                    // messages are not written to the queue
                    return channel.close(() => connection.close());
                }

                sender({
                    to: receiver,
                    subject: subject+ ": " + title,
                    text: message + " Dies ist zu erledigen bis: " + date,
                    html: "<h1>"+ message + " Dies ist zu erledigen bis: "+ date + "</h1>"
                }, sendNext);


                console.log("Ausgeführt!");

            });
        });
    });

},
```

Abbildung 46: Javascriptcode sendEmail-Funktion
Quelle: eigene Darstellung

3.4.9 triggerserver.js

Dieser Server dient dazu, die Nachrichten in der AMQP Warteschlange zu konsumieren, welche jeden Tag für das Löschen eines älteren Eintrages versendet werden.

Der Server verbindet sich zur MySQL Datenbank und ach zum AMQP Server. Die Verbindung zur MySQL Datenbank ist in der Abbildung 47 dargestellt.

```
const http = require('http');
const amqplib = require('amqplib/callback_api');

var mysql = require('mysql');

var con = mysql.createConnection({
    host: "localhost",
    user: "root",
    password: "root"
});
```

Abbildung 47: Javascriptcode createConnection
Quelle: eigene Darstellung

Die Verbindung zum AMQP Server sowie das Löschen eines älteren Eintrages ist in der Abbildung 48 ersichtlich.

```
function receiveTrigger(){
    var amqp = require('amqplib/callback_api');
    amqp.connect('amqp://localhost', function (error0, connection) {
        if (error0) {
            throw error0;
        }
        connection.createChannel(function (error1, channel) {
            if (error1) {
                throw error1;
            }

            var queue = "TriggerQueue";

            channel.assertQueue(queue, {
                durable: true
            });


            channel.consume(queue, function (msg) {
                con.query("DELETE FROM homework WHERE created < DATE_SUB(NOW(), INTERVAL 7 DAY)", function (err, result, fiel
                    if (err) {
                        return;
                    } else {
                    }
                });


            }, {
                noAck: true
            });

        });
    });
```

Abbildung 48: Javascriptcode receiveTrigger-Funktion
Quelle: eigene Darstellung

3.4.10 Mailserver.js

Dieser Service dient dazu einen Mailserver zu erstellen. Dazu wurde ein SMTP Server erstellt. Die Abbildung 49 zeigt die Konfiguration des Mailservers auf. Der Server hört auf den Port 1000. Dies ist in der Abbildung 50 dargestellt.

```
'use strict';


console.log("Gestartet")
const SMTPServer = require('smtp-server').SMTPServer;

// Setup server
const server = new SMTPServer({

    // log to console
    logger: false,

    // not required but nice-to-have
    banner: 'Welcome to My Awesome SMTP Server',

    // disable STARTTLS to allow authentication in clear text mode
    disabledCommands: ['STARTTLS'],

    // Accept messages up to 10 MB. This is a soft limit
    size: 10 * 1024 * 1024,

    // Setup authentication
    // Allow all usernames and passwords, no account checking
    onAuth(auth, session, callback) {
        return callback(null, {
            user: {
                username: auth.username
            }
        });
    },
```

Abbildung 49: Javascriptcode Mailserver
Quelle: eigene Darstellung

```
// start listening
server.listen(1000, 'localhost');
```

Abbildung 50: Javascriptcode server.listen
Quelle: eigene Darstellung

3.4.11 subscriber.js

Die subscriber.js-Datei enthält die Funktionalität, sich von einem externen Server mit dem AMQP zu verbinden und dort die Nachricht zu konsumieren. Ausserdem wurde «nodemailer» installiert, der es ermöglicht, eine Nachricht per E-Mail zu versenden. Die Abbildung 51 zeigt den Import der beiden Libraries auf und erstellt einen Transport, welcher die E-Mail Angaben des Absenders enthält.

```
const amqplib = require('amqplib/callback_api');
const nodemailer = require('nodemailer');



var transport = nodemailer.createTransport({
    service: 'gmail',
    auth: {
     user: 'amqptestwise@gmail.com',
     pass: ''
    }
 });
```

Abbildung 51: Javascriptcode subscriber.js nodemailer
Quelle: eigene Darstellung

Die Abbildung 52 zeigt auf, wie die Verbindung zu AMQP hergestellt wird.

```
// Create connection to AMQP server
amqplib.connect('amqp://localhost', (err, connection) => {
    if (err) {
        console.error(err.stack);
        return process.exit(1);
    }
    connection.createChannel((err, channel) => {
        if (err) {
            console.error(err.stack);
            return process.exit(1);
        }

        queue = "MailAnEltern";
        channel.assertQueue(queue, {
            durable: true
        }, err => {
            if (err) {
                console.error(err.stack);
                return process.exit(1);
            }
```

Abbildung 52: Javascriptcode AMQP Connection
Quelle: eigene Darstellung

Nachdem die Nachricht via AMQP konsumiert wurde, wird diese per Mail an die entsprechende E-Mail-Adresse versendet. Diesen Vorgang wird in Abbildung 53 dargestellt.

```
channel.consume(queue, data => {
    if (data === null) {
        return;
    }

    let message = JSON.parse(data.content.toString());

    transport.sendMail(message, (err, info) => {
        if (err) {
            console.error(err.stack);
            return channel.nack(data);
        }
        console.log('Nachricht ausgestellt: %s', info.messageId);
        channel.ack(data);
    });
});
```

Abbildung 53: Javascriptcode Nachricht ausgestellt
Quelle: eigene Darstellung

3.5 Installationsanleitung für Windows

Dieses Kapitel dokumentiert und unterstützt die Inbetriebnahme des Prototypen auf einer Windowsumgebung (localhost). Die wesentlichen Schritte werden in Kapitel 3.5.2 erklärt.

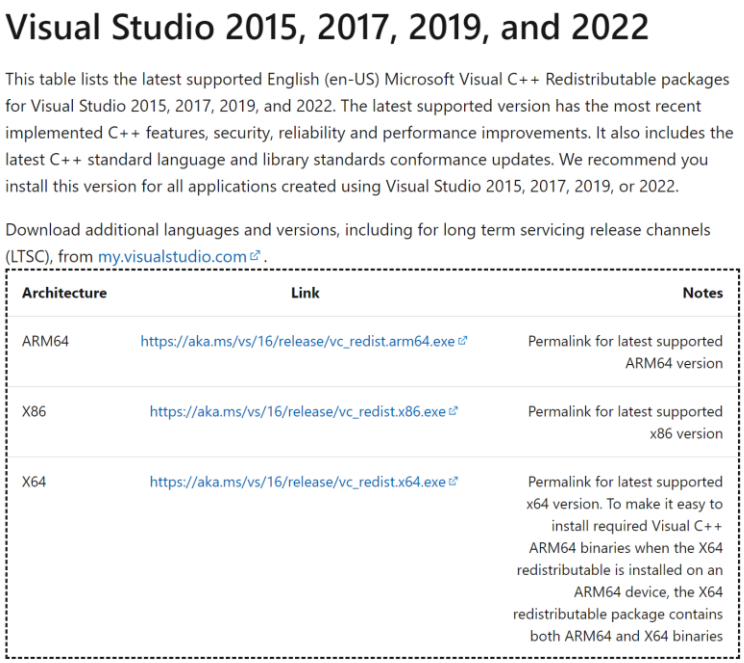
3.5.1 Voraussetzungen

Für die Installation des Prototyps werden folgende Dateien bzw. Programme benötigt:

- MySQL Workbench
- NodeJS
- hosting.rar
- Gmail Adresse

3.5.2 Installation

1. NodeJS (LTS) Windows Installer herunterladen und installieren: https://nodejs.org/en/download/
   ➔ Überall auf «next» klicken und dann auf «install»
2. MySQL Workbench installieren https://dev.mysql.com/downloads/workbench/
   ➔ Benötigt als Voraussetzung Microsoft Visual C++ Redistributable for Visual Studio 2019 (siehe Abbildung 54): https://docs.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist?view=msvc-170



Visual Studio 2015, 2017, 2019, and 2022

This table lists the latest supported English (en-US) Microsoft Visual C++ Redistributable packages for Visual Studio 2015, 2017, 2019, and 2022. The latest supported version has the most recent implemented C++ features, security, reliability and performance improvements. It also includes the latest C++ standard language and library standards conformance updates. We recommend you install this version for all applications created using Visual Studio 2015, 2017, 2019, or 2022.

Download additional languages and versions, including for long term servicing release channels (LTSC), from my.visualstudio.com.

Architecture	Link	Notes
ARM64	https://aka.ms/vs/16/release/vc_redist.arm64.exe	Permalink for latest supported ARM64 version
X86	https://aka.ms/vs/16/release/vc_redist.x86.exe	Permalink for latest supported x86 version
X64	https://aka.ms/vs/16/release/vc_redist.x64.exe	Permalink for latest supported x64 version. To make it easy to install required Visual C++ ARM64 binaries when the X64 redistributable is installed on an ARM64 device, the X64 redistributable package contains both ARM64 and X64 binaries

Abbildung 54: Visual Studio
Quelle: https://docs.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist?view=msvc-170

3. In der MySQL Workbench die «setup.sql» Datei importieren und ausführen. Diese erstellt das benötigte Schema, 2 User (Eli und Max mit dem gehashten Passwort 1) sowie ein Admin Account (Username: Admin, Passwort: 1).
4. Installieren von «Erlang» (benötigt für RabbitMQ): https://www.erlang.org/downloads
5. Installieren des RabbitMQ Servers: https://github.com/rabbitmq/rabbitmq-server/releases/download/v3.9.8/rabbitmq-server-3.9.8.exe
6. RabbitMQ Command Prompt starten (siehe Abbildung 55)

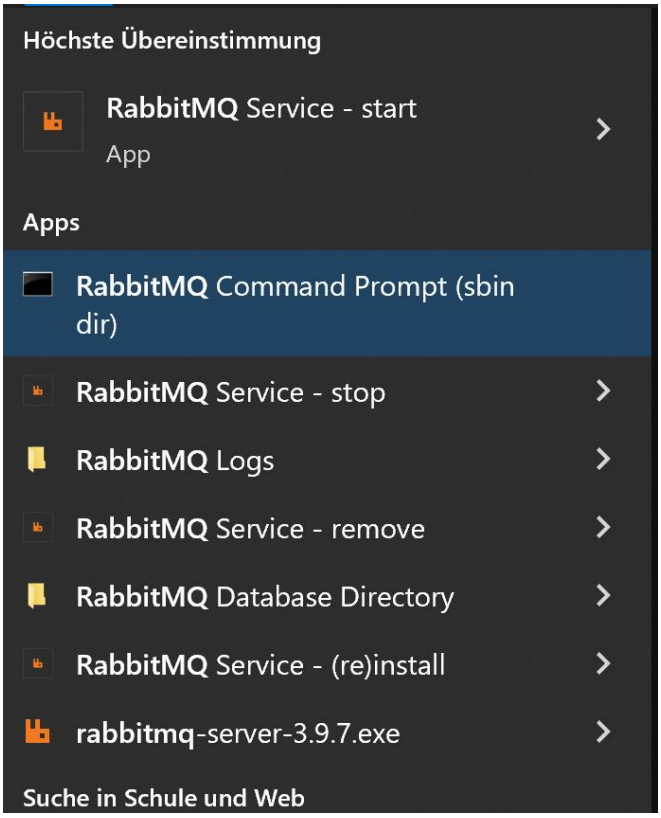


Abbildung 55: Suchleiste RabbitMQ
Quelle: eigene Darstellung

7. In der RabbitMQ Command Prompt folgende Befehlszeile absenden:
   «`rabbitmq-plugins enable rabbitmq_management`»
   Dies ermöglicht Rabbit MQ User Interface. Dieser wird dann unter **localhost:15672** laufen. (**Login**; Username: *guest*, Passwort: *guest*)
8. Entpacken der hosting.rar Datei
9. Windows Powershell öffnen (siehe Abbildung 56) und in das entpackte Verzeichnis wechseln (`cd <Verzeichnispfad>`)

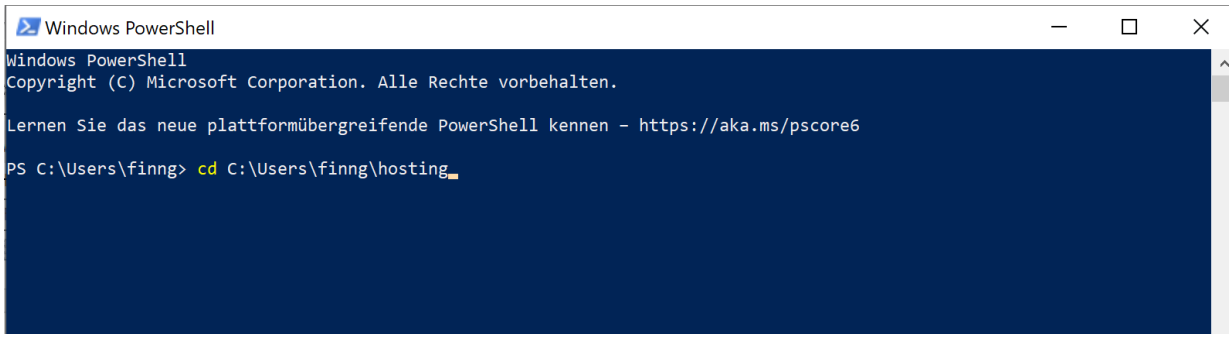


Abbildung 56: Powershell Terminal
Quelle: eigene Darstellung

10. Alle Server starten:
    a. Im Hosting Verzeichnis:
        - `node server.js`
    - Im Mailserver Verzeichnis (`cd /nodemailer`)
        - `node subscriber.js`
        - `node mailserver.js`
    - Im TriggerServer (`cd /trigger_server`)
        - `node triggerserver.js`
11. Nun kann die Webseite über «localhost:8082» sowie das User Interface von RabbitMQ über «localhost:15672» erreicht werden. Der Admin Accountzugang lautet wie folgt: RabbitMQ Username: *guest*, Passwort: *guest*
12. Um die E-Mail korrekt versenden zu können muss folgendes angepasst werden:
    a. Nodemailer Credentials ändern (siehe Abbildung 57)

```
var transport = nodemailer.createTransport({
    service: 'gmail',
    auth: {
      user: 'amqptestwise@gmail.com',
      pass: ''
    }
});
```

Abbildung 57: Javascriptcode Absendermailadresse
Quelle: eigene Darstellung

user: die gewünschte E-Mail Adresse des Absenders hinzufügen

pass: Das Passwort angeben

Wichtig: Im Gmail Account den Zugriff durch weniger sichere Apps zulassen (siehe Abbildung 58).

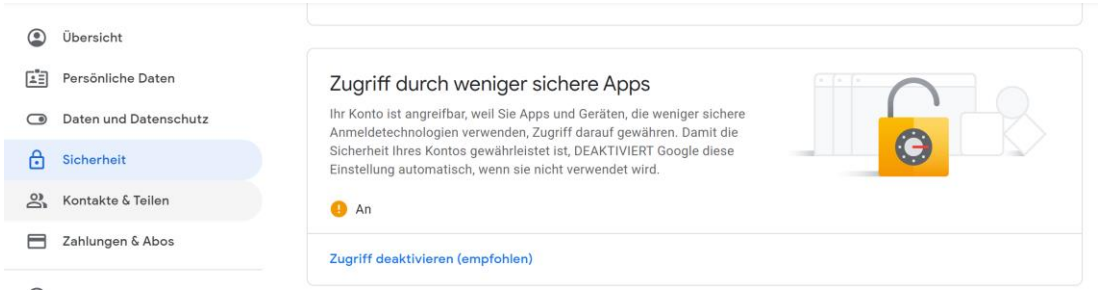


Abbildung 58: Übersicht Gmail-Menü
Quelle: eigene Darstellung

13. In der persistenceService.js-Datei in der Zeile 37 angeben, an wen die Message gesendet werden soll (siehe Abbildung 59).

```
37      amqppublisher.sendEmail('finn.graf@ost.ch', fach, title, messageone, date);
38  }
```

Abbildung 59: Javascriptcode Empfängermailadresse
Quelle: eigene Darstellung

14. **Ende der Dokumentation** – der Prototyp ist nun funktionsfähig und startbereit.

3.6 Ergebnisse

In diesem Kapitel werden die konkreten Ergebnisse des Prototypen visuell dargestellt. Nach der erfolgreichen Installation des Prototypen auf einer Windows-Umgebung sollten die Ergebnisse als Demo selbst ersichtlich sein.

In der Abbildung 60 können sich die Lehrpersonen und die Schüler/innen mit Ihrem Login anmelden. Mit dem Klicken des «Lets go»-Button werden die Login User auf die individuellen Ansichten weitergeleitet. In der Abbildung 61 wird die Sicht der Lehrperson dargestellt. Die Abbildung 62 zeigt die Sicht der Schülerinnen und Schüler.

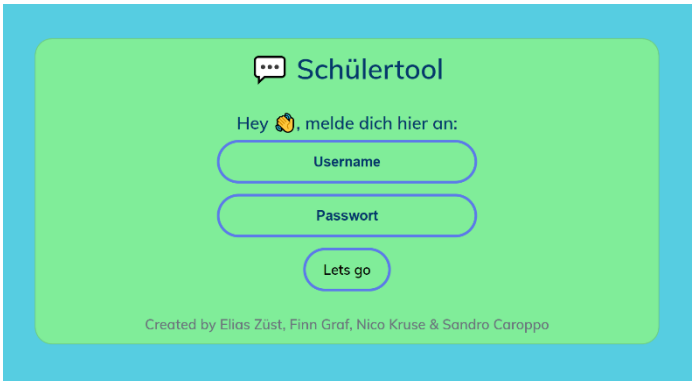


Abbildung 60: Login Ansicht für Schülerinnen und Schüler und Lehrpersonen
Quelle: eigene Darstellung

Die Abbildung 61 mit der Bezeichnung «My Homework» präsentiert die Ansicht der Schülerinnen und Schüler. Für jedes Schulfach wird eine eigene Farbe verwendet. Die Hausaufgaben werden fortlaufend hochgeschaltet und sind für alle gleichzeitig ersichtlich. Mit dem Button «open» können die genauen Inhalte angezeigt werden. Um zu verstehen, was in den genaueren Inhalten zu sehen ist, wird in der Abbildung 64 aufgezeigt.

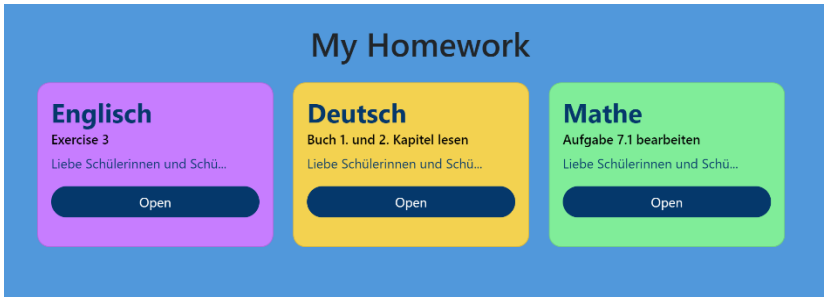


Abbildung 61: Schüler/innen Ansicht zu den Hausaufgaben
Quelle: eigene Darstellung

Damit Hausaufgaben ersichtlich sind, müssen diese von den Lehrpersonen erfasst werden. Die Abbildung 62 demonstriert diesen Ablauf. Im ersten Feld kann per Dropdown-Liste einen Wert aus der vorgegebenen Liste ausgewählt werden. Die Auswahlmöglichkeiten der Liste sind auf Mathe, Franz. Englisch, Deutsch und Informatik beschränkt. Im zweiten Feld kann ein passender Titel gewählt werden. Die Titel sollten dabei kurz und prägnant sein, denn für grössere Beschreibungen ist das dritte Feld geplant. Der Zeitpunkt der Abgabe einer Hausaufgabe kann mit dem Klicken auf den Kalender-Piktogramm in schwarz festgelegt werden. Um den Prozess abzuschliessen, muss noch per Auswahl definiert werden, ob die Eltern per E-Mail automatisch bezüglich der neuen Hausaufgaben informiert werden sollen.

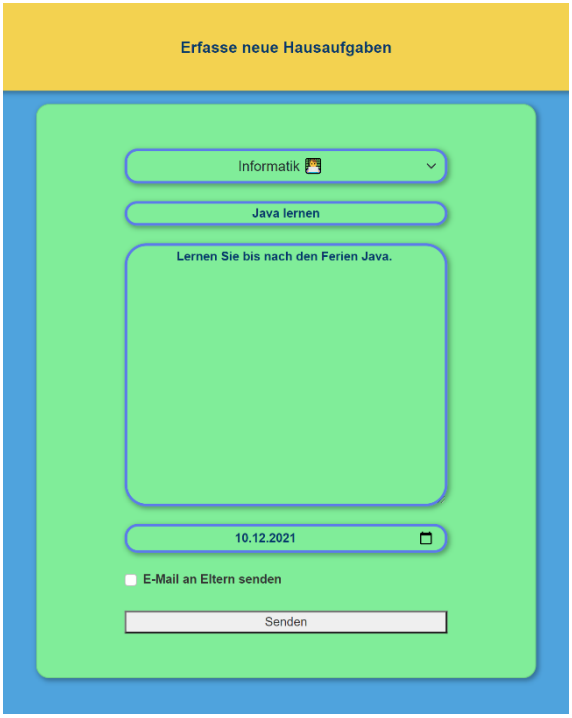


Abbildung 62: Hausaufgaben erfassen
Quelle: eigene Darstellung

Die Lehrpersonen haben zudem die Möglichkeit, erfasste Hausaufgaben auch wieder zu löschen. Dies kann mit einem einfachen Klick auf den dunkelblauen Balken erledigt werden. Die Abbildung 63 verdeutlicht dies. Das Löschen einer Hausaufgabe wird dabei automatisch synchronisiert.

Abbildung 63: Hausaufgaben wieder löschen
Quelle: eigene Darstellung

Damit Schüler/innen die Inhalte von den Hausaufgaben detaillierter anschauen können, müssen diese den «open» Button anklicken. Geschieht dies, kommen sie auf die in Abbildung 64 angefügte Ansicht. Zum einem ist eine detaillierte Beschreibung der Hausaufgaben vorhanden und zum anderen eine automatisch geöffnete Chat Funktion. Falls Fragen oder Unklarheiten zu einem Auftrag vorhanden sind, können sich die Schülerinnen und Schüler untereinander austauschen. Der Chat-Verlauf wird dabei automatisch gespeichert. Unten rechts sind weitere Informationen wie Abgabedatum und CC an Eltern erfasst.

Abbildung 64: Ansicht detaillierten Inhalte von Hausaufgaben
Quelle: eigene Darstellung

4 Schlussfolgerung

Abschliessend wird in diesem Kapitel auf die Zielerreichung der Projektarbeit eingegangen und diskutiert. Hierzu werden die in Kapitel 1.2 aufgelisteten Projektziele in Tabelle 5 nochmals herangezogen. Diese Ziele dienten in dieser Arbeit als Wegweiser, wodurch sich die Projektgruppe auf das wesentliche fokussieren konnte.

Der Schlussbericht deckt technologische Aspekte von Message Oriented Middleware anhand bestehender Literatur in Breite und Tiefe ab. Der Einsatz von MOM wird ebenfalls aufgezeigt.	**teilweise erreicht**
Die Marktanalyse verschafft einen Überblick über bestehende MOM-Produkte und zeigt die besten Anbieter an.	**teilweise erreicht**
Der erstellte Prototyp ist lauffähig und deckt die Bedürfnisse des Anwendungsfalles ab.	**erreicht**
Der erstellte Prototyp lässt sich als Vorlage für grössere Anwendungen nutzen.	**erreicht**

Tabelle 5: Zielerreichung
Quelle: eigene Darstellung

Wie in Tabelle 5 ersichtlich ist, wurden zwei von vier gesetzten Zielen in dieser Arbeit teilweise erreicht. Dies lag hauptsächlich daran, dass im Bereich der Message Oriented Middleware wenig bis keine aktuellen Literaturwerke vorhanden sind, die in diese Arbeit integriert werden konnten. Technologische Aspekte wurden daher mit Informationen abgedeckt, die direkt von den Anbietern von solchen MOM-Systemen stammen. Das gleiche Prinzip gilt für die durchgeführte Marktanalyse. Es wurden keine Zahlen und Statistiken zu Verkäufen oder Downloads gefunden, die dabei behilflich sein konnten. Somit bediente sich die Projektgruppe aus bestehenden Reviews und Bewertungen von drei IT-Communities. Obwohl also MOM-Systeme breitflächig in der Informatik Verwendung finden, so kann das gleiche in der aktuellen Literatur der letzten zehn Jahre nicht behauptet werden. Es besteht somit Nachholbedarf, um das Thema auf den neuesten Stand zu bringen und zu dokumentieren.

Im Vergleich gelang es der Projektgruppe ein Prototyp mit integrierter Middleware zu implementieren und für ein praxisnaher Anwendungsfall konkret einzusetzen. Der Vorteil von Middleware liegt nicht nur in der zusätzlichen Architekturschicht, die als Sicherheit dient, sondern auch in der Skalierbarkeit. Somit liesse sich das erstellte Prototyp auf grössere Anwendungsfälle wie beispielweise Hochschulen skalieren und anpassen. Durch den vermehrten Einsatz von solchen Systemen in der IT, besteht dementsprechend viel Hilfe seitens der Coding-Community von verschiedenen Plattformen. Dadurch konnte die Projektgruppe die Ziele betreffend Prototyp erreichen.

Quellenverzeichnis

ActiveMQ. (2021). *Stomp*. Abgerufen am 30.10.2021 von https://activemq.apache.org/stomp

Algolia. (2021). *Using the Bayesian Average in Ranking*. Abgerufen am 25.10.2021 von https://www.algolia.com/doc/guides/solutions/ecommerce/relevance-optimization/tutorials/bayesian-average/

Amazon AWS. (ohne Datum a). *Nachrichtenwarteschlangen*. Abgerufen am 23.10.2021 von https://aws.amazon.com/de/message-queue/

Amazon AWS. (ohne Datum b). *Amazon Simple Queue Service*. Abgerufen am 26.10.2021 von https://aws.amazon.com/de/amazon-mq/

Amazon AWS. (ohne Datum c). *Amazon Simple Queue Service*. Abgerufen am 26.10.2021 von https://aws.amazon.com/de/sns/

Amazon AWS. (ohne Datum d). *Amazon Simple Queue Service*. Abgerufen am 26.10.2021 von https://aws.amazon.com/de/sqs/

Amqp.org. (2021). *AMQP is an open Internet (or "wire") Protocol standard for message-queuing communications*. Abgerufen am 02.11.2021 von https://www.amqp.org/product/overview

Apache Kafka. (2017). *Apache Kafka*. Abgerufen am 26.10.2021 von https://kafka.apache.org/

AsyncAPI. (ohne Datum a). *Introduction*. Abgerufen am 03.11.2021 von https://www.asyncapi.com/docs/getting-started

AsyncAPI. (ohne Datum b). *Event-Driven Architectures*. Abgerufen am 03.11.2021 von https://www.asyncapi.com/docs/getting-started/event-driven-architectures

AsyncAPI. (ohne Datum c). *Coming from OpenAPI*. Abgerufen am 03.11.2021 von https://www.asyncapi.com/docs/getting-started/coming-from-openapi

Bruns, R. & Dunkel, J. (2010). *Event-Driven Architecture, Softwarearchitektur für ereignisgesteuerte Geschäftsprozesse.* Berling Heidelberg: Springer-Verlag

CloudAMQP. (2021a). *Stomp*. Abgerufen am 30.10.2021 von https://www.cloudamqp.com/docs/stomp.html

CloudAMQP. (2021b). *About CloudAMQP*. Abgerufen am 26.10.2021 von https://www.cloudamqp.com/about_us.html

Dean, T. (2009), *Network+ Guide to Networks* (5. Aufl.). Hampshire: Cengage Learning.

ETH. (ohne Datum). *Publish/Subscribe* [Vorlesung]. Abgerufen am 23.10.2021 von http://se.inf.ethz.ch/old/teaching/ss2007/0284/Slides/PublishSubscribe.pdf

Google Cloud. (2021, 9. November). *Was ist Pub/Sub?* Abgerufen von https://cloud.google.com/pubsub/docs/overview

IBM. (2021a, 04. November). *Securing web services applications at the transport level*. Abgerufen am 06.11.2021 von https://www.ibm.com/docs/en/was/9.0.5?topic=security-securing-web-services-applications-transport-level

IBM. (2021b, 28. Oktober). *Securing AMQP Clients*. Abgerufen am 06.11.2021 von https://www.ibm.com/docs/en/ibm-mq/9.1?topic=securing-amqp-clients

IBM. (2021c, 28. Oktober). *Features and functions of Advanced Message Security*. Abgerufen am 06.11.2021 von https://www.ibm.com/docs/en/ibm-mq/9.1?topic=security-features-functions-advanced-message

IBM. (2020, 23. Januar). *Message Broker*. Abgerufen am 26.10.2021 von https://www.ibm.com/cloud/learn/message-brokers

IBM. (2018, 30. Oktober). *Hauptmerkmale und Vorteile des Message-Queuing*. Abgerufen am 23.10.2021 von https://www.ibm.com/docs/de/ibm-mq/7.5?topic=ssfksj-7-5-0-com-ibm-mq-pro-doc-q002630--htm

IBM. (ohne Datum). *IBM MQ*. Abgerufen am 26.10.2021 von https://www.ibm.com/products/mq?lnk=STW_US_STESCH&lnk2=trial_MQ&pexp=def&psrc=none&mhsrc=ibmsearch_a&mhq=ibm%20mq

Jakarta EE. (ohne Datum). *Jakarta EE, Building an open-source ecosystem for cloud native enterprise Java.* Abgerufen von https://jakarta.ee/about/

Johansson, L. (2019, 21. November). *What is AMQP and why is it used in RabbitMQ?* CloudAMQP. Abgerufen von https://www.cloudamqp.com/blog/what-is-amqp-and-why-is-it-used-in-rabbitmq.html

Lightbend. (2021). *Message Driven vs. Event Driven.* Abgerufen am 22.10.2021 von https://developer.lightbend.com/docs/akka-platform-guide/concepts/message-driven-event-driven.html

Masurel, P. (2013, 17. März). *Of bayesian average and star ratings*. Abgerufen am 25.10.2021 von https://fulmicoton.com/posts/bayesian_rating/

Microsoft. (2021). *Ereignisgesteuerter Architekturstil.* Abgerufen am 22.10.2021 von https://docs.microsoft.com/de-de/azure/architecture/guide/architecture-styles/event-driven

Microsoft Azure. (2021a). *Service Bus*. Abgerufen am 27.11.2021 von https://azure.microsoft.com/de-de/services/service-bus/

Microsoft Azure. (2021b). *Queue Storage*. Abgerufen am 27.11.2021 von https://azure.microsoft.com/de-de/services/storage/queues/

Microsoft Azure. (2021c). *Event grid*. Abgerufen am 27.11.2021 von https://azure.microsoft.com/de-de/services/event-grid/

Microsoft Azure. (2021d). *Event Hubs*. Abgerufen am 27.11.2021 von https://azure.microsoft.com/de-de/services/event-hubs/

Mqtt.org. (2020). *MQTT: The Standard for IoT Messaging*. Abgerufen am 28.10.2021 von https://mqtt.org/

Mullet, D. (2000). *Managing IMAP*. Sebastopol: O'Reilly Media.

Neurodump. (2011, 25. August). *Was ist SASL? (Simple Authentication and Security Layer)*. Abgerufen am 06.11.2021 von https://neurodump.cmplx.de/2011/08/25/was-ist-sasl-simple-authentication-and-security-layer/

Oracle. (2010a). *Message-Oriented Middleware (MOM)*. Abgerufen am 22.10.2021 von https://docs.oracle.com/cd/E19340-01/820-6424/aeraq/index.html

Oracle. (2010b). *Point-To-Point Messaging*. Abgerufen am 23.10.2021 von https://docs.oracle.com/cd/E19316-01/820-6424/aerbj/index.html

Postman. (2021a). *What is Postman?* Abgerufen am 03.11.2021 von https://www.postman.com/product/what-is-postman/

Postman. (2021b). *Postman JavaScript reference*. Abgerufen am 03.11.2021 von https://learning.postman.com/docs/writing-scripts/script-references/postman-sandbox-api-reference/

RabbitMQ. (2021a). *TLS Support*. Abgerufen am 06.11.2021 von https://www.rabbitmq.com/ssl.html

RabbitMQ. (2021b). *RabbitMQ is the most widely deployed open source message broker.* Abgerufen am 26.10.2021 von https://www.rabbitmq.com/#getstarted

Reich, Ch. (ohne Datum). *Message Oriented Middleware (MOM) Java Message Service (JMS)* [Vorlesung]. Hochschule Furtwangen. Abgerufen am 23.10.2021 von https://docplayer.org/1307927-Message-oriented-middleware-mom-java-message-service-jms.html

Rhoton, J. (1999). *Programmer's Guide to Internet Mail: SMTP, POP, IMAP and LDAP.* Amsterdam: Elsevier.

Sandoval, K. (2019, 17. Januar). *7 Protocols Good for Documenting With AsyncAPI.* Nordicapis. Abgerufen von https://nordicapis.com/7-protocols-good-for-documenting-with-asyncapi/

Testautomatisierung.org. (ohne Datum). *Postman (API)*. Abgerufen am 03.11.2021 von https://www.testautomatisierung.org/lexikon/postman-api/

Tibco. (2021). *Was ist eine ereignisgesteuerte Architektur?* Abgerufen am 22.10.2021 von https://www.tibco.com/de/reference-center/what-is-event-driven-architecture

Anhänge

Anhang A: Admin CSS

```
.background{
    background-color: #ECECEC;;

    -webkit-animation: bgcolorchange 45s infinite; /* Chrome, Safari, Opera */
    animation: 45s infinite bgcolorchange;
  }
  @keyframes bgcolorchange {
    0% {
      background-color: #5390d9;
    }
    33% {
      background-color: #4ea8de;
    }
    66% {
      background-color: #56cfe1;
    }
    100% {
      background-color: #72efdd;
    }
  }
  /* Chrome, Safari, Opera */
   @-webkit-keyframes bgcolorchange {
        0%   {background: #80ED99;}
        25%  {background: #F3D250;}
        75%  {background: #F78888;}
        100% {background: #5C7AEA;}

}

.spacer500{
    padding-top: 250px;
  }

  .spacer100{
    padding-top: 100px;
  }

  @media (min-width: 992px) and (max-width : 1920px){
    .ownMarginClass{
      padding-left: 300px;
      padding-right: 300px;
    }
  }

  .titleHomework{
    text-align: center;
    font-weight: bold;
    color: #05386B;
    font-size: 18px;
  }
```

```
.containerown{
  width: 100%;
  height: 100px;
  background-color: #F3D250;
  box-shadow: 2px 2px 5px 1px rgba(0,0,0,0.3);
}

.ownSettingsButton{
  background: transparent;
  border: none;
  font-size: 40px;
}

.InputStyle {
  border-radius: 25px;
  border: solid 3px #5C7AEA;
  background: transparent;
  text-align: center;
  font-weight: bold;
  font-family:Arial;
  color: #05386B;
  box-shadow: 2px 2px 5px 1px rgba(0,0,0,0.3);

}


.InputStyle:focus{
  color: white;
  font-weight: bold;
  background-color: #5C7AEA;
  box-shadow: 2px 2px 5px 1px rgba(0,0,0,0.3);
  text-align: left;
  padding: 6px 10px;
}

.InputStyle::placeholder { /* Chrome, Firefox, Opera, Safari 10.1+ */
  color:#05386B;
  opacity: 1; /* Firefox */
}

.InputStyle:focus::placeholder { /* Chrome, Firefox, Opera, Safari 10.1+ */
  color:transparent;

  opacity: 1; /* Firefox */
}
```

```
  .InputStyleMessage {
    border-radius: 25px;
    border: solid 3px #5C7AEA;
    background: transparent;
    height: 300px;
    text-align: left;
    font-weight: bold;
    font-family:Arial;
    color: #05386B;
    box-shadow: 2px 2px 5px 1px rgba(0,0,0,0.3);
    text-align: center;
  }


  .InputStyleMessage:focus{
    color: white;
    font-weight: bold;
    background-color: #5C7AEA;
    box-shadow: 2px 2px 5px 1px rgba(0,0,0,0.3);
    text-align: left;
    padding: 10px 10px;
  }

  .InputStyleMessage::placeholder { /* Chrome, Firefox, Opera, Safari 10.1+ */
    color:#05386B;
    opacity: 1; /* Firefox */
  }

  .InputStyleMessage:focus::placeholder { /* Chrome, Firefox, Opera, Safari
10.1+ */
    color:transparent;

    opacity: 1; /* Firefox */
  }

  .card{
    border-radius: 15px;
    padding-left: 100px;
    padding-right: 100px;
    padding-top: 50px;
    padding-bottom: 50px;
    box-shadow: 2px 2px 5px 1px rgba(0,0,0,0.3);
    background-color: #80ED99;
    }
```

```
.dateInput{
  border: solid 3px #5C7AEA;
  border-radius: 15px;
  box-shadow: 2px 2px 5px 1px rgba(0,0,0,0.3);
  color:#05386B;
  font-weight: bold;
  padding: 3px 10px;
  text-align: center;
  background-color: transparent;
}

.dateInput::placeholder{
  color:#05386B;
  font-weight: bold;
}


  /*SELECT schön darstellen */
  select {
    -moz-appearance: none;
    -webkit-appearance: none;
    appearance: none;
    border: none;
    -moz-padding-start: calc(10px - 3px);
    width: 100%;
    height: 40px;
    -moz-padding-start: calc(10px - 3px);
    padding-left: 10px;
    font-family: 'Open Sans', sans-serif;
    font-size: 16px;
    box-shadow: 2px 2px 5px 1px rgba(0,0,0,0.3);
    border-radius: 15px;
    border: solid 3px #5C7AEA;
    outline: none;
    cursor: pointer;
    text-align: center;
    background-color: transparent;

  }
  select::-ms-expand {
    display: none;
  }
  select option {
    color: #666;
  }
  select:focus::-ms-value {
    background-color: transparent;
  }
```

```
.select-wrapper {
  /* ... */
  position: relative;
}
.select-wrapper::before {
  font-family: fontello;
  content: "\f107";
  font-size: 20px;
  position: absolute;
  right: 15px;
  top: 10px;
  color: #fff;
  pointer-events: none;
}
```

Anhang B: Admin HTML

```
<head>
    <link rel="stylesheet"
href="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/css/bootstrap.min.css"
/>

</head>
<body>
    <div class="background">
      <div class="containerown">
        <div class="h-100 row align-items-center">
      <div class="titleHomework">
        Erfasse neue Hausaufgaben
      </div>
    </div>
      </div>


          <div class="h-100 row align-items-center">
            <div class="container ownMarginClass mt-4 ownClass">
              <div class="wow fadeInUp col-lg-auto col-md-auto col-sm-auto
col-xs-auto">

          <div class="card">
            <select class="form-select" name="fach" id="fachselector">
              <option value="Mathe">Mathe </option>
              <option value="Deutsch">Deutsch </option>
              <option value="Französisch">Französisch </option>
              <option value="Englisch">Englisch </option>
              <option value="Informatik">Informatik </option>
           </select><br>
          <input type="text" id="title" class="InputStyle"
placeholder="Titel"> <br>
          <textarea type="text" id="message" class="InputStyleMessage"
placeholder="Nachricht"></textarea><br>
          <input type="text" onfocus="(this.type = 'date')" id="date"
class="dateInput" placeholder="Abgabedatum eintragen"> <br>
          <div class="form-check">
          <input class="form-check-input" type="checkbox" id="parentsCC">
          <label for="parentsCC" class="form-check-label">E-Mail an Eltern
senden</label></div><br>
          <button class="button" (click)="msgSend()"> Senden </button>
        </div>


            </div>

     </div>
     </div>
```

```
    <div class="spacer100"></div>
    <div class="spacer100"></div>
    <div class="containerown">
     <div class="h-100 row align-items-center">
   <div class="titleHomework">
     Erfasste Hausaufgaben
   </div>
  </div>
    </div>


   <div class="spacer100"></div>

<div class="container">

   <div class="h-20">
     <div class="container mt-4">
         <div class="row align-items-center" id="dataID">


       <template #createdContainer>
       </template>

       </div>
       </div>
       </div>

  </div>
  </div>

</body>
```

Anhang C: Admin TypeScript

```
import { Component, OnInit, ViewChild, ViewContainerRef,
ComponentFactoryResolver } from '@angular/core';
import { CreatedHomeworkTemplateComponent } from '../created-homework-
template/created-homework-template.component';
import { WebSocketService } from '../web-socket.service';
import Swal from 'sweetalert2';

@Component({
  selector: 'app-admin',
  templateUrl: './admin.component.html',
  styleUrls: ['./admin.component.css']
})
export class AdminComponent implements OnInit {
  @ViewChild('createdContainer', { read: ViewContainerRef })
  entry!: ViewContainerRef;
  index: any = 0;
  newClass: any;
  newID: any;
  array: any = []

  constructor(private webSocketService: WebSocketService, private resolver:
ComponentFactoryResolver) { }

  createComponent(fach: any, title: any, message: any, date: any, parentsCC:
any, messageLong: any, id: any) {
    // @ts-ignore: Object is possibly 'null'.
    var text = fach + title + message + date + parentsCC;

    if (this.array.includes(text) == false) {
      this.array.push(text);

      const factory =
this.resolver.resolveComponentFactory(CreatedHomeworkTemplateComponent);
      // @ts-ignore: Object is possibly 'null'.
      const componentRef = this.entry.createComponent(factory, 0);

      componentRef.instance.fach = fach;
      componentRef.instance.title = title;
      componentRef.instance.message = message;
      componentRef.instance.date = date;
      componentRef.instance.parentsCC = parentsCC;
      componentRef.instance.id = id;
      componentRef.instance.messageLong = messageLong;

      this.newClass = document.getElementsByClassName('identifyClass')[0];
      this.newID = document.getElementsByTagName('app-created-homework-
template')[0];
      this.newID.setAttribute('id', id);

      switch (fach) {
```

```
        case "Mathe": {
          this.newClass.classList.add('cardOne');
          break;
        }
        case "Deutsch": {
          this.newClass.classList.add('cardTwo');
          break;
        }
        case "Französisch": {
          this.newClass.classList.add('cardThree');
          break;
        }

        case "Englisch": {
          this.newClass.classList.add('cardFour');
          break;
        }

        case "Informatik": {
          this.newClass.classList.add('cardFive');
          break;
        }
        default: {
          this.newClass.classList.add('cardThree')
          break;
        }
      }

    }

}

ngOnInit(): void {
  this.webSocketService.listen('getData').subscribe((data: any) => {


    var newMessage = JSON.parse(data);
    var id = newMessage['id'];
    var fach = newMessage['fach'];
    var title = newMessage['title'];
    var message = newMessage['message'];
    if(message.length > 27){
    var messageShort = message.substr(0, 27) + "...";
    } else{
    messageShort = message;
    }
    var date = newMessage['date'];
    var parentsCC = newMessage['parentsCC'];
```

```
      this.createComponent(fach, title, messageShort, date, parentsCC,
message, id);
      var created = document.getElementsByTagName('app-created-homework-
template').length;
      var i = 0;
      while (i < created) {
        var newdoc = document.getElementsByTagName('app-created-homework-
template')[i];
        newdoc.setAttribute('class', 'col-auto');
        i++;
      }

    });
  }

msgSend(){
  // @ts-ignore: Object is possibly 'null'.
  var fach = document.getElementById("fachselector").value;
  // @ts-ignore: Object is possibly 'null'.
  var title = document.getElementById('title').value;
  // @ts-ignore: Object is possibly 'null'.
  var message = document.getElementById('message').value;
    // @ts-ignore: Object is possibly 'null'.
    var date = document.getElementById('date').value;
      // @ts-ignore: Object is possibly 'null'.
  if(document.getElementById('parentsCC').checked == true){
    // @ts-ignore: Object is possibly 'null'.
  var parentsCC = "yes";
  }else{
    var parentsCC = "no";
  }

  var current_date = new Date()
  var c_date = current_date.toISOString().split('T')[0]

  var creation_date = c_date;

  this.webSocketService.emit6('newHomework', fach, title, message, date,
creation_date, parentsCC);

  Swal.fire({
    icon: 'success',
    title: 'Dein Eintrag wurde erstellt. Die Schüler haben ihn erhalten!',
    showConfirmButton: true
  })
```

```
    // @ts-ignore: Object is possibly 'null'.
    var fach = document.getElementById("fachselector").value = null;
    // @ts-ignore: Object is possibly 'null'.
    var title = document.getElementById('title').value = null;
    // @ts-ignore: Object is possibly 'null'.
    var message = document.getElementById('message').value = null;
      // @ts-ignore: Object is possibly 'null'.
      var date = document.getElementById('date').value = null;
        // @ts-ignore: Object is possibly 'null'.
    document.getElementById('parentsCC').checked == null;
}

}
```

Anhang D: Login CSS

```css
@import url('https://fonts.googleapis.com/css?family=Damion|Muli:400,600');

body {
  width: 100%; height: 100vh; overflow: hidden;
}

.background{
  background-color: #ECECEC;
  padding-bottom: 373px;

  -webkit-animation: bgcolorchange 45s infinite; /* Chrome, Safari, Opera */
  animation: 45s infinite bgcolorchange;
}

@keyframes bgcolorchange {
  0% {
    background-color: #5390d9;
  }
  33% {
    background-color: #4ea8de;
  }
  66% {
    background-color: #56cfe1;
  }
  100% {
    background-color: #72efdd;
  }
}

/* Chrome, Safari, Opera */
 @-webkit-keyframes bgcolorchange {
      0%   {background: #80ED99;}
      25%  {background: #F3D250;}
      75%  {background: #F78888;}
      100% {background: #5C7AEA;}

}
.spacer500{
  padding-top: 250px;

}

.ownCardHeader{
  color: #05386B;
  font-weight: bold;
  padding: 10px 0px;
  font-size: 32px;

}
```

```
.card-title{
  color: #05386B;
  font-weight: bold;
}

.ownCardFooter{
  color: #05386B;
  font-weight: bold;
  border: black;
  padding: 10px 0px;
}

.card{
  border-radius: 20px;
  margin: 0px 0px;
  background-color: #80ED99;
}
.card:hover{
  transform: scale(1.05);

}

body{
  font-family: 'Muli', sans-serif;
  background:
url("https://www.toptal.com/designers/subtlepatterns/patterns/geometry2.png");
}

h2 {
    font-family: 'Damion', cursive;
    font-weight: 400;
    color: #4caf50;
    font-size: 35px;
    text-align: center;
    position: relative;
}

h2:before {
    position: absolute;
    content: '';
    width: 100%;
    left: 0;
    top: 22px;
    background: #b5cab6;
    height: 1px;
}
```

```
h2 i {
    font-style: normal;
    background: #fff;
    position: relative;
    padding: 10px;
}

:focus{outline: none;}

.InputStyle {
  border-radius: 25px;
  border: solid 3px #5C7AEA;
  background: transparent;
  width: 300px;
  padding: 10px 20px;
  margin-bottom: 12px;
  text-align: center;
  font-weight: bold;
  font-family:Arial;
  color: #05386B;
}


.InputStyle:focus{
  color: white;
  font-weight: bold;
  text-align: center;
  background-color: #5C7AEA;
}

.InputStyle::placeholder { /* Chrome, Firefox, Opera, Safari 10.1+ */
  color:#05386B;
  opacity: 1; /* Firefox */
}

.InputStyle:focus::placeholder { /* Chrome, Firefox, Opera, Safari 10.1+ */
  color:transparent;

  opacity: 1; /* Firefox */
}
.ownButton{
  border-radius: 25px;
  border: solid 3px #5C7AEA;
  padding: 10px 20px;
  background: transparent;
  font-weight: bold;
}
```

```
.ownButton:hover{
  transform: scale(1.1);
  border-radius: 25px;
  border: solid 3px #5C7AEA;
  padding: 10px 20px;
  background: #5C7AEA;
  color: white;
  font-weight: bold;
}

.ownSettingsButton{
  background: transparent;
  border: none;
  font-size: 40px;
}

@media (min-width: 992px) and (max-width : 1920px){
  .container{
    padding-left: 300px;
    padding-right: 300px;
  }
}
```

Anhang E: Login HTML

```
<header>
  <script src="../../../src/assets/custom.js"></script>
  <script src="../../../src/assets/send.js"></script>
  <script src="../../../src/assets/receive.js"></script>
  <script type="text/javascript" src="../sendRabbitMQ.js"></script>

</header>
<body>
  <div class="body">
    <div class="background">
      <div class="spacer500"></div>
        <div class="container">
          <div class="card text-center wow fadeInUp col-lg-auto col-md-auto
col-sm-auto col-xs-auto">
            <div class="ownCardHeader wow fadeInUp">
              💬 Schülertool
            </div>
            <div class="card-body">
              <h5 class="card-title wow fadeInUp">Hey 👏, melde dich hier
an:</h5>
              <div id="login" class="">
                <form>
                  <input class="InputStyle wow fadeInUp"
placeholder="Username" type="text" id="usernameInput" autofocus="true">
                  <br>
                  <input class="InputStyle wow fadeInUp"
placeholder="Passwort" type="password" id="passwordInput" autofocus="false">
                  <br>
                  <button type="button" class="ownButton wow fadeInUp"
(click)="login()">Lets go</button>
                </form>
              </div>
            </div>
          <div class="ownCardFooter text-muted wow fadeInUp">
            Created by Elias Züst, Finn Graf, Nico Kruse & Sandro Caroppo
          </div>
      </div>
    </div>
  </div>
</div>
</body>
<footer>
</footer>
```

Anhang F: Login TypeScript

```
import { Component, OnInit } from '@angular/core';
import { WebSocketService } from '../web-socket.service';
import * as sha512 from 'js-sha512';
import { Router } from '@angular/router';
import Swal from 'sweetalert2';
import { NgwWowService } from 'ngx-wow';
import {User} from '../user'

var myUser: any;
@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {
    public meinUsername = myUser;


    getUser(){
      return myUser.getUsername();
    }

    constructor(private webSocketService: WebSocketService, private router:
Router, private wowService: NgwWowService) {
      this.wowService.init();
    }

  ngOnInit(): void {
    this.webSocketService.listen('loginAuthorized').subscribe((data: any) => {
      var username = data;
      myUser = new User(username);
      Swal.fire({
        icon: 'success',
        title: 'Du hast dich als ' + data + ' erfolgreich eingeloggt!',
        showConfirmButton: false,
        timer: 1750
      })

      if(username == 'Admin'){
        this.router.navigate(['/admin']);
      }else{
      this.router.navigate(['/chat', username]);
      }
    });
```

```
    this.webSocketService.listen('loginNotAuthorized').subscribe((data: any)
=> {
      var username: string = data;
      Swal.fire({
        icon: 'error',
        title: 'Oops...',
        text: 'Dein Login für den User ' + data + " ist fehlgeschlagen",
        showCancelButton: false,
        timer: 1500
      })

    });



  }

  login() {
    // @ts-ignore: Object is possibly 'null'.
    var usernamemsg = document.getElementById('usernameInput').value;
    // @ts-ignore: Object is possibly 'null'.
    var passwordmsg = document.getElementById('passwordInput').value;
    this.webSocketService.emit('username', usernamemsg);

    var hashedpassword = sha512.sha512(passwordmsg);

    this.webSocketService.emit2('mysqldata', usernamemsg, hashedpassword);
    //this.router.navigate(['/chat']);
  }


}
```

Anhang G: User CSS

```
.btn-primary{
    margin-left: 10px;

}

.inbox-div{
    margin-left: 30%;
}


.spacer{
    margin-left: 12px;
}

.heighter{
    margin-top: 120px;
}

#messageFeed {
    height: calc(300px); /* space for fixed navbar (48px), "main" title
(71px), footer (53px) */
    overflow-y: scroll;
    margin-bottom: 30px;
  }

#messages {
    background-color: transparent;
    list-style-type: none;
}

#individualMessage{
    margin-top: 12px;
    margin-bottom: 12px;
    padding-top: 10px;
    padding-bottom: 10px;
    border-radius: 25px;
    padding-left: 30px;
    color: #05386B;
    font-weight: bold;

    background-color: #80ED99;
}
```

```
  #messages footer {
    position: fixed;
    bottom: 0;
    left: 0;
    width: 100%; /* 100% of page width */
    padding: 0 15px 15px 15px; /* standard edge margin (15px) */
    height: 53px; /* space for input height (38px) + bottom padding (15px) */
    background-color: #fff;

  }

  .ownClass{
    margin-left: 17%;

}

/*Bootstrap 5 hinzugefügt*/
.container{
  width: 80%;
}

  .spacer500{
    padding-top: 250px;

  }
  .spacer100{
    padding-top: 50px;
  }
```

```
  .background{
    height: 100vh;
    padding-bottom: 350px;
      -webkit-animation: bgcolorchange 45s infinite; /* Chrome, Safari, Opera
*/
      animation: 45s infinite bgcolorchange;
    }
    @keyframes bgcolorchange {
      0% {
        background-color: #5390d9;
      }
      33% {
        background-color: #4ea8de;
      }
      66% {
        background-color: #56cfe1;
      }
      100% {
        background-color: #72efdd;
      }
    }
    /* Chrome, Safari, Opera */
     @-webkit-keyframes bgcolorchange {
          0%   {background: #80ED99;}
          25%  {background: #F3D250;}
          75%  {background: #F78888;}
          100% {background: #5C7AEA;}

  }

  ::-webkit-scrollbar {
    width: 6px;
}

/* Track */
::-webkit-scrollbar-track {
  box-shadow: inset 0 0 8px #80ED99;
    -webkit-box-shadow: inset 0 0 8px #80ED99;
    -webkit-border-radius: 10px;
    border-radius: 10px;
}

/* Handle */
::-webkit-scrollbar-thumb {
    -webkit-border-radius: 10px;
    border-radius: 10px;
    background: #3E00FF;
    -webkit-box-shadow: inset 0 0 6px #3E00FF;
    box-shadow: inset 0 0 6px #3E00FF;;
}
```

```
::-webkit-scrollbar-thumb:window-inactive {
    background: black;
}

  .cardOne{
      background-color: #80ED99;
  }

  .cardTwo{
    background-color: #F3D250;;
}
  .cardThree{
    background-color: #F78888;;
}

  .cardFour{
  background-color: #c77dff;
}

  .cardFive{
  background-color: #b8f2e6;
}

  .card-title{
      color: #05386B;
      font-weight: bold;
      font-size: 32px;
  }

  .card-subtitle{
      color: black;
  }

  .btn{
      background-color: #05386B;
      border-color: #05386B;
      width: 100%;
      color: white;
      border-radius: 25px;

  }

  .btn:hover{
        box-shadow: 0 12px 16px 0 rgba(0,0,0,0.24), 0 17px 50px 0
rgba(0,0,0,0.19);
      background-color: #3E00FF;
      border-radius: 25px;
      font-weight:bold;
  }
```

```
  .card{
    border-radius: 15px;
    }

  .card:hover{
    border-radius: 15px;
    transform: scale(1.1);
    box-shadow: 0 12px 16px 0 rgba(0,0,0,0.24), 0 17px 50px 0
rgba(0,0,0,0.19);
    background-color: #5C7AEA;

}

  .titlewise{
      margin-left: 40%;
      margin-right: 30%;
  }

  * Grow Rotate */
  .hvr-grow-rotate {
    display: inline-block;
    vertical-align: middle;
    -webkit-transform: perspective(1px) translateZ(0);
    transform: perspective(1px) translateZ(0);
    box-shadow: 0 0 1px rgba(0, 0, 0, 0);
    -webkit-transition-duration: 0.3s;
    transition-duration: 0.3s;
    -webkit-transition-property: transform;
    transition-property: transform;
  }
  .hvr-grow-rotate:hover, .hvr-grow-rotate:focus, .hvr-grow-rotate:active {
    -webkit-transform: scale(1.1) rotate(4deg);
    transform: scale(1.1) rotate(4deg);
  }



 /* Responsive Styles */
 @media screen and (max-width: 767px) {

    .ownClass{
        margin-left: 225px;
    }
  }

  @media screen and (max-width: 575px) {
    .ownClass{
        margin-left: 100px;
    }  }
```

Anhang H: User HTML

```
<head>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.1/dist/css/bootstrap.min.css"
    integrity="sha384-
F3w7mX95PdgyTmZZMECAngseQB83DfGTowi0iMjiWaeVhAn4FJkqJByhZMI3AhiU"
crossorigin="anonymous">

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.1/dist/js/bootstrap.min.js"
    integrity="sha384-
skAcpIdS7UcVUC05LJ9Dxay8AXcDYfBJqt1CJ85S/CFujBsIzCIv+l9liuYLaMQ/"
    crossorigin="anonymous"></script>

<script src="https://code.jquery.com/jquery-3.2.1.slim.min.js"
    integrity="sha384-
KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN"
    crossorigin="anonymous"></script>

<script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js
"
    integrity="sha384-
ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"
    crossorigin="anonymous"></script>
</head>

<body>

    <div class="background wow fadeInUp">


        <div class="spacer100"></div>
        <div>
            <h1 class="titlewise wow fadeInUp">My Homework</h1>
        </div>
        <div class="h-20">
            <div class="container mt-4 ownClass">
                <div class="row align-items-center" id="dataID">
                    <template #messagecontainer>
                    </template>
                </div>
            </div>
        </div>


    </div>
</body>

<footer></footer>
```

Anhang I: User TypeScript

```
import { Component, OnInit, ViewChild, ViewContainerRef,
ComponentFactoryResolver } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { WebSocketService } from '../web-socket.service';
import { HomeworkTemplateComponent } from '../homework-template/homework-
template.component';


@Component({
  selector: 'app-chat',
  templateUrl: './user.component.html',
  styleUrls: ['./user.component.css'],
})



export class ChatComponent implements OnInit {

  @ViewChild('messagecontainer', { read: ViewContainerRef })
  entry!: ViewContainerRef;

  array: any = [];
  newClass: any;
  index: any = 0;
  newID: any;
  username  = this.router.snapshot.paramMap.get('username');

  constructor(private router: ActivatedRoute, private webSocketService:
WebSocketService, private resolver: ComponentFactoryResolver) {
  }
```

```
  createComponent(fach: any, title: any, message: any, date: any, parentsCC:
any, messageLong: any, id: any) {
    // @ts-ignore: Object is possibly 'null'.
    //this.entry.clear();
    var text = fach + title + message + date + parentsCC;

    if (this.array.includes(text) == false) {
      this.array.push(text);
      const factory =
this.resolver.resolveComponentFactory(HomeworkTemplateComponent);
      // @ts-ignore: Object is possibly 'null'.
      const componentRef = this.entry.createComponent(factory, 0);

      componentRef.instance.fach = fach;
      componentRef.instance.title = title;
      componentRef.instance.message = message;
      componentRef.instance.date = date;
      componentRef.instance.parentsCC = parentsCC;
      componentRef.instance.id = id;
      componentRef.instance.messageLong = messageLong;
      this.index++;

      this.newClass = document.getElementsByClassName('identifyClass')[0];
      this.newID = document.getElementsByTagName('app-homework-template')[0];
      this.newID.setAttribute('id', id);

      switch (fach) {
        case "Mathe": {
          this.newClass.classList.add('cardOne');
          break;
        }
        case "Deutsch": {
          this.newClass.classList.add('cardTwo');
          break;
        }
        case "Französisch": {
          this.newClass.classList.add('cardThree');
          break;
        }

        case "Englisch": {
          this.newClass.classList.add('cardFour');
          break;
        }

        case "Informatik": {
          this.newClass.classList.add('cardFive');
          break;
        }
```

```
        default: {
          this.newClass.classList.add('cardThree');
        break;
        }
      }
    } else {
    }
  }


  ngOnInit(): void {
    this.webSocketService.listen('clear').subscribe((data: any) => {
      this.entry.clear();
    });
    this.username  = this.router.snapshot.paramMap.get('username');
    this.webSocketService.listen('getData').subscribe((data: any) => {

      var newMessage = JSON.parse(data);
      var fach = newMessage['fach'];
      var title = newMessage['title'];
      var message = newMessage['message'];
      var id = newMessage['id'];
      if(message.length > 27){
      var messageShort = message.substr(0, 27) + "...";
      } else{
      messageShort = message;
      }
      var date = newMessage['date'];
      var parentsCC = newMessage['parentsCC'];

      this.createComponent(fach, title, messageShort, date, parentsCC,
message, id);

      var created = document.getElementsByTagName('app-homework-
template').length;
      var i = 0;
      while (i < created) {
        var newdoc = document.getElementsByTagName('app-homework-
template')[i];
        newdoc.setAttribute('class', 'col-auto');
        i++;
      }

    });


  }
```

```
  openDetails(id: any) {
    alert(id);
  }

  onLoad() {



  }


}
```

Anhang J: Homework Template CSS

```
.btn-primary{
    margin-left: 10px;

}

.inbox-div{
    margin-left: 30%;
}

.spacer{
    margin-left: 12px;
}

.heighter{
    margin-top: 120px;
}

#messageFeed {
    height: calc(300px);
    overflow-y: scroll;
    margin-bottom: 30px;
  }

.messages {
    background-color: transparent;
    list-style-type: none;
    padding: 0;
    margin: 0;
}

.ownbackground{
  background-color: #5390d9;
}

#messageLong{
  margin-bottom: 100px;
}

#modalfooter{
  background: #F3D250;
}

#modalheader{
  background: #F3D250;
}

.chatInput{
  border-radius: 15px;
}
```

```
textarea {
  resize: none;
  height: 30px;
}

#description{
  font-weight: bold;
  font-size: 18px;
}

#individualMessage{
    margin-top: 12px;
    margin-bottom: 12px;
    padding-top: 10px;
    padding-bottom: 10px;
    border-radius: 25px;
    padding-left: 30px;
    color: #05386B;
    font-weight: bold;

    background-color: #80ED99;
}
.modal-text{
  font-size: 20px;
  color: #05386B;
}

  #messages footer {
    position: fixed;
    bottom: 0;
    left: 0;
    width: 100%; /* 100% of page width */
    padding: 0 15px 15px 15px; /* standard edge margin (15px) */
    height: 53px; /* space for input height (38px) + bottom padding (15px) */
    background-color: #fff;

  }

  .ownClass{
      margin-left: 20%;

  }

  .spacer500{
    padding-top: 250px;

  }
```

```
  .backgroundOfModal{
    background-color: #5C7AEA;
  }
  .background{
      background-color: #ECECEC;;

      -webkit-animation: bgcolorchange 45s infinite; /* Chrome, Safari, Opera
*/
      animation: 45s infinite bgcolorchange;
    }
    @keyframes bgcolorchange {
      0% {
        background-color: #5390d9;
      }
      33% {
        background-color: #4ea8de;
      }
      66% {
        background-color: #56cfe1;
      }
      100% {
        background-color: #72efdd;
      }
    }
    /* Chrome, Safari, Opera */
     @-webkit-keyframes bgcolorchange {
          0%   {background: #80ED99;}
          25%  {background: #F3D250;}
          75%  {background: #F78888;}
          100% {background: #5C7AEA;}

  }

  ::-webkit-scrollbar {
    width: 6px;
}

/* Track */
::-webkit-scrollbar-track {
  box-shadow: inset 0 0 8px #80ED99;
    -webkit-box-shadow: inset 0 0 8px #80ED99;
    -webkit-border-radius: 10px;
    border-radius: 10px;
}
```

```
/* Handle */
::-webkit-scrollbar-thumb {
    -webkit-border-radius: 10px;
    border-radius: 10px;
    background: #3E00FF;
    -webkit-box-shadow: inset 0 0 6px #3E00FF;
    box-shadow: inset 0 0 6px #3E00FF;;
}
::-webkit-scrollbar-thumb:window-inactive {
    background: black;
}

  .cardOne{
      background-color: #80ED99;
      height: 200px;
  }

  .cardTwo{
    background-color: #F3D250;
    height: 200px;
}
.cardThree{
    background-color: #F78888;
    height: 200px;
}

.cardFour{
  background-color: #c77dff;
  height: 200px;
}

  .cardFive{
  background-color: #b8f2e6;
  height: 200px;
}
  .card-title{
      color: #05386B;
      font-weight: bold;
      font-size: 32px;
  }

  .card-subtitle{
      color: black;
  }
```

```
  .card-text{
    color: #05386B;
  }

  .btn{
      background-color: #05386B;
      border-color: #05386B;
      width: 100%;
      color: white;
      border-radius: 25px;

  }

  .btn:hover{
        box-shadow: 0 12px 16px 0 rgba(0,0,0,0.24), 0 17px 50px 0
rgba(0,0,0,0.19);
      background-color: #3E00FF;
      border-radius: 25px;
      font-weight:bold;
  }

  .card{
    border-radius: 15px;
  }

  .card:hover{
    border-radius: 15px;
    transform: scale(1.1);
    box-shadow: 0 12px 16px 0 rgba(0,0,0,0.24), 0 17px 50px 0
rgba(0,0,0,0.19);
    background-color: #5C7AEA;

}

  .titlewise{
      margin-left: 40%;
      margin-right: 30%;
  }


  .testchat{
    background-color: red;
  }
```

```
 .hvr-grow-rotate {
   display: inline-block;
   vertical-align: middle;
   -webkit-transform: perspective(1px) translateZ(0);
   transform: perspective(1px) translateZ(0);
   box-shadow: 0 0 1px rgba(0, 0, 0, 0);
   -webkit-transition-duration: 0.3s;
   transition-duration: 0.3s;
   -webkit-transition-property: transform;
   transition-property: transform;
 }
 .hvr-grow-rotate:hover, .hvr-grow-rotate:focus, .hvr-grow-rotate:active {
   -webkit-transform: scale(1.1) rotate(4deg);
   transform: scale(1.1) rotate(4deg);
 }

 a, u {
   text-decoration: none;
 }

/* Responsive Styles */
@media screen and (max-width: 767px) {

   .ownClass{
       margin-left: 225px;
   }
 }

 @media screen and (max-width: 575px) {
   .ownClass{
       margin-left: 100px;
   }
 }
```

Anhang K: Homework Template HTML

```
<!-- Modal (Fullscreen)-->
<div class="modal fade" id="exampleModalToggle{{id}}" aria-hidden="true" aria-
labelledby="exampleModalToggleLabel"
  tabindex="-1">
  <div class="modal-dialog modal-fullscreen">
    <div class="modal-content ownbackground">
      <div class="modal-header" id="modalheader">
          <h5 class="modal-title card-title wow fadeInUp"
id="exampleModalToggleLabel">{{fach}}: {{title}}</h5>
        <button type="button" class="btn-close" data-bs-dismiss="modal" aria-
label="Close"></button>
      </div>
      <div class="modal-body card-body">
        <div class="container mt-4">
        <div class="card-subtitle wow fadeInUp"
id="description">Beschreibung</div><br>
        <div class="modal-text wow fadeInUp"
id="messageLong">{{messageLong}}</div>
        <br>
        <!--Chat-->
        <div id="testchat">
            <div class="row wow fadeInUp">
              <h1>Fragen der Klasse über {{title}}</h1>
              <div id="messageFeed">
                <ul class="messages wow fadeInUp" id="messages{{id}}">
                </ul>
              </div>
              <div class="input-group mb-3">
                <textarea (keyup.enter)="btnClicked()" type="text" class="form-
control chatInput" id="messageTextarea{{id}}" placeholder="Hier kannst du als
{{myName}} chatten..."></textarea>
                <div class="input-group-append{{id}}">
                  <button class="btn btn-outline-secondary" type="button"
id="send{{id}}"
                    (click)="btnClicked()">Send</button>
                </div>
              </div>
            </div>
          </div>

        </div>


      </div>
      <div class="modal-footer card-text" id="modalfooter">
        <div class="card-subtitle">Weitere Informationen:</div>
```

```
        <div>Abgabedatum: {{dateTransformed}} | CC an Eltern gesendet:
{{parentsCCTransformed}}</div>
      </div>
    </div>
  </div>
</div>

<div class="col mb-3 ml-3 wow fadeInUp">
  <a class="card hvr-grow-rotate identifyClass wow fadeInUp" data-bs-
toggle="modal" href="#exampleModalToggle{{id}}" (click)="getData({id})"
role="button" type="button" style="width: 18rem;">
    <div class="card-body wow fadeInUp">
      <h5 class="card-title wow fadeInUp">{{fach}} </h5>
      <h6 class="card-subtitle mb-2 wow fadeInUp">{{title}}</h6>
      <p class="card-text wow fadeInUp">{{message}}</p>
      <a data-bs-toggle="modal" href="#exampleModalToggle{{id}}" role="button"
type="button"
        class="btn identifyButton wow fadeInUp">Open</a>
    </div>
  </a>
</div>
```

Anhang L: Homework Template TypeScript

```
import { Component, Input, OnInit } from '@angular/core';
import { WebSocketService } from '../web-socket.service';
import * as $ from 'jquery';
import { LoginComponent } from '../login/login.component';



@Component({
 selector: 'app-homework-template"',
 templateUrl: './homework-template.component.html',
 styleUrls: ['./homework-template.component.css'],
 template: ``
})


export class HomeworkTemplateComponent implements OnInit {
 @Input() messager: any = "Platzhalter";
 @Input() fach: string | undefined;
 @Input() title: string | undefined;
 @Input() message: string | undefined;
 @Input() date: string | undefined;
 @Input() parentsCC: string | undefined;
 @Input() messageLong: string | undefined;
 @Input() id: any | undefined;

 public myName: any = this.loginComponent.getUser();
 public parentsCCTransformed: any;


 public dateTransformed: any;


 constructor(private webSocketService: WebSocketService, private
loginComponent: LoginComponent) {



 }




 ngOnInit(): void {
  var identity = this.id;

  // @ts-ignore: Object is possibly 'null'.
```

```
var year = this.date.substring(0, 4);
// @ts-ignore: Object is possibly 'null'.
var month = this.date.substring(5, 7);
// @ts-ignore: Object is possibly 'null'.
var day = this.date.substring(8, 10);


this.dateTransformed = day + "." + month + "." + year

if (this.parentsCC == '0') {
 this.parentsCCTransformed = "Nein"
} else {
 this.parentsCCTransformed = "Ja"
}


var i = 0;
var currentDate = new Date().getDay;

var chatid = 0;
var objid = 0;

this.webSocketService.listen('onDelete').subscribe((data: any) => {

 var el = document.getElementById(data);
 // @ts-ignore: Object is possibly 'null'.
 if (el != null) {
  el.remove();
 }

});

this.webSocketService.listen('test' + identity).subscribe((data: any) => {

 var myobject = JSON.parse(data);

 var size = Object.keys(myobject).length;
 var i = 0;

 for (var key in myobject) {
  if (myobject.hasOwnProperty(key)) {


   var firstname = myobject[key].message
   chatid = myobject[key].chatid;
   objid = myobject[key].id;
   var username = myobject[key].username;

   var list = document.getElementById('messages' + chatid);
```

```
    //DATE
    if ($('[id=' + objid.toString() + chatid.toString() + ']').length < 1) {
     var datum = new Date();

     if (datum.getDay == currentDate) {
      var hours = document.createElement('li');
      hours.style.cssText = 'border-top-left-radius: 25px;border-top-right-
radius: 25px;padding-left: 30px;color: #05386B;font-weight: bold;background-
color: #80ED99; font-size: 10px;'

      hours.appendChild(document.createTextNode(datum.getHours().toString() +
":" + (datum.getMinutes() < 10 ? '0' : '') + datum.getMinutes() + " Uhr" + "
von " + username));
      // @ts-ignore: Object is possibly 'null'.
      list.appendChild(hours);
     }

     else {
      var date = document.createElement('li');
      date.style.cssText = 'border-top-left-radius: 25px;border-top-right-
radius: 25px;padding-left: 30px;color: #05386B;font-weight: bold;background-
color: #80ED99; font-size: 12px;'

      date.appendChild(document.createTextNode(datum.toString()));
      // @ts-ignore: Object is possibly 'null'.
      list.appendChild(date);
     }

     var entry = document.createElement('li');
     entry.setAttribute("id", objid.toString() + chatid.toString());
     entry.style.cssText = 'margin-bottom: 12px;padding-bottom: 10px;border-
bottom-left-radius: 25px;border-bottom-right-radius: 25px;padding-left:
30px;color: #05386B;font-weight: bold;background-color: #80ED99;'


     entry.appendChild(document.createTextNode(firstname));

     // @ts-ignore: Object is possibly 'null'.
     list.appendChild(entry);

     var element = document.getElementById(objid.toString() +
chatid.toString());
     // @ts-ignore: Object is possibly 'null'.
     element.scrollIntoView();

    }
```

```
    }
   }
  });



 }

 btnClicked() {
  // @ts-ignore: Object is possibly 'null'.
  /*var sender = document.getElementById('username'+this.id).value;*/
  var sender = this.myName;
  // @ts-ignore: Object is possibly 'null'.
  if (document.getElementById('messageTextarea' + this.id).value != null &&
document.getElementById('messageTextarea' + this.id).value != '') {
   // @ts-ignore: Object is possibly 'null'.
   var msg = document.getElementById('messageTextarea' + this.id).value;
   // @ts-ignore: Object is possibly 'null'.
   this.webSocketService.emit3('data', msg, this.id, sender);
   // @ts-ignore: Object is possibly 'null'.
   document.getElementById('messageTextarea' + this.id).value = '';

  }







 }


 getData(id: any) {
  this.webSocketService.emit('getMessageHistory', id)
 }
}
```

Anhang M: CreatedHomeworkTemplate CSS

```
.btn-primary{
  margin-left: 10px;

}

.inbox-div{
  margin-left: 30%;
}

.spacer{
  margin-left: 12px;
}

.heighter{
  margin-top: 120px;
}

#messageFeed {
  height: calc(300px); /* space for fixed navbar (48px), "main" title (71px),
footer (53px) */
  overflow-y: scroll;
  margin-bottom: 30px;
 }

.messages {
  background-color: transparent;
  list-style-type: none;
  padding: 0;
  margin: 0;
}

#individualMessage{
  margin-top: 12px;
  margin-bottom: 12px;
  padding-top: 10px;
  padding-bottom: 10px;
  border-radius: 25px;
  padding-left: 30px;
  color: #05386B;
  font-weight: bold;

  background-color: #80ED99;
}
.modal-text{
 font-size: 20px;
```

```
 color: #05386B;
}

 #messages footer {
  position: fixed;
  bottom: 0;
  left: 0;
  width: 100%; /* 100% of page width */
  padding: 0 15px 15px 15px; /* standard edge margin (15px) */
  height: 53px; /* space for input height (38px) + bottom padding (15px) */
  background-color: #fff;

 }

 .ownClass{
   margin-left: 20%;

 }

 .spacer500{
  padding-top: 250px;

 }

 .backgroundOfModal{
  background-color: #5C7AEA;
 }
 .background{
   background-color: #ECECEC;;

   -webkit-animation: bgcolorchange 45s infinite; /* Chrome, Safari, Opera */
   animation: 45s infinite bgcolorchange;
  }
  @keyframes bgcolorchange {
   0% {
    background-color: #5390d9;
   }
   33% {
    background-color: #4ea8de;
   }
   66% {
    background-color: #56cfe1;
   }
   100% {
    background-color: #72efdd;
   }
  }
  /* Chrome, Safari, Opera */
   @-webkit-keyframes bgcolorchange {
```

```
    0%  {background: #80ED99;}
    25% {background: #F3D250;}
    75% {background: #F78888;}
    100% {background: #5C7AEA;}

 }

 ::-webkit-scrollbar {
  width: 6px;
}

/* Track */
::-webkit-scrollbar-track {
 box-shadow: inset 0 0 8px #80ED99;
  -webkit-box-shadow: inset 0 0 8px #80ED99;
  -webkit-border-radius: 10px;
  border-radius: 10px;
}

/* Handle */
::-webkit-scrollbar-thumb {
  -webkit-border-radius: 10px;
  border-radius: 10px;
  background: #3E00FF;
  -webkit-box-shadow: inset 0 0 6px #3E00FF;
  box-shadow: inset 0 0 6px #3E00FF;;
}
::-webkit-scrollbar-thumb:window-inactive {
  background: black;
}

 .cardOne{
   background-color: #80ED99;
   height: 200px;
 }

 .cardTwo{
  background-color: #F3D250;
  height: 200px;
}
.cardThree{
  background-color: #F78888;
  height: 200px;
}

.cardFour{
 background-color: #c77dff;
 height: 200px;
}
```

```
 .cardFive{
 background-color: #b8f2e6;
 height: 200px;
}
 .card-title{
   color: #05386B;
   font-weight: bold;
   font-size: 32px;
 }

 .card-subtitle{
   color: black;
 }



 .card-text{
  color: #05386B;
 }

 .btn{
   background-color: #05386B;
   border-color: #05386B;
   width: 100%;
   color: white;
   border-radius: 25px;

 }

 .btn:hover{
    box-shadow: 0 12px 16px 0 rgba(0,0,0,0.24), 0 17px 50px 0 rgba(0,0,0,0.19);
   background-color: #3E00FF;
   border-radius: 25px;
   font-weight:bold;
 }

 .card{
  border-radius: 15px;
  width: 18rem;
 }

 .card:hover{
  border-radius: 15px;
  transform: scale(1.1);
  box-shadow: 0 12px 16px 0 rgba(0,0,0,0.24), 0 17px 50px 0 rgba(0,0,0,0.19);
  background-color: #5C7AEA;

}
```

```
.titlewise{
  margin-left: 40%;
  margin-right: 30%;
}

.ownChatSpace{
 margin-top: 300px;
}
.testchat{
 background-color: red;
}


.hvr-grow-rotate {
 display: inline-block;
 vertical-align: middle;
 -webkit-transform: perspective(1px) translateZ(0);
 transform: perspective(1px) translateZ(0);
 box-shadow: 0 0 1px rgba(0, 0, 0, 0);
 -webkit-transition-duration: 0.3s;
 transition-duration: 0.3s;
 -webkit-transition-property: transform;
 transition-property: transform;
}
.hvr-grow-rotate:hover, .hvr-grow-rotate:focus, .hvr-grow-rotate:active {
 -webkit-transform: scale(1.1) rotate(4deg);
 transform: scale(1.1) rotate(4deg);
}

a, u {
 text-decoration: none;
}

/* Responsive Styles */
@media screen and (max-width: 767px) {

 .ownClass{
   margin-left: 225px;
 }
}

@media screen and (max-width: 575px) {
 .ownClass{
   margin-left: 100px;
 }
}
```

Anhang N: CreatedHomeworkTemplate HTML

```
<div class="col-lg-auto col-md-auto col-sm-auto col-xs-auto mb-3 ml-3 wow
fadeInUp">
  <div class="card hvr-grow-rotate identifyClass wow fadeInUp">
    <div class="card-body wow fadeInUp">
      <h5 class="card-title wow fadeInUp">{{fach}} </h5>
      <h6 class="card-subtitle mb-2 wow fadeInUp">{{title}}</h6>
      <p class="card-text wow fadeInUp">{{message}}</p>
      <button type="button" class="btn btn-danger" id="deleteButton{{id}}"
(click)="deleteHomework(id)">🗑</button>
    </div>
  </div>
</div>
```

Anhang O: CreatedHomeworkTemplate TypeScript

```
import { Component, OnInit, Input } from '@angular/core';
import { WebSocketService } from '../web-socket.service';

@Component({
 selector: 'app-created-homework-template',
 templateUrl: './created-homework-template.component.html',
 styleUrls: ['./created-homework-template.component.css']
})
export class CreatedHomeworkTemplateComponent implements OnInit {
 @Input() fach: string | undefined;
 @Input() title: string | undefined;
 @Input() message: string | undefined;
 @Input() date: string | undefined;
 @Input() parentsCC: string | undefined;
 @Input() messageLong: string | undefined;
 @Input() id: any | undefined;

 constructor(private webSocketService: WebSocketService) { }


 ngOnInit(): void {

  this.webSocketService.listen('onDelete').subscribe((data: any) => {

   var el = document.getElementById(data);
   // @ts-ignore: Object is possibly 'null'.
   if(el != null){
   el.remove();
   }

  });

 }

 deleteHomework(id: any){
  this.webSocketService.emit('deleteID', id)
 }


}
```

Anhang P: WebSocketService TypeScript

```
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { io } from 'socket.io-client';

@Injectable({
 providedIn: 'root'
})
export class WebSocketService {
 socket: any;
 constructor() {
  this.socket = io();
 }

 listen(eventName: string){
  return new Observable((subscriber) =>{
   this.socket.on(eventName, (data: any) =>{
    subscriber.next(data);
   })
  });
 }

 emit(eventName: string, data: any){
  this.socket.emit(eventName, data);
 }

 emit2(eventName: string, data: any, data2: any){
  this.socket.emit(eventName, data, data2);
 }

 emit3(eventName: string, data: any, data2: any, messageSender: any){
  this.socket.emit(eventName, data, data2, messageSender);
 }

 emit6(eventName: string, fach: any, title: any, message: any, date: any,
creation_date: any, parentsCC: any){
  this.socket.emit(eventName, fach, title, message, date, creation_date,
parentsCC);
 }
}
```

Anhang Q: server.js

```
#!/usr/bin/env node
const express = require('express');
const app = express();
const http = require('http');
const server = http.createServer(app);
var persistenceService = require('./persistenceService')

io = require('socket.io')(server);


const bodyParser = require('body-parser');
const { EventEmitter } = require('stream');
const { checkLogin } = require('./persistenceService');
const amqpservice = require('./amqpservice');

app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());

app.use(express.static('firstapp'));


var emitter = new EventEmitter();

server.listen(8082, () => {
  console.log('Server listening on *:8082');
});

/* JEDEN TAG um Mitternach überprüfen, ob Einträge älter als 7 Tage sind, wenn
dem so ist, werden diese gelöscht!*/
resetAtMidnight();

function resetAtMidnight() {
  var now = new Date();
  var night = new Date(
    now.getFullYear(),
    now.getMonth(),
    now.getDate() + 1, // nächster Tag
    0, 0, 0,// um 0.00 Uhr
  );
  var msToMidnight = night.getTime() - now.getTime();

  setTimeout(function() {
    var current_date = new Date()
    var c_date = current_date.toISOString().split('T')[0]
    amqpservice.sendTriggerViaAMQP();
  }, msToMidnight);
}
```

```
io.on('connection', function (socket) {
  console.log('Ein neuer Client hat sich verbunden:' + socket.id);
  // SOCKET LISTENER
  socket.on('data', function (data, id, messageSender) {
    send(data, 'hello'+id, id, messageSender);
  });

  socket.on('getMessageHistory', function(chatid){
    persistenceService.getMessagesFromDatabaseExported(chatid);
  })

  socket.on('deleteID', function(data){
    persistenceService.deleteFromDatabase(data);
  })
  socket.on('mysqldata', function (data, data2) {
    //Schaut ob Passwort übereinstimmt
    logIn();
    function logIn(){
      checkLogin(data, data2, function(wert){
        if(wert == true){
          socket.username = data;
          socket.emit('loginAuthorized', data);
        }else{
          socket.emit('loginNotAuthorized', data);
        }
      });
    }

  });


  //Erhalte Socket Message des neuen Hausaufgabeneintrags
  socket.on('newHomework', function (fach, title, message, date, creation_date,
parentsCC) {
    var obj = {};
    obj['fach'] = fach;
    obj['title'] = title;
    obj['message'] = message;
    obj['date'] = date;
    obj['creation_date'] = creation_date;
    obj['parentsCC'] = parentsCC;
    var json = JSON.stringify(obj);

    sendNewHomework(json, 'homeworkqueue');


  });
```

```
var sendNewHomework = function (message, queuevar) {
  var amqp = require('amqplib/callback_api');

  amqp.connect('amqp://localhost', function (error0, connection) {
    if (error0) {
      throw error0;
    }
    connection.createChannel(function (error1, channel) {
      if (error1) {
        throw error1;
      }
      var queue = queuevar;
      var msg = message;
      channel.assertQueue(queue, {
        durable: true
      });
      channel.sendToQueue(queue, Buffer.from(msg));
      console.log(" [x] Nachricht gesendet %s", msg);
      receiveNewHomework();

    });
    setTimeout(function() {
      connection.close();
    }, 500);


  });



};

var receiveNewHomework = function () {
  var amqp = require('amqplib/callback_api');
  amqp.connect('amqp://localhost', function (error0, connection) {
    if (error0) {
      throw error0;
    }
    connection.createChannel(function (error1, channel) {
      if (error1) {
        throw error1;
      }

      var queue = 'homeworkqueue';

      channel.assertQueue(queue, {
        durable: true
      });
```

```
        console.log(" [*] Es wird auf eine neue Nachricht gewartet", queue);
        channel.consume(queue, function (msg) {

          console.log(" [x] Nachricht erhalten %s:", msg.content.toString());

          var newMessage = JSON.parse(msg.content.toString());
          var fach = newMessage['fach'];
          var title = newMessage['title'];
          var messageone = newMessage['message'];
          var date = newMessage['date'];
          var creation_date = newMessage['creation_date'];
          var parentsCC = newMessage['parentsCC'];

          if (parentsCC == 'yes') {
            var ccint = 1;
          } else {
            ccint = 0;
          }
          persistenceService.insertIntoDatabase(fach, title, messageone, date,
creation_date, ccint);


        }, {
          noAck: true
        });
        setTimeout(function() {
          connection.close();
        }, 500);
      });

    });




  };


  function send(message, queuevar, idchat, messageSender) {
    var amqp = require('amqplib/callback_api');

    amqp.connect('amqp://localhost', function (error0, connection) {
      if (error0) {
        throw error0;
      }
      connection.createChannel(function (error1, channel) {
        if (error1) {
          throw error1;
```

```
        }

        var myMessageObject = {};
        myMessageObject['message'] = message;
        myMessageObject['messageSender'] = messageSender;
        var myMessageObjectString = JSON.stringify(myMessageObject);

        var queue = queuevar;
        var msg = message;

        channel.assertQueue(queue, {
          durable: true
        });
        channel.sendToQueue(queue, Buffer.from(myMessageObjectString));

        console.log(" [x] Nachricht gesendet %s", myMessageObjectString);

        receiveMSG(queuevar, idchat);

      });
      setTimeout(function() {
        connection.close();
      }, 500);
    });
  };



function receiveMSG(queuevar, idchat) {
  var amqp = require('amqplib/callback_api');
  amqp.connect('amqp://localhost', function (error0, connection) {
    if (error0) {
      throw error0;
    }
    connection.createChannel(function (error1, channel) {
      if (error1) {
        throw error1;
      }
      var queue = queuevar;
      channel.assertQueue(queue, {
        durable: true
      });

      console.log(" [*] Es wird auf eine Nachricht gewartet", queue);
      channel.consume(queue, function (msg) {

        var messageObject = JSON.parse(msg.content)

        console.log(messageObject.message)
```

```
        console.log(" [x] Nachricht erhalten: %s", messageObject.message);

        persistenceService.insertMessageIntoDatabase(idchat,
messageObject.message, messageObject.messageSender)


      }, {
        noAck: true
      });
      setTimeout(function() {
        connection.close();
      }, 500);

    });
  });
}
});
```

Anhang R: publisherService.js

```
const amqplib = require('amqplib/callback_api');

// Create connection to AMQP server

module.exports = {
  sendEmail: function(receiver, subject, title, message, date){
    amqplib.connect('amqp://localhost', (err, connection) => {
      if (err) {
        console.error(err.stack);
        return process.exit(1);
      }

      // Create channel
      connection.createChannel((err, channel) => {
        if (err) {
          console.error(err.stack);
          return process.exit(1);
        }

        queue = "MailAnEltern";

        // Ensure queue for messages
        channel.assertQueue(queue, {
          // Ensure that the queue is not deleted when server restarts
          durable: true
        }, err => {
          if (err) {
            console.error(err.stack);
            return process.exit(1);
          }

          // Create a function to send objects to the queue
          // Javascript object is converted to JSON and then into a Buffer
          let sender = (content, next) => {
            let sent = channel.sendToQueue(queue,
Buffer.from(JSON.stringify(content)), {
              // Store queued elements on disk
              persistent: true,
              contentType: 'application/json'
            });
            if (sent) {
              return next();
            } else {
              channel.once('drain', () => next());
            }
          };
```

```
          let sendNext = () => {

              console.log('E-Mail in RabbitMQ gelegt');
              // Close connection to AMQP server
              // We need to call channel.close first, otherwise pending
              // messages are not written to the queue
              return channel.close(() => connection.close());
            }

            sender({
              to: receiver,
              subject: subject+ ": " + title,
              text: message + " Dies ist zu erledigen bis: " + date,
              html: "<h1>"+ message + " Dies ist zu erledigen bis: "+ date +
"</h1>"
            }, sendNext);


            console.log("Ausgeführt!");

        });
      });
    });

  },


}
```

Anhang S: persistenceService.js

```
var mysql = require('mysql');
var amqppublisher = require('./publisherService');
var amqpservice = require('./amqpservice')

var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "root"
});

myArray = new Array();

con.connect(function (err) {
  if (err) throw err;
  console.log("MYSQL Connected!");


});

con.query("USE myhomework;", function (err, result) {
  if (err) throw err;

});


module.exports = {


insertIntoDatabase: function(fach, title, messageone, date, creation_date,
ccint) {
  con.query("INSERT INTO homework(fach, title, message, date, created,
parentsCC) VALUES('" + fach + "', '" + title + "','" + messageone + "','" +
date + "','" + creation_date + "'," + ccint + ");", function (err, result,
fields) {
    if (err) {
      return false;

    } else {
      console.log("MYSQL Eintrag erstellt!");
      if(ccint == 1){
        amqppublisher.sendEmail('finn.graf@ost.ch', fach, title, messageone,
date);
      }
      getDataFromDatabase();
    }
  });
```

```
},

checkLogin: function(data, data2, _callback) {
  con.query("SELECT password FROM users WHERE USERNAME='" + data + "';",
function (err, result, fields) {
    if (err) {
      return;
    } else {
      if (result.length > 0) {
        password = result[0].password;
        if (password == data2) {
          getDataFromDatabase();
          _callback(true);



        } else {
          _callback(false);


        }
      } else {
        _callback(false);

      }

    }
  });

},

deleteFromDatabase: function(id) {
  con.query("DELETE from homework where id="+id+";", function (err, result,
fields) {
    if (err) {
      return;
    } else {
      io.emit('onDelete', id);
    }
  });

},


insertMessageIntoDatabase: function(chatid, message, username) {
  con.query("INSERT INTO messages(chatid, message, username) VALUES('" + chatid
+ "', '" + message + "','" + username + "');", function (err, result, fields) {
    if (err) {
      console.log(err);
```

```
      return false;

    } else {

      getMessagesFromDatabase(chatid);
    }
  });
},

getMessagesFromDatabaseExported: function(chatid){
  getMessagesFromDatabase(chatid);
},

deleteAfter7Days: function() {
  con.query("DELETE FROM homework WHERE created < DATE_SUB(NOW(), INTERVAL 7
DAY)", function (err, result, fields) {
    if (err) {
      return;
    } else {
    }
  });

},




}

function getMessagesFromDatabase(chatid){

  var array = [];

  if(typeof chatid == 'object'){
  newchatid = chatid.id;
  } else{
    newchatid = chatid;
  }


  /*"SELECT * FROM messages where chatid= '"+chatid+"';"*/
  con.query("SELECT * FROM messages where chatid="+newchatid+";", function
(err, result, fields) {
    if (err) {
      console.log(err)
      return;
    } else {
      if (result.length > 0) {
        var i = 0;
```

```
        while (i < result.length){
        var obj = {};
        obj['id'] = result[i].id;
        obj['chatid'] = result[i].chatid;
        obj['message'] = result[i].message;
        obj['username'] = result[i].username;
        var objString = JSON.stringify(obj);


        array[array.push(obj)];
        /*amqpservice.sendMessageBackViaAMQP(objString,
'nachrichtback'+obj['chatid'], obj['chatid']) */



        /*io.emit('test'+result[i].chatid, objString); */
        i++;

        }
        var json = JSON.stringify(array);

        amqpservice.sendMessageBackViaAMQP(json, 'nachrichtback'+obj['chatid'],
obj['chatid'])


      }
    }


  });

}


function getDataFromDatabase(table){
  con.query("SELECT * FROM homework;", function (err, result, fields) {
    if (err) {
      return;
    } else {
      if (result.length > 0) {
        var i = 0;
        while (i < result.length){
        var obj = {};
        obj['id'] = result[i].id;
        obj['fach'] = result[i].fach;
        obj['title'] = result[i].title;
        obj['message'] = result[i].message;
        obj['date'] = result[i].date;
        obj['parentsCC'] = result[i].parentsCC;
```

```
        var objString = JSON.stringify(obj);

        io.emit('getData', objString);


        i++;









        }

      }
    }
  });
}
```

Anhang T: amqpservice.js

```
module.exports = {

  sendMessageBackViaAMQP: function(message, queuevar, idchat) {
    var amqp = require('amqplib/callback_api');

    amqp.connect('amqp://localhost', function (error0, connection) {
      if (error0) {
        throw error0;
      }

      connection.createChannel(function (error1, channel) {
        if (error1) {
          throw error1;
        }

        var queue = queuevar;
        var msg = message;

        channel.assertQueue(queue, {
          durable: true
        });
        channel.sendToQueue(queue, Buffer.from(msg));

        receiveMessageBackViaAMQP(msg, queuevar, idchat);


      });
      setTimeout(function() {
        connection.close();
      }, 500);
    });

  },
  sendTriggerViaAMQP


}

function sendTriggerViaAMQP(){
  var amqp = require('amqplib/callback_api');

  amqp.connect('amqp://localhost', function (error0, connection) {
    if (error0) {
      throw error0;
    }
    connection.createChannel(function (error1, channel) {
```

```
      if (error1) {
        throw error1;
      }

      var queue = "TriggerQueue";
      var msg = "Item found, that should be deleted.";

      channel.assertQueue(queue, {
        durable: true
      });
      channel.sendToQueue(queue, Buffer.from(msg));


    });
    setTimeout(function() {
      connection.close();
    }, 500);
  });


}




function receiveMessageBackViaAMQP(message, queuevar, idchat){
  var amqp = require('amqplib/callback_api');
  amqp.connect('amqp://localhost', function (error0, connection) {
    if (error0) {
      throw error0;
    }
    connection.createChannel(function (error1, channel) {
      if (error1) {
        throw error1;
      }

      var queue = queuevar;

      channel.assertQueue(queue, {
        durable: true
      });


      channel.consume(queue, function (msg) {


        io.emit('test'+idchat, msg.content.toString());
```

```
      }, {
        noAck: true
      });
      setTimeout(function() {
        connection.close();
      }, 500);
    });
  });

}
```

Anhang U: triggerserver.js

```
const http = require('http');
const amqplib = require('amqplib/callback_api');

var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "root"
});

con.connect(function (err) {
  if (err) throw err;
  console.log("MYSQL Connected!");
});

con.query("USE myhomework;", function (err, result) {
  if (err) throw err;

});

const hostname = 'localhost';
const port = 3000;

receiveTrigger();

const server = http.createServer((req, res) => {
 res.statusCode = 200;
 res.setHeader('Content-Type', 'text/plain');
 res.end('Hello World');
});

server.listen(port, hostname, () => {
 console.log(`Server running at http://${hostname}:${port}/`);
});

function receiveTrigger(){
  var amqp = require('amqplib/callback_api');
  amqp.connect('amqp://localhost', function (error0, connection) {
    if (error0) {
      throw error0;
    }
    connection.createChannel(function (error1, channel) {
      if (error1) {
        throw error1;
      }
```

```
      var queue = "TriggerQueue";

      channel.assertQueue(queue, {
        durable: true
      });


      channel.consume(queue, function (msg) {
        con.query("DELETE FROM homework WHERE created < DATE_SUB(NOW(),
INTERVAL 7 DAY)", function (err, result, fields) {
          if (err) {
            return;
          } else {
          }
        });


      }, {
        noAck: true
      });

    });
  });




}
```

Anhang V: mailserver.js

```
'use strict';


console.log("Gestartet")
const SMTPServer = require('smtp-server').SMTPServer;

// Setup server
const server = new SMTPServer({

  // log to console
  logger: false,

  // not required but nice-to-have
  banner: 'Welcome to My Awesome SMTP Server',

  // disable STARTTLS to allow authentication in clear text mode
  disabledCommands: ['STARTTLS'],

  // Accept messages up to 10 MB. This is a soft limit
  size: 10 * 1024 * 1024,

  // Setup authentication
  // Allow all usernames and passwords, no account checking
  onAuth(auth, session, callback) {
    return callback(null, {
      user: {
        username: auth.username
      }
    });
  },

  // Handle message stream
  onData(stream, session, callback) {
    console.log('Streaming message from user %s', session.user.username);
    console.log('------------------');
    stream.pipe(process.stdout);
    stream.on('end', () => {
      console.log(''); // ensure linebreak after the message
      callback(null, 'Message queued as ' + Date.now()); // accept the message
once the stream is ended
    });
  }
});

server.on('error', err => {
  console.log('Error occurred');
  console.log(err);
```

```
});

// start listening
server.listen(1000, 'localhost');
```

Anhang W: subscriber.js

```
const amqplib = require('amqplib/callback_api');
const nodemailer = require('nodemailer');



var transport = nodemailer.createTransport({
  service: 'gmail',
  auth: {
   user: 'amqptestwise@gmail.com',
   pass: ''
  }
 });

// Create connection to AMQP server
amqplib.connect('amqp://localhost', (err, connection) => {
  if (err) {
    console.error(err.stack);
    return process.exit(1);
  }
  connection.createChannel((err, channel) => {
    if (err) {
      console.error(err.stack);
      return process.exit(1);
    }

    queue = "MailAnEltern";
    channel.assertQueue(queue, {
      durable: true
    }, err => {
      if (err) {
        console.error(err.stack);
        return process.exit(1);
      }

      // Only request 1 unacked message from queue
      // This value indicates how many messages we want to process in parallel
      channel.prefetch(1);

      channel.consume(queue, data => {
        if (data === null) {
          return;
        }

        let message = JSON.parse(data.content.toString());

        transport.sendMail(message, (err, info) => {
          if (err) {
```

```
            console.error(err.stack);
            return channel.nack(data);
          }
          console.log('Nachricht ausgestellt: %s', info.messageId);
          channel.ack(data);
        });
      });
    });
  });
});
```

Anhang X: Berechnung Marktanalyse

Produkt	ITCentralStation A_Reviews	Mittelwert_Reviews	TrustRadius A_Reviews	Mittelwert_Reviews2	G2 A_Reviews	Mittelwert_Reviews4	Anzahl Reviews Total (ART)
Apache Kafka	18	7.7	80	9	61	9	159
IBM MQ	37	8.1	31	8.6	86	8.6	154
Amazon SQS	1	8	18	9.2	97	8.6	116
RabbitMQ	5	8	29	8.4	19	8	53
Google Pub/Sub	0	0	17	8.5	25	8.8	42
Solace PubSub+	9	8.9	0	0	2	9	11
Apache Active MQ	1	7	0	0	14	8.6	15
Red Hat AMQ	2	9	0	0	5	7.4	7
TIBCO EMS	0	0	61	7.3	5	10	66
ZeroMQ	0	0	0	0	22	8.6	22
Azure Service Bus	0	0	0	0	23	7.4	23
m		**8.1**		**8.5**		**8.545454545**	

Im ersten Schritt wurden pro Plattform Mittelwerte errechnet.

Produkt	BayesianAVG	BayesianAVG	BayesianAVG	MittelwertBaysianAVG	Total Reviews
Apache Kafka	7.842857143	8.9	8.85014985	8.531002331	159
IBM MQ	8.1	8.560784314	8.585893417	8.415559244	154
Amazon SQS	8.090909091	8.831578947	8.587115247	8.503201095	116
RabbitMQ	8.066666667	8.440816327	8.333951763	8.280478252	53
Google Pub/Sub	0	8.5	8.661157025	8.580578512	42
Solace PubSub+	8.478947368	0	8.573863636	8.526405502	11
Apache Active MQ	8	0	8.562809917	8.281404959	15
Red Hat AMQ	8.25	0	8.381818182	8.315909091	7
TIBCO EMS	0	7.596296296	8.753246753	8.174771525	66
ZeroMQ	0	0	8.568531469	8.568531469	22
Azure Service Bus	0	0	8.048370497	8.048370497	23
C	**10**	**20**	**30**	**8.384201134**	

In einem zweiten Schritt wurde pro Produkt (einzeln pro Plattform) mithilfe der Variablen **C** und **m** ein Bayesian Average berechnet. Daraus wiederum ein Durchschnitt pro Produkt ausgerechnet und aus diesen wiederum ein Durchschnitt aller Produkte aus allen Plattformen (dunkelgraue Zelle, diese wird im nächsten Schritt als Variable **m** verwendet).

Produkt	BayesianAVG_Total	Gewichtung	mit Berücksichtigung der Gewichtung
Apache Kafka	8.495882427	1	8.495882427
IBM MQ	8.407873432	1	8.407873432
Amazon SQS	8.467357733	1	8.467357733
RabbitMQ	8.330829166	1	8.330829166
Google Pub/Sub	8.473851676	0.98	8.304374643
Solace PubSub+	8.409844545	0.98	8.241647654
Apache Active MQ	8.36047894	0.98	8.193269361
Red Hat AMQ	8.375814392	0.98	8.208298104
TIBCO EMS	8.265042908	0.98	8.09974205
ZeroMQ	8.440524292	0.95	8.018498077
Azure Service Bus	8.278391482	0.97	8.030039737
C	50		

Im letzten Schritt wurden die Zahlen aus den vorigen Schritten benutzt, um ein totales Bayesian Average (über alle Plattformen pro Produkt) einerseits ohne und andererseits mit Gewichtung zu berechnen. Die letzte Spalte entspricht den Werten aus der finalen Rangliste.