TECHNISCHE
HOCHSCHULE
LÜBECK

Bachelor-Thesis

Entwicklung einer dezentralen Progressive Web App für Studentenwohnheime

Vorgelegt von: Herrn Benjamin Krause

Fachbereich: Elektrotechnik und Informatik

Studiengang: Informationstechnologie und Design

Erstprüfer/in: Prof. Dr. Kratzke

Ausgabedatum: 02. November 2020
Abgabedatum: 02. Februar 2021

Aufgabenstellung:

Für die Bewohner von Studentenwohnheimen soll eine App entwickelt werden, die diesen im sozialen und organisatorischen Bereich von Studentenwohnheimen pragmatische Interaktionsmöglichkeiten bietet und auf einem dezentralen und selbstorganisierendem Ansatz basiert.

Studentenwohnheime sind häufig Heimat von mehreren hundert Studierenden mit hoher Fluktuation. Häufig existieren jedoch keine Online-Plattformen für Studentenwohnheimsbelange. Meist wird hierzu (selbstorganisiert) auf Facebook-Gruppen ausgewichen, die von einzelnen Bewohnern erstellt werden. Verlassen die Gruppenadministratoren das Wohnheim, verwaisen diese Online-Gruppen häufig und es müssen andere Organisationsformen in unregelmäßigen Abständen durch Bewohner gefunden werden.

Um eine möglichst attraktive und sich selbstorganisierende Plattform für Studentenwohnheime zu bieten, soll zunächst ermittelt werden, welche Funktionen die Bewohner nutzen würden. Solche Funktionen könnten beispielsweise ein Wohnheimflohmarkt, ein Foodshare-Programm, Meldungen an den Hausmeister, Annahme von Paketen, eine WG-Einkaufsliste, Grillplatz-Buchungen, Parties, wichtige Kontaktdaten (Hausmeister, etc.) usw. sein.

Features mit ausreichendem Bedarf sollen implementiert werden. Ferner soll eine geeignete technische Architektur für eine solche App entwickelt werden, die möglichst auf zentrale und zu hostende Serverkomponenten verzichtet und Kommunikationskanäle nutzt von denen bekannt ist, dass Studierende über diese verlässlich erreichbar sind (bspw. Email). Der Ansatz kann als eine Art Social-Network-over-Email-Ansatz bezeichnet werden. Dabei soll Nutzern der App nicht deutlich werden, dass im Hintergrund EMails genutzt werden. Nutzer, die die App nicht nutzen wollen, sollen dennoch in der Lage sein, Nachrichtenströmen rund um Wohnheimsbelange per EMail (lesend) zu folgen.

Aufgrund der Besonderheit des Use Cases sind folgende Randbedingung einzuhalten:

- Die App soll als Progressive Web App (PWA) mit einem geeigneten WebUI-Framework (z.B. Flutter oder vglb.) entwickelt werden.
- Die Wahl der Sprache obliegt dem Bearbeiter der Arbeit, soll aber nachvollziehbar begründet werden.
- Die EMail-Anbindung an die PWA kann mittels IMAP/SMTP-Proxies mit REST-API erfolgen.
- Die Entwicklung dieses Proxies gehört zum Umfang dieser Arbeit (dabei ist nur der Protokollteil von IMAP/SMTP abzubilden, der von der PWA auch tatsächlich benötigt wird).

- Bei technischer Eignung können jedoch ggf. existierende Open Source IMAP/SMTP-REST-API- Proxies genutzt werden.

- In diesem Fall kann auf die Entwicklung eines eigenen Proxies verzichtet werden.

- Die gewählte Lösung soll nachvollziehbar begründet werden.

- Die PWA sowie der Mail-Proxy soll als Container bereitgestellt werden und automatisiert auf der Kubernetes Infrastruktur des myLabs mittles einer Gitlab CI Pipeline ausgebracht werden können. Hierzu kann der Gitlab Service des myLab genutzt werden. Technische Unterstützung beim Aufbau der Pipeline durch den Laboringenieur des myLab ist dabei möglich.

Teilaufgaben:

- Bedarfsanalyse von Funktionen für Studentenwohnheime. Hierbei sind Bewohner des Studentenwohnheims geeignet einzubeziehen

- Analyse denkbarer technischer Lösungsansätze (dezentral, selbstorganisierend).

- Architekturentwurf der Anwendung

- Die Funktionsfähigkeit der App soll mittels Softwaretests geeignet nachgewiesen werden.

- Die Nutzbarkeit der App soll systematisch evaluiert werden. Hierbei sind Bewohner von Studentenwohnheimen geeignet einzubeziehen.

- Dokumentation der oben angegebenen Schritte (Bachelor-Arbeit).

Inhaltsverzeichnis

1 Einleitung

Studentenwohnheime sind für viele Studierende die Heimat während des Studiums. Durch die hohe Fluktuation in den Wohnheimen gibt es aber häufig keine einheitliche Plattform für die soziale und organisatorische Interaktion zwischen den Studierenden. In den meisten Fällen wird auf bekannte Plattformen wie Facebook, WhatsApp oder andere zurückgegriffen, die es ermöglichen, über Gruppen Inhalte und Informationen auszutauschen.
Diese Methoden bringen jedoch mehrere Probleme mit sich.
Oft bedarf es bei Gruppen einer Person, die diese administriert. Ziehen die Administratoren weg und es findet sich kein Nachfolger, brechen solche Strukturen häufig zusammen. Selbst gehostete Lösungen sind seltener anzutreffen, da sie noch mehr Aufwand und Pflege bedürfen.
Zudem sind die Plattformen in ihrem Funktionsumfang beschränkt und können nicht auf spezifische Bedürfnisse von Nutzern eingehen. Häufig werden, um die Funktionalität doch in einer bestimmten Art und Weise nutzen zu können, vorhandene Funktionen entfremdet oder weitere Tools genutzt.
Des Weiteren lehnen einige Bewohner die Nutzung von Plattformen wie Facebook grundsätzlich ab, da diese große Mengen an Nutzerdaten für Werbezwecke analysieren oder an Dritte verkaufen. Unter diesen Umständen ist es schwer, ein flächendeckendes Netzwerk zwischen Bewohnern eines Studentenwohnheims aufzubauen.
Verweigert einer oder mehrere die Nutzung, hat dies Auswirkungen auf alle Bewohner.

Die in der Arbeit entwickelte Anwendung soll möglichst viele der zuvor erwähnten Probleme lösen können. Die Daten, die von den Nutzern beispielsweise durch Chats produziert werden, sollen nicht zentral in einer Datenbank, sondern möglichst dezentral gespeichert werden. Somit fällt die Administration von Hard- und Software größtenteils weg und die Datensicherheit wird erhöht, da die Daten unter der Kontrolle des Nutzers sind.
Um dieses Ziel zu erreichen, gibt es eine Vielzahl an Protokollen. Matrix beispielsweise bietet die Möglichkeit, zwischen verschiedenen Home-Servern (Server, auf denen Nutzer angemeldet sind) zu kommunizieren. Zudem sind Grundfunktionalitäten wie Einzel- und Gruppenchats bereits vorhanden. Jedoch bedarf es bei der Nutzung von Matrix eines eigenen Home-Servers oder einer Registrierung bei einem bereits bestehenden.

Das bessere Protokoll ist jedoch die E-Mail. Sie unterstützt ebenso wie Matrix einige Grundfunktionalitäten, auf denen aufgebaut werden kann. Ebenso werden Daten bei unterschiedlichen E-Mail-Providern gespeichert. Die ausschlaggebenden Gründe sind jedoch die hohe flächendeckende Nutzung und die Zuverlässigkeit der Infrastruktur, da jeder Bewohner eines Studentenwohnheims bereits, privat oder durch die Hochschule, mindestens eine oder sogar mehrere E-Mail-Adressen besitzt. Der Nutzer muss sich lediglich mit seinem vorhandenen Postfach anmelden, um die Plattform zu nutzen.

Die Protokolle hinter den E-Mails sind bereits lange verfügbar und ausgereift. Da E-Mails zuverlässig zwischen Nutzern versendet werden können, ist der Datenaustausch der Plattform somit vorhanden und erprobt.

In den nächsten beiden Kapiteln werden Anforderungen an diese Arbeit erhoben, beschrieben und gewichtet. Dazu wird analysiert, welche Funktionen von Bewohnern benötigt und daher in die PWA mit eingebunden werden sollten.
Im vierten Kapitel werden die Funktionsweise der Plattform sowie Eigenheiten der Protokolle und benötigte Komponenten näher erläutert, während im fünften Kapitel auf Details sowie Probleme und Techniken der Implementierung eingegangen wird.
Ansätze für die Sicherheit der Anwendung werden im sechsten Kapitel entwickelt. Im siebten Kapitel wird überprüft ob die Konzepte und Funktionsweise der PWA dem Nutzer durch die Implementierung des Prototypen vermittelt werden kann.
Die zuvor in Kapitel 2 aufgestellten Anforderungen werden im achten Kapitel validiert und überprüft. Ob es möglich und technisch lohnend ist, eine solche Anwendung über E-Mails abzubilden, wird in Kapitel 9 erläutert.

2 Anforderungen an die Arbeit

Die in Tabelle 2.1 aufgelisteten Anforderungen wurden aus der Aufgabenstellung und den allgemeinen Themenfeldern der Softwareentwicklung erhoben.

Nummer	**Titel**	**Beschreibung**
1	Architekturentwurf	Um zu vermitteln, wie die Anwendung arbeiten soll, muss ein Architekturentwurf angefertigt werden. Aus diesem soll hervorgehen, wie die einzelnen Komponenten miteinander kommunizieren.
2	Bedarfsanalyse für Funktionen	Um den Funktionsumfang der Progressive-Web-App (*PWA*) zu bestimmen, muss eine Bedarfsanalyse durchgeführt werden.
3	Einsatz der E-Mail	Um darzulegen, wie die E-Mail als Informationsmittel eingesetzt wird, muss beschrieben werden, was für Daten mit anderen ausgetauscht und wie diese strukturiert werden.
4	Entwicklung eines Prototypen	Um zu beweisen, dass die erarbeiteten Architekturentwürfe und Konzepte funktionieren, muss ein Prototyp, bestehend aus PWA und E-Mail-Proxy, entwickelt werden.
5	Evaluation der Anwendung	Der Prototyp der PWA soll systematisch evaluiert werden, hierfür sind die Bewohner des Wohnheims sinnvoll einzubeziehen. Wird keine Evaluation durchgeführt, könnte es zu Unverständnissen bei der Benutzung der Oberfläche oder der Technik dahinter führen.
6	Sicherheitskonzept	Es soll ein Konzept für die Sicherheit der Anwendung (beispielsweise Verschlüsselung von E-Mails) erstellt werden.

Tabelle 2.1: Anforderungstabelle

3 Umfrage zum Funktionsumfang

Welche Funktionen in der PWA von den Bewohnern der Wohnheime benötigt werden und welche nicht, wurde mittels einer Umfrage untersucht. Hierzu wurden zunächst Vorschläge aus der Aufgabenstellung entwickelt oder bei Kommilitonen erfragt. Anschließend wurde die dadurch gesammelte Liste von Anwendungen, mit der Möglichkeit weitere Ideen zu ergänzen, den Studierenden der TH-Lübeck zur Bewertung vorgelegt.

- Einfacher Chat
- Private Gruppenchats
- Öffentliche Gruppenchats
- Organisationsmöglichkeit für einen Wohnheimflohmarkt
- Foodsharing-Programm
- Grillplatzbuchung
- Veranstaltungsinformationen
- News-Forum
- Kontaktdaten (bspw. des Hausmeisters)
- Meldungen an den Hausmeister

Die Befragten konnten die Funktionen mit den Auswahlmöglichkeiten aus der Tabelle 3.1 bewerten. Den Auswahlmöglichkeiten sind jeweils Punkte zugeordnet worden, damit ein Durchschnitt zur späteren Auswertung der Umfrage berechnet werden kann. Die Punkte gehen von 1 bis 4, wobei eins das beste und vier das schlechteste Ergebnis darstellt. Die Möglichkeit "*Ist mir egal*" wird nicht in das Endergebnis mit eingerechnet, sondern als Enthaltung gewertet.

Auswahlmöglichkeit	**Punkte**
Das brauche ich!	1
Könnte nützlich sein	2
Wahrscheinlich brauche ich es nicht	3
Werde ich definitiv nicht nutzen!	4
Ist mir egal	-

Tabelle 3.1: Gewichtung der Auswahlmöglichkeiten

3.1 Auswertung der Funktionen

Aus den Daten der Umfrage konnte eine Tabelle (siehe 3.2) erstellt werden, aus der sich die Durchschnittsbewertung einer jeden Funktion ablesen lässt. Die Funktionen sind nach dem Durchschnitt geordnet, sodass die wichtigste Funktion oben und die unwichtigste unten steht.

Funktionen	Das brauche ich!	Könnte nützlich sein	Wahrscheinlich brauche ich es nicht	Werde ich definitiv nicht nutzen!	Ist mir egal	Durchschnitt
Meldungen an den Hausmeister	44	28	5	2	4	1,56
News-Forum	40	36	5	1	1	1,60
Kontaktdaten (bspw. des Hausmeisters)	42	28	9	1	3	1,61
Veranstaltungsinformationen (bspw. Partys)	37	31	7	3	5	1,69
Foodsharing-Programm	30	31	8	7	7	1,89
Grillplatzbuchung	29	34	11	5	4	1,90
Organisationsmöglichkeiten für einen Wohnheimflohmarkt	26	31	13	6	7	1,99
Einfacher Chat	13	50	13	2	5	2,05
Private Gruppenchats	14	40	17	5	7	2,17
Öffentliche Gruppenchats	15	38	21	5	4	2,20

Tabelle 3.2: Auswertung der Funktionen

Einige Befragte haben weitere Funktionsvorschläge eingereicht. Diese sind in der folgenden Liste aufgeführt. Die Vorschläge wurden im Voraus gefiltert, da manche Vorschläge nicht zu der Art von Anwendung passen oder primär nicht mit dem Thema übereinstimmen.

- Einkaufslisten, Putzpläne
- Kleinanzeigen-Funktionen / Tauschbörse / Objekt-Sharing (Drucker, Bohrmaschinen) / Wohnheimflohmarkt
- Paket-Such-Zentrale (Wer hat mein Paket angenommen?)

- Interessen-Gruppenchats (Interessen wie bspw. Sport, Kochen, Neulinge in Lübeck, Lerngruppen ...)
- Interaktion mit dem Heimrat des Studentendorfs in Lübeck

3.2 Beschreibung der Funktionen

Um einen Überblick zu geben, wie die Funktionen in der Theorie arbeiten sollen, werden sie im Folgenden einzeln erklärt. Besteht eine Überschneidung oder Gleichheit der Funktionalität, werden die Funktionen gemeinsam beschrieben.

Chatfunktionen (Gruppen, 1zu1, öffentlich, privat)

Ein Chat dient in erster Linie zur Textkommunikation zwischen mindestens zwei oder vielen Bewohnern. Dabei gibt es mindestens einen Administrator, welcher in der Lage ist, den Chat zu verwalten. Unter die Verwaltung fallen das Einladen und Entfernen von Mitgliedern sowie das Ernennen oder Degradieren von Administratoren. Ein Mitglied des Chats hat selbst immer die Möglichkeit, sich aus dem Chat zu entfernen, auch wenn es kein Administrator ist.
Chats können privat oder öffentlich gestellt werden. Der Unterschied zwischen den beiden liegt darin, dass öffentliche Chats mit Einladung oder ohne betreten werden können. Hingegen sind private Chats nur mittels einer Einladung betretbar.
Zusätzlich soll die Einstellung, dass nur Administratoren Nachrichten versenden können, möglich sein.
Interessen-Gruppenchats unterscheiden sich nicht von privaten oder öffentlichen Chats, denn sie setzen nur einen expliziten Fokus auf ihre Gesprächsthemen.

Die hier beschriebenen Grundfunktionen eines Chats mit seinen beliebigen Einstellungen bilden die Grundlage, der im Weiteren beschriebenen Funktionen.

Einkaufslisten, ToDo-Listen

Bei Einkaufslisten und ToDo-Listen handelt es sich im Allgemeinen um Listen zum Abhaken. Diese können innerhalb eines Chats versendet und von allen Mitglieder bearbeitet werden.

Putzpläne

Ein Putzplan könnte eine Zuordnung von Namen zu Aktionen sein.
Die Zuordnung könnte in einem Zeitraum X gewechselt werden. Da es sich um ein festes Schema handelt, wäre die einfachste Möglichkeit, einfach zu rotieren.
Mit der folgenden Tabelle wird die Rotation noch einmal dargestellt.

Name	**1. Woche**	**2. Woche**	**3. Woche**	**4. Woche**
Küche	Liesa	My	Freddy	Liesa
Badezimmer	Freddy	Liesa	My	Freddy
Flur	My	Freddy	Liesa	My

Tabelle 3.3: Putzplan Rotationsbeispiel

Meldungen an den Hausmeister

Mit Meldungen an den Hausmeister soll es den Bewohnern des Wohnheims ermöglicht werden, schnellstmöglich Probleme in der Wohnung zu melden. Ein Bewohner kann dabei aus einer Liste von Problemen auswählen oder in einem Freitext seine Probleme schildern. Im Hintergrund wird eine E-Mail an den Hausmeister oder an anderes Personal übermittelt.

News-Forum und Veranstaltungsinformationen

News- und Veranstaltungsinformationen sollen die Bewohner über die neusten Dinge in einem Wohnheim informieren.
Die Informationen können über jeden Chat verschickt und zusätzlich in einem Chat-Verlauf hervorgehoben werden.
Durch die Zugriffs- und Sendeeinstellungen eines Chats können so öffentliche oder private News Feeds oder theoretisch sogar simple Blogs betrieben werden.

Kontaktdaten

Die Bewohner sollen die wichtigsten Kontaktdaten immer griffbereit haben, sei es nun für das Wohnheim oder auch die Hochschule oder Universität. Es soll ein Kontaktbuch geführt werden, in dem diese Kontakte aufgeführt werden. Aufgelistete Informationen können unter anderem Telefonnummern, Büronummern, Adressen oder auch Öffnungszeiten sein.

Ein Foodsharing-Programm, Kleinanzeigen, Tauschbörsen, Objekt-Sharing, Grillplatzbuchung, Wohnheimflohmarkt, Paket-Such-Zentrale

Bei den in der Überschrift aufgeführten Funktionen handelt es sich heruntergebrochen immer um Geben und Nehmen. Wobei es sich nicht immer um ein unentgeltliches Geben handeln muss, beispielsweise könnte es sich auch um den Verkauf von Gegenständen handeln.

Sie funktionieren alle auf dieselbe Art und Weise. Jemand erstellt ein Inserat (etwas zum Tauschen, Verleihen, Verschenken) und teilt dieses mit allen in einem Chat. Dabei können ein Titel und eine Beschreibung, aber auch eine Gegenleistung mit übertragen werden.
Eine Paket-Such-Zentrale ist ebenfalls mit den bereits beschriebenen Mitteln abbildbar. Jemand, der sein Paket sucht, veröffentlicht ein Inserat mit den Angaben zu seinem vermissten Paket.
Zudem könnten Paketannahmen an andere Personen übertragen werden. Hierfür könnte wieder ein Inserat veröffentlicht werden, in welchem Datum und ungefähre Zeit der Lieferung stehen. Ein weiterer Bewohner könnte sich daraufhin melden und gemeinsam mit der inserierenden Person die Details abklären.

4 Architektur

In diesem Kapitel werden die einzelnen Softwarekomponenten sowie die Schnittstellen zwischen diesen Systemen näher erläutert.
Die Funktionsweise und der grobe Aufbau des Systems werden in Kapitel 4.1 beschrieben. In Kapitel 4.2 wird genauer auf die Funktionsweise des E-Mail-Proxys und in 4.3 auf die der Progressive-Web-App eingegangen.
Nach der Erläuterung der Komponenten werden in Kapitel 4.4 Vor- und Nachteile der Architektur aufgezeigt.

4.1 Funktionsweise

Wie aus der Aufgabenstellung hervorgeht, soll die Anwendung möglichst dezentral arbeiten. Im Falle der Anwendung bedeutet dezentral, dass E-Mails nicht in einer selbst gehosteten Datenbank verwaltet werden, sondern auf vielen Providern verteilt sind. Zusätzlich zu der Speicherung bei den Providern werden die Informationen, welche aus den E-Mails generiert werden, auf dem Gerät des Nutzers gespeichert. Theoretisch wäre es somit möglich, nach dem Abrufen der Provider die Daten dort zu löschen und nur die lokal gespeicherten Daten zu behalten.

Um E-Mails in der Anwendung nutzen zu können, werden die folgenden drei Komponenten benötigt.

1. **E-Mail-Provider** - Sie dienen in erster Linie einem Nutzer dazu, E-Mails mit anderen Nutzern auszutauschen. Dabei ist egal, ob der Sender einen anderen Provider als die Empfänger nutzt, denn die E-Mails werden immer an den jeweiligen Provider gesendet.

   Die Provider werden standardmäßig per *IMAP* (Internet Message Access Protocol) und *SMTP* (Simple Mail Transfer Protocol) angesprochen. Da diese beiden Protokolle standardisiert sind, ist es möglich, aus ein und derselben Anwendung unterschiedliche Provider anzusprechen.

   Was auf der Seite der Provider mit einer E-Mail passiert, wird in dieser Arbeit nicht behandelt. Es wird davon ausgegangen, dass versendete E-Mails mit korrekten Daten zuverlässig bei den Empfängern ankommen.

2. **E-Mail-Proxy** - Um mit den E-Mail-Providern kommunizieren zu können, wird ein E-Mail-Proxy benötigt, der stellvertretend für den Nutzer mit den Providern kommuniziert. Dies ist zwingend nötig, da moderne Webbrowser aktuell nicht in der Lage sind, die E-Mail-Protokolle *SMPT* und *IMAP* zu nutzen [15] [14].

   Der E-Mail-Proxy soll per REST-Schnittstelle angesprochen werden.

3. **Progressive-Web-App** - Die letzte Komponente ist die Web-App, über welche der Nutzer mit anderen Nutzern kommuniziert. Die PWA kann über den E-Mail-Proxy auf das E-Mail-Postfach des Nutzers zugreifen und dort für sie relevante E-Mails suchen. Die E-Mails werden von der PWA verarbeitet und persistiert. Ebenfalls können neue E-Mails erstellt und versendet werden.

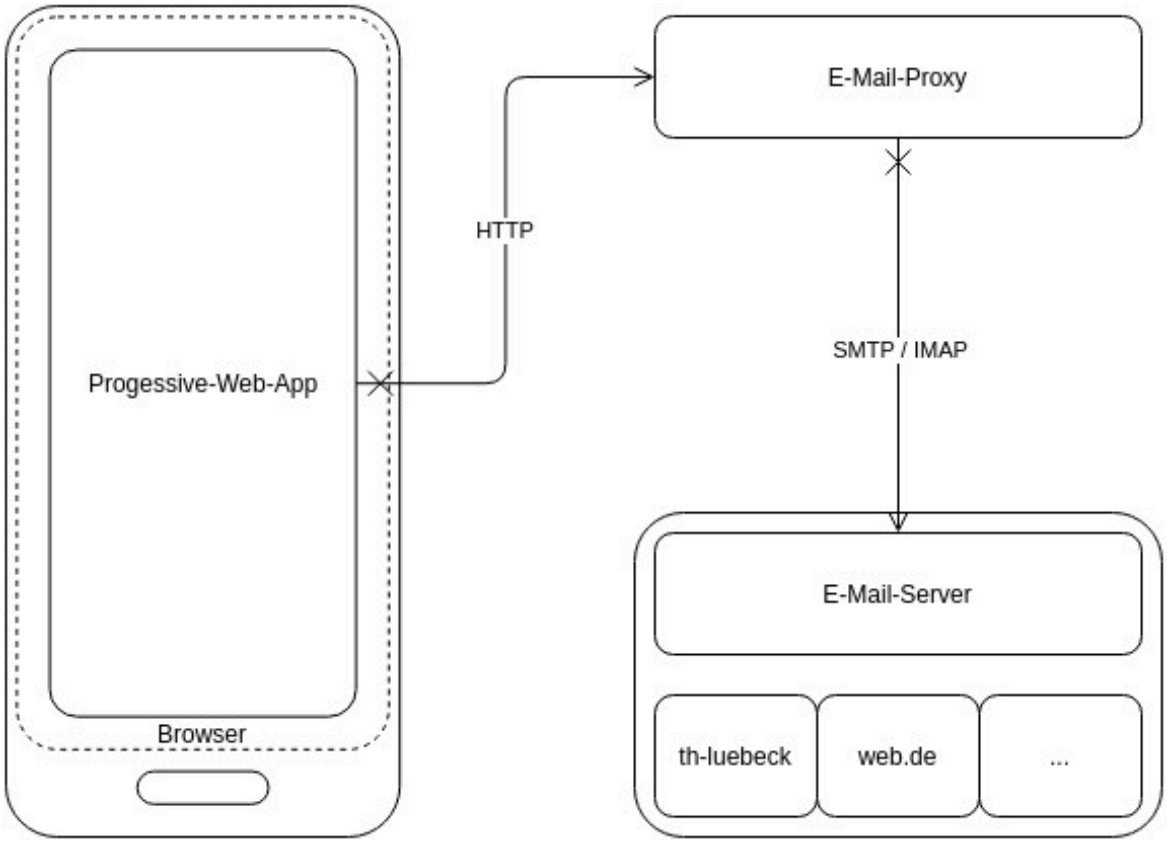

Abbildung 4.1: Der Komponenten-Überblick der Architektur

4.2 E-Mail-Proxy

Wie bereits erwähnt, nutzt die PWA den E-Mail-Proxy als Tool, um mit den E-Mail-Providern zu kommunizieren.
Die PWA ist in der Lage, E-Mails zu versenden oder abzurufen. Um dies zu ermöglichen, werden die Verbindungsdaten für den E-Mail-Provider, die Zugangsdaten des Nutzers und zusätzliche Parameter passend zu der gewünschten Aktion benötigt.
Die Verbindungsdaten beinhalten, welche URL, welcher Port und welche Verschlüsselung für die Protokolle *SMTP* und *IMAP* genutzt werden sollen. Diese Informationen können im E-Mail-Proxy hinterlegt werden, um allen Nutzern die Möglichkeit zu bieten, den passenden Provider zu finden. Zusätzlich wird bei jedem Request Bandbreite gespart. Dieser Mechanismus wird in Kapitel 5.2.1 näher beschrieben.

Die Zugangsdaten des Nutzers bestehen aus der E-Mail-Adresse und dem Passwort und je nach Provider zusätzlich aus einem separaten Nutzernamen. Diese Informationen werden nicht zwischengespeichert, weshalb sie bei jeder neuen Anfrage mitgesendet werden müssen.

Das *SMTP* dient dazu, E-Mails zu versenden. Dazu wird nebst den Zugangs- und Verbindungsdaten auch die E-Mail selbst, bestehend aus Kopfzeile (Header) und Körper (Body) von der PWA übertragen. Die E-Mail wird im Vorhinein auf fehlende Daten oder syntaktische Fehler überprüft. Inhaltliche Fehler, wie eine nicht existierende E-Mail-Adresse, werden nicht überprüft. Ist die E-Mail syntaktisch korrekt, wird diese versendet und eine Bestätigung an die PWA zurück gesendet. Treten Fehler auf, werden diese mit einer Fehlermeldung zurück gesendet.

Das *IMAP* wird verwendet, um mit dem Postfach des eingeloggten Nutzers zu interagieren. Über diese Verbindung ist es möglich, E-Mails zu suchen und zu verwalten. Die Verwaltung besteht darin, E-Mails zu löschen oder auch in einen anderen Posteingang oder Ordner zu verschieben.
E-Mails können mittels eines Suchparameters spezifisch gesucht werden. Dieser Vorgang wird in Kapitel 5.2.2 näher beschrieben.
Bei einer Art Social-Network-over-E-Mail ist zu erwarten, dass viele E-Mails versendet werden. Das führt dazu, dass der Standardposteingang, in dem die meisten Provider eingehende E-Mails ablegen, schnell gefüllt ist. Um ein aufgeräumtes Postfach zu hinterlassen, sollen E-Mails, die für die PWA bestimmt sind, in einen separaten Ordner abgelegt werden, welcher durch die PWA verwaltet wird.

4.3 Progressive-Web-App

Die Progressive-Web-App ist das wesentliche Programm für den Nutzer und besteht aus zwei Komponenten.
Die erste Komponente ist der Main-Thread, der die grafische Oberfläche beinhaltet. Als zweite Komponente übernimmt ein Web-Worker die Businesslogik, d.h. die Verarbeitung

eingehender E-Mails und Ablegen im lokalen Speicher des Gerätes sowie das Erstellen neuer E-Mails und Versenden über den E-Mail-Proxy. Diese beiden Komponenten können miteinander über Events kommunizieren und somit Daten austauschen.

Durch die Aufteilung soll eine stabile grafische Oberfläche gewährleistet werden, da durch das Empfangen und Verschicken von E-Mails und durch das Speichern von Daten Aussetzer in der grafischen Oberfläche vorkommen können.
Ein größerer Vorteil lässt sich zudem mit der Auslagerung auf einen Service-Worker, statt nur auf einen simplen Web-Worker erreichen.
Der Unterschied zwischen Beiden liegt darin, dass der Service-Worker auch ohne den Main-Thread arbeitet und Daten im Hintergrund abrufen und verarbeiten kann. Ebenfalls ist es möglich, die PWA über einen Service-Worker offline zur Verfügung zu stellen.

In den folgenden beiden Abschnitten wird einzeln auf die grafische Oberfläche und auf die Businesslogik eingegangen.

4.3.1 Main-Thread und Grafische Oberfläche

Wie aus der Aufgabenstellung hervorgeht, soll die grafische Oberfläche der Progressive-Web-App mittelst eines WebUI-Frameworks umgesetzt werden. Um eine Auswahl zu treffen, wurde eine kleine Nutzwertanalyse durchgeführt, die anhand von selbst bestimmten Kriterien zu einer Entscheidung führen soll.
Für eine Vorauswahl der möglichen Frameworks wurde das *Real World-Repository* [7] auf Github genutzt. In diesem können Entwickler Demo-Applikationen für Frontend-, Backend- und App-Frameworks verlinken, welche jeweils festgelegten Spezifikationen folgen. Diese Demos sollen Entwicklern eine Übersicht vermitteln und gleichzeitig Beispiele bereitstellen.
Aufgrund der großen Menge an UI-Frameworks wurde sich auf die drei größten JavaScript Frameworks und Flutter beschränkt. Nach dem *Real World-Repository* sind die drei größten JavaScript Frameworks aktuell React, Angular und Vue.

Die Kriterien in der folgenden Tabelle werden in die Bewertung aufgenommen, zusätzlich wird eine Gewichtung für jedes Kriterium mit aufgeführt. Die Kriterien und die Gewichtung wurden vom Ersteller auf der Grundlage von persönlicher Erfahrung festgelegt.

Kriterium	Beschreibung	Gewichtung	Kürzel
Funktionsumfang	Welche Funktionen sind standardmäßig dabei oder werden mit angeboten?	30%	F
Online-Hilfestellungen	Wie ist die Auswahl an Hilfestellungen (Videos, Beiträge, Tutorials, Framework-Dokumentation)?	20 %	OH
Einstieg in das Framework	Wie einfach ist der Einstieg für einen Entwickler?	15%	E
Persönliche Präferenz	Wie sind der erste Eindruck, die Syntax, die generelle Spracheinstellung oder auch die Anwenderfreundlichkeit?	35%	PP
		100%	

Tabelle 4.1: Kriterien für die Nutzwertanalyse der WebUI-Frameworks mit Gewichtung

Kriterien wie beispielsweise Downloadzahlen des Frameworks, Größe in kompiliertem bzw. verpacktem Zustand oder auch Geschwindigkeit wurden nicht in der Liste von Kriterien berücksichtigt. Grundsätzlich sind diese Kriterien wichtig, jedoch sind sie für diese Entwicklung nicht von Belang. Endgeräte verfügen mittlerweile über eine hohe Leistungs- und Speicherkapazität, weshalb die meisten Anwendungen diese Kapazitäten selten ausreizen.

In der Tabelle 4.2 werden Punkte von 1 bis 10 für die in Tabelle 4.1 festgelegten Kriterien verteilt. Dabei werden jeweils erst die Punkte ohne Gewichtung und in der folgenden Zeile die Punkte mit Gewichtung aufgeführt.

WebUI-Framework	F	OH	E	PP	Gesamt
Angular	9	8	7	5	29
Angular mit Gewichtung	2,7	1,6	0,75	2	7,05
Flutter	9	8	8	3	29
Flutter mit Gewichtung	2,7	1,6	1,2	1,2	6,7
React	6	8	7	7	30
React mit Gewichtung	1,8	1,6	1,05	2,8	7,25
Vue	8	8	8	10	34
Vue mit Gewichtung	2,4	1,6	1,2	4	9,2

Tabelle 4.2: Punkteverteilung der Nutzwertanalyse des WebUI-Frameworks

Auf Basis der Tabelle 4.2 fällt die Entscheidung auf das WebUI-Framework Vue.

Aufbauend auf Vue wird die Architektur für die grafische Oberfläche erstellt. Wie in React, Angular und Flutter ist es auch in Vue möglich, ein State-Management-System zu nutzen. Ein State-Management-System erleichtert die Verwaltung von Daten zur Laufzeit des Programms. Verwendet wird daher VueX, eine spezielle Implementierung eines State-Management-Systems für Vue, die auf dem Flux-Pattern [5] aufbaut, das von Facebook entwickelt wurde.
Das Flux-Pattern regelt den Fluss von Daten innerhalb einer Anwendung. Die Daten fließen immer in einer bestimmten Reihenfolge von Komponente zu Komponente. Um diesen Kreislauf an Informationen zu ermöglichen, verwendet Flux vier Bausteine. In Abbildung 4.2 ist der Kreislauf zu sehen.

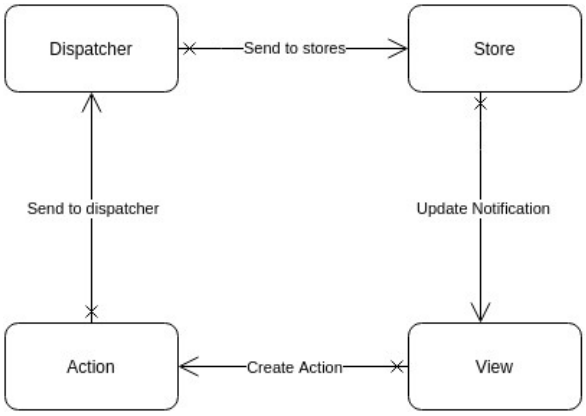

Abbildung 4.2: Datenfluss nach dem Flux-Schema

- **View** - Die *View* ist für das Darstellen der grafischen Oberfläche zuständig. Sie beinhaltet ebenfalls Controller um bspw. auf Click- oder Input-Events zu reagieren.
- **Action** - Eine *Action* ist die Beschreibung eine Aktion, die ausgeführt werden soll, beispielsweise "Lege einen neuen Nutzer an". *Actions* können u. a. durch das Drücken eines Knopfes in der *View* erstellt werden.
- **Dispatcher** - Erstellte *Actions* werden an den *Dispatcher* gegeben, der diese an alle *Stores* weiterleitet.
- **Store** - Ein *Store* enthält Daten, welche durch *Actions* verändert werden können. Es kann unterschiedliche *Stores* für unterschiedliche Kontexte geben, beispielsweise hält ein *Store* nur die Daten über den Nutzer, ein anderer die über seine Chats. Da alle *Stores* die *Actions* erhalten, muss jeder *Store* selbst entscheiden, ob die jeweilige *Action* für ihn relevant ist.

  Wurden Daten in einem *Store* verändert, sendet dieser ein Update-Event an die *View*, woraufhin sich diese bei Bedarf aktualisiert.

Die Funktionsweise von VueX unterscheidet sich nicht grundlegend von Flux, denn die Daten fließen auch hier in einem Kreis von Komponente zu Komponente, wie in Abbildung 4.3 dargestellt.

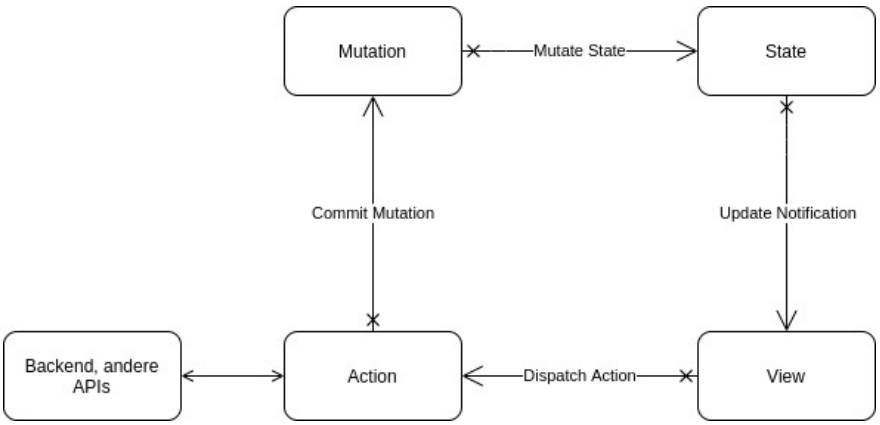

Abbildung 4.3: Datenfluss in VueX

Eine Unterschied ist, dass VueX nur einen *Store* besitzt, in welchem der aktuelle *State* der Anwendung gespeichert wird. Alle Informationen werden so an einem Ort (einer *single source of trust*) hinterlegt [17]. Aus diesem Grund wird auch kein *Dispatcher* benötigt, welcher alle *Stores* kennt.
Ein weiterer Unterschied ist die Aufteilung von *Actions* und *Mutations*. *Actions* sind nach wie vor dazu da, Aktionen auszuführen und Daten zu verarbeiten, jedoch verändern sie selbst keine Daten im *State*. Für die Veränderung von Daten im *State* sind die *Mutations* verantwortlich. Sie übernehmen die Aufgabe, eingehende Daten im *State* zu überschreiben und können aus *Actions* aufgerufen werden.
Bei einer wachsenden Anwendung ist es gut möglich, dass der *Store* aufgeblasen wird. Um dagegen vorzugehen, ist es möglich, den *Store* in mehrere Module mit eigenen *Actions*, eigenen *Mutations* und einem eigenen *State* aufzuteilen [17].

4.3.2 Web-Worker und Businesslogik

Der Web-Worker bildet die Businesslogik innerhalb der PWA. Er läuft parallel zum Main-Thread und kommuniziert mit diesem über Events, welche zusätzliche Daten enthalten können.
Der Web-Worker ist, wie bereits erwähnt, dafür zuständig, eingehende und ausgehende E-Mails zu verarbeiten. Über den E-Mail-Proxy eingehende E-Mails werden in *Messages* umgewandelt. *Messages* beziehen sich immer auf eine E-Mail, enthalten jedoch nur die benötigten Informationen aus dem E-Mail-Header und dem E-Mail-Body. Unwichtige Informationen werden somit heraus gefiltert und gelangen nicht in die Datenbank.

Jede E-Mail und jede *Message* besitzen eine Zugehörigkeit zu einem Chat, diese wird über die ID des Chats übermittelt. Anhand dieser ID werden sie den Chats zugeordnet und anschließend in der Datenbank des Webbrowsers gespeichert. Existiert kein Chat für die E-Mail, wird diese ignoriert. Die einzige Ausnahme ist eine Einladung (*InviteMessage*) zu einem Chat. Diese E-Mail enthält Informationen wie die aktuellen Mitglieder, Administratoren oder Zugriffsrechte des Chats und kann ohne vorhandenen Chat verarbeitet werden.

Werden andere Nachrichten an einen Chat übergeben, überprüft dieser selbständig, ob sie weiter verarbeitet werden können. Ist der Versender der E-Mail kein Mitglied des Chats, wird die Nachricht ignoriert, genauso wenn er keine Berechtigung für die in der E-Mail beschriebene Aktion hat. In einem privaten Chat dürfen nur Administratoren neue Personen dem Chat hinzufügen. Dies ist ein Sicherheits-Mechanismus, um zu verhindern, dass E-Mails in das System gelangen, die nicht über die PWA verschickt werden und somit frei manipuliert werden können.
Sind alle E-Mails verarbeitet, wird ein Event mit den neuen Daten des Chats an den Main-Thread geschickt.
Um einen kontinuierlichen Datenfluss zu gewährleisten, werden in einem festen Intervall neue E-Mails, falls vorhanden, vom E-Mail-Proxy bezogen und verarbeitet.

Wie zuvor beschrieben, ist der Web-Worker zusätzlich dafür verantwortlich, neue E-Mails zu verschicken. Bekommt der Web-Worker von dem Main-Thread ein Event mit den benötigten Informationen, werden diese zunächst überprüft und dann zu einer E-Mail verarbeitet. Für diesen Vorgang werden lediglich ein Chat und eine *Message* benötigt, denn aus einem Chat und einer *Message* können E-Mails generiert werden.
Die erstellte E-Mail wird nun über den E-Mail-Proxy verschickt. Fehler, die bei der Versendung der E-Mail auftreten, werden an den Main-Thread gesendet.
Alle erfolgreich verarbeiteten Nachrichten werden in der Datenbank des Browsers abgelegt, um diese zu persistieren. Die Veränderungen werden, ebenso wie die Fehler, an den Main-Thread und somit an die grafische Oberfläche übermittelt.

4.4 Vorteile und Nachteile der Architektur

4.4.1 Vorteile

Vorteile durch Vue und VueX Durch die fluxartige Architektur wird die Datenarbeit deutlich vereinfacht. Die Fehlersuche, aber auch das automatisierte Testen werden durch den Aufbau vereinfacht [16]. Ebenfalls müssen sich keine Gedanken gemacht werden, wie die grafische Oberfläche auf das Aktualisieren von Daten reagiert, da dies von Vue und VueX übernommen wird.

Auslagerung der Businesslogik Durch die Auslagerung der Businesslogik kann die Anwendung größere Datenmengen besser verarbeiten, sodass die grafische Oberfläche nicht unter Geschwindigkeitseinbußen leidet. Zudem ist es möglich, die Logik einfach auf einen Service-Worker zu übertragen, der im Hintergrund arbeiten kann, ohne dass die PWA geöffnet ist.

Dezentrale Infrastruktur Es müssen keine zentralen Systeme gehostet werden, in denen E-Mails oder Nutzerdaten gespeichert werden. Die Arbeit der Speicherung von E-Mails wird an die jeweiligen E-Mail-Provider übertragen.

4.4.2 Nachteile

Alte Protokolle

Die Nutzung von *SMTP* und *IMAP*, welche zwingend nötig sind, um über E-Mails zu kommunizieren, kann zu Problemen in der Architektur führen. Der E-Mail-Proxy wird zu einem *Bottleneck*, denn alle Nutzer prüfen über diesen ihre E-Mail-Postfächer oder verschicken neue E-Mails. Fällt der E-Mail-Proxy aus, funktioniert die gesamte Anwendung nicht mehr.

Die Nachteile, die *SMTP* und *IMAP* mit sich bringen, könnten jedoch in Zukunft durch das *JSON-Meta-Application-Protocol* (*JMAP*) gelöst werden.
Das 2019 spezifizierte Protokoll ist für die Nutzung von Mobile- und Web-Anwendungen konzipiert und baut auf *HTTP* auf [12]. Jedoch ist die Verbreitung bislang sehr gering, weshalb es noch einige Zeit dauern dürfte, bis es flächendeckend benutzt werden kann.

5 Implementierung

5.1 E-Mails

Die E-Mail ist in der Anwendung das grundlegende Austauschformat, auf ihr baut die gesamte Kommunikation zwischen den Nutzern auf. Um diese Kommunikation zu gewährleisten, müssen gewisse Formate eingehalten werden.
Grundsätzlich besteht eine E-Mail aus einem Header (Kopfzeile) und einem Body (Körper, Inhalt), welche spezifische Formate haben. Der *Header* besteht aus Feldern, welche beispielsweise angeben, von wem die E-Mail stammt oder an wen sie gehen soll. Manche Felder sind optional, wie zum Beispiel das *user-agent*-Feld, das angibt, über welche Anwendung die E-Mail verschickt wurde. Andere hingegen sind zwingend nötig, wie das *from*-Feld (gibt an, von wem die E-Mail stammt), das *to*-Feld (an wen soll die E-Mail gesendet werden) oder auch die *message-id*, welche zur eindeutigen Identifizierung der E-Mail dient. Grundlegend folgt die Implementierung der PWA und des E-Mail-Proxys den Vorgaben von RFC 5322 [10].
Um das reibungslose Mitlesen von E-Mails zu garantieren, sollen in E-Mails, welche über die PWA verschickt werden, immer Inhalte im Betreff und im Body stehen.

E-Mails, die über die PWA bzw. den E-Mail-Proxy versendet oder empfangen werden, sind reine Texte. Multi-Part E-Mails werden nicht unterstützt.

Generierung von *message-ids*

Wie aus RFC 5322 ebenfalls hervorgeht, sind *message-ids* immer aus zwei Teilen aufgebaut [10].
Der erste Teil sollte sich hierbei aus einzigartigen Werten zusammensetzen. Als Beispiel werden Prozess-IDs des Computers oder auch das Datum in Millisekunden aufgeführt. Natürlich kann die *message-id* auch mit beliebigen anderen Werten erstellt werden. Es ist nur zu beachten, das eine einzigartige *message-id* dabei entsteht.
Der zweite Teil beinhaltet die Domain, über die die E-Mail verschickt wird und die durch ein @ vom ersten Teil getrennt ist.

In der Implementierung wird die *message-id* erstellt aus einem festen Wert, welcher zur Identifizierung dient (siehe Kapitel 5.2.2), dem aktuellen Datum in Millisekunden und einer zufälligen 16 stelligen Zeichenkette auf der linken und der benötigten Domain auf der rechten Seite. Der feste Wert für die *message-id* ist derselbe wie für den *user-agent*. Eine mögliche *message-id* könnte wie folgt aussehen:

wohnheimpwa-1610964168373-45p4M=ToncalVQF+@example.net

5.2 E-Mail-Proxy

Um E-Mails aus der PWA versenden zu können, ist es zwingend nötig, eine serverseitige Schnittstelle bereit zu stellen. Diese Schnittstelle soll möglichst einfach zu bedienen sein. Die Wahl der Sprache für den Proxy fiel auf Python, denn die Sprache besitzt standardmäßig Libraries für den Umgang mit *SMTP* und *IMAP*. Für die Web-Schnittstelle wird das Framework *FastAPI* verwendet. Dieses ermöglicht die schnelle Erstellung von Schnittstellen mit dem normalen Toolset von Python, zudem erstellt es automatisch eine interaktive Dokumentation, welche zur Laufzeit des E-Mail-Proxys aufgerufen und getestet werden kann [4].
Kombiniert ergibt sich eine leicht zu erweiternde Grundlage für neue Features.

5.2.1 Caching von Provider-Daten

Wie bereits in der Architektur erwähnt, ist es vorteilhaft, die Verbindungsdaten der Provider im E-Mail-Proxy zwischen zuspeichern. Zum einen wäre die Datenmenge bei Anfragen an den E-Mail-Proxy reduziert. Zum anderen müssen die Daten nicht neu validiert werden, sondern werden einmal beim Anlegen validiert und können dann ohne Probleme genutzt werden.
Der Nutzer profitiert ebenfalls aus den Zwischenspeicherung, denn wenn sein Provider bereits hinterlegt wurde, kann er die Verbindungsdaten direkt nutzen.

Um einen neuen Provider in das System zu bringen, müssen lediglich die folgenden Daten an den dafür vorgesehenen Endpunkt geschickt werden:

- **id** - Die ID des Providers wird in Anfragen mitgeschickt.
- **name** - Anzeigename des Providers.
- **baseUrl** - URL zur Website des Providers.
- **imapUrl** - URL des IMAP Endpunktes des Providers.
- **imapPort** - Port des IMAP Endpunktes des Providers.
- **imapEncryption** - Verschlüsselungsart, die von IMAP genutzt wird.
- **smtpUrl** - URL des SMTP Endpunktes des Providers.
- **smtpPort** - Port des SMTP Endpunktes des Providers.
- **smtpEncryption** - Verschlüsselungsart, die von SMTP genutzt wird.

Nach der Übertragung wird zuerst überprüft, ob nicht schon ein Provider mit derselben ID im System vorhanden ist. Befindet sich die ID schon im System, wird anschließend überprüft, ob sich die Verbindungsdaten unterscheiden. In diesem Fall wird dem Nutzer eine Fehlermeldung zurückgesendet.
Wird keine Übereinstimmung gefunden, folgt eine Überprüfung, ob die *IMAP* und *SMTP* Verbindungen valide sind. Hierfür werden beide Verbindungen ausprobiert. Schlägt eine

oder beide Verbindungen fehl, wird eine Fehlermeldung mit der jeweils fehlgeschlagenen Verbindung zurückgegeben.
Entstehen beim Testen keine Fehler, wird der Provider in das System eingetragen und der Nutzer darüber informiert. Ab diesem Zeitpunkt ist es möglich, in allen folgenden Anfragen nur die Provider-ID zu hinterlegen und die Verbindung zu nutzen.

5.2.2 Identifizierung von E-Mails

Um nicht alle E-Mails des Nutzers durchsuchen zu müssen, wird das in RFC3501 beschriebene *SEARCH*-Kommando verwendet [8]. Es ermöglicht die providerseitige Suche von E-Mails mittels einer einfachen Syntax. Dabei können sowohl der Header als auch der Body durchsucht werden.
Für die PWA sind zwei Arten von E-Mails interessant. Erstens die E-Mails, welche über die PWA verschickt werden, und zweitens Antworten auf E-Mails aus der PWA.
Die beiden E-Mail-Arten werden mit der folgenden Zeichenkette gesucht:

> HEADER user-agent wohnheimpwa SINCE <DD.MM.JJJJ> OR HEADER references wohnheimpwa SINCE <DD.MM.JJJJ>

Mit der Zeichenkette werden providerseitig alle E-Mails mit dem Wert *wohnheimpwa* im *user-agent*-Feld oder im *references*-Feld gesucht. Zusätzlich werden E-Mails gefiltert, die seit dem Tag der letzten erhaltenen E-Mail verschickt wurden. Da die Syntax nur das Filtern nach vollen Tagen und keine nach Zeiten zulässt, werden die E-Mails in der PWA erneut gefiltert. Alle E-Mails, die nach dem Filtern noch übrig sind, werden weiterverarbeitet.

Wurden alle E-Mails verarbeitet, wird der Zeitpunkt der letzten erhaltenen E-Mail in der Datenbank gespeichert, um bei der nächsten Suche wieder genutzt werden zu können.

5.2.3 Moderierung der Postfächer

Eine Moderierung der Postfächer des Nutzers ist grundlegend nicht zwingend nötig, jedoch entstehen durch die Anwendung potentiell viele E-Mails, die ein Postfach schnell füllen können. Dadurch kann die Suche nach wichtigen E-Mails erschwert werden.
Um kein überlaufendes Postfach zu hinterlassen, sollen die vom E-Mail-Proxy gefundenen E-Mails in einen separaten Ordner verschoben werden. Eine E-Mail wird in dem Moment verschoben, wenn sie vom E-Mail-Proxy als für die PWA relevant angesehen wird. Die Information, in welches Postfach die E-Mail verschoben werden soll, muss mit jeder Anfrage an den E-Mail-Proxy übermittelt werden.

5.2.4 Filterung von Header-Feldern

Viele Provider verwenden sogenannte *X-Header-Fields*, um zusätzliche Features zu ermöglichen. Diese werden durch den Provider erstellt und in die E-Mail eingetragen.
Aus RFC 6648 geht hervor, dass die *X-Header-Fields* veraltet sind und nicht mehr genutzt werden sollten [13]. Der E-Mail-Proxy überprüft die *X-Header-Fields* und könnte diese ausfiltern, da sie in der PWA sowieso nicht verwendet werden.

5.3 Progressive-Web-App

5.3.1 Chats

Um innerhalb der PWA sinnvoll *Messages* zu verwalten, werden diese in Chats zusammen gefasst. Jede *Message* enthält eine Referenz auf den Chat, die von diesem überprüft wird, bevor er diese hinzufügt. Auf diese Weise ist es möglich *Messages* nach dem Versenden genau zuzuordnen.
Je nach Typ der *Message* wird beispielsweise geprüft, ob der Absender ein Mitglied oder ein Administrator des Chats ist. Ist der Absender nicht berechtigt oder die *Message* enthält Fehler, wird diese abgewiesen und nicht weiter verarbeitet. Über abgelehnte *Messages* wird der Absender nicht informiert, sie werden einfach ignoriert.
Ein Chat kann ebenfalls dafür genutzt werden, um ToDo-Listen oder auch Putzpläne verfügbar zu machen. Diese können dann nur innerhalb des Chats von dessen Mitgliedern genutzt werden.

5.3.2 Warum *Messages*, statt E-Mails gespeichert werden

In den ersten Versionsständen der PWA wurden noch komplette E-Mails gespeichert, dies wurde jedoch schnell verworfen, da daraus einige Probleme entstehen können.
Als Erstes ist zu beachten, dass die E-Mails auch Informationen enthalten, die für die PWA und den tatsächlichen Inhalt der Nachricht redundant sind. Header-Felder wie der *mime-type* oder der *content-type* haben in der Verarbeitung der E-Mails durch die PWA keine Relevanz, sie nehmen somit nur Speicherplatz in Anspruch.

Der zweite Punkt ist, dass auf kleiner geschnittenen Klassen leichter aufgebaut werden kann. Die Grundklasse enthält nur Informationen über den Absender, das Absendedatum, die ID der E-Mail und zu welchem Chat sie gehört. Spezielle *Messages* können durch die Implementierung, aufbauend auf der Grundklasse, realisiert werden und enthalten nur die nötigsten Daten aus der entsprechenden E-Mail.

Grundsätzlich sieht die Implementierung wie in Abbildung 5.1 auf der nächsten Seite aus. Die Basis stellt die abstrakte Klasse *AMessage*. Aus ihr können nun beispielsweise die *TextMessage*, welche eine einfache Text-Nachricht darstellt, oder die *InviteMessage* zum Einladen weiterer Nutzer in einen Chat erstellt werden.

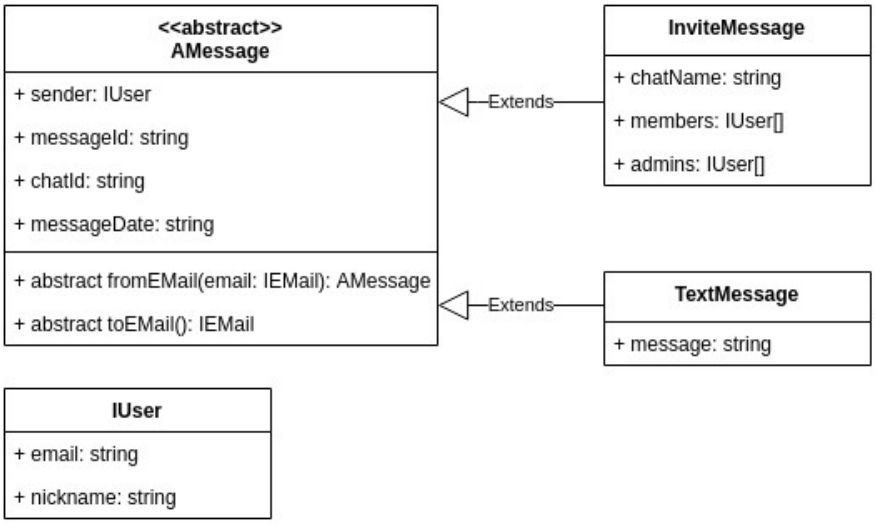

Abbildung 5.1: Beispiel der *Message* Ableitungen

5.3.3 Von der E-Mail zur Message und zurück

Damit die Anwendung arbeiten kann, ist es nötig, von einer E-Mail zu einer *Message* zu gelangen. Erklärt wird dieser Vorgang anhand einer *InviteMessage* und einer *TextMessage*. Die Abbildung 5.2 stellt den Vorgang grafisch dar.

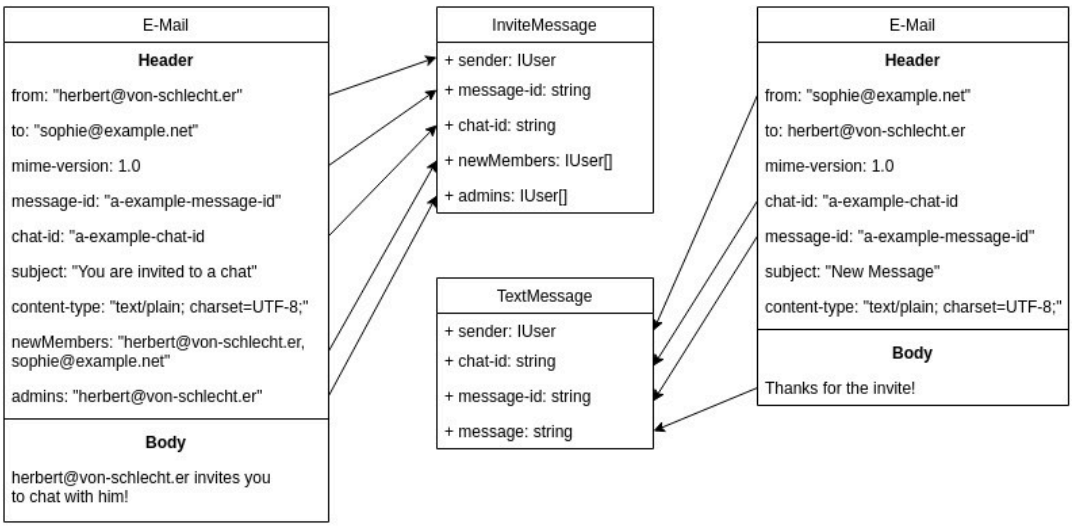

Abbildung 5.2: Erzeugen einer *Message* aus einer E-Mail

Das Erzeugen einer *Message* aus einer E-Mail ist im Vergleich zum umgekehrten Vorgang einfacher, da eine E-Mail bereits alle benötigten Informationen beinhaltet. Aus dem *Header* können die meisten Informationen einfach übernommen werden. Der *sender* wird aus dem *from* Feld extrahiert, genau so wie die *message-id*, die *chat-id* oder auch die Information, welche Mitglieder neu eingeladen werden. Der Inhalt für eine *TextMessage* kann

direkt aus dem E-Mail-Body übernommen werden.

Um eine *Message* an andere Nutzer zu verschicken, ist es nötig eine E-Mail zu erzeugen. Im Gegensatz zur Erzeugung einer *Message* aus einer E-Mail sind für diesen Vorgang mehrere Datenquellen nötig.
Für die Umwandlung einer *TextMessage* in eine E-Mail wird die *Message* selbst und ein Chat benötigt. Aus der *Message* wird der Body, die *message-id* sowie der *sender* übernommen. Der Chat liefert die Information, an wen die E-Mail versendet werden soll, nämlich an alle Nutzer des Chats, mit Ausnahme des Absenders.
Die fehlenden Informationen, wie beispielsweise der *content-type* oder die *mime-version* werden aus einer statischen Liste entnommen, welche in der Anwendung hinterlegt ist.

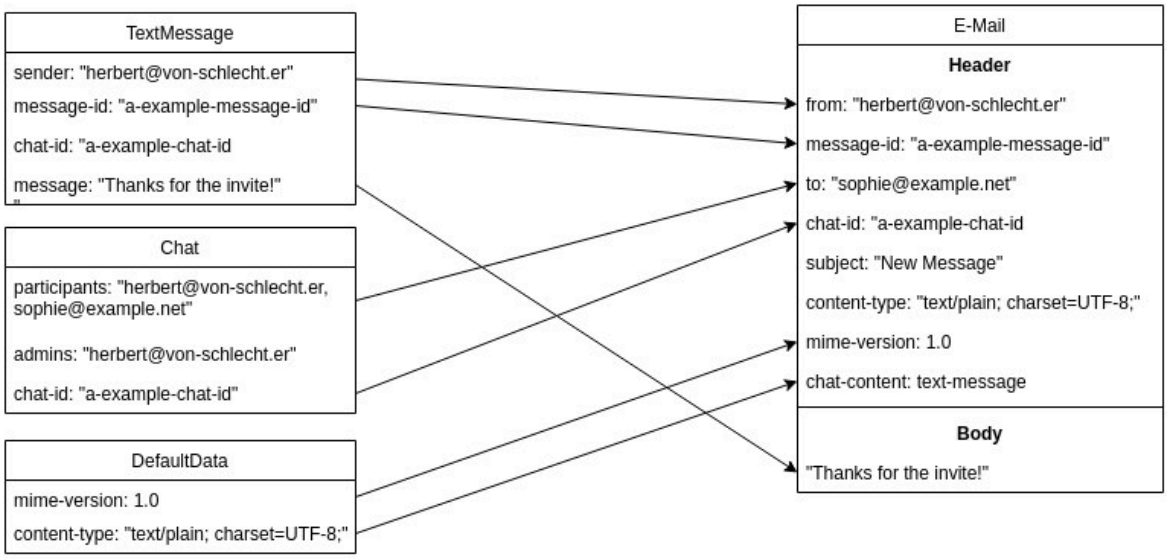

Abbildung 5.3: Erzeugen einer E-Mail aus einer *Message*

5.3.4 Actionbasierte Kommunikation zwischen Main-Thread und Web-Worker

Die Kommunikation zwischen einem Web-Worker und dem Main-Thread baut auf einfachen Events auf, welche einen Namen und einen Inhalt haben. Um die Kommunikation zu standardisieren, wird die Klasse *WebWorkerEvent* verwendet, welche den Typen des Events, die zu transportierenden Daten und eine eindeutige ID beinhaltet. Die Klasse besitzt selbst keine Logik, da sie nur zum Austauschen von Daten verwendet wird. Diese *WebWorkerEvents* können an den Web-Worker oder von ihm zurück gesendet werden.

Um einfacher zwischen VueX und dem Web-Worker zu kommunizieren, werden das *ActionWebWorkerEvent* und das *ResponseWebWorkerEvent* verwendet, welche beide das *WebWorkerEvent* erweitern (siehe Abbildung 5.4).

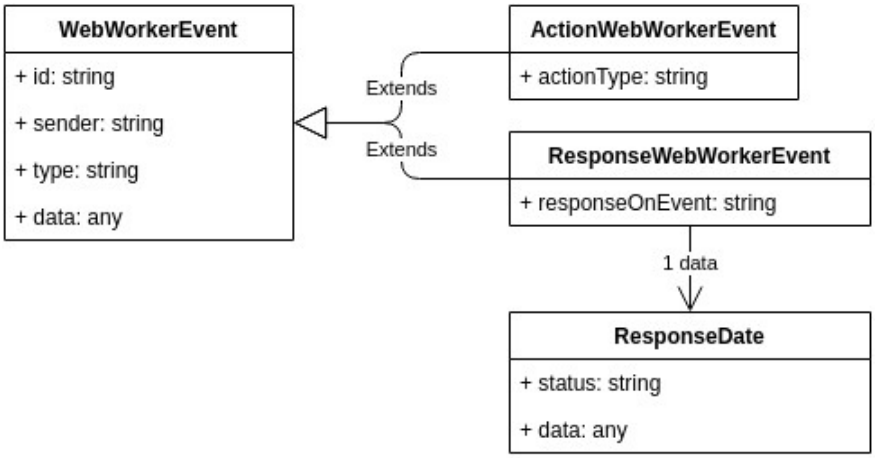

Abbildung 5.4: Aufbau von WebWorkerEvents

Das *ActionWebWorkerEvent* speichert lediglich zusätzlich den Typen der *Action*. Es handelt sich dabei um dieselben *Action*-Typen, die auch in VueX genutzt werden.
Das *ResponseWebWorkerEvent* ist dafür bestimmt, auf *WebWorkerEvents* zu antworten. Dafür wird das ursprüngliche Event verlinkt und mit neuen zusätzlichen Daten zurückgesendet.

Voll ausgeschöpft wird das Potential dieser beiden Events durch einen promise-basierten *EventListener* (siehe Abbildung 5.5), auch *WebWorkerResponseListener* genannt. Ein *Promise*-Objekt ist ein Objekt mit zwei *callback*-Funktionen, eine für erfolgreiche und eine für fehlerhafte Verarbeitung, die entsprechend aufgerufen werden.

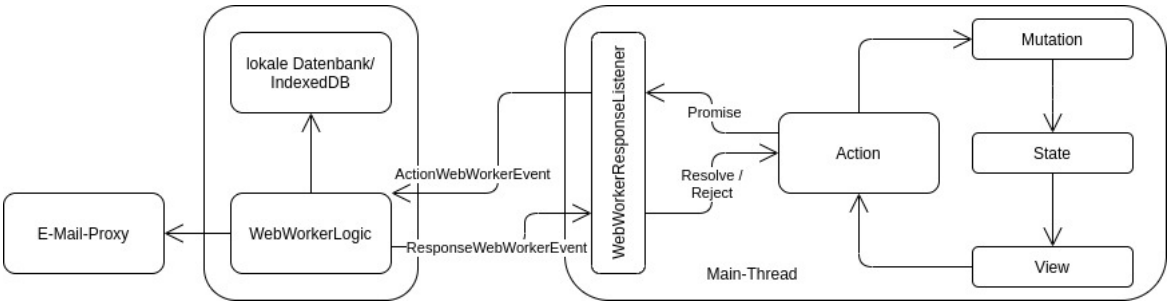

Abbildung 5.5: Kommunikation zwischen Main-Thread und Web-Worker

Events, die über diesen *WebWorkerResponseListener* verschickt werden, können drei Antworten haben. Die Erste ist ein erfolgreiches *ResponseWebWorkerEvent*, welches vom Web-Worker zurückkommt, in diesem Fall wird vom hinterlegten Promise die *Resolve*-Funktion aufgerufen. Die zweite Möglichkeit ist ein negatives *ResponseWebWorkerEvent*, welches die *Reject*-Funktion des Promise aufruft. Die dritte Möglichkeit ist, dass nach einer gewissen Zeitspanne noch immer kein Event zurückgekommen ist. Wie im zweiten Fall wird die *Reject*-Funktion aufgerufen, es wird jedoch ein anderer Fehler übergeben.

Dieser Aufbau ermöglichst es die *Actions*, welche normalerweise auf dem Main-Thread verarbeitet werden, auf den Web-Worker auszulagern.

5.4 E-Mail-Typen

In der Anwendung werden viele unterschiedliche E-Mails zwischen Nutzern verschickt. Anhand der unterschiedlichen Aufgaben, die sie erfüllen, lassen sie sich verschiedenen Typen zuordnen.
Um Nutzer in einen Chat einzuladen, werden *Invite* E-Mails / *Messages* verschickt, für eine normale Text-Nachricht eine *TextMessage*.
Alle unterschiedlichen Typen besitzen eine Basis an Informationen, wie die ID des Chats, zu dem sie gehören. Jedoch besitzen sie auch eigene Daten, die für die Verarbeitung notwendig sind.
Wie bereits in 5.1 beschrieben, haben E-Mails eine festgelegte Struktur und benötigen gewisse Grundinformationen, um versendet zu werden. In diesem Abschnitt wird weitergehend beschrieben, wie die Funktionen technisch funktionieren und welche Inhalte untereinander ausgetauscht werden.
Alle E-Mails die über die PWA verschickt werden, benötigten die folgenden grundlegenden Daten.

E-Mail-Typ	**Daten**	**Beschreibung**
Basisinformationen	**chat-id** - ID des zugehörigen Chats. **chat-content** - Beschreibt den E-Mail-Typ.	Sind in jeder anderen E-Mail enthalten.

Tabelle 5.1: Grundinformationen einer E-Mail aus der PWA

5.4.1 Chats

Wie aus Abschnitt 3.2 hervorgeht, sollen Chats als Grundlage für die Kommunikation zwischen Nutzern dienen. Sie beinhalten eine Liste von Nutzern und Nachrichten, die untereinander ausgetauscht werden. Durch die Festlegung, ob ein Chat öffentlich ist und ob nur Administratoren Nachrichten in ihm senden dürfen, können verschiedene Arten von Chats erzeugt werden.
In der folgenden Tabelle werden die daraus resultierenden E-Mail-Typen mit den beinhaltenden Daten aufgeführt.

E-Mail-Typ	**Extra *Header-Fields***	**Beschreibung**
Invite	**chat-name** -Name des jeweiligen Chats. **new-members** - E-Mail-Liste der neu Nutzer. **admins** - E-Mail-Liste der Chat-Administratoren. **is-public** - Gibt an, ob der Chat öffentlich oder privat ist. **only-admins-can-send** - Gibt an, ob nur Administratoren in der Lage sind Nachrichten zu versenden. **chat-type** - Gibt an, welche Art von Chat erstellt werden soll.	Wird genutzt, um neue Nutzer in einen bereits existierenden oder neu zu gründenden Chat einzuladen.
TextMessage	Es werden keine zusätzlichen Felder im Header benötigt.	Wird genutzt, um einfache Text-Nachrichten zu übermitteln. Die Nachricht selbst steht im Body der E-Mails.
DeleteUser= FromChat	**delete-user-email** - E-Mail-Adresse des Nutzers, der aus dem Chat entfernt werden soll.	Wird genutzt, um Nutzer aus einem Chat zu entfernen. Diese Nachricht kann sowohl von Administratoren gesendet, als auch von dem Nutzer gesendet werden, der den Chat verlassen möchte.
Promote	**promote-user-email** - E-Mail-Adresse des Nutzers, der befördert oder degradiert werden soll. **to-admin** - Ist ein boolescher Wert und beschreibt, ob der Nutzer zu einem Administrator oder zu einem normalen Nutzer werden soll.	Wird genutzt, um normale Nutzer zu Administratoren zu befördern oder Administratoren zu normalen Nutzern zu degradieren.
JoinRequest	Es werden keine zusätzlichen Felder im Header benötigt.	Eine Join-Request-E-Mail wird verwendet, um öffentlichen Chats beizutreten. Die E-Mail wird an einen aktuellen Administrator des öffentlichen Chats versendet.

Tabelle 5.2: Generelle E-Mail-Typen eines Chats

5.4.2 Einkaufslisten, ToDo-Listen

Eine ToDo-Liste besteht aus zwei Elementen, der Liste selbst und den Punkten, die auf ihr stehen. Die Punkte auf der Liste können erstellt, gelöscht und als erledigt oder unerledigt markiert werden.

E-Mail-Typ	**Daten**	**Beschreibung**
NewToDoList	**todo-list-name** - Name der ToDo-Liste.	Diese E-Mail wird verwendet, um eine neue ToDo-Liste anzulegen.
ChangeToDo	**reference** - Referenziert mittels der ID auf zugehörige ToDo-Liste. **state** - Wird verwendet, um den Zustand zu wechseln. Dabei sind die folgenden Zustände möglich: *new*, *check*, *uncheck* oder *delete*.	Mittels einer *ChangeToDo*-E-Mail kann ein neuer Punkt in der ToDo-Liste erstellt, gelöscht, als erledigt oder unerledigt markiert werden

Tabelle 5.3: E-Mails einer ToDo-Liste

5.4.3 Putzpläne

Ein Putzplan wird wie eine ToDo-Liste an einen Chat gebunden und ist auch nur über diesen aufrufbar. Dabei wird in einer Tabelle festgehalten, wer welche Aufgabe übernehmen muss. Die Aufgaben werden dabei in einem Zyklus von x Wochen rotiert, sodass jeder einmal jede Aufgabe zu erledigen hat.
Zu beachten ist, dass es nur so viele Aufgaben geben kann, wie es Mitglieder in dem Chat gibt.
Dabei kommen die folgenden E-Mails zum Einsatz.

E-Mail-Typ	**Daten**	**Beschreibung**
NewPlan	**members-task-mapping** - Zuordnung von Nutzern und Aufgaben. Die Zeichenkette wird nach dem folgenden Zuordnungsschema aufgebaut: Nutzer-Aufgabe, Nutzer-Aufgabe, ... **start-week** - Woche, in der der Plan startet.	Ist die initiale E-Mail zum Erstellen eines Putzplanes. Soll dieser bspw. geändert werden, wird einfach ein neuer Putzplan erstellt und der alte überschrieben.

Tabelle 5.4: E-Mails eines Putzplans

5.4.4 Meldungen an den Hausmeister

Meldungen an den Hausmeister werden über einen normalen Chat an den Hausmeister gesendet. Für die Kommunikation werden einfache *TextMessages* verwendet, welche über eine spezifische Seite in der *PWA* versendet werden können.

5.4.5 News-Forum und Veranstaltungsinformationen

Ein News-Forum oder ein Chat, über den Veranstaltungsinformationen versendet werden, ist grundsätzlich ein normaler Chat. Jedoch kann durch die in 5.4.1 beschriebenen Chat-einstellungen festgelegt werden, wer alles Nachrichten senden kann und ob der Chat öffentlich oder nur privat zugänglich ist.
Um Neuigkeiten in einem Chat hervorzuheben, können News mit einem Titel und einer Nachricht versehen werden. Im Chatverlauf werden diese durch Darstellung des Titels und der Nachricht besonders hervorgehoben.

E-Mail-Typ	**Daten**	**Beschreibung**
News	Inhalte werden im Betreff und Body der E-Mail abgelegt.	Eine News-Nachricht wird wie eine normale Text-Nachricht ohne weitere Inhalte verschickt. Der Titel wird im Betreff abgelegt und der Inhalt im Body der E-Mail.

Tabelle 5.5: E-Mails für News-Foren

5.4.6 Kontaktdaten

Die Liste von Kontaktdaten ist kein Feature, das mit anderen Nutzer kommuniziert, daher gibt es dafür keine E-Mail-Typen.

Die Kontaktdaten umfassen zum einen statisch eingetragene Kontakte bspw. zu den Lübecker Hochschulen, rund um das Studentendorf Lübeck, aber auch andere möglicherweise nützliche Kontakte. Diese Kontakte sind nicht vom Nutzer editierbar.
Zum anderen werden neue Kontakte erstellt, wenn durch Chats unbekannte E-Mail-Adressen ins System gelangen. Die Kontakte beinhalten die E-Mail-Adresse und einen Nicknamen, welcher an vielen Stellen in der PWA eingesetzt wird. Der Nickname kann vom Nutzer geändert werden.

5.4.7 Kleinanzeigen und Tauschbörsen

Dieses Feature kann für vielerlei Dinge genutzt werden, zum Beispiel für Kleinanzeigen, Tauschbörsen, Foodsharing oder auch zum Suchen von Paketen im Wohnheim.
Der Absender kann über die Anzeige kontaktiert werden.

E-Mail-Typ	**Daten**	**Beschreibung**
NewAd	**ad-id** - ID des Inserats **title** - Titel des Inserats **price-type** - Art der Gegenleistung **price-amount** - Preisangabe	Wird zur Erstellung eines neuen Inserats verwendet. Es enthält die benötigten Daten wie Titel, Beschreibung (wird aus dem E-Mail-Body übernommen) sowie den Preis / die Gegenleistung. Die E-Mail kann aber auch dafür verwendet werden, bereits vorhandene Inserate zu aktualisieren. Hierbei wird dieselbe ID des Inserates verwendet, zusätzlich werden aber die zu verändernden Werte angegeben.
DeleteAd	**ad-id** - ID des Inserats, welches gelöscht werden soll.	Wird verwendet, um ein Inserat zu löschen.

Tabelle 5.6: E-Mails zum Verwalten von Kleinanzeigen, Tauschbörsen oder ähnlichem

6 Sicherheitskonzept

Das Sicherheitskonzept gliedert sich in drei Teile. In Teil eins (Kap. 6.1) wird der Umgang mit eingehenden E-Mails behandelt, d. h. ob sie weiter verarbeitet und dem Nutzer angezeigt werden. Der zweite Teil (Kap. 6.2) umfasst das Thema Ende-zu-Ende-Verschlüsselung (E2ee). Aufgrund von Zeitmangel und der Anforderung, dass Chatinhalte auch lesend ohne Nutzung der PWA verfolgbar seinen sollen, wurde das Konzept der Verschlüsselung nicht umgesetzt. Der letzte Teil (Kap. 6.3) befasst sich mit der Handhabung von Zugangsdaten des Nutzers und deren Speicherung.

6.1 Sicherheit durch das Aussortieren von E-Mails

Um die Chats der Nutzer vor unautorisierten Nachrichten zu schützen, ist es notwendig, eingehende E-Mails zu prüfen. Kommt ein Angreifer an Daten wie beispielsweise die ID eines Chats, könnte er versuchen, diesen mit selbst erstellten E-Mails zu infiltrieren. Denn E-Mails können einfach selbst erstellt, mit zusätzlichen Daten, welche die PWA nutzt, versehen und an Mitglieder des Chats versendet werden.
Mögliche Angriffe wären zum Beispiel das Versenden einfacher Text-Nachrichten, aber auch der Versuch, Zugang zu einem privaten Chat zu erhalten.

Um solche Fälle zu verhindern, wird bei jeder neuen Nachricht überprüft, ob diese Nachricht auch wirklich in den Chat darf.
Ist der Absender kein Mitglied des Chats oder ist die Funktion, die genutzt werden soll, nur für Administratoren verfügbar, wird die Nachricht abgewiesen und nicht weiterverarbeitet.
Wurde die Nachricht durch die Überprüfung nicht heraus gefiltert, wird diese weiterverarbeitet und dem Chats hinzugefügt. Nachrichten mit bereits bekannten IDs können nicht erneut dem Chat hinzugefügt werden.

6.2 Sicherheit durch Verschlüsselung

Ein weiterer Sicherheitsaspekt wäre das Verschlüsseln von E-Mails. Dieses Kapitel umfasst das Thema grob und gibt eine kleine Übersicht darüber, wie das Verschlüsseln von E-Mails in die PWA eingebaut werden könnte. Eine Implementierung der Verschlüsselung wurde im Prototypen nicht umgesetzt.

Ein Verfahren, zur Verschlüsselung von E-Mails ist *OpenPGP*, welches in RFC 4880 2007 in der Version 5 standardisiert wurde [9]. *OpenPGP* ist eine Technologie, mit der neben Ver-

schlüsselungen auch Signaturen oder Kompressionen von Daten vorgenommen werden können. Jedoch ist für diese Arbeit nur der Teil zum Verschlüsseln interessant.
OpenPGP verwendet zum Verschlüsseln von E-Mails ein hybrides Verschlüsselungsverfahren. Es wird ein Schlüssel (*Session-Key*) verwendet, der einmalig genutzt wird und durch ein asymmetrisches Verschlüsselungsverfahren verschlüsselt wird. Die Nachricht selbst wird mit einem symmetrischen Verschlüsselungsverfahren durch den *Session-Key* verschlüsselt und zusammen mit dem *Session-Key* an einen anderen Nutzer geschickt.
Bei der asymmetrischen Verschlüsselung gibt es pro Nutzer ein Schlüsselpaar bestehend aus einem öffentlichen und einem privaten Schlüssel. Der öffentliche Schlüssel kann an jeden Kommunikationspartner verteilt werden, um Antwortnachrichten zu verschlüsseln. Nur der Besitzer des privaten Schlüssels ist in der Lage, die mit dem zugehörigen öffentlichen Schlüssel verschlüsselte Nachricht zu entschlüsseln.

Das Problem bei der Verschlüsselung ist nicht die Übertragung der Nachricht mit dem benötigten *Session-Key*, sondern die Übertragung des eigenen öffentlichen Schlüssels. Hier gibt es viele Wege, beispielsweise ein reales Treffen, bei welchem ein Schlüsselaustausch stattfindet oder ein Online-Dienst, welcher öffentliche Schlüssel annimmt, damit andere Nutzer sie von dort beziehen können.
Jedoch sind wenige Nutzer in der Lage oder wollen den Aufwand betreiben, sich ein Schlüsselpaar zu erstellen und den öffentlichen Schlüssel zu verteilen. Um diesen Vorgang des Schlüsselaustausches zu automatisieren, könnte das *AutoCrypt* Verfahren genutzt werden, welches auf *OpenPGP* aufbaut. *AutoCrypt* ist jedoch kein Programm, das installiert wird, sondern eine Spezifikation, die den Austausch von öffentlichen Schlüsseln in 1 zu 1 und 1 zu N Kommunikationen behandelt [2].

Der Schlüsselaustausch nach dem Verfahren von *AutoCrypt* wird über das Versenden von E-Mails abgewickelt [3]. Dabei wird beispielsweise von Alice eine unverschlüsselte E-Mail an Bob geschickt, da sie an dieser Stelle noch keinen öffentlichen Schlüssel von Bob besitzt. Im Header-Feld *autocrypt* der E-Mail ist der öffentliche Schlüssel von Alice enthalten. Empfängt Bob diese Nachricht und stellt fest, dass im Header der öffentliche Schlüssel vermerkt ist, wird dieser gespeichert. Bob kennt nun den öffentlichen Schlüssel von Alice [3]. Bei der nächsten E-Mail, die Bob an Alice sendet, verwendet er dasselbe Verfahren wie Alice zuvor. Er schreibt seinen öffentlichen Schlüssel in den Header der E-Mail und Alice kann diesen bei Empfang ebenfalls speichern.
Die darauf folgenden E-Mails können nun alle mittelst *OpenPGP* verschlüsselt werden und vom jeweils anderen entschlüsselt werden.

Um öffentliche Schlüssel in einer 1 zu N Kommunikation auszutauschen, gibt es die Möglichkeit, das Header-Field *autocrypt-gossip* zu verwenden, bei welchem der öffentliche Schlüssel von jedem bekannten Empfänger mit gesendet wird. Jedoch raten die Entwickler dazu, eher das normale Autocrypt-Header-Feld zu verwenden, da dieses eine höhere Vertrauenswürdigkeit aufweist [3].

6.3 Umgang mit Zugangsdaten

Der Umgang mit Zugangsdaten ist ein wichtiger Punkt innerhalb der Anwendung. Der aktuelle Stand der PWA ist, dass das Passwort des Nutzers während der Laufzeit dort unverschlüsselt hinterlegt ist. Wird im Login-Prozess der Haken gesetzt, dass das Passwort in der lokalen Datenbank gespeichert werden darf, ist es sogar unverschlüsselt für längere Zeit persistiert. Diese Art der Handhabung mag für einen frühen Prototypen verkraftbar sein, jedoch ist diese Variante nicht für den Einsatz stark verbreiteter Software geeignet.

Ein möglicher Lösungsweg wäre beispielsweise der Einsatz von *OAuth*, das seit Oktober 2012 mit RFC 6749 spezifiziert ist [11]. *OAuth* ist ein Verfahren, welches das klassische Client-Server Authentifizierungsverfahren ablöst, bei dem die Anwendung das Passwort und den Nutzernamen zur Authentifizierung nutzt. Stattdessen teilt der Server, mit dem kommuniziert wird, sogenannte Token aus, mit welchen sich der Client als berechtigt ausweist, eine bestimmte Ressource zu nutzen. Dieses Verfahren hat den Vorteil, dass keine Passwörter durch die Anwendung gespeichert werden müssen. Durch die Vergabe von Token ist es zudem möglich, bestimmten Anwendungen mehr Rechte auf Ressourcen zu geben und anderen nicht. Auch können Zugriffe von einzelnen Anwendungen komplett widerrufen werden, ohne das Passwort zu ändern, was ansonsten zur Folge hätte, das alle Anwendungen keinen Zugriff mehr hätten.
Das Problem bei diesem Ansatz ist, dass *SMTP* und *IMAP* diesen Weg der Authentifizierung bislang nicht verwenden. Die Ausnahmen bilden Google und Microsoft, welche beide die Option eingebaut haben, *OAuth* zu nutzen [1][6].
Theoretisch wäre es möglich, die Zugangsdaten des Nutzer serverseitig im E-Mail-Proxy zu speichern und für die PWA *OAuth*-Token zu generieren. Jedoch würde dieser Ansatz das Problem nur von der PWA auf den E-Mail-Proxy verschieben und es nicht lösen.
Die Lösung des Problems läge darin, dass die E-Mail-Provider dem Vorbild von Google und Microsoft folgen und ebenfalls die Möglichkeit implementieren, *OAuth* in ihren Anwendungen zu nutzen.

7 Nutzertests

Um zu zeigen, dass der entwickelte Prototyp genutzt werden kann, wurden mit sechs Personen Usability-Tests durchgeführt. Die Personen wohnen aktuell oder wohnten in einem Studentenwohnheim.
Da die PWA aus Bedenken an die Datensicherheit noch nicht für einen großen Test mit beispielsweise mehr als 20 oder 30 Personen geeignet ist, wurde nur in kleinem Rahmen getestet. Ebenso sind noch einige Bugs bekannt, welche behoben werden müssten, damit die PWA ohne Erklärung genutzt werden kann.
In den einzelnen Tests wurden den Test-Probanden Aufgaben gestellt, welche sie mithilfe der PWA lösen sollten. Die Aufgaben umfassten das Einloggen in der PWA, das Einrichten der Postfächer, das Beitreten eines angepinnten Chats (bspw. Kleinanzeigen oder Foodsharing), Absenden von neuen Inseraten, Erstellen eines neuen Chats mit einer Person und das Versenden von Meldungen an den Hausmeister.
In den folgenden Abschnitten werden die Beobachtungen und resultierenden Verbesserungen aufgeführt.

Einloggen

Die Probanden bekamen zu Beginn des Tests eine kurze Einweisung, wie das System grundsätzlich funktioniert und welche Daten zum Einloggen benötigt werden. Der Prozess des Einloggens in die PWA verlief bei Providern wie Web.de oder Outlook / Office 365 ohne Probleme. Bei der TH-Lübeck gab es aufgrund dessen, dass die Matrikelnummer als Login-Name zum Einloggen benötigt wird, leichte Startschwierigkeiten, denn der Knopf, um das Feld für den Login-Namen freizugeben, wurde erst spät oder nicht von selbst gefunden.
Beim Provider Gmail gab es laufend Probleme, da sich nicht eingeloggt werden konnte. Vermutlich liegt es an falschen Verbindungsdaten des Providers, die für den Login nicht funktionieren, vom E-Mail-Proxy nach einem Test allerdings als valide eingestuft wurden.

Um den Login für den Nutzer einfacher zu gestalten, sollte erklärt werden, wie das System grundsätzlich funktioniert und man sich einloggt. Ebenfalls sollte auf Besonderheiten wie beispielsweise der TH-Lübeck hingewiesen werden, um künftig Verwirrung zu vermeiden.

Postfächer einrichten

Damit die PWA richtig arbeiten kann, muss der Nutzer festlegen, welche Postfächer nach neuen E-Mails durchsucht werden dürfen. Hierfür gibt es eine eigene Seite, auf welcher alle Postfächer mit einer Checkbox zum Auswählen aufgelistet sind.
Um den Nutzer darüber zu informieren, dass es nötig ist, ein Postfach auszuwählen, wurde ein Banner auf der Startseite platziert. Auf breiten Bildschirmen wurde das Banner zwar wahrgenommen, jedoch nicht weiter beachtet. Einer der Probanden nahm es nicht wahr, da er es für eine Art Werbebanner hielt.
Das Auswählen der Postfächer selbst konnte ohne Probleme durchgeführt werden.

Damit neu eingeloggte Nutzer schneller begreifen, was von ihnen verlangt wird, wäre es sinnvoll, das Banner über den gesamten Bildschirm zu strecken oder direkt nach dem Einloggen auf diese Seite umzuleiten.

Angepinnten Chats beitreten

Um einem Nutzer beispielsweise das Foodsharing oder das Ausleihen von Spielen oder Ähnliches zu erleichtern, wurden vorgefertigte Chats auf der Startseite angepinnt. Die Probanden sollten sich einen Chat aussuchen, welchem sie beitreten wollen.
Die Knöpfe, um einem dieser Chats beizutreten, wurden ohne Probleme gefunden und auch genutzt. Verwirrung kam auf, als nach einem Klick auf den Knopf nichts passierte.
Um einem öffentlichen Chat beizutreten, wird zunächst eine E-Mail an einen Administrator des Chats geschickt, die dieser beantwortet. Dieser Vorgang kann einige Sekunden bis mehrere Minuten dauern, abhängig davon, ober der Administrator aktuell online ist, aber auch von dem Intervall, in dem neue E-Mails gesucht werden. Da dem Nutzer nicht angezeigt wird, dass etwas im Hintergrund passiert, wurde öfter auf den Knopf gedrückt. Dies könnte durch Anzeige einer Nachricht oder mit Lade-Indikatoren verbessert werden. Auch die Information, dass der Nutzer dem Chat beigetreten ist (wenn eine E-Mail vom Administrator zurückgekommen ist) wurde nicht bemerkt, da sich lediglich die Beschriftung des Knopfes ändert. Dies könnte durch farbliches Hervorheben oder durch weitere Benachrichtigungen gelöst werden.

Arbeiten mit Inserat-Chats

Der Umgang mit einem Inserat-Chat (Foodsharing, Kleinanzeigen, ...) lief ohne große Probleme vonstatten. Das Editieren und Löschen von eigenen Inseraten wurde schnell gefunden, ebenso die Funktion, um auf fremde Inserate zu antworten.
Der Knopf zum Erstellen eines neuen Inserats, welcher sich am unteren rechten Rand der PWA befindet, wurde auf großen Bildschirmen schwer gefunden, da er zu weit außerhalb des Sichtfeldes des Nutzers lag. Auf kleinen Bildschirmen beispielsweise von Handys wurde der Knopf schneller wahrgenommen, da er mehr im Sichtfeld des Nutzers liegt.
Das Erstellen des Inserates selbst verlief ohne Probleme und wurde positiv angenommen.

Arbeiten mit Text-Chats

Durch den Aufbau, welcher sich an Text-Chats wie WhatsApp, Signal oder Telegram orientiert, wurde sich hier schnell zurechtgefunden und die Funktionen konnte ohne Probleme genutzt werden.

Erstellen neuer Chats

Ebenso wie beim Erstellen von Inseraten ist der Knopf hierfür bei größeren Bildschirmen zu weit außerhalb des Blickfeldes von Nutzern, weshalb das Finden dieser Funktion ebenfalls länger dauerte.
Beim Erstellen der Chats wurde bemängelt, dass die Eingabefelder nicht eindeutig genug beschriftet sein. Grundsätzlich wurde aber auch diese Funktion ohne große Probleme genutzt.

Meldungen an den Hausmeister

Die Funktion "Meldungen an den Hausmeister" wurde schnell gefunden, da sie oben auf der Startseite platziert wurde. Zur Meldung eines Schadens gibt es einen Auswahlbereich, welcher sich je nach ausgewähltem Schaden verändert und weitere Auswahlmöglichkeiten anbietet. Diese Funktion wurde sehr gut angenommen und für die intuitive Arbeitsweise gelobt.
Aus der Auswahl von Optionen wird die Schadensmeldung generiert, welche an den Hausmeister geschickt werden kann. Teilweise wurden jedoch auch nicht ausgewählte Optionen übernommen, die die Meldung des Schadens verfälscht haben.

Endauswertung

Im Grundsätzlichen wurde die Anwendung gut angenommen und die Konzepte hinter den Funktionen verstanden. Es sind beim Testen noch einige Bugs oder Stellen zum Verbessern aufgefallen, welche auf jeden Fall gefixt werden sollten.
Am Ende eines jeden Testes wurden die Probanden gefragt, ob sie die PWA oder generell eine Anwendung dieser Art nutzen würden. Die Frage wurde dabei grundsätzlich immer bejaht, jedoch wurden dafür Anforderungen gestellt. Den Nutzern ist wichtig, dass die Prozesse, wie und was verarbeitet wird, transparent sind. Zusätzlich sollte beachtet werden, dass die Postfächer der Nutzer aufgeräumt bleiben.

8 Validierung und Nachweis der Anforderungen

Validierung Anforderung 1

> Um zu vermitteln, wie die Anwendung arbeiten soll, muss ein Architekturentwurf angefertigt werden. Aus diesem soll hervorgehen, wie die einzelnen Komponenten miteinander kommunizieren.

Ein Architekturentwurf wurde in Kapitel 4 dargestellt. Dieser umfasst eine Übersicht, welche Komponenten es gibt und wie sie untereinander kommunizieren. Die Funktionsweise wird für jede Komponente einzeln erläutert.
Außerdem wurde in Kapitel 4.4 bereits ein Fazit zum Architekturentwurf aufgestellt.

Validierung Anforderung 2

> Um den Funktionsumfang der Progressive-Web-App (*PWA*) zu bestimmen, muss eine Bedarfsanalyse durchgeführt werden.

Die Bedarfsanalyse (Kap. 3) erfolgte mittelst einer Online-Umfrage.
Die Umfrage hat ergeben, dass durchaus Interesse bei den Bewohnern eines Wohnheims vorhanden ist. Dabei ging hervor, dass es den Bewohner nicht an einer Anwendung zum Chatten mangelt, sondern an einer, die möglichst viele Anwendungsfälle spezifisch für dieses Wohnheim abdeckt. Das Verkaufen, Tauschen oder Leihen von Gegenständen wurde dabei hervorgehoben sowie das Vernetzen unterschiedlicher Interessengruppen.
Die in Kapitel 3.1 genannten Funktionen können mit dem Prototypen der PWA abgebildet werden.

Validierung Anforderung 3

> Um darzulegen, wie die E-Mail als Informationsmittel eingesetzt wird, muss beschrieben werden, was für Daten mit anderen ausgetauscht und wie diese strukturiert werden.

Wie die E-Mail als Informationsmittel in der PWA eingesetzt wird, ist aus Kapitel 5.1 sowie aus Kapitel 5.4 abzulesen. Als erstes wird der allgemeine Aufbau von über die PWA versendeten E-Mails erläutert. Als zweites wird erklärt, wie die Funktionen, welche in Kapitel 3.1 erhoben wurden, funktionieren und welche Inhalte dabei über eine E-Mail ausgetauscht werden.

Validierung Anforderung 4

> Um zu beweisen, dass die erarbeiteten Architekturentwürfe und Konzepte funktionieren, muss ein Prototyp, bestehend aus PWA und E-Mail-Proxy, entwickelt werden.

Es wurden zwei Prototypen entwickelt, einer für den E-Mail-Proxy und einer für die PWA. Die Online-Dokumentation des E-Mail-Proxys ist unter `https://wohnheim-ep.projects.mylab.th-luebeck.de/` und die PWA unter `https://wohnheim-pwa.projects.mylab.th-luebeck.de/` zu finden. Das Git-Repository für den E-Mail-Proxy, die PWA und diese Arbeit ist unter `https://git.mylab.th-luebeck.de/thesis/kratzke/wohnheim-pwa` für zugangsberechtigte Nutzer einsehbar.

(a) QR-Code E-Mail-Proxy (b) QR-Code PWA (c) QR-Code Git-Repository

Abbildung 8.1: QR-Codes zu den Links der Anwendung und des Git-Repository

Der Fokus des E-Mail-Proxys lag darauf, die von der PWA benötigten Teile von *SMTP* und *IMAP* abzudecken. Multi-Part-E-Mails, mit denen beispielsweise Bilder verschickt werden können, wurden nicht umgesetzt, grundsätzlich ist dies aber möglich.
Die Entwicklung der PWA gliederte sich in die Entwicklung des Frontends und des Backends auf. Durch das WebUI-Framework Vue und das State-Management-System VueX wurde die Entwicklungsarbeit deutlich vereinfacht.
Das Auslagern des Backends auf einen Web-Worker ist interessant, erschwert jedoch teilweise das Debugging oder Anpassen des Codes. Für den Prototypen der PWA wäre es nicht zwingend nötig gewesen.

Beide Prototypen (E-Mail-Proxy und PWA) befinden sich in einem recht frühen Stadium der Entwicklung. Es finden sich noch einige Bugs und teilweise ist der Ablauf von Vorgängen in der Oberfläche nicht klar und eindeutig strukturiert. Jedoch zeigen die beiden Prototypen im Zusammenspiel, dass eine Anwendung komplett über E-Mails funktionieren kann.

Von großflächigem Einsatz der PWA ist zum aktuellen Zeitpunkt jedoch abzuraten, da noch große Lücken zum Thema Sicherheit geschlossen bzw. gelöst werden müssten.

Validierung Anforderung 5

> Der Prototyp der PWA soll systematisch evaluiert werden, hierfür sind die Bewohner des Wohnheims sinnvoll einzubeziehen. Wird keine Evaluation durchgeführt, könnte es zu Unverständnissen bei der Benutzung der Oberfläche oder der Technik dahinter führen.

Es wurden insgesamt sechs Usability-Tests mit aktuellen und ehemaligen Bewohnern des Lübecker Studentendorfs durchgeführt. Aus den Ergebnissen (Kap. 7) geht hervor, dass die PWA größtenteils gut strukturiert und nutzbar ist. Jedoch sind einige Fehler aufgefallen, welche einer reibungslosen Nutzung im Wege stehen.

Validierung Anforderung 6

> Es soll ein Konzept für die Sicherheit der Anwendung (beispielsweise Verschlüsselung von E-Mails) erstellt werden.

Das Sicherheitskonzept (Kap. 6) umfasst die Aussortierung von E-Mails, das Verschlüsseln von E-Mails und das Speichern und Benutzen der Zugangsdaten der Nutzer.
Das Aussortieren von E-Mails soll die Nutzer davor schützen, in der PWA ungewollte E-Mails zu bekommen.
In den Prototypen sind die Punkte Verschlüsselung und Handhabung von Nutzerdaten nicht eingeflossen, was die Anwendung unsicher macht. Darüber, dass E-Mails ohne *E2ee* versendet werden, kann hinweg gesehen werden, jedoch ist der Tatsache, dass die Zugangsdaten des Nutzers aktuell unverschlüsselt im Browser liegen, ein großes Problem. Für den produktiven Einsatz muss dieses Problem noch gelöst werden.

9 Fazit

Das Ziel dieser Arbeit ist die Entwicklung einer Webanwendung zur Vernetzung von Bewohnern eines Studentenwohnheims. Die Webanwendung soll dabei nur über E-Mails kommunizieren.
Um E-Mails überhaupt in einen Webbrowser zu bekommen, ist es notwendig, mit E-Mail-Providern kommunizieren zu können. Da moderne Webbrowser aktuell nicht in der Lage sind, direkt über *SMTP* und *IMAP* zu kommunizieren, ist es zwingend nötig, einen Proxy für diese Protokolle zu entwickeln.
Weiterhin wurde eine Progressive-Web-App (PWA) entwickelt, die mittelst des E-Mail-Proxy die Postfächer der Nutzer nach neuen E-Mails durchsucht. Werden neue E-Mails gefunden, lädt der E-Mail-Proxy diese herunter und sendet sie an die PWA, welche diese weiter verarbeitet und schlussendlich dem Nutzer in Form von Chats präsentiert.
Der Funktionssatz der PWA beschränkt sich dabei nicht nur auf reine Text-Kommunikation zwischen zwei oder mehreren Personen, sondern bietet auch die Möglichkeit, hausspezifische Funktionen zu nutzen, wie z. B. dem Hausmeister des Wohnheims Schäden im Haus zu melden. Zudem ist es möglich, über spezielle Chats Kleinanzeigen, Foodsharing oder auch eine Paketsuchzentrale abzubilden. Nutzer eines solchen Chats können neue Dinge inserieren oder beispielsweise die Besitzer bestehender Inserate anschreiben. Ebenfalls bietet die PWA ein Kontaktregister mit wichtigen Kontakten rund um das Wohnheim und die Lübecker Hochschulen und Universitäten.
Alle Inhalte, die über die Anwendung ausgetauscht werden, können über das normale Postfach zwar lesend verfolgt werden, eine Interaktion mit anderen Nutzer ist jedoch nur über die PWA möglich.

Mit dem Prototypen der PWA ist bewiesen, dass eine solche Anwendung nur über E-Mails funktionieren kann. Dabei ist jedoch zu beachten, dass die Kommunikation über E-Mails einen deutlich spürbaren Mehraufwand in der Implementierung verursacht. Dies liegt zum einen an der Pflege eines E-Mail-Proxys als Schnittstelle zu den E-Mail-Providern und zum anderen an der Notwendigkeit, E-Mails vor der Verarbeitung auf gefälschte Header-Felder zu prüfen. Ebenfalls ist eine Synchronisierung von Daten erforderlich, wenn beispielsweise ein neuer Nutzer einem Chat beitritt. Bereits vorhandene Daten müssen dann zunächst von den ursprünglichen Absendern erneut versendet werden.

Mit großflächigerer Verbreitung des Protokolls *JMAP* könnte der E-Mail-Proxy in Zukunft komplett aus der Architektur entfernt werden. Da dieses Protokoll im Gegensatz zu *SMTP* und *IMAP* auf *HTTP* aufbaut, ist es für den direkten Einsatz einer E-Mail-Anwendung im Webbrowser ausgelegt.

Die Prozesse, wie E-Mails in der PWA verarbeitet und Daten gehandhabt werden, könnten in Zukunft durch klarere und durchdachtere Strukturen vereinfacht werden. Es gibt viele Edge-Cases, die in der aktuellen Architektur nicht beachtet oder schlecht gehandhabt wurden. Die Idee, Prozesse auf einen Web-Worker auszulagern, ist grundsätzlich keine schlechte Idee, jedoch sollte geklärt werden, welche Daten in welcher Art und Weise verarbeiten werden.

Um die entwickelte PWA in den produktiven Betrieb zu bringen, sollten die durch den Nutzertest erhaltenen Anregungen eingearbeitet und gefundene Fehler behoben werden. Themen wie die Handhabung von Zugangsdaten der Nutzer und eine Ende-zu-Ende-Verschlüsselung sollten neu überdacht werden.
Sind diese Punkte geklärt und umgesetzt, könnte die PWA in einem größer angelegten Nutzertest erneut überprüft werden. Grundsätzlich steht der Verbreitung in Wohnheimen jedoch kaum etwas im Wege!

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] *Authenticate an IMAP, POP or SMTP connection using OAuth*. Microsoft. URL: https://docs.microsoft.com/en-us/exchange/client-developer/legacy-protocols/how-to-authenticate-an-imap-pop-smtp-application-by-using-oauth (besucht am 27. 01. 2021).

[2] *Autocrypt Convenient End-to-End Encryption for E-Mail*. Autocrypt. URL: https://autocrypt.org/index.html (besucht am 22. 01. 2021).

[3] *Example Data Flows and State Transitions*. Autocrypt. URL: https://autocrypt.org/examples.html (besucht am 22. 01. 2021).

[4] *FastAPI*. tiangolo. URL: https://fastapi.tiangolo.com/ (besucht am 22. 01. 2021).

[5] *flux-concepts*. Facebook. URL: https://github.com/facebook/flux/tree/master/examples/flux-concepts (besucht am 14. 11. 2020).

[6] *OAuth 2.0 Mechanism*. Google. URL: https://developers.google.com/gmail/imap/xoauth2-protocol (besucht am 27. 01. 2020).

[7] *RealWorld example apps*. gothinkster. URL: https://github.com/gothinkster/realworld (besucht am 30. 11. 2020).

[8] *RFC 3501*. Network Working Group. 2003. URL: https://tools.ietf.org/html/rfc3501 (besucht am 11. 11. 2020).

[9] *RFC 4880*. Network Working Group. 2007. URL: https://tools.ietf.org/html/rfc4880 (besucht am 22. 01. 2021).

[10] *RFC 5322*. Network Working Group. 2008. URL: https://tools.ietf.org/html/rfc5322 (besucht am 11. 11. 2020).

[11] *RFC 6749*. Internet Engineering Task Force. 2012. URL: https://tools.ietf.org/html/rfc6749 (besucht am 25. 01. 2021).

[12] *RFC 8620*. Internet Engineering Task Force. 2019. URL: https://tools.ietf.org/html/rfc8260 (besucht am 11. 11. 2020).

[13] *RFC 8887*. Internet Engineering Task Force. 2012. URL: https://tools.ietf.org/html/rfc6648 (besucht am 14. 01. 2021).

[14] *TCP and UDP Socket API*. W3C. 23. Juli 2015. URL: https://www.w3.org/TR/tcp-udp-sockets/ (besucht am 20. 01. 2021).

[15] *TCP Socket API*. Mozilla. URL: https://developer.mozilla.org/en-US/docs/Archive/B2G_OS/API/TPC_Socket_API (besucht am 20. 01. 2021).

[16] William Wambua. *Understanding Flux Architecture*. 23. Nov. 2020. URL: https://medium.com/swlh/understandingFlux-architecture-9060e5a0399c (besucht am 12. 01. 2021).

[17] *What is VueX?* VueJs. URL: https://vuex.vuejs.org/ (besucht am 12. 01. 2021).