

PRAXIS DER SOFTWAREENTWICKLUNG

KARLSRUHER INSTITUT FÜR TECHNOLOGIE

INSTITUT FÜR TELEMATIK

LEHRSTUHL PROF. DR. MARTINA ZITTERBART



Modern Messaging Platform: ChatSphere

Entwurfsdokumentation

Alexander Wank Niklas Seyfarth

Julien Midedji Berthold Niemann

Alexander Brese

betreut von: Tim Gerhard und Markus Jung

2. Juli 2018

Inhaltsverzeichnis

1. Softwarearchitektur	5
1.1. Technologien	5
1.1.1. Vue.js	6
1.1.2. Vuetify	6
1.1.3. Vue-Apollo	6
1.1.4. Apollo-Link-State	6
1.1.5. Apollo-InMemory-Cache	6
1.1.6. Apollo-Cache-Persist	6
1.1.7. Graphql-Java	6
1.1.8. OrmLite	6
1.1.9. MariaDB	6
1.1.10. Ormlite-rx	6
1.1.11. Caddy	6
1.1.12. Jetty	6
1.1.13. ORY Hydra	6
1.2. Entwurfsmuster	6
1.2.1. Client-Server-Architektur	7
1.2.2. Model-View-ViewModel	7
1.2.3. Flux-Pattern	8
1.3. Architektur	9
1.3.1. Relationale Datenbank	9
1.3.2. ORM	9
1.3.3. GraphQL	10
1.3.4. SPA	11
1.3.5. PWA	12
1.3.6. Websockets	13
1.3.7. Data Management	14
1.3.8. Ablauf eines GraphQL-Query	16
1.3.9. GraphQL-Websocket-Transport	16
2. Einführung	17
2.1. Softwareartefakte	18
2.2. Programmablauf	18

Inhaltsverzeichnis

3. API	22
3.1. Authorisierungsserver	22
3.2. GraphQL-API	24
4. Frontend	29
4.1. Apollo-link-state	29
4.1.1. Registrierung	29
4.1.2. Search	30
4.2. Client-Module	32
4.3. Modulbeschreibungen	35
4.3.1. Registrierung	35
4.3.2. Login	38
4.3.3. Navigation	40
4.3.4. Profile	42
4.3.5. Settings	46
4.3.6. Chatlist	48
4.3.7. Chats	50
4.3.8. Groupchats	52
4.3.9. Contactlist	54
4.3.10. Search	57
4.4. Router-Links	60
4.5. Service Worker	61
5. Datenmodelle	62
6. Backend	70
6.1. Module	73
6.1.1. Server (de.chatsphere.server)	73
6.1.2. IO (de.chatsphere.io)	73
6.1.3. Features (de.chatsphere.feature)	73
7. Kommunikationsabläufe	75
8. User-Stories	75
8.1. Login	75
8.2. Registrierung	75
8.3. Chatlist	75
8.4. Chats	76

Inhaltsverzeichnis

8.5. Groupchats	77
8.6. Profile	78
8.7. Contactlist	78
8.8. Settings	79
8.9. Search	79
A. Anhang	82
A.1. Glossar	85

1. Softwarearchitektur

In ChatSphere kommen Technologien und Paradigmen zu dynamischen Webanwendungen zum Einsatz, die dem aktuellen Stand der Technik entsprechen sollen. Dieses Kapitel skizziert die zugrunde liegenden Entwurfsmuster, sowie Architekturstile und erklärt die grundlegenden Modelle und Werkzeuge zur Bewältigung der Anforderungen.

1.1. Technologien

In diesem Kapitel sollen alle verwendeten Technologien vorgestellt und ihr Einsatz beschrieben werden. Darin enthalten sind Frameworks oder auch Plugins/Packages.

1. Softwarearchitektur

1.1.1. Vue.js

1.1.2. Vuetify

1.1.3. Vue-Apollo

1.1.4. Apollo-Link-State

1.1.5. Apollo-InMemory-Cache

1.1.6. Apollo-Cache-Persist

1.1.7. Graphql-Java

1.1.8. OrmLite

1.1.9. MariaDB

1.1.10. Ormlite-rx

1.1.11. Caddy

1.1.12. Jetty

1.1.13. ORY Hydra

1.2. Entwurfsmuster

In diesem Kapitel sollen bewährte → Design-Patterns vorgestellt werden. Die gewählten Design-Patterns werden kurz beschrieben und der Einsatz in der Anwendung gezeigt.

1. Softwarearchitektur

Diagramme sollen helfen die Vorkommnisse der Design-Patterns in unserem System zu verdeutlichen, um dann in weiteren Kapiteln auf die konkrete Realisierung eingehen zu können.

1.2.1. Client-Server-Architektur

Abbildung 8 zeigt wie sich die Anwendung auf Client und Server aufteilen lässt und soll die Client-Server-Modellierung deutlich machen. Mehrere Clients führen mittels eines Browsers das `index.html` Artefakt aus. Dieses Artefakt bietet die Oberfläche für den Benutzer um die Dienstleistung in Anspruch nehmen zu können.

Der Server muss die Dienstleistungen bereitstellen. Er unterhält umfangreiche Komponenten und koordiniert kommende Anfragen des Benutzers, indem er sie an die Komponenten delegiert. Zur Konfiguration und zum Betrieb der Komponenten müssen wiederum dedizierte Artefakte erstellt werden, die Ziel der Implementierung sein werden.

Anfragen zur Registrierung und zum Login werden in der ORY-Hydra Komponente behandelt. Der Caddy-Webserver stellt Funktionen bereit für eine sichere Webpräsenz. MariaDB verwaltet persistente Datenbestände und Jetty führt die Anwendungslogik aus.

1.2.2. Model-View-ViewModel

Das Model-View-ViewModel (MVVM) ist eine Variante des Model-View-Controllers (MVC). Das MVVM soll auch wie das MVC spätere Änderungen oder Erweiterungen erleichtern und die Wiederverwendbarkeit der einzelnen Komponenten ermöglichen. MVVM unterscheidet sich zum MVC durch den Datenbindungsmechanismus.

Modell Erfasst die Daten und Inhalte und beschreibt ihre Struktur

View Zuständig für die Oberfläche um Daten des Modells vom Benutzer graphisch einzusehen.

ViewModel Sorgt für die Auswertung der Benutzereingabe die am View gemacht werden.



Abbildung 1: Model-View-ViewModel umfasst drei Komponenten mit dedizierten Aufgaben

Sie dient außerdem als Bindeglied zum Model und ruft deren Dienste und tauscht Daten aus, falls diese über die View eingefordert werden.

Das ViewModel hat keine Kenntnis über die View und stellt lediglich die Attribute und Methoden zur Verfügung um eine Datenbindung eingehen zu können.

1.2.3. Flux-Pattern

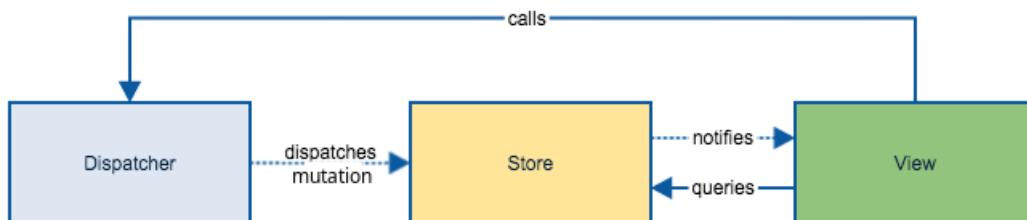


Abbildung 2: Datenfluss im Flux-Pattern

Flux ist ein Architekturmuster welches unidirektionalen Datenfluss fordert.

Wenn ein Benutzer mit einer **View** interagiert, dann propagiert die **View** dieses Ereigniss an den zentralen **Dispatcher**.

Der **Dispatcher** stellt eine Schnittstelle für die **View** um Änderungen am **Store** vornehmen zu können. Im zentralen **Store** liegen lokale Daten die von der **View** erfragt werden können und über den **Dispatcher** geändert. Empfangene Ereignisse vom **Dispatcher**

1. Softwarearchitektur

werden verarbeitet und die daraus resultierenden Daten werden von allen **Views** übernommen.

1.3. Architektur

Im folgenden wird erörtert welche Technologien behelfsmäßig zur Umsetzung der Architektur zum Einsatz kommen. Die gewählten Technologien gehören zum aktuellen Entwicklungsstandard in der Webanwendung und werden ausführlich erklärt.

1.3.1. Relationale Datenbank

Eine relationale Datenbank ist eine Sammlung von Relationen (mathematische Beschreibung einer Tabelle), in denen Datensätze (Tupel), die aus einer Reihe von Attributwerten bestehen, abgespeichert sind.

Relationale Datenbanken sind stets konsistent, eindeutig strukturiert und redundanzfrei (Normalisierung). Dadurch wird die Wartung vereinfacht und Datenkonsistenz gewährleistet.

Einsatz

1. MariaDB ist die Realisierung einer relationalen Datenbank und kommt in ChatSphere zum Einsatz. Siehe Kapitel 5.
2. Die Relationen sind im selben Kapitel in den Abbildungen 45 bis 49 beschrieben.

1.3.2. ORM

Das 'object-relational mapping' (ORM) vermittelt zwischen einer relationalen Datenbank und einer objektorientierten Software. Ziel ist es den konzeptionellen Widerspruch zwischen dem Verwalten von Daten einer OOP und dem Ablegen von Daten in Tabellen bei relationalen Datenbanken aufzulösen.

1.3.3. GraphQL

GraphQL ist eine Abfragesprache zur strukturierten Abfrage von Daten, die ein Typsystem zur Beschreibung der API enthält und eine Laufzeitumgebung zur Ausführung der Abfragen auf einem Server anbietet. Dabei besteht eine Abfrage aus mindestens einer der folgenden Operationen:

- Query: Eine lesende Anfrage
- Mutation: Eine Anfrage zum Schreiben/Verändern von Daten
- Subscription: Client schickt lesendes Abonnement auf Änderungen und Server informiert mit neuen Daten (Push)

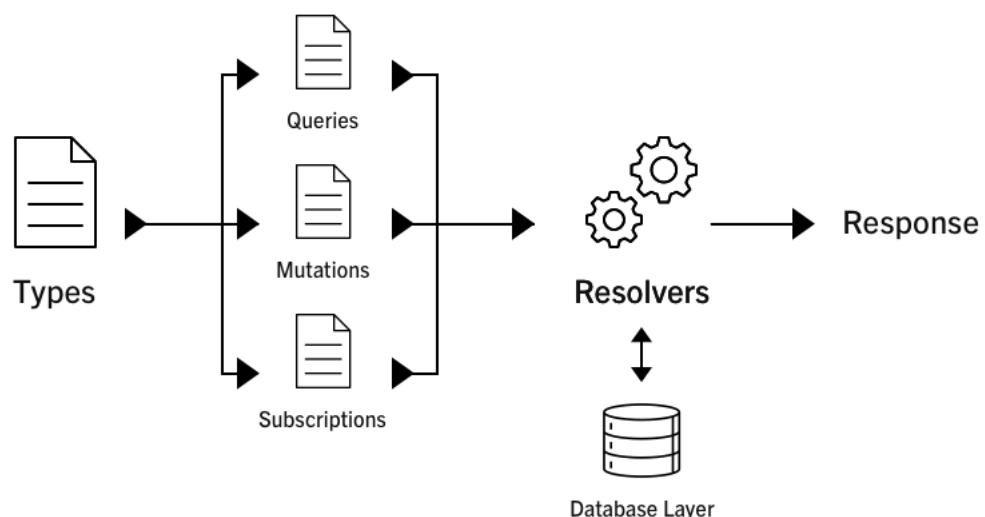


Abbildung 3: GraphQL Funktionsweise

Eine Operation kann Parameter und einen Rückgabewert haben. Das Typsystem wird von GraphQL verwendet um Anfragen zu validieren und ist eine Grundlage für Tool-Support (Syntax Checking, Auto completion, Dokumentation, ...). Es besteht aus skalaren Typen, Objekten, Interfaces, Listen, Enums, ... Die Laufzeitumgebung kümmert sich um das Parsen, Validieren von Client-Anfragen und Serialisieren der Antworten, wobei es oft Daten ermittelt (z.B. aus Datenbank) oder Authentifizierungen durchführt (z.B. über

1. Softwarearchitektur

oAuth Server). Vorteile von GraphQL sind:

- Perfektes Fetching: Weder Overfetching (man bekommt nicht mehr Daten als man braucht) noch Underfetching (man bekommt alle Daten mit genau einem Request)
→ wichtig bei schlechten Verbindungen (z.B. mobil)
- Typisierte API Beschreibung: Automatische Validierung durch GraphQL, Fehlermeldungen in der Entwicklung und zur Laufzeit
- Subscriptions: Kein Long-time polling o.ä. notwendig

Einsatz

1. Die vollständige GraphQL-API ist in Kapitel 3.2 beschrieben.

1.3.4. SPA

Eine Single-Page Application (SPA) ist eine Webanwendung, die aus einem einzigen HTML-Dokument besteht und deren Inhalten dynamisch durch Interaktion mit dem Benutzer nachgeladen werden. Die verstärkte clientseitige Ausführung der Webanwendung ermöglicht Serverlast zu reduzieren und selbstständige Webclients umzusetzen.

Vorteile einer SPA sind:

- Weniger Client-Server-Roundtrips
- Verringerte Wartezeiten durch Verzicht auf Navigation zwischen verschiedenen Webseiten
- Keine Unterbindung laufender clientseitiger Vorgänge oder WebSocket-Verbindungen
- Der Sitzungszustand kann Clientseitig gespeichert werden. Dies erspart die künstliche Emulation von Sitzungszuständen auf Basis von Cookies.

Einsatz

1. Softwarearchitektur

1. Die Anwendung ist in mehrere Client-Module unterteilt, welche Oberflächen bereitstellen zwischen denen navigiert werden kann. Siehe Abbildung 17
2. Um Inhalte dynamisch laden zu lassen behilft man sich mit einem Page-Router. Erforderlich dafür ist eine Beschreibung der URL-Pfade wie in Abbildung 42

1.3.5. PWA

Eine Progressive Web App (PWA) ist eine Website, die Merkmale einer nativen App besitzt. PWAs können wie eine Website mit HTML5, CSS3 und Javascript realisiert werden. Die Service Worker einer PWA bringen Offline-Funktionalitäten durch Caching.

Funktionsweise einer PWA:

- Der Benutzer startet eine PWA, indem er in einem Browser die URL des Webservers eingibt.
- Der Benutzer bekommt eine auf sein Endgerät angepasste Webseite zu sehen.

Eine PWA bietet mehrere Funktionalitäten an, die auf Progressive Enhancement Technologie beruhen:

- Add-To-Homescreen: Der Benutzer kann die PWA als Icon zu dem HomeScreen seines Smartphones hinzufügen, um zu einem späteren Zeitpunkt durch Klicken auf das Icon die PWA wieder zu öffnen.
- Push Notifications: Benutzer können auf ihrem Handy Benachrichtigungen erhalten, wie sie aus Apps bekannt sind. PWA's verfolgen konsequent einen Mobile First Ansatz.
- Offline-Modus: Mit der Caching Funktion der Service Worker einer PWA stehen einmal abgerufene Inhalte auch offline zur Verfügung. PWA's verfolgen konsequent einen Offline First Ansatz.

Eine PWA hat folgende Vorteile:

1. Softwarearchitektur

- Progressive: Funktioniert für jeden Benutzer eines Webbrowsers
- Responsive: Auf jedes Endgerät angepasst, egal ob Desktop, Mobil, Tablet oder was auch immer als Nächstes kommt
- Verbindungsunabhängig: Durch Service Worker offline möglich

Einsatz

1. Der Service Worker ist die wesentliche Komponente einer PWA und eine einzige Javascript-Datei deren Lebenszyklus in Abbildung 43 dargestellt ist.

1.3.6. Websockets

WebSocket ist ein neues Protokoll (TCP basiert) für bidirektionale Kommunikationen über HTTP. Im Gegensatz zu einer reinen HTTP-Verbindung, bei der jede Aktion des Servers eine vorhergehende Anfrage des Clients erfordert, reicht es bei einem WebSocket aus, wenn der Client (Webanwendung) die Verbindung öffnet. Der Server (Webserver) kann neue Informationen über diese bestehende Verbindung an den Client ausliefern ohne auf eine neue Verbindung des Clients zu warten. Bei einer reinen HTTP-Verbindung kann der Server den Client bei Neuerungen nur dann benachrichtigen (Subscription), wenn der Client in regelmäßigen Zeitabständen eine neue Verbindung aufbaut und eine Anfrage absendet (long polling). Eine einzelne WebSocket Nachricht kann jede Größe annehmen und es können unbegrenzt viele Nachrichten ausgetauscht werden. Nachrichten werden dabei nur sequenziell versendet. Bei einer WebSocket Verbindung wird zuerst ein Handshake durchgeführt, der ähnlich zu einem HTTP-Header ist und vollständig abwärtskompatibel zu diesem ist.

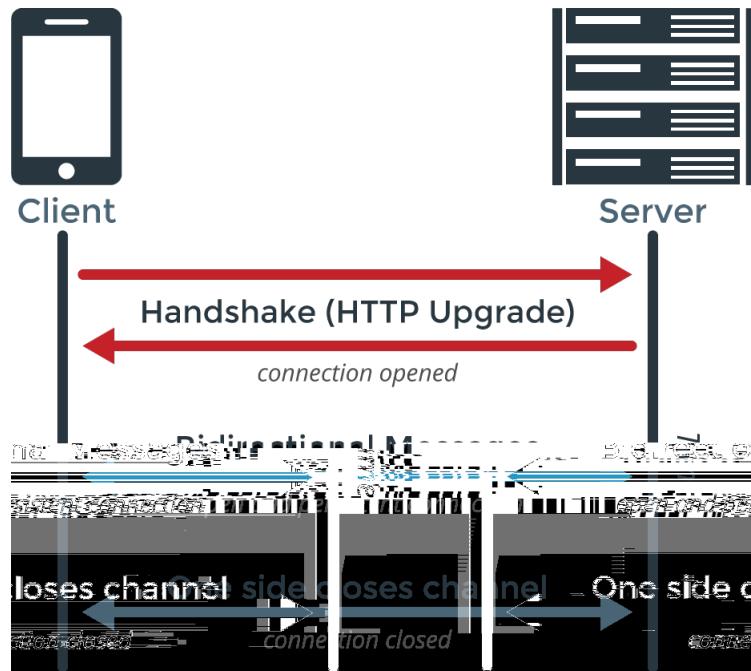


Abbildung 4: WebSocket Funktionsweise

1.3.7. Data Management

In ChatSphere werden Daten an drei unterschiedlichen Orten verwaltet:

- Beim Server in der Datenbank (MariaDB)
- Beim Client im Arbeitsspeicher (In-Memory Cache)
- Beim Client im Festspeicher (Vuex-Persist, Apollo-Persist-Cache)
- Die App-Shell im Festspeicher: Der statische Client ohne dynamische Daten (vom Server)

Dabei sind folgende Daten zu unterscheiden:

- Interne Anwendungsdaten: Daten die beispielsweise zwischen Komponenten ausgetauscht werden oder allgemein nur der Zustand der Applikation betreffen nicht aber den Zustand der Benutzerdaten.

1. Softwarearchitektur

- Benutzerdaten: Daten zwischen Client und Server ausgetauscht werden und wichtig für den Benutzer sind nicht aber für die Applikation selbst
- Applikation: Das ist die gesamte Applikation die gemäß PWA gespeichert wird, um einen App ähnlichen Offline-Modus zu ermöglichen

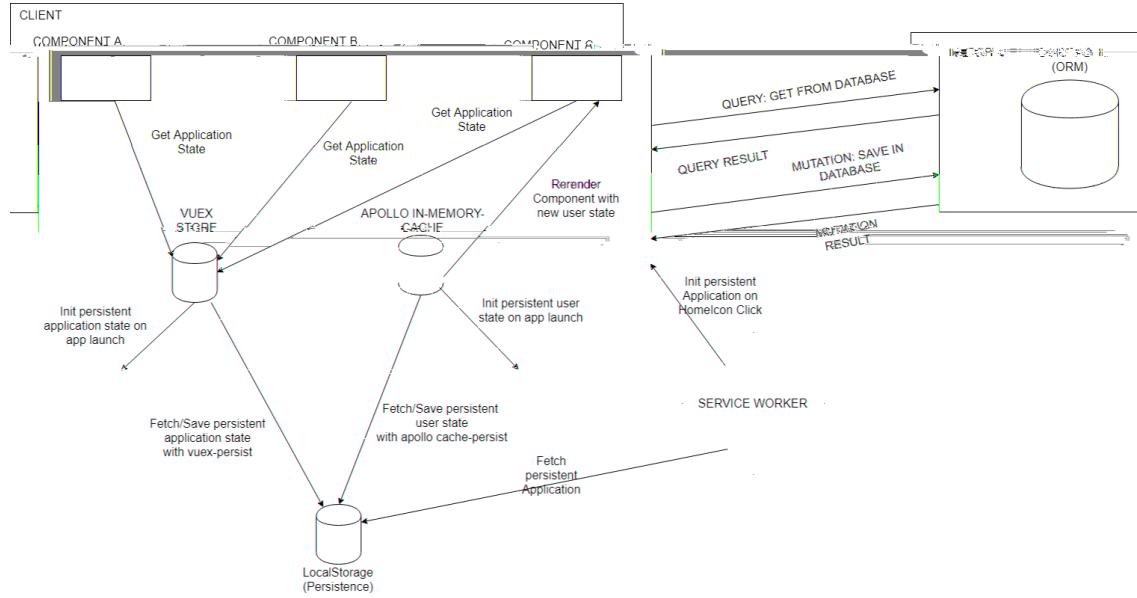


Abbildung 5: Data Management ChatSphere

Der Service Worker speichert eine App-Shell persistent im lokalen Speicher des Endgeräts und holt diese bei Abruf der Anwendung im Browser hervor. Die Funktionalität der App-Shell (Apollo-cache-persist oder Vuex-persist) überprüft zunächst, ob Daten von einer vorherigen Ausführung der Anwendung im lokalen Speicher vorliegen. Falls dies der Fall ist, werden diese ausgegeben. Ansonsten werden neue Daten per GraphQL Query vom Server angefragt. Diese holt der Server aus der Datenbank, die diese Daten persistent speichert. Der Client kann auch Daten per GraphQL Mutation an den Server schicken, die der Server dann in der Datenbank speichert. Nachdem die Daten vom Server an den Client übergeben wurden, speichert dieser die Daten im Arbeitsspeicher (Apollo-cache), sodass diese während der Ausführung verfügbar sind. Außerdem werden festgelegte Daten auch noch persistent gespeichert (Apollo-cache-persist oder Vuex-persist). Währenddessen ist es für den Client möglich lokale Zustände zwischen Komponenten oder lokale Anwendungszustände im Arbeitsspeicher abzulegen (Vuex oder Apollo-link-state). Nach dem Beenden des Clients wird der Arbeitsspeicher bereinigt, das bedeutet der Apollo-Cache

ist leer.

1.3.8. Ablauf eines GraphQL-Query

Der Client (ApolloClient) sendet eine nach dem GraphQL SDL konforme Query Anfrage serialisiert als String über eine WebSocket-Adresse an den Server. Dieser authorisiert zunächst die Anfrage und konvertiert daraufhin den empfangenen String in einen AST (Abstract Syntax Tree). Außerdem validiert er diesen beim Konvertieren. Ein validier AST wird dann den Feldern nach rekursiv bearbeitet, indem für jedes Feld ein vom Server definierter Resolver die benötigten Daten holt. Dies geschieht meist über die Datenbank. Um auf den Datenbestand zuzugreifen benutzt er die ORM Schnittstelle und die darin definierten DAO's (Database Abstract Object). Über diese kann er auf die abgebildete relationalen Datenbank (MariaDB) zugreifen, die den physischen Datenbestand verwaltet. Sobald er die Daten erhalten hat, sendet er diese in der bestehenden WebSocket Verbindung die Daten in JSON serialisiert an den Client (ApolloClient) zurück.

1.3.9. GraphQL-Websocket-Transport

2. Einführung

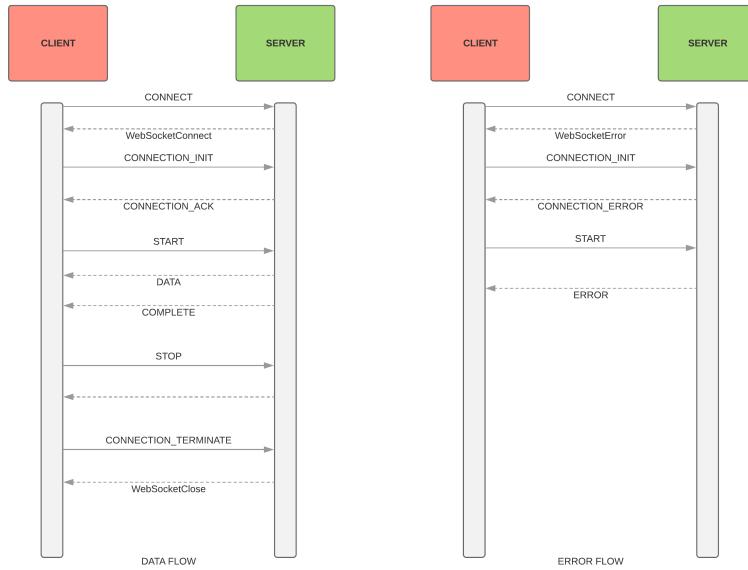


Abbildung 6: GraphQL transport over websocket specification

2. Einführung

Der → Instant Messaging Dienst ChatSphere wird als → Webanwendung mit zeitgemäßen und zukunftsträchtigen Technologien als → Client-Server-Applikation realisiert. Dieser Architekturstil sieht eine zentrale Anbieterinstanz (Server) vor, welcher Anfragen (Requests) der einzelnen Endgeräte (Clients) in zustandslosen Antworten (Response) beantwortet.

Entsprechend kommt es zur fundamentalen Aufteilung zwischen:

- **Client:** Eine in → JavaScript programmierte → Progressive Web Application als Frontend basierend auf dem → Framework VueJS.
- **Server:** Ein in → Java geschriebener Backend- → Servlet basierend auf der GraphQL-Java → Library .
- → **API :** Eine in → GraphQL SDL spezifizierte → Schnittstelle für die Kommunikation zwischen Client und Server.

2. Einführung

- **Datenbank:** Ein in → SQL definiertes Schema mit welchem auf einem Datenbankserver Anwendungsdaten → persistent gespeichert werden.

In der Praxis ist die Berücksichtigung weiterer Aspekte wie einer Transportverschlüsselung, verschiedener → Policies und Standards sowie nicht zuletzt die Interaktion mit Diensteanbietern notwendig, weswegen beim Deployment ein → Reverse-Proxy und ebenso ein Authorisierungsserver für den Anmeldevorgang eingesetzt wird. Sie gehören zur Realisierung sind allerdings nur Nebenschauplätze der Entwicklung.

2.1. Softwareartefakte

Aus dieser Architektur ergeben sich unweigerlich die einzelnen bei der Entwicklung entstehenden oben genannten Artefakte: Client-Ressourcen, Servlet, API-Spezifikation, Datenbankschema und Konfigurationsdateien. Intern erreicht der Reverse-Proxy die einzelnen Server über verschiedene → Ports , hierbei den Authorisierungsserver an 4444 und den Jetty-Server an Port 6000.

2.2. Programmablauf

Das besprochene Design-Pattern - die Client-Server-Architektur aus Kapitel 1.2.1 - kommt hier zum Einsatz und wird nochmal detaillierter modelliert:

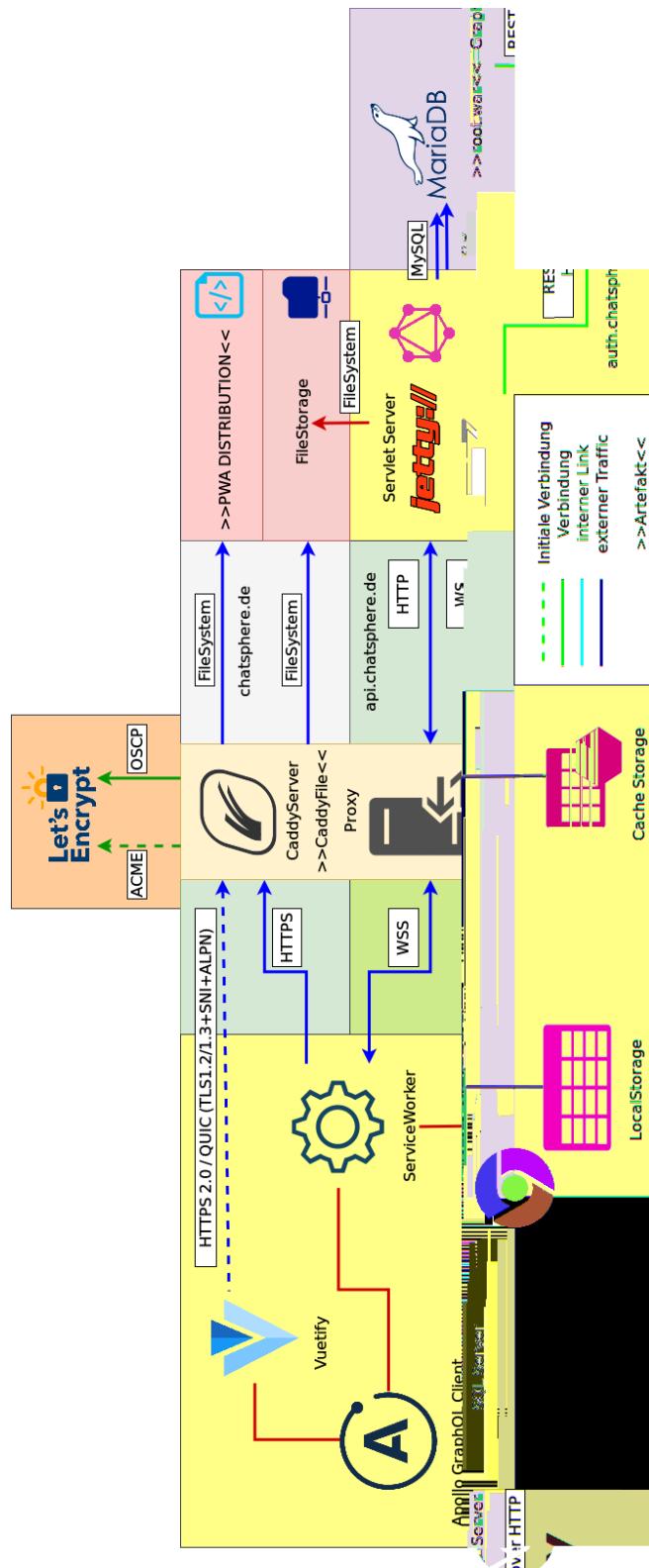
Clientseite und Serverseite von ChatSphere lassen sich in Module unterteilen die in Abbildung ?? und 9 dargestellt sind.

Jedes Modul umfasst einen Aufgabenbereich. Auf der Clientseite kommt das MVVM-Muster wie es in Kapitel 1.2.3 eingeführt wurde zum Einsatz. Wie in der Abbildung gezeigt wird können Komponenten zu einen der Kategorien des MVVM-Musters (View, ViewModel und Model) zugeordnet werden:

1. Alle Module die sich dem View zuordnen lassen sind in Abbildung ?? mit dem

2. Einführung

Abbildung 7: Frontend: »VueJS« mit »Apollo GraphQL Client« interagierend mit Webbrowser-APIs, veranschaulicht durch das »Google Chrome Logo«. Backend: Hinter dem Reverse-Proxy »CaddyServer« gekapselter »Jetty-Server« der das Java-Servlet ausführt, interagierend mit dem Dateisystem, der Datenbank »MariaDB« und dem Authorisierungsserver »ORY HYDRA«



2. Einführung

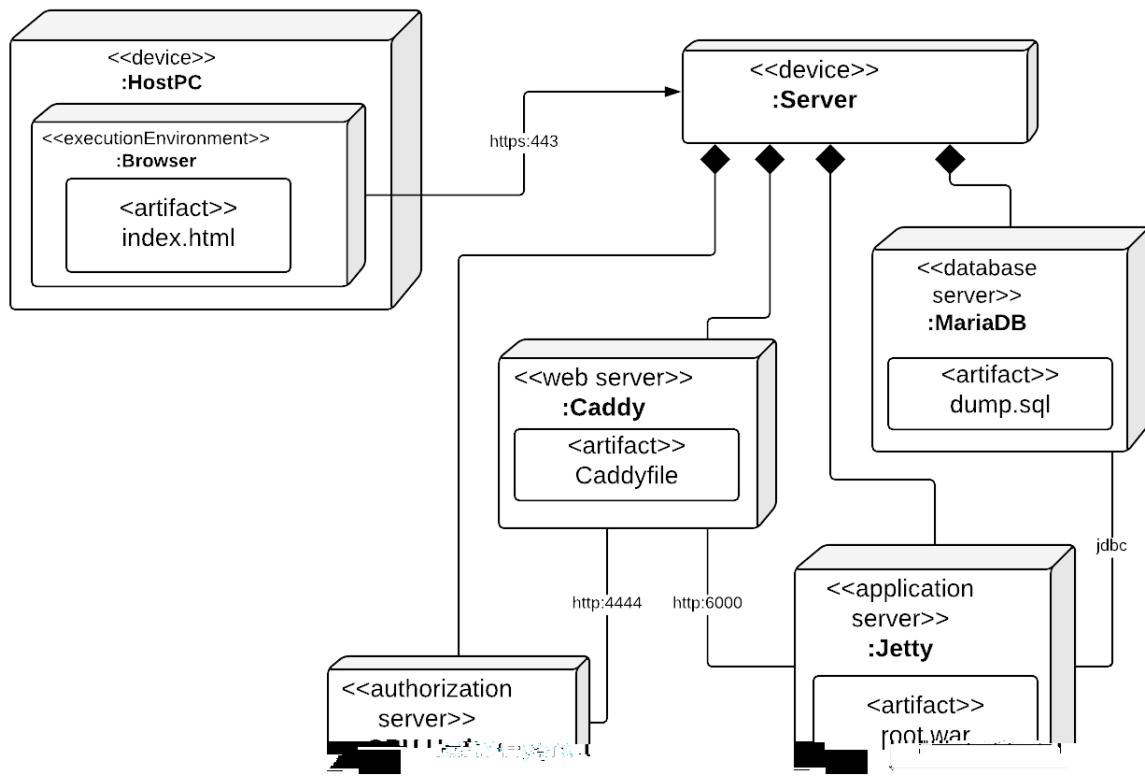


Abbildung 8: Verteilungsdiagramm - ChatSphere-Artefakte auf den beteiligten Geräten

«View»-Stereotyp gekennzeichnet und werden im Folgenden als Client-Modul bezeichnet. Jedes einzelne der Client-Module konsumiert eine Teilmenge der Schnittstelle welche von der UI-Logic bereitgestellt wird.

2. In der **ViewModel**-Box von Abbildung ?? aufgefasst: UI-Logic, die zu allen Views zusammengefasste Logik die erforderlich ist zur Verarbeitung auf Clienseite..
3. Die Komponente mit dem «Store»-Stereotyp die noch Teil des ViewModels ausmacht ist der zentrale Store des Flux-Patterns (Siehe Kapitel 1.2.3). Dieser ist mittels Apollo-link-state realisiert ist, bietet die Schnittstellen zur Verwaltung der lokalen Daten der Anwendung. Falls Daten nicht lokal vorhanden sind wird die Anfrage über das integrierte Vue-apollo Modul (zuständig für GraphQL Anfragen) weiterleitet und vom Server verarbeitet.

2. Einführung

Die Serverseite bietet mit dem GraphQL Server einen Endpunkt zur serverseitigen Verarbeitung der Daten an. Der GraphQL Server delegiert die Arbeit für die Verarbeitung der Daten und die Auswertung der Semantik an die anwendungsspezifischen Klassen die als "Business Logic" zusammengefasst sind.

Für alle Funktionalitäten, die die Persistenz der Daten erfordern, ist das "Data Storage"-Modul verantwortlich.

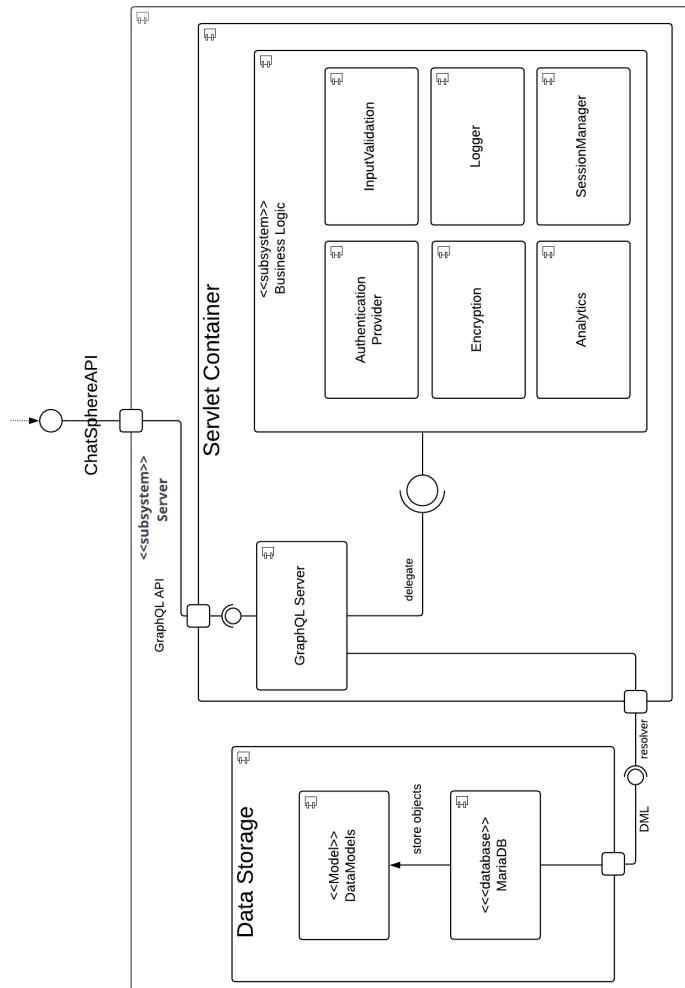


Abbildung 9: Serverseite von ChatSphere

3. API

3. API

Frontend und Backend kommunizieren über eine einheitliche API-Schnittstelle miteinander. Beim API-Typ wurde GraphQL gewählt und die API wird in der GraphQL Schema Definition Language (GraphQL SDL) spezifiziert, da hierbei das Schema einer Abfrage vom Client vorgegeben wird, welcher in einem einzelnen Roundtrip sein Informationsbedürfnis stillen kann, im Gegensatz zu verbreiteten Representational State Transfer (REST) Schnittstellen, deren Definition in OpenAPI 3.0 spezifiziert wird, bei denen die Form der Antworten fest spezifiziert ist.

CORS Moderne Webbrower implementieren die Same-Origin-Policy (SOP) wodurch Domainübergreifende Anfragen verweigert werden. Um diese dennoch zu gestatten, ist die Implementierung des Cross Origin Ressource Sharing (CORS) Protokolls erforderlich. Die Unterstützung hierfür wird vom vorgeschalteten Reverse-Proxy übernommen.

3.1. Authorisierungsserver

Die Durchführung einer Authentifizierung ist im oAuth 2.0 Protokoll vereinheitlicht, welches für die Anmeldung an der API zum Einsatz kommt. Die Implementierung des Protokolls übernimmt der Authrosierungsserver ORY HYDRA, welcher eine REST-API bereitstellt und den Consent-Flow spezifiziert.

Consent-Flow

1. **Initialisierung** die Anwendung startet die oAuth 2.0 Authorisierung mit einem besuchen der Issuer-URL (Adresse des Authorisierungsservers), welcher daraufhin den Benutzer - falls keine Sitzung besteht - zum Login-Provider weiterleitet.
2. **Benutzeroauthentifizierung** Der Login-Provider fragt die Login-Informationen ab, führt die Authentifizierung durch, authorisiert den Benutzer und leitet diesen weiter an die von der API des Authorisierungsservers vorgegebene Adresse.
3. **Anwendungsauthentizierung** Der Consent-Provider fragt die Anwendungsbe rechtigungen ab, authorisiert die Anwendung und leitet den Benutzer weiter an die

3. API

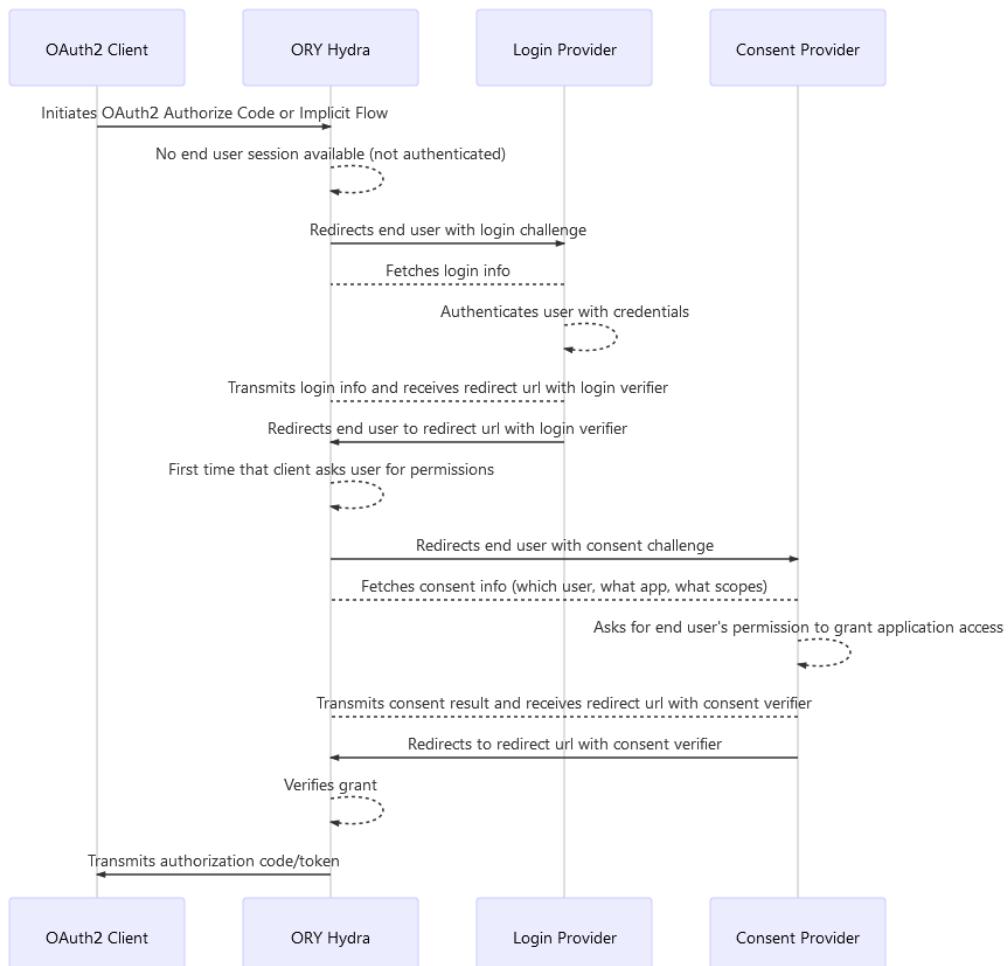


Abbildung 10: ORY HYDRA Consent-Flow (Apache 2.0) aus den ORY Hydra Docs

durch den Authorisierungsserver vorgegebene Adresse.

4. **Authentifizierung** Der Authentifizierungsserver erteilt die Berechtigung und leitet dem Benutzer von der zu autorisierenden Applikation festgelegten Adresse weiter.

Für die Realisierung des Consent-Flow müssen 3 Adressen bereitgestellt werden, um die Implementierung des oAuth 2.0 Protokolls durch ORY Hydra zu realisieren.

- **Login Provider**, welcher die oAuth 2.0 Attribute beim Authorisierungsserver abfragt, den Benutzer authentifiziert (durch i.d.R. ein Anmeldeformular) und das

3. API

Ergebnis der Benutzerauthentifizierung dem Authorisierungsserver mitteilt.

- **Consent Provider**, welcher das gleiche wie der Login-Provider durchführt mit dem Unterschied, dass die Berechtigungen der Anwendungen authentifiziert werden.
- **Error Provider** eine Seite welche im Fehlerfall aufgerufen wird.

Bei anschließenden Anfragen an den API-Server wird an diesen ein Token mitgesendet. Beim Authentifizierungsserver können über dessen REST-Schnittstelle Informationen zu diesem, insbesondere zu dessen Gültigkeit, abgefragt werden.

3.2. GraphQL-API

GraphQL ist der Objektorientierten Modellierung entlehnt, entsprechend sind Interfaces, Enums, Vererbungen und Assoziationen möglich.

Die GraphQL-API ist in drei unterschiedliche Abfrage-Typen unterteilt, wodurch sich drei voneinander getrennte Diagramme gleicher Objektbasis ergeben.

- **Query** behandelt einmalige Lesezugriffe
- **Mutation** behandelt Datenmanipulationen
- **Subscription** behandelt Benachrichtigungen bei Änderungen

Grundlegend ist zu beachten, dass ein Aufruf der API immer durch einen konkreten Benutzer erfolgt und sämtliche Anfragen in seinem Kontext beantwortet werden. Entsprechend genügt es, dass der Benutzer Zugang zu seinem eigenen Account-Objekt hat und nur die öffentlichen Profile anderer Benutzer im »User« Objekt betrachten kann, mit denen er (wie auch in der Datenbank modelliert) als Kontakt in Beziehung stehen kann.

Chattypen und Chatnachrichten werden als Interface mit verschiedenen konkreten erweiternden Implementierungen angesehen. Um die Erweiterbarkeit zu gewährleisten, wurden

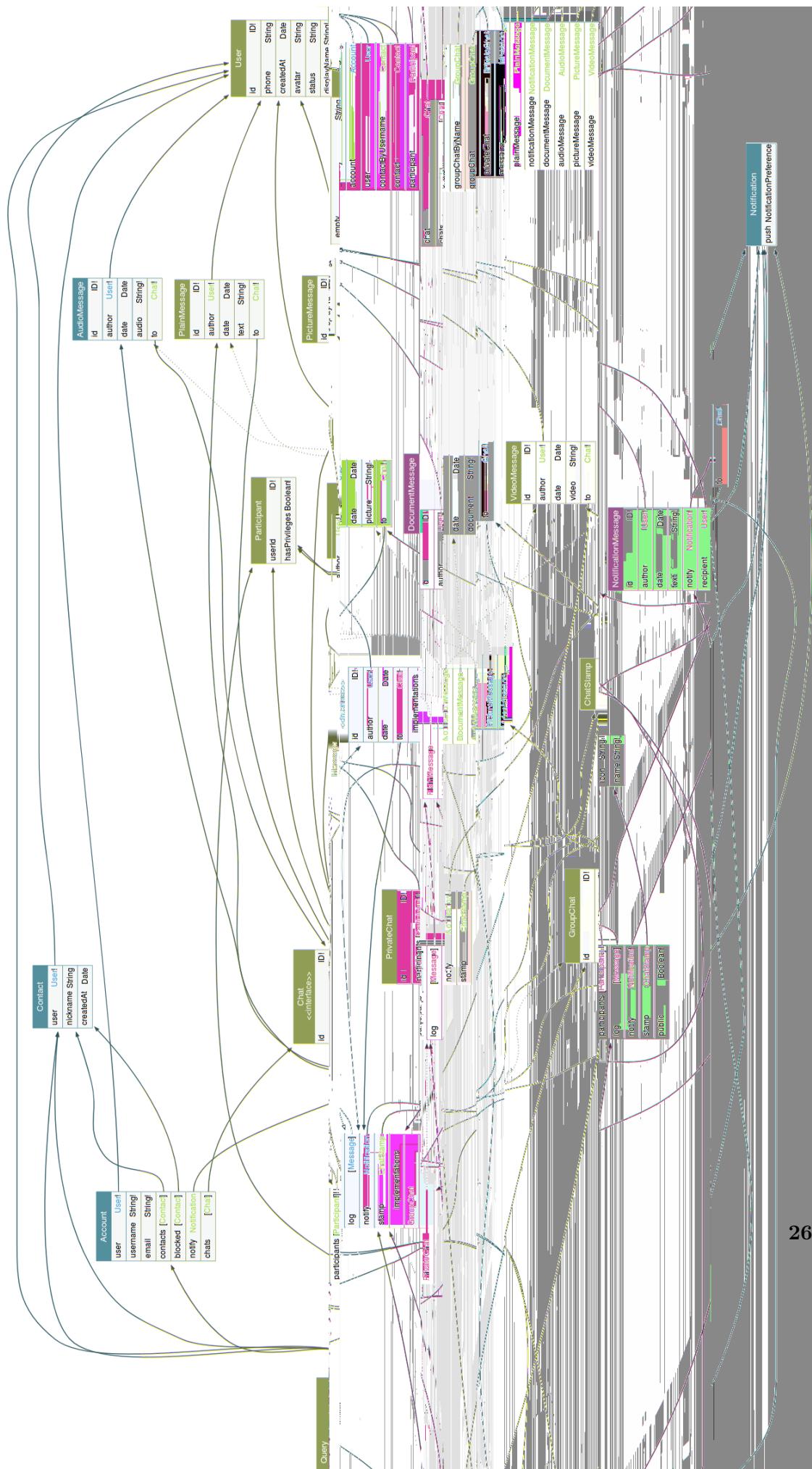
3. API

zwischen Chats und Benutzern ein Teilnehmerobjekt eingeführt, wodurch zukünftig teilnehmerspezifische Attribute hinzugefügt werden können.

Ebenso wird im Falle der privaten Chats der Chatname des Gegenübers angezeigt. Die Logik hierfür wurde allerdings auf den Server durch den Einsatz eines ChatStamp-Objektes ausgelagert, wodurch bei Abfragen keine Fallunterscheidung der Chattypen getroffen werden muss und zugleich mehr Freiraum für zukünftige etwaige Anpassungen geschaffen wird.

Im Falle der verschiedenen Nachrichtentypen wird die Logik für Unterschiedungen zwischen den Typen auf den Client ausgelagert, da dieser notwendigerweise Audio, Video und Bildinhalte voneinander unterscheiden und verschieden darstellen muss, ebenso wahlweise verschiedene Parameter angeben darf.

3. API



3. API



Abbildung 13: API Subscription Object Graph

4. Frontend

In diesem Kapitel werden im Detail, die auf der Clientseite existierenden Komponenten und deren Zusammenhang beschrieben

4.1. Apollo-link-state

In Kapitel 1.2.3 wurde das Flux-Pattern eingeführt. Der zentrale Store ist eine Komponente die Teil des ViewModels ausmacht und nun für unsere Anwendung hier modelliert wird.

Üblicherweise würden lokale Anwendungsdaten in einem separaten VueX Store gespeichert und verwaltet werden. Mit der Integration von vue-apollo zum Aufrufen von Serverdaten bietet es sich an die gleichnamige Bibliothek Apollo-link-state zu verwenden welches sich nahtlos mit vue-apollo integrieren lässt.

Apollo-link-state übernimmt die Realisierung für einkommende Anfragen und Änderungen im Store. Es genügt die Daten die zum Bestand gehören zu definieren und das Verhalten bei Schreiboperationen zu definieren. Zum Aufrufen oder zum Aktualisieren der lokalen Daten benutzt man GraphQL queries und mutations so wie man es mit Serverdaten schon macht.

4.1.1. Registrierung

Aufgabe Die Aufgabe des zentralen Stores in diesem Modul ist das Zwischenspeichern von Daten des Registrierungsprozesses. Dazu wird eine Schnittstelle angeboten damit das Registrierungs Modul Name, Passwort und Email abspeichern kann. Das Registrierungs Modul braucht nicht mehr die Eingaben zwischen den Komponenten weiterzuleiten und zu koordinieren sondern erfragt dann den zentralen Store als Quelle seiner Daten.

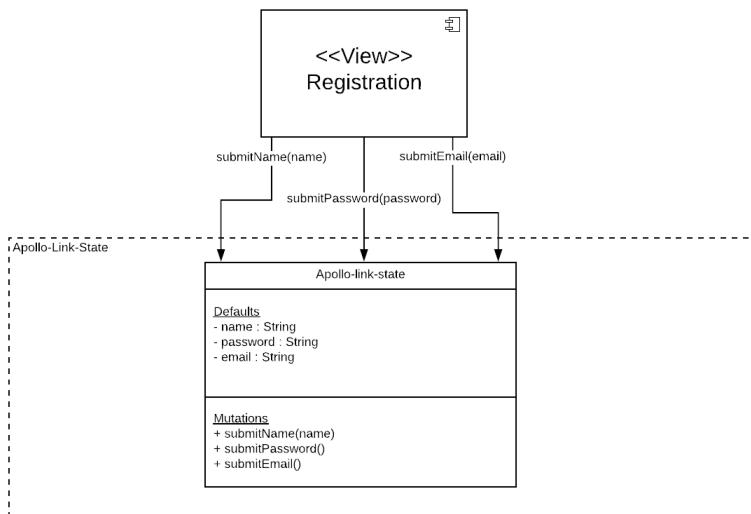


Abbildung 14: Apollo-link-state. Übernimmt die Aufgaben des zentralen Stores im Flux-Pattern.

UI-Logic

- **name**: Speichert Benutzername des Benutzers.
- **password**: Speichert Passwort des Benutzers.
- **email**: Speichert Passwort des Benutzers.
- **submit*()**: Die lokal gespeicherte Eingabe wird überschrieben

4.1.2. Search

Aufgabe Problem ist dass das Search Modul und die Toolbar in der Komponentenhierarchie in keiner Weise miteinander verwandt sind. Da jedoch der Benutzer die Sucheingaben in der Toolbar macht - aber die Ergebnisse im Search Modul angezeigt werden, muss der Apollo-link-state vermitteln.

4. Frontend

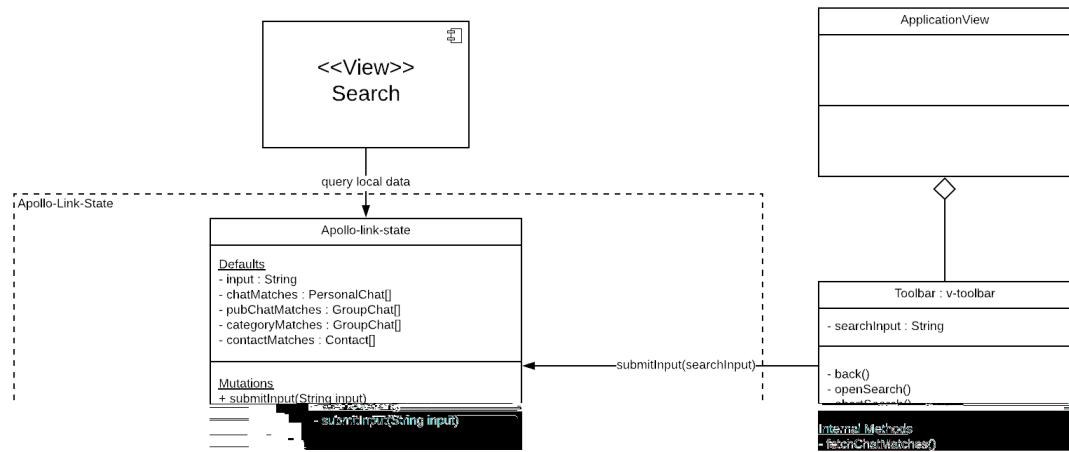


Abbildung 15: Apollo-link-state als Vermittler zwischen Komponenten.

UI-Logic

- **input:** Die Eingabe des Benutzers
- **chatMatches:** Gefundene Übereinstimmungen für eigene offene Chats
- **pubChatMatches:** Gefundene Übereinstimmungen für eigene Gruppenchats
- **categoryMatches:** Öffentliche Chats deren Name/Kategorie mit dem Suchbegriff übereinstimmen
- **contactMatches:** Benutzer deren Name mit dem Suchbegriff übereinstimmen
- **submitInput():** Jede einzelne Eingabe mit der Tastatur, die ein Benutzer macht löst die `submitInput` Methode aus die im Store die Eingabe aktualisiert um nach neuen Übereinstimmungen zu suchen.

4. Frontend

- **fetch***(): Internal Methods sind Methoden die nicht über ein Mutation Query aufgerufen werden können sondern nur Logik kapseln. Jeder Aufruf von `submitInput` soll nach neuen Ergebnissen im Server suchen, die von diesen Methoden übernommen werden. Das Search Modul beobachtet gefundenen Ergebnisse und macht gegebenenfalls einen Query um sie anzuzeigen.

4.2. Client-Module

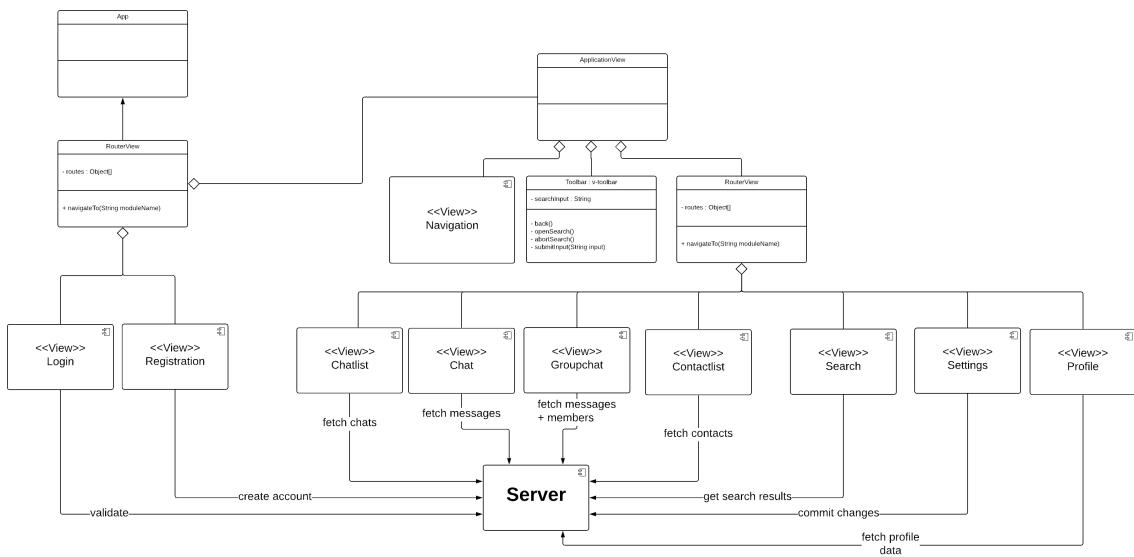


Abbildung 16: Komponentendiagramm des Frontends

Abbildung 16 listet alle Client-Module auf. Die Client-Module sind in einer Hierarchie angeordnet mit `App` als Container aller Client-Module. Der Benutzer navigiert im Laufe der Betriebszeit zwischen den Client-Modulen.

Neben den Client-Modulen zeigt Abbildung 16 die `RouterViews`. `RouterViews` sind Platzhalter - sinngäbliche "Steckdosen" für Client-Module. Sie umfassen mehrere Client-Module von denen sich ein einzelnes ausgewähltes Client-Modul rendern lässt. So wird, wie in der Abbildung dargestellt, das Login- oder Registrierungs-Modul jeweils isoliert dargestellt. Alle andere Client-Module welche die `ApplicationView` ausmachen sind nur im eingeloggtem Zustand sichtbar und werden in Kombination mit einer Toolbar und der Navigationsleiste angezeigt.

4. Frontend

Die Client-Module müssen die ursprünglichen Daten die darzustellen sind vom Server erfragen. Hierbei hat jedes Client-Modul individuelle Ansprüche am Datenbestand.

Es gibt folgende Module die nun im folgenden Kapitel näher beschrieben werden:

- Registrierung
- Login
- Navigation
- Profile
- Settings
- Chatlist
- Chat
- Groupchat
- Contactlist
- Search

Jedes der zehn Client-Module umfasst mehrere → View-Komponenten , die für die Darstellung der Daten verantwortlich ist.

In Abbildung 17 werden die Möglichen Wechsel der View Komponenten dargestellt. Dabei werden über Aktionen des Benutzers, beispielsweise öffnen der ChatsSphere website, entsprechende Benutzeroberflächen dargestellt. In Abbildung 17 werden so alle möglichen Aktionen des Benutzers und den resultierenden Wechsel der Benutzeroberfläche beschrieben.

4. Frontend

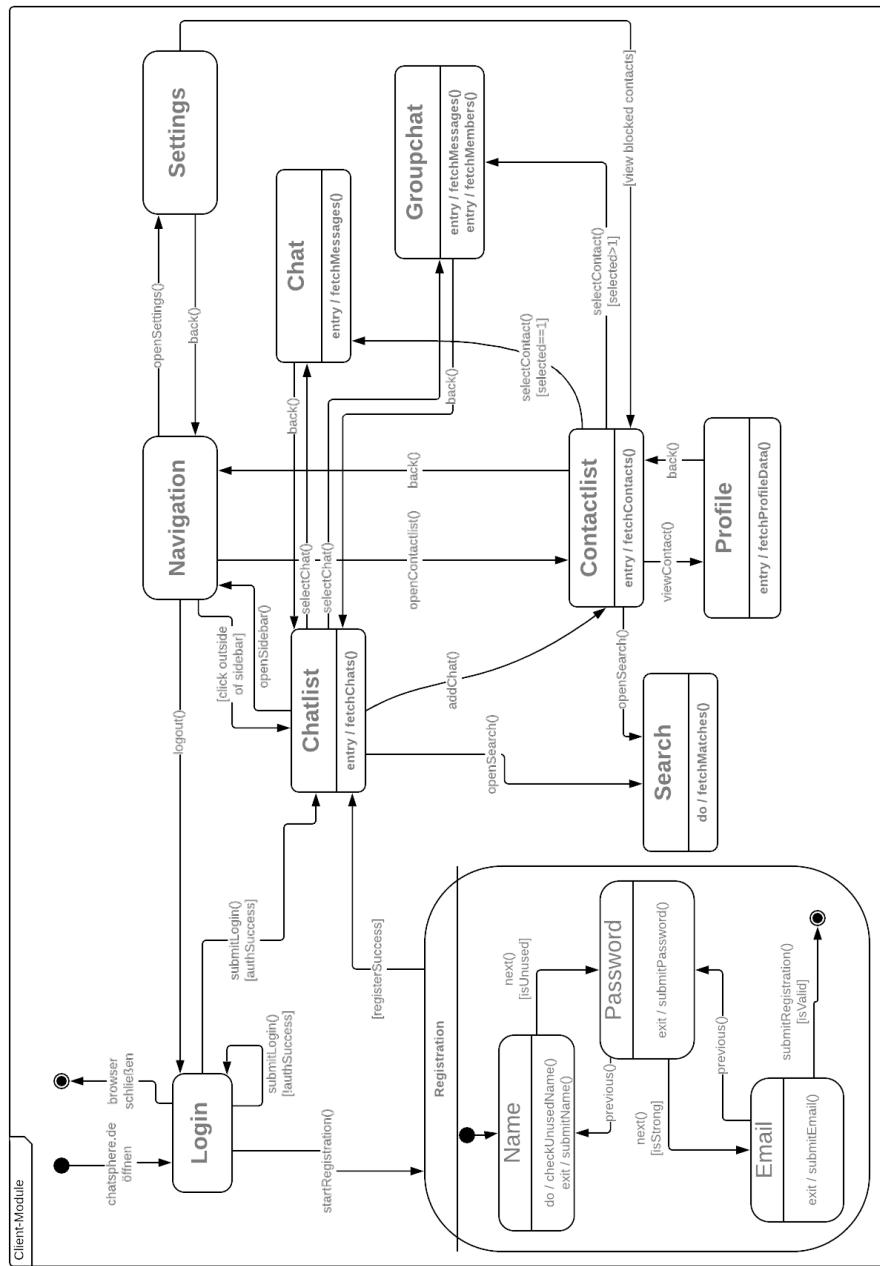


Abbildung 17: Wechsel zwischen Benutzeroberflchen der Single-Page-Application

4.3. Modulbeschreibungen

4.3.1. Registrierung

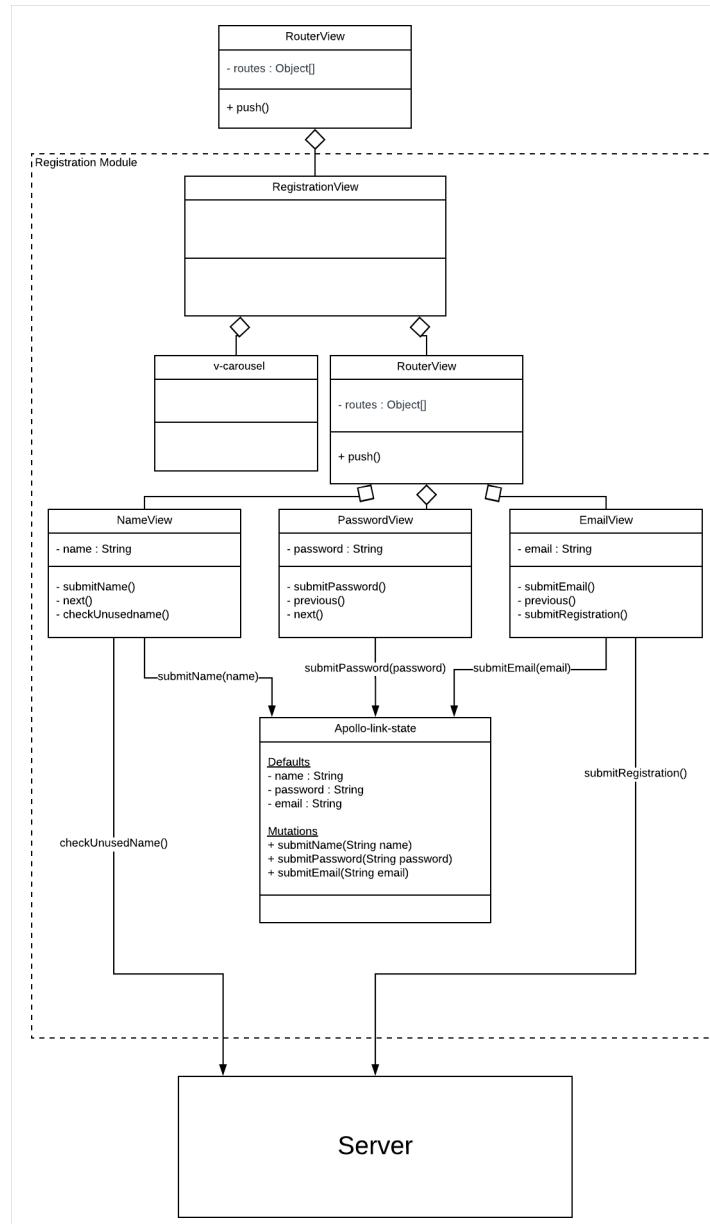


Abbildung 18: Registrierungs Modul. Benutzereingaben werden an den zentralen Store weitergeleitet

4. Frontend

Aufgabe Das Modul sorgt sich um die Registrierung eines Benutzeraccounts. Es leitet den Benutzer durch alle Schritte der Registrierung und gibt dem Benutzer Feedback zu seinen Eingaben. Die Eingaben werden an den zentralen Store weitergeleitet um sie zum Schluss vollständig an den Server zu übermitteln.

View-Komponente Das Registrierung Modul umfasst mehrere View-Komponenten: `NameView`, `PasswordView` und `EmailView` die dem Benutzer jeweils ein Eingabefeld präsentieren.

Die Eingabefelder werden in der UI-Logik der Registrierung an die Attribute `name`, `password`, `email` → gebunden .

Von jeder Komponente kann vor- und zurücknavigiert werden.

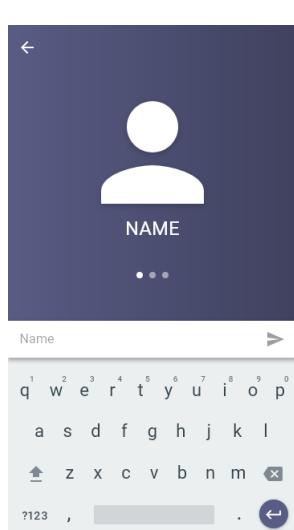


Abbildung 19: NameView

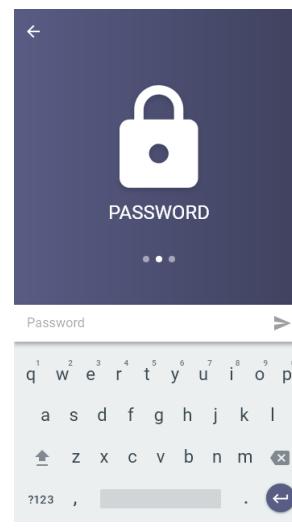


Abbildung 20: Password-View

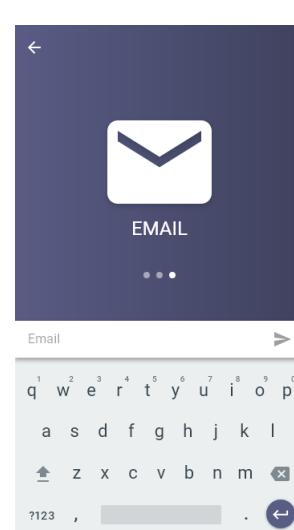


Abbildung 21: EmailView

UI-Logic

- `name`: Speichert Benutzername des Benutzers.
- `password`: Speichert Passwort des Benutzers.
- `email`: Speichert E-Mail des Benutzers.
- `previous()`: Springt einen Schritt im Registrierungsverfahren zurück.
- `next()`: Springt einen Schritt im Registrierungsverfahren weiter.

4. Frontend

- **submit*():** Die eingegebenen Daten werden unmittelbar validiert um auf falsche Eingaben aufmerksam zu machen (Name vergeben, Passwort zu kurz, E-Mail ungültig). Bei Erfolg wird die Eingabe an Apollo-link-state weitergeleitet.
- **submitRegistration():** Im letzten Schritt werden alle Eingaben nochmals validiert und dann an den Server übermittelt um einen neuen Account zu erstellen. Es werden folgende Daten übergeben:

```
CreateAccountInput {  
    email: String!  
    username: String!  
    password: String!  
}
```

4. Frontend

4.3.2. Login

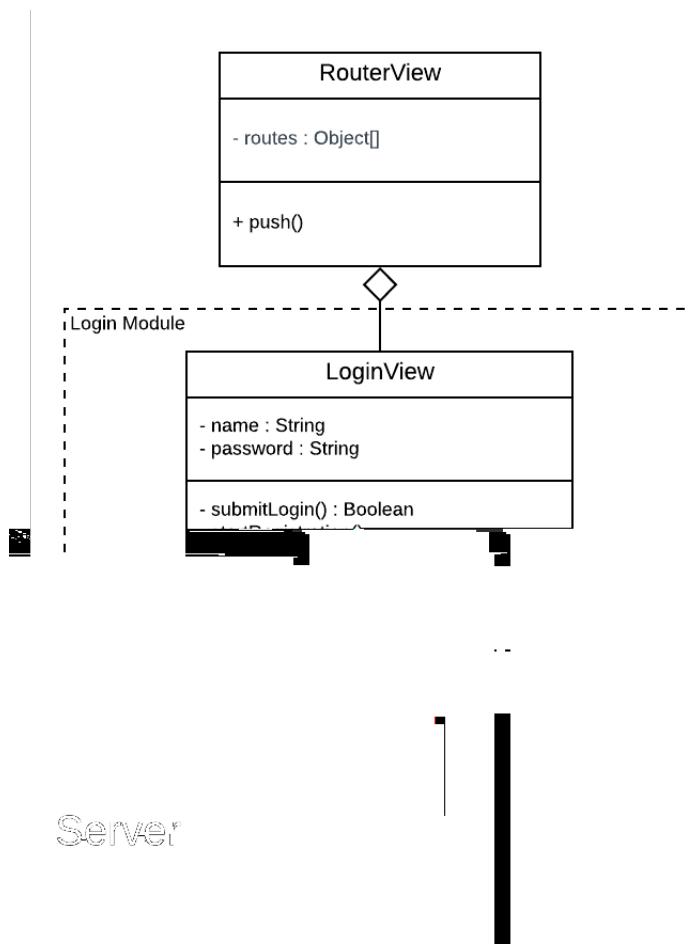


Abbildung 22: Login Modul

Aufgabe Das Login Modul ist dafür zuständig, die Oberfläche der Authentifizierung bereitzustellen. Es ist die erste Oberfläche die der Benutzer beim Start der Anwendung sieht. Eingaben des Benutzers werden zwischengespeichert, clientseitig auf Fehler (z.B. leeres Feld) geprüft und abschließend an den Server zur Verarbeitung weitergeleitet.

4. Frontend

View-Komponenten Das Login Modul umfasst als einziges die "LoginView" View-Komponente. Die Eingabefelder werden in der UI-Logik mit `username`, `password` verbunden.

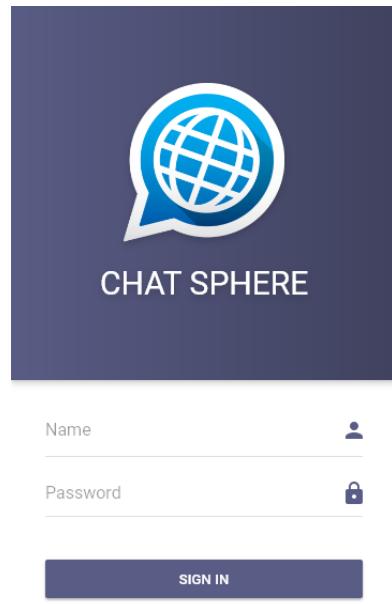


Abbildung 23: Aussehen der LoginView

UI-Logic

- `name`: Speichert Benutzername des Benutzers.
- `password`: Speichert Passwort des Benutzers.
- `startRegistration()`: Navigiert zum Registrierung Modul.
- `submitLogin`: Im letzten Schritt werden alle Eingaben nochmals validiert und dann an den Server übermittelt um den Benutzer einzuloggen. Es werden folgende Daten übergeben:

```
LoginInput {  
    username: String!  
    password: String!  
}
```

4. Frontend

4.3.3. Navigation

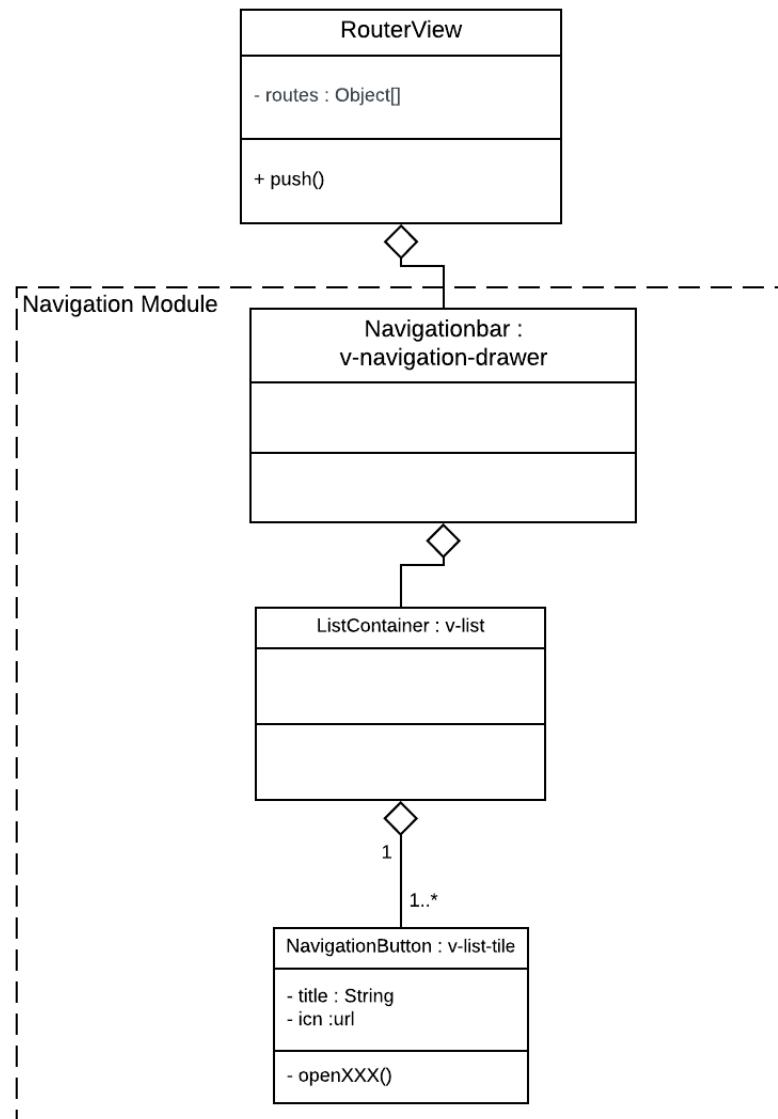


Abbildung 24: Mit der Navigation können andere Module aufgerufen werden

4. Frontend

Aufgabe Das Navigations Modul ist zuständig, die Oberfläche für die Navigation durch die Application zu bieten. Zu welchem Modulen der Benutzer navigieren kann ist in Abbildung 17 dargestellt.

View-Komponenten Die NavigationView ist nur eine Seitenleiste. Sie umfasst eine Liste von Links zu anderen Modulen. Siehe Abbildung 25 für eine bildliche Darstellung.

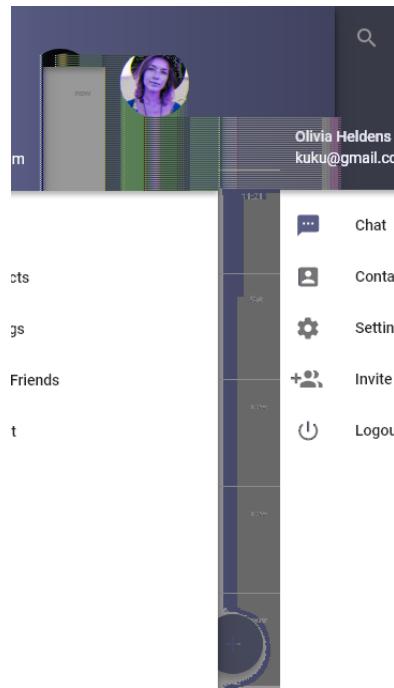


Abbildung 25: Navigation Seitenleiste

UI-Logic

- Damit die Navigation weiß zu welchem Modul sie springen muss behilft sie sich der URLs des Vue-Routers. Siehe Abbildung 42
- **title:** Jeder Verweis hat eine textliche Beschreibung
- **icn:** Jeder Verweis hat ein Bild zur Beschreibung
- **openXXX():** Jeder Verweis implementiert das Routing zu XXX.

4. Frontend

4.3.4. Profile

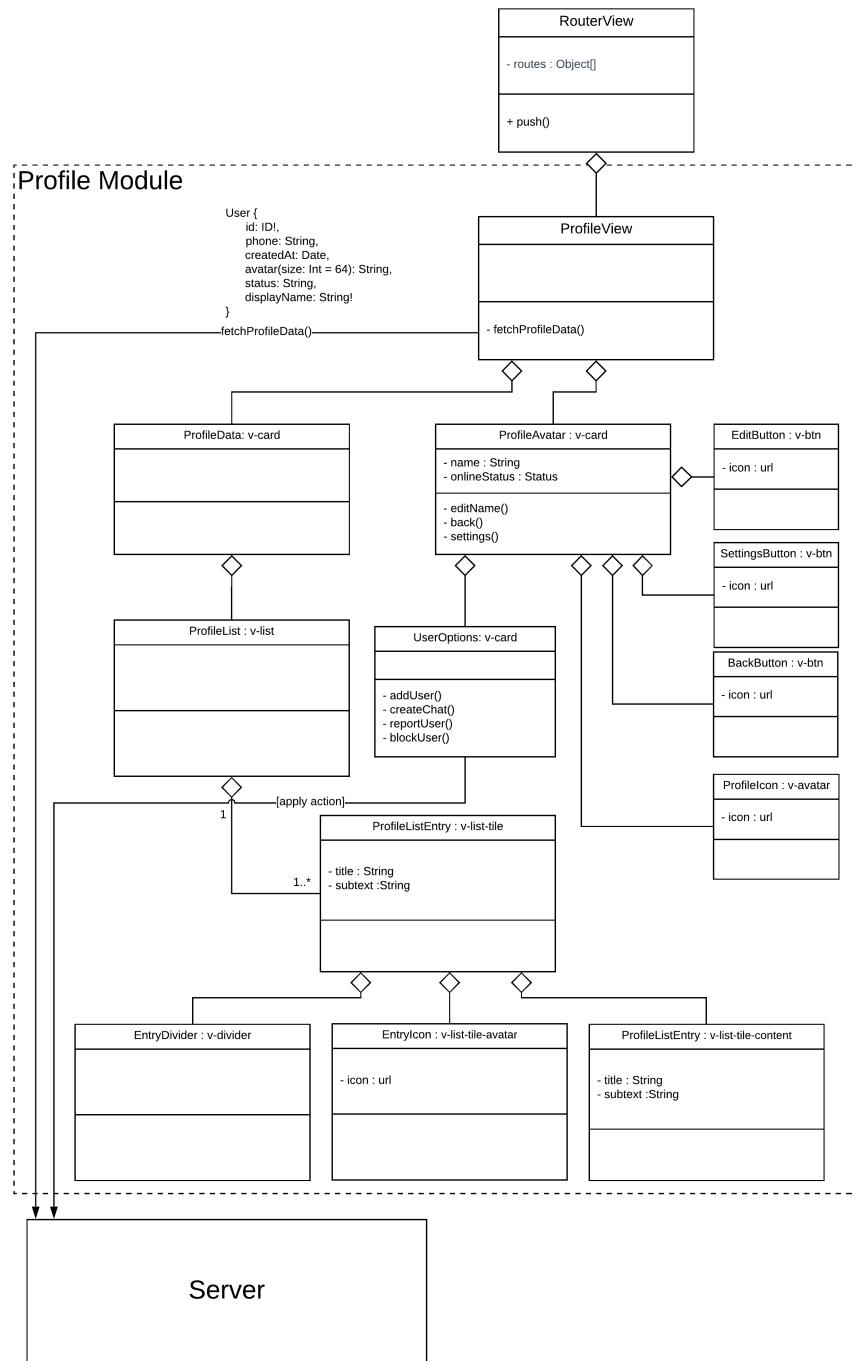


Abbildung 26: Das Profil Modul, um fremde Profile anzuzeigen zu können

4. Frontend

Aufgabe Das Modul gibt eine Übersicht über die Profile anderer Benutzer. Von hier aus soll der Benutzer zu weiteren Funktionen navigieren können, wie z.B. "Kontakt hinzufügen" oder "Kontakt blockieren" realisiert um neue Kontakte hinzuzufügen zu löschen oder zu blockieren.

View-Komponenten Die ProfileView besteht aus zwei Komponenten: ProfileAvatar, ProfileData. ProfileData ist eine Liste von Icon-Text-Einträgen. ProfileAvatar gibt Name, Status und Avatar an.

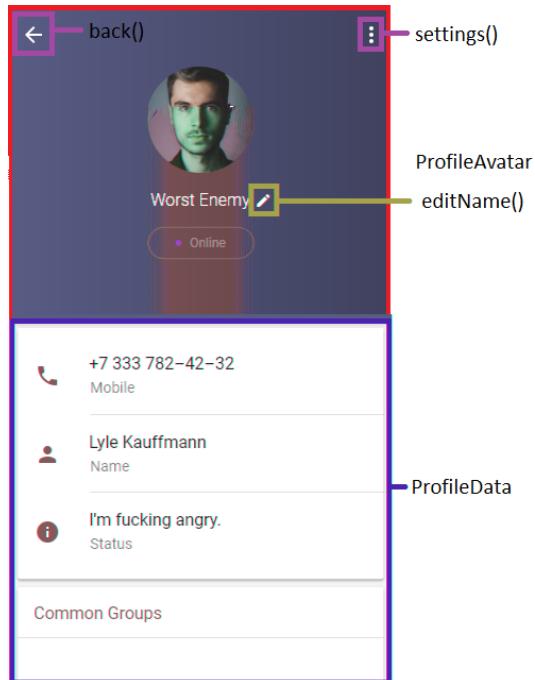


Abbildung 27: ProfileView Komponente

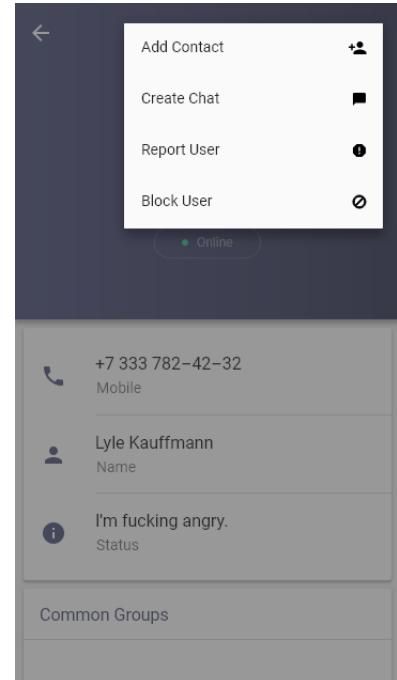


Abbildung 28: Benutzer Optionen

4. Frontend

UI-Logic

- Das Datenmodell eines Profils. Beinhaltet folgende Daten die angezeigt werden:
 - `icon`: Das Bild zum Profil.
 - `name`: Spitzname des angezeigten Benutzers/ sonst voller Name.
 - `onlineStatus`: Online-Status. Kann die Werte `ONLINE`, `ABWESEND` oder `OFFLINE` annehmen.
- Zu jeder Information gibt es eine eigene `ProfileListEntry` die im `title` die Information trägt und im `subtext` den Bezeichner
 - * Telefonnummer
 - * Vor- und Nachname
 - * Statusnachricht
- `back()`: Navigiert zurück zur Contactlist View.
- `settings()`: Optionen sind hier die Möglichkeiten mit welchen man mit Benutzer interagieren kann (Abbildung 28). Mögliche Optionen sind:
 - Als Kontakt hinzufügen (`addUser()`)
 - Chat erstellen (`createChat()`)
 - Benutzer melden (`reportUser()`)
 - Benutzer blockieren (`blockUser()`)
- `editName()`: Ändert den Spitznamen falls der Benutzer als Kontakt hinzugefügt wurde.

4. Frontend

Actions

```
input CreateContactInput {  
  id: ID!  
  contactId: ID!  
}
```

```
input CreatePrivateChatInput {  
  id: ID!  
  participantIds: [ID!]!  
}
```

```
input ReportUserInput {  
  id: ID!  
  reportID: ID!  
  reason: String!  
}
```

```
input BlockUserInput {  
  id: ID!  
  blockID: ID!  
}
```

4. Frontend

4.3.5. Settings

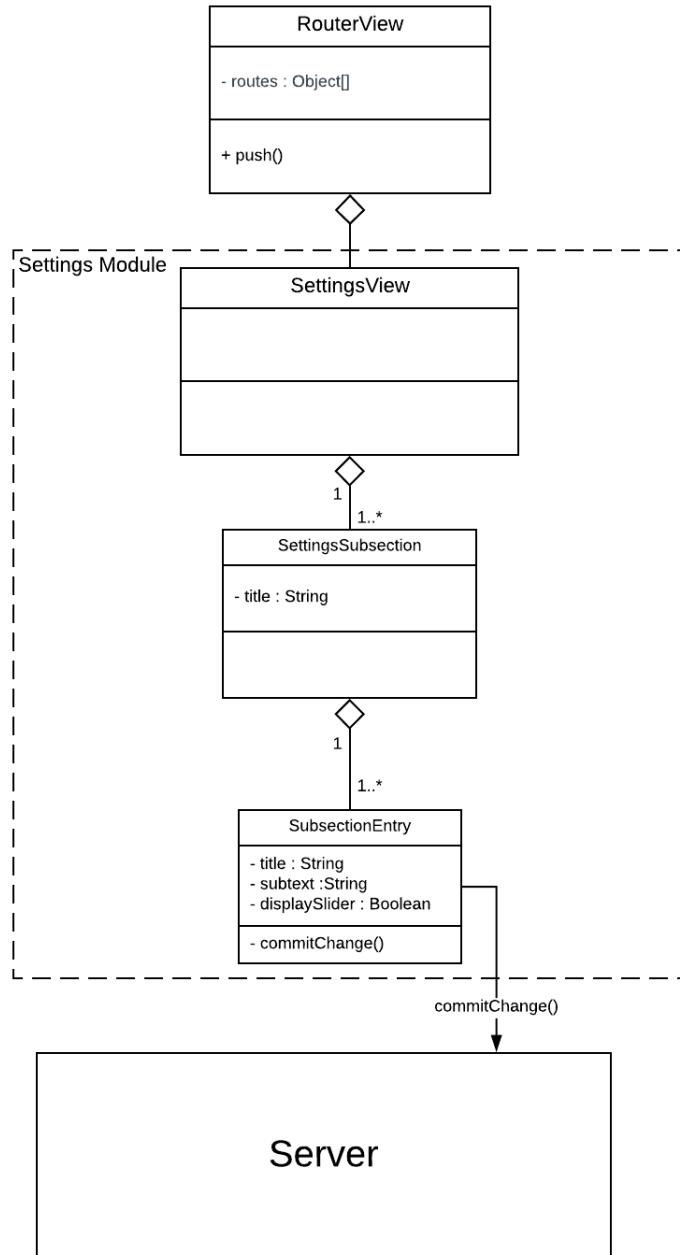


Abbildung 29: Das Modul für Einstellungen, um die Anwendung anzupassen

4. Frontend

Aufgabe Das Modul ermöglicht es dem Nutzer verschiedene Einstellungen anzupassen:

- Blockierte Kontakte verwalten
- Privatsphäreinstellungen verwalten
- Erwähnungsbenachrichtigungen ein-/auszuschalten
- Push-Benachrichtigungen ein-/auszuschalten

View-Komponenten

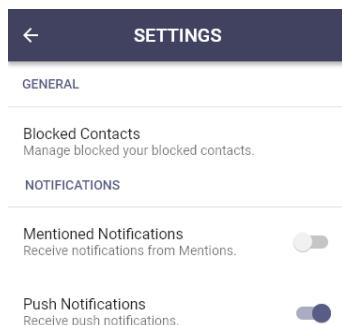


Abbildung 30: SettingsView

UI-Logic

- `commitChange()`: Jede SubsectionEntry steht sinngemäß für eine Einstellung die der Benutzer anpassen kann. Änderungen vom Benutzer werden an den Server weitergeleitet.
- `commitChange()` bei "Blocked Contacts": Das Contactlist Modul wird angezeigt mit einer Liste aller blockierten Profile. Keine Änderungen werden an den Server geschickt. Der Benutzer kann Änderungen in der Profil View (siehe Profil Modul) vornehmen.

4. Frontend

4.3.6. Chatlist

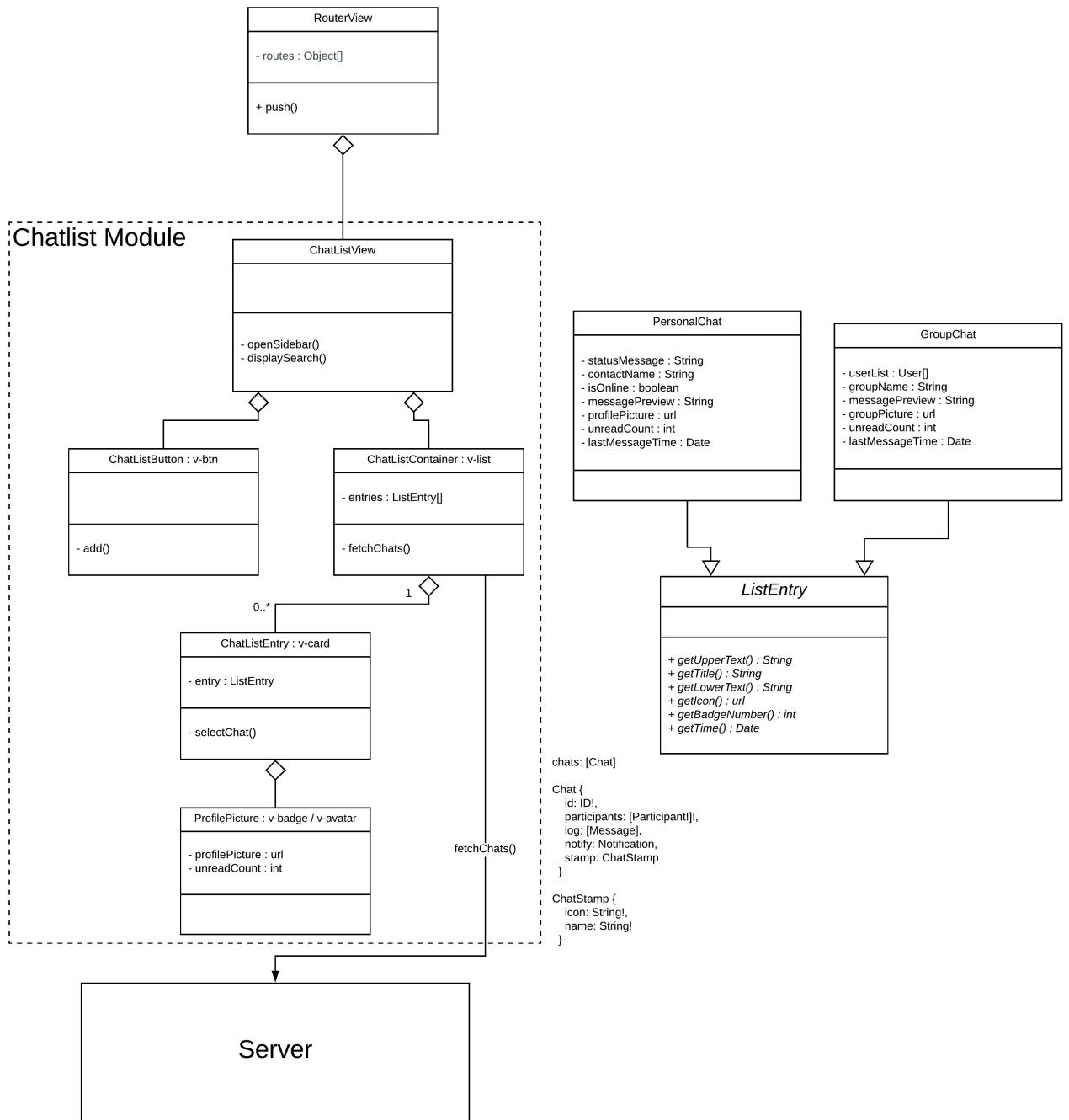


Abbildung 31: Das Chatlist Modul, welches alle Chats, an denen man teilnimmt, auflistet

4. Frontend

Aufgabe Die ChatList View-Komponente ist für die Darstellung der Chatliste verantwortlich. Über diese können Chats hinzugefügt werden oder zu Chats navigiert werden. Über die Chatliste kann die Suche oder Navigation geöffnet werden.

View-Komponenten Zum Modul der Chatlist gehört die ChatListView. Diese besteht aus einem ChatListContainer die einzelne ChatListEntries enthält. Jede ChatListEntry umfasst ein Profilbild und zusätzlichen Informationen. Die ChatListView hat zusätzlich einen ChatListButton zum Hinzufügen neuer Chats.

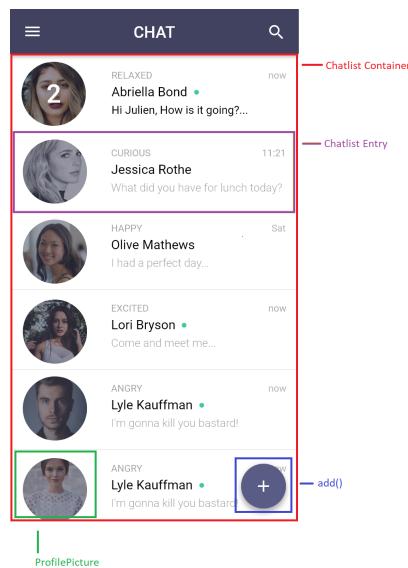


Abbildung 32: Beispielhafte Liste von Chats

UI-Logic

- `openSidebar()`: Öffnet die Navigation.
- `displaySearch()`: Öffnet die Suche.
- `ListEntry[]`: Der Benutzer hat kann an mehreren Chats gleichzeitig teilnehmen.
In jedem ListEntry Element werden Daten zum Chat angezeigt die abhängig vom Chatverlauf und vom Typ des Chats (`PersonalChat` oder `GroupChat`)
- `add()`: Öffnet die Contactlist um mit einem Kontakt einen neuen Chat anfangen zu können.

4. Frontend

4.3.7. Chats

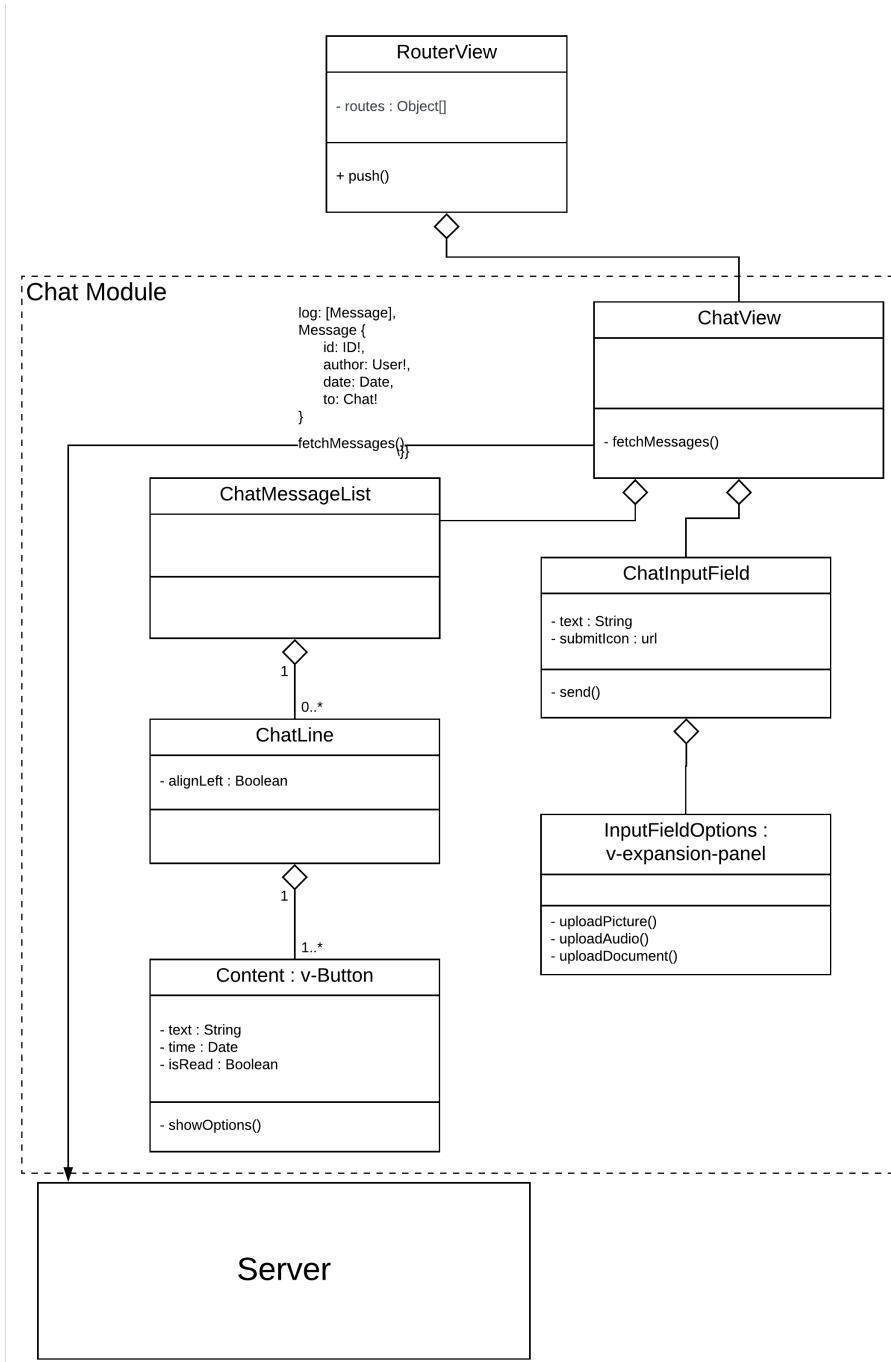


Abbildung 33: Das Chat Modul zur Kommunikation mit Kontakten

Aufgabe Im Chat findet die Kommunikation mit Kontakten statt. Es werden alle zuletzt gesendeten und empfangen Nachrichten dargestellt.

4. Frontend

View-Komponenten Die hauptsächliche Komponente ist die ChatView. Sie umfasst den MessageContainer welcher die letzten Nachrichten auflistet und die MessageBox um Nachrichten zu senden. Innerhalb der MessageBox sind weitere Optionen gegeben zum Versenden multimedialer Nachrichten oder zur Bestätigung der Nachricht.

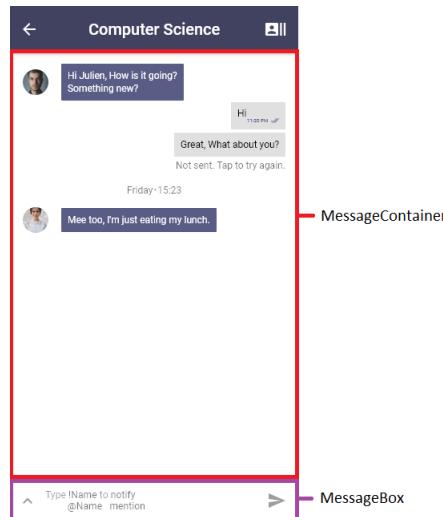


Abbildung 34: ChatView Komponente

UI-Logic

- **alignLeft:** Eigene Nachrichten werden anders dargestellt als Nachrichten anderer Benutzer: Eigene Nachrichten werden rechts ausgerichtet und Nachrichten anderer Benutzer links. Durch dieses Attribut kann man eigene Nachrichten von fremden unterscheiden.
- **text:** In der MessageBox eingetippte Texte werden in dieser Variable festgehalten.
- **send():** Die in `message` gespeicherte Inhalte werden an den Server übermittelt, die MessageBox wird leergeräumt und die Nachricht wird an den Teilnehmer des Chats weitergeleitet.
- **showOptions():** Macht die Oberfläche sichtbar um Änderungen an der Nachricht vorzunehmen: Sie gibt dem Benutzer die Möglichkeit Nachrichten zu bearbeiten oder zu löschen.
- Zu den weiteren Optionen um multimediale Nachrichten zu versenden gehören `uploadPicture()`, `uploadAudio()` und `uploadDocument()`. Es erscheint jeweils ein Dialog der das Hochladen einer Datei ermöglicht.

4. Frontend

4.3.8. Groupchats

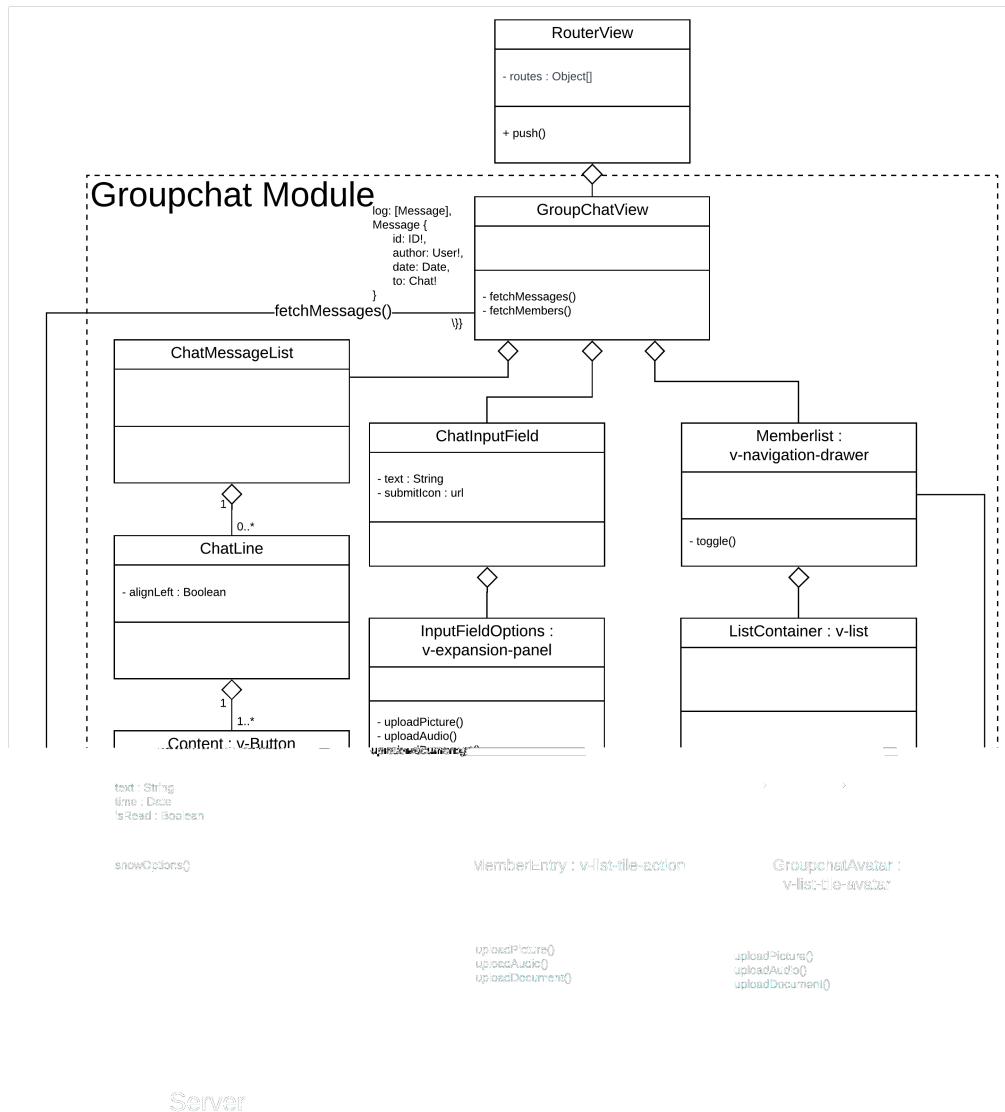


Abbildung 35: Das Modul zu Groupchats. Erweitert um die Teilnehmerliste

4. Frontend

Aufgabe Im Gruppenchat findet die Kommunikation mit mehreren Teilnehmern statt.

Das Groupchat Modul unterscheidet sich vom Chat Modul dadurch, dass zusätzlich eine Teilnehmerliste existiert.

View-Komponenten Die Komponenten aus dem Chat Modul (4.3.7) werden übernommen. Hinzukommt die Seitenleiste als neue Komponente zur Übersicht aller Teilnehmer des Chats.

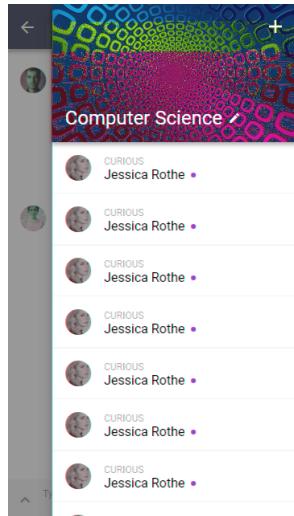


Abbildung 36: Seitenleiste

UI-Logic

- `toggle()`: Die Seitenleiste wird angezeigt/versteckt.

4. Frontend

4.3.9. Contactlist

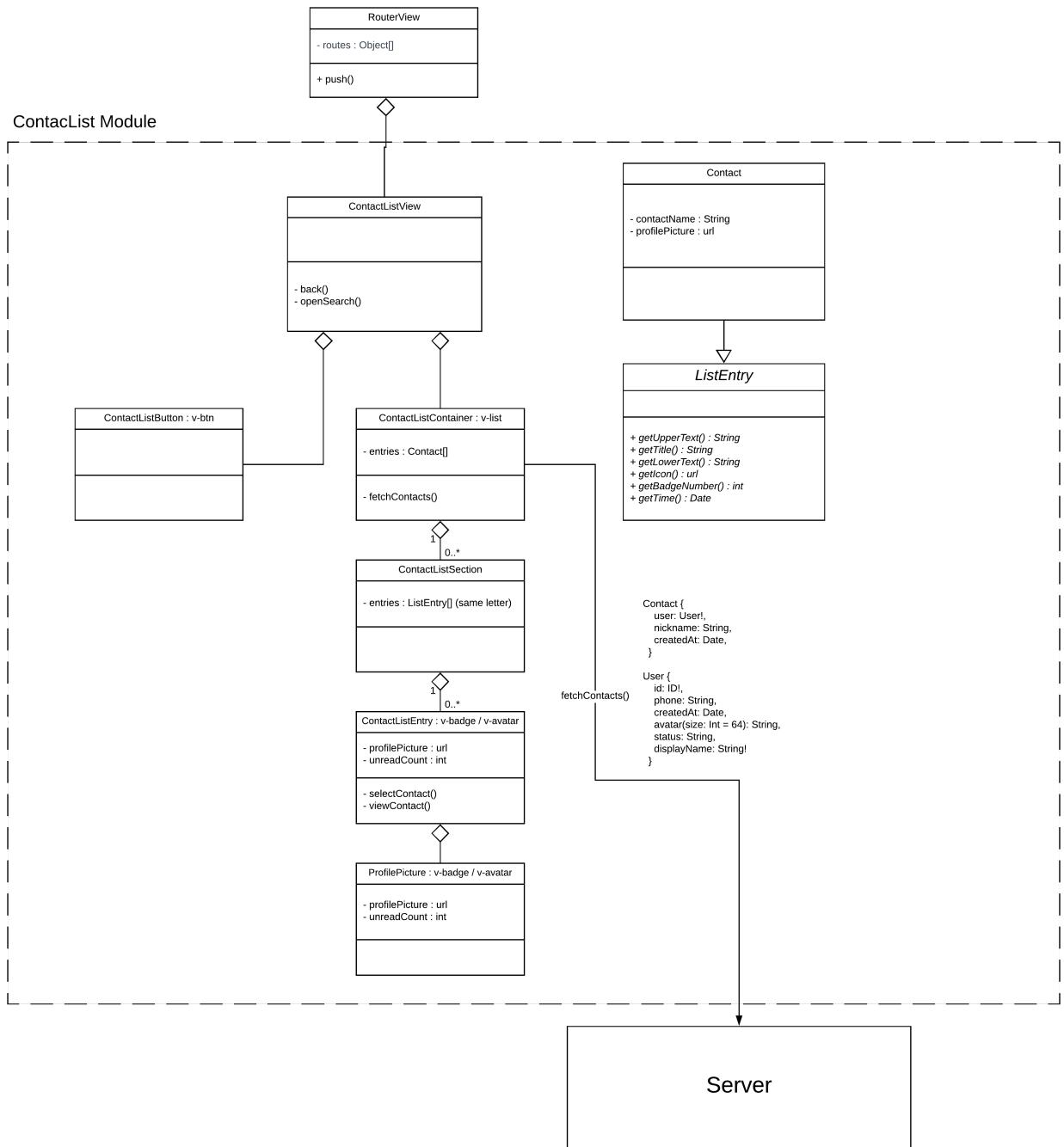


Abbildung 37: Das Modul Contactlist ist zur Auflistung aller Kontakte verantwortlich

4. Frontend

Aufgabe Die Contactlist gibt eine Oberfläche um derzeitig hinzugefügte Kontakte des Benutzers alphabetisch anzuzeigen. Sie erfüllt in verschiedenen Situationen den gleichen Zweck: Die Auswahl der Kontakte die man zum Gruppenchat hinzufügen will, die Auswahl mit welchem Kontakt man einen Chat startet oder die Suche nach Profilen eines Kontakts.

View-Komponenten Jeder Eintrag entspricht einem ContactListEntry. ContactListEntries werden in ContactListSections gruppiert. Jede ContactListSection soll einem Buchstaben entsprechen worin die Kontakte mit dem entsprechenden Spitznamen liegen. Die ContactListSections werden dann alphabetisch im ContactList-Container sortiert.

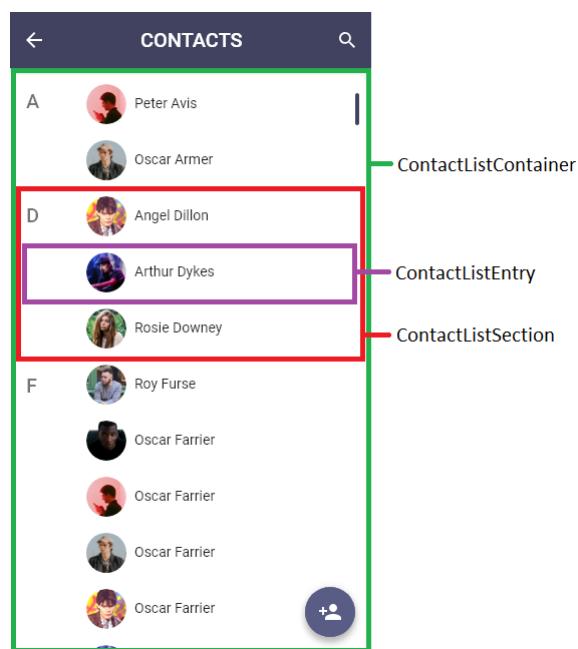


Abbildung 38: ContactlistView

4. Frontend

UI-Logic

- `entries`: ListEntries werden aufgelistet, wobei im Gegensatz zu PersonalChat und GroupChat (siehe 4.3.6) weniger Informationen preisgegeben werden. Nur `title` und `Icon` sind implementiert.
- `back()`: Navigiert zum vorherigen View.
- `openSearch()`: Öffnet die Suchleiste.
- `selectContact()`: Wählt einen Kontakt aus, um ihn später zu einem Chat hinzufügen zu können.
- `viewContact()`: Öffnet das Profil des Kontakts.

4. Frontend

4.3.10. Search

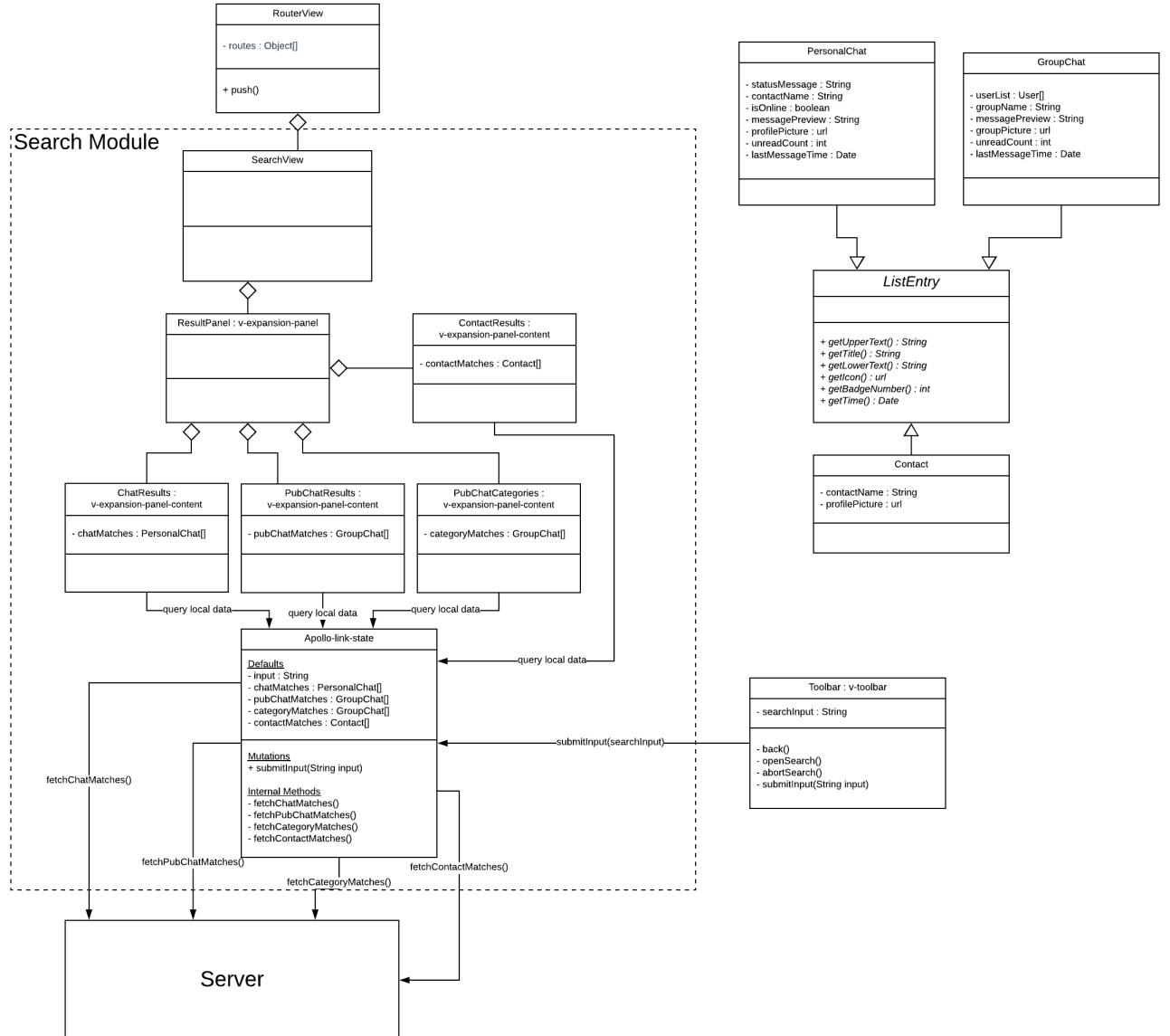


Abbildung 39: Das Modul für die Suche

Aufgabe Das Modul zur Suche soll dem Benutzer eine Eingabefläche anbieten und dem Benutzer Vorschläge präsentieren. Die Suche unterscheidet private und öffentliche Chats oder Kontakte. Sie soll es dem Benutzer ermöglichen nach öffentlichen Chats zu suchen oder nach einem bestimmten Kontakt in seiner Kontaktliste zu suchen.

4. Frontend

View-Komponenten Die SearchView besteht aus vier aufklappbaren → Reitern : ChatResults, PubChatResults, ContactResults und PubChatCategories. Jeder davon ist ein Container für ListEntries.

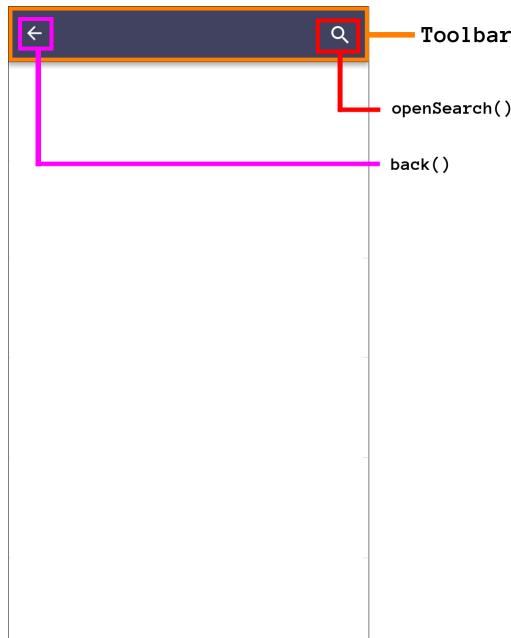


Abbildung 40: Toolbar

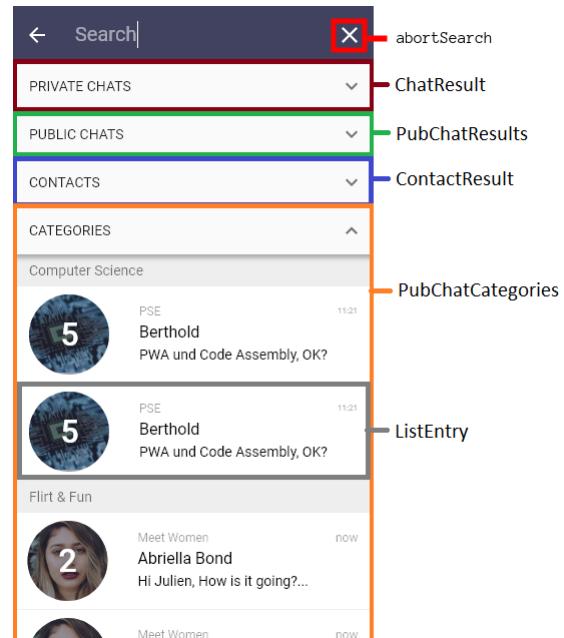


Abbildung 41: SearchView

UI-Logic

- **input:** Die Eingabe des Benutzers welche direkt an den Server für Suchanfragen übermittelt werden.
- **chatMatches:** Resultate für eigene private Chats die vom Server empfangen wurde.
- **pubChatMatches:** Resultate für öffentliche Chats.
- **contactMatches:** Resultate für Kontakte.
- **categoryMatches:** Resultate für öffentliche Chat-Kategorien.

4. Frontend

Zur Toolbar:

- `openSearch()`: Öffnet die Suche.
- `back()`: Schließt die Suche und navigiert zur vorherigen View.
- `abortSearch()`: Löscht die derzeitige Eingabe.

4.4. Router-Links

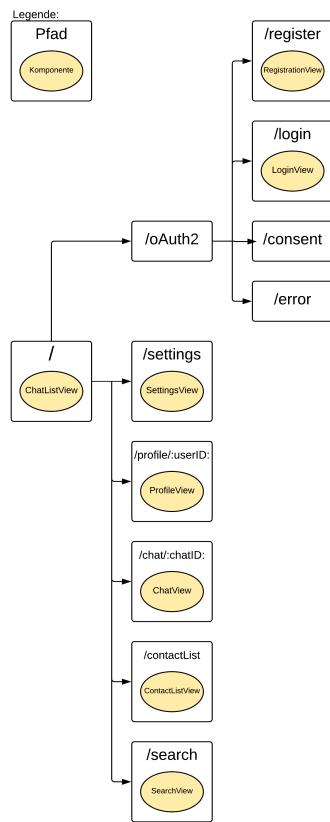


Abbildung 42: Baumdiagramm Router-Links

In Abbildung 42 sind die Pfade und die dazugehörigen View Komponenten als Baumdiagramm dargestellt. Dabei wird im Root-Verzeichnis die ChatListView Komponente angezeigt. Des Weiteren wird unter /oAuth2 der Registrierungs und Login Prozess bereitgestellt.

Ebenso:

- **/settings:** Hier wird die Einstellungs-Komponente dargestellt
- **/profile/:userID:** : Hier wird das Profil des Users mit der übergebenen UserID dargestellt
- **/chat/:chatID:** : Hier wird der Chat mit der übergebenen ChatID dargestellt.
- **/contactList:** Hier wird die Kontaktliste dargestellt.
- **/search:** Hier wird die Suche dargestellt.

4.5. Service Worker

Der Service Worker ist eine einzelne JavaScript-Datei und erlaubt es JavaScript Code getrennt vom Hauptthread im Hintergrund auszuführen. Wie in Kapitel 1.3.5 schon erwähnt ist ein Service Worker einer der wesentlichen Bestandteile einer PWA.

Der Service Worker hat für ChatSphere folgende Aufgaben zu erfüllen:

Offline-Nutzung Illustriert an Abbildung 43: Damit Benutzer Nachrichten Offline einsehen wird beim ersten Start der Anwendung der Service Worker installiert (dem einzigen Zeitpunkt, zu dem eine Internetverbindung notwendig ist). Ein Cache wird erstellt und alle benötigten Ressourcen heruntergeladen und gespeichert. Sofern keine Konflikte mit anderen bereits aktiven Service Worker besteht wurde der Service Worker erfolgreich aktiviert. Ohne eine vorhandene Internetverbindung werden nun Anfragen des Benutzers aus dem Cache geladen. Sofern Anfragen stattfinden zu denen nichts im Cache zu finden ist, wird wieder die Internetverbindung erforderlich sein oder ein Fehler ausgegeben.

Push-Notification Illustriert an Abbildung 44: Damit Benutzer neue Nachrichten mitgeteilt kriegen ohne die Anwendung im Vordergrund zu haben braucht es den Service-Worker. Der Service-Worker läuft bei geschlossener Anwendung weiterhin aktiv im Hintergrund. Dazu muss er beim Aufrufen der web page (wie oben schonmal erwähnt) im Browser (unser user agent in der Abbildung) registriert werden. Der Service Worker abonniert die Push Komponente des Servers und hat mittels der PushSubscription die Möglichkeit die Subscription zu kündigen. Mit der bestehenden Verbindung kann der Server bei gewissen Ereignissen nun Notifications senden. Falls der Benutzer seine Einstellungen ändert kündigt er die Subscription.

5. Datenmodelle

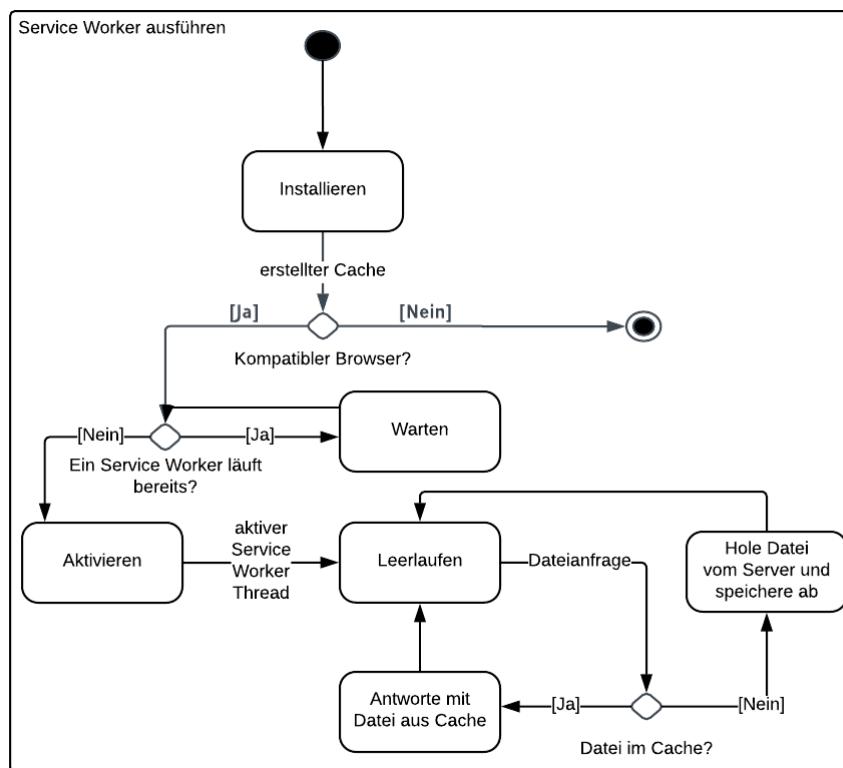


Abbildung 43: Aktivitätsdiagramm zur Offline-Funktion eines Service Workers

5. Datenmodelle

Zur persistenten Datenspeicherung wird das SQL-Datenbankmanagementsystem MariaDB eingesetzt.

- Das Datenbankdesign sieht zurzeit nicht vor, dass Benutzer Chatprotokolle lesen können (oder die Chats noch sehen), wenn sie nicht mehr Teil einer Konversation sind.
- Erwähnungen von Benutzern haben derzeit keinen Datenbanksupport
- Benachrichtigungen werden implizit - und nicht explizit - verwaltet, durch einen Zeitstempel der angibt, bis zu welchem Zeitpunkt im Log Einträge gelesen wurden.

5. Datenmodelle

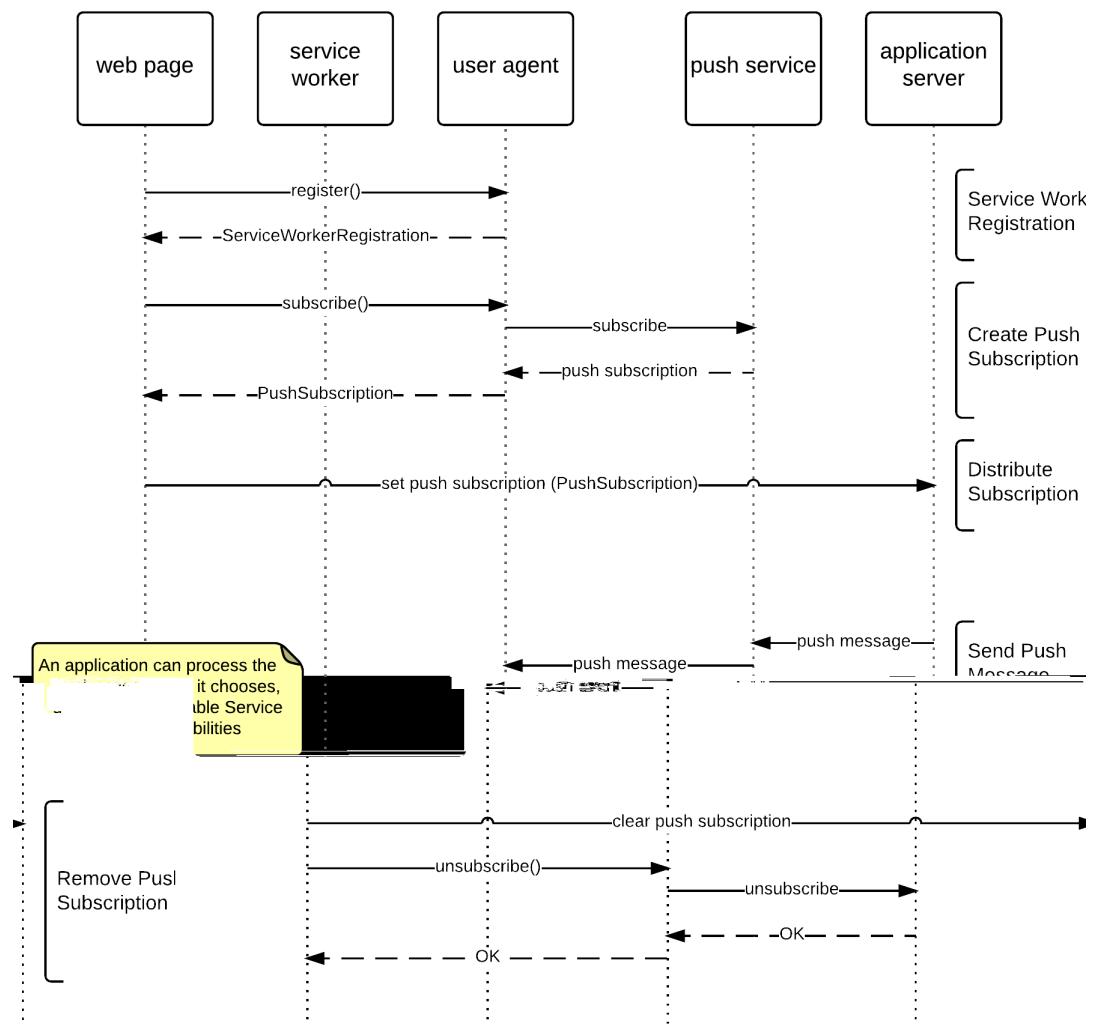


Abbildung 44: Sequenzdiagramm für beispielhafte Push-Notification

den. Im Gegensatz zu Speicherung der inkrementellen Nachrichten-ID, erfährt der Benutzer so auch Überarbeitungen.

- Bearbeitete Nachrichten können aufgrund verschiedener Erstell- und Änderungszeitpunkte erkannt werden. Eine explizite Flag wird nicht gesetzt. Eine Versionierung mit Zugang zu vorherigen Bearbeitungen findet nicht statt.
- Während der Änderung der E-Mail-Adresse wird die alte bereits verifizierte Adresse überschrieben.

5. Datenmodelle

- WK440 Sticer haben derzeit keinen Datenbanksupport. Es ist derzeit vorgesehen, dass diese vom Java-Server-System vorgegeben werden und Benutzer keine eigenen Pakete erstellen können. Es ließen sich aber zusätzliche Tabellen für den Markup-Syntax-Parser erweitern.
- W6XX beinhaltet den Globalen Chat, welcher nur in-Memory des JavaServers existiert, da dieser nicht persistent ist. Erforderliche Atribute wie Sprache und Country/Provinzcode werden in den Benutzereigenschaften gespeichert.

Ebenso ist zu beachten, dass das Sitzungsmanagement vom Authorizationserver übernommen wird und somit nicht Gegenstand der Datenbankmodellierung ist.

5. Datenmodelle

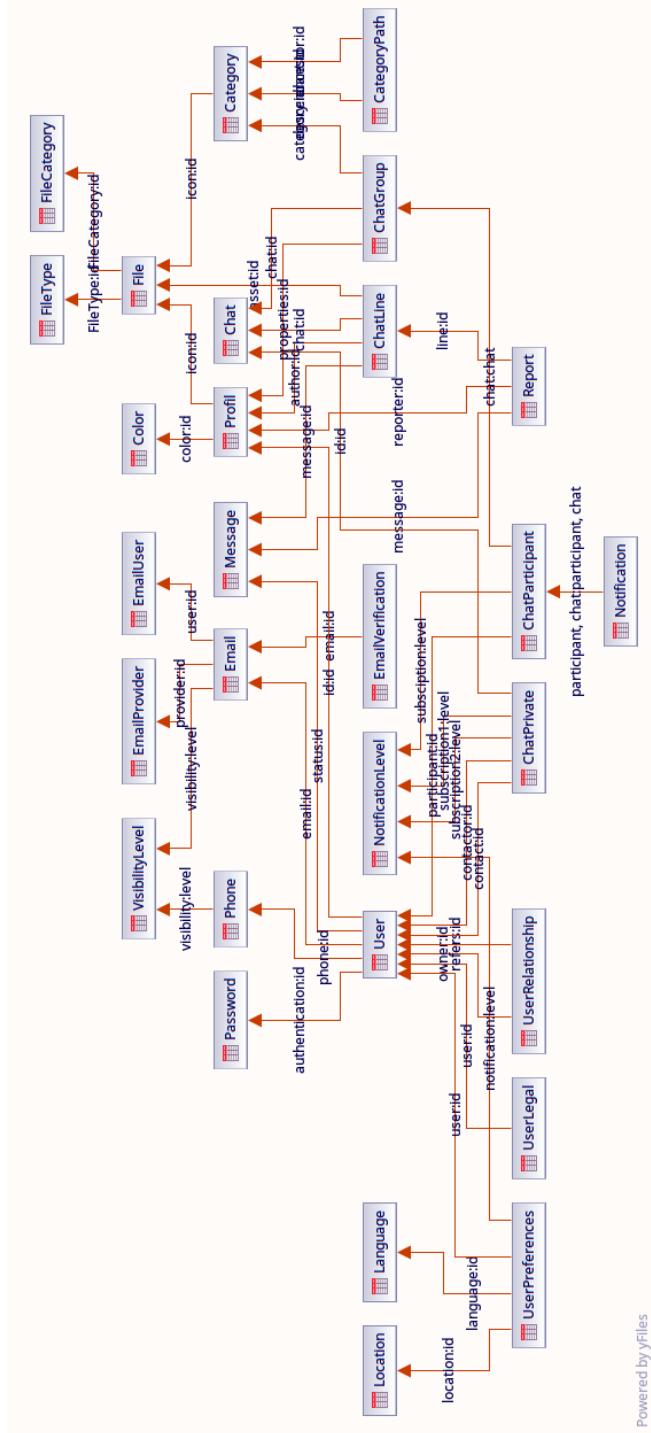


Abbildung 45: Gesamtübersicht der Datenbank

Grundlegende ist die Datenbank unterteilt in Lookup-Tabellen wie die NotificationLevel-

5. Datenmodelle

Tabelle, die Benutzertabelle, welcher in mehrere Tabellen vertikal aufgeteilt wurde sowie Referenzen auf Eigenschaften-Attribute, die in einzelnen Tabellen verwaltet werden, sowie die Relationen zwischen Benutzern und Chats.

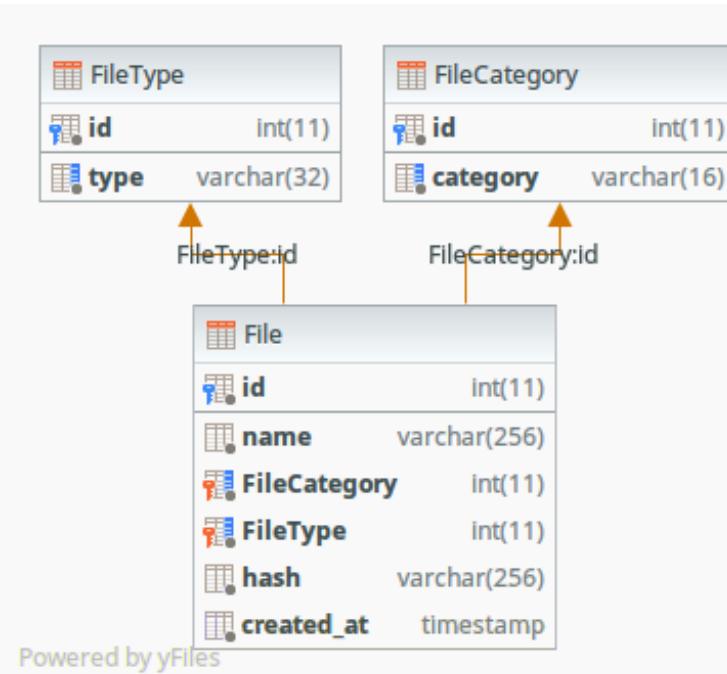


Abbildung 46: Datenspeicherung von Dateien verschiedener Dateitypen.

In der Datenbank wird der MIME-Type der Datei abgespeichert, sowie ein Hash-Wert der Datei. Die Binärdaten, also die Dateien selbst, werden außerhalb der Datenbank vom API-Server verwaltet und anhand des Hash-Wertes referenziert.

Die in der ChatListe angezeigten Eigenschaften wie der Benutzername bzw. der formatierte Anzeigename, die Hintergrundfarbe als auch ein Icon, werden in dem 'Profil' Objekt gekapselt, wodurch diese im Falle von Gruppenchats eine einheitliche Struktur zur Speicherung bietet und Redundanz der Spalten verhindert.

Ebenso wird dadurch erreicht, dass sämtliche über die Suche auffindbaren Namen innerhalb einer einzelnen Tabelle organisiert werden.

Einige Benutzereigenschaften wurden durch vertikales Splitting der Tabelle in eigenen Tabellen abgegrenzt. Dies verkleinert die Haupttabelle und vereinfacht bei Attributen wie derjenigen der Tabelle UserLegal ein etwaiges Refactoring.

5. Datenmodelle

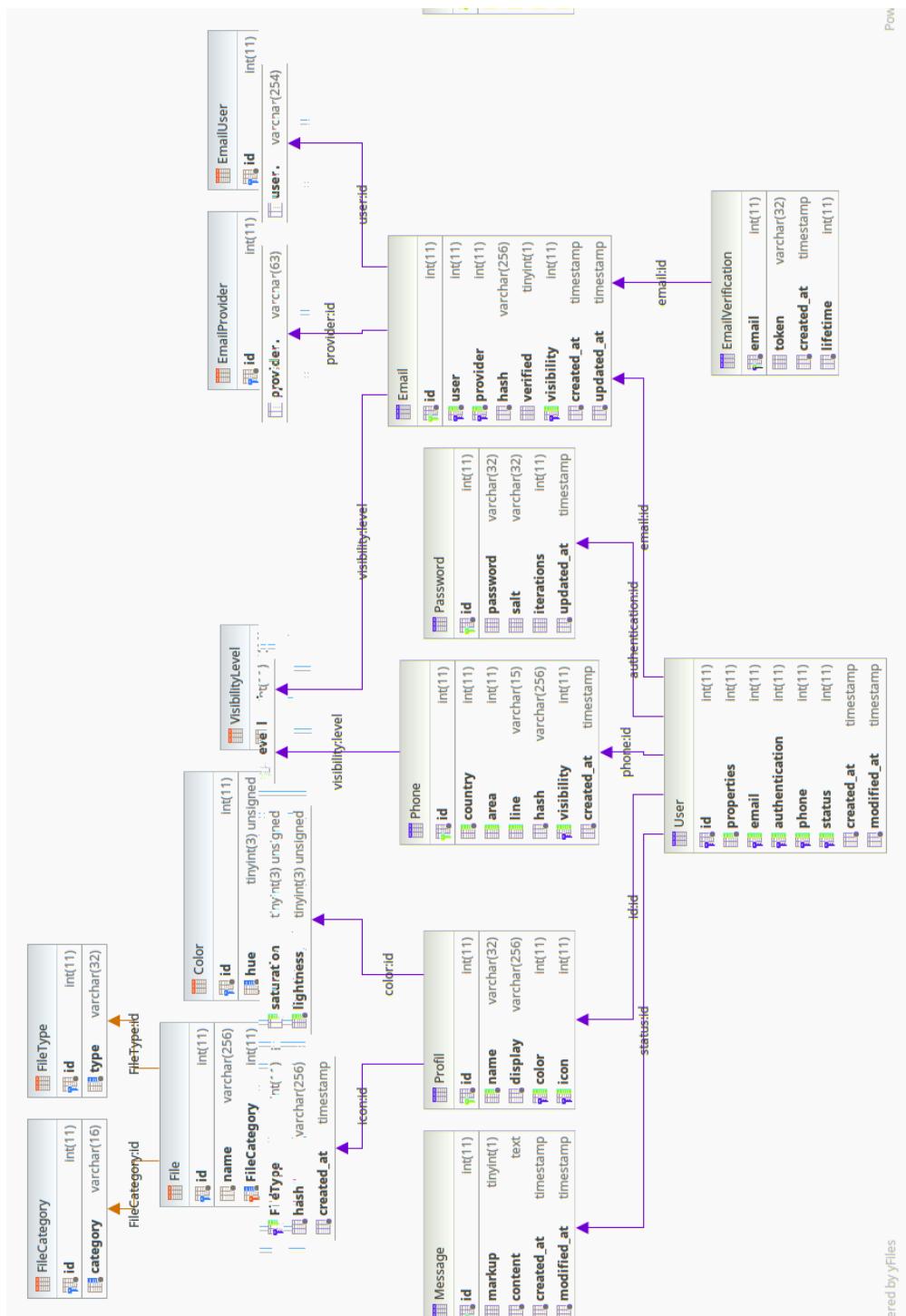


Abbildung 47: Verwaltung einzelner Benutzerprofileigenschaften

5. Datenmodelle

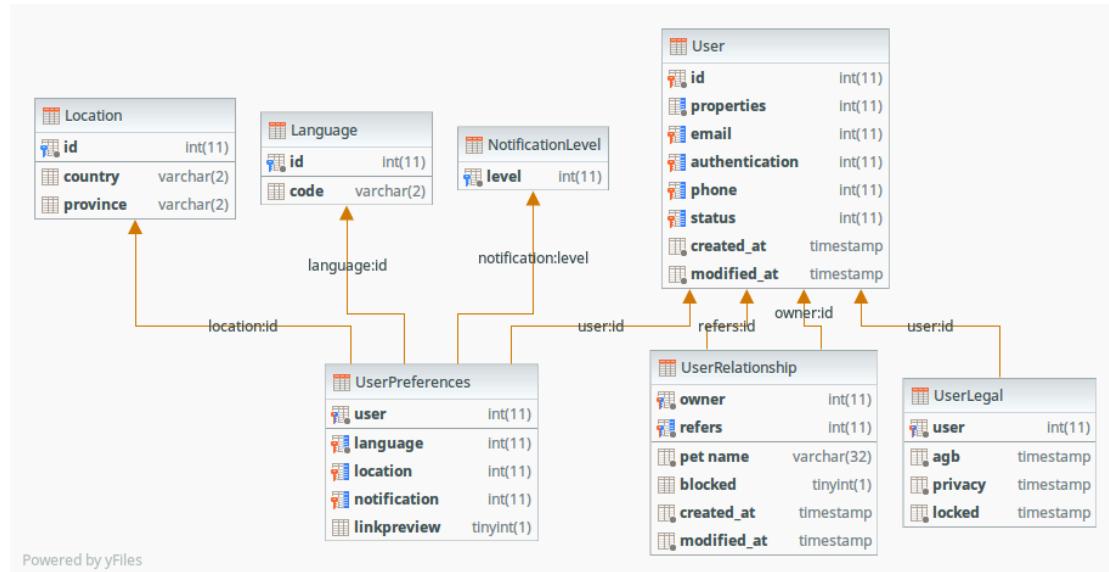


Abbildung 48: Datenhaltung der Benutzereinstellungen

Die Chattypen Private-Chats und Gruppenchats werden getrennt voneinander in eigenen Tabellen verwaltet. Die öffentlichen Gruppenchats werden durch das Vorhandensein eines gesetzten Kategorie-Attributes bestimmt. Die Baumhierarchie der Kategorien wurden als Closure-Tabelle organisiert. Dabei wird von einem Parent jedes Child einmal gespeichert, was zwar zu quadratischem Speicheraufwand im worst-case führt, dafür aber die Erstellung, Verwaltung, Aktualisierung und ebenso die Queries vereinfacht.

5. Datenmodelle

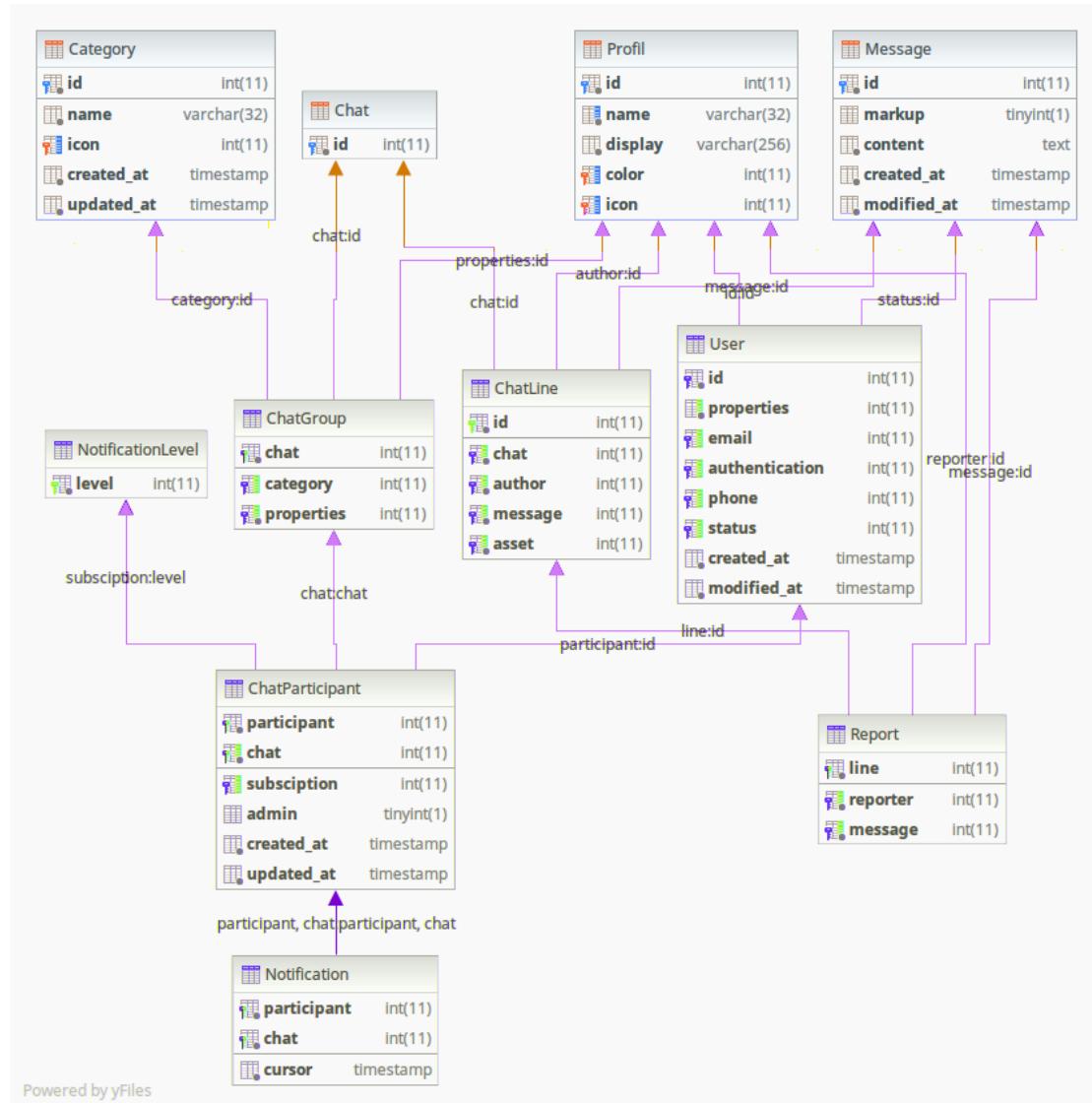


Abbildung 49: Organisation und Struktur der Chats

Eine vollständige Datenbankübersicht, geeignet zum Ausdrucken auf ggf. DIN A3 lässt sich in Discord finden. (Einfach mal nach »Datenbank« suchen)

6. Backend

- **Controller:** Das Backend implementiert den GraphQL-API-Server in Java. Jeder API-Methode besitzt einen eigenen Resolver über den diese aufgelöst wird. Ein Liste dieser findet sich im Kapitel API wieder.
- **Modell** Die GraphQL-API-Objekte stellen geradewegs die einzelnen vom Modell verwalteten Objekteinheiten dar.
- **Speicher** Das Modell wiederum ist abstrahiert von der darunterliegenden Speicherrepräsentation, zu der das ORM und der lokale Dateispeicher (FileRepository) gehören. Das ORM ergibt sich aus dem Datenbankmodellierung.
- **Schnittstellen** Parallel zum Speicher wird über Schnittstellen auf den Autorisation-Server oder Schnittstellen für Passwordhashing und Mailversand zugegriffen, um die erweiterten Anforderungen der Kürkriterien zu implementieren.

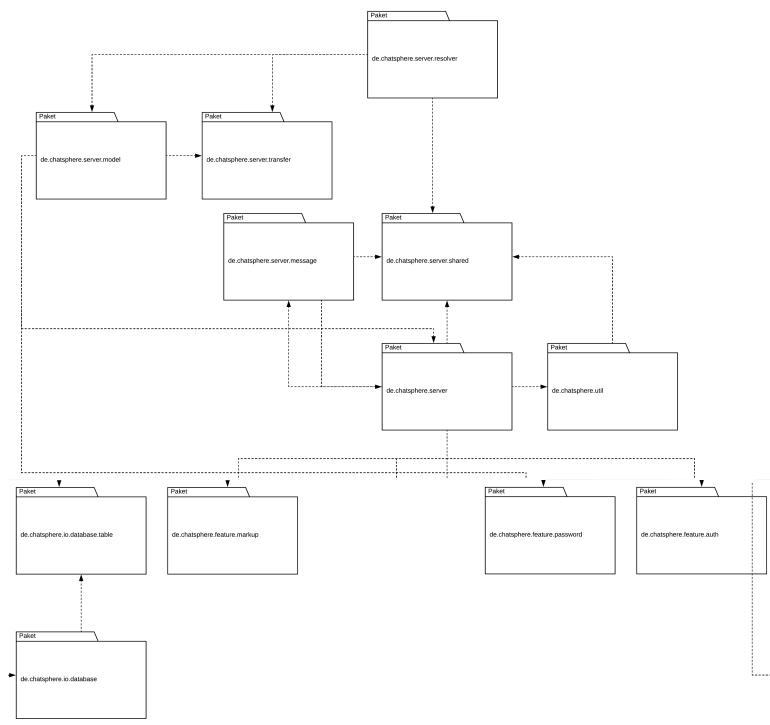


Abbildung 50: Backend Paketdiagramm

6. Backend

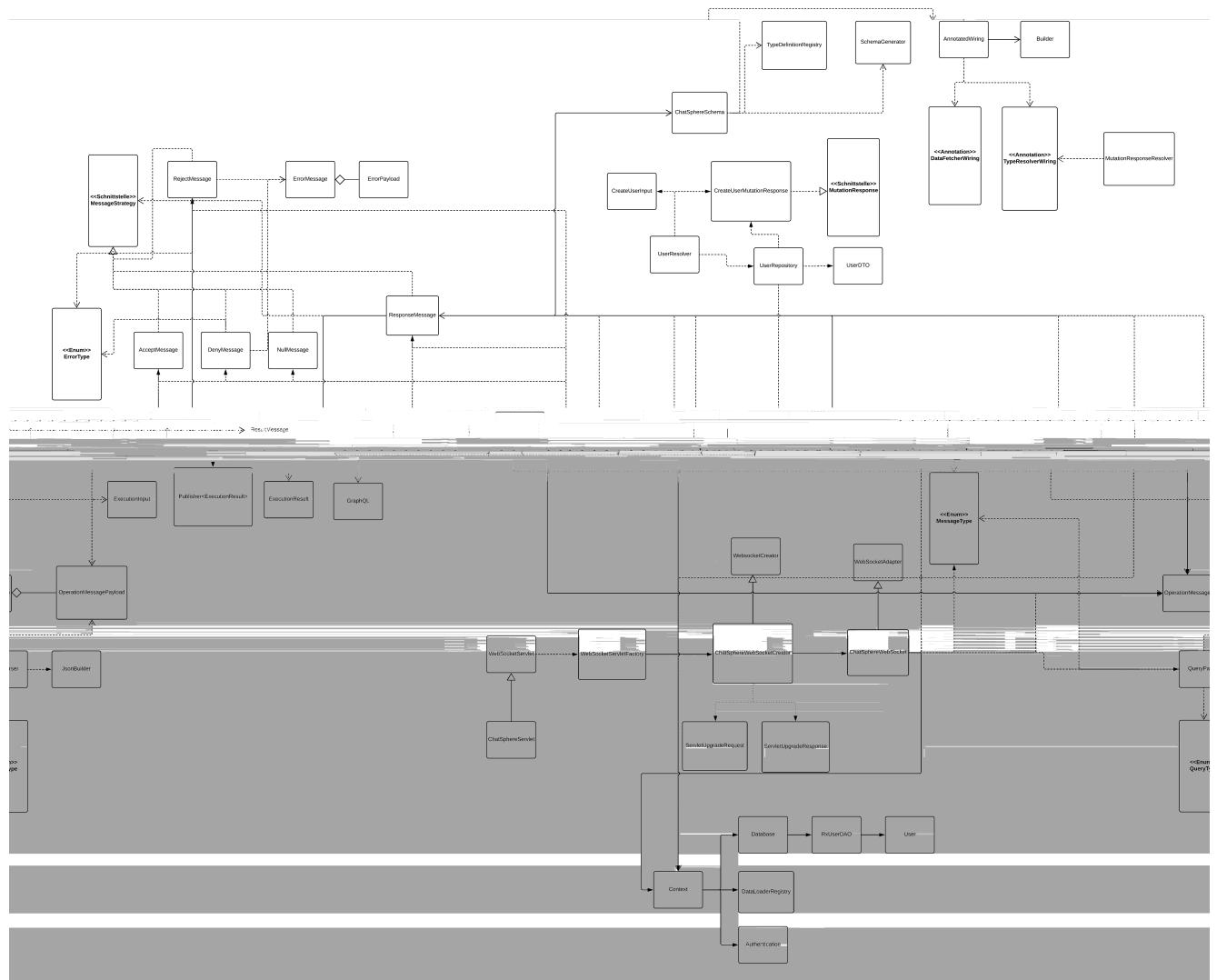


Abbildung 51: Backend UML

6. Backend

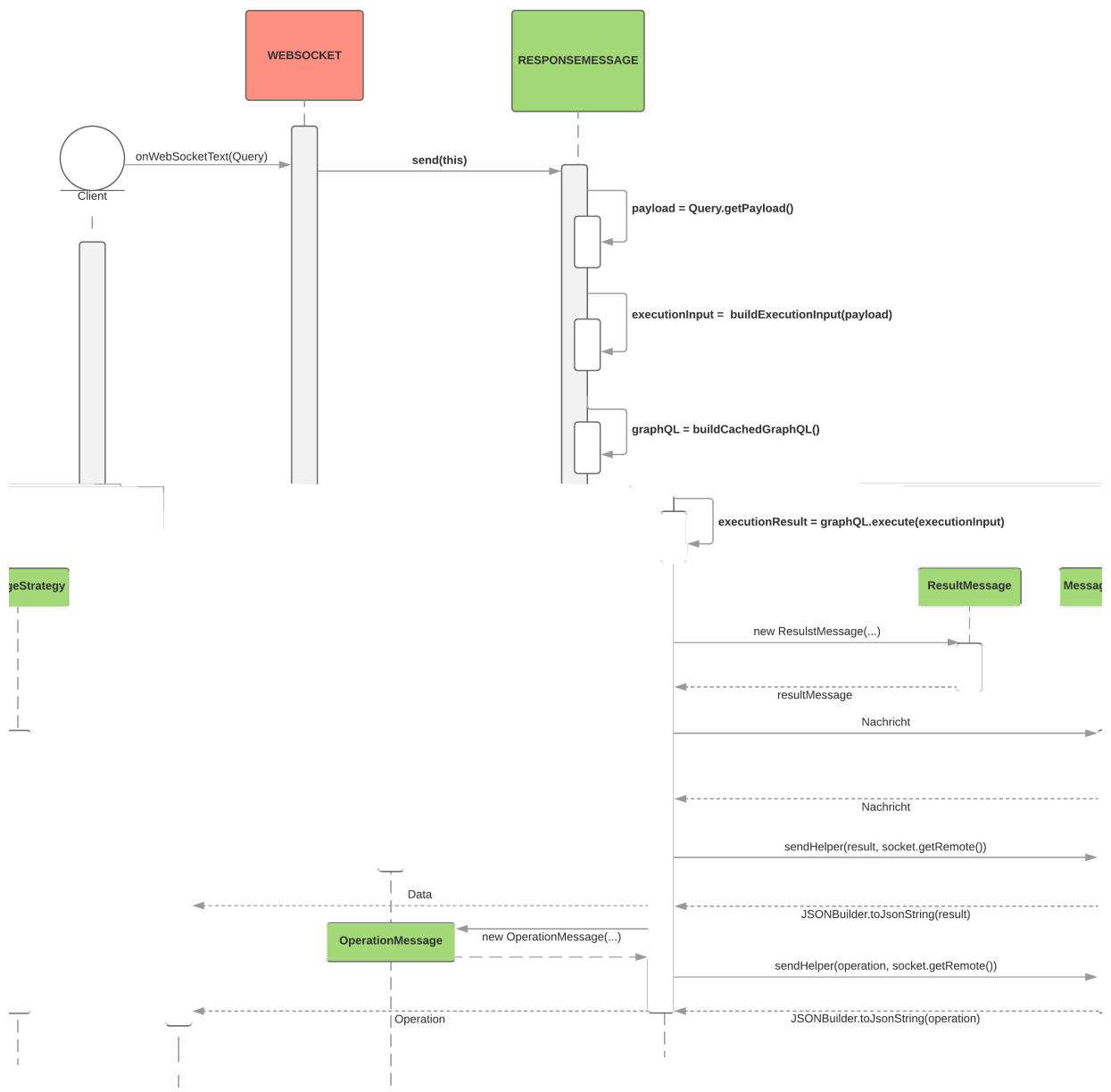


Abbildung 52: Client Server Kommunikation

6. Backend

6.1. Module

6.1.1. Server (de.chatsphere.server)

Im Server Paket liegt die implementierung des GraphQL Servers.

6.1.2. IO (de.chatsphere.io)

Das IO Paket abstrahiert konkrete Lese- und Schreibmechanismen, was es ermöglicht, ohne konkretes Wissen über das Ziel Daten zu speichern und lesen.

de.chatsphere.io.database

Dieses Paket ermöglicht Zugriff auf die Datenbank.

de.chatsphere.io.file

Dieses Paket erlaubt es, ins Dateisystem zu schreiben.

de.chatsphere.io.mail

Dieses Paket bietet eine Schnittstelle und mögliche Implementierungen eines email-Servers.

de.chatsphere.io.rest

Dieses Paket ermöglicht es auf REST-APIs zugreifen zu können.

6.1.3. Features (de.chatsphere.feature)

In den Feature-Unterpaketen werden die komplexeren Berechnungen ausgelagert, um Plug-In mäßig verwendet werden zu können.

de.chatsphere.feature.auth

Dieses Paket enthält Klassasen, mit denen über einen REST-Client die Validität eines Benutzers festgestellt werden kann.

de.chatsphere.feature.mail

Dieses Paket ermöglicht das Versenden von emails, z.B. zur Validierung der email-Adresse

6. Backend

eines Benutzers.

de.chatsphere.feature.markup

Dieses Paket ermöglicht das validieren von Markup-Syntax.

de.chatsphere.feature.password

Dieses Paket enthält Hashing und Salting Algorhitmen um Passwörter vor unerlaubtem Zugriff zu schützen.

7. Kommunikationsabläufe

7. Kommunikationsabläufe

Aktivitätsdiagramme

8. User-Stories

8.1. Login

1. Als Besitzer mehrerer Geräte möchte ich auf allen die Anwendung benutzen können, um nicht ständig das Gerät wechseln zu müssen.
2. Als Benutzer eines gemeinsamen Geräts, möchte ich dass andere nicht auf mein Account zugreifen können, damit andere nicht in meinem Namen Konversationen halten können.
3. Als Seitenbesucher, kann ich trotzdem auf mein Account zugreifen, falls ich mein Passwort vergessen habe.

8.2. Registrierung

4. Als Seitenbesucher, möchte ich anfangen die App benutzen zu können, um mit anderen in Kontakt zu treten.
5. Als Benutzer soll niemand anders Zugriff auf meine Chatverläufe kriegen, um meine Privatsphäre zu wahren.

8.3. Chatlist

6. Als Benutzer möchte ich die neusten Nachrichten angezeigt kriegen, um keine Nachricht aus Versehen zu übersehen.

8. User-Stories

7. Als Benutzer möchte ich die Anzahl von Nachrichten dargestellt kriegen, damit ich die Übersicht behalten kann wie viel ich verpasst habe.
8. Als Benutzer der App möchte ich mir vergangene Konversationen ansehen, um vergangene Informationen herzuholen und Erinnerungen zu wecken.
9. Als Benutzer möchte ich eine Liste von Chats kriegen, um eine Übersicht meiner Konversationen zu kriegen.
10. Als Benutzer will ich Chats löschen können, um meine Liste übersichtlich zu halten.

8.4. Chats

11. Als Freund möchte ich den anderen Freunden eine Nachricht schreiben, um mit ihnen in Kontakt bleiben zu können.
12. Als Benutzer möchte ich über neue Nachrichten benachrichtigt werden, um nicht selber darauf aufpassen zu müssen.
13. Als Benutzer möchte ich meine versendete Nachrichten ändern können, um Fehler die ich gemacht habe beheben zu können.
14. Als Benutzer möchte ich außer Texte auch Bilder und Videos senden, damit ich nicht erst die Medien woanders hochladen muss zum Teilen mit anderen.
15. Als Benutzer möchte ich aufgenommene Audio-Dateien versenden können, damit ich mir manchmal das Tippen einer Nachricht sparen kann.
16. Als Benutzer möchte ich auf Links anklicken können, um mir das kopieren und einfügen in einen Browser sparen zu können.
17. Als Benutzer möchte ich meine versendete Nachrichten löschen können, damit ich eine Nachricht neu verfassen kann.
18. Als Benutzer möchte ich Nachrichten formatieren können, um Übersichtlichkeit

8. User-Stories

reinzubringen.

19. Als Benutzer sollen vergangene Konversationen und Freunde gespeichert bleiben, damit ich sie wieder aufrufen kann.
20. Als Benutzer möchte ich Emojis versenden, um alle Gefühle ausdrücken zu können.
21. Als Benutzer hab ich eine Möglichkeit von böswillige Dingen zu berichten, um andere Benutzer davor vorzeitig schützen zu können.

8.5. Groupchats

22. Als Benutzer möchte ich andere erwähnen können, damit sie wissen können, dass ich sie direkt anspreche.
23. Als Benutzer will ich Chats an denen ich teilnehme stummschalten können, um nicht gestört zu werden.
24. Als Leiter einer Gruppe möchte ich einen gemeinsamen Chat mit meiner Gruppe haben, um nicht mit allen einzeln sprechen zu müssen.
25. Als Admin eines Gruppen-Chats möchte ich Leute einladen können, um gleichgesinnte auf meine Gruppe aufmerksam zu machen.
26. Als Admin eines Gruppen-Chats möchte ich weitere Admins ernennen, um die Verwaltung an andere delegieren zu können.
27. Als Admin eines Gruppen-Chats kann ich Benutzer entfernen, um meine Ansprüche an den Chat durchsetzen zu können.
28. Als Admin eines Gruppen-Chats möchte ich meinen Chat öffentlich machen, damit Gleichgesinnte unserem Chat beitreten können.
29. Als Verein möchten wir mit der breiten Masse in Kontakt treten, um auf uns aufmerksam machen zu können.

8. User-Stories

30. Als Philomat möchte ich mich mit fremden Menschen austauschen können, um mein Wissen und Horizont erweitern zu können.

8.6. Profile

31. Als Benutzer möchte ich mein Profil anzeigen lassen, um zu sehen wie mich andere einsehen können.

32. Als Benutzer möchte ich mein Profil gestalten können, um meine Persönlichkeit inszenieren zu können.

33. Als Benutzer möchte ich von mir gemachte Fotos hochladen, damit andere mich sehen können.

34. Als Benutzer möchte ich meinen gesamten Freunden meinen Status mitteilen können, um nicht allen einzeln von meinem Zustand berichten zu müssen.

35. Als Freund möchte ich meinen Freunden meine derzeitige Handynummer zeigen lassen, damit sie mich anrufen können wenn nötig.

36. Als Freund von vielen Leuten mit ähnlichen Namen möchte ich sie auseinanderhalten können, um sie nicht aus Versehen zu verwechseln.

8.7. Contactlist

37. Als Freund will ich mehreren Leuten die gleiche Nachricht schicken können, um mich nicht wiederholen zu müssen.

38. Als Freund möchte ich eine Liste von allen meinen Freunden kriegen, um eine Übersicht meiner hinzugefügten Freunde zu haben.

39. Als Benutzer möchte ich andere Leute als Freund hinzufügen können, um mit neuen Leuten in Kontakt treten zu können.

8. User-Stories

40. Als Benutzer möchte ich Freunde löschen können, um meine Kontaktliste übersichtlich zu halten.

41. Als Benutzer Sozialer Netzwerke kann ich Freunde aus dem Sozialen Netzwerk auf die App übertragen, um schneller in der App mit ihnen in Kontakt zu treten.

8.8. Settings

42. Als Deutscher will ich auswählen können in welcher Sprache die App ist, damit ich die Texte verstehe.

43. Als Benutzer möchte ich meine Daten anpassen können, falls sich etwas ändern sollte.

44. Als Benutzer kann ich andere Leute die mich stören blockieren, um nicht belästigt zu werden.

45. Als Bilingualer möchte ich auswählen in welchen Sprachen in Globalen Chats gesprochen wird, um mein Sprachkenntnis üben zu können.

46. Als EU-Bewohner will ich eine Übersicht meiner Daten einsehen und diese ändern können, um die Kontrolle zu haben was die Anwendung über mich weiß.

8.9. Search

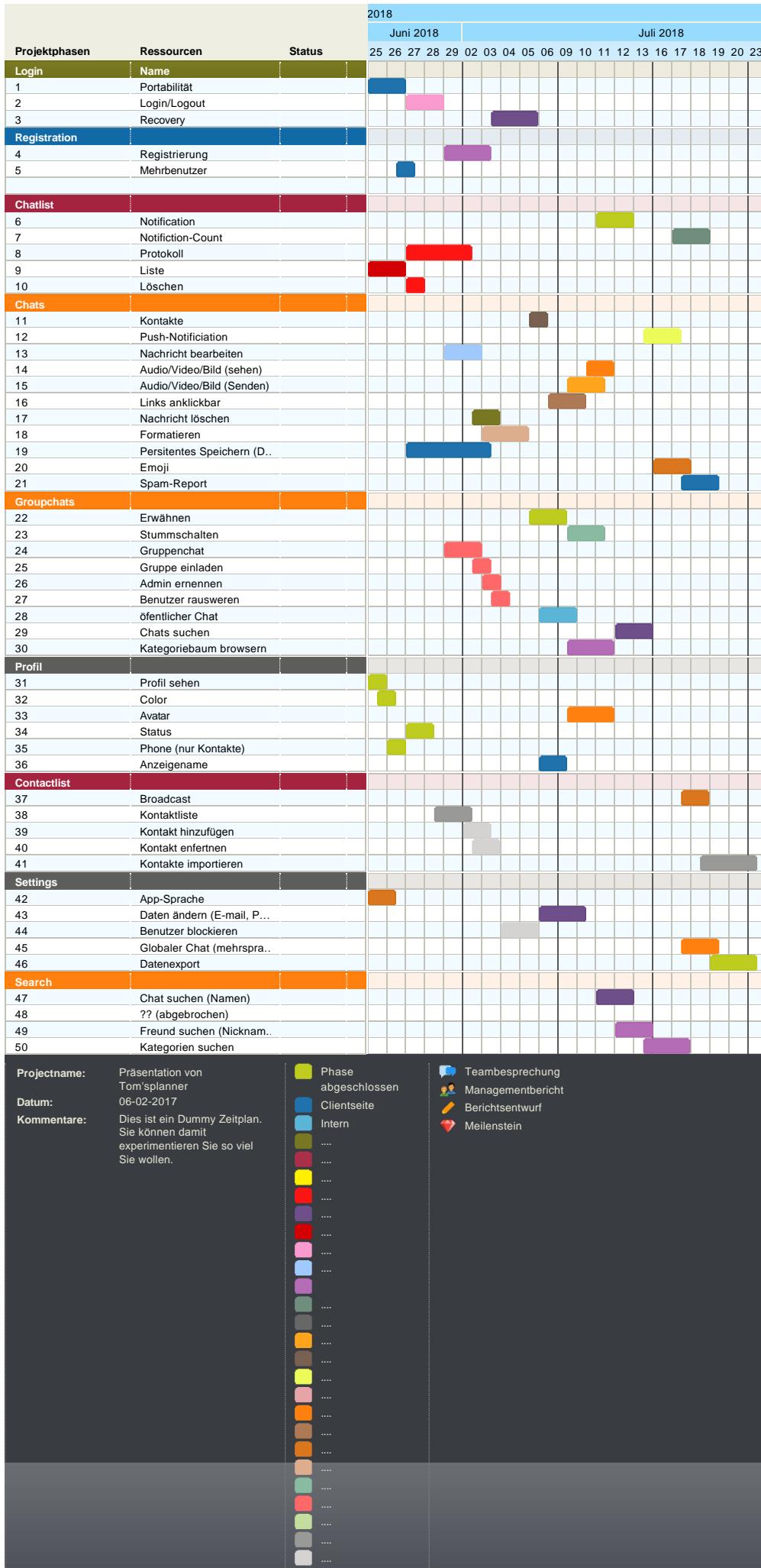
47. Als Benutzer kann ich bestimmte Chats über ihren Namen suchen, um schnell zu einer Konversation zu kommen.

48. Als Freund kann ich mit der Suche, über den Spitznamen meiner Freunde, jemand einfacher finden.

49. Als Freund kann ich mit der Suche über den Spitznamen meiner Freunde schauen, ob ein Freund in meiner Liste fehlt.

8. User-Stories

50. Als Hobbyspezialist möchte ich relevante Chats finden, um mich mit anderen Interessierten austauschen zu können.



A. Anhang

Abbildungsverzeichnis

1.	Model-View-ViewModel umfasst drei Komponenten mit dedizierten Aufgaben	8
2.	Datenfluss im Flux-Pattern	8
3.	GraphQL Funktionsweise	10
4.	WebSocket Funktionsweise	14
5.	Data Management ChatSphere	15
6.	GraphQL transport over websocket specification	17
7.	Veranschaulichung der Architektur	19
8.	Verteilungsdiagramm - ChatSphere-Artefakte auf den beteiligten Geräten	20
9.	Serverseite von ChatSphere	21
10.	ORY HYDRA Consent-Flow (Apache 2.0) aus den ORY Hydra Docs	23
11.	API Query Object Graph	26
12.	API Mutation Object Graph	27
13.	API Subscription Object Graph	28
14.	Apollo-link-state. Übernimmt die Aufgaben des zentralen Stores im Flux-Pattern.	30
15.	Apollo-link-state als Vermittler zwischen Komponenten.	31
16.	Komponentendiagramm des Frontends	32
17.	Wechsel zwischen Benutzeroberflächen der Single-Page-Application	34
18.	Registrierungs Modul. Benutzereingaben werden an den zentralen Store weitergeleitet	35
19.	NameView	36
20.	PasswordView	36
21.	EmailView	36
22.	Login Modul	38
23.	Aussehen der LoginView	39
24.	Mit der Navigation können andere Module aufgerufen werden	40
25.	Navigation Seitenleiste	41
26.	Das Profil Modul, um fremde Profile anzuzeigen zu können	42
27.	ProfileView Komponente	43
28.	Benutzer Optionen	43

Abbildungsverzeichnis

29.	Das Modul für Einstellungen, um die Anwendung anzupassen	46
30.	SettingsView	47
31.	Das Chatlist Modul, welches alle Chats, an denen man teilnimmt, auflistet	48
32.	Beispielhafte Liste von Chats	49
33.	Das Chat Modul zur Kommunikation mit Kontakten	50
34.	ChatView Komponente	51
35.	Das Modul zu Groupchats. Erweitert um die Teilnehmerliste	52
36.	Seitenleiste	53
37.	Das Modul Contactlist ist zur Auflistung aller Kontakte verantwortlich .	54
38.	ContactlistView	55
39.	Das Modul für die Suche	57
40.	Toolbar	58
41.	SearchView	58
42.	Baumdiagramm Router-Links	60
43.	Aktivitätsdiagramm zur Offline-Funktion eines Service Workers	62
44.	Sequenzdiagramm für beispielhafte Push-Notification	63
45.	Gesamtübersicht der Datenbank	65
46.	Datenspeicherung von Dateien verschiedener Dateitypen.	66
47.	Verwaltung einzelner Benutzerprofileigenschaften	67
48.	Datenhaltung der Benutzereinstellungen	68
49.	Organisation und Struktur der Chats	69
50.	Backend Paketdiagramm	70
51.	Backend UML	71
52.	Client Server Kommunikation	72

Tabellenverzeichnis

A.1. Glossar

API Von einem System zur Verfügung gestellte Teil eines Programms, der zur Anbindung an das eigene System benutzt werden kann.

Client-Server-Architektur Softwarestruktur für verteilte Anwendungen. Eine Server bietet eine Dienstleistung an, die von mehreren Clients in Anspruch genommen wird (Übersetzt: Kellner-Kunde-Architektur).

Design-Patterns Design-Patterns (deutsch: Entwurfsmuster) sind wiederverwendbare Vorlagen zur Lösung von gängigen Problemen die sich in der Softwarearchitektur häufig finden.

Framework Ein Rahmengerüst in welchem nach dem → Hollywood-Prinzip eigener Code eingefügt wird.

gebunden Siehe Kapitel 1.2 zu Model-View-ViewModel. Eingabelemente (Textfelder, Buttons, ...) müssen an Attribute "gebunden" werden, damit das ViewModel (die UI-Logik) Ereignisse und Änderungen wahrnehmen kann (vergleichbar mit Beobachter-Muster: Mit den Eingabelementen als Subjekt und den Attributen als Beobachter).

GraphQL SDL In der »GraphQL Schema Definition Language« werden GraphQL-APIs spezifiziert.

Hollywood-Prinzip Nach dem Motto »Don't call us, we call you« wird eigener Programmcode an vorgesehenen Stellen hinzugefügt und durch ein → Framework automatisch aufgerufen.

Instant Messaging (IM) Ein Mittel oder System zur unmittelbaren Übertragung von elektronischen Nachrichten.

Tabellenverzeichnis

Java Eine plattformunabhängige Objektorientierte Programmiersprache, welche durch

Tabellenverzeichnis

Reverse-Proxy Ein Proxy ist ein Vertreter, der anstelle einer selbst agiert. Während ein Forward-Proxy Anfragen für einen Client ausführt, beantwortet ein Reverse-Proxy Anfragen anstelle eines Servers. Der Proxy leitet häufig antwortet weiter und übernimmt zusätzliche Aufgaben wie Caching (Zwischenspeichern), Encryption (Verschlüsselung) oder überprüfung von vorgaben wie Policys (Richtlinien).

Servlet Ein Container welcher von einem Webserver ausgeführt wird und Anfragen beantwortet.

SQL Mit der »Structured Query Language« werden Anfragen an einen SQL-Datenbankserver formuliert.

View-Komponente View-Komponenten sind → Komponenten eines Moduls die sich ausschließlich mit der Oberfläche der Anwendung beschäftigen. Sie gehören im Flux-Pattern der Kategorie des Views an. Implementiertechnisch lassen sich View-Komponenten 1:1 auf Single-File-Components von Vue.js abbilden. Somit lassen sich ähnlich wie Klassen aus Java modellieren.

Webanwendung .

Tabellenverzeichnis

MIT License

Copyright (c) 2018 Alexander Wank, Niklas Seyfarth, Julian Miedjil, Berthold Niemann & Alexander Brese

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.