Implementierung von Push-Benachrichtigungen für ein Angular-basiertes Intranet Chat System

Praxis II - Projektbericht 2

des Studiengangs Informatik

an der Dualen Hochschule Baden-Württemberg Stuttgart

von

Maximilian Hengl

11.09.2023

Bearbeitungszeitraum	12 Wochen
Matrikelnummer, Kurs	4907952, TINF21E
Ausbildungsfirma	telent GmbH, Backnang
Betreuer	Tobias Leich

Erklärung

Ich versichere hiermit, dass ich meinen Praxis II - Projektbericht 2 mit dem Thema: *Implementierung von Push-Benachrichtigungen für ein Angular-basiertes Intranet Chat System* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Stuttgart, 11.09.2023

Maximilian Hengl

Inhaltsverzeichnis

Abkürzungsverzeichnis

ID	Identifikationsnummer
HTTP	Hypertext Transfer Protocol
CORS	Cross-Origin Resource Sharing
JSON	JavaScript Object Notation
BSON	Binary JSON
MVC	Model View Controller
URL	Uniform Resource Locator
AMQP	Advanced Message Queuing Protocol
npm	Node Package Manager
CLI	Command Line Interface
PWA	progressive Web App
DNS	Domain Name System
CI	Continuous Integration
IP	Internet Protocol
HTML	Hypertext Markup Language
GPO	Group Policy Object
API	Application Programming Interface

Abbildungsverzeichnis

1 Einleitung

Laut einer Datenerhebung des Messengers WhatsApp nutzten im Frebruar 2020 zwei Milliarden Menschen seinen Dienst. WhatsApp ist der meistbenutzte Messenger in Deutschland und seine Nutzerzahlen steigen monoton.

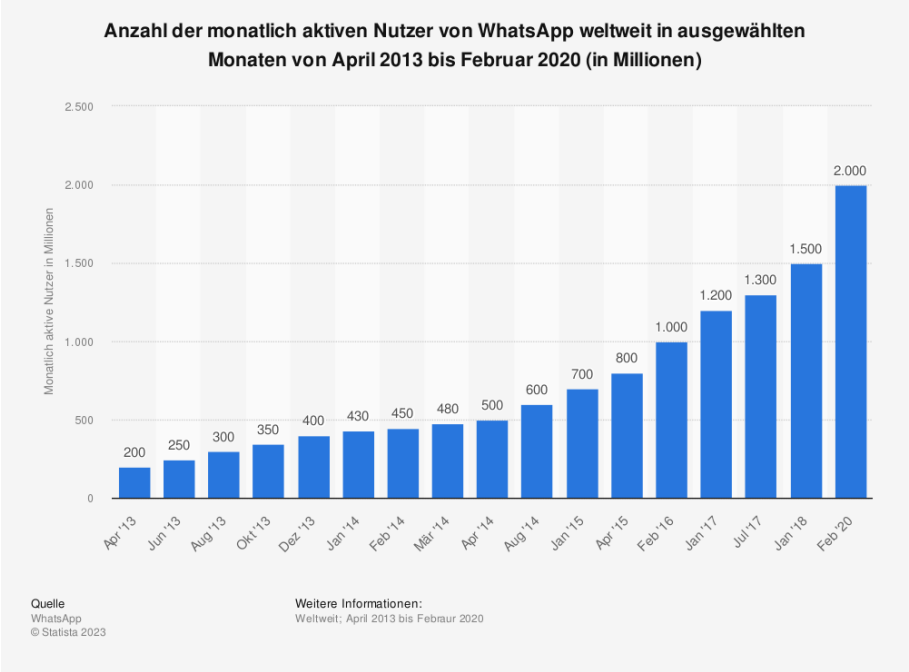

Abbildung 1.1: Anzahl der monatlich aktiven Nutzer von WhatsApp weltweit [14]

Messenger werden immer beliebter, weil sie einen einfachen und schnellen Austausch ermöglichen. Damit dieser Austausch so schnell stattfinden kann, dürfen Nutzer keine neuen Chat-Nachrichten verpassen. Sie werden über sogenannte Push-Benachrichtigungen erinnert, die auf dem Bildschirm ihres Endgeräts erscheinen.

In dieser Arbeit geht es um die Implementierung solcher Benachrichtigungen in ein bestehendes Chat-System. Es werden die Hintergründe und die Umsetzung dieses Projektes erklärt. Außerdem wird die Funktionsweise von Push-Benachrichtigungen dargestellt und der geschriebene Quelltext getestet. Viele der Technologien, die in diesem Projekt verwendet werden, werden genauer erklärt. Am Ende wird die Arbeit in der Entwicklung reflektiert und die Fortführung dieses Projektes beschrieben.

2 Aufgabenstellung und Zielsetzung

In dieser Arbeit sollen Push-Benachrichtigungen in einem bestehenden Chat-System implementiert werden.

2.1 Anforderungen

Für die Arbeit gelten folgende Anforderungen:

- Die Quelltext-Architektur soll geplant werden.
- Es soll möglich viel Quelltext wiederverwendbar sein.
- Er soll leicht zu verstehen sein.
- Push-Benachrichtigungen sollen ein- und ausgeschalten werden können.
- Sie sollen von Nutzern verstanden werden.
- Sie sollen zu dem entsprechenden Chat weiterleiten.

2.2 Mehrwert für das Unternehmen

Die telent GmbH ist ein Unternehmen der kritischen Infrastruktur. Als solches muss sie hohe Standards in den Bereichen Verfügbarkeit und Sicherheit erfüllen [11]. Verwenden Mitarbeiter Messenger wie WhatsApp, kann keine Datensicherheit gewährleistet werden. Als US-amerikanisches Unternehmen ist der Inhaber von WhatsApp dazu verpflichtet, Daten den Sicherheitsbehörden der USA mitzuteilen, wenn diese es als nötig betrachten [12]. Somit können keine sensiblen Informationen über WhatsApp geteilt werden.

Um dennoch sensible Informationen teilen zu können, gibt es die interne Plattform telentWorX. Dort können Dokumente abgelegt werden, Termine ausgemacht werden und vieles mehr. Außerdem gibt es Team- und Projekträume, die nicht für jeden Mitarbeiter zugänglich sind. Als sichere Alternative für eine einfache textuelle Kommunikation gibt es in telentWorX Chat-Räume. Diese werden aber nur selten benutzt, da Nutzer häufig nicht mitbekommen, wenn sie neue Nachrichten bekommen haben.

Stattdessen wird für Kurznachrichten ein Chat benutzt, der nur Rechner-Rechner-Verbindungen zulässt. Er kann Nachrichten nur für die Nutzungsdauer des Chats speichern und sie können nur versandt werden, wenn der Empfänger online ist. Trotz dieser Nachteile wird er dem telentWorX Chat vorgezogen, weil der Empfänger benachrichtigt wird, wenn eine Chat-Nachricht eintrifft.

Durch Push-Benachrichtigungen können Nutzer über neue Chat-Nachrichten informiert werden. Sie müssen dann nicht mehr auf den anderen Chat zugreifen und es gehen keine Informationen mehr verloren.

Wenn die Benachrichtigungen wiederverwendbar sind, können sie auch für abonnierte Dokumente oder Nachrichten und Termine verwendet werden. So kann sichergestellt werden, dass Mitarbeiter Änderungen an wichtigen Dokumenten mitbekommen und sie relevante Termine nicht verpassen.

3 Stand der Technik

3.1 TypeScript

TypeScript ist eine Erweiterung der Skriptsprache JavaScript, die Objektorientierung und Typisierung zulässt. Sie wird zu JavaScript kompiliert und ist dadurch in jedem modernen Browser lauffähig. Jeder gültige JavaScript Code ist auch gültiger TypeScript Code.

Im Gegensatz zu JavaScript Object Notation (JSON) können Klassen in TypeScript auch Konstruktoren und Methoden beinhalten. Datentypen werden wegen der Kompilierung nicht zur Laufzeit überprüft und können sowohl dynamisch als auch statisch sein. Jedes Objekt und jeder Datentyp ist mit den Objekten beziehungsweise Datentypen, die in der Erbhierarchie über ihnen stehen, kompatibel. Der unspezifischste Dateityp ist `any` [2].

3.2 Node

Node ist eine serverseitige JavaScript-Laufzeitumgebung. Anstelle von nebenläufigen Prozessen setzt Node auf asynchrone Ereignisbehandlung. Eine Anwendung registriert ein Interesse an bestimmten Ereignissen. Tritt ein Ereignis ein, wird sie darüber benachrichtigt und kann es über eine sogenannte Rückruffunktion abhandeln. Ereignisse können Mausklicks, Benutzereingaben, aber auch frei gewordener Speicherplatz oder Systembenachrichtigungen sein. Sie werden in einer Warteschlange gespeichert bis sie behandelt werden können. So kann die Anwendung bis dahin anderen Aufgaben nachgehen und wird nicht durch die Behandlung eines Ereignisses blockiert. Durch die ereignisgesteuerte Architektur kann ein Server schnell mit Benutzereingaben umgehen, ohne dass sich der Programmierer Gedanken über die Effekte einer Parallelisierung machen muss. [19]

Über den Node Package Manager (npm) können quelloffene Pakete und Module installiert und deren Versionen verwaltet werden. In dem Manifest `package.json` sind alle Abhängigkeiten gespeichert, die benötigt werden, damit die Anwendung funktioniert. Die Pakete und Module selbst sind im Ordner `node_modules` zu finden. Das Manifest und der Ordner sind dort zu finden, wo sie über die Kommandozeile installiert wurden. Module können über die Funktion `require` benutzt werden. [5][10]

3.3 MongoDB

MongoDB ist ein nicht-relationales Datenbankmanagementsystem. Es kann Daten in Form von Dokumenten speichern und über einen Schlüssel oder den Dokumentinhalt finden. Ein Dokument wird in Form einer Binary JSON (BSON) in eine sogenannte Kollektion geschrieben und folgt keinem vordefinierten Schema. Schreib-, Such- und Löschoperationen können deutlich schneller ausgeführt werden als in relationalen Datenbankmanagementsystemen. [9]

Dafür gibt es verschiedene Gründe. Um ein verschachteltes Objekt zu speichern, müssen in relationalen Datenbanken mehrere Tabellen angelegt werden, die sich über Schlüssel aufeinander beziehen. In einer MongoDB Datenbank kann dieses Objekt als ein Dokument in einer Kollektion gespeichert werden. So werden Schreiboperationen reduziert und Vereinigungen vermieden. Dokumente können über Filter gesucht werden, die ebenfalls als JSON geschrieben werden. [17, S. 137 – 139]

3.4 Docker

Docker ist eine quelloffene Software, die zur Isolierung von Anwendungen in sogenannten Containern dient.

Bei Anwendungen mit vielen Abhängigkeiten kann die korrekte Installation aufwendig sein. Wenn Abhängigkeiten außerdem aktualisiert werden sollen, kann es zu Fehlern kommen, die eine korrekte Ausführung der Anwendung verhindern. Um diese Probleme zu umgehen, können Anwendungen mittels Docker als ausführbare Speicherabbilder verteilt werden. So ein Abbild enthält das Programm selbst und all seine Abhängigkeiten bis zur Tiefe des Betriebssystems.

Die Abbilder werden durch eine Textdatei definiert, in der sowohl Abhängigkeiten als auch Umgebungsvariablen modular notiert sind. Sie kann über Git aktualisiert werden, was eine Versionierung des Abbildes erlaubt. Einzelne Dienste können so aktualisiert werden, ohne das gesamte Abbild verändern zu müssen. Außerdem ist es möglich, neuen beziehungsweise aktualisierten Quelltext zu testen bevor er in die Textdatei eingetragen wird. Für diese Aufgabe kann eine CI-Brücke verwendet werden.

Das Abbild selbst teilt sich den Linux-Kernel mit dem Gastgeberrechner, weshalb dieser über einen Kernel verfügen muss. Dadurch ist ein Abbild effizienter und schmaler als eine virtuelle Maschine, die für den gleichen Zweck genutzt werden könnte. Außerdem ist es relativ einfach, zwei Programme zusammenzuführen. Dafür muss nur die definierende Textdatei geändert werden. Bei einer Virtuellen Maschine müsste zunächst eine Brücke realisiert werden, die zwei virtuelle Maschinen verbindet. Dieser Ansatz ist aufwendig und ineffizient. [3]

3.5 Angular

Angular ist ein Framework für die modulare Web-Entwicklung mit JavaScript oder TypeScript. Es unterstützt das Architekturmuster MVC, bei dem eine Software in die Komponenten Datenmodell, Ansicht und Programmsteuerung aufgeteilt ist. Datenmodell und Ansicht werden hierbei über die Programmsteuerung verbunden, die die Programmlogik enthält. Dadurch können beispielsweise Änderungen an der Darstellung vorgenommen werden, ohne die Daten oder die Logik verändern zu müssen [13].

Der modulare Aufbau von Angular-Projekten erlaubt die Wiederverwendung von Komponenten. Dadurch können Codezeilen reduziert und somit auch Zeit und Speicherplatz gespart werden. Komponenten und Dienste können über das Angular Command Line Interface (CLI) samt aller Abhängigkeiten erzeugt und entfernt werden. [7]

3.6 Progressive Web Apps

Eine progressive Web App (PWA) ist eine Webseite, die als Anwendung installiert werden und teilweise offline arbeiten kann. Die PWA-spezifischen Merkmale werden durch Service Worker realisiert. Diese sind ein neuer Netzstandard, der vor allem von Chrome und auf Chromium basierenden Browsern unterstützt wird. Mögliche Nutzen einer PWA können der Zugriff auf den Systemspeicher, den Medienspieler oder die Anmeldeinformationsverwaltung, die Hintergrundsynchronisierung und das Senden von Push-Benachrichtigungen sein. [16]

Service Worker sind generische, ereignisgesteuerte Skripte, die in einem Browser für eine Webseite registriert werden. Sie werden durch Ereignisse gestartet und am Laufen gehalten [1, motivations]. Eine Registrierung ist ein Tupel aus einem Speicherschlüssel, einem URL, der den Gültigkeitsbereich des Service Workers angibt, und einem Zeitstempel, der die letzte Benutzung des Service Workers angibt. Außerdem hat die Registrierung mindestens eine Warteschlange, in der die Aufgaben des Service Workers gespeichert werden [1, service-worker-registration-concept].

Der Service Worker behandelt verschiedene Ereignisse, wie das Eintreffen einer Nachricht, eines Fehlers oder eine Änderung am Service Worker selbst [1, document-context-events].

3.7 Push-Benachrichtigungen

Push-Benachrichtigungen sind Benachrichtigungen, die direkt auf dem Bildschirm eines Benutzers erscheinen. Sie bestehen aus einem kurzen Text und optional einem Bild. Der Zweck einer Push-Benachrichtigung ist, eine Interaktion des Benutzers mit der App oder Webseite auszulösen, an die der Nutzer weitergeleitet wird, wenn er die Benachrichtigung anklickt [6].

Push-Benachrichtigungen werden zunächst an einen Dienst gesendet, der sie in einer Warteschlange speichert. Dieser versucht sie dann an ein Endgerät zu senden bis sie erfolgreich übertragen oder überschrieben werden. Die meisten Dienste überschreiben Nachrichten nach einer festgelegten Zeit. Seltener werden sie überschrieben, wenn eine neue Nachricht eintrifft, die die selbe App betrifft.

Um eine reaktionsschnelle Punkt-zu-Punkt-Verbindung mit einem Endgerät aufrechtzuerhalten, tauschen der Dienst und das Endgerät Taktnachrichten aus. Wurde eine Benachrichtigung zugestellt, wird die nächste Taktnachricht zeitlich nach hinten verschoben, um Bandbreite einzusparen. Sender und Empfänger authentifizieren sich gegenseitig, um zu verhindern, dass Angreifer Benachrichtigungen mit schädlichem Inhalt senden oder mitlesen. Außerdem kann die Nachricht End-zu-End-verschlüsselt werden. [21]

3.8 telentWorX

telentWorX ist die Intranet-Plattform der telent-GmbH. Auf ihr werden auf sieben verschiedene Arten Informationen geteilt. Es gibt die Kategorien Nachrichten, Termine, Favoriten, Dokumente, Seiten, Vorgänge und Chats.

Nachrichten sind chronologisch geordnet und bestehen aus einem Titel und einem Artikel aus Text und Bildern. Bei Nachrichten und Dokumenten gibt es eine Kommentarfunktion. Sie können für Urlaubsübergaben, als Einladungen zu Versammlungen und Festen oder als kurze Info über Personalveränderungen und Betriebsvereinbarungen genutzt werden.

Termine sind ebenfalls chronologisch geordnet und bestehen aus einem Datum, einer Uhrzeit, einem Titel und einer Beschreibung. Außerdem besteht die Möglichkeit, Termine in den Microsoft Outlook Kalender zu exportieren. Sie werden für Unterweisungen, Seminare und Betriebsversammlungen genutzt. Häufig beinhalten sie auch einen Anmeldelink zu einer Videokonferenz.

Favoriten können frei angeordnet werden und bestehen aus einem Link, einem Titel und einer Kurzbeschreibung. Sie werden genutzt, um schneller zu häufig genutzten Webseiten zu finden. Dazu zählen beispielsweise die Speisekarte der Kantine, der Videokonferenzraum, das Git-Verzeichnis und das Jira-Backlog des Teams sowie die grafischen Benutzeroberflächen für Urlaubsbuchungen, Stundennachweise, Reisekostenabrechnungen und Bestellanforderungen.

Dokumente sind hierarchisch in Ordnern sortiert und bestehen aus allen gängigen Dateitypen. Es kann sich um Dokumentdateien, Tabellen, Präsentationen, Bilder und Videos handeln. Unterstützt werden alle Dateiformate, die auch vom quelloffenen Büro-Software-Paket OnlyOffice unterstützt werden. Außerdem werden die Formate TXT, PDF, JPG, PNG, GIF, MP4, VSD und MSG unterstützt. Die Kategorie Dokumente soll als neuer Speicherplatz die Serverlaufwerke ablösen, auf denen geteilte Dokumente bisher gespeichert wurden. Auf diese ist kein Zugriff möglich, wenn das Laptop nicht per Kabel an das Firmennetz angeschlossen ist.

Seiten sind ebenfalls hierarchisch sortiert und bestehen aus Ober- und Unter-Seiten. Sie können Text, Tabellen, Verlinkungen und Anhänge beinhalten. Sie sind ähnlich aufgebaut wie die Wiki-Software Confluence und werden hauptsächlich für Tipps, Leitfäden, Vereinbarungen und Ideensammlungen genutzt.

Vorgänge sind nach Änderungsdatum sortiert und beinhalten Aufgabenbeschreibungen. Sie können verschiedene Beziehungen zueinander haben, priorisiert, Mitarbeitern zugewiesen und abgeschlossen werden. Sie sind ähnlich aufgebaut wie Aufgaben in der Projektmanagement-Software Jira und werden auch als solche benutzt.

Chats sind nach dem Datum der neusten Chat-Nachricht geordnet und dienen dem einfachen textuellen Austausch unter ausgewählten Personen. Es können Gruppen- aber auch Privatchats erstellt werden. Manche Chats sind Räumen zugewiesen.

Die Plattform telentWorX verfügt außerdem über eine Suchfunktion und ein Berechtigungskonzept. Es gibt verschiedene Räume für Abteilungen, Teams und Projekte. Ein Nutzer kann nur Informationen aus den Räumen sehen, in denen er sich befindet. Die Räume werden von den Entwicklern verwaltet. In den Räumen gibt es die Rollen Raumleitung, Stellvertretung, Assistenz, Mitglied und Gast. Alle Rollen haben verschiedene Berechtigungen. Über die Suchmaschine iFinder können Informationen aller Kategorien gefunden werden. Suchergebnisse können nach Zeitspanne, Dateiart, Ordner, Raum, Autor, Sprache und Schlagwörtern gefiltert werden.

3.8.1 Aufbau

telentWorX ist in verschiedene Services aufgeteilt, die über HTTP oder das WebSocket-Protokoll miteinander kommunizieren. Für dieses Projekt sind der `FrontendSvc`, der `MessageSvc` und der `ChatSvc` relevant. Mit Letzterem wird nur kommuniziert. Veränderungen finden in den anderen beiden Diensten statt. Weil der wesentliche Teil des `MessageSvc` nur die Server-Datei ist, wird er nicht genauer betrachtet. Der ChatSvc wird auch nicht weiter betrachtet, weil an ihm keine Veränderungen vorgenommen werden und er in der weniger geläufigen Programmiersprache Perl geschrieben ist.

Das Angular Frontend ist für eine bessere Übersichtlichkeit in folgenden sechs Kategorien gegliedert:

1-interfaces	Interfaces definieren die Methoden, Eigenschaften und Ereignisse, die von einer Klasse implementiert werden müssen. Sie dienen der Interoperabilität und der Austauschbarkeit von Klassen.
2-services	Dienste sind unabhängige und wiederverwendbare Funktionen. Sie kommunizieren über standardisierte Schnittstellen und können unabhängig vom Rest des Systems aktualisiert werden.
3-dashlets	Dashlets dienen der Darstellung von Informationen. Sie sind in Rastern angeordnet und können wiederverwendet werden.
4-components	Komponenten sind die kleinsten wiederverwendbaren Bausteine einer Website. Sie können verschiedene Funktionen erfüllen.
5-modules	Module sind Sammlungen von Funktionen, Klassen und Variablen, die eine Bestimmte Funktionalität bereitstellen. Sie können Komponenten und Dashlets beinhalten.
6-shared	Unter *shared* wird Quelltext gespeichert, der von mehreren Komponenten oder Diensten genutzt wird.

4 Vorbereitung

4.1 Einrichtung der Entwicklungsumgebung

Bei der Entwicklung wird telentWorX über ein Bash-Skript auf einer virtuellen Maschine ausgeführt. Die Webseite ist dann auf dem Gastgeberrechner erreichbar. In diesem Abschnitt wird die Einrichtung der Entwicklungsumgebung besprochen.

Zunächst wurden die Virtualisierungssoftware VirtualBox, ihre Gasterweiterungen und ein Speicherabbild des Betriebssystems Ubuntu heruntergeladen. Nach Installation der VirtualBox wurde das Abbild als CD in eine neue virtuelle Maschine eingelegt. Diese bekam 10 GB Arbeitsspeicher und zwei Prozessorkerne. Weil das Laptop, auf dem gearbeitet wurde, nur 8 GB Arbeitsspeicher hatte, musste er um 8 GB erweitert werden. Außerdem wurde eine Netzwerkadressübersetzung eingestellt, so dass die von der virtuellen Maschine ausgeführte Webseite auf dem Gastgeberrechner abrufbar ist.

Nach dem Start der Virtuellen Maschine wurde Ubuntu installiert. Dann wurden die Gasterweiterungen eingelegt und alle Treiber für das Betriebssystem installiert. Danach wurde in den Einstellungen der VirtualBox eine gemeinsame Zwischenablage aktiviert, damit Quelltext zwischen dem Gastgeber- und dem Gastrechner kopiert werden kann. Anschließend wurden über die Kommandozeile der Perl-Interpreter, die Containervirtualisierungssoftware Docker, das Orchestrierungssystem Kubernetes, Node, Angular und GNU parallel installiert, das die Nebenläufigkeit von Bash-Skripten ermöglicht. Diese Programme sind alle nötig, um telentWorX ausführen zu können. Bis auf Angular, Node und Docker sind sie allerdings nicht für die Entwicklung in diesem Projekt relevant.

Anschließend wurden ein GitLab Account und ein Token für das Projekt telentWorX erstellt. So konnte das Projekt durch die Versionsverwaltungssoftware Git in einen lokalen Ordner geklont werden. Dann konnten Module über geklonte Bash-Skripte installiert werden. Nach der Einrichtung eines MongoDB-Servers wurde die Datenbank über ein Skript kopiert.

Damit die Webseite auf dem Gastgeberrechner erreichbar ist, musste in der Hosts-Datei, die bei Windows unter `C:\Windows\System32\drivers\etc\hosts` zu finden ist, ein DNS-Alias eingetragen werden.

Nachdem telentWorX lauffähig war, wurden für die Entwicklung noch der Quelltext-Editor Visual Studio Code und Studio 3T, eine grafische Benutzeroberfläche für MongoDB, installiert.

4.2 Struktur des Projektes

In der folgenden Grafik wird die Struktur des Projektes dargestellt. In grüner Schrift sind Dateien zu sehen, die neu erstellt werden. Dateien, die modifiziert werden, sind in brauner Schrift dargestellt.

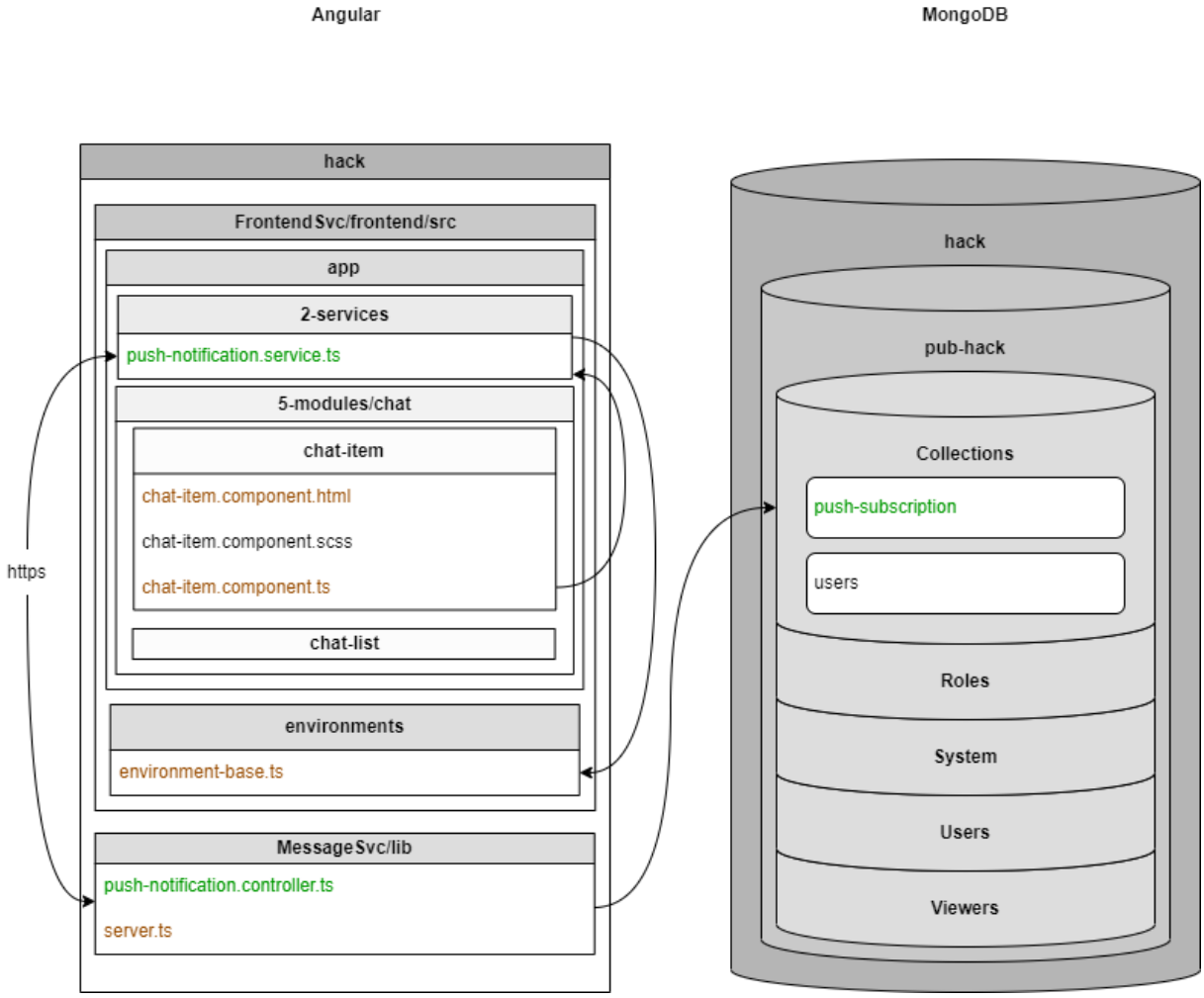

Abbildung 4.1: Grafik zur Veranschaulichung der telentWorX Struktur

5 Umsetzung

Der in diesem Kapitel gezeigte Code ist nicht vollständig. Nebensächliche Bestandteile werden erwähnt aber nicht gezeigt.

5.1 Berechtigung speichern

Im `FrontendSvc` wird über den Befehl `ng generate service push-notification` ein neuer Service erstellt. In dessen Konstruktor werden der HTTP-Client und der Service Worker `SwPush` übergeben. Außerdem wird die Variable `sub` initialisiert, in der die Berechtigung gespeichert wird, die für das Senden von Benachrichtigungen nötig ist. Diese wird vom Service Worker angefragt und muss nicht definiert sein, weil es auch sein kann, dass sie noch nicht existiert.

```
constructor(private swPush: SwPush, private http: HttpClient) {
  this.swPush.subscription.subscribe(sub => this.sub = sub);
}
```

Die Funktion `subscribe` fragt den Browser über den Service Worker nach der Berechtigung. Die Berechtigung muss anschließend in einer Datenbank gespeichert werden. `SwPush` benutzt einen öffentlichen Schlüssel, damit Benachrichtigungen für Angreifer nicht in Klartext sichtbar sind. Dieser muss zusammen mit einem passenden privaten Schlüssel generiert werden.

```
private async subscribe(): Promise<void> {
  this.sub = await this.swPush
  .requestSubscription({ serverPublicKey: this.publicKey });
  this.http.post(this.address, this.sub)
  .subscribe(res => console.log('Subscription successful!', res));
}
```

Über eine Funktion `unsubscribe` kann die Berechtigung wieder zurückgerufen und aus der Datenbank gelöscht werden. Sowohl die Eintragung in die Datenbank als auch die Löschung aus dieser werden im Backend geregelt. Dafür wird die Berechtigung jeweils per HTTP-POST übertragen.

```
private unsubscribe(): void {
  this.http.post(this.address + '/unsubscribe', this.sub).subscribe();
  this.swPush.unsubscribe();
  this.sub = null;
  console.log('Successfully unsubscribed!');
}
```

Damit beide Funktionen über einen Schalter aufgerufen werden können, muss überprüft werden, ob die Berechtigung bereits eingefordert wurde. Dafür gibt es die Funktion `toggle`. Diese entscheidet, ob eine Berechtigung angefordert werden soll oder zurückgenommen werden soll. Außerdem fängt sie mögliche Fehler ab und schreibt sie in die Konsole.

```
toggle(): void {
  try {
    if (this.sub === null) {
      this.subscribe();
    } else {
      this.unsubscribe();
    }
  } catch (error) {
    console.error(error);
  }
}
```

Im Backend `MessageSvc` regelt ein Controller den Zugriff auf die MongoDB. Zunächst muss in einer Klasse festgelegt werden, wie ein Eintrag in die Datenbank aussieht. Es sollen die Benutzer-ID und die Berechtigung gespeichert werden.

```
class PushSubscriptionTable {
  user: ObjectId;
  sub: PushSubscription;
}
```

Außerdem muss eine Kollektion in der MongoDB erstellt werden. Hierfür wird das Programm Studio 3T geöffnet, über das eine Verbindung zur MongoDB hergestellt wird und eine neue Kollektion in der Datenbank `pub-hack` hinzugefügt wird. Diese Kollektion heißt `push-subscription`.

Über die Funktion `setPushSubscription` wird eine HTTP-Post-Anfrage entgegen genommen und der Inhalt in der Kollektion gespeichert.

```
private async setPushSubscription(
  req: Request, res: Response, next: NextFunction
): Promise<void> {
  try {
    const result = await this.collection.insertOne({
      user: req.user._id,
      sub: req.body,
    });
    res.status(200).send({ id: result.insertedId.id });
  } catch (error) {
    next(error);
  }
}
```

Um zu überprüfen, ob die Daten in der Kollektion angekommen sind, wird konventionell die ID des Eintrags zurück an das Frontend gesendet. Kam es zu einem Fehler, wird die Fehlermeldung an das Frontend gesendet. Im Frontend wird diese dann in der Konsole ausgegeben.

Damit der Controller funktioniert, wird im Konstruktor die Datenbank übergeben. Außerdem wird dort über einen Router die Funktion aufgerufen. Dieser Router wird dem bestehenden Server durch den Ausdruck app übergeben.

```
const pushNotification = new PushNotificationController(pubDb);
app.use('/push-notification', pushNotification.router);
```

Damit die HTTP-Anfragen entgegengenommen werden können, muss die Verbindung authentifiziert sein und es müssen CORS-Header definiert sein.

Der Datenbankeintrag sieht folgendermaßen aus:

```
{
  "_id" : ObjectId("112358d1522375990e9971"),
  "user" : ObjectId("2357111311131171d1f25292b"),
  "sub" : {
    "endpoint" : "https://example.com/push/1234",
    "expirationTime" : null,
    "keys" : {
      "p256dh" : "55566666472563445533999",
      "auth" : "a074e1171f1ca71011"
    }
  }
}
```

Er beinhaltet die Eintrags-ID, die Benutzer-ID und die Berechtigung, bestehend aus einem URL, der den Browser identifiziert, einem Auslaufdatum und zwei Schlüsseln. Der Schlüssel mit dem Namen `p256dh` dient der Verschlüsselung der Benachrichtigungen. Der Schlüssel mit dem Namen `auth` dient der Authentifizierung des Senders.

Um den Eintrag wieder zu löschen, gibt es die Funktion `deletePushSubscription`. Sie ist aufgebaut wie die Funktion `setPushSubscription` und versucht, die Berechtigung über den Befehl `findOneAndDelete({ sub: req.body })` des MongoDB-Clients zu suchen und anschließend zu löschen.

Der Schalter in der Chat-Komponente des Frontends überprüft, ob bereits eine Berechtigung existiert und zeigt darauf basierende eine Glocke beziehungsweise eine durchgestrichene Glocke neben dem Schalter an.

```
<app-checkbox-slider [checked]="pushNotification.subscribed()"
(checkedChange)="pushNotification.toggle()">
  <i [class]="'fal fa-fw ' + (pushNotification.subscribed() ?
  'fa-bell-on' : 'fa-bell-slash')"></i> Push-Benachr.
</app-checkbox-slider>
```

5.2 Benachrichtigung senden

Um eine Benachrichtigung zu senden, muss zunächst die Berechtigung aus der Datenbank geladen werden. Hierfür gibt es eine Funktion `getPushSubscriptions` im Controller. Diese filtert die Berechtigungen nach einer Liste an Benutzer-IDs.

```
private async getPushSubscriptions(ids: string[]):
  Promise<PushSubscription[]> {
  const filter: Filter<PushSubscriptionTable> = {};
  filter.user = { $in: ids.map((id): ObjectId => new ObjectId(id)) };
  const subTables = await this.collection.find(filter).toArray()
  as PushSubscriptionTable[];
  return subTables.map((subTable) => subTable.sub);
}
```

Die Funktion gibt eine Liste von Berechtigungen in Form eines Versprechens zurück. Ein Versprechen ist ein gängiger Datentyp bei der asynchronen Programmierung. Es wird verwendet, weil Daten auf dem Übertragungsweg verloren gehen oder korrumpiert werden können.

Bei Funktionsaufruf wird das Versprechen abgewartet. In der Funktion `sendNotification` werden die zurückgegebenen Berechtigungen genutzt, um eine Benachrichtigung an alle adressierten Nutzer mit Berechtigung zu senden. Jede Fehlsendung wird einzeln abgefangen, um einen Fehlschlag der ganzen Funktion zu vermeiden.

```
async sendNotification(userIds: string[], payload: any): Promise<void> {
  const subs = await this.getPushSubscriptions(userIds);
  console.log('GOT SUBSCRIPTION INFORMATION!');
  for (const sub of subs) {
    try {
      await webpush
        .sendNotification(sub, JSON.stringify(payload), this.options);
      console.log('SENT!');
    } catch (error) {
      console.error(error);
    }
  }
}
```

Damit Benachrichtigungen gesendet werden können, wird das Protokoll Web Push benötigt. Dieses wird als Modul über den Node Package Manager (npm) im `MessageSvc` installiert. Dessen Methode `sendNotification` hat die Berechtigung, die Nutzdaten und die Freiwilligen Identifikationsdaten als Parameter. Die freiwilligen Identifikationsdaten bestehen aus dem öffentlichen Schlüssel, der bereits bei der Funktion `subscribe` im Frontend verwendet wurde, dem zugehörigen privaten Schlüssel und einer E-Mail-Adresse.

```
private options = {
  vapidDetails: {
    subject: 'mailto:sender@telent.de',
    publicKey: vapidKeys.publicKey,
    privateKey: vapidKeys.privateKey
  }
};
```

Im Server, wo die Funktion `sendNotification` aufgerufen wird, muss die Nutzlast für die Benachrichtigung festgelegt werden, bevor sie der Funktion übergeben wird. Sie beinhaltet Informationen über den Sender der Chat-Nachricht, den Chat selbst und den Inhalt der Nachricht. Außerdem wird in der Nutzlast festgelegt, was passieren soll, wenn die Benachrichtigung angeklickt wird. Die Chat-Nachricht wird vom Chat-Data-Service im Frontend über das WebSocket-Protokoll an das Backend übergeben und anschließend in einer AMQP-Warteschlange gespeichert. Von dort aus können die Nachrichten gelesen werden.

Zunächst wird der Sender über seine Benutzer-ID aus einer Kollektion in der MongoDB geladen. Die Benutzer-ID kann aus der Chat-Nachricht entnommen werden. Der Titel der Benachrichtigung wird aus dem Namen des Senders und dem Namen des Chats zusammengesetzt.

```
const sentBy = await pubDb.collection<UserDbo>('users')
.findOne({ _id: new ObjectId(message.payload.created_by) }) as UserDbo;
const title = 'Neue Nachricht von ' + sentBy.firstname + ' '
+ sentBy.lastname + ' in "' + message.payload.chatname + '"';
```

Der Körper beinhaltet den Text der Nachricht. Dieser kann direkt verwendet werden. Bevor die Nutzlast erstellt wird, muss noch geregelt werden, was passieren soll, wenn die Benachrichtigung angeklickt wird. Standardmäßig soll sich dann der Chat in einem neuen Tab öffnen. Für den URL zu dem Chat wird die Chat-ID aus der Nachricht entnommen. Der Befehl für die Öffnung des URL wird von einem Service Worker im Frontend abgehandelt.

```
onActionClick = {
  default: {
    operation: 'navigateLastFocusedOrOpen',
    url: 'Kompetenzraum-Intranet/Chats/' + message.payload.chat
  }
};
```

Die Operation `navigateLastFocusedOrOpen` öffnet den Chat in einen neuen Tab, wenn telentWorX in keinem Tab geöffnet ist, navigiert zum Chat, wenn telentWorX offen ist, und aktualisiert den Chat, wenn der Chat offen ist.

In der Konstante `payload` wird die gesamte Nutzlast gespeichert und anschließend versendet. Unter `vibrate` wird eine Vibration für mobile Endgeräte in Millisekunden angegeben. Das Endgerät vibriert in diesem Fall erst 100 ms, dann 50 ms nicht und dann nochmal 100 ms.

```
const payload = {
  notification: {
    title: title,
    body: message.payload.text,
    data: { onActionClick: onActionClick },
    vibrate: [100, 50, 100]
  }
};

try {
  await pushNotification.sendNotification(allowedUsers, payload);
} catch (error) {
  console.error(error);
}
```

6 Tests

6.1 Aufbau der Testumgebung

Um die Funktionalität zu testen wurde eine Umgebung aufgebaut, auf die jedes Teammitglied Zugriff hat. Hierfür mussten die vorgenommenen Änderungen zunächst als Branch in GitLab veröffentlicht werden und erfolgreich die CI-Brücke durchlaufen. Bei einem Fehlschlag konnte in einer Konsole der Auslöser dafür ausfindig gemacht werden und anschließend in einem neuen Commit behoben werden.

Bei der Umgebung handelt es sich um einen Docker Container, der über eine grafische Benutzeroberfläche in telentWorX angelegt werden konnte. Nachdem das Projekt die CI-Brücke überstand, konnte eine neue Domäne im Administratoren-Raum von telentWorX registriert werden. Diese setzte sich aus dem Namen der Anwendung, dem Abteilungsnamen und der Aufgabennummer zusammen. Ihr wurde eine freie IP-Adresse zugewiesen.

Danach wurde die Produktivumgebung dupliziert und dem Duplikat anschließend ein neuer Name und die neue Domäne zugewiesen. Die Datenbank wurde automatisch eingerichtet und von der Produktivumgebung kopiert.

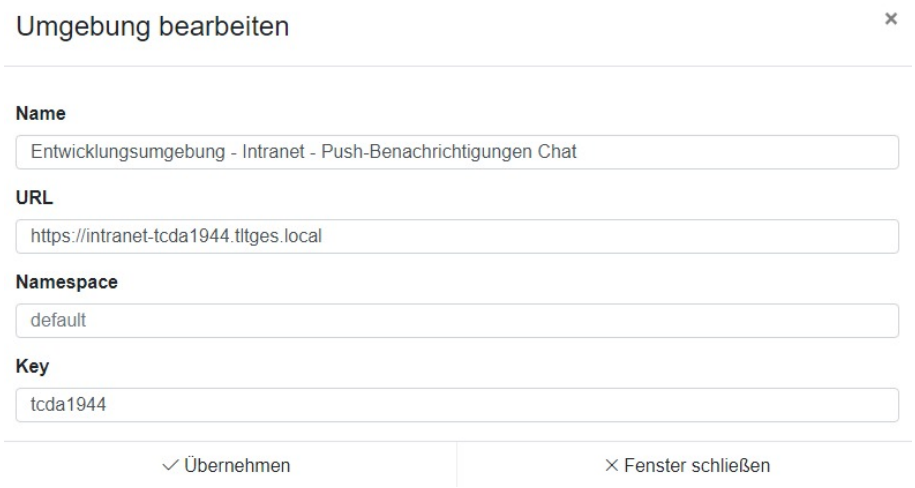

Abbildung 6.1: Grafische Benutzeroberfläche zur Zuweisung einer Domäne

Das Frontend `FrontendSvc` und das Backend `MessageSvc` wurden auf den neuesten Commit des Branches gesetzt, auf dem die in dieser Arbeit vorgenommenen Änderungen liegen. Anschließend wurde die Umgebung veröffentlicht und war unter der neu registrierten Domäne abrufbar.

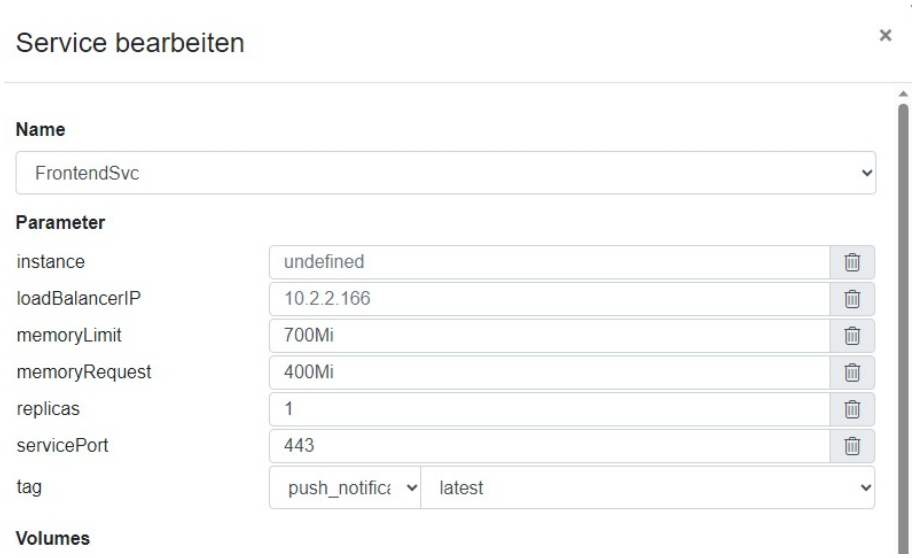

Abbildung 6.2: Grafische Benutzeroberfläche zur Bearbeitung eines Dienstes

6.2 Vorgehen beim Test

Verschiedene Mitarbeiter konnten über die Domäne auf die Testumgebung zugreifen und haben sowohl die Berechtigungsanforderung als auch die Push-Benachrichtigungen selbst in verschiedenen Browsern und auf diversen Geräten getestet.

Zunächst wurde die Berechtigung über das Klicken eines Schalters angefordert. Das hatte bei verschiedenen Browsern andere Auswirkungen. Diese werden im Abschnitt 6.4 besprochen. In der Abbildung 6.3 ist der aktivierte Schalter zu sehen. Die Glocke daneben ist durchgestrichen, wenn der Schalter deaktiviert ist. In der Konsole steht `Subscription successful!`, wenn eine Berechtigung in der Datenbank gespeichert wurde.

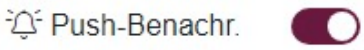

Abbildung 6.3: Schalter für Push-Benachrichtigungen

Danach wurden Nachrichten verschiedener Länge an den Nutzer versandt. So konnte festgestellt werden, wie die Nachrichten in der Benachrichtigung angezeigt werden und ob es

eine Größe gibt, bei der sie nicht mehr angezeigt werden. Außerdem konnten Emojis und Sonderzeichen getestet werden.

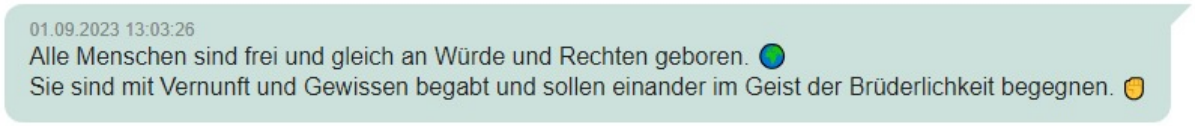

Abbildung 6.4: Chat-Nachricht mit Emojis und Zeilenumbruch

Anschließend wurde die Berechtigung über den Schalter wieder deaktiviert und es wurden erneut Nachrichten versandt. Wenn in der Konsole `Successfully unsubscribed!` stand und keine Benachrichtigungen mehr kamen, war auch die Entfernung der Berechtigung erfolgreich. In der Datenbank sollte die Berechtigung auch wieder entfernt sein.

6.3 Fehlersuche

Die Fehlersuche kann in größeren Projekten anspruchsvoll sein, da viele Elemente innerhalb des Projekts voneinander abhängig sein können. Um Fehler schnell finden zu können, muss der ganze Kommunikationsweg untersucht werden, der beispielsweise bei dem Senden einer Nachricht entsteht. Ein Graph kann helfen, den Kommunikationsweg nachzuvollziehen.

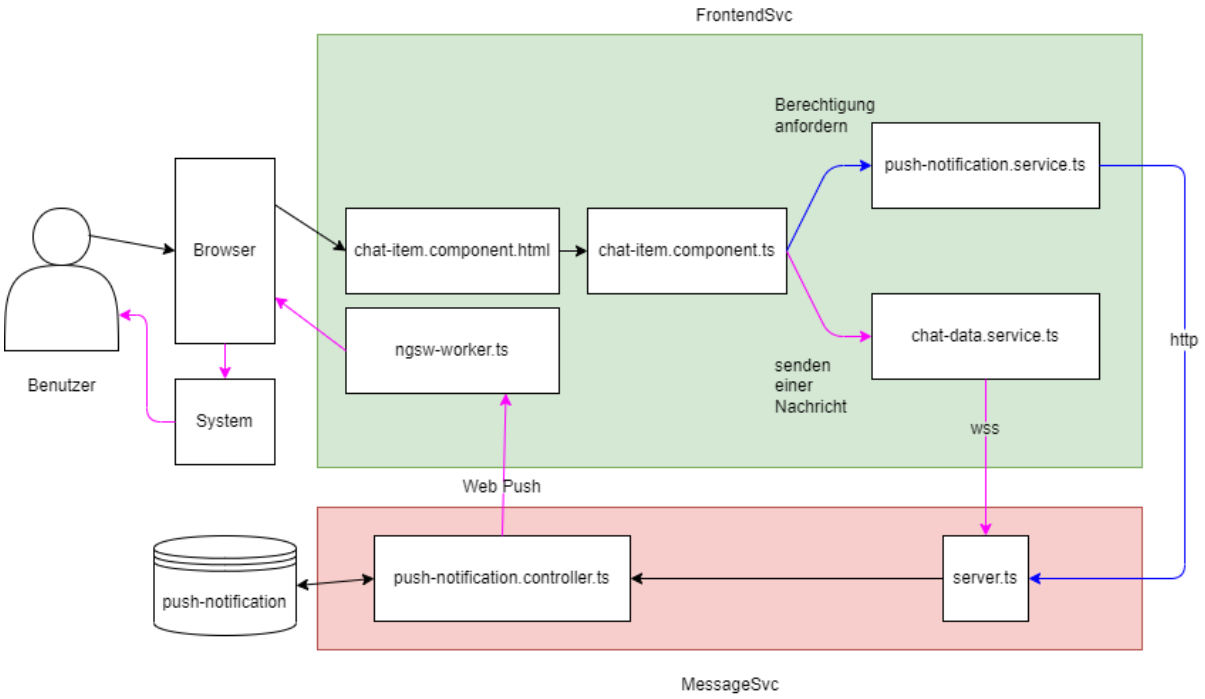

Abbildung 6.5: Graph zur Veranschaulichung des Kommunikationsflusses innerhalb des Projekts

Bei der Anforderung der Berechtigung kann der Browser, die Chat-Komponente, der Benachrichtigungsservice, der Server, der Controller oder die Datenbank eine Fehlerquelle sein. Wird eine Nachricht gesendet, kommen der Service Worker, das Web-Push-Protokoll und das Betriebssystem hinzu. Statt dem Benachrichtigungsservice wird der Chat-Daten-Service genutzt. Auch bei der Kommunikation zwischen Frontend und Backend können Fehler auftreten. Beispielsweise können bei HTTP-Anfragen Probleme durch fehlende CORS-Kopfdaten auftreten.

6.4 Probleme und Verbesserungsvorschläge

In diesem Abschnitt werden Probleme dargestellt, die bei den Tests verschiedenen Nutzern aufgefallen sind. Der weitere Umgang mit diesen Problemen wird in Kapitel 8 besprochen. In Kapitel 7 wird unter anderem die Relevanz der aufgetretenen Probleme erörtert.

6.4.1 Dialogfenster bei Berechtigungsanforderung

Bei der Anforderung der Berechtigung im Browser Edge öffnete sich kein Dialogfenster. Stattdessen erschien eine durchgestrichene Glocke als Symbol in der Navigationsleiste (siehe Abbildung 6.6). Das lag an den adaptiven Anfragen, die Microsoft im Februar des Jahres 2021 eingeführt hat. Dialogfenster werden nur angezeigt, wenn eine große Anzahl an Nutzern die Berechtigung zu gelassen hat. So wolle Microsoft verhindern, dass Nutzer durch unerwünschte Anfragen gestört werden [18].

Abbildung 6.6: Ausschnitt aus der Navigationsleiste von Edge

Bei Klick auf die Glocke öffnete sich dann ein Dialogfenster und Benachrichtigungen konnten zugelassen werden (siehe Abbildung 6.7). In Chrome war es problemlos möglich, Berechtigungen über ein Dialogfenster zu akzeptieren. Auch in der Entwicklungsumgebung hat das funktioniert, obwohl diese ebenfalls in Edge lief. Andere Browser wurden nicht untersucht, da sie im Unternehmen nicht genutzt werden.

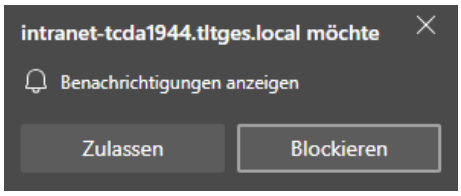

Abbildung 6.7: Dialogfenster nach Anfrage für Push-Benachrichtigungen

6.4.2 Ankunft und Darstellung der Benachrichtigung

In der Benachrichtigung konnten Emojis nach dem Zeichensatz des Betriebssystems angezeigt werden. Zeilenumbrüche und kursive Schrift wurden nach HTML-Syntax angezeigt (siehe Abbildung 6.8). Es gab im Chat keinen Knopf, der das Schreiben kursiver Buchstaben ermöglichte. Außerdem konnten Nachrichten über die Eingabetaste versandt werden. Dennoch war es möglich, einen kopierten Text zu versenden, der Zeilenumbrüche oder kursive Schrift enthält. Diese wurden auch im Chat als solche dargestellt. Bei Privatchats wurde der Name des Empfängers als Chatname angezeigt. Nachrichten aus Gruppenchats konnten diesbezüglich korrekt dargestellt werden.

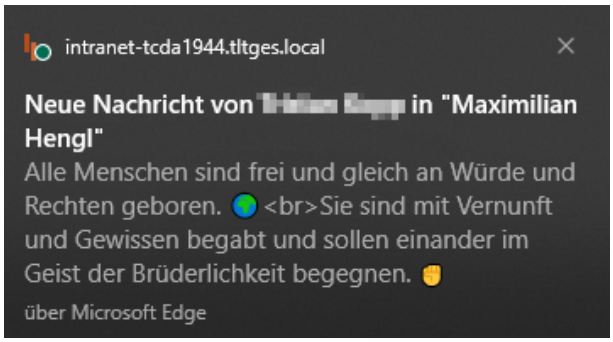

Abbildung 6.8: Eingetroffene Benachrichtigung

Bei über 128 Zeichen wurde der Nachrichtentext abgeschnitten und mit drei Punkten am Ende der Nachricht angedeutet, dass sie noch länger ist. Ab einer bestimmten Nutzlastgröße, die nicht genauer ermittelt werden konnte, wurde keine Benachrichtigung mehr angezeigt.

Auf Mobilgeräten mit dem Betriebssystem Android konnte nie eine Benachrichtigung angezeigt werden. Außerdem konnte telentWorX nicht als App installiert werden. Die Gründe dafür sind unbekannt.

6.4.3 Benachrichtigungen nur aus ausgewählten Chats

Wurden Benachrichtigungen in einem Chat aktiviert, war in jedem Chat des Nutzers innerhalb des Browsers der Schalter umgelegt. Es wurden auch aus jedem Chat Benachrichtigungen empfangen. Das lag daran, dass das Vorhandensein einer Berechtigung über den Service Worker im Frontend überprüft wurde. Die Berechtigungen wurden auch nicht zusammen mit einer Liste an erlaubten Chats gespeichert, nach denen Benachrichtigungen gefiltert werden konnten.

6.4.4 Zeitpunkt der Benachrichtigungen

Benachrichtigungen trafen direkt nach dem Versenden einer neuen Chat-Nachricht ein, ungeachtet dessen, ob der Empfänger bereits den Chat geöffnet hat oder nicht. So kann die Benachrichtigung einen Empfänger aus seinem Arbeitsfluss reißen und daher als störend empfunden werden. Das kann dazu führen, dass Benutzer wieder die Benachrichtigungen ausschalten und neue Nachrichten verpassen [6][15][20]. Um das zu verhindern, können Benachrichtigungen erst gesendet werden, wenn der Empfänger eine Chat-Nachricht, die schon länger eingetroffen ist, noch nicht gelesen hat. Dafür ist eine Lesebestätigung nötig.

7 Diskussion und Schlussfolgerung

In diesem Kapitel werden die Priorität einzelner Probleme und der Aufwand für die Behebung dieser diskutiert. Anschließend wird das Gesamtprojekt analysiert und aus Problemen, die während der Entwicklung aufgetreten sind, werden Schlüsse für die Arbeitsweise in zukünftigen Projekten gezogen.

7.1 Priorität und Lösbarkeit einzelner Probleme

Dadurch dass sich im Browser Edge kein Dialogfenster bei der Anforderung einer Berechtigung öffnet, können unerfahrene Nutzer glauben, die Berechtigung sei schon erteilt. Diese Nutzer werden keine Benachrichtigungen erhalten, weil sie nicht wissen, wie sie die Berechtigung dazu erteilen. Dieses Problem hat eine dementsprechend hohe Priorität. Jedoch ist es programmatisch nicht lösbar. Die einzigen beiden Möglichkeit, das Problem zu umgehen, sind, die Browser-Einstellungen aller Nutzer zu ändern oder die Nutzer über den Umgang damit zu informieren [18]. Die Browser-Einstellungen können von einer internen Abteilung für alle Mitarbeiter festgelegt werden. Hierfür werden sogenannte Group Policy Objects (GPOs) über den GPO-Editor angelegt und im Windows-Verzeichnisdienst Active Directory einer Nutzer- oder Gerätegruppe zugeordnet.

Die Darstellung von Zeilenumbrüchen oder Kursivschrift als HTML-Tags ist nicht von hoher Relevanz, weil sie in der Eingabeleiste des Chats auch nur durch HTML-Tags verwendet werden können. Um das Problem zu beheben müssen die Tags aus der Chat-Nachricht ersetzt werden, bevor der Körper der Nutzlast durch die Chat-Nachricht initialisiert wird. Hierfür ist nur eine Zeile Quelltext nötigt, da die TypeScript-Funktion `string.replace` genau das bereits kann. Diese Funktion bekommt als Attribut einen regulären Ausdruck, nach dem sie sucht, und einen Substring, durch den sie alle Substrings ersetzt, auf die der Ausdruck zutrifft. Der reguläre Ausdruck `/<\/?[^>]+(>|$)/g` repräsentiert alle HTML-Tags eines Strings.

Der Titel eines Privatchats sollte nicht der Name des Empfängers sein. Um dieses Problem zu lösen, reicht es, wenn bei Nachrichten, die nur an einen Nutzer adressiert sind, der Gruppenname weggelassen wird. Das kann realisiert werden, indem die Länge der Liste aus Empfänger-IDs in einer bedingten Anweisung überprüft wird. Ist sie länger als eins, wird der Gruppenname dem

Titel der Benachrichtigung hinzugefügt. Die Lösung dieses Problems bedarf also drei Zeilen Quelltext.

Um zu vermeiden, dass Benachrichtigungen eine zu hohe Nutzlast haben, kann der Körper einer Benachrichtigung gekürzt werden. Da nur 128 Zeichen angezeigt werden, wäre es sinnvoll, den Körper nur durch die ersten 128 Zeichen der Chat-Nachricht zu initialisieren. Hierfür kann die Methode `string.substring` verwendet werden. Sie gibt einen Substring zurück, der durch seinen Anfangs- und Endindex im String definiert wird. Weil alle Benachrichtigungen auch kommen sollen, hat dieses Problem eine hohe Priorität. Für seine Lösung muss nur eine Zeile Quelltext geändert werden.

Mitarbeiter sollten auch auf ihren Geschäftstelefonen benachrichtigt werden, wenn eine neue Chat-Nachricht eintrifft. Der Browser Chrome braucht in älteren Versionen einen Google Cloud Messaging API-Schlüssel. Ebenso der Samsung Internet Browser. Dieser Schlüssel ist in der Google Entwicklerkonsole erhältlich [8]. Dort kann ein neues Projekt angelegt werden, nachdem ein Nutzerkonto erstellt wurde. Innerhalb des Projekts kann unter `Anmeldedaten` > `Anmeldedaten erstellen` > `API-Schlüssel` ein neuer Schlüssel generiert werden [4]. Dieser kann dann dem Objekt `options` im Controller des `MessageSvc` als `gcmAPIKey` übergeben werden (vgl. Abschnitt 5.2). Benachrichtigungen auf Geschäftstelefonen sind also nur möglich, wenn die telent GmbH dafür ein Google-Konto einrichtet.

Damit Benachrichtigungen nur aus ausgewählten Chats angezeigt werden, muss in der MongoDB Kollektion zu jeder Berechtigung auch eine Liste an Chats gespeichert werden, für die diese Berechtigung gültig ist. Diese Liste kann dann aktualisiert werden, wenn ein neuer Chat Benachrichtigungen senden darf. Eine neue Berechtigung darf nur angefragt werden, wenn noch keine Berechtigung existiert. Ob eine Berechtigung für diesen Chat existiert oder nicht, darf im Frontend nicht mehr nur über den Service Worker abgefragt werden. Existiert bereits eine Berechtigung für den Benutzer, muss das Backend per HTTP gefragt werden, ob die Berechtigung auch für diesen Chat gilt. Dafür müssten Berechtigungen in der Funktion `getPushSubscriptions` zusätzlich nach dem Chat gefiltert werden. Die Funktion `sendNotification` müsste dann ebenfalls einen Chat als Attribut haben, um die Berechtigungen abfragen zu können (vgl. Abschnitt 5.2). Dadurch wäre der Controller nicht mehr für andere Push-Benachrichtigungen wiederverwendbar. Außerdem könnten Nachrichten in neu erstellten Chats verpasst werden, weil keine Benachrichtigung dafür ankam.

Damit eine Benachrichtigung nur versendet wird, wenn der Empfänger die betreffende Chat-Nachricht noch nicht gelesen hat, müssen die Empfänger nach Lesebestätigungen ge-

filtert werden. Hierfür kann nach dem Eintreffen einer Chat-Nachricht die JavaScript Methode `setTimeout` verwendet werden. Sie führt eine als Parameter übergebene Funktion nach einem in Millisekunden übergebenem Zeitraum aus. Im Frontend gibt es im Dokument `chat-data.service.ts` eine Funktion, die neue Chat-Nachrichten lädt, wenn ein Benutzer den Chat öffnet. Diese Funktion könnte die ID des Nutzers an das Backend übergeben. Die IDs können dort in einer Liste gespeichert werden. Nach dem abgelaufenen Zeitraum können die Nutzer-IDs dieser Liste aus der Liste aller Empfänger über die Methode `array.filter` herausgefiltert werden. Die übrig gebliebenen Empfänger können an die Funktion `sendNotification` übergeben werden (vgl. Abschnitt 5.2). Das Problem kann also gelöst werden, ohne die Wiederverwendbarkeit der Push-Benachrichtigungen zu behindern. Allerdings findet eine Kommunikation zwischen Frontend und Backend statt, was einen erhöhten Entwicklungsaufwand bedeutet.

7.2 Arbeitsweise

Die Entwicklungsumgebung konnte dank guter Dokumentation größtenteils selbst eingerichtet werden. Dennoch kam es zu Problemen wegen falscher Versionen mancher Softwarepakete. Diese konnten in Zusammenarbeit mit dem Betreuer behoben werden.

Bei der Einarbeitung in das Projekt war vieles über den Aufbau von telentWorX, den Umgang mit MongoDB aber auch die Funktionsweise eines Node-Servers unklar. Es wurde viel Zeit damit verbracht, die Kommunikation zwischen einzelnen Funktionen nachzuvollziehen. Der erste Umgang mit Unklarheiten bestand darin, vorhanden Quelltext zu analysieren. Wenn das nicht gereicht hat, um Funktionsweisen zu verstehen, wurde im Internet zu den Problemen recherchiert.

Am effektivsten war es, bei Unklarheiten über fremden Quelltext den Entwickler zu fragen. War der Entwickler beschäftigt, konnte über den Quelltexteditor analysiert werden, wie bestimmte Funktionen verwendet wurden. Bei importiertem Quelltext konnte über Rechtsklick die Definition der importierten Funktion beziehungsweise des Objekt betrachtet werden. Außerdem konnten Implementierungen des Quelltextes analysiert werden. Über die Suchfunktion konnten auch Funktionen gefunden werden, die über HTTP miteinander kommunizieren.

Über die Benutzeroberfläche Studio 3T konnten Kollektionen in der MongoDB durchsucht werden. So konnte eine Möglichkeit gefunden werden, an benötigte Daten wie Benutzernamen zu

kommen. In einer Readme-Datei wurde Dokumentiert, wie Befehle der MongoDB zu benutzen sind.

Durch tägliche Videokonferenzen mit der ganzen Abteilung konnten Probleme bei der Entwicklung benannt und später gelöst werden. Außerdem hat ein Einblick in den Alltag anderer Entwickler das Verständnis für die Arbeit der Abteilung gefördert. Durch den Abteilungsleiter wurden Anforderungen von Nutzern an telentWorX kommuniziert, was eine andere Perspektive auf das Projekt ermöglichte.

Während der Entwicklung der Benachrichtigungsfunktion fiel auf, dass Funktionen mit viel Quelltext dazu neigen, unübersichtlich zu werden. Deshalb besteht beispielsweise der Controller aus mehreren kleinen Funktionen, deren Nutzen durch den Namen der Funktion beschrieben ist. Auch Variablen und Konstanten sind weitestgehend eindeutig benannt.

Die Verwendung von `then` nach dem Aufruf einer asynchronen Funktion führt zu einer Verschachtlung. Um stark verschachtelte Funktionen zu verhindern, ist es besser die Funktion als asynchron zu deklarieren und Funktionsaufrufe innerhalb mit `await` abzuwarten. Die gesamte Funktion kann später in einer einzigen Ausnahmebehandlungsanweisung aufgerufen werden. So kann die Komplexität des Quelltextes reduziert werden.

8 Zusammenfassung und Ausblick

Als Unternehmen der kritischen Infrastruktur braucht die telent GmbH eine vor Datendiebstahl sichere Kommunikation unter Mitarbeitern. Diese wird durch einen Chat in der Intranet-Plattform telentWorX garantiert. Dieser Chat wurde allerdings kaum benutzt, weil Mitarbeiter darin geteilte Nachrichten nicht wahrnahmen. Push-Benachrichtigungen können Nutzer auf neue Chat-Nachrichten aufmerksam machen und so dafür sorgen, dass der Chat häufiger benutzt wird.

In dieser Arbeit sollten Push-Benachrichtigungen in dem Chat der Plattform telentWorX implementiert werden. Diese sollten ein- und ausgeschalten werden können, von Nutzern verstanden werden und zu einem Chat weiterleiten. Außerdem sollte der Quelltext möglichst wiederverwendbar und leicht zu verstehen sein.

Für die Arbeit wurde die Programmiersprache TypeScript verwendet. Diese lässt als Erweiterung der Skriptsprache JavaScript Objektorientierung und Typisierung zu. Sie wird zu JavaScript kompiliert und ist daher mit allen Programmierschnittstellen für JavaScript kompatibel [2].

Push-Benachrichtigungen sind Benachrichtigungen, die auf dem Bildschirm eines Endgerätes erscheinen. Sie werden vom Backend an einen Dienst vermittelt, der diese dann wiederum an ein Endgerät sendet [21]. Dort werden sie von einem Skript behandelt, das im Browser registriert wurde. Dieses Skript wird Service Worker genannt [1].

Im Frontend des Projektes wurde das modulare Framework Angular verwendet [7]. Dort wurde von einem Service Worker ein Objekt angefordert, das das senden von Push-Benachrichtigungen erlaubt. Dieses wurde dann an das Backend geschickt, wo es als BSON in einer Kollektion des Datenbankmanagementsystems MongoDB gespeichert wurde [17]. Das Backend bestand aus einem WebSocket-Server, der Chat-Nachrichten in eine Warteschlange überführte.

Anschließend konnten Push-Benachrichtigungen aus dem Inhalt der Chat-Nachrichten erstellt und über das Protokoll Web-Push versendet werden. Web-Push wurde über den Node Package Manager (npm) installiert und über den Node-Befehl `require` implementiert [10].

Um Push-Benachrichtigungen wieder auszuschalten wurde die Berechtigung im Frontend über den Service Worker wieder zurückgerufen. Im Backend wurde sie aus der Kollektion gelöscht.

Um das Projekt zu testen wurde ein Docker-Abbild erstellt, einer Domäne zugewiesen und ausgeführt. In einer Textdatei sind alle Abhängigkeiten des Projektes gespeichert. Diese definiert das Abbild und konnte über eine Benutzeroberfläche bearbeitet werden [3]. Über die Domäne konnten Mitarbeiter auf eine Testumgebung von telentWorX zugreifen, in der Push-Benachrichtigungen implementiert wurden.

Beim Testen der Benachrichtigungen fielen Darstellungsprobleme und funktionale Fehler auf. So wurde im Browser Edge vorerst kein Dialogfenster angezeigt, indem Benachrichtigungen akzeptiert oder blockiert werden konnten. Für die Behebung dieses Fehlers kann mit einer zuständigen Abteilung besprochen werden, ob die Einstellungen des Browsers für alle Mitarbeiter geändert werden sollen.

In der Benachrichtigung selbst wurden HTML-Tags angezeigt, Privatchats wurden falsch dargestellt und die Nutzlast der Benachrichtigung wurde überschritten. Diese Fehler werden in naher Zukunft behoben, weil sie nicht viele Kapazitäten brauchen.

Ein funktionaler Fehler ist die fehlende Darstellung von Benachrichtigungen auf mobilen Endgeräten. Da für die Lösung dieses Problems ein Google Konto nötig ist, muss erst geklärt werden, ob das für das Unternehmen problematisch wäre.

Um Benachrichtigungen nur aus ausgewählten Chats senden zu können, muss der Quelltext erheblich verändert werden, wodurch unter anderem die Wiederverwendbarkeit der Push-Benachrichtigungen verloren geht. Daher wird dieser Ansatz in nächster Zeit nicht verfolgt.

Damit Benutzer nur über ungelesene Chat-Nachrichten benachrichtigt werden, muss ebenfalls viel Quelltext verändert werden. Diese Neuerungen ändern aber nichts an der Wiederverwendbarkeit des Quelltextes. Weil Mitarbeiter möglichst nicht durch die Benachrichtigungen von ihrer Arbeit abgelenkt werden sollen, ist eine Umsetzung dieses Ansatzes bereits in Planung.

Literatur

[1] Jake Archibald u. a. *Service Workers: W3C Candidate Recommendation Draft*. Einsicht am 29.08.2023. 2022. URL: https://www.w3.org/TR/2022/CRD-service-workers-20220712.

[2] Gavin Biermann, Martin Abadi und Mads Torgersen. *Understanding TypeScript*. Uppsala, Schweden, 2014. DOI: 10.1007/978-3-662-44202-9_11.

[3] Carl Boettiger. *An introduction to Docker for reproducible research*. Santa Cruz, Vereinigte Staaten von Amerika, 2015. DOI: 10.1145/2723872.2723882.

[4] Google Cloud. *API-Schlüssel erstellen*. Einsicht am 05.09.2023. URL: https://cloud.google.com/docs/authentication/api-keys?hl=de#create.

[5] Devlin Basilan Duldulao und Ruby Jane Leyva Cabagnot. *Getting Started with the Node Package Manager*. Berkeley, Vereinigte Staaten von Amerika, 2021. DOI: 10.1007/978-1-4842-6975-6_2.

[6] Diana Gavilan und Gema Martinez-Navarro. *Exploring user's experience of push notifications: a grounded theory approach*. Madrid, Spanien, 2022. DOI: 10.1108/QMR-05-2021-0061.

[7] G. Geetha u. a. *Interpretation and Analysis of Angular Framework*. Chennai, Indien, 2022. DOI: 10.1109/ICPECTS56089.2022.10047474.

[8] GitHub. *Dokumentation zu web-push*. Einsicht am 05.09.2023. URL: https://github.com/web-push-libs/web-push#readme.

[9] Cornelia Győrödi u. a. *A comparative study: MongoDB vs. MySQL*. Oradea, Rumänien, 2015. DOI: 10.1109/EMES.2015.7158433.

[10] npm Inc. *About packages and modules*. Einsicht am 29.08.2023. URL: https://docs.npmjs.com/about-packages-and-modules.

[11] Bundesamt für Justiz. *§ 8a BSIG: Sicherheit in der Informationstechnik Kritischer Infrastrukturen*. Berlin, Deutschland, 2009. URL: https://www.gesetze-im-internet.de/bsig_2009/__8a.html.

[12] Europäische Kommision. *EU-U.S. Data Privacy Framework*. Brüssel, Belgien, 2023. URL: https://ec.europa.eu/commission/presscorner/api/files/attachment/875613/EU-US%20Data%20Privacy%20Framework.pdf.pdf.

[13] Abraham Leff und James Rayfield. *Web-application development using the Model/View-/Controller design pattern*. Seattle, Vereinigte Staaten von Amerika, 2001. DOI: 10.1109/EDOC.2001.950428.

[14] L. Lohmeier. *Anzahl der monatlich aktiven Nutzer von WhatsApp weltweit in ausgewählten Monaten von April 2013 bis Februar 2020*. Einsicht am 05.09.2023. 2023. URL: https://de.statista.com/statistik/daten/studie/285230/umfrage/aktive-nutzer-von-whatsapp-weltweit/.

[15] Conor O'Brien u. a. *Should I send this notification? Optimizing push notifications decision making by modeling the future*. San Francisco, Vereinigte Staaten von Amerika, 2022. DOI: 10.48550/arXiv.2202.08812.

[16] Thomas Steiner. *What is in a Web View: An Analysis of Progressive Web App Features When the Means of Web Access is not a Web Browser*. Lyon, Frankreich, 2018. DOI: 10.1145/3184558.3188742.

[17] Vasan Subramanian. *MongoDB*. Berkeley, Vereinigte Staaten von Amerika, 2019. DOI: 10.1007/978-1-4842-4391-6_6.

[18] Microsoft Edge Team. *Introducing adaptive notification requests in Microsoft Edge*. Einsicht am 01.09.2023. 2021. URL: https://blogs.windows.com/msedgedev/2021/02/16/introducing-adaptive-notification-requests-in-microsoft-edge/.

[19] Stefan Tilkov und Steve Vinoski. *Node.js: Using JavaScript to Build High-Performance Network Programs*. Nikosia, Zypern, 2010. DOI: 10.1109/MIC.2010.145.

[20] Vanessa Vella und Chris Porter. *Wait a Second! Assessing the Impact of Different Desktop Push Notification Types on Software Developers*. Kaiserslautern, Deutschland, 2022. DOI: 10.1145/3552327.3552328.

[21] Ian Warren u. a. *Push Notification Mechanisms for Pervasive Smartphone Applications*. Auckland, Neuseeland, 2014. DOI: 10.1109/MPRV.2014.34.