CONNECT MESSENGER

GFOS Innovations-Award 2021

Exposee

Dies ist die Dokumentation zu dem während des Wettbewerbs programmierten Messenger. Sie erläutert die Projektdurchführung und die allgemeine Umsetzung. Des Weiteren sind eine Installationsanleitung sowie ein Benutzerhandbuch angefügt.

Florian Noje, Lukas Krinke & Friedrich Böttger
mit Lehrer Michael Albrecht

Inhaltsverzeichnis

1. Projekt

1.1 Vorbereitung

1.1.1 Aufgabenstellung

Das Ziel des Projektes sollte es sein, einen browserbasierten Messenger zu konzipieren und zu programmieren. Das Backend sollte dabei in Java geschrieben werden; für das Frontend durfte ein beliebiges JavaScript Framework verwendet werden. Der Messenger sollte übliche Funktionalitäten, wie zum Beispiel das Versenden von Nachrichten oder die Erstellung von Gruppen, bieten, durfte aber durch eigene Ideen erweitert werden.

1.1.2 Erste Überlegungen

Zunächst haben wir Pläne für die einzelnen Komponenten angefertigt. Diese umfassen folgenden Punkte:

- Frontend
  - Skizzen der wichtigsten Masken
  - Grobe Designentscheidungen, wie Farben, etc.
  - Navigation: Wie kann der User intuitiv zwischen den Seiten navigieren?
- Backend
  - Liste aller Funktionen, die essentiell sind
  - Struktureller Aufbau
  - Welche Technologien benötigen wir des Weiteren?
- Datenbank
  - Entwurf aller Tabellen und zugehöriger Attribute

Für alle wichtigen Angelegenheiten wurde ein Notizbuch erstellt, in das jeder von uns jederzeit wichtige Ideen oder existierende Fehler hineinschreiben konnte. So konnte eine reibungslose Kommunikation sichergestellt werden und keine Ideen, beziehungsweise Fehler in der Software, sind in Vergessenheit geraten.

1.1.3 Pläne für weitere Features

Bevor wir mit der eigentlichen Implementierung begonnen haben, überlegten wir uns, welche Funktionen wir zum Ende der Projektzeit noch zusätzlich einbauen wollen. Dafür haben wir eine weitere nach Priorität geordnete Liste angelegt, sodass wir sie dann möglichst schnell eins nach dem anderen abarbeiten konnten. Nennenswerte Beispiele sind: Diverse Einstellungen, um das Konto für einen Nutzer zu optimieren, 2-Faktor-Authentifizierung, das Antworten auf Nachrichten und vieles mehr.

1.1.4 Aufgabenverteilung

Nachdem alle wichtigen Vorbereitungen gemeinsam getroffen wurden, haben wir uns an die eigentliche Implementierung des Projektes gesetzt. Dafür haben wir uns nach unseren Stärken aufgeteilt. Lukas und Friedrich haben sich mit der Umsetzung des Backends auseinandergesetzt, da sie aufgrund des Informatik-Leistungskurses und eigener Projekte in diesem Bereich bereits einiges an Erfahrung sammeln konnten. Florian hat sich um die Implementierung des Frontendes gekümmert, da er schon mehrere Anwendungen mit Angular programmiert hat und sich auf diesem Gebiet gut auskennt. Ziel war es, Mitte April mit der Implementierung fertig zu sein, damit anschließend gemeinsam die Dokumentation verfasst werden konnte.

1.2 Umsetzung

1.2.1 Verwendete Technologien

Für das Frontend haben wir Angular (Version 11.0.2) als Framework mit Angular Material (Version 11.0.4) für das Design genutzt. So konnte einfach zwischen Komponenten und Logik kommuniziert werden. Alle benötigten Module mit zugehöriger Versionsnummer sind in der *package-lock.json* aufgelistet.

Das Backend wurde entsprechend der Aufgabenstellung in JavaEE 8 geschrieben. Als Datenbank haben wir die in NetBeans integrierte JavaDB (Version 10.14.2.0) genutzt. In dem Docker-Image wird jedoch eine MySQL-Datenbank verwendet. Bereitgestellt wird es mit dem GlassFish-Application Server (Version 5.1.0). Zur Realisierung einiger Features benötigten wir folgende Bibliotheken:

Dependencies	Runtime-Dependencies	Java-Dependencies
commons-codec-1.11.jar gson_2.8.1.jar java-jwt-3.14.0.jar activation-1.1.jar javax.mail-1.6.0.jar	jackson-annotations-2.11.0.jar jackson-core-2.11.0.jar jackson-databind-2.11.0.jar	JDK 1.8 javaee-api-8.0.jar

Tabelle 1: Backend Dependencies

1.2.2 Durchführung

Das Frontend wurde zunächst mit statischen Beispiel-Datensätzen implementiert, um ohne ein funktionierendes Backend das Design zu entwickeln. Zudem konnten so Backend-Fehler ausgeschlossen werden und sich zunächst ausschließlich auf das Design konzentriert werden.

Währenddessen wurde zuerst die Datenbank erstellt. Anschließend wurden die einzelnen Routen geschrieben, die vorher beschlossen wurden. Während der Umsetzung wurden die Datenbanktabellen immer wieder um einzelne Attribute erweitert. Bei der Implementierung der einzelnen Routen und Methoden wurde nach dem „Trial and Error-Prinzip“ vorgegangen. Getestet wurden die Routen zunächst mit dem Tool „Postman“.

Als die beiden Komponenten die grundlegenden erwarteten Funktionen beinhalteten, wurden sie verknüpft. Um grobe Fehler direkt auszuschließen, haben wir dies gemeinsam gemacht, da jeder sich mit seinem Teil am besten auskennt.

Nachdem nun die Aufgabenstellung erfüllt war, hatten wir noch genügend Zeit, weitere Funktionen einzubauen. Während Friedrich und Lukas sich damit beschäftigt haben, hat Florian sich in Docker eingelesen, sodass wir unser Projekt schlussendlich auch als Docker-Compose zur Verfügung stellen können.

1.2.3 Generelle Software-Architektur

Als generelle Struktur haben wir eine RESTful-Webservice-Anwendung gewählt. Diese besteht aus dem Frontend, welches der Nutzer im Browser aufruft. Dieses kommuniziert per HTTP über eine Middleware, in diesem Fall über eine dreischichtige JavaEE-Architektur, mit der Datenbank. Das nachstehende Schaubild zeigt diese Struktur detailliert auf. Die einzelnen Schichten werden im Folgenden genauer erläutert.

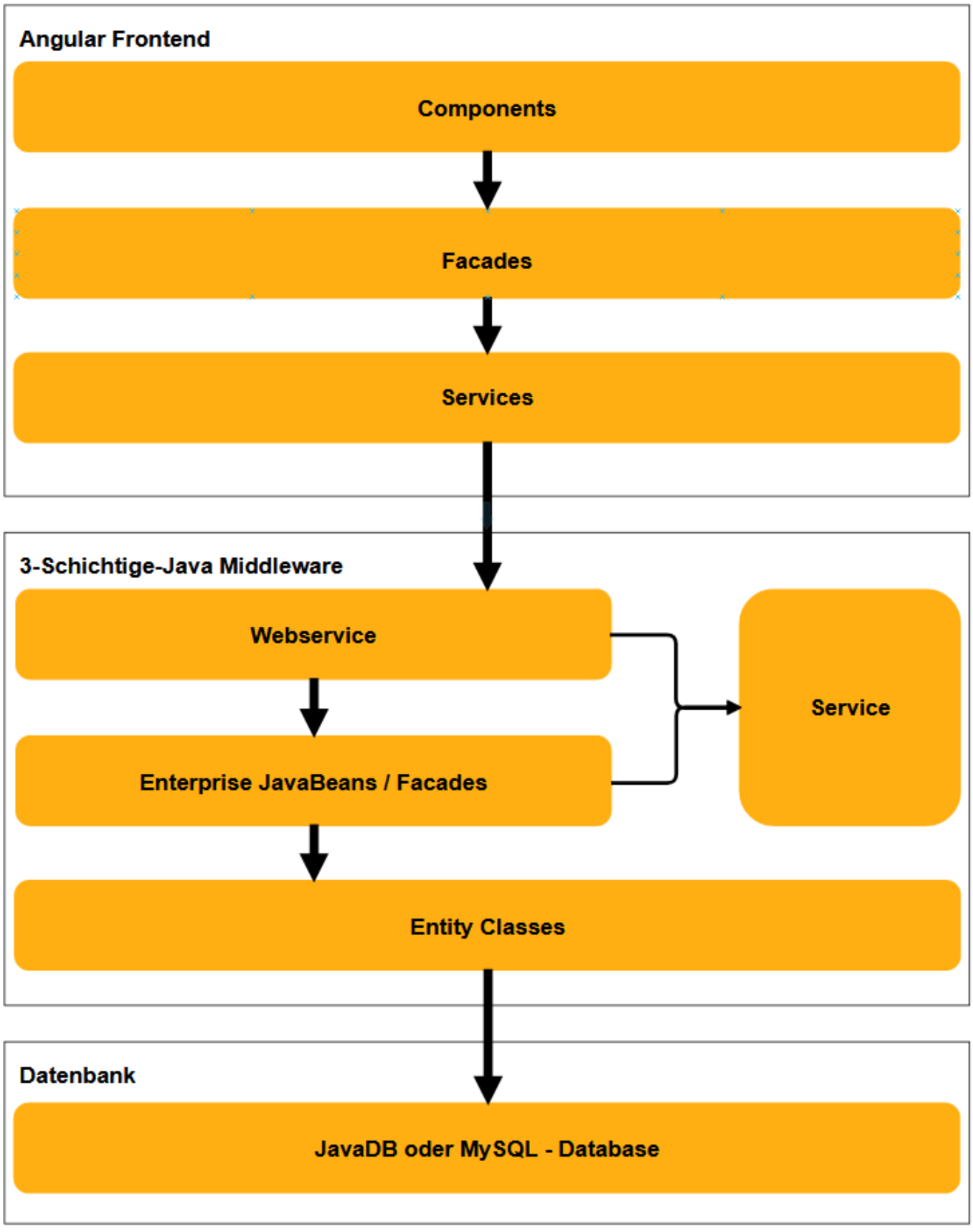


Abbildung 1: Software Architektur

Folgendes Client-Server-Diagramm veranschaulicht noch einmal die Kommunikation zwischen Front- und Backend und den Datenfluss innerhalb des Backends.

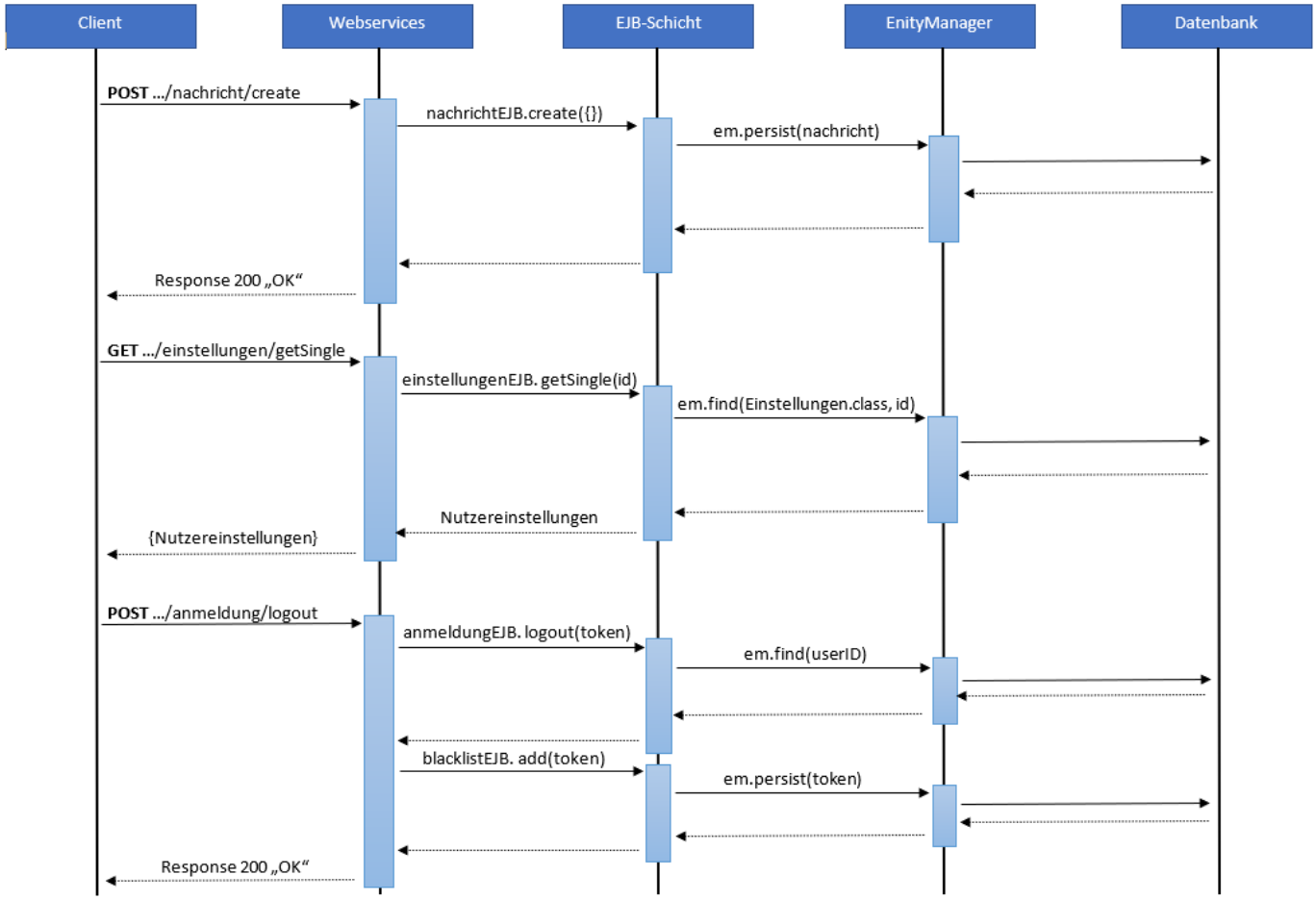


Abbildung 2: Client-Server Kommunikation

Wie zu erkennen ist, sendet der Client eine Anfrage an das Backend. Dieses verarbeitet dann die übergebenen Daten oder sucht angeforderte Ressourcen und schickt schließlich eine Antwort an den Client zurück, welche vom Frontend entsprechend visualisiert wird.

1.2.4 Backend

Architektur

Wie bereits erwähnt, gliedert sich unser Backend in eine dreischichtige Java-Architektur, welche sich in Webservices, Fassaden (EJBs) und Entitätsklassen unterteilt. Diese Struktur verbessert sowohl die Übersicht über das gesamte Projekt, als auch die Flexibilität und das Debugging. Änderungen können dank klarer Unterteilung einfach und gezielt vorgenommen werden, ohne eventuell Bugs in anderen Teilen der Anwendung zu verursachen. Um diese Vorteile noch besser auszunutzen, haben wir uns zudem an dem modernen Prinzip der Microservices orientiert und einzelne Funktionen, nach ihrer Zugehörigkeit geordnet, auf viele Webservices aufgeteilt.

Zur Verknüpfung mit der Datenbank nutzen wir die Java Persistence API, kurz JPA, mit dem EntityManager. Sie ermöglicht es, die Datenbankeinträge als Java-Objekte bereitzustellen und zudem im Persistence-Context zu speichern, sodass nicht immer neue Datenbankabfragen getätigt werden müssen.

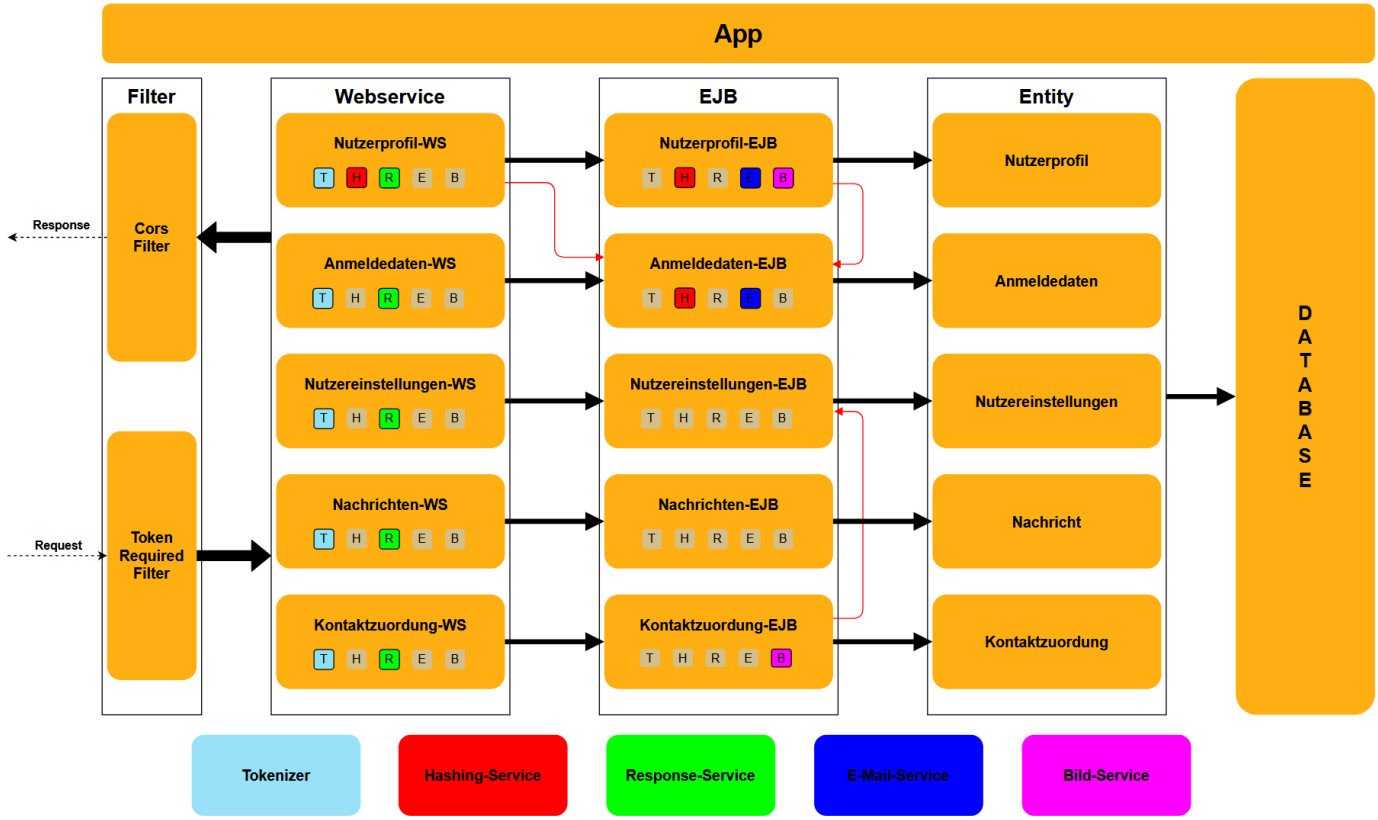


Abbildung 3: Architektur des Backends anhand ausgewählter Klassen

Das obenstehende Schaubild verdeutlicht die Beziehungen zwischen den Klassen und Schichten innerhalb des Backends. Eingehende Requests werden zunächst auf ein gültiges Token überprüft; einige Routen wie beispielsweise das Registrieren oder der Login sind davon ausgenommen. Ist ein gültiges Token enthalten, ruft der Webservice die entsprechende Methode in dem EJB auf, welche über den EntityManager mit den Entitätsklassen interagieren. Zusätzlich können auch Methoden aus Service-Klassen aufgerufen werden, welche zur Vermeidung von Redundanzen in eigene Klassen ausgelagert wurden. Mit den angeforderten Daten wird schlussendlich ein Response-Objekt erzeugt, in welchem im CORS-Filter noch HTTP-Header für die Erlaubnis des *Cross Origin Ressource Sharing* gesetzt sind. Das vollständige Response-Objekt wird dann an den Client zurückgesandt.

ConnectBackend-1.0 1.0 API

Packages

Package	Description
Application	Grundbaustein der Application
EJB	Fassaden mit der Geschäftlogik
Entity	Automatisch generierte Entitätsklassen der Datenbank
Filter	Filter für HTTP-Requests
Parser_Class	Klassen für das Parsen von JSON-Objekten
Service	Services der Application
WebService	WebServices für das Entgegennehmen der Anfragen

Abbildung 4: Alle Packages des Backends

Die gesamte API-Dokumentation mit den Beschreibungen aller Klassen, welche mit JavaDoc erstellt wurde, befindet sich unter „GFOS_docs_2021\Dokumentation\Java_Doc_Entry_point“ und kann im Browser geöffnet werden.

Filter

In unserer Architektur sind neben den drei erwähnten Schichten auch noch zwei Filter, der CORS-Filter und der TokenRequired-Filter, enthalten. Sie implementieren das ContainerResponseFilter- oder ContainerRequestFilter-Interface. Diese erlauben es, die Response beziehungsweise die Request „abzufangen“ und eventuell zu modifizieren. Beim CORS-Filter nutzen wir dies, um jeder ausgehenden Response die Header anzufügen, die CORS erlauben. Dies war nötig, um auch den von Angular gesendeten Preflight-Requests die nötigen Header hinzuzufügen. Der TokenRequired-Filter dient der Authentifizierung einer Anfrage, indem er das Token aus dem Authorization-Header überprüft. Er wird per NameBinding allen gewünschten Funktionen vorangestellt. Die Funktionsweise der Authentifizierung wird später beim Punkt „Sicherheit“ erläutert.

Package Filter

Filter für HTTP-Requests

See: Description

Class Summary

Class	Description
CorsFilter	CORS-Filter Dieser Filter fügt jeder Request Header hinzu, die CORS erlauben.
TokenRequiredFilter	TokenRequiredFilter Dieser Filter dient bie Methoden, die nur für authentifizierte Nutzer zur Verfügung stehen, zur Verifizierung des Tokens.

Annotation Types Summary

Annotation Type	Description
TokenRequired	Dieses Interface dient als Name-Binding für den TokenRequiredFilter.

Abbildung 5: Package Filter

Webservices

Die Webservices, welche wir wie bereits erläutert in viele Microservices unterteilt haben, nehmen die HTTP-Requests an die unterschiedlichen URIs an. Wir verwenden hierfür die passenden Protokoll-Methoden von HTTP (POST, GET, PUT, DELETE) und orientieren uns an dem CRUD-Prinzip (Create, Read, Update, Delete). Der Webservice ruft anschließend die zugehörige Methode in der Fassade auf. Die Rückgabe der Fassade gibt er in Form eines Response-Objektes an den Client zurück. Im Falle eines Fehlers wird eine Response mit dem passenden HTTP-Status-Code und der Fehlermeldung zurückgegeben, damit das Frontend diese visualisieren kann.

Package WebService

WebServices für das Entgegennehmen der Anfragen

See: Description

Class Summary

Class	Description
AnmeldedatenWS	Webservice für Anmeldung und Authentifizierung Diese Klasse stellt Routen bezüglich der Anmeldung sowie Authentifizierung der Nutzer bereit.
EmailWS	Webservice für das Versenden von Emails Diese Klasse stellt Routen bezüglich der Versendung von Emails bereit.
GruppenbildWS	Webservice für Gruppenbilder Diese Klasse stellt Routen bezüglich der Gruppenbilder bereit.
GruppenprofilWS	Webservice für Gruppenprofile Diese Klasse stellt Routen bezüglich der Gruppenprofile bereit.
GruppenzuordnungWS	Webservice für Gruppenzuordnungen Diese Klasse stellt Routen bezüglich der Gruppenzuordnungen bereit.
KontaktzuordnungWS	Webservice für Kontaktzuordnungen Diese Klasse stellt Routen bezüglich der Kontaktzuordnungen bereit.
NachrichtenWS	Webservice für Nachrichten Diese Klasse stellt Routen bezüglich des Schreibens, Versenden und Empfangen von Nachrichten bereit.
NutzereinstellungenWS	Webservice für Nutzereinstellungen Diese Klasse stellt Routen bezüglich der Nutzereinstellungen bereit.
NutzerprofilWS	Webservice für Nutzerprofile Diese Klasse stellt Routen bezüglich der Nutzerprofile bereit.
ProfilbildWS	Webservice für Profilbilder Diese Klasse stellt Routen bezüglich der Profilbilder bereit.

Package WebService Description

WebServices für das Entgegennehmen der Anfragen

Abbildung 6: Package WebService

Fassaden – Enterprise Java Beans

In den Fassaden befindet sich die Geschäftslogik, die vom Webservice aufgerufen wird. Mithilfe einer Instanz des Entity-Managers aus der JPA werden hier Daten abgefragt, aktualisiert, hinzugefügt oder gelöscht. Er stellt zudem durch das Object-Relational-Mapping (ORM) die Datenbankeinträge als Java-Objekte zur Verfügung, sodass mit diesen auch Überprüfungen vorgenommen werden können. Somit stellen die Fassaden die direkte Schnittstelle mit der Datenbank dar. Um auf dem Server Ressourcen zu schonen, sind sie als zustandslose EJBs deklariert und werden im Webservice mit der Annotation @EJB instanziiert.

Package EJB

Fassaden mit der Geschäftlogik

See: Description

Class Summary

Class	Description
AnmeldedatenEJB	Fassade für die Anmeldung und Authentifizierung Diese Klasse stellt Methoden bezüglich der Anmeldung sowie Authentifizierung der Nutzer bereit.
BlackListEJB	Fassade für die Blacklist der Tokens Diese Klasse stellt Methoden bezüglich der Blacklist für noch gültige Tokens bereit, die allerdings nicht mehr genutzt werden dürfen.
GruppenbildEJB	Fassade für die Gruppenbilder Diese Klasse stellt Methode bezüglich der Gruppenbilder bereit.
GruppenprofilEJB	Fassade für die Gruppenprofile Diese Klasse stellt Methoden bezüglich der Gruppenprofile bereit.
GruppenzuordnungEJB	Fassade für die Gruppenzuordnungen Diese Klasse stellt Methoden bezüglich der Gruppenzuordnungen bereit.
KontaktzuordnungEJB	Fassade für die Kontaktzuordnungen Diese Klasse stellt Methoden bezüglich der Kontaktzuordnungen bereit.
NachrichtenEJB	Fassade für die Nachrichten Diese Klasse stellt Methoden bezüglich Nachrichten bereit.
NutzereinstellungenEJB	Fassade für die Nutzereinstellungen Diese Klasse stellt Methoden bezüglich der Nutzereinstellungen bereit.
NutzerprofilEJB	Fassade für die Nutzerprofile Diese Klasse stellt Methoden bezüglich der Nutzerprofile bereit.
ProfilbildEJB	Fassade für die Profilbilder Diese Klasse stellt Methode bezüglich der Profilbilder bereit.
SystemNachrichtEJB	Fassade für die Systemnachrichten Diese Klasse stellt Methoden bezüglich der Systemnachrichten bereit.

Abbildung 7: Package EJB

Services

Einige Methoden, die in mehreren Klassen benötigt werden, haben wir aufgrund der Vermeidung von Redundanzen und zur Übersichtlichkeit des Codes in Services ausgelagert. Sie sind nach den Funktionalitäten der enthaltenen Methoden geordnet. Um auch hier serverseitig Ressourcen zu sparen, wurden diese ebenfalls als zustandlose EJB-Container definiert. Die einzelnen Klassen instanziieren sie wie die Fassaden mit der @EJB Annotation und können daraufhin alle Methoden aus diesen nutzen.

Ein Beispiel hierfür ist der ResponseService. Er stellt eine Methode zur Konstruktion eines Response-Objektes mit einem JSON-String an Daten und eine Methode zum Bauen eines Response-Objektes mit einem Fehler bereit. Somit muss dieses Objekt nicht in jeder Methode jedes Webservices konstruiert werden, sondern lediglich in den Webservices als EJB initialisiert werden. Mit „response.build(JSON-String)“ wird am Ende jeder Webservice-Methode dann ein Response-Objekt erstellt.

Package Service

Services der Application

See: Description

Class Summary

Class	Description
BildService	Service für Bilder Diese Klasse stellt Methoden für das Setzen von Bildern bereit.
EmailService	Service für das Versenden von Emails Diese Klasse stellt Methoden für das Versenden von Emails bereit.
HashingService	Service für das Hashen der Passwörter Diese Klasse stellt Methoden für das Hashen der Passwörter bereit.
ResponseService	Service für das Erstellen von Response-Objektes Diese Klasse stellt Methoden für das Erstellen von Response-Objekten zur Kommuniktion mit dem Frontend bereit.
Tokenizer	Service für das Erstellen und verifizieren von Tokens für die Authentifizierung des User Diese Klasse stellt Methoden für das Erstellen, Verifizieren und Auslesen von JSON-Webtokens bereit.

Abbildung 8: Package Service

Parser Classes

Um vom Frontend auch individuelle Daten im JSON-Format entgegennehmen zu können, die nicht einem Objekt der Entitätsklassen entsprechen, wurden eigene Klassen erstellt. Diese bestehen aus den benötigten Attributen sowie zugehörigen get- und set-Methoden und einem Konstruktor.

Package Parser_Class

Klassen für das Parsen von JSON-Objekten

See: Description

Class Summary

Class	Description
DeleteAttributes	Klasse, die zum Parsen eingehender JSON-Objekte genutzt wird.
GelesenGruppeAttributes	Klasse, die zum Parsen eingehender JSON-Objekte genutzt wird.
LeaveGroupAttributes	Klasse, die zum Parsen eingehender JSON-Objekte genutzt wird.
ResetPw	Klasse, die zum Parsen eingehender JSON-Objekte genutzt wird.
Status	Klasse, die zum Parsen eingehender JSON-Objekte genutzt wird.
UpdateNutzerdaten	Klasse, die zum Parsen eingehender JSON-Objekte genutzt wird.
ZuordnungAttributes	Klasse, die zum Parsen eingehender JSON-Objekte genutzt wird.

Package Parser_Class Description

Klassen für das Parsen von JSON-Objekten

Abbildung 9: Package Parser_Class

Entity Classes

Die Entitätsklassen werden vom EntityManager verwendet, um die Datenbankeinträge zu sogenannten Plain Old Java Objects, kurz POJOs, umzuwandeln. Dadurch kann in den Fassaden wie mit normalen Java-Objekten gearbeitet werden. Aufgrund des Persistence-Contexts werden Änderungen an diesen direkt in die Datenbank übernommen, wodurch Daten beispielsweise sehr unkompliziert und schnell aktualisiert werden können.

Um bei einigen Routen jedoch nur relevante Attribute an den Client zu geben, die dort auch wirklich angezeigt werden, müssen diese aus dem Persistence-Context entfernt werden. Ansonsten würden durch das Setzen auf null auch die Datenbankeinträge gelöscht. Theoretisch kann dafür die Funktion „detach()“ des EntityManagers genutzt werden, jedoch gab uns das Implementieren einer eigenen clone()-Methode, welche im Prinzip selbiges tätigt, mehr Freiraum, da die Attribute der Objekte beliebig modifiziert werden können. Zudem wird der JSON-String, welcher später an das Frontend gesendet wird, dadurch erheblich verkürzt. Attribute, die in der clone()-Methode auf null gesetzt werden, werden von dem GSON-Parser nicht übernommen, während EntityManager.detach() auch die „leeren“ Attribute mit in den JSON-String schreibt. Ein direkter Vergleich zeigt, dass dadurch eine erhebliche Menge an Datenvolumen gespart werden kann:

EntityManager.detach()

{"nachrichtid":"1","bezug":"?","inhalt":"Hallo, dies ist ein Test.","gesendetam":"Feb 9, 2021 7:47:13 AM","gelesen":"Feb 9, 2021 7:47:13
AM","empfaengerid":{"userid":"2","vorname":"Test2","nachname":"Test2","telefonnummer":"4567","beitrittsdatum":"Feb 9, 2021 7:46:45 AM","status":"Y","zuletztonline":"Feb 9, 2021 7:46:45
AM","kontaktzuordnungList":[],"kontaktzuordnungList1":[],"gruppenprofilList":[],"gruppenzuordnungList":[]},"senderid":{"userid":"1","vorname":"Test1","nachname":"XYZ","telefonnummer":"1234","beitrittsdatum":"Feb 9, 2021 7:46:12
AM","status":"X","zuletztonline":"Feb 9, 2021 7:46:45
AM","kontaktzuordnungList":[],"kontaktzuordnungList1":[],"gruppenprofilList":[],"gruppenzuordnungList":[]}}

Eigene clone()-Methode

{"nachrichtid":"1","bezug":"?","inhalt":"Hallo, dies ist ein Test.","gesendetam":"Feb 9, 2021 7:47:13 AM","gelesen":"Feb 9, 2021 7:47:13
AM","empfaengerid":{"vorname":"Test2","nachname":"Test2"},"senderid":{"vorname":"Test1","nachname":"XYZ"}}

Package Entity

Automatisch generierte Entitätsklassen der Datenbank

See: Description

Class Summary

Class	Description
Anmeldedaten	Automatisch generierte Entitätsklasse der Tabelle ANMELDEDATEN
Blacklist	Automatisch generierte Entitätsklasse der Tabelle BLACKLIST
Codes	Automatisch generierte Entitätsklasse der Tabelle CODES
Gruppenbild	Automatisch generierte Entitätsklasse der Tabelle GRUPPENBILD
Gruppenprofil	Automatisch generierte Entitätsklasse der Tabelle GRUPPENPROFIL
Gruppenzuordnung	Automatisch generierte Entitätsklasse der Tabelle GRUPPENZUORDNUNG
Kontaktzuordnung	Automatisch generierte Entitätsklasse der Tabelle KONTAKTZUORDNUNG
Nachricht	Automatisch generierte Entitätsklasse der Tabelle NACHRICHT
Nutzereinstellungen	Automatisch generierte Entitätsklasse der Tabelle NUTZEREINSTELLUNGEN
Nutzerprofil	Automatisch generierte Entitätsklasse der Tabelle NUTZERPROFIL
Profilbild	Automatisch generierte Entitätsklasse der Tabelle PROFILBILD
Systemnachricht	Automatisch generierte Entitätsklasse der Tabelle SYSTEMNACHRICHT

Abbildung 10: Package Entity

Datenbank

Alle Nutzer erhalten nach der Registrierung ein eigenes Nutzerprofil, was das Schlüsselelement in unserer Datenbankstruktur ist. Fast alle weiteren Tabellen beziehen sich mithilfe von Fremdschlüsseln auf die Nutzerprofile, wodurch auch der EntityManager auf alle Objekte und somit Attribute eines Nutzers zugreifen kann. Die Nutzerprofile und meistens auch die zugehörigen Datensätze aus anderen Tabellen, werden mithilfe von UUIDs als Primärschlüssel eindeutig identifiziert. Tabellen, welche einen zusammengesetzten Primärschlüssel benötigen würden, wurden um eine automatisch generierte ID in Form eines Integers erweitert, um die Identifikation in der Anwendung zu vereinfachen.

Ein Problem, auf das wir zu Beginn der Projektzeit stießen, waren die Profil- und Gruppenbilder, da es nicht die Aufgabe einer SQL-Datenbank ist, Dateien im konventionellen Sinn zu speichern. Letztendlich entschieden wir uns für eine Speicherung als Base64-String, welche wir als Binary Large Object (kurz BLOB) in der Datenbank speichern konnten. Um die Performance trotzdem aufrechtzuerhalten, werden sie in einer eigenen Tabelle getrennt von den restlichen Nutzerdaten verwaltet. Näheres zu diesem Aspekt wird auch unter dem Punkt Datenverbrauch erläutert.

Um atomare Datensätze zu erhalten und Redundanzen zu vermeiden, haben wir uns bei der Planung und Erstellung der Datenbank an den ersten drei Normalisierungen orientiert. Größtenteils werden so auch funktionale und transitive Abhängigkeiten vermieden, sodass wir eine möglichst gute Performance der Datenbank sicherstellen können. Bei einigen Tabellen haben wir uns jedoch auch bewusst dagegen entschieden, um gerade während des Entwicklungsprozesses die Übersichtlichkeit der Tabellen zu gewährleisten. Kontakt- und Gruppenzuordnung werden für jeden Nutzer einzeln gespeichert, da so individuelle Einstellungen wie Nutzername, Angepinnt-Status oder Stummschalten ermöglicht werden.

Das folgende vereinfachte ER-Diagramm veranschaulicht die Beziehungen der Tabellen und somit auch der Entitätsklassen zueinander. Zur Übersichtlichkeit wurden die Attribute außenvorgelassen, sie werden anschließend für jede Tabelle einzeln dargestellt.

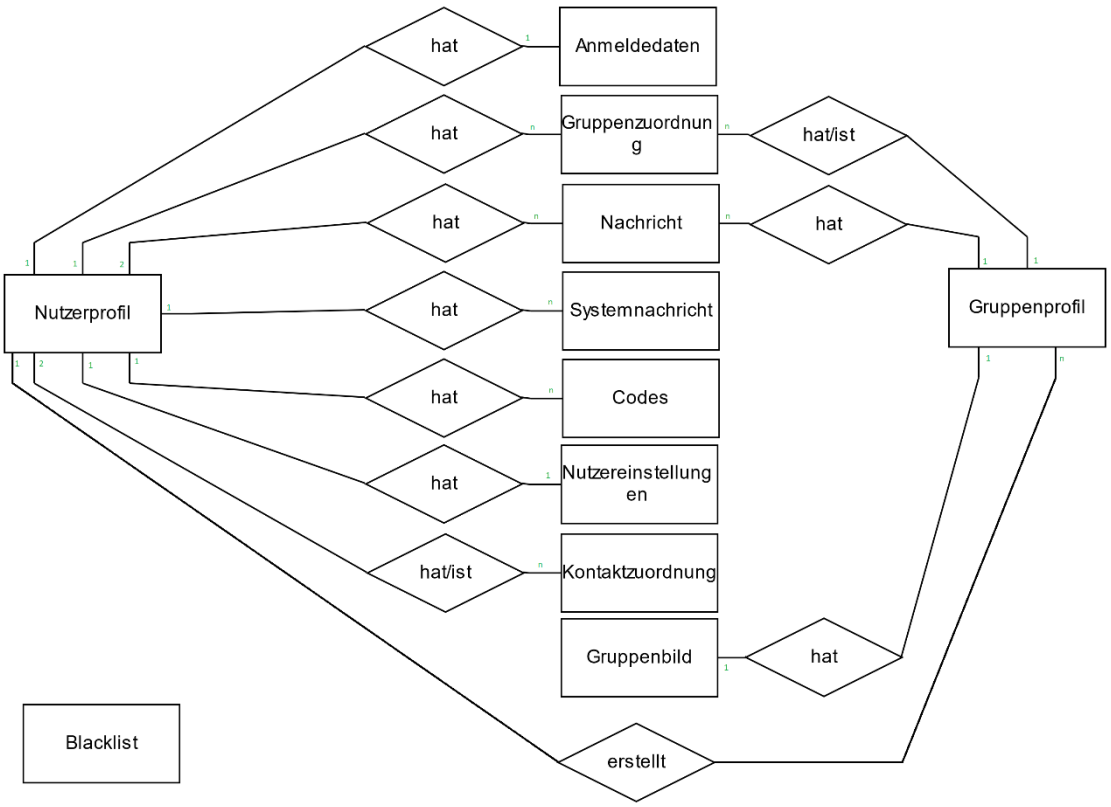


Abbildung 11: Vereinfachtes ER-Diagramm zur Veranschaulichung der Entitätsbeziehungen

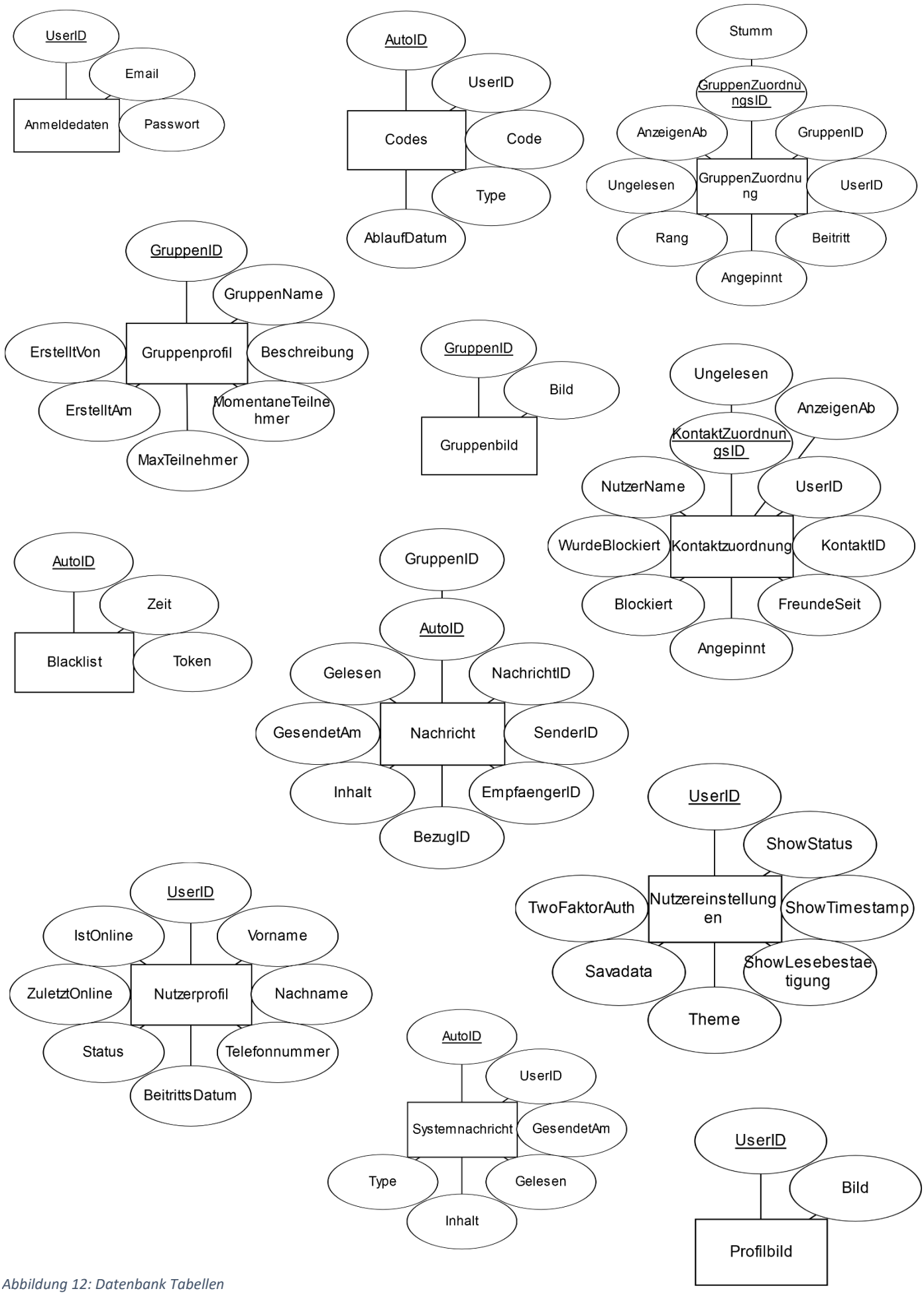


Abbildung 12: Datenbank Tabellen

1.2.5 Frontend

Architektur

Wie bei aktuellen Web-Applikationen üblich, wurde das Frontend in einem JavaScript-Framework programmiert. Wir haben uns für Angular mit TypeScript entschieden. Es handelt sich um eine SPA, eine Single-Page-Application. Die einzelnen Komponenten werden in der App-Komponente per Angular-Routing initialisiert. Dies bedeutet nach einmaliger minimal längerer Ladezeit beim ersten Aufruf der Seite deutlich kürzere Ladvorgänge beim Wechseln zwischen den Ansichten, was bei einem Messenger häufig der Fall ist.

Identisch zum Backend haben wir uns auch hier für eine mehrschichtige Architektur entschieden, wobei die einzelnen Schichten bestimmte Aufgabenbereiche abdecken. Auch hier kann die Software dadurch einfach erweitert oder Fehler behoben werden.

Das folgende Schaubild veranschaulicht die Struktur inklusive der Kommunikation allgemein. Im Folgenden werden die einzelnen Bestandteile genauer erläutert.

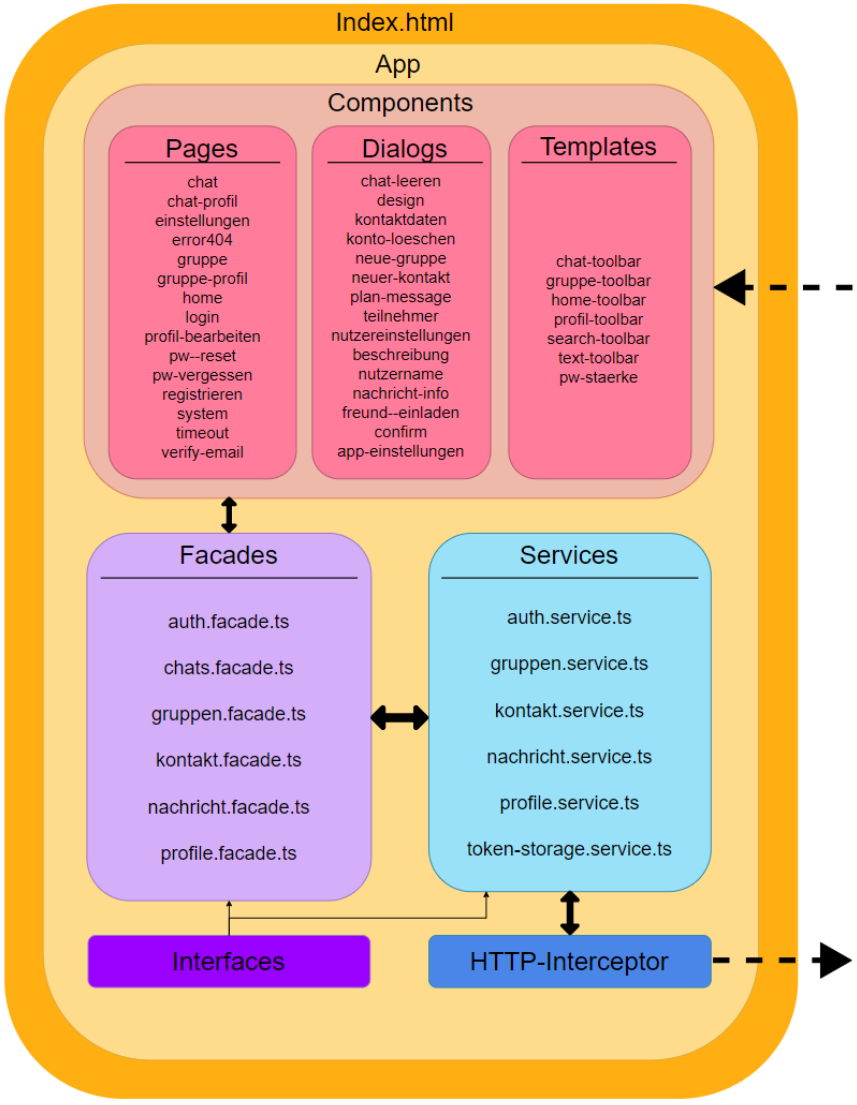


Abbildung 13: Architektur des Frontends

Die gesamte Dokumentation des Frontends mit den Beschreibungen aller Skripte, welche mit TypeDoc erstellt wurde, befindet sich unter „GFOS_docs_2021\Dokumentation\Type_Doc_Entry_point“ und kann im Browser geöffnet werden.

Angular

Neben der Vorerfahrung von Florian haben wir uns auch für Angular entschieden, da es mit Angular Material eine Lösung für das Design unserer App bietet, die sehr gut in das restliche Framework zu integrieren ist. Überzeugt hat uns außerdem der Komponenten-basierte Aufbau. Die Komponenten können dank zahlreicher Möglichkeiten einfach untereinander kommunizieren und fördern somit eine noch bessere Projektstruktur.

Die einzelnen Components sind zudem leicht verständlich, da im Gegensatz zu anderen Frameworks das Template (HTML), Stylesheet (SCSS) und die eigentliche TypeScript-Klasse getrennt voneinander vorliegen. TypeScript hilft uns dabei, die App einfacher zu debuggen, da die meisten Fehler durch das strict-type-checking direkt vom Compiler erkannt werden und behoben werden können.

RxJS

RxJS ist eine Bibliothek für JavaScript, die reaktive Programmierung ermöglicht und standardmäßig bei einem Angular-Projekt mit installiert ist.

Für die Kommunikation über HTTP mit dem Backend nutzen wir den Angular HTTP-Client. Dieser gibt immer Observables mit einem Datenstream zurück, auf das „Beobachter“ subscriben können, um die Daten zu erhalten. Solange sie nicht unsubscriben, werden sie automatisch über Änderungen „informiert“. Des Weiteren erfahren auch alle von einem Datenstream abhängigen Datenstreams über den Observer von dieser Änderung. Gerade bei einem Messenger, bei dem sich ständig Daten aktualisieren, können Daten so ohne weitere Abfragen einfach aktualisiert oder mehrfach verwendet werden.

Außerdem erlaubt RxJS uns mit den zahlreichen Pipes die Daten zu filtern oder sogar zu manipulieren. Dadurch und dank des implementierten State-Managements können insgesamt einige Requests an das Backend eingespart werden, was zum einen den Datenverbrauch für die Nutzer reduziert, als auch das Backend an sich entlastet.

Ebenen

Die unterste Ebene, die Components, dienen lediglich der Visualisierung von vorhandenen Daten, oder zur Eingabe von neuen, oder zu aktualisierenden, Daten. Woher diese Daten kommen, ist jedoch nicht von Relevanz. Abgefragt werden diese Daten von den Services auf der Ebene des Core-Layers mithilfe des HTTP-Clients von Angular. Hier ist es im Gegenzug nicht relevant, wo diese Daten benötigt werden. Deshalb dienen Fassaden als „Vermittler“ für die Daten zwischen Service und Component. Eine Instanz des Services ist somit nicht im Component nötig und die UI ist von der Daten-Ebene getrennt. Auf dieser Ebene kann auf die Observables subscribt werden oder, wie wir es in mehreren Fällen gemacht haben, ein State-Management implementiert werden.

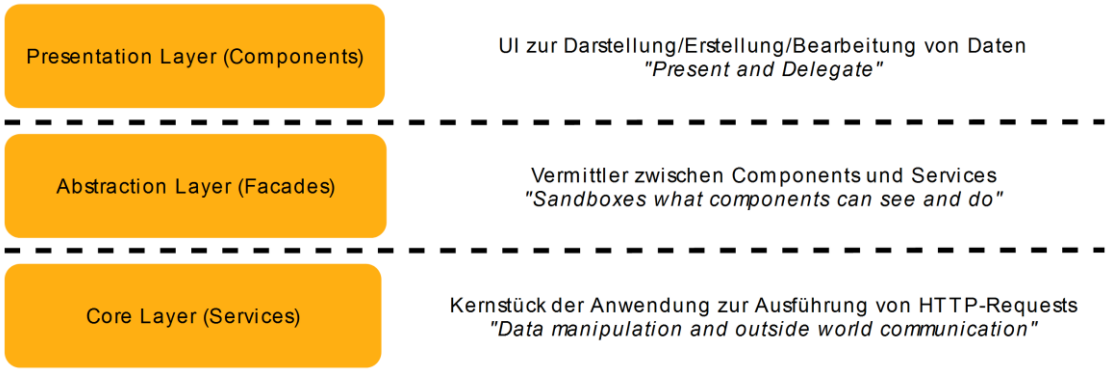


Abbildung 14: High-Level Abstraction nach https://dev-academy.com/angular-architecture-best-practices/

Reloading

Um einen modernen Messenger zu realisieren, muss ein automatisches Aktualisieren der Daten, insbesondere der Nachrichten in einem Chat, vorhanden sein. Beim Chatten möchte man direkt über neue Nachrichten informiert werden und diese nicht erst nach einem Schließen und erneuten Aufrufen der Seite angezeigt bekommen.

Da wir eine RESTful-API über HTTP implementiert haben, kann bei unserer App nur in Richtung vom Client zum Server kommuniziert werden. Ein „Abfragen nur nach Bedarf“ ist somit nicht möglich. Dies wäre bei einer Verbindung über WebSockets von Vorteil, da in diesem Fall auch der Server aufgrund des Bestehenbleibens der Verbindung Daten ohne Abfrage an den Client schicken kann.

Wir mussten deshalb einen Kompromiss zwischen aktuellen Daten und möglichst geringem Datenverbrauch finden. Entschieden haben wir uns für einen automatischen Reload alle zehn Sekunden, sofern es die Qualität der Internetverbindung zulässt. So kann der Nutzer rechtzeitig über neue Nachrichten oder ähnliches informiert werden, ohne ein zu hohes Datenvolumen zu verbrauchen.

State-Management

In unserer App haben wir ein State-Management eingebaut, um den Datenverbrauch für die Nutzer zu verringern. Im Folgenden werden die Vorteile sowie die Umsetzung genauer am Beispiel der Kontakt- und Gruppenliste erläutert, die einen zentralen Punkt in unserer Applikation darstellen.

Das State-Management sorgt dafür, dass die Kontakte und Gruppen zentral in einer Fassade als aktueller State gespeichert sind, auf den diverse Komponenten zugreifen können. Solange der Nutzer auf der Chatliste bleibt, werden die Kontakte, wie unter dem Punkt Reloading erwähnt, alle zehn Sekunden neu vom Server abgefragt. Anstatt sie direkt in der Komponente zu aktualisieren, wird zuerst der State überschrieben. Durch diese Änderung ändert sich auch die Anzeige (reaktive Programmierung in Form von Observables). Wechselt der Nutzer beispielsweise von der Kontaktlist zu einem Chat und anschließend zu dem Nutzerprofil des Kontaktes, so nutzt diese Komponente den selben State aus der Fassade. Somit muss keine erneute HTTP-Request getätigt werden; das Profil kann einfach aus der Kontaktliste gefiltert werden.

Die Vorteile sind eine Entlastung des Backends von der technischen Seite, da weniger Requests bearbeitet werden müssen. Aber auch für den Nutzer verbessert sich das Benutzererlebnis, da die App bei Folgeseiten schneller lädt und die Nutzer Datenvolumen sparen, gerade dadurch, dass die Bilder nicht erneut abgefragt werden. Hat ein Nutzer einmal keine gute Internetverbindung, so werden die Listen nicht alle zehn Sekunden aktualisiert, sondern erst, wenn die vorherige Aktualisierung beendet wurde. So wird eine riesige, sich aufbauende „Schlange“ an Requests verhindert, die praktisch nie enden würde.

Realtime-Search

Auf der Home-Ansicht und in einem Chat bieten wir den Nutzern die Möglichkeit einer Echtzeitsuche. Mit RxJS ist dies leicht zu implementieren, indem der Suchbegriff direkt mit der Kontaktliste oder den Nachrichten verglichen wird und diese dementsprechend gefiltert werden. Ohne neue Request an den Server wird dem Nutzer so direkt das Suchergebnis angezeigt.

Um eine möglichst weiträumige und differenzierte Suche zu ermöglichen, werden sowohl Suchbegriff als auch die Namen (Vorname, Nachname und Nutzername) oder die Nachricht zu jeweils einem zusammengeschrieben String aus Kleinbuchstaben ohne Leerzeichen umgeformt.

Progressive Web App

Wie die Aufgabenstellung vorgeschrieben hat, wurde der Messenger browserbasiert mit einem JavaScript-Framework entwickelt. Da Messenger jedoch vorwiegend auf Mobilgeräten genutzt werden, bieten wir über ein Webmanifest eine PWA an. Diese kann der Nutzer sich wie eine native App installieren und anschließend bequem über seinen Home-Bildschirm öffnen. Des Weiteren wird die Suchleiste des Browsers ausgeblendet, wodurch der Eindruck einer nativen App entsteht und die volle Größe des Displays ausgenutzt werden kann. Ein automatisch von Angular generierter Service-Worker bietet zudem Caching-Funktionalität für statischen Content wie das Logo. Manuell oder mit einem Online-Tool könnten noch weitere Funktionen hinzugefügt werden. Beispielsweise könnten Push-Benachrichtigungen versendet werden, auch wenn die Website nicht mehr aktiv geöffnet ist.

Viewports

Wie soeben aufgegriffen, werden Messenger in der Regel auf mobilen Endgeräten verwendet, da sie in der heutigen Welt zu einem täglichen Begleiter geworden sind. Aufgrund dessen wählten wir für das Design der App den Mobile-First-Approach und priorisierten damit das Smartphone als Endgerät. Dank dieses Ansatzes ist es leichter, auch die anderen Viewports zu gestalten. Insgesamt haben wir für folgende Gerätetypen und Ausrichtungen einen Breaktpoint über CSS-MediaQueries konfiguriert:

- Smartphone Hoch- und Querformat
- Tablet Hoch- und Querformat
- Computermonitore
- Computermonitore mit besonders hoher Auflösung (insbesondere WQHD)

Dabei gibt es im Endeffekt zwei Grundformen des Designs, welche bei den unterschiedlichen Gerätetypen nochmals weitreichend angepasst wurden. Dies sind einmal die kleineren Displays, wo man von der Chatliste zu den Chats zwischen einzelnen Seiten wechselt und zum anderen große Bildschirme, wo in einer Sidenav die Chatliste angezeigt wird und daneben der Chat aufgerufen wird.

User-Experience

Design

Für das Design nutzen wir wie zu Beginn erwähnt Angular Material, welches auf Material Design basiert. Dafür wurde ein App-übergreifendes Theme erstellt, dessen Farben eine generelle Ordnung und Struktur in der App schaffen. Dadurch konnten wir ohne großen Aufwand ein modernes und zeitgemäßes Design verwirklichen, das intuitiv zu bedienen ist. Praktisch alle Features sind selbsterklärend und werden von bekannten Icons visualisiert. Wie bei aktuellen Websites und Software üblich, kann der Nutzer zudem zwischen einem Light- und Dark-Mode unterscheiden. Alle Ansichten werden dementsprechend angepasst. Bei der Registrierung oder einer späteren Änderung kann der Nutzer sich beide durch eine Live-Änderung anschauen.

Fehler & Hilfen

Um die User-Experience weiter zu erhöhen und im selben Zug nicht nötige Requests an das Backend zu verhindern, werden die meisten Fehler bei Eingaben des Nutzers bereits im Frontend erkannt. Der Nutzer muss somit nicht warten, bis das Backend mit einer Fehlermeldung antwortet, sondern kann seine Eingabe direkt den Anforderungen anpassen. Um die Input-Felder zu validieren, nutzen wir die Validators von Angular und eigens programmierte.

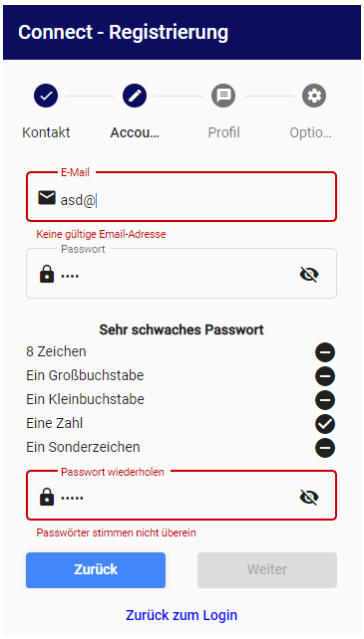


Abbildung 15: Fehleranzeige an Inputs

Bei den eigentlichen Requests können natürlich trotzdem noch Fehler auftreten. Um diese dem Nutzer anzuzeigen, nutzen wir Toastr, welche am unteren Bildschirmrand erscheinen. Diese zeigen wir auch bei einer erfolgreichen Operation, damit der Nutzer immer eine Bestätigung bekommt, dass seine Anfrage bearbeitet wurde.

An einigen Stellen geben wir dem Nutzer auch Hilfen, um seine Eingabe zu optimieren oder zu korrigieren. Beispielsweise ist hier das Wählen eines sicheren Passwortes zu nennen, wo interaktiv die Stärke des eingegebenen Passwortes überprüft wird (siehe Abb. rechts).

Abbildung 16: Toastr zur Visualisierung von Erfolg oder Fehler

Loading Spinner

In unserer App nutzen wir außerdem verschiedene Loading-Spinner, um dem Nutzer zu visualisieren, dass ein Prozess oder Daten noch laden. Bei einigen Aktionen nutzen wir den Ngx-Spinner mit Backdrop, damit der Nutzer in dieser Zeit keine weiteren Anfragen starten kann, wodurch sich die Ladezeit erheblich verlängern könnte. Von besonderer Relevanz ist dies bei einer unzureichenden Internetverbindung. Auf der Home-Seite mit der Liste an Chats und Gruppen haben wir uns für Skeleton-Loader entschieden, die dem Nutzer bereits während des Ladevorgangs eine Vorstellung der Ansicht bieten.

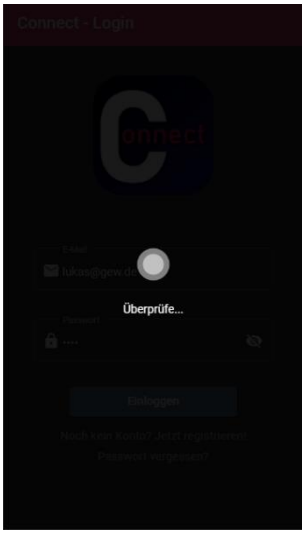


Abbildung 17: Loading-Spinner mit Backdrop

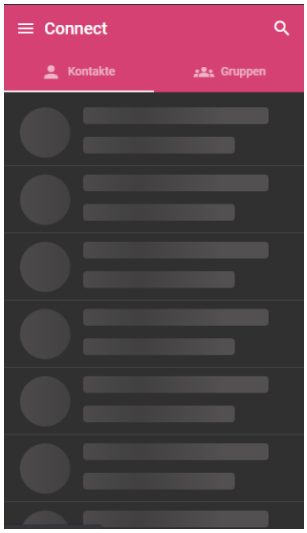


Abbildung 18: Skeleton-Loader

1.2.6 Sicherheit

Eine wichtige Angelegenheit neben möglichst differenzierten und gut durchdachten Features war es für uns, die Anwendung möglichst gut und effizient gegen Angriffe von außen abzusichern, sodass sie den heutigen Sicherheitsstandards weitestgehend, sofern es uns möglich war, entspricht. Aus diesem Grund haben wir folgende Sicherheitsfeatures eingebaut.

Hashing der Passwörter

Die Passwörter der Nutzer dürfen auf keinen Fall als Klartext in der Datenbank stehen, um im Falle eines Daten-Lecks weiterhin die Sicherheit der Nutzerkonten zu gewährleisten. Des Weiteren ist das Speichern in kryptografischer Form in der DSVGO vorgeschrieben, ein Verstoß könnte juristische Folgen haben. Aus diesen Gründen werden die Passwörter in unserer Applikation mit einer Einwegfunktion gehasht und der resultierende Hashwert in der Datenbank gespeichert. Aus diesem Hashwert ist das eigentliche Passwort nicht mehr zu berechnen. Bei einem Login wird das eingegebene Passwort dann erneut mit derselben Funktion gehasht und mit dem Hashwert aus der Datenbank verglichen.

Wir nutzen für das Hashing die kryptografische Funktion PBKDF2 mit dem Keyed-Hash Message Authentication Code SHA512. Ein Hacker könnte nun trotzdem noch mit einem sogenannten Rainbowtable-Angriff versuchen, die Passwörter herauszufinden. Dafür hat er möglichst viele denkbare Passwörter mit der verwendeten Funktion gehasht und in einer Datenbank gespeichert. Um ein Passwort daraufhin zu knacken, vergleicht er das gehashte Passwort mit allen Werten aus der Datenbank und sucht nach einer Übereinstimmung. Um dies zu erschweren, hängen wir an jedes Passwort vor dem Hashen noch einen geheimen SALT-Wert an, der das Passwort um 32 Zeichen verlängert.

Verifizierung der E-Mail

Da die E-Mail-Adresse in unserem System eine zentrale Rolle spielt (Login, Kontakterstellung, Passwort zurücksetzen), muss der Nutzer sie nach der Registrierung einmalig verifizieren. So kann sichergestellt werden, dass er Zugriff auf das angegebene E-Mail-Konto hat. Zudem werden auch Bot-Angriffe erschwert.

Dafür schicken wir bei erfolgreicher Registrierung eine E-Mail an die angegebene Adresse, die einen Link mit einem persönlichen Code in der URL enthält, der bei Aufruf des Links im Browser die E-Mail im System verifiziert. Über den zufällig erstellten Code kann das zugehörige Konto eindeutig verifiziert werden. Anschließend kann der Nutzer sich normal einloggen. Verifiziert er seine E-Mail-Adresse hingegen nicht innerhalb von acht Stunden, so wird das Konto gelöscht, damit der eigentliche Besitzer dieser E-Mail-Adresse sich gegebenenfalls selbst registrieren kann.

Passwort zurücksetzen

Hat ein Nutzer sein Passwort vergessen, so kann er es bequem per E-Mail-Adresse zurücksetzen. Dafür muss er auf der Login-Seite lediglich auf „Passwort zurücksetzen“ klicken. Ihm wird anschließend eine E-Mail mit einem persönlichen Link inklusive Code, so wie bei der Verifizierung der E-Mail-Adresse, zugesandt. Über diesen erreicht er eine Seite, auf der er mit einem gültigen Code (15 Minuten gültig) ein neues Passwort festlegen kann.

Der persönliche Link ist dabei notwendig, um das Zurücksetzen des eigenen Passwortes durch Außenstehende zu verhindern. Theoretisch kann nämlich jeder über das Formular eine E-Mail anfordern, jedoch erhält nur der Besitzer des Postfaches den Link. Dies hat er durch die Verifizierung zu Beginn sichergestellt. Erhält man folglich eine E-Mail, obwohl man das Passwort nicht zurücksetzen wollte, so ist dies ein Anzeichen für einen Versuch eines unautorisierten Zugriffs. Es wird dringlichst empfohlen, in den Einstellungen unter dem Punkt „Kontaktdaten“ ein neues Passwort festzulegen.

Authentifizierung & Sitzungsmanagement

Um sicherzustellen, dass nur authentifizierte Nutzer Zugriff auf das System haben, muss der Server überprüfen können, ob die Anfrage von einer berechtigten Person kommt. Für die Realisierung nutzen wir JSON Web Tokens, kurz JWT. Diese sind immer wie folgt aufgebaut:

xxxxx.yyyyy.zzzzz

Die unterschiedlichen Teile des Tokens werden jeweils mit einem Punkt voneinander getrennt. Der erste Teil ist der Header, in dem der Algorithmus zur Erstellung des Tokens und der Typ gespeichert wird. In unserem Fall ist dies „HS256“ und als Typ „JWT“. Im Payload, dem folgenden Teil, werden nützliche Informationen als Base64Url kodiert. Dies umfasst den Aussteller des Tokens, das Ablaufdatum und die UserID im Subject. Dabei handelt es sich um die „registered Claims“. Theoretisch könnten hier beliebig viele weitere private Claims angelegt werden. Damit die Tokens nun als gültig oder ungültig identifiziert werden können, wird noch der letzte Teil, die Signature benötigt, welche mithilfe eines SECRET_KEYS bei der Ausstellung des Tokens erstellt wurde.

Bei einem erfolgreichen Login sendet der Server als Antwort ein Token, welches zehn Minuten lang gültig ist und im Anschluss noch fünf Minuten verlängert werden kann, an den Client. Dieses wird je nach Verfügbarkeit im Session- bzw. Localstorage gespeichert. Bei jeder folgenden Request an den Server, wird dieses dann im Authorization-Header der HTTP-Request nach dem Bearer-Schema mitgesandt. Um es nicht manuell jeder Request hinzufügen zu müssen, nutzen wir den Angular HTTP-Interceptor, der den Authorization-Header bei jeder Request selbständig setzt. Hier wird auch der Status der Response systemübergreifend auf einen Fehlercode überprüft, um bei einem Fehler den Nutzer darauf aufmerksam zu machen. Wird für die Route ein gültiges Token benötigt, gekennzeichnet durch das @TokenRequired-NameBinding, so wird in dem TokenRequiredFilter das Token aus dem Header auf Gültigkeit überprüft. Ist es gültig, so wird die gewünschte Ressource angefordert und an den Client eine Response gesendet. Ist das Token hingegen vor maximal fünf Minuten abgelaufen, so wird ein neues Token mit dem Status 409 an das Frontend gesendet und das Frontend fragt mit dem neuen, wieder gültigen Token erneut den Server an. Eine Ausnahme bilden hierbei die Request, die in einem definierten Intervall automatisch erneut aufgerufen werden, wie beispielsweise die Chat- und Gruppenliste. Da hier keine Aktivität des Nutzers festgestellt werden kann, wird das Token nicht erneuert. Ist das Token ungültig und kann nicht mehr erneuert werden, so wird direkt eine Response mit dem Status 401 „Unauthorized“ an den Client gesandt. Der Nutzer wird daraufhin automatisch ausgeloggt. So haben wir auch ein Sitzungsmanagement mit Session-Timeout realisiert. Vor erneuter Abfrage von Daten muss der Nutzer sich erneut einloggen.

Da auch ein Wechsel der Ansicht, wo eventuell aufgrund des State-Managements keine Request getätigt werden muss, als aktive Aktion des Nutzers zu werten ist, überprüft ein RouteGuard auch dann das Token serverseitig. So wird sichergestellt, dass auch hier eventuell ein neues Token ausgestellt wird und ein aktiver Nutzer nicht ausgeloggt wird. Zudem sichert der Guard Seiten, die nur authentifizierte Nutzer aufrufen können. Unbefugte können beispielsweise nicht auf die Home-Seite mit der Chatliste gelangen, ohne sich vorher einzuloggen. Auch sie würden auf die Timeout/Unauthorized-Seite weitergeleitet werden.

Die im Payload kodierte UserID stellt des Weiteren sicher, dass Nutzer nur Zugriff zu ihren eigenen Ressourcen haben, weil das Backend die UserID aus dem Payload des Tokens dekodiert und sie nicht als Klartext im Body einer POST-Request steht. Um ein gültiges Token mit einer anderen UserID zu erhalten, müsste sich der Nutzer mit den zugehörigen Account-Daten im System anmelden.

2-Faktor-Authentifizierung

Heutzutage besteht bei den meisten Applikationen die Möglichkeit einer 2-Faktor-Authentifizierung. Dies erhöht die Sicherheit, da Unbefugten der Zugang nochmals erschwert wird. Selbst wenn ein Unbefugter die Zugangsdaten eines Nutzers erlangt hat, muss er beim Login-Prozess mithilfe eines Codes bestätigen, dass er Zugriff auf das zugehörige E-Mail-Postfach hat.

Wir aktivieren die 2-Faktor-Authentifizierung nicht standardmäßig, da es dem Nutzer selbst überlassen bleiben soll. Möchte er seine Daten besser schützen und nimmt dafür jedoch einen etwas längeren Anmelde-Vorgang in Kauf, so kann er die 2-Faktor-Authentifizierung unter den Einstellungen beim Punkt App-Einstellungen aktivieren. Ab dem nächsten Login-Vorgang erhält er immer eine E-Mail mit einem Code, der zur Verifizierung des Vorgangs eingegeben werden muss, bevor er Zugriff auf seine Chatliste und alle weiteren Funktionen erhält.

Datenschutz

Der Schutz von privaten Daten spielt in der heutigen Gesellschaft eine immer größere Rolle. Wie bei gängigen Messengern bieten wir aus diesem Grund mehrere Funktionalitäten, um die Privatsphäre der Nutzer zu schützen. Direkt bei der Registrierung kann ausgewählt werden, ob andere Nutzer das eigene Profilbild, den Status und den Online-Status sehen dürfen. Außerdem kann auch auf das Senden von Lesebestätigungen verzichtet werden, dies betrifft jedoch Einzelchats, in Gruppen versendet jeder Nutzer Lesebestätigungen, da Gruppen hauptsächlich der schnellen Kommunikation von wichtigen Nachrichten dienen.

Das Hinzufügen von Kontakten basiert des Weiteren auf der Zuordnung über die E-Mail-Adressen der Nutzer. So kann weitestgehend sichergestellt werden, dass Unbekannte nicht willkürlich mit jedem Nutzer schreiben können, da es sehr schwierig ist, eine E-Mail-Adresse zu erraten. Sollte der Fall dennoch eintreten, so kann ein Nutzer auf der Profilseite blockiert oder gelöscht werden.

Auch der Datenschutz wird dadurch erhöht, weil Kontakte nicht über eine Suche aller Nutzer erstellt werden, wodurch viele persönliche Details preisgegeben werden würden. Ist man mit unbekannten Personen in einer Gruppe, die nicht auf der eigenen Kontaktliste stehen, kann man diese trotzdem ohne Preisgabe der E-Mail-Adresse als Kontakte hinzufügen.

1.2.7 Datenverbrauch

Bei einem Messenger spielt der Datenverbrauch eine große Rolle, da man auch unterwegs häufig auf seine Chats zugreifen möchte, jedoch kein WLAN zur Verfügung steht. Um den eigenen Datenvolumen-Verbrauch so gering wie möglich zu halten, haben wir einige Vorkehrungen getroffen. Einige, wie unsere Reloading-Strategie oder das State-Management, haben wir bereits genauer erläutert. Auf zwei weitere Punkte möchten wir hier noch genauer eingehen.

Bilder & Dateien

Da in der Aufgabenstellung nur die Nutzung einer SQL-Datenbank erlaubt war, mussten wir für die Speicherung der Profilbilder, welche wir auf jeden Fall realisieren wollten, eine andere Möglichkeit finden, als die konventionellen, heute gängigen Methoden. Normalerweise würde man sie auf einem FTP-Server, in einem Content Delivery Network oder auch Cloud-Speichern wie Firebase oder DigitalOcean speichern. Wir mussten sie allerdings in der MySQL-Datenbank speichern, ohne hohe Performance-Einbrüche zu erleiden. Deshalb haben wir die Profil- und Gruppenbilder in einen Base64-String umgewandelt und als BLOB gespeichert. Die Requests für die Chat- und Gruppenliste mit den Profil- und Gruppenbildern nehmen bei einigen Kontakten und Gruppen allerdings recht viel Zeit in Anspruch und verbrauchen einige Megabyte an Datenvolumen. Je nach Qualität und Auflösung des Bildes kann sich das Volumen nochmals erheblich erhöhen. Auf das Versenden von Bildern oder Dateien in einem Chat haben wir aus diesem Grund verzichtet.

Datensparmodus

Um Nutzern aus dem eben erwähnten Grund die Möglichkeit zu geben, Datenvolumen zu sparen, wurde des Weiteren ein Datensparmodus entwickelt. Wird dieser aktiviert, so werden die Bilder nicht mehr geladen und stattdessen wie bei Nutzern ohne Profilbild ein Shorttag angezeigt. Ein direkter Vergleich zeigt die Wirkung des Datensparmodus, auch bei unterschiedlicher Auflösung der Bildern:

	3 Bilder mit geringer Auflösung		1 Bild mit hoher Auflösung	
Ohne Datensparmodus	376ms	1,5MB	381ms	1,7MB
Mit Datensparmodus	95ms	1,8KB	64ms	1,2KB

Tabelle 2: Datensparmodus

Wie an der Tabelle zu erkennen ist, lässt sich mit dem Datensparmodus eine erhebliche Menge Datenvolumen einsparen und auch die Ladezeit der App verkürzen.

1.3 Nachbetrachtung

Welche Ziele wurden erreicht?

Insgesamt lässt sich sagen, dass alle unsere Ziele erreicht wurden. Die in der Aufgabenstellung vorgegebenen Aspekte waren recht schnell implementiert, sodass wir noch einige Wochen Zeit hatten, um eigene Features zu implementieren. Dies betrifft vor allem Features bezüglich der Sicherheit des Systems und der Verminderung des benötigten Datenvolumens. All dies konnte mit einer intuitiven und modernen UI verbunden werden, welche eine gute User-Experience ermöglicht. Die letzten zwei Wochen konnten des Weiteren noch genutzt werden, um gemeinsam die Dokumentation zu verfassen, die letzten Fehler zu beheben und das Projekt sogar als Docker-Images zur Verfügung zu stellen.

Wo besteht noch Verbesserungsbedarf? Welche Features hätten wir gerne noch eingebaut?

Trotz der ausführlichen und sicheren Umsetzung gibt es an einigen Stellen Optimierungsbedarf, welcher mit den vorgegebenen Technologien jedoch nicht möglich war oder zu dem wir doch keine Zeit mehr hatten.

- Wie erwähnt, werden in unserer App alle fehlerhaften Aktionen direkt im Frontend unterbunden. Dies verbessert die User-Experience, stellt allerdings auch eine kleine Sicherheitslücke dar. Theoretisch kann sich ein Nutzer unbefugt Zugang zu diesen Funktionen verschaffen, obwohl er dazu keine Berechtigung hat. Aus diesem Grund würden wir zusätzlich noch eine serverseitige Überprüfung bei diesen Routen einbauen.
- Das Reloaden der Daten auf der Home-Seite oder in den Chats verursacht zum Teil einen hohen Datenverbrauch, gerade bei den Kontakten und Gruppen mit Bildern. Hätten wir noch Zeit gehabt, hätten wir deshalb noch eine Route eingebaut, die beim Server abfragt, ob es neue Daten gibt und nur wenn dies der Fall ist, die eigentlichen Daten wirklich neu abfragt.
- Wie unter dem Punkt Datenverbrauch/Bilder bereits erwähnt, hätten wir die Bilder nicht in einer SQL-Datenbank gespeichert, sondern eine andere Speicherung bevorzugt. Hätten wir dies umgesetzt, wäre auch das Verschicken von Bildern und Dateien eine mögliche Erweiterung der App.
- Da wir unsere App bereits als PWA zur Verfügung stellen, ist bereits ein einfacher Service-Worker integriert. Gerade bei einem Messenger wäre es sinnvoll, über diesen Push-Benachrichtigungen zu verwirklichen, um den Nutzer auf neue Nachrichten hinzuweisen, selbst wenn er die App nicht geöffnet hat.
- Die 2-Faktor-Authentifizierung hätten wir ebenfalls gerne weiter verbessert. Momentan muss jeder Login-Vorgang mit einem Code verifiziert werden, bei modernen Anwendungen jedoch nur bei einem Login von einem neuen Gerät oder nach einer gewissen Zeitspanne. Diese Funktionalität würde die Sicherheit aufrechterhalten und gleichzeitig die User-Experience noch einmal verbessern.

Fazit

Alles in allem lässt sich festhalten, dass sich unser Zeitmanagement bewährt hat und wir alle Ziele umsetzen konnten. Trotz hohen Aufwandes hat es sehr viel Spaß gemacht und wir haben eine Menge gelernt. Dank der Docker-Images konnte die App auch auf einem Server betrieben werden. Dadurch konnten wir den Messenger praxisnah testen und nutzen und waren begeistert von seiner Funktionsweise, die wir in einem so kurzen Zeitraum implementiert haben.

2. Benutzerhandbuch

2.1 Inhaltsverzeichnis

2.2 Installation

Zur Installation des Projektes stellen wir Ihnen zwei Varianten zur Verfügung. Sie können es zum einen über die Netbeans IDE aufsetzen oder zum anderen über Docker. Wir empfehlen Ihnen, sofern Sie mit der Nutzung von Docker vertraut sind, die Installation mit dem Docker-Compose File, da dies weniger Aufwand für Sie bedeutet. Folgen Sie bitte je nach Entscheidung der folgenden Anleitung.

2.2.1 Installation mit Docker

Alle benötigten Dateien für eine Installation mit Docker finden Sie in dem Ordner *GFOS_docs_2021\Projekt\ConnectDocker.* Die DOCKERFILEs für Datenbank, Backend und Frontend befinden sich in den so benannten Ordnern zusammen mit den nötigen Skripts. Diese dürfen auf keinen Fall ausgetauscht werden!

Wir stellen Ihnen alle Software-Komponenten als Docker-Compose zur Verfügung, um Ihnen die Installation möglichst einfach zu gestalten. Wir erläutern Ihnen die Installation hier am Beispiel der Nutzung der aktuellsten Docker-Desktop Version (compose ist Teil der CLI) auf einem Windows-PC. Nutzen Sie ein anderes Betriebssystem oder eine ältere Docker-Version, so finden Sie die zugehörigen Commands in der README des Verzeichnisses.

1. Schritt

Damit der GlassFish-Server im Folgenden eine Verbindung zur Datenbank aufbauen kann, muss diese zunächst mit der Datei „setupNewDB.bat“ gestartet werden. Es wird ein neues Image mit leerer Datenbank erstellt und in einem Container gestartet. In Docker-Desktop können Sie die Konsolen aller gestarteten Container innerhalb dieses Container-Netzwerkes beobachten. Gleichzeitig stellen wir Ihnen unter Port 3970 phpMyAdmin für die Verwaltung der Datenbank zur Verfügung. Die Zugangsdaten dazu sind der Nutzer „connect“ mit dem Passwort „gfos21FFL“. Möchten Sie ein Image einer bereits erstellten Datenbank erneut starten, so nutzen Sie bitte die Datei „startExistingDB.bat“. Sobald Sie sicherstellen können, dass die Datenbank erreichbar ist und Anfragen entgegennehmen kann, fahren Sie mit Schritt 2 fort.

2. Schritt

Mit der Datei „setupFEandBE.bat“ erstellen Sie neue Images für das Front- und Backend, welche in zwei Containern gestartet werden. Haben Sie bereits ein Image erstellt, so können Sie dieses mit der Datei „startFEandBE.bat“ wieder starten. Hier ist das Frontend anschließend unter Port 4200 erreichbar.

Möchten Sie die Anwendung beenden, so können Sie dies mit der Datei „stopAll.bat“ tun. Diese beendet alle laufenden Container des Netzwerkes.

Konfiguration für Serverbetrieb

Möchten Sie unsere Software nicht nur lokal, sondern auf einem Server testen, was dank Docker einfach möglich ist, so müssen Sie zwei Variablen anpassen:

1. Für den Versand der E-Mails, welche Links, die den Nutzer zu der Homepage weiterleiten, enthalten, muss in der *docker-compose* die Umgebungsvariable „HOST“ auf die IP-Adresse oder Domain des Servers gesetzt werden, auf dem die Anwendung laufen soll.
2. Damit das Frontend mit dem Backend über HTTP kommunizieren kann, muss in der Datei *environment.prod.ts* im Verzeichnis *GFOS_docs_2021\Projekt\ConnectDocker\Angular_Frontend\ConnectFrontend\src\environments* der Host in der Umgebungsvariable „baseURL“ ebenfalls zu der IP-Adresse oder Domain des Betriebsservers geändert werden.

Zudem empfehlen wir Ihnen eine SSL-Verschlüsselung, um die Daten der Nutzer vor Zugriffen von außerhalb des Netzwerkes zu schützen. Über beispielsweise *Let´s Encrypt* erhalten Sie ein Zertifikat kostenlos innerhalb weniger Sekunden.

Sollte der Container des Backends abrupt mit der Fehlermeldung *„standard_init_linux.go:190: exec user process caused "no such file or directory""* stoppen, so ändern Sie bitte den Line-Feed der Datei *GFOS_docs_2021\Projekt\ConnectDocker\GlassFish_Backend/bootstrap.sh* von CRLF zu LF.

2.2.2 Installation mit Netbeans

Backend

Um das Backend auf diesem Weg aufzusetzen, benötigen Sie wie erwähnt die Netbeans IDE. Wir haben die LTS Version 12.0 mit der JDK 8.281 genutzt. Es ist wichtig, dass Sie keine neuere JDK als Version 8 nutzen, da der von uns genutzte GlassFish Application-Server nur die Versionen bis JDK-8 unterstützt. Netbeans können Sie unter folgendem Link herunterladen: **NetBeans installieren**

Sobald Sie Netbeans installiert haben müssen Sie das Projekt in Netbeans importieren. Dafür klicken Sie oben links auf „File“ und anschließend auf den Reiter „Open Projects“. Wählen Sie dann unter Ihrem Speicherort den Ordner *GFOS_docs_2021\Projekt\Local\Connect_Backend* aus. Dieser wird normalerweise von Netbeans visuell hervorgehoben.

Sollten Sie nebenstehende Fehlermeldung erhalten, so klicken Sie bitte auf „Install nb-javac“ und starten Sie die IDE bei Aufforderung neu.

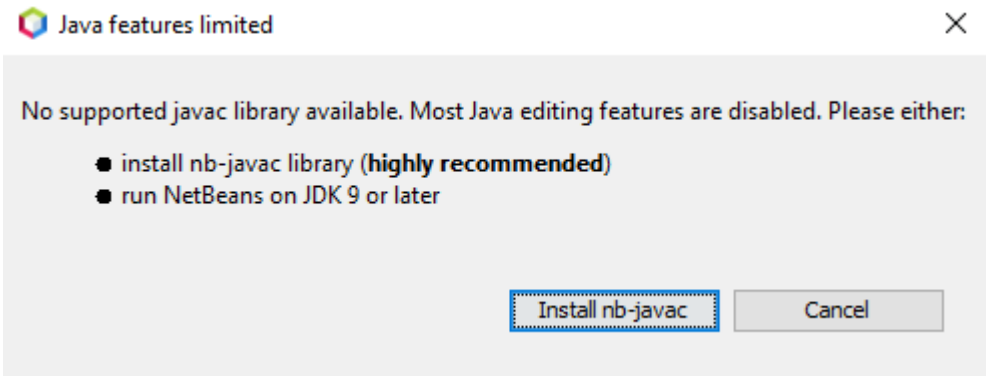


Abbildung 19: NetBeans Fehler beim Importieren des Backends

Haben Sie das Projekt erfolgreich importiert, so benötigen Sie noch den GlassFish Application-Server, um das Backend später für die Nutzung bereitzustellen. Dafür navigieren Sie bitte zum Reiter „Tools“ in der Navigationsleiste und dann zum Reiter „Servers“. In dem sich öffnenden Fenster klicken Sie links unten auf „Add Server“ und wählen „GlassFish Server“ aus. Drücken Sie danach auf „Next“ und wählen Sie als Version 5.1.0 aus. Akzeptieren Sie nun noch die Lizenzbedingungen und klicken Sie auf „Download now“. Der GlassFish Server wird nun automatisch in das richtige Verzeichnis installiert. Mit einem Klick auf „Next“ und „Finish“ schließen Sie das Fenster wieder. Da wir in unserem Projekt auch Emails versenden, es aber in der aktuellen GlassFish Version einen Bug gibt, muss in dem Installationsverzeichnis noch ein Ordner gelöscht werden:

In dem Ordner *C:\Users\<User>\GlassFish_Server\glassfish\modules\endorsed* muss aus dem .jar-File *grizzly-npn-bootstrap* der Ordner „sun“ vollständig entfernt werden. Nutzen Sie dafür einen Encoder ihrer Wahl, beispielsweise WinRAR.

In einem letzten Schritt müssen Sie nun noch eine Datenbank einrichten. Hierzu haben wir die in Netbeans integrierte JavaDB genutzt. Auch hierfür stellen wir Ihnen zwei Wege bereit:

Der erste, einfachere, Weg besteht lediglich darin, die Datei „import_to_Netbeans“ im Verzeichnis *GFOS_docs_2021\Projekt\Local\Database* auszuführen. Diese kopiert die Datenbank mit

Beispieldatensätzen in das Standard-Netbeans-Verzeichnis. Die Emailadresse setzt sich jeweils aus unserem Vornamen und der Endung „@gew.de“ zusammen; das Passwort ist zu Testzwecken immer „1234“.

Die andere Möglichkeit besteht darin, die Datenbank manuell mit dem Skript „JavaDB_init“ einzurichten. Dafür klicken Sie bitte in der Sidebar am linken Bildschirmrand auf den „Services“ Tab und klappen dann den Reiter „Databases“ aus und machen einen Rechtsklick auf „Java DB“. Sollte der Services-Tab bei Ihnen nicht angezeigt werden, so können Sie ihn mit der Tastenkombination Strg + 5 einblenden. Wählen Sie nun „Create Database“. Im neuen Fenster erstellen Sie dann eine Datenbank mit dem Namen „ConnectDB“ und legen als User „FFL“ mit gleichnamigem Passwort „FFL“ an. Bestätigen Sie ihre Eingaben mit OK.

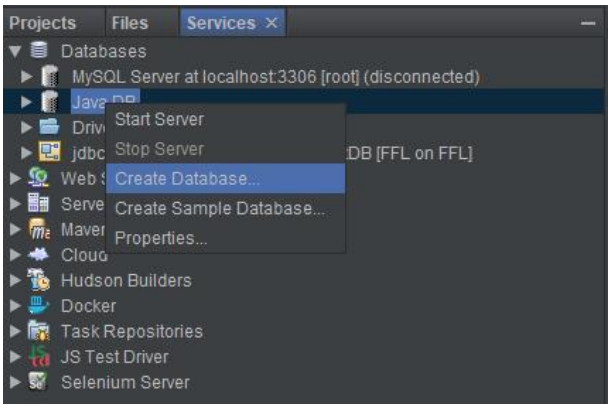


Abbildung 20: Hinzufügen einer neuen Datenbank

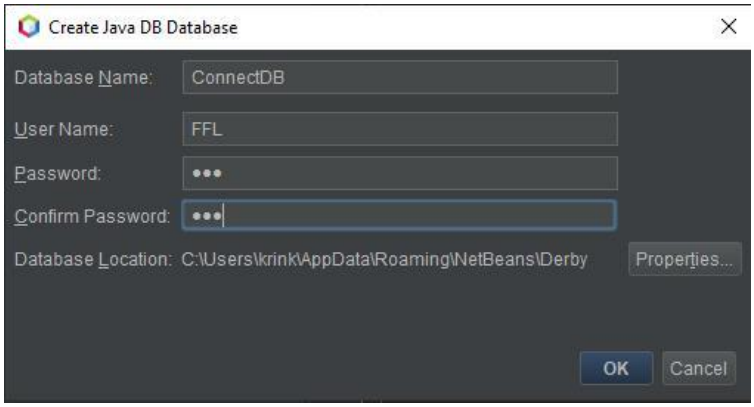


Abbildung 21: Konfigurieren der Datenbank

Führen Sie nun einen Doppelklick auf *jdbc:derby://localhost:1527/ConnectDB [FFL on FFL]* aus, wodurch Sie sich mit der Datenbank verbinden. Beim ersten Mal müssen Sie sich mit den eben angelegten Nutzerdaten einloggen, wir empfehlen hier die Option „Remember Password“. Klappen Sie nun den Reiter „FFL“ aus, ist dieser bei Ihnen noch nicht vorhanden, so suchen Sie unter „Other Schemas“ nach „FFL“ und wählen es per Rechtsklick und Klick auf „Set as Default Schema“ als Standard-Schema aus. Nach einem Rechtsklick auf „Tables“ klicken Sie auf „Execute Command“. In dem sich öffnenden Textfeld fügen Sie den Inhalt der Datei „JavaDB_init“ unter *GFOS_docs_2021\Projekt\Local\Database* ein und führen es mit einem Klick auf das Icon  aus , welches Sie am oberen Rand des Feldes finden.

Das Backend ist nun betriebsbereit. Sie starten den GlassFish-Server mit einem Klick auf das Icon in der Toolbar. Er läuft nun unter Port 8080 auf *localhost*.

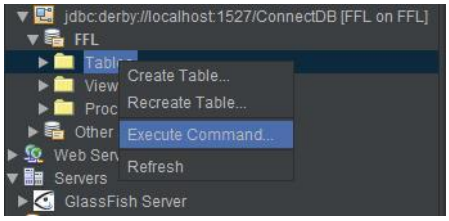


Abbildung 22: Tabellen erzeugen

Frontend

Für das Frontend benötigen Sie die JavaScript-Laufzeitumgebung NodeJS. Sollten Sie diese nicht installiert haben, so laden Sie sie bitte von folgender Seite herunter und folgen Sie den Installationsanweisungen: **NodeJS installieren**. Anschließend installieren Sie bitte mit der Datei „installDependencies.bat“ im Ordner *GFOS_docs_2021\Projekt\Local\Connect_Frontend* die benötigten Packages. Im Anschluss können Sie das Frontend mit der Datei „startFrontend.bat“ im selbigen Ordner lokal starten. Es ist im Browser dann unter *locahost:4200* erreichbar.

SMTP-Server konfigurieren

Möchten Sie nicht den von uns konfigurierten SMTP-Server für Gmail nutzen, so können Sie in der jeweiligen Version (Local oder Docker) unter *src\main\java\Service\EmailService.java* die Konfiguration anpassen.

2.3 Quick-Start

Öffnen Sie die Anwendung im Browser, so werden Sie immer zuerst auf der Login-Seite landen. Um sich zunächst ein Konto anzulegen, klicken Sie bitte „Noch kein Konto? Jetzt registrieren!“.

Auf der Registrierungsseite werden Sie Schritt für Schritt durch den Prozess geleitet. Zunächst müssen Sie Ihre Kontaktdaten und dann Ihre Accountdaten für das Anmelden im System eingeben. Anschließend können Sie bereits hier Ihr Profil und Ihre Einstellungen nach Belieben konfigurieren und die Registrierung abschließen.

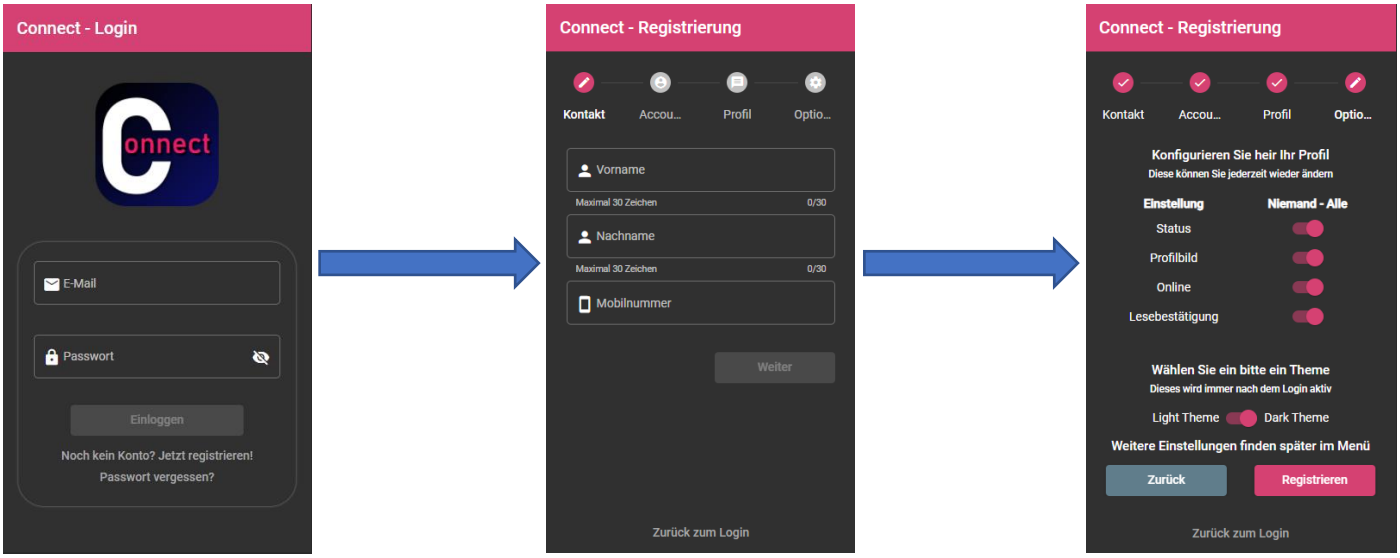


Abbildung 23: Registrierungsvorgang

Im Anschluss an die Registrierung wird Ihnen eine E-Mail zur Bestätigung mit einem Link zugesandt. Öffnen Sie diesen bitte im Browser um Ihre E-Mail-Adresse zu verifizieren. Erst danach können Sie sich einloggen. Sie landen anschließend bei korrekten Anmeldedaten auf der Home-Ansicht. Hier finden Sie Ihre Chats und Gruppen.

Weitere Aktionen wie beispielsweise die Einstellungen finden Sie mit einem Klick auf das Menü-Icon in der linken, oberen Ecke. Hier können Sie sich auch vom System abmelden.

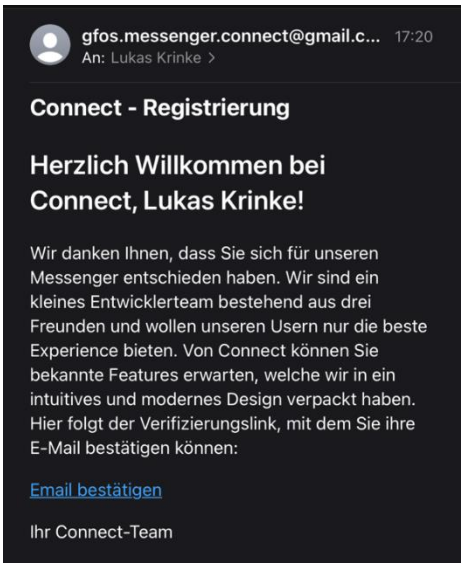


Abbildung 24: Verifizierungsemail

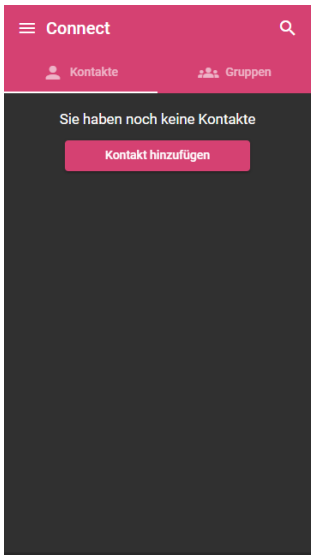


Abbildung 25: Home-Ansicht

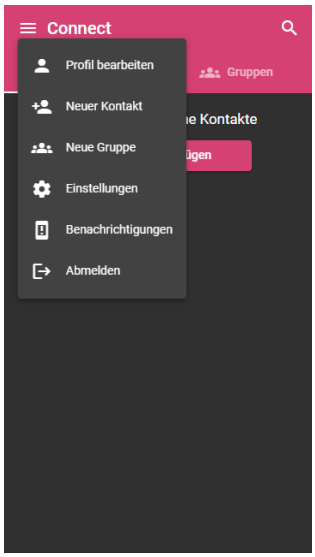


Abbildung 26: Home-Menü

2.4 Mein Konto

Profil bearbeiten

Um ihr Profil, welches von anderen Nutzern eingesehen werden kann, anzupassen, klicken Sie im Menü auf der Home-Seite auf „Profil bearbeiten“. Auf der sich öffnenden Seite können Sie ein neues Profilbild anlegen oder ein altes löschen. Des Weiteren können Sie Ihren Status anpassen. Mit einem Klick auf „Profil aktualisieren“ werden die Änderungen direkt übernommen.

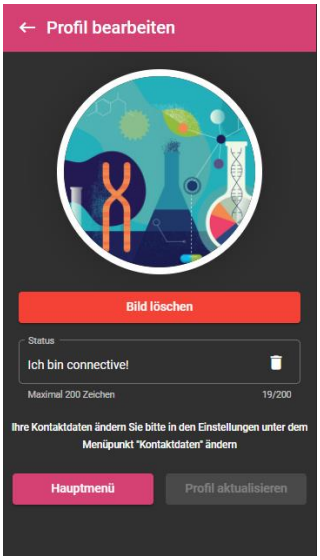


Abbildung 27: Profil bearbeiten

Einstellungen

Über das Menü auf der Home-Seite erreichen Sie ebenfalls die Einstellungen. Hier können Sie Ihr Konto nach Belieben konfigurieren.

Kontaktdaten

Hier können Sie Ihre Kontaktdaten ändern, wie Name, E-Mail oder Passwort. Für letzteres ist die Eingabe des aktuellen Passwortes erforderlich.

Privatsphäre

Hier können Sie Einstellungen bezüglich der Privatsphäre konfigurieren, so wie es bereits beim Registrieren möglich war.

App-Einstellungen

Hier können Sie die 2-Faktor-Authentifizierung für eine erhöhte Sicherheit oder den Datensparmodus für einen geringeren Datenverbrauch aktivieren. Beachten Sie bitte, dass Bilder dann nicht mehr angezeigt werden.

Design

Hier können Sie das Theme der App anpassen. Es stehen Ihnen ein Light- und ein Dark-Theme zur Auswahl.

Freund einladen

Hier können Sie mit einer E-Mail-Adresse einen Freund dazu einladen, ebenfalls Connect zu nutzen.

Konto löschen

Hier können Sie Ihr Konto mit sämtlichen zugehörigen Daten unwiderruflich löschen.

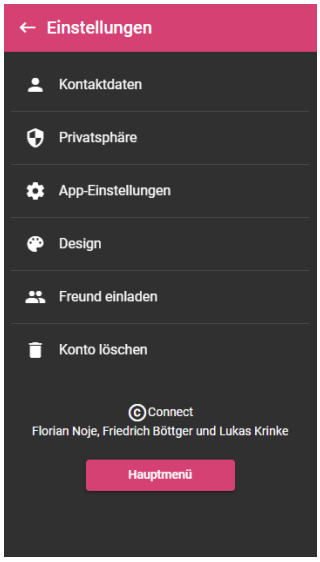


Abbildung 28: Einstellungen

Passwort zurücksetzen

Haben Sie Ihr Passwort vergessen, so können Sie beim Login über die Schaltfläche „Passwort vergessen?“ eine E-Mail mit einem Link zur Festlegung eines neuen Passwortes anfordern. Dieser Link führt Sie auf die nebenstehende Seite. Wir empfehlen Ihnen, sich an unseren Kriterien für ein starkes Passwort zu orientieren.

Beachten Sie bitte, dass der Link nach Anforderung nur 15 Minuten lang gültig ist.

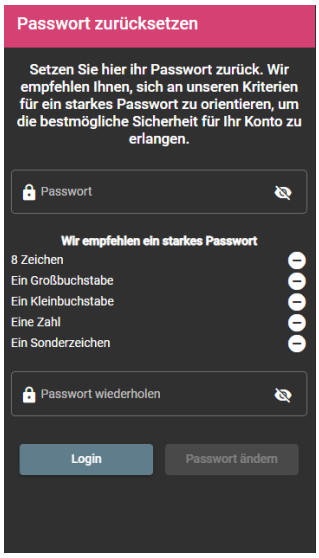


Abbildung 29: Passwort zurücksetzen

System-Benachrichtigungen

Ebenfalls im Menü auf der Home-Seite finden Sie die Benachrichtigungen des Systems. Hier informieren wir Sie über aktuelle Updates. Des Weiteren erhalten Sie Benachrichtigungen, wenn ein Nutzer Sie als Kontakt hinzugefügt oder entfernt hat. Gleiches gilt für das Hinzufügen oder Entfernen zu beziehungsweise aus einer Gruppe. Auch Änderungen bezüglich des Rangs innerhalb einer Gruppe werden Ihnen hier mitgeteilt.

Abbildung 30: Benachrichtigungen

2.5 Kontakte & Gruppen

Home-Ansicht

Auf der Home-Seite finden Sie alle Ihre Kontakte und Gruppen, unterteilt in zwei Reiter. Diese werden nach Ihrer Aktualität geordnet. Alle wichtigen Informationen, wie zum Beispiel der Online-Status oder die Anzahl an ungelesenen Nachrichten werden Ihnen hier angezeigt. Des Weiteren steht Ihnen über das Lupen-Icon oben rechts die Möglichkeit einer Suche zur Verfügung. So können Sie selbst bei einer langen Kontakt- oder Gruppenliste schnell den gewünschten Chat finden.

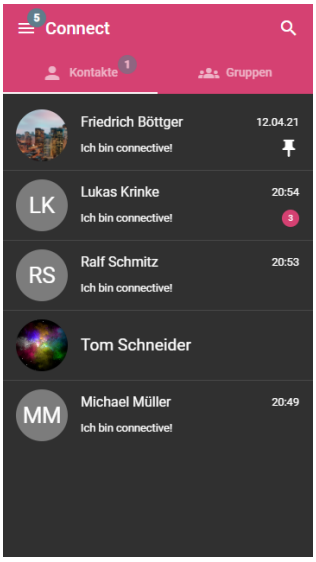


Abbildung 31: Home-Ansicht

Kontakt hinzufügen

Über das Menü-Icon links oben können Sie unter dem Punkt „Neuer Kontakt“ einen Kontakt zu Ihrer Kontaktliste hinzufügen. Dafür müssen Sie lediglich die E-Mail-Adresse der Person kennen, mit der diese sich bei Connect registriert hat.

Kontaktprofil

Um das Profil eines Kontaktes einzusehen, kann man in einem Chat oben rechts in dem Menü auf „Profil anschauen“ klicken. Auf der folgenden Seite werden alle wichtigen Informationen angezeigt. Am unteren Ende der Seite können Sie die Kontaktzuordnung Ihrerseits konfigurieren. Dies umfasst einen Nutzernamen, den Sie einspeichern können und unter dem der Nutzer dann in Ihrer Kontaktliste angezeigt wird. Zudem können Sie den Chat mit einem Kontakt anpinnen, wodurch er in der Kontaktliste immer oben angezeigt wird. Durch das Blockieren wird das gegenseitige Schreiben von Nachrichten verhindert und der Kontakt kann Ihr Profil nicht mehr einsehen. Abschließend können Sie einen Kontakt hier auch für immer löschen.

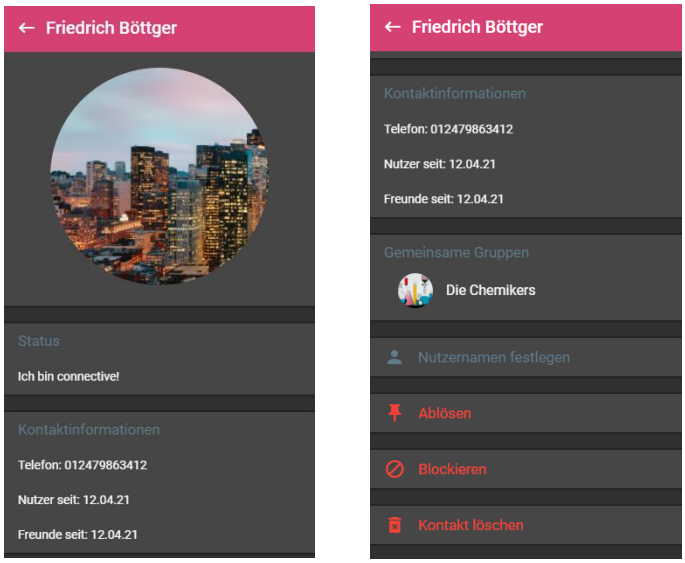


Abbildung 32: Kontakt-Profil

Gruppe erstellen

Über das Menü-Icon links oben können Sie unter dem Punkt „Neue Gruppe“ eine neue Gruppe erstellen. In einem ersten Schritt müssen Sie einen Gruppennamen sowie eine Beschreibung festlegen, optional können Sie auch ein Bild hinzufügen. Anschließend können Sie auf der zweiten Seite die maximale Teilnehmerzahl bestimmen und aus Ihrer Kontaktlist die ersten Teilnehmer auswählen. Später können auch noch Personen über Ihre E-Mail-Adresse hinzugefügt werden, die nicht in Ihrer Kontaktliste sind. Um die Gruppe erfolgreich zu erstellen, muss mindestens ein Teilnehmer hinzugefügt werden.

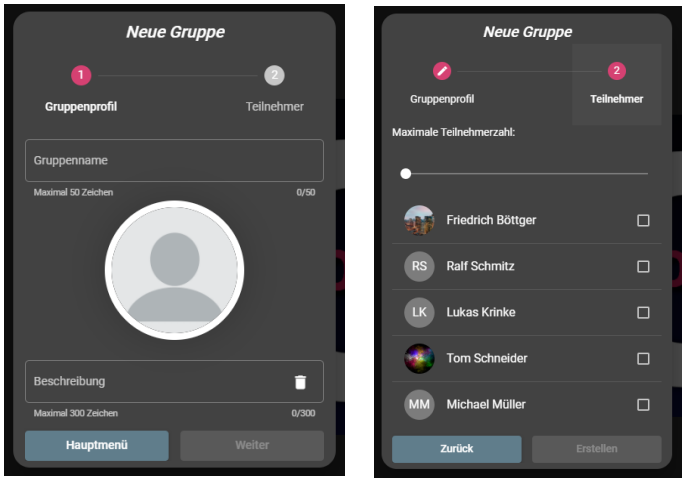


Abbildung 33: Erstellung einer neuen Gruppe

Gruppenprofil

Das Profil einer Gruppe erreichen Sie wie bei einem Kontakt über das Menü oben rechts in der Chat-Ansicht einer Gruppe unter dem Punkt „Gruppe anschauen“. Auch hier finden Sie alle relevanten Informationen zu der Gruppe selbst und zu den Teilnehmern. Als Mitglied haben Sie keine Konfigurationsrechte, Sie können lediglich Ihre Gruppenzuordnung konfigurieren, was das Anpinnen, Stummschalten und Verlassen einer Gruppe umfasst. Als Verwalter können Sie bereits die Einstellungen der Gruppe anpassen, was das Gruppenbild, den Gruppennamen und die Beschreibung beinhaltet. Des Weiteren können Sie weitere Mitglieder zum Verwalter befördern oder bestehende Verwalter degradieren. Die Berechtigung zum Hinzufügen oder Entfernen von Teilnehmer jeglichen Rangs besitzen nur Admins. Sie können Teilnehmer entweder per E-Mail-Adresse oder aus Ihrer Kontaktliste hinzufügen, die maximale Teilnehmeranzahl kann dabei nicht überschritten werden.

Jedes Gruppenmitglied kann jedoch den Teilnehmerdialog öffnen, in dem Informationen zu einem Mitglied angezeigt werden und dieses zu den eigenen Kontakten hinzugefügt werden kann. Je nach eigenem Rang können weitere Aktionen, wie zuvor beschrieben, durchgeführt werden.

Möchten Sie als einziger Admin eine Gruppe verlassen, so muss in einem sich öffnenden Fenster ein neuer Admin festgelegt werden. Verlassen Sie als einziger Teilnehmer die Gruppe, so wird diese unwiderruflich gelöscht.

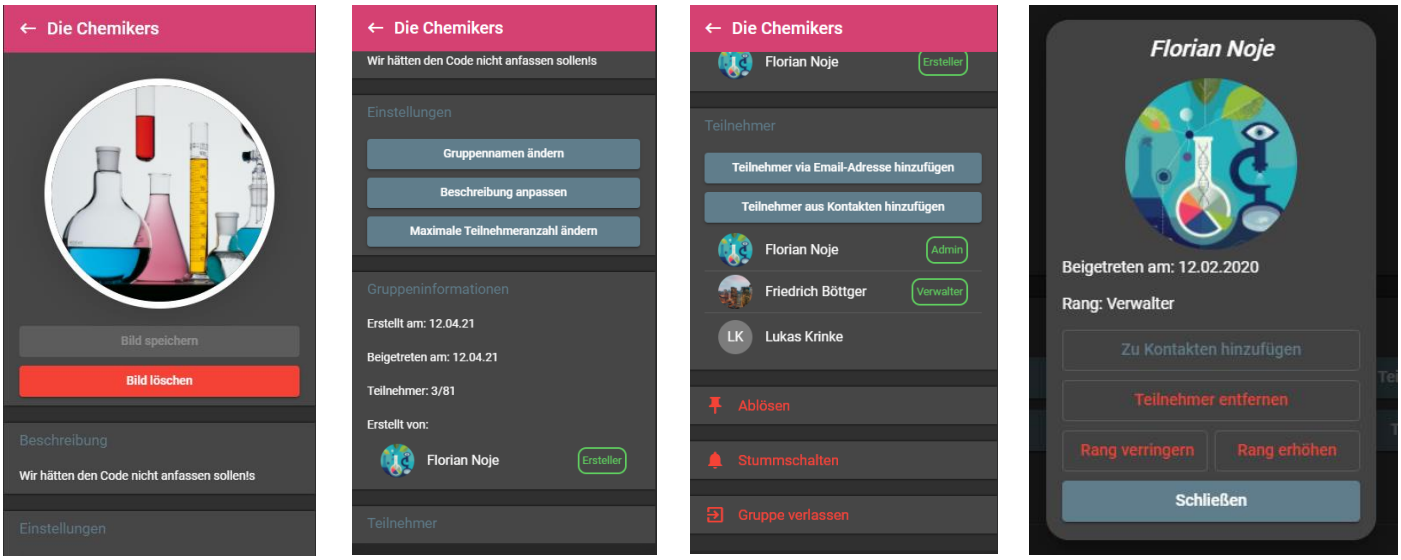


Abbildung 34: Gruppen-Profil

Abbildung 35: Teilnehmer-Info

2.6 Chats

Innerhalb eines Chats können Sie mit einer Person oder einer Gruppe kommunizieren, wobei Ihre Nachrichten auf eine Länge von 1000 Zeichen begrenzt sind. Mit einem Klick auf den Pfeil oder der Bestätigung durch Enter wird Ihre Nachricht an den Kontakt oder alle Personen einer Gruppe versendet.

Nachrichten-Menü

In dem Nachrichten-Menü, welches Sie mit einem Klick auf den Pfeil unten links bei einer Nachricht erreichen, können Sie weitere Aktionen für diese Nachricht durchführen. Handelt es sich um eine von Ihnen versendete Nachricht, so können Sie diese nachträglich wieder löschen. Des Weiteren können Sie auf alle Nachrichten, also Ihre eigenen und auch die von anderen Nutzern, antworten. Klicken Sie dafür auf den Reiter „Antworten“. Unten rechts erscheint eine blaue Anzeige, die Sie darauf aufmerksam macht, dass Ihre folgende Nachricht als Antwort auf eine vorherige versendet wird. Mit einem Klick auf diese Anzeige wird der Vorgang abgebrochen und Sie können eine normale Nachricht verschicken. Der letzte Reiter öffnet einen Informations-Dialog, in dem Sie zu einer Nachricht erfahren, wann der Kontakt beziehungsweise die Mitglieder einer Gruppe diese gelesen haben.

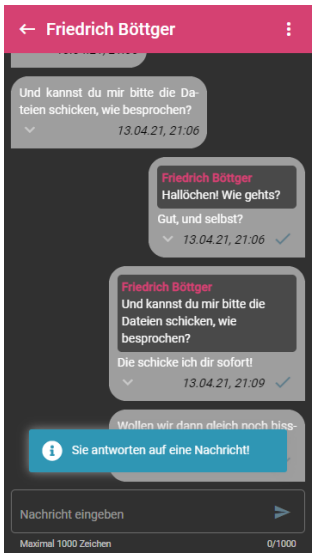
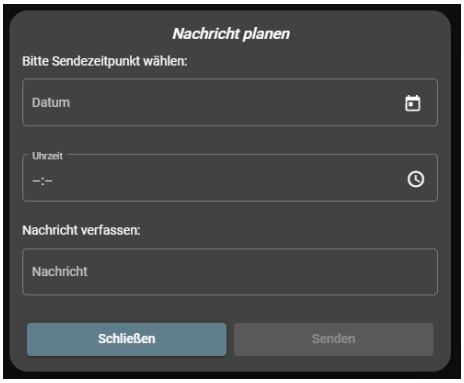


Abbildung 36: Antworten

Chat-Menü

In dem Chat-Menü, welches Sie über die drei Punkte oben rechts am Bildschirmrand öffnen können, stehen Ihnen weitere Aktionen zur Verfügung. Über den Reiter „Nachrichten suchen“ können Sie gezielt nach einer Nachricht suchen. Möchten Sie eine Nachricht im Voraus planen, so können Sie dies über den Punkt „Nachricht planen“ tun. Wählen Sie hierzu das Datum und die Uhrzeit aus, zu der die Nachricht an den Kontakt oder alle Gruppenmitglieder versendet werden soll. Beachten Sie, dass dies nur für die folgenden Tage und nicht den aktuellen möglich ist. Entscheiden Sie sich vor dem Versand dazu, die Nachricht wieder zu löschen, wird die Nachricht gar nicht erst versendet. Als letzte Option bieten wir Ihnen noch das Leeren des Chatverlaufes. Der Kontakt oder die anderen Gruppenmitglieder können diese jedoch weiterhin sehen. Hier können Sie zwischen vielen Zeitangaben wählen, ab welchem Datum Nachrichten nicht mehr angezeigt werden sollen.



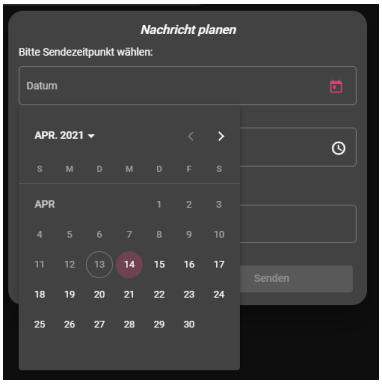


Abbildung 37: Nachrichtenversand planen

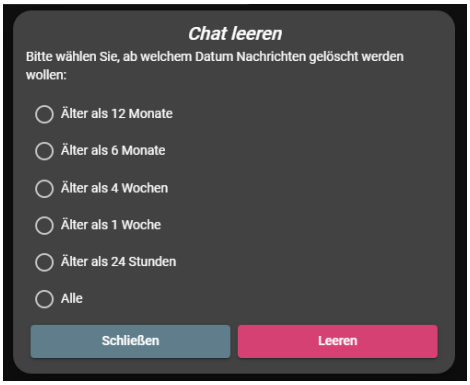


Abbildung 38: Chatverlauf leeren