

Hochschule Aalen
Fakultät Informatik
Sommersemester 2022
Team G

Cloud and Distributed Computing

Software Architecture Document

Haushaltsapp

Revision 1.0.0 - Juni 2022

Bearbeiter des Projekts

Simon Ruttmann	80751
Michael Ulrich	77607
Veronika Scheller	79888

Simon Ruttmann	80751@studmail.hs-aalen.de
Michael Ulrich	77607@studmail.hs-aalen.de
Veronika Scheller	79888@studmail.hs-aalen.de

18. Juni 2022

Inhaltsverzeichnis

1	Einführung und Ziele	3
1.1	Anforderungsbeschreibung	3
1.2	Zielbeschreibung	5
2	Randbedingungen	7
2.1	Technische Randbedingungen der Clientseite	7
2.2	Technische Randbedingungen der Serverseite	7
2.3	Organisatorische Randbedingungen	9
3	Kontextabgrenzung	10
3.1	Akteuere	10
3.2	Technischer Kontext:	
	Kommunikationskanäle & Protokolle	11
3.2.1	statbureau:	11
3.2.2	User:	11
4	Lösungsstrategie	12
4.1	Technologieentscheidungen	12
4.2	Top-Level-Architektur	13
5	Bausteinsicht	14
5.1	Whitebox Gesamtsystem	14
5.1.1	Blackbox Frontend	15
5.1.2	Blackbox Userservice	16
5.1.3	Blackbox Contentservice	17
5.1.4	Blackbox Inflationsservice	18
5.1.5	Blackbox Advertisementservice	18
5.1.6	Blackbox Chatsservice	19
5.1.7	Blackbox Relational Databasemodule	20
5.1.8	Blackbox Documentbased Databasemodule	20
5.1.9	Blackbox In-Memory Databasemodule	21
5.1.10	Blackbox Data Transfer Objects and Validation	21
5.1.11	Whitebox Frontend	22
5.1.12	Whitebox Userservice	24
5.1.13	Whitebox Contentservice	25
5.1.14	Whitebox Inflationsservice	26
5.1.15	Whitebox Advertisementservice	26
5.1.16	Whitebox Chatsservice	27
5.1.17	Whitebox Relational Databasemodule	28
5.1.18	Whitebox Documentbased Databasemodule	29
5.1.19	Whitebox In-Memory Databasemodule	29
5.1.20	Whitebox Data Transfer Objects and Validation	30
6	Laufzeitsicht	31
6.1	User Service	31

6.2	Content Service	35
6.3	Chat Service	36
7	Verteilungssicht	37
8	Querschnittliche Konzepte	38
8.1	Persistence	38
8.1.1	Entity-Relationship-Diagramm	38
8.1.2	Relationales Diagramm	38
8.1.3	Redis Datenmodel	39
8.1.4	MongoDB Datenmodel	39
8.2	REST-API's	40
8.2.1	Userservice	40
8.2.2	Contentservice	41
8.2.3	Inflationsservice	42
8.2.4	Advertisementsservice	42
8.2.5	Chatservice	42
8.3	User Interface	42
8.4	Communication/Integration	43
8.4.1	Inflation API	43
8.4.2	Keycloak	43
8.4.3	Spring Boot	43
8.4.4	Hibernate	43
9	Risiken und technische Schulden	44
10	Architekturentscheidungen	45
11	Anhang	46
11.1	Klassendiagramm des Backends	46
11.2	Wireframe	47

1 Einführung und Ziele

Dokumentation: Simon Ruttmann

Umsetzung: Simon Ruttmann, Veronika Scheller, Michael Ulrich, Robin Röcker

Dieses Dokument beschreibt die vorliegende Software 'Haushaltsapp'. Bei der Anwendung handelt es sich grundlegend um ein benutzergetriebenes Managementtool zur Erfassung, Verwaltung und Visualisierung von Ausgaben und Einnahmen.

Das vorliegende Dokument ist nach dem arc42 Template Revision 8.0 DE aufgebaut.

1.1 Anforderungsbeschreibung

Fortfolgend werden in dieser Sektion die wesentlichen Anforderungen der Anwendung 'Haushaltsapp' beschrieben.

Da es sich um ein Agil getriebenes Projekt handelt, wurden die Anforderungen nach agilen Techniken in Form von User Stories ermittelt.

Zudem wurden den User Stories die Prioritäten 'Very High', 'High', 'Medium', 'Low' und 'Very Low' zugeordnet und anschließend nach ihrer optimalen Bearbeitungsreihenfolge nummeriert.

Number	Priority	User Story
1	Very High	As a user i want to be able to register a new account so that i can have a personal space for my data
2	Very High	As a user i want to log in so that i can enter my personal space and see my data
3	Very High	As a user i want to log out so that i can be sure no one else on my pc can see my data
4	Very High	As a user i want to create saving entries for me so that i can view them later
5	Very High	As a user i want to see all my saved entries to receive an overview
6	High	As a user i want to edit saving entries as entries may change or i entered something wrong
7	High	As a user i want to delete saving entries as they may no longer be relevant

Number	Priority	User Story
8	Medium	As a user i want to be able to categorize my saving entries
9	Medium	As a user i want to sort saving entries by a category
10	Medium	As a user i want to be able to edit a category, if i accidentally changed one, or found a better name for it
11	Medium	As a user i want to be able to delete a category, including all the associated entries as i might don't need them anymore
12	Medium	As a user i want to be able to create a group so that i can add my shared appartment mates, or my family
13	Medium	As a user i want to be able to receive and accept invitations to an existing group so that i can enter that group
14	Medium	As a user i want to be able to leave a group so that i don't see their data anymore
15	Medium	As a user i want to switch views so that i can see only groupspecific data or my data
16	Medium	As a user i want to create and edit saving entries not for me, but for my group
17	Medium	As a user i want to categorize the saving entries of the group
18	Medium	As a user i want to see which person in the group has saving entries
19	Low	As a user i want to be in multiple group at the same time
20	Low	As a user i want to see all saving entries on my mobile device
20	Low	As a user i want to see a visualization of my summarized expenditures so that i can quickly overview
21	Low	As a user i want to see a visualization of my past activities, to evaluate myself

Number	Priority	User Story
22	Low	As a user i want to be able to predict the costs in the future
23	Very Low	As a user i want to be able to chat within a group
24	Very Low	As a user i want to see all recent and old messages when chatting withing a group

1.2 Zielbeschreibung

Diese Sektion beschreibt die Ziele, nach denen die Anwendung gemessen wird.

Die dafür herangezogenen Kriterien sind auf den ISO 20150 Standard zurückzuführen und wurden um weitere für das Projekt spezifische Kriterien ergänzt.

Folgend sind die Kriterien mit der Beschreibung des Zielszustandes dokumentiert.

Kriterium	Beschreibung
Funktionale Eignung	Vollständig hinsichtlich User-Story definierter Softwarefunktionen. Funktional korrekt
Zuverlässlichkeit	Clientanwendung ist Fehlertolerant gegenüber Falscheingaben von Benutzern. Die Verfügbarkeit und Zuverlässigkeit der Serveranwendung wird durch fehlerhafte Anfragen nicht beeinflusst
Sicherheit	Die Anwendung schützt alle sensiblen Daten der Benutzer, wie z.B. Passwörter oder Spareinträge von unautorisierter Einsicht oder Manipulation
Wartbarkeit	Die Anwendung ist modular aufgebaut und nutzt wiederverwendbare Komponenten. Dies gilt sowohl für clientseitige Visualisierungsbausteine also auch für geteilte funktionale Komponenten der Serveranwendung

Leistungsfähigkeit	Die Clientanwendung muss, insbesondere bei großer Datenlast, unmittelbar auf Usereingaben reagieren. Dies gilt ebenso bei der vollständige Darstellung des Sachverhalts bei der initialien Ladung.
Bedienbarkeit	Die Funktionen der Anwendung müssen intuitiv bedienbar sein.
Kompatibilität	Teillösungen der Anwendung, z.B. Module, Services müssen kompatibel zueinander sein.
Übertragbarkeit	Die Anwendung muss mittels einer Virtualisierungslösung betreibbar sein
Skalierbarkeit	Die Anwendung muss gesteigerte Leistungsanforderungen durch Skalierung oder Vermehrung der Einzelkomponenten erfüllen können

2 Randbedingungen

Dokumentation: Simon Ruttmann

Umsetzung: Vorgegeben

In dieser Sektionen werden Randbedinungen und Vorgaben der Anwendung "Haushaltsapp" beschrieben, welche direkten Einfluss auf den Entwurf, die Implementierung und den Entwicklungsprozess nehmen.

Aufgrund der großen Breite an Vorgaben ist eine tabellarische Darstellung nicht geeignet. Daher werden die folgenden Randbedingungen überwiegend in textueller und stichpunktartiger Form beschrieben.

2.1 Technische Randbedingungen der Clientseite

Die vorliegende Anwendung muss als Web-Anwendung implementiert werden. Darüber hinaus muss die hierfür zu entwickelnde UI mittels eines der folgenden Frontend-Webframeworks realisiert werden:

- React
- Angular
- Vue.js
- Swelte
- Ionic

Die Anwendung muss zudem für mobile Endgeräte optimiert sein.

2.2 Technische Randbedingungen der Serverseite

Die zu entwickelnde Software muss einer Microservice-Architektur folgen. Dies inkludiert, das jeder Mirkroservice und jede verwendete Datenbank der Anwendung mittels des Tools Docker in einem abgeschlossenen Container auszuführen ist.

Die zu implementierende Anwendung muss Echtzeitdaten mittels eines Subscribe/Publish Systems erfassen und verarbeiten.

Die Realisierung des Backends muss mittels einer der folgenden Umgebungen realisiert werden.

- NodeJS
 - Unter Verwendung von Express
 - Unter Verwendung von LoopBack
- Python
 - Unter Verwendung von Falcon
 - Unter Verwendung von Flask
 - Unter Verwendung von Django REST Framework
- Spring Boot
- KTor
- Go
 - Unter Verwendung von Go Micro
 - Unter Verwendung von Go Kit
 - Unter Verwendung von Gizmo
 - Unter Verwendung von Kite

Die in der Anwendung implementierten Endpunkte müssen mittels eines der folgenden Tools dokumentiert und getestet werden.

- Swagger
- apiDoc

Zudem muss mindestens ein Teil der Persistierung von Daten der Anwendung mittels NoSQL Datenbanksystemen realisiert werden. Zusätzlich ist explizit die Nutzung eines SQL Datenbanksystems erwünscht.

Eine weitere Vorgabe des Projekts ist das Deployment der Anwendung mittels einer der folgenden Plattformen:

- bwCloud
- Azure

2.3 Organisatorische Randbedingungen

Die Entwicklung muss unter der Verwendung von Git und Trello stattfinden. Hierfür wurde ein Bitbucket-Repository und Trello-Board zur Verfügung gestellt, welche regelmäßig genutzt und aktualisiert werden müssen.

Der Entwicklungsprozess hat Agil zu erfolgen.

Während der Entwicklung sind darüber hinaus Meilensteine definiert, welche zu einem festgelegten Zeitpunkt bewältigt werden müssen. Zu jedem Meilenstein ist ein Meeting durchzuführen, in dem der aktuelle Entwicklungsstand vorgestellt und validiert wird.

Zu folgenden, zeitlich fest vorgegebenen Meilensteinen ist Bericht zu erstatten.

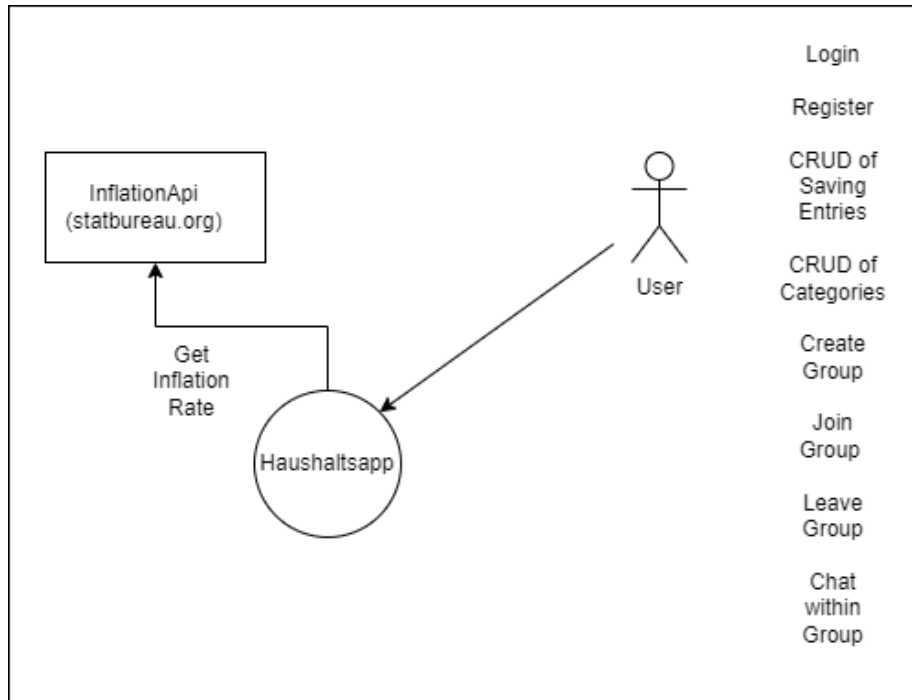
Exploratory Sprint Review	12.04.2022
Architectural Spike Sprint Review	03.05.2022
Alpha	17.05.2022
Beta	14.06.2022
Certification	28.06.2022

Zu jedem oben genannten Termin ist der aktuelle Zustand des zu verwendenden Trello-Boards, in Form von Screenshots, einzureichen.

3 Kontextabgrenzung

Dokumentation: Michael Ulrich, Simon Ruttmann

Umsetzung: Veronika Scheller, Michael Ulrich, Simon Ruttmann



3.1 Akteure

statbureau.org: Externe API, welche zur Beschaffung der Inflationsrate vom InflationService verwendet wird. Eine Datenanfrage (GET-Request) wird einmal bei Start des Services gesendet und dann alle 10 Tage wiederholt.

User: Ein User ist der Hauptnutzer des Systems und kann über die Benutzeroberfläche mit diesem interagieren.

3.2 Technischer Kontext: Kommunikationskanäle & Protokolle

3.2.1 statbureau:

HTTP GET Request

3.2.2 User:

Eingaben (Nur HTTP)

- Login
- Register
- Einträge erstellen, bearbeiten und löschen
- Kategorien erstellen, bearbeiten und löschen
- Gruppen erstellen, beitreten und löschen
- Andere Nutzer zu bestehenden Gruppe einladen
- Nachrichten innerhalb einer Gruppe senden

Ausgaben (HTTP & WS)

- Einsehen von Gruppen bei denen der User Mitglied ist (HTTP)
- Einsehen von Gruppeneinträgen (HTTP)
- Einsehen von angelegten Kategorien (HTTP)
- Einsehen von gesendeten Nachrichten innerhalb der Gruppe (HTTP & WS)
- Einsehen von gefilterten Daten (HTTP)

4 Lösungsstrategie

Dokumentation: Veronika Scheller, Simon Ruttmann

Umsetzung: Veronika Scheller, Michael Ulrich, Simon Ruttmann

In dieser Sektion wird ein kurzer Überblick über grundlegende Entscheidungen und Lösungsansätze vermittelt.

4.1 Technologieentscheidungen

In dieser Sektion werden im folgenden die Entscheidungen für die eingesetzte Technologien erläutert.

Für die Entwicklung des Frontends wurde sich für React entschieden.

Dies liegt darin begründet, dass die Entwickler dieses Projekts mit dieser Technologie bereits Erfahrung besitzen.

Durch die bestehenden Erfahrungen sind die Möglichkeiten und Grenzen der Javascript-Library bekannt und konnten mit den Anforderungen an dieses Projekt, grundlegend beschrieben in Kapitel 1.2, gegenübergestellt werden.

Das Framework hierbei bietet 'Out of the box' bereits einen großen Teil an Funktionen, mit denen das Frontend schnell und effizient entwickelt werden kann, ohne dabei mit einem großen Overhead konfrontiert zu werden. Dies wäre z.B. mit dem vollwertigen MVC-Framework Angular der Fall.

Durch den komponentenbasierten Architekturansatz gewährleistet React zudem eine skalierbare Entwicklung des Frontends.

Die Entwicklung des Backends geschieht auf Basis von Java mittels des Springboot-Frameworks. Dies liegt darin begründet, dass dies aus den gegebenen technischen Rahmenbedingungen in Kapitel 2.2, neben Python, die einzige stark typisierte, objektorientierte Sprache ist.

Durch die Verwendung von Java kann auf bereits bekannte objektorientierte Konzepte zurückgegriffen werden. Welches es erleichtert skalierbare, wartungsfreie und fehlerfreie Software zu entwickeln.

Python kann aufgrund seiner mangelnden Performance die wesentlichen Anforderungen aus Kapitel 1.2 nur unzureichend erfüllen und ist daher nicht einsetzbar.

Die Virtualisierungstechnologie Docker wurde in Kapitel 2.2. vorgegeben.

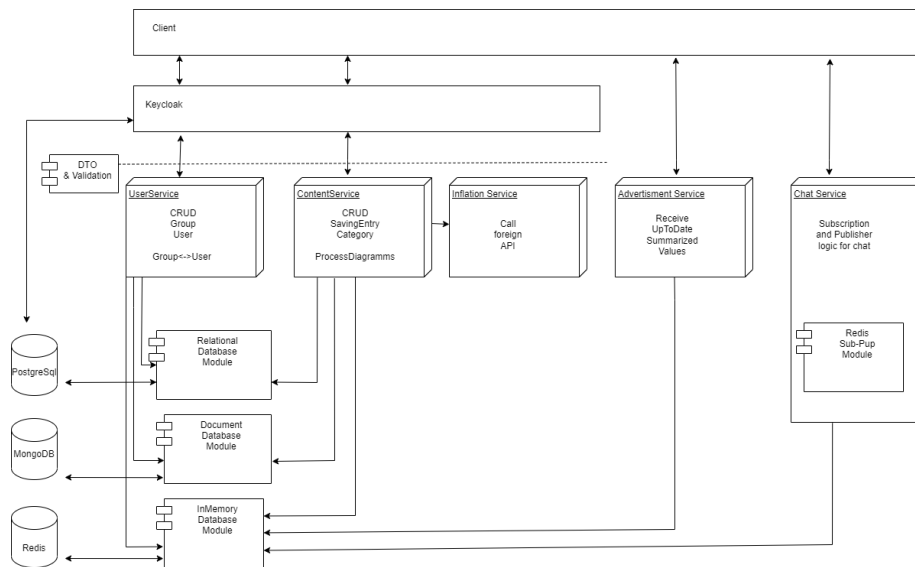
Da auf die Anwendung eine Reihe von sensiblen Daten erfasst, visualisiert und verwaltet, sind die Anforderungen an die Sicherheit der Anwendung sehr hoch. Daher wurde sich für die Verwendung des Open-Source-Softwareprodukts Keycloak entschieden. Keycloak ermöglicht hierbei Single Sign-On mit Identity and Access Management für moderne Anwendungen und Dienste.

4.2 Top-Level-Architektur

Im folgenden werden die wichtigsten architektonischen Entscheidungen und Lösungsansätze zur Entwicklung der Anwendung 'Haushaltsapp' erläutert.

Die Anwendung wird gemäß der Anforderungen aus den technischen Randbedingungen der Serverseite mittels einer Microservice-Architektur realisiert.

Im folgenden ist die Top-Level-Architektur der Software dargestellt.



Die Anwendung ist im wesentlichen in Microservices, Module, Datenbanken, dem Reverse Proxy Keycloak und den Client unterteilt.

Der Client stellt Anfragen an die Services. Diese passieren den reverse Proxy, welcher die Anfragen authentifiziert oder ablehnt. Die Services nehmen die ankommenden Anfragen an und führen ihre Geschäftslogik unter der Verwendung der implementierten Module aus.

Die Module bieten hierbei Funktionalität an, welche von mehreren Services genutzt werden. Dies betrifft z.B. die Persistierung von Objekten oder Validierung von gemeinsam genutzten Data Transfer Objects.

Die Verwendung von Modulen ermöglicht die gemeinsame Nutzung von Funktionen, wodurch die Anwendung frei von Codedupikationen bleibt und dadurch wartbarer ist.

Der Einsatz von mehreren verschiedenen Datenbanksystemen entspringt den Anforderungen aus den technischen Randbedingungen.

Zum anderen ermöglicht der Einsatz dieser Datenbanksysteme eine effiziente und skalierbare Umsetzung der Anforderungen bei gleichzeitiger Einhaltung der definierten Zielbeschreibungen.

5 Bausteinsicht

Dokumentation: Veronika Scheller, Michael Ulrich, Simon Ruttmann

Umsetzung: Veronika Scheller, Michael Ulrich, Simon Ruttmann, Robin Röcker

Eine genauere Aufteilung der Umsetzung ist in der Readme des Quellcode-Ordners im Projekt beschrieben.

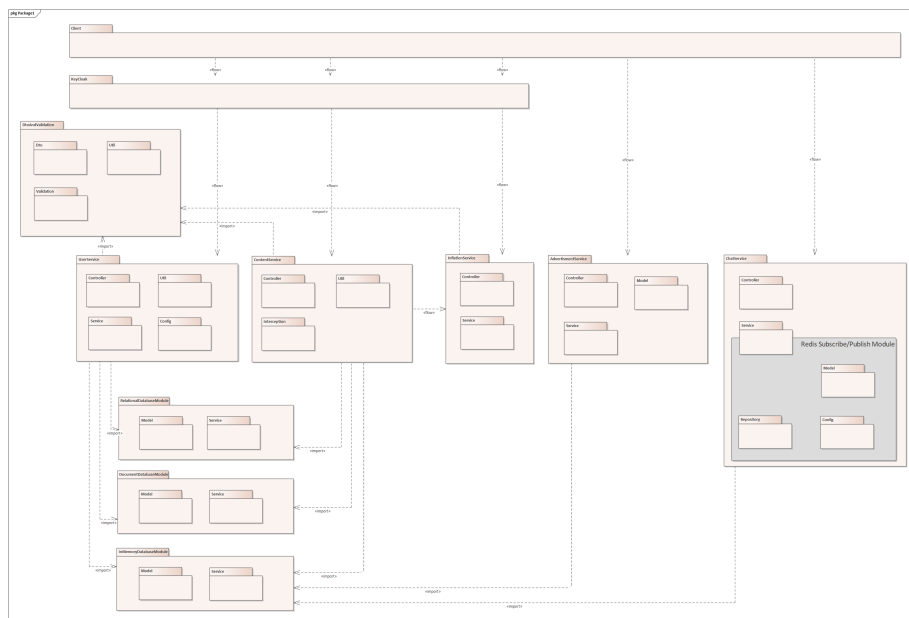
In dieser Sektion wird eine hierarchische Sammlung von Black- und Whiteboxen behandelt, welche zur Realisierung der Anwendung 'Haushaltsapp' implementiert worden sind.

Im folgenden wird hierarchisch auf die wichtigsten Bausteine in kurzer und vereinfachter Form eingegangen.

5.1 Whitebox Gesamtsystem

Die Zerlegung des Gesamtsystems erfolgt mittels fünf extern erreichbarer Services, dem Client, dem Keycloak-Reverseproxy, sowie vier Modulen.

Im folgenden ist die Zerlegung des Gesamtsystems dargestellt. Die Datenbanksysteme sind in der Grafik nicht dargestellt und können dem Top-Level-Architekturdiagramm entnommen werden.



Die Anwendung wurde entsprechend ihres Domänenmodells aufgeteilt. So wurde unter anderem die Verwaltung von Usern und Gruppen von der Verwaltung der Datenstämme dieser getrennt. Die Services, welche diese Funktionen implementieren bilden den Kern der Anwendung.

Zusätzliche Funktionen wie z.B. das Versenden und Erhalten von Nachrichten oder die Einbindung von externen API's wurde über jeweils separate Services realisiert. Diese erweitern den Kern der Anwendung ohne direkten Einfluss auf diesen zu nehmen, wodurch die Stabilität und Sicherheit der Anwendung erhöht wird.

Die Beschreibungen und Verantwortungen der einzelnen Bausteine können der folgenden Black- und Whitebox Beschreibung entnommen werden.

5.1.1 Blackbox Frontend

Verantwortungen

- Bietet die Interaktionsschnittstelle mit dem Benutzer
- Stellt Anfragen an das Backend, um Daten zu erhalten und diese zu visualisieren
- Wertet Benutzeraktionen aus und stellt ggf. Anfragen an Services des Backends um Daten zu manipulieren
- Hält seinen eigenen Client-Store für erhaltene Daten aus dem Backend

Schnittstellen

Diese Komponente nutzt einen Großteil der in den Backend definierten Schnittstellen. Darunter fallen die Schnittstellen der folgenden Services:

- Userservice
- Contentservice
- Advertisementservice
- Chatservice

Qualitäts-/Leistungsmerkmale

- Responsive Design
- Clientseitige Datenspeicherung unter Verwendung eines Redux-Stores
- Visualisierung der Daten in Diagrammen und Tabellen
- Filtereinstellungen
- Ansichten zur Modifizierung oder Erstellung von Daten
- Chatsystem mit Echtzeitdatenverarbeitung

Daten

Alle zu verwaltenden Daten werden in einem clientseitigen Speicher abgelegt. Die Daten werden mittels Http-Requests oder Ws-Verbindungen vom Backend erhalten. Vom Benutzer generierte oder modifizierte Datenbestände werden an das Backend übermittelt.

5.1.2 Blackbox Userservice

Verantwortungen

- Erstellung und Verwaltung von Usern
- Erstellung und Verwaltung von Gruppen
- Erstellung und Verwaltung von Einladungen

Schnittstellen

Dieser Service bietet folgende Rest-Schnittstellen an:

invitation-controller		^
PUT	/userservice/invitation/decline/{groupId}	▼ 🔒
PUT	/userservice/invitation/accept/{groupId}	▼ 🔒
POST	/userservice/invitation/invite	▼ 🔒
GET	/userservice/invitation/receive	▼ 🔒
group-controller		^
POST	/userservice/group/register	▼ 🔒
GET	/userservice/group	▼ 🔒
DELETE	/userservice/group/{groupId}	▼ 🔒
DELETE	/userservice/group/leave/{groupId}	▼ 🔒
user-controller		^
GET	/userservice/user	▼ 🔒
GET	/userservice/user/usernames	▼ 🔒

Weitere Informationen unter Rest-API's

Qualitäts-/Leistungsmerkmale

- Benutzerdaten sind aufrufbar
- Erstellen, löschen, verlassen und bereitstellen von Gruppen
- Versenden, bekommen, akzeptieren und ablehnen von Einladungen
- Die Anfragen an diesen Service werden mittels zusätzlicher Sicherheitsmechanismen auf Authorisierung geprüft

Daten

Alle Daten, welche in direkter Verantwortung des Userservices liegen, werden über Zugriff auf das relationale Datenbankmodul auf der PostgreSQL Datenbank abgelegt.

Gegebenenfalls erstellt der Userservice initial Stammdaten und legt diese unter Benutzung des dokumentenbasierten Datenbankmoduls auf der MongoDB ab. Die dort erstellten Daten werden nicht weiter von diesem Service verwaltet.

5.1.3 Blackbox Contentservice

Verantwortungen

- Erstellen, editieren und löschen von Usercontent
- Durchführung komplexer Berechnungen auf Gruppendaten

Schnittstellen

Dieser Service bietet folgende Rest-Schnittstellen an:

saving-entry-controller		^
GET	/content/savingEntry/{groupId}	▼ 🔒
PUT	/content/savingEntry/{groupId}	▼ 🔒
POST	/content/savingEntry/{groupId}	▼ 🔒
GET	/content/savingEntry/{groupId}/{savingEntryId}	▼ 🔒
DELETE	/content/savingEntry/{groupId}/{savingEntryId}	▼ 🔒
category-controller		^
GET	/content/category/{groupId}	▼ 🔒
PUT	/content/category/{groupId}	▼ 🔒
POST	/content/category/{groupId}	▼ 🔒
GET	/content/category/{groupId}/{categoryId}	▼ 🔒
DELETE	/content/category/{groupId}/{categoryId}	▼ 🔒
processing-controller		^
GET	/content/processing/{groupId}	▼ 🔒
POST	/content/processing/{groupId}	▼ 🔒

Weitere Informationen unter Rest-API's

Qualitäts-/Leistungsmerkmale

- Erstellen, löschen, editieren und bereitstellen von Spareinträgen
- Erstellen, löschen, editieren und bereitstellen von Kategorien
- Berechnung, Aufbereitung und Versand von Benutzerkontent
- Die Anfragen an diesen Service werden mittels zusätzlicher Sicherheitsmechanismen auf Authorisierung geprüft

Daten

Alle Persistenzdaten des Contentservice, werden innerhalb der MongoDB unter Benutzung des Dokumentdatabase Moduls abgelegt.

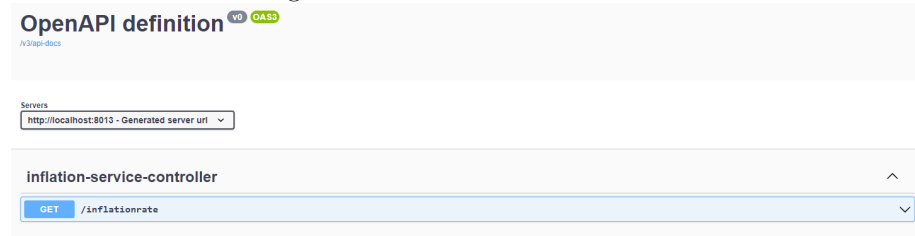
5.1.4 Blackbox Inflationsservice

Verantwortungen

- Bereitstellung von Inflationsdaten
- Regelmäßige Aktualisierungen der Inflationsdaten

Schnittstellen

Dieser Service bietet folgende Rest-Schnittstelle an:



Weitere Informationen unter Rest-API's

Qualitäts-/Leistungsmerkmale

- Antwortet unabhängig von der externen Api auf Datenanfrage
- Erhält in regelmäßigen Abständen aktualisierte Daten von der externen Api

Bekannte Probleme/Risiken

- Riskante singuläre Abhängigkeit von der referenzierten API

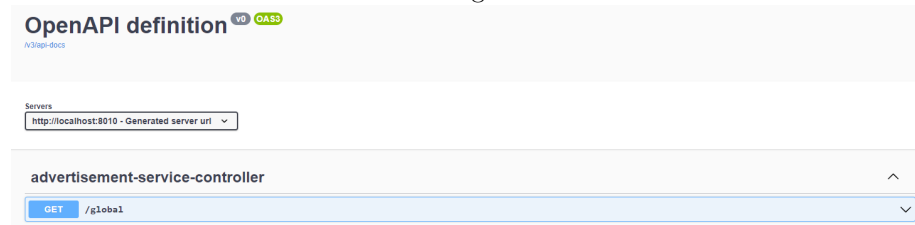
5.1.5 Blackbox Advertisementservice

Verantwortungen

- Bereitstellung von Werbedaten (Anzahl versendeter Nachrichten, registrierter User und erstellter Spareinträge)
- Bereitstellung der Frontend Ressourcen

Schnittstellen

Der Service bietet einen Endpunkt für die Bereitstellung der Frontend-Ressourcen. Darüber hinaus bietet er folgende Rest-Schnittstelle an:



Weitere Informationen unter Rest-API's

Qualitäts-/Leistungsmerkmale

- Bereitstellung aller erlaubten Frontend-Ressourcen
- Liefert auf Anfrage alle Werbedaten

Daten

Die Werbedaten liegen innerhalb der Redis Datenbankinstanz. Auf die Daten wird mittels des Redis-Moduls zugegriffen.

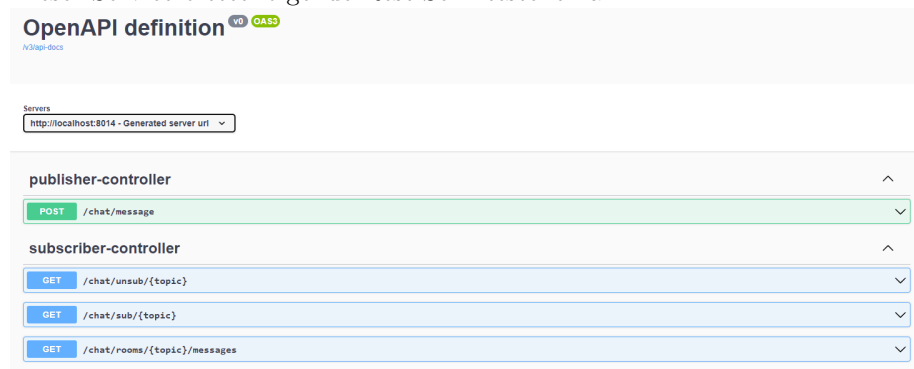
5.1.6 Blackbox Chatservice

Verantwortungen

- Versand und Empfang von Chatnachrichten
- Persistenz von Chatnachrichten

Schnittstellen

Dieser Service bietet folgende Rest-Schnittstellen an:



Weitere Informationen unter Rest-API's

Qualitäts-/Leistungsmerkmale

- Benachrichtigt alle verbundenen Nutzer einer Gruppe sobald an diese eine Nachricht gesendet wird
- Alle alten Nachrichten einer Gruppe sind abrufbar
- Es kann eine Websocket Verbindung zum Service hergestellt werden
- Es können neue Nachrichten an eine Gruppe gesendet werden

Daten

Alle Persistenzdaten des ChatSERVICE, werden innerhalb des Redis Moduls abgelegt. Aktive Verbindungsinformationen werden vom Service selbst gehandhabt.

Bekannte Probleme/Risiken

- Verbindung zwischen Backend und Redis wird nicht getrennt, kann bei hohem Gruppenaufkommen zu Bottlenecks führen

5.1.7 Blackbox Relational Databasemodule

Verantwortungen

- Persistierung von Gruppen und Personen, sowie deren Beziehungen zu einander

Schnittstellen

Für die Persistierung von Daten ist die Schnittstelle `IRelationalDatabaseService` nach außen verfügbar. Die Schnittstelle wird vom `UserService` und dem Sicherheitssystem des `Contentservice` genutzt.

Qualitäts-/Leistungsmerkmale

- Bereitstellen von Methoden zur Persistierung von Objekten auf der relationalen Datenbank
- Stellt persistierbare Objekte zur Verfügung

Daten

Das Modul hält eine konstante Verbindung zur relationalen Datenbank PostgreSQL, um Queries auf dieser abzusetzen.

5.1.8 Blackbox Documentbased Databasemodule

Verantwortungen

- Persistierung von Daten auf der MongoDB

Schnittstellen

Für die Persistierung von Daten ist die Schnittstelle `IDocumentDatabaseService` nach außen verfügbar. Die Schnittstelle wird vom `User-` und `Contentservice` genutzt.

Qualitäts-/Leistungsmerkmale

- CRUD-Operationen werden zur Verfügung gestellt
- Konsistenz bei Löschung oder Editierung von Kategorien wird gemäß Anforderungen sichergestellt.

Daten

Das Module hält eine konstante Verbindung zur MongoDB, um Queries auf dieser abzusetzen.

5.1.9 Blackbox In-Memory Databasemodule

Verantwortungen

- Persistenz von Daten auf der Redis Datenbank
- Bereitstellung von Daten aus der Datenbank
- Einfacher Verbindungsaufbau zur Redis Datenbank

Schnittstellen

Für die Persistierung von Daten ist die Schnittstelle `IRedisDatabaseService` nach außen verfügbar. Die Schnittstelle wird vom `UserService`, `Contentservice`, `AdvertisementService` und `ChatService` genutzt, um relevante Counter zu erhöhen und bereitzustellen, sowie um Chatnachrichten zu persistieren und versendete Nachrichten zu erhalten.

Qualitäts-/Leistungsmerkmale

- Inkrementierung und Bereitstellung von relevanten Zählern
- Bereitstellung und Persistierung von Chatnachrichten
- Verbindungsherstellung zur Datenbank

Daten

Über das Modul kann eine Verbindung zur einer Redis Datenbank hergestellt werden. Auf dieser werden die Daten abgelegt.

5.1.10 Blackbox Data Transfer Objects and Validation

Verantwortungen

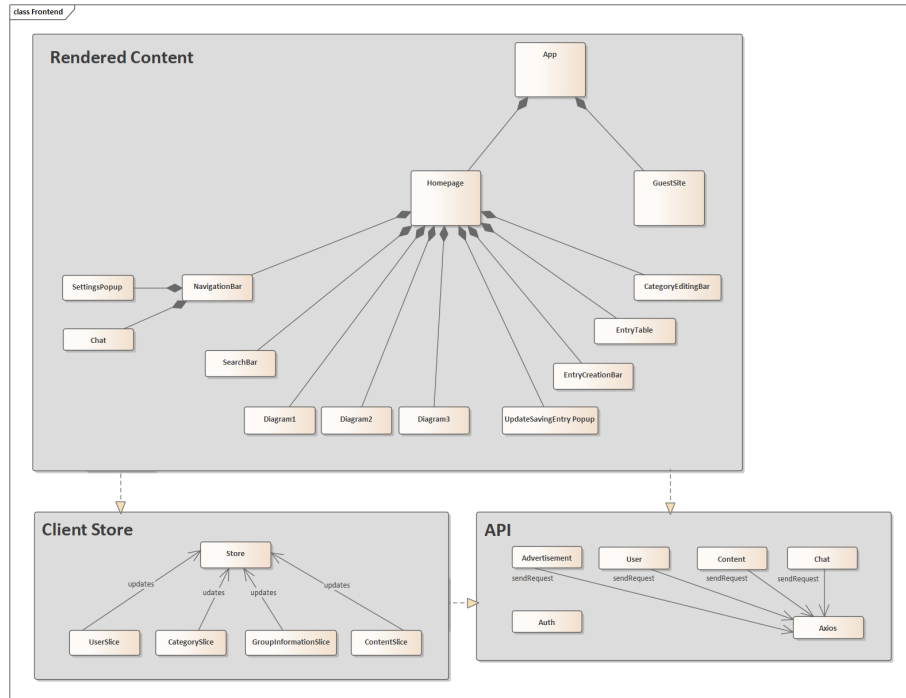
- Kollektion von gemeinsam genutzten Data Transfer Objects
- Beinhaltet für erhaltene Data Transfer Objects Validatoren um diese zu prüfen.

Schnittstellen

Die angebotenen Data Transfer Object sind nach Import des Moduls verfügbar. Validatoren können über die `ValidatorFactory` erhalten werden.

5.1.11 Whitebox Frontend

Im folgenden ist eine vereinfachte Darstellung des Frontends dargestellt.



Das Frontend ist grundlegend in drei Module unterteilt.

Bei dem in der Grafik dargestellten Render-Content handelt es sich um eine Kollektion von React-Components, welche jeweils einen Teil der UI darstellen.

Die Startseite wird vollständig von der Komponente 'GuestSite' gerendert. Innerhalb der Seite kann auf eine Login- und Registerseite navigiert werden. Diese Seiten sind in der vorliegenden Grafik nicht dargestellt, da sie von Keycloak zur Verfügung gestellt werden. Der Zugriff auf Keycloak geschieht auf Basis des Moduls 'API'. Dieses Modul wird auf dieser Seite ebenfalls verwendet, um Daten von den Servern anzufordern.

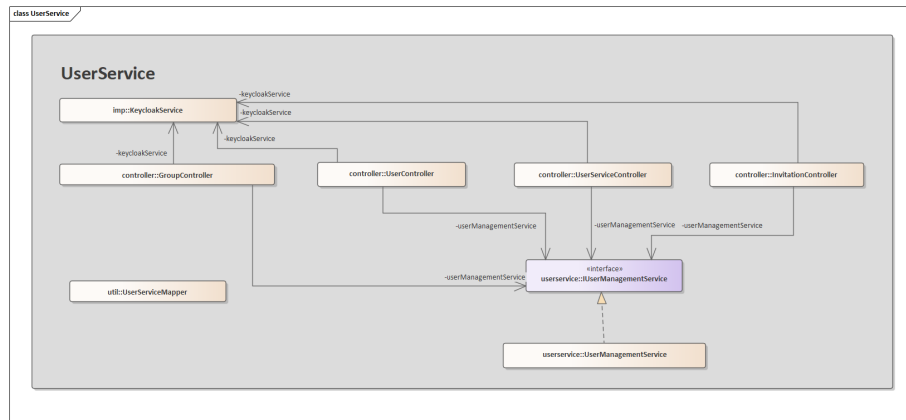
Die 'Homepage' Komponente stellt eine weitere Seite der Anwendung dar. Diese Seite ist mit vielen Funktionen ausgestattet, die durch eine Komposition einer Reihe von Einzelkomponenten umgesetzt ist. Jede Einzelkomponente rendert hierbei einen Ausschnitt der Seite. Funktionen der Einzelkomponenten greifen hierbei auf das Modul 'Client-Store' zu, um neue Werte abzulegen, vorhandene zu modifizieren oder neue Werte anzufordern.

Das zweite Modul 'API' implementiert die Initialisierung und Konfiguration von Keycloak. Diese befindet sich im Diagram in der Komponente 'Auth' und exportiert ein Objekt, welches eine Reihe von Runtime-Funktionen beinhaltet. Des Weiteren beinhaltet das Modul 'API' eine Kollektion von Funktionen, mit denen Requests an die Mircoservices gestellt werden können.

Das letzte Modul 'Client Store' ist verantwortlich für die Speicherung von Daten. Der Speicher wurde mittels Redux implementiert. Insgesamt lässt sich der Speicher in vier Teilbereiche unterteilen. Jeder Teilbereich erhält hierbei einen eigenen Slice, welcher für die Modifikation und Initialisierung dieses Bereiches zuständig ist. Zur Modifikation bietet jeder Slice eine Reihe von Funktionen in Form von Reducern an. Darüber hinaus werden Middleware-Funktionen angeboten, welche mittels des API-Moduls Requests an die Server senden und anschließend auf Basis der Antwort des Requests den Speicher aktualisieren.

5.1.12 Whitebox Userservice

Im folgenden ist eine vereinfachte Darstellung des Services dargestellt. Eine ausführliche Variante des Klassendiagramms kann aus dem Anhang entnommen werden.



Der Userservice bildet mithilfe der drei Controller 'GroupController', 'UserController' und 'InvitationController' eine Schnittstelle um die Benutzer- und Gruppenverwaltung zu ermöglichen.

Alle Endpunkte des Services sind durch Keycloak vor unautorisiertem Zugriff geschützt.

Die Controller bildet hierbei die Endpunkte und geben Anfragen an den IUserManagementServices weiter. Der UserManagementService beinhaltet die Geschäftslogik für folgende Punkte:

- Erstellen von Gruppen und Personen
- Bereitstellen von Userinformationen
- Löschen von Gruppen
- Erstellen, Annehmen und Ablehnen von Einladungen

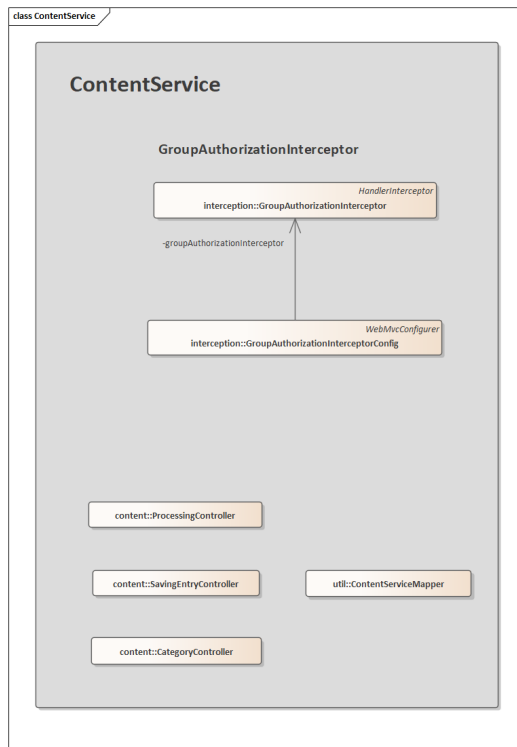
Der UserManagementService greift zur Bearbeitung der oben genannten Punkte auf das 'Relational Databasemodule' zu um Personen-, Gruppen- und Einladungsentitäten zu persistieren.

Bei der Erstellung von Personen wird implizit eine Gruppe erstellt, welche explizit dem Benutzer zugewiesen ist. Im Rahmen der Gruppenerstellung wird unter Verwendung des 'Documentbased Databasemoduls' ein Gruppendokument erstellt und mit einer Reihe von Standardkategorien befüllt.

Zu der oben genannten Funktionalität wird zusätzlich geprüft, ob der Absender der Anfrage die Berechtigungen besitzt, die gewünschte Funktion auszuführen.

5.1.13 Whitebox Contentservice

Im folgenden ist eine vereinfachte Darstellung des Services dargestellt. Eine ausführliche Variante des Klassendiagramms kann aus dem Anhang entnommen werden.



Der Contentservice bietet eine Reihe von Schnittstellen an, mit denen die Daten einer Gruppe (hier auch Personengruppe) manipuliert werden können.

Alle Endpunkte des Services sind zum einen durch Keycloak von unangemeldeten Anfragen geschützt. Zum anderen mittels eines `GroupAuthorizationInterceptor`, welcher sicherstellt, dass der Anfrageabsender die notwendigen Berechtigungen besitzt, die Daten anzufordern.

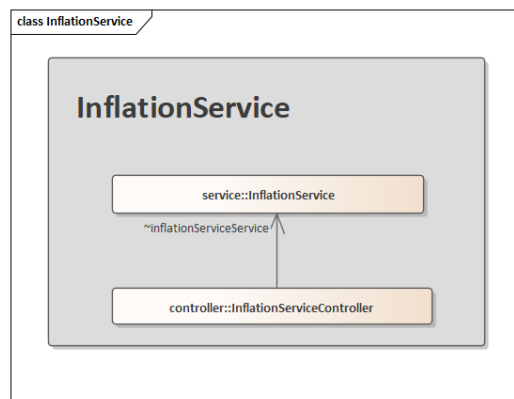
Für die Manipulation werden zwei Rest-Endpunkte angeboten. Der Erste mittels dem `'CategoryController'`, welcher alle CRUD-Operationen für Kategorien anbietet und unter Verwendung des `'Documentbased Databasemodule'` implementiert.

Zum Zweiten einem `'SavingEntryController'`. Dieser implementiert ebenfalls mittels des `'Documentbased Databasemodules'` alle CRUD-Operationen für Spareinträge und gibt die Endpunkte nach außen frei.

Der Service besitzt zudem einen Endpunkt um auf Basis von übergebenen Filterinformationen und weiterer Parameter die Daten einer Gruppe zu filtern und anschließend in spezielle Formate aufzubereiten, welche vom Frontend innerhalb von Diagrammen direkt dargestellt werden können. Dieser Endpunkt wird mittels dem ProcessingController als Http-Endpoint freigegeben.

5.1.14 Whitebox InflationService

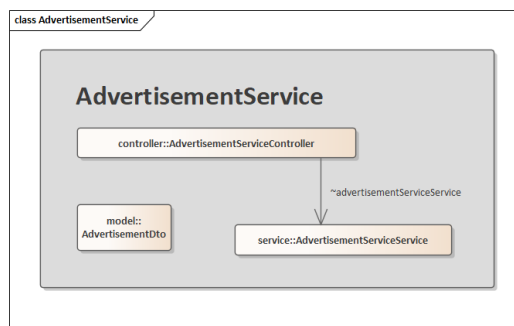
Im folgenden ist eine vereinfachte Darstellung des Services dargestellt. Eine ausführliche Variante des Klassendiagramms kann aus dem Anhang entnommen werden.



Der InflationService besteht aus zwei Komponenten, einem Service, welcher alle 10 Tage neue Daten einer API anfragt und einem Controller, welcher den Zugriff auf die aktuellsten Daten per Get Request von außen ermöglicht.

5.1.15 Whitebox AdvertisementService

Im folgenden ist eine vereinfachte Darstellung des Services dargestellt. Eine ausführliche Variante des Klassendiagramms kann aus dem Anhang entnommen werden.

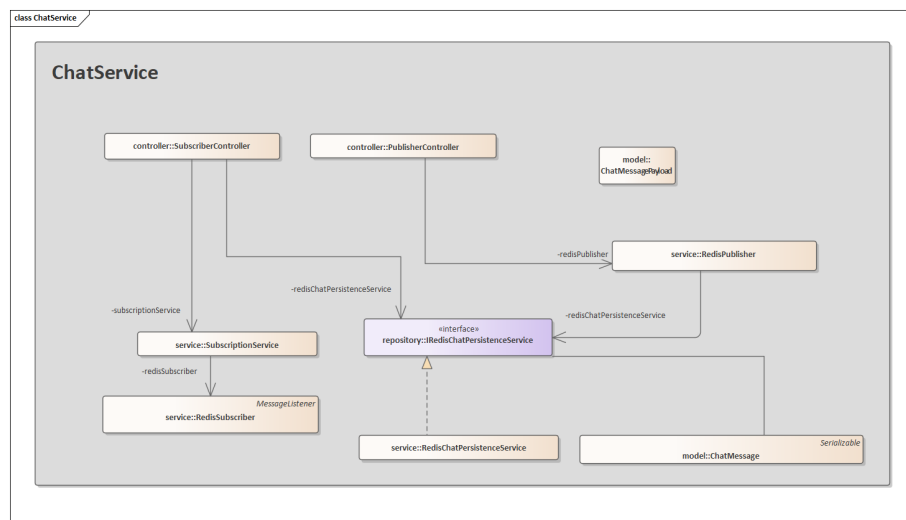


Der Advertisementservice ist für die Bereitstellung von werberelevanten Daten verantwortlich. Die Bereitstellung erfolgt über einen Endpunkt, welcher mittels eines Services, relevante Daten von einer Redis Datenbankinstanz abfragt.

Zudem stellt der Advertisementservice das Frontend, in Form der index.html, mittels zwei weiteren Endpunkte bereit.

5.1.16 Whitebox Chatservice

Im folgenden ist eine vereinfachte Darstellung des Services dargestellt. Eine ausführliche Variante des Klassendiagramms kann aus dem Anhang entnommen werden.



Der Chatservice ist für die Implementierung des Chatsystems zuständig.

Der Subscriber Controller stellt Endpunkte bereit, die es einem User ermöglichen das interne Abonnieren des Redis Sub/Pub-Systems anzustoßen und alte Chatnachrichten einer Gruppe über den RedisChatPersistenceService anzufragen. Die Abonniierung erfolgt mittels des Subscriberservices.

Der Subscriberservice ermöglicht es, neue Listener zu einem Topic für das Redis Sub/Pub-System zu erstellen und auch wieder zu entfernen.

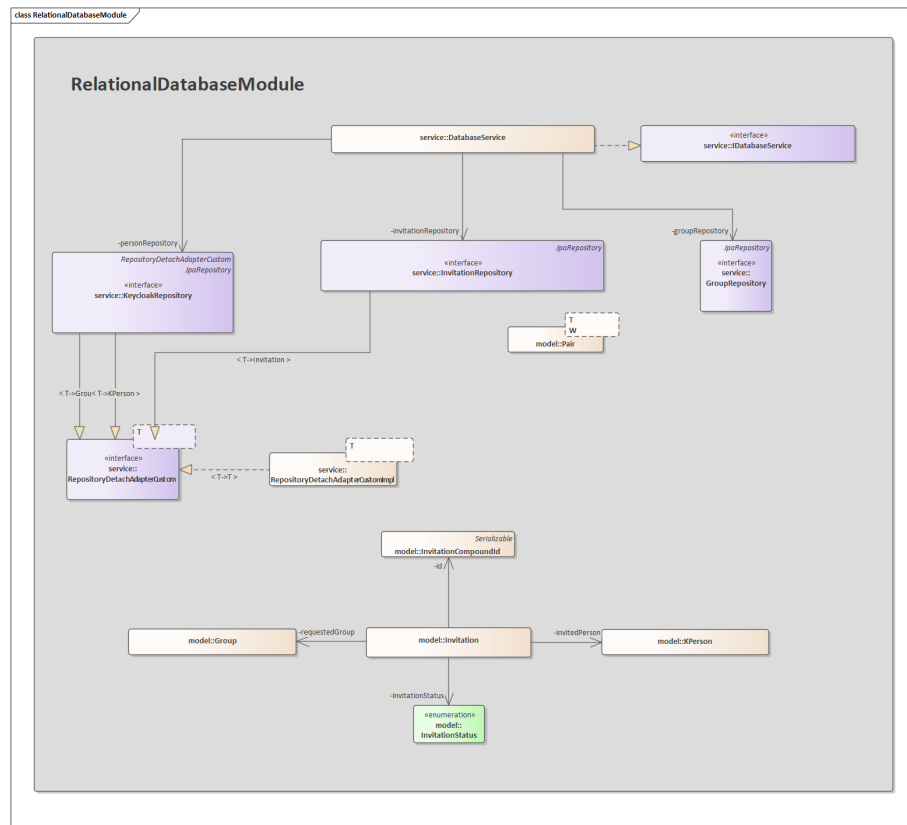
Der hierbei erstellten RedisSubscriber-Listener kontaktieren bei Nachrichtenerhalt über das Redis-SubPub Moduls, alle verbundenen Websockets des Topics.

Der Publisher Controller ermöglicht es, einem Nutzer über eine Schnittstelle neue Nachrichten an eine Gruppe zu senden. Hierfür verwendet dieser den Publisherservice.

Der Publisherservice bietet die Funktionalität, eine Nachricht an das Topic eines Redis Sub/Pub-Systems zu versenden und über die Implementierung des RedisPersistenceService diese Nachricht zu persistieren.

5.1.17 Whitebox Relational Databasemodule

Im folgenden ist eine vereinfachte Darstellung des Moduls dargestellt. Eine ausführliche Variante des Klassendiagramms kann aus dem Anhang entnommen werden.

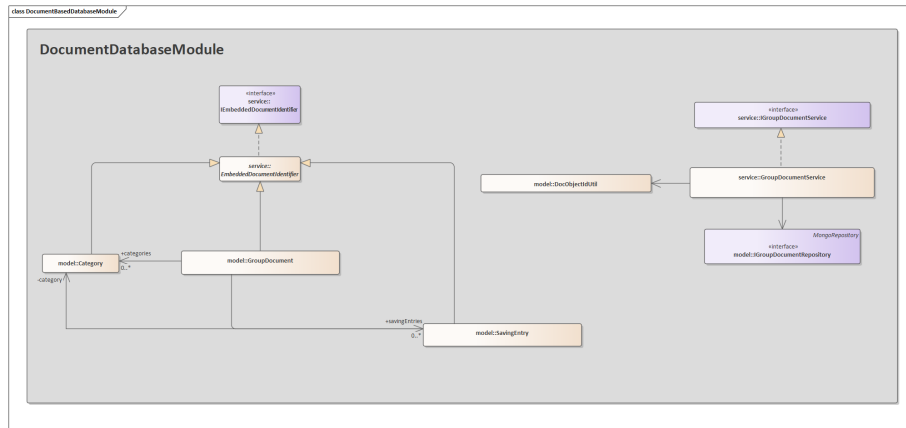


Das Relational Databasemodule dient als High-Level Persistenzlayer zur relationalen Datenbank. Hierfür bietet dieses die Schnittstelle `IDatabaseService` an, um die Entitäten abzufragen, zu modifizieren und zu löschen.

Die Implementierung der IDatabaseService verwendet hierfür sowohl JPA-Repositories, als auch den ORM-Mapper Hibernate. Diese stellt sicher, dass die Datenbankbestände, insbesondere aller M:N Beziehungen, von einem konsistenten Zustand in einen Weiteren übergehen.

5.1.18 Whitebox Documentbased Databasemodule

Im folgenden ist eine vereinfachte Darstellung des Moduls dargestellt. Eine ausführliche Variante des Klassendiagramms kann aus dem Anhang entnommen werden.

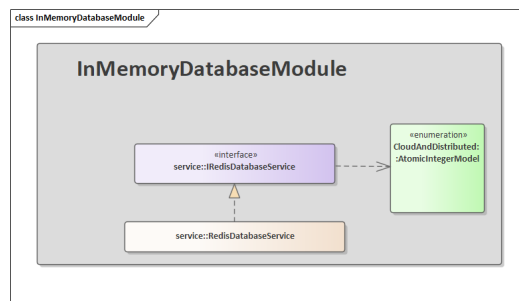


Auf die dokumentenbasierte Datenbank wird exklusiv mittels des document-based Databasemodules zugegriffen. Diese bildet ähnliche wie das Relational Databasemodule ein High-Level Persistenzlayer. Über das Modul wird mittels der Schnittstelle IGroupDocumentService zugegriffen.

Die Implementierung erfolgt mittels Zugriff auf ein JPA-Repository.

5.1.19 Whitebox In-Memory Databasemodule

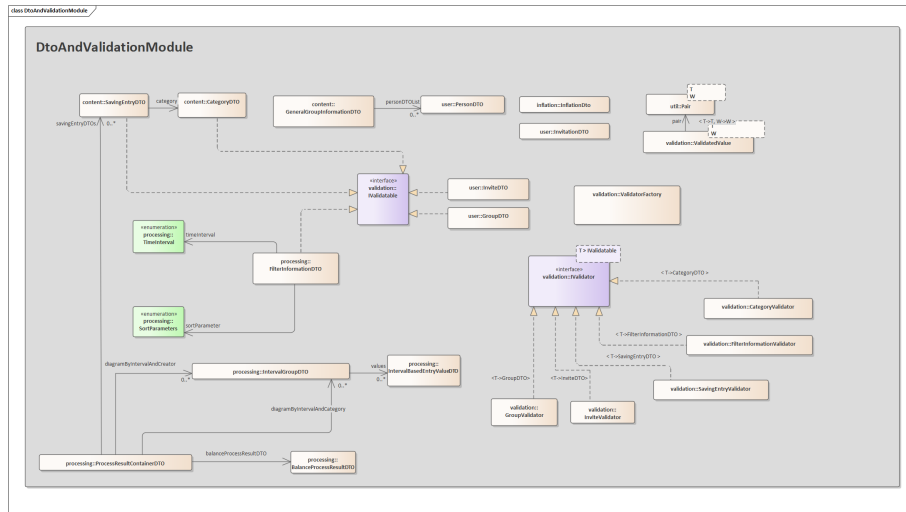
Im folgenden ist eine vereinfachte Darstellung des Moduls dargestellt. Eine ausführliche Variante des Klassendiagramms kann aus dem Anhang entnommen werden.



Das In-Memory Databasemodul ermöglicht dem importierenden Service eine Verbindung zu einer Redis Datenbankinstanz. Desweiteren bietet das Modul die Möglichkeit einen von drei festgelegten Atomic Integern innerhalb der Datenbank zu inkrementieren und aus dieser zu holen.

5.1.20 Whitebox Data Transfer Objects and Validation

Im folgenden ist eine vereinfachte Darstellung des Moduls dargestellt. Eine ausführliche Variante des Klassendiagramms kann aus dem Anhang entnommen werden.



Im Modul Data Transfer Objects (DTO) and Validation, befinden sich DTOs und Validierungseinheiten, welche von verschiedenen Services gemeinsam verwendet werden. Services importieren gemeinsam genutzte DTOs und Validierungseinheiten aus diesem Modul.

Die Validierungseinheiten besitzen Logik um DTOs, welche bei Anfragen übermittelt werden, auf Korrektheit zu prüfen. Die Validatoren können mittels der Validatorfactory für ein zu prüfendes DTO angefordert werden und mittels einer Validate-Methode das DTO prüfen.

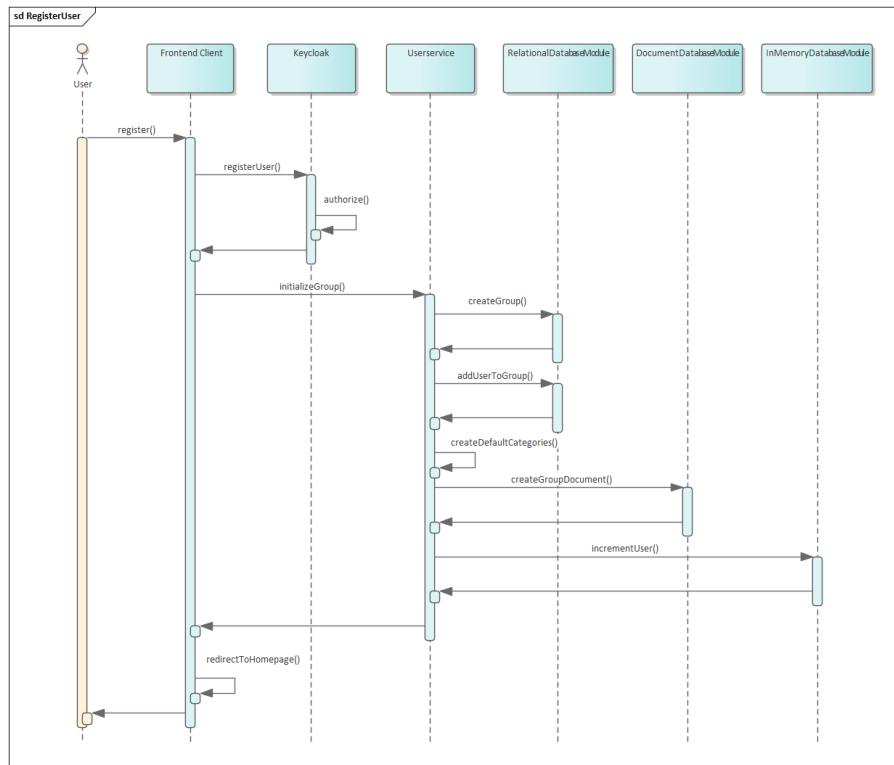
6 Laufzeitsicht

Dokumentation: Michael Ulrich, Simon Ruttmann

Umsetzung: Michael Ulrich, Simon Ruttmann, Veronika Scheller

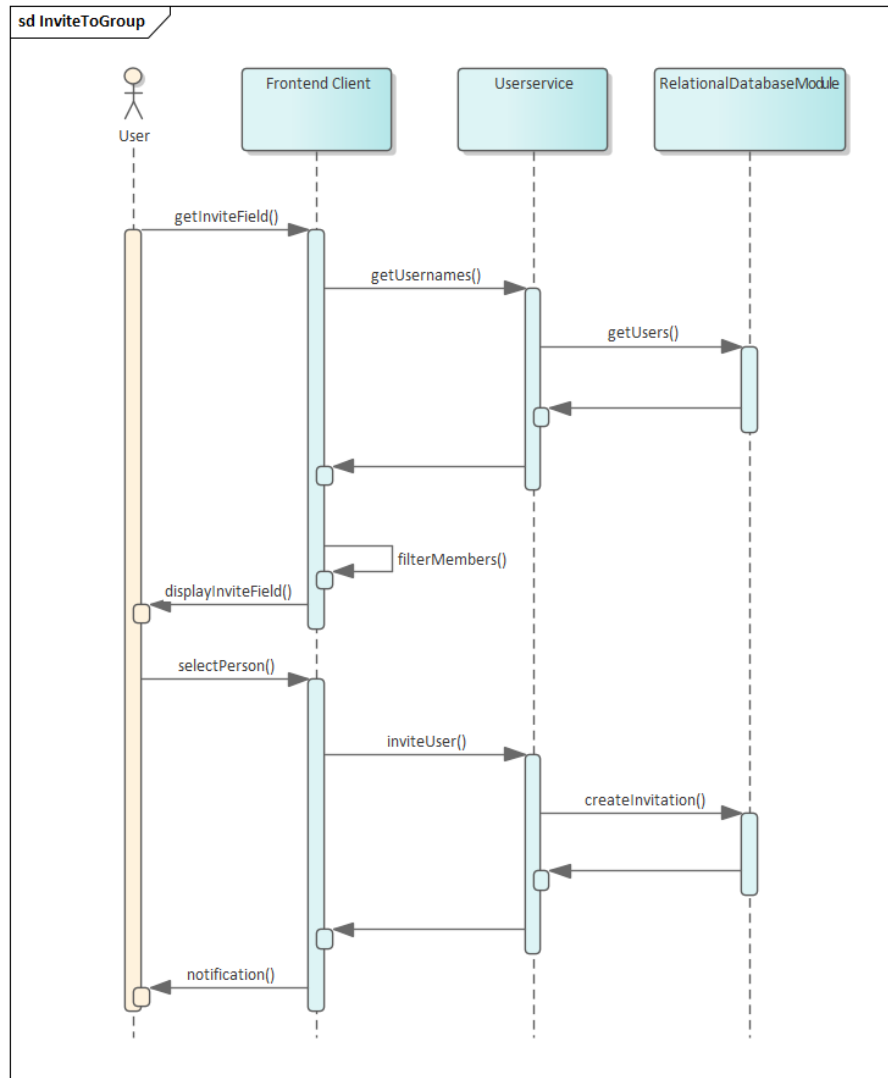
6.1 User Service

Ablauf: Nutzer registration



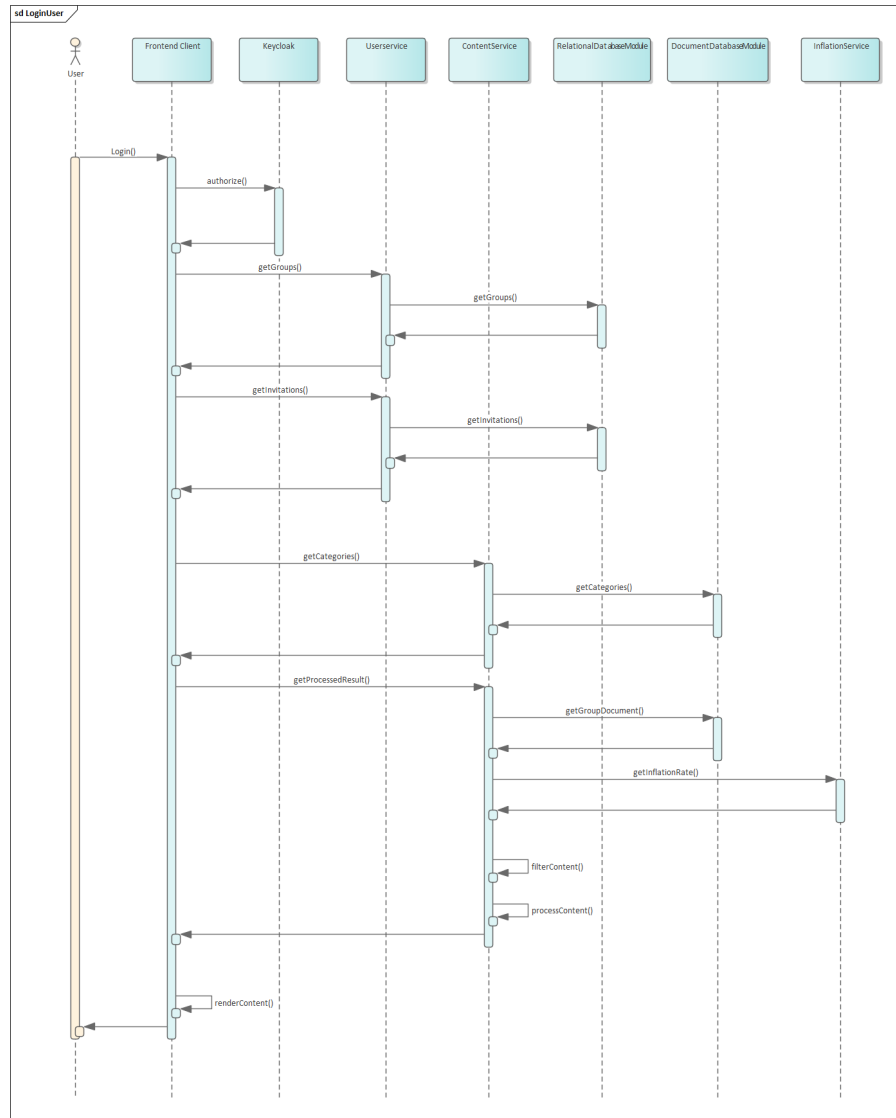
Nach einem Authorisierungsprozess unter Keycloak, wird ein neuer Nutzer nach einer Tokenzuweisung und Gruppenerstellung vom System auf seine Homepage weitergeleitet. Da ein User als Gruppe mit nur einer Person behandelt wird, muss beim Registrieren immer eine so genannte Ich-Gruppe mit erstellt werden.

Ablauf: Gruppeneinladung



Um einen Nutzer in eine Gruppe einzuladen muss zunächst eine Einladung erstellt und versendet werden. Es ist hierbei wichtig das es sich bei dem einzuladenden Nutzer um einen registrierten User handelt. Der eingeladene Nutzer erhält eine Mitteilung in seinem Einstellungsbereich und hat die Optionen der Gruppe beizutreten oder die Einladung abzulehnen.

Ablauf: Einloggen

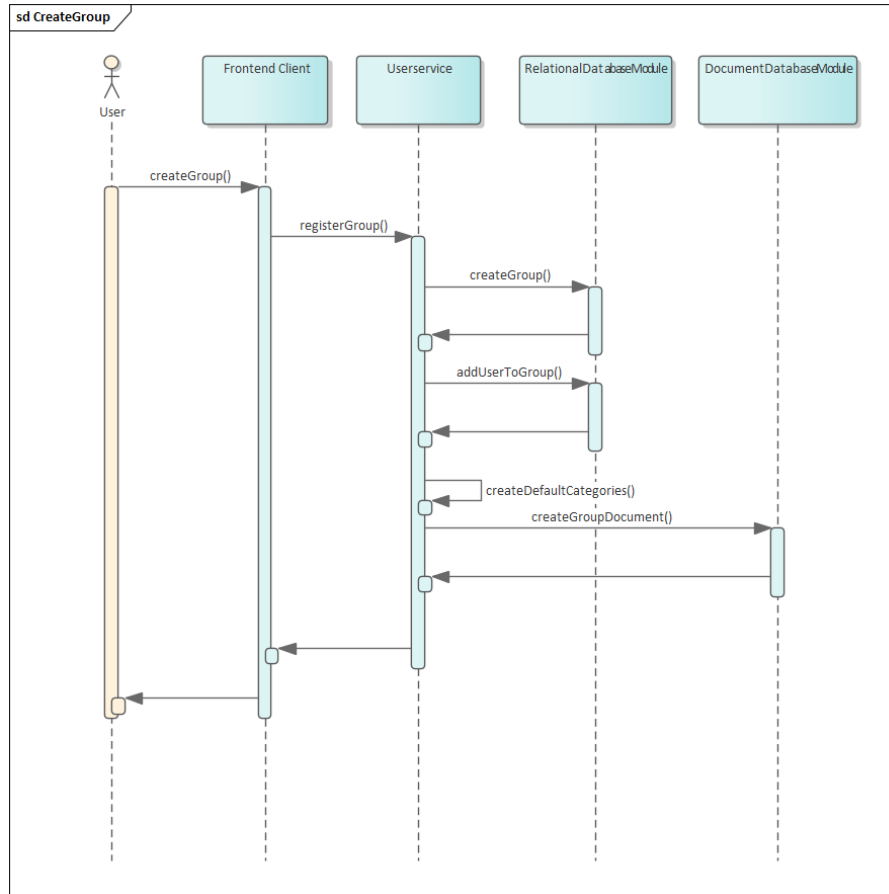


Beim Anmeldevorgang muss nach einer erfolgreichen Authentifizierung eine Reihe von Daten geladen werden. Das Frontend fordert hierbei parallel vom User- und ContentService alle erforderlichen Daten an.

Der Userservice liefert hierbei alle Gruppen, sowie alle offenen Einladungen des Benutzers.

Über den ContentService erhält das Frontend alle Kategorien, sowie ein ProcessResult-Objekt, welches alle Spareinträge und Diagramminformationen beinhaltet. Ein separates Laden der Spareinträge ist daher nicht erforderlich.

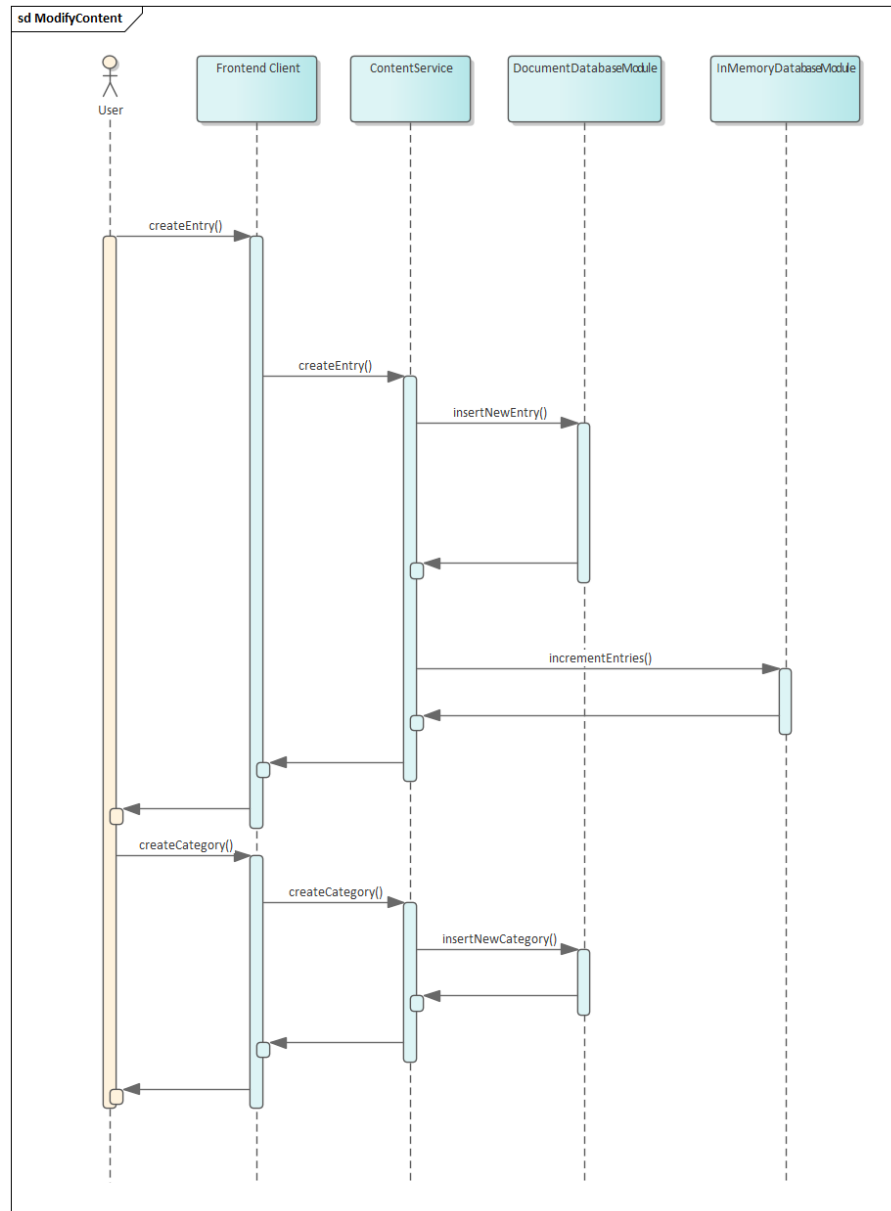
Ablauf: Gruppe erstellen



Um eine Gruppe zu erstellen muss sichergestellt werden das alle Aktionen in der richtigen Reihenfolge ablaufen, da Abhängigkeiten vorliegen. Es muss zuerst ein neuer Eintrag in der Relationalen Datenbank angelegt werden, dann der Creator zur Gruppe hinzugefügt werden, gefolgt von dem Erstellen eines Gruppendokuments mit Standardkategorien in der Mongo Datenbank.

6.2 Content Service

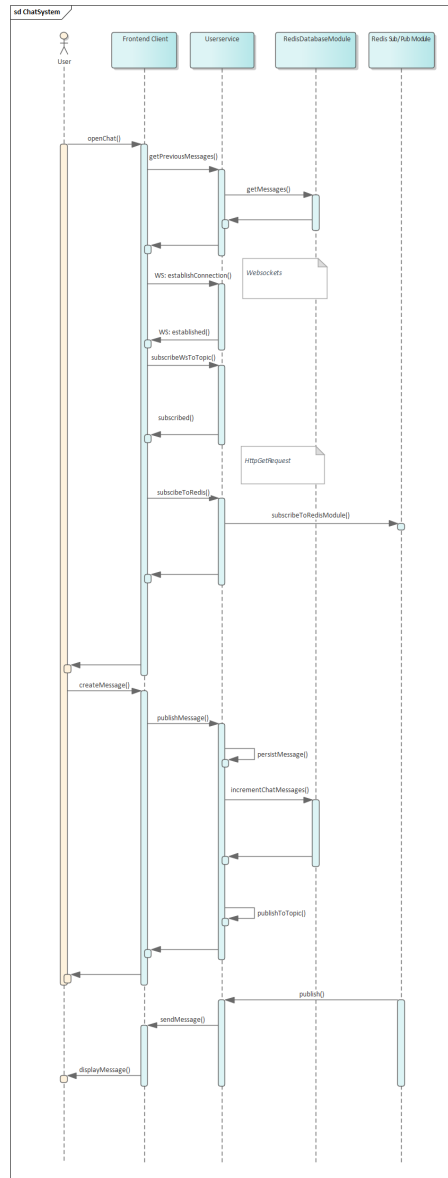
Ablauf: Inhalt editieren



Beim Erstellen von benutzergenerierten Inhalten gibt es die Möglichkeit, sowohl Spareinträge, als auch Kategorien zu erstellen. Beim Erstellen eines Spareintrags muss sichergestellt werden, dass der dazugehörige Zähler im Redis Datenbankmodul hochgezählt wird.

6.3 Chat Service

Ablauf: Nachricht schreiben & empfangen



Der ChatService ist komplexer, da das Redis Sub/Pub-Modul als Messagebroker für den Chatnachrichtenversand eingebunden wurde. Das resultierende System enthält zwei parallel laufende Websockets, einer für die Kommunikation zwischen Front-Backend und ein weiterer für die Redis-Backend Kommunikation.

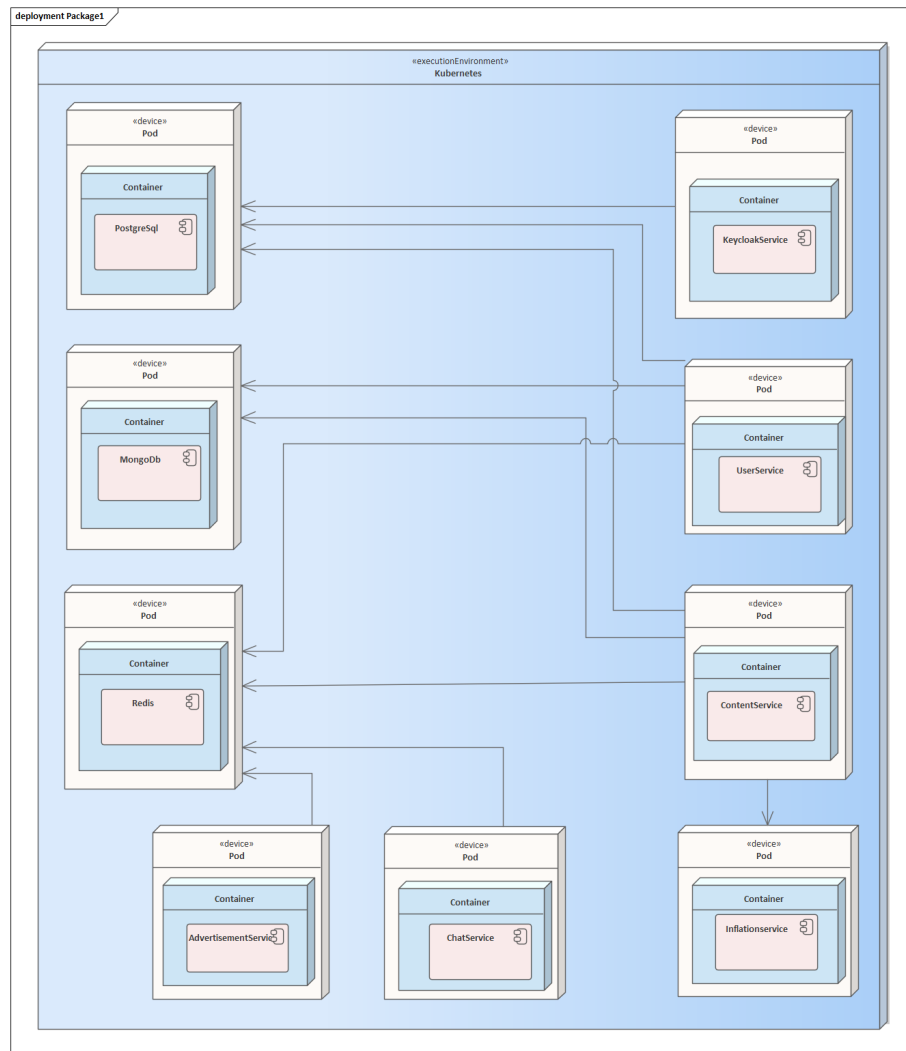
7 Verteilungssicht

Dokumentation: Simon Ruttmann

Umsetzung: None

In dieser Sektion wird die technische Infrastruktur beschreiben, auf der die zu entwickelnde Software betrieben wird.

Im folgenden zeigt das Deployment-Diagramm die Infrastruktur des Software.



Alle Microservices und Datenbanken laufen, wie in den Randbedingungen beschrieben, in einem separaten Container.

Die Inbetriebnahme erfolgt durch die Verteilung der einzelnen Container mittels einer Deployment-Datei auf ein Kubernetes Cluster.

Im Cluster befinden sich auf jedem Node ein oder mehrere Pods. Allen Containern wird hierbei ein Pod zugewiesen, in dem diese betrieben werden. Es wurde sich bewusst dagegen entschieden alle Container in einem Pod zu betreiben. In diesem Fall, wäre die Skalierung der Anwendung nicht möglich, da in diesem Fall eine Skalierung eine Duplikation aller Services verursachen würde. Dies würde insbesondere die Datenbanken und den reverse Proxy betreffen.

Die Kommunikation der Microservices und Datenbanken erfolgt über Kubernetes-Services.

8 Querschnittliche Konzepte

Dokumentation: Veronika Scheller, Simon Ruttmann

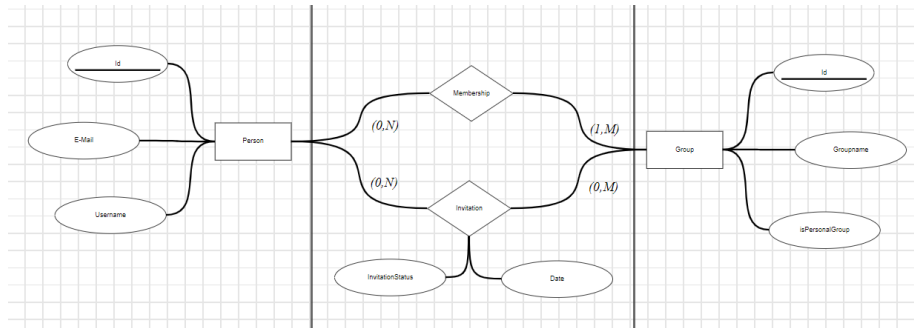
Umsetzung: Veronika Scheller, Simon Ruttmann, Michael Ulrich

8.1 Persistence

Fortfolgend werden die Schemas der einzelnen Datenbanksysteme beschrieben.

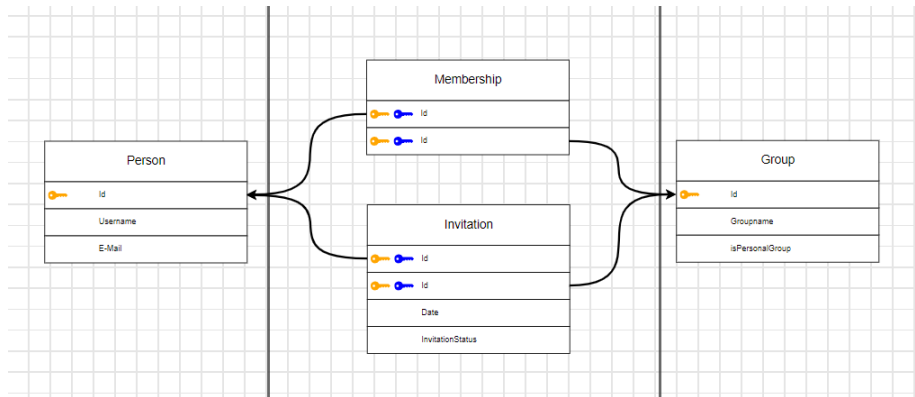
8.1.1 Entity-Relationship-Diagramm

Das fortfolgende Diagramm zeigt die Beziehung zwischen den Entitäten Person und Gruppe.



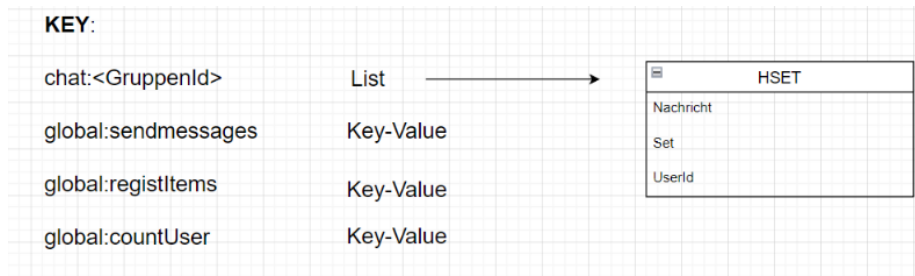
8.1.2 Relationales Diagramm

Aus dem vorhergehenden ER-Diagramm ergibt sich folgendes relationales Modell, über welches im System mittels dem OR-Mapper Hibernate zugegriffen wird. Hierbei ist anzumerken, dass Keycloak ebenfalls die Person-Relation verwendet. Keycloak benötigt hierbei eine Reihe weiterer Attribute. Diese wurden zugunsten der Übersichtlichkeit nicht dargestellt.



8.1.3 Redis Datenmodell

Das Persistenzmodell der Redis Datenbank enthält drei einfache Key-value Paare zur Speicherung von inkrementierbaren Zählern. Darüber hinaus beinhaltet es zur Persistierung von Chatnachrichten pro Gruppe eine Liste, in der jeder Eintrag eine Hashtabelle referenziert. Die Hashtabelle beinhaltet Informationen über eine Chatnachricht.



8.1.4 MongoDB Datenmodell

Die Dokumentendatenbank beinhaltet alle Daten einer Gruppe. Diese Gruppen bestehen aus einer Kollektion von Kategorien und Spareinträgen, welche vom Benutzer erstellt, gelöscht und editiert werden können.

Hierbei ist zu beachten, dass jeder Benutzer eine privaten Gruppe erhält, in dem seine persönlichen Spareinträge und Kategorien abgelegt werden.

Über die private Gruppen hinweg, existieren Gruppen aus mehreren Personen, welche von Benutzern erstellt werden.


```

Mongo
{GruppenId,
 Spareinträge { [
   SpareintragsId,
   Name,
   Kategorie,
   Kosten,
   Datum,
   Username,
   Beschreibung
 ]},
 Kategorien {[
   Lebensmittel,
   Miete
 ]}
}

```

8.2 REST-API's

Dokumentation: Veronika Scheller

Umsetzung: Veronika Scheller, Michael Ulrich, Simon Ruttmann

Im folgenden werden die REST-API's des Systems beschrieben. Im Projekt ist innerhalb des `api` Ordners eine visuelle Ansicht aller mittels Swagger generierten Rest-Schnittstellen verfügbar.

8.2.1 Userservice

User Controller:

Dieser Kontroller liefert Informationen über den die angemeldeten Benutzer des Systems. Diese kann für eine einzelnen Person durch die Angabe ihrer Id angefragt werden. Das `PersonDto` liefert die Id, den Usernamen und die Email zurück. Zudem kann ein Array angefragt werden, welches alle Usernamen der registrierten Personen beinhaltet.

Group Controller:

Der Kontroller liefert die Möglichkeit eine Gruppe zu registrieren, alle eigenen Gruppen zu bekommen, aus Gruppen auszutreten und Gruppen zu löschen. Erstellt man eine Gruppe oder werden alle Eigenen angefragt, wird ein GroupDto bzw. ein Array von GroupDto's als Antwort geliefert.

Das GroupDto beinhaltet die Id, Gruppennamen und den Boolean 'Person-group'. Verlässt man eine Gruppe oder löscht diese wird die Bestätigung in Form eines 200 HTTP-Statuses gesendet. Die dafür zu erfüllenden Bedingungen sind, dass die Gruppe mit dieser Id existiert, der angemeldete User Mitglied dieser ist und es sich nicht um eine persönliche Gruppe handelt.

Invitation Controller:

Dieser Kontroller ermöglicht es eine Person in eine Gruppe einzuladen, die eigenen offenen Einladungen einzusehen, eine Einladung anzunehmen oder abzulehnen. Als Antwort erhält man je nach Request ein oder mehrere InvitationDtos. Sie beinhalten den Gruppennamen sowie ihre Id, das Erstellungsdatum der Einladung und den Status der Einladung.

8.2.2 Contentservice**Category Controller:**

Dieser Kontroller ermöglicht es neue Kategorien für eine Gruppe zu erstellen, den Namen einer Kategorie zu ändern, alle Kategorien einer Gruppe zu erhalten und die Kategorie zu löschen.

Die zurückgegebenen CategoryDtos bestehen dabei aus generierten Ids und dem Namen der Kategorie.

Saving Entry Controller:

Der Kontroller ermöglicht es einen neuen Eintrag zu erstellen, diesen zu bearbeiten, einen bestimmten sowie alle Einträge einer Gruppe zu bekommen und einen Eintrag zu löschen.

Die zurückgegebenen SavingEntryDtos enthalten dabei jeweils die Id, den Eintragsnamen, die Kosten, die Kategorie in Form eines CategoryDtos, das Erstelldatum, den Namen des Erstellers und eine Beschreibung, die optional vom Ersteller auszufüllen ist.

Processing Controller:

Mittels diesen Kontrollers können Informationen einer Gruppe angefragt werden. Zurückgegeben wird ein `GeneralGroupInformationDto`, welches den Gruppennamen, sowie eine Liste der Mitglieder beinhaltet.

Zudem können für eine Gruppe aufbereitete Daten angefordert werden. Hierfür werden der Schnittstelle Filterinformationen übergeben. Die Schnittstelle übergibt daraufhin ein so genanntes `ProcessResultContainerDto`. Dieses beinhaltet eine Liste von gefilterten und sortierten Einträgen. Zusätzlich beinhaltet es Objekte, in denen die Summe der gefilterten Spareinträge gruppiert nach Zeitintervall, Kategorie und Ersteller angegeben ist.

8.2.3 Inflationsservice

Inflation Controller:

Der Inflationsskontroller fragt eine externe API an, die ihm die aktuelle Inflationsrate sowie den letzten Updatezeitpunkt dieser Rate übermittelt.

8.2.4 Advertisementservice

Advertisement Controller:

Dieser Kontroller liefert in einem `AdvertisementDto` drei Werte zurück, welche die Anzahl der registrierten User, der gespeicherten Einträge sowie der geschickten Chatnachrichten repräsentieren.

Zusätzlich können mittels dieses Kontrollers Frontend-Ressourcen angefragt werden.

8.2.5 Chatsservice

Subscriber Controller:

Mittels diesen Kontrollers wird eine Subscription auf Chatnachrichten einer Gruppe ermöglicht. Hierbei werden alle bisher versendeten Chatnachrichten für die Gruppe übermittelt.

Publish Controller:

Mit diesem Kontroller kann man unter Angabe der GruppenId als Topic und des eingetragenen Usernamens eine Nachricht an den Chat senden.

8.3 User Interface

Das User Interface besteht aus einer React-Anwendung. Die Ressourcen der Anwendung werden über den Advertisementservice angefordert.

Das User Interface besteht aus mehreren Ansichten. Darunter zählen die Gastseite, die Homepage, die Loginseite und die Registrierungsseite. Die Login- und Registrierungsseite wird von Keycloak gestellt.

Im Wesentlichen handelt es sich bei der UI dennoch um eine Single-Page Anwendung, da ein Großteil der Interaktionsmöglichkeiten und Anwendungsfälle innerhalb der Homepage stattfinden. Auf dieser Seite besitzt der Benutzer die Möglichkeit mittels Eingabefelder, Selektoren, Datumsauswähler und Buttons mit der Anwendung zu interagieren.

Alle Ansichten der Anwendung sind bis zu einer minimalen Größe von 320px Responsive und somit für jedes gängige Mobilgerät optimiert.

Die Konzeption des User Interfaces erfolgte auf Basis des Wireframing-Tools Figma. Auf Basis des darin entstandenen Prototypen wurde die UI entwickelt. Während der Entwicklung sind Abweichungen der React-Anwendung zum Prototypen, zugunsten einer besseren User-Experience entsanden. Die Abweichungen sind hierbei überwiegend auf struktureller Ebene erfolgt.

Die Sichten des verwendeten Wireframing-Tools können aus dem Anhang entnommen werden.

8.4 Communication/Integration

8.4.1 Inflation API

Die InflationAPI wird von statbureau.org bereitgestellt und bietet eine Schnittstelle an, die mit Requestparametern auf das Land, in diesem Fall Deutschland, eingestellt werden kann. Ein Scheduler holt sich in regelmäßigen Abständen, gerichtet nach den vorgegebenen Updatehäufigkeit der API, die neuen aktualisierten Daten.

8.4.2 Keycloak

Keycloak ist eine Open Source Software, die sich um die Authentifizierung und das Usermanagement kümmert. So kann der Zugang zu der Applikation gesichert und Überwacht werden.

8.4.3 Spring Boot

Spring Boot ist ein Open-Source-Framework, das die Komplexität der Java-Programmierung reduziert, indem Grundkonfigurationen vorhanden und damit ein schneller Start in die Programmierprozess möglich ist.

8.4.4 Hibernate

Hibernate ist ein Open-Source-Persistenz- und ORM-Framework. Unter Spring Boot bietet es sich besonders an dieses als Objekt/Relational Mapper zu benutzen, da die Integration in wenigen Schritten erledigt ist und die beiden Komponenten höchst kompartibel sind.

9 Risiken und technische Schulden

Dokumentation: Veronika Scheller, Michael Ulrich, Simon Ruttman

Im folgenden werden auf Mängel und Risiken der Anwendung eingegangen.

- Ein Deployment auf Azure ist, wie abgesprochen, nicht durchgeführt worden
- Wird die Gruppe eines Users mit offener Session gelöscht, so wird diese dem User noch angezeigt. Der User wird bei Auswahl auf die Personengruppe weitergeleitet. Ein effizientes entfernen der Gruppe aus allen momentan aktiven Mitgliedern der Gruppe müsste mittels Serverpush realisiert werden.
- Im Frontend wären noch Quality of Live und Design improvements möglich

10 Architekturentscheidungen

Dokumentation: Simon Ruttmann

In dieser Sektion werden die wichtigsten Architekturentscheidungen beschrieben.

Einführung von Modulen

Grundsätzlich wurde sich entschieden im Backend Module einzuführen. Durch die Einführung ergeben sich folgende Vorteile:

- Logikduplikation kann effektiv vermieden werden
- Für die Datenbanken kann ein Persistenz-Layer eingeführt werden
- DAO'S können gemeinsam genutzt werden

Verwendung des Redids Sub-Pub Modules

Es wurde sich entschieden den Chat-service mittels des Redis Sub-Pub Moduls zu realisieren. Dies liegt darin begründet, dass dadurch nicht ein Service alle aktiven Verbindungen zu allen offenen Sessions halten muss. Eine Skalierung des Services wäre ohne Verwendung eines Brokers nicht möglich.

Verwendung der Redis Datenbank zur Speicherung von Werbedaten

Innerhalb der Anwendung werden die Anzahl der versendeten Nachrichten, die Anzahl der registrierten Nutzer und die Anzahl der erstellten Einträge persistiert. Für die Persistierung wurde entschieden, die Werte innerhalb einer Redis Datenbank abzulegen. Dies liegt darin begründet, dass die Redis Datenbank aufgrund der Verwendung durch das Redis Sub-Pub Modul bereits benötigt wird. Des Weiteren ist dieses Datenbanksystem im Vergleich zur verwendeten MongoDB und PostgreSQL schnell. Eine Ermittlung der Werte zur Laufzeit wurde ausgeschlossen, da dies zu einem vollständigen Scan bei der Ermittlung der Anzahl erstellter Spareinträge innerhalb der MongoDB führen würde. Zudem würde in diesem Fall die Löschung eines Eintrags das Endergebnis beeinflussen.

Verwendung des Redux-Stores im Frontend

Im Frontend wurde die Datenhaltung mittels einer Redux-Store implementiert. Dies ermöglicht es die Anzahl an Anfragen an die Services zu verringern. Dies führt zu einer Entlastung der Services, sowie einer besseren Reaktionszeit der Clientanwendung.

11 Anhang

11.1 Klassendiagramm des Backends

11.2 Wireframe

Register

Bild 1
Anzahl versendeter Nachrichten

Register

E-Mail

Username

Password

Register

Abbrechen

Bild 3
Anzahl der User

Login

Bild 1
Anzahl versendeter Nachrichten

Login

E-Mail

Password

LOGIN

Abbrechen

Bild 3
Anzahl der User

Startseite

Register

Login

Hier steht ein wundervoller Slogan für wundervolle Menschen

Bild 1

Anzahl versendeter Nachrichten

Bild 2

Anzahl der Einträge

Bild 3

Anzahl der User

Hier beschreiben wir ein schöne App für schöne Leute

Startseite - Angemeldet

Start

Ansicht ▼

Ich

⚙️

Logout

Diag1

Diag2

Diag3

all | WG | FAM | ME

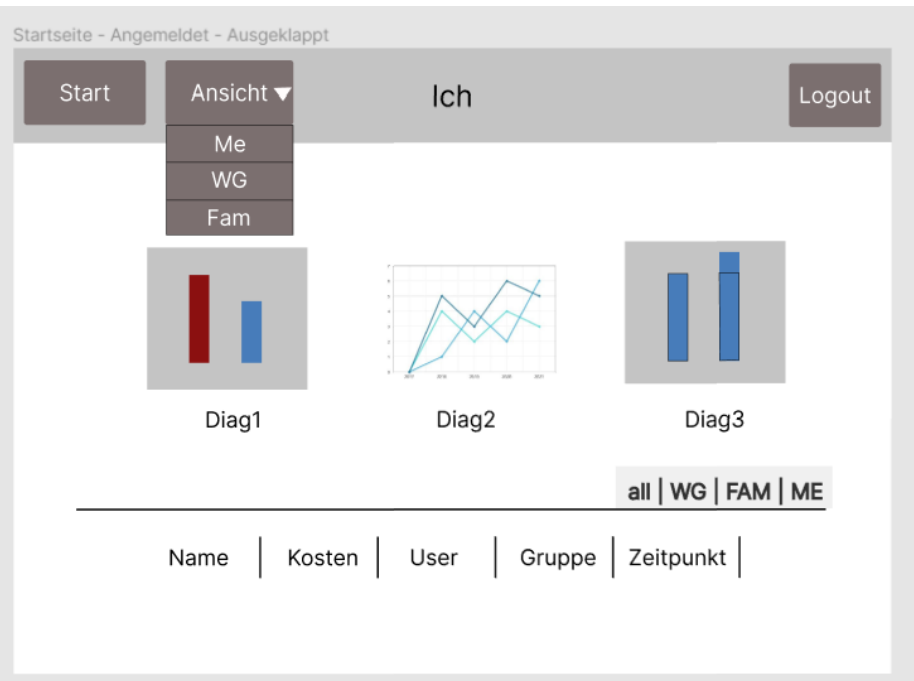
Name

Kosten

User

Gruppe

Zeitpunkt



Eintragsmenü

Start

Ansicht ▼

WG

Logout

Name

Kosten

Kategorie ▼

neuer Eintrag

genauer

Lernen

Food

Hund

Miete

+

Zeitraum <Woche> | < > | Datum | User: | Kategorie:

CHAT

Diag1

Diag2

Diag3

Name	Kosten	User	Datum & Zeitpunkt	Kategorie	Beschreibung
Burger King	20,00€	Heinrich	23.03.2022 13:00Uhr	#Food	Monatsende
Hund	1200,00€	Sofie	20.03.2022 03:00Uhr	#Hund	

Erweitertes Eintragsmenü

Start

Ansicht ▼

WG

Logout

Name

Kosten

Kategorie ▼

neuer Eintrag

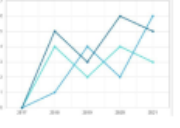
Datum

Beschreibung

weniger



Diag1



Diag2



Diag3

CHAT

Chat offen

Start

Ansicht ▼

WG

Logout

Name

Kosten

Kategorie ▼

neuer Eintrag

genauer



Diag1



Diag2

X

51