

BACHELORARBEIT

Analyse der Auswirkung von Progressive Web Apps auf bestehende Web Apps

durchgeführt am
Studiengang Informationstechnik & System-Management
an der
Fachhochschule Salzburg GmbH

vorgelegt von
Refik Kerimi



Studiengangsleiter: FH-Prof. DI Dr. Gerhard Jöchl
Betreuer: DI Norbert Egger BSc

Salzburg, September 2018

Eidesstattliche Erklärung

Ich versichere an Eides statt, dass ich die vorliegende Bachelorarbeit ohne unzulässige fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt und alle aus ungedruckten Quellen, gedruckter Literatur oder aus dem Internet im Wortlaut oder im wesentlichen Inhalt übernommenen Formulierungen und Konzepte gemäß den Richtlinien wissenschaftlicher Arbeiten zitiert, bzw. mit genauer Quellenangabe kenntlich gemacht habe. Diese Arbeit wurde in gleicher oder ähnlicher Form weder im In- noch im Ausland in irgendeiner Form als Prüfungsarbeit vorgelegt und stimmt mit der durch die Begutachter beurteilten Arbeit überein.

Salzburg, am 1.09.2018



Refik Kerimi

1410555043

Matrikelnummer

Allgemeine Informationen

Vor- und Zuname:	Refik Kerimi
Institution:	Fachhochschule Salzburg GmbH
Studiengang:	Informationstechnik & System-Management
Titel der Bachelorarbeit:	Analyse der Auswirkung von Progressive Web Apps auf bestehende Web Apps
Schlagwörter:	Progressive Web App, Manifest.json, Service Workers, Home Banner, Offline
Betreuer an der FH:	DI Norbert Egger BSc

Kurzfassung

Abstract

Danksagung

Danken möchte ich vor allem meinem Betreuer für die Unterstützung bei dieser Bachelorarbeit.

Besonderer Dank gilt auch meiner Familie und Freunden, die uns während des Studiums in allen Belangen immer unterstützt haben.

Inhaltsverzeichnis

Abkürzungsverzeichnis	i
Abbildungsverzeichnis	ii
Tabellenverzeichnis	iii
Listingverzeichnis	iv
1 Einleitung	1
1.1 Motivation	1
1.2 Zielsetzung	2
2 Grundlagen	3
2.1 Geschichte Softwareentwicklung	4
2.2 Native Apps	4
2.3 Web Applikationen	4
2.4 Hybrid Applikationen	4
2.5 Progressive Webapplikationen	5
3 Features und Merkmale	6
3.1 Aufbau Progressive Web Apps (PWA)	6
3.2 Unterschiede PWA, Native Applikation und Web-Apps	6
3.3 Web App Manifest	8
3.4 Add to Homescreen	9
3.5 Service Worker	11
3.6 Push Notifikation	14
3.7 Geolocation API	16
4 Implementierung	17
4.1 Anforderungsanalyse	17
4.2 Umsetzung der Anforderungen	17
4.3 Ausgewählte Programmiersprache und IDE	17
4.4 Ordnerstruktur	18
4.5 Manifest	18
4.6 Add to Homescreen	20
4.7 Service Worker und Cache API	20

4.8	Offline Modus	23
4.9	Push Notifications	25
4.10	Geolocation API	26
5	Funktionstest/Validierung	27
5.1	Ausgangsbedingung und Ausgrenzung	27
5.2	Testen auf Mobilten Geräten und Android Studio Emulator	27
5.3	Lighthouse	28
5.4	Add to Homescreen	30
5.5	Service Worker	31
5.6	Push Notifikation	32
5.7	Geolocation	33
5.8	Vergleich mit Native App	34
6	Zusammenfassung und Ausblick	35
	Literatur	36

Abkürzungsverzeichnis

PWA	Progressive Web Applikation
Web-App	Web Applikationen
JS	JavaScript
JSON	JavaScript Object Notation
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets

Abbildungsverzeichnis

2.1	Smartphonennutzung [1]	3
3.1	PWA Komponenten	6
3.2	Kompabilität Manifest.json [2]	10
3.3	Service Worker als Proxy [3]	11
3.4	Registrierung Service Worker	12
3.5	Registrierung Service Worker	13
3.6	Erstinstallation Service Worker [4]	13
3.7	Kompabilität Service Worker [2]	14
3.8	Kompabilität Push Notifikation [2]	15
3.9	Kompabilität Geolocation [2]	16
4.1	Ordner Struktur	18
4.2	Cache	23
4.3	Datenaufruf Service Worker	24
4.4	Konsolenmeldung Geolocation	26
5.1	Aktivieren der Entwicklertools auf Android 8.1.0	27
5.2	Anzeige der Verbindung auf Google Chrome 67	28
5.3	Lighthouse Plugin Chrome 67	29
5.4	Debugging	29
5.5	Lighthouse Überblick	29
5.6	Lighthouse fehlende PWA-Features	29
5.7	Add to Homescreen	30
5.8	Service Worker Status	31
5.9	Offline	31
5.10	Registrierung Push Notification	32
5.11	Push Notification	32
5.12	Registrierung Geolocation	33
5.13	Standortermittlung	33

Tabellenverzeichnis

3.1	Veröffentlichung und Installation [5]	6
3.2	Zugriff [5]	7
3.3	Funktionen [5]	7

Listings

3.1	Manifest.json [6]	8
3.2	Manifest in das Projekt implementieren [6]	8
3.3	beforeinstallpromptEvent [7]	9
3.4	Service Worker Navigator [8]	11
3.5	Service Worker Register [4]	12
4.1	Manifest in das Projekt implementieren [6]	19
4.2	Add to Homescreen Funktion	20
4.3	Registrierung	20
4.4	Installation	21
4.5	Aktivierung	21
4.6	fetch Calbackfunktion	22
4.7	Cache	23
4.8	Geolocation Support [9]	25
4.9	Geolocation Support [10]	26

1 Einleitung

Durch die Markteinführung des Smartphones hat sich unser Leben gravierend geändert. Nicht nur unsere Kommunikation, sondern unser Leben im Allgemeinen, ist durch dieses Gerät erleichtert worden. Das Smartphone, Tablet und der PC sind ständig im Einsatz um Informationen abzurufen, Musik zu hören, Kontakte zu pflegen, zu telefonieren etc. Kurz nach der Erfindung des smarten Handys kam ein weiterer Markt hinzu, der sich parallel dazu entwickelt hat. Es wurden neue Berufe gegründet wie z.B.: der Native App Entwickler. Native Apps werden speziell an das Betriebssystem angepasst und können somit im Gegensatz zu einer Standard Web Applikation die Ressourcen eines Mobilen Gerätes optimal nutzen.

In der heutigen Zeit müssen Informationen jederzeit und überall für den Benutzer verfügbar sein. Unternehmen werden dadurch immer mehr gefordert, da ihre Anwendungen für jedes Gerät unabhängig von dem Betriebssystem, der Bildschirmgröße etc. funktionieren müssen. Das Ganze benötigt eigene Entwickler die sich auf die jeweiligen Plattformen spezialisieren und auch dadurch zu höheren Entwicklungskosten führen. Um Kosten zu reduzieren und die Entwicklungen zu vereinheitlichen startete Google ein neues Konzept, die Progressive Web Applikation. Diese Technologie soll es ermöglichen, dass sich Web-Anwendungen anfühlen wie Native Anwendungen.

1.1 Motivation

Wie im vorigen Kapitel beschrieben werden native Applikationen für ein bestimmtes Betriebssystem optimiert. Diese haben den Vorteil die Hardware des Gerätes nutzen zu können und somit sind komplexere Anwendungen realisierbar. Doch diese sind für das gesamte Projekt kostspieliger, da im Gegensatz zu den Web-Anwendungen für jedes System eigene Entwickler benötigt werden. Die Progressive Web Applikation(PWA)soll die Vorteile von nativen Apps und von Webanwendungen vereinen und dem Nutzer ein Gefühl geben, dass man es mit einer auf das System angepassten Anwendung zu tun hat.

1.2 Zielsetzung

Das Ziel ist es, mit Hilfe eines Smart Home App Prototypen die Unterschiede und Auswirkungen einer PWA auf bestehende Web-Apps zu untersuchen. Dem Prototypen werden die PWA typischen Features wie das Hinzufügen auf dem Startbildschirm, Offline arbeiten, die Pushfunktionen, das Zugreifen auf Gerätefunktionen hinzugefügt. Während dieser Arbeit werden an diesen Features die Vorteile, Nachteile, Entwicklung, Betrieb und User Experience einer Progressive Web Applikation betrachtet. Basis Technologien der Webentwicklung und verwendete Frameworks (z.B.: JavaScript(JS),ReactJS, NodeJS, Yarn,...) werden in dieser Arbeit nicht behandelt.

2 Grundlagen

Wie in Kapitel 1 beschrieben, hat der stetige Zuwachs von Smart Phones **SP!**s [1] zum Umdenken bei der Planung und beim Entwickeln von Webapplikationen geführt. Zu Beginn jedes Projektes steht die Entscheidung an, welche Technologien und Tools zur Entwicklung verwendet werden sollen um die bestmöglichen Ergebnisse zu erhalten. Wenn die falschen Methoden gewählt werden, kann das zu gravierenden Fehlern in der Applikation führen, die sich erst mit Fortdauer der produktiven Verwendung ersichtlich machen. Die Frage ist, ob man sich für eine Anwendung die auf das Betriebssystem zugeschnitten ist oder doch für eine plattformübergreifende Webanwendung entscheidet. Beide Methoden haben Vorteile und Nachteile und werden im Zuge dieser Arbeit betrachtet. Den Kern der Arbeit aber stellten, die von Google entwickelten PWA [11] da.

Die PWAs sollen den Spagat zwischen diesen beiden Anwendungen schaffen. Eventuell könnte diese neue Form der Appentwicklung die traditionellen Technologien gar zur Gänze ablösen? Der Trend der letzten Jahren geht in Richtung der mobilen Nutzung und da ist das Smart Phone klar wie, in Abbildung 2.1 dargestellt, voran.

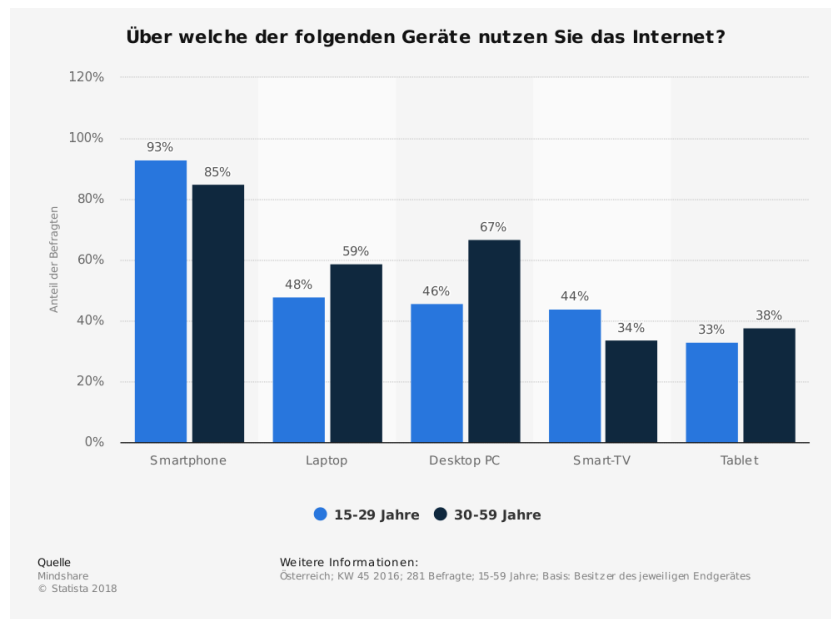


Abbildung 2.1: Smartphonennutzung [1]

2.1 Geschichte Softwareentwicklung

Im Laufe der Jahre wurden verschiedenste Softwares entwickelt, die mehr oder weniger nützlich für unseren Alltag waren. Der Begriff Software wurde 1958 vom US-amerikanischen Statistiker John W. Turkey eingeführt. Zu Beginn bildeten Software und Hardware eine Einheit. Erst nach der Entscheidung durch die US Regierung, dass IBM die Hardware und die Software separat verrechnen sollte, wurden sie getrennt.

Die Software bildet das Gehirn eines Computers. Nach der Entscheidung der US-Regierung entstanden erstmals rein softwareorientierte Unternehmen wie Microsoft oder SAP [12] [13].

2.2 Native Apps

Native Apps sind speziell für eine Plattform angepasste Anwendungen. Diese werden speziell für ein bestimmtes Betriebssystem konzipiert und haben in der Regel Zugriff auf alle Ressourcen eines Gerätes [14]. Hauptsächlich werden zur Programmierung für Mobile Geräte die Hochsprachen Java (Android) und Swift(IOS) verwendet. Native Apps können in App Stores heruntergeladen werden.

Die bekanntesten sind Apple Store und Google Play [15].

2.3 Web Applikationen

Im Gegensatz zu den Native Apps sind Web Applikationen (Web-App) speziell programmierte Webseiten [15]. Web-App funktionieren nach dem Server-Client Prinzip und werden vom Browser aufgerufen. In der Regel werden Web-Apps auf der Basis von JS, CSS und HTML5 entwickelt. Die Verarbeitung erfolgt auf dem Webserver oder auf der Cloud. Der größte Vorteil ist sicherlich der unkomplizierte Zugang im Gegensatz zu den Native App [16]. Durch die Einführung von Responsive Frameworks wie z.B.: Bootstrap, SemantikUI oder Foundation um nur die bekanntesten zu nennen, wurde die Webentwicklung vielseitiger in der Verwendung. Durch diese Technologien können viele Bildschirmgrößen mit wenig Aufwand abgedeckt werden [17].

2.4 Hybrid Applikationen

Hybrid Apps verbinden die Eigenschaften den in Kapitel 2.2 und 2.3 genannten Technologien. Zum einen verwenden sie die webbasierende Client-Server Technologie zum anderen kann man mit einer Hybrid App auf Gerätefunktionen wie Kamera und Kalender zugreifen [18].

2.5 Progressive Webapplikationen

Progressive Web Applikationen sind im Grunde eine Weiterentwicklung von einer Web-App. Diese Technologie der Webentwicklung wird durch die immer schneller wachsende Welt der Webanwendungen immer wichtiger. Dem User wird das Gefühl gegeben er arbeitet mit einer Native App. Das Herausragende dabei ist, im Gegensatz zu einer Hybrid App, dass jede bestehende Web-App in eine PWA umgebaut werden kann. Durch Hinzufügen einer Manifest Datei und eines Service Worker werden Features hinzugefügt, die es ermöglichen offline zu arbeiten oder das Icon der App auf den Desktop oder Home-Bildschirm zu speichern [11]. Google definiert die PWA wie folgt [19]:

- **Progressive** - funktioniert für alle User unabhängig vom Browser
- **Responsive** - passt sich jedem Gerät an
- **Verbindungsunabhängig** - funktioniert auch bei schlechtem oder gar keinem Internetzugang
- **App-like** - fühlt sich an wie eine Native App
- **Aktuell** - Durch die Wartung des Service Workers immer auf dem aktuellsten Stand
- **Sicher** - wird nur über HTTPS bereitgestellt
- **Erkennbar** - erkennbar dank das W3C Manifest durch Suchmaschinen
- **Wiedereinschaltbar** - wird durch die Funktion Push Notification erreicht
- **Installierbar** - Ermöglicht das Hinzufügen auf dem Startbildschirm

3 Features und Merkmale

In diesem Kapitel werden die Komponenten der Progressive Web Applikation (PWA) erklärt. Weiters werden durch die folgenden Tabellen die wichtigsten Punkte zwischen den verschiedenen Technologien gegenübergestellt.

3.1 Aufbau Progressive Web Apps (PWA)

Die PWAs sind keine neuen Technologien, vielmehr sind es verbesserte Strategien, Methoden und APIs wie in Abbildung 3.1 zu sehen. Sie erleichtern dem User die Benutzung und den Zugriff einer Web-App [20].



Abbildung 3.1: PWA Komponenten

3.2 Unterschiede PWA, Native Applikation und Web-Apps

In den folgenden Tabellen 3.1, 3.2 und 3.3 wird versucht die Features und Merkmale gegenüberzustellen um die Auswahl zu erleichtern. Die Unterschiede in den Punkten Veröffentlichung, Installation, Zugriff und Funktionen werden verglichen.

	PWA	Native	Web App
Veröffentlichung	Es werden verschiedene Entwicklerkonten benötigt Play Store und Apple Store	keine Entwicklerkonten benötigt	keine Entwicklerkonten benötigt
Installation	App muss aus einem der App-Stores downgeloadet werden	Wird mit einem Klick auf dem Startbildschirm hinzugefügt	keine Funktion
Updates	über App-Store	Serverseitig	Serverseitig

Tabelle 3.1: Veröffentlichung und Installation [5]

	PWA	Native	Web App
Offline-Zugriff	Verfügbar	Man muss die App einmal online nutzen, dann sollten die Inhalte im Cache offline verfügbar sein.	nicht möglich
Starten im Vollbildmodus	Verfügbar	Verfügbar	nicht möglich
Kundenbindung	sehr hoch, Kunden verbringen viel Zeit	App ist wie ein Tap, das macht es für den Kunden leichter zu wechseln	wie PWA

Tabelle 3.2: Zugriff [5]

	PWA	Native	Web App
Push-Nachrichten	Verfügbar	Verfügbar (nur für Android)	Verfügbar (mit zusätzliche Tools)
Geolocation	Verfügbar	Verfügbar	Verfügbar
Kamera-/Mikrofonzugriff	Verfügbar	Verfügbar	Verfügbar
Gerätevibration	Verfügbar	Verfügbar	nicht Verfügbar
Responsive	Verfügbar	Verfügbar	Verfügbar
Akkuladestatus	Verfügbar	Verfügbar	nicht Verfügbar
Zugriff auf Kontakte und Kalender	Verfügbar	nicht Verfügbar	nicht Verfügbar
Telefon: SMS oder Anrufe	Verfügbar	nicht Verfügbar	nicht Verfügbar

Tabelle 3.3: Funktionen [5]

Wie in den Tabellen ersichtlich bietet die PWA eine Reihe von Vorteilen, z.B. Push Notifications und Offline-Zugriff, die bei der Benutzung behilflich sein können. Aufgrund dessen bietet sie eine gute Alternative zu den Native App. Probleme machen die Betriebssysteme, da nicht alle Funktionen auf den verschiedenen Systemen zur Verfügung stehen [5]. In den nächsten Kapiteln werden die Methoden und APIs in der Theorie und im Kapitel 4 die praktische Anwendung an einer selbst erstellten App erklärt.

3.3 Web App Manifest

Das App Manifest ist eine JSON Datei die dem Browser verrät, wie sich die Web-App bei der Installation auf dem Startbildschirm verhält. Im Manifest werden der Name, der Kurzname, die Größe, das Aussehen der Icons und weitere Eigenschaften definiert. Dessen Zweck ist es der Anwendung auf dem Startbildschirm ihr Aussehen zu verleihen. Das App Manifest.json Datei wird in die gleiche Ebene wie die Index.html Datei in das Projekt eingepflegt und über den folgenden Link-Tag im Header implementiert:

```
1 <link rel="manifest" href="/<Dateiname>">
```

Listing 3.1: Manifest.json [6]

Bei Anwendungen mit mehreren HTML-Seiten muss der Link-Tag auf jeder Seite eingefügt werden. Im Listing 3.2 ist ein Auszug vom Aufbau dargestellt:

```
1  "name": "PWA Smart Home RMJ",
2  "short_name": "PWA_SHL_RMJ",
3  "start_url": "./",
4  "scope": ".",
5  "display": "standalone",
6  "background_color": "#003399",
7  "theme_color": "#3F51C5",
8  "icons": [
9    {
10     "src": "./static/img/light48.png",
11     "type": "image/png",
12     "sizes": "48x48"
13   }
14 ]
```

Listing 3.2: Manifest in das Projekt implementieren [6]

Im Grunde sind alle Key Value Paare selbst erklärend und auch auf <https://developers.google.com/web/fundamentals/web-app-manifest/> sehr gut beschrieben [6].

3.4 Add to Homescreen

Diese Funktion erleichtert es den Benutzern die App auf dem Desktop oder Startbildschirms zu installieren. Nach der Installation wird die PWA zum launcher hinzugefügt und wie alle anderen installierten Apps ausgeführt. Um den Banner auf dem mobilen Gerät anzuzeigen, müssen folgende Kriterien erfüllt werden [7].

- die App ist noch nicht installiert
- muss min 30 Sekunden lang mit der Domäne interagieren
- beinhaltet ein Web App Manifest mit folgenden Werten:
 - Kurzname oder Name
 - icons - muss ein 192px und ein 512px großes Icon enthalten
 - Startadresse
 - Anzeige muss eines der folgenden sein: fullscreen, standalone oder minimal-ui
- darf nur über HTTPS aufrufbar sein
- beinhaltet einen Service Worker mit einem Fetch-Event-Handler

Nachdem die oben genannten Bedingungen erfüllt wurden, wird ein Eventlistener gestartet siehe Listing 3.3

```
1 let deferredPrompt;  
2  
3 window.addEventListener('beforeinstallprompt', (e) => {  
4   // Prevent Chrome 67 and earlier from automatically showing the  
5     prompt  
6   e.preventDefault();  
7   // Stash the event so it can be triggered later.  
8   deferredPrompt = e;  
9 });
```

Listing 3.3: beforeinstallpromptEvent [7]

In der Abbildung 3.2 sieht man die Browserkompatibilität des Manifest Files zum Stand Juli 2018.

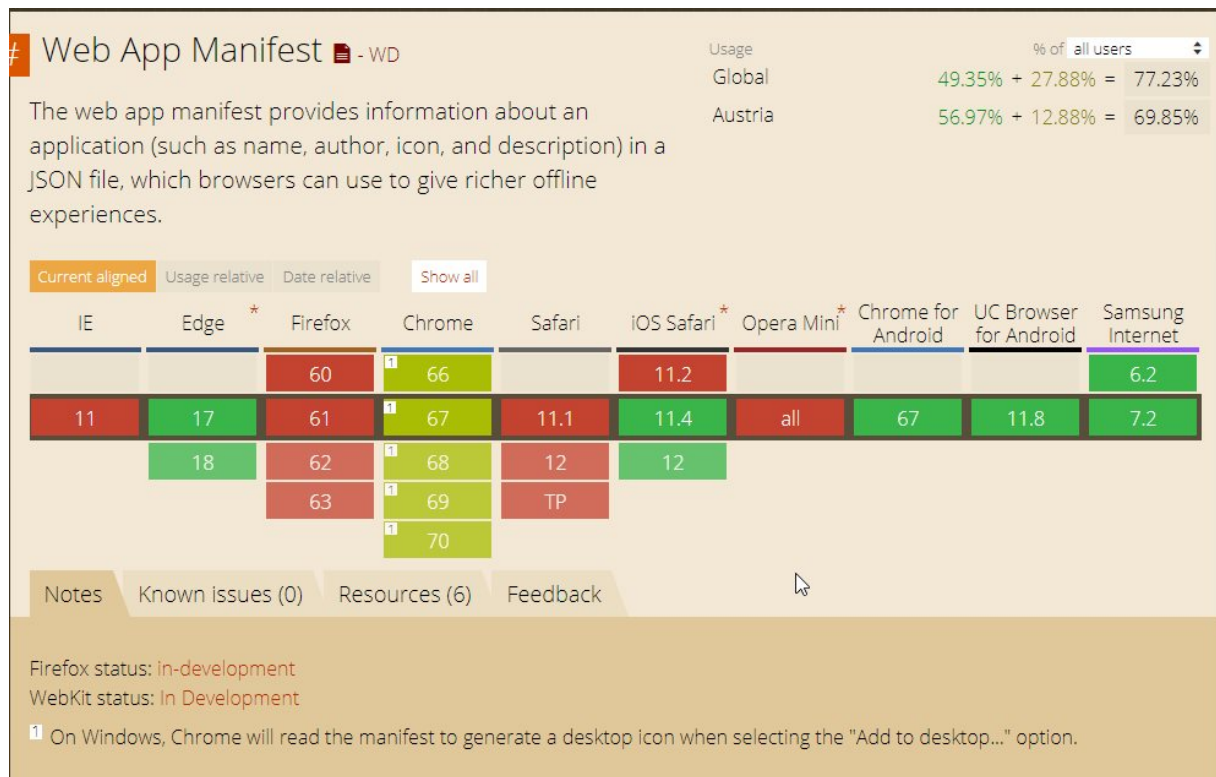


Abbildung 3.2: Kompatibilität Manifest.json [2]

3.5 Service Worker

Der Server Worker ist ein Script, das vom Browser im Hintergrund ausführt [4]. Mit Hilfe des Server Worker ist es möglich die Web-App offline zu betreiben, Push Notifikationen zu erhalten und gecachte Daten abzurufen. Server Worker verhalten sich wie Proxy-Server, welche in einer Zwischenschicht vom Browser und dem Netzwerk sitzen, wie in der Abbildung 3.3 zu sehen ist.

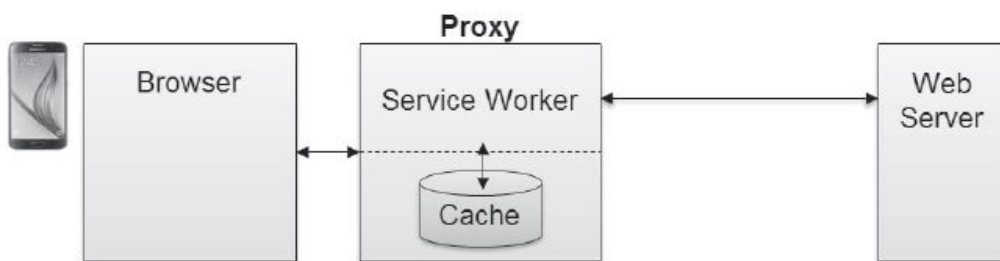


Abbildung 3.3: Service Worker als Proxy [3]

Ein Server Worker wird von einem Worker-Kontext ausgeführt, hat keinen DOM Zugriff und wird als Haupt-Java Script Thread verwendet [21] [22]. Der komplizierteste Teil des Server Worker ist sein Lebenszyklus. Die Aufgabe der Lebenszyklen sind wie folgt definiert:

- Offlinverwendung
- Störung eines anderen Service Workers verhindern
- Stellt sicher, dass nur ein Service Worker für eine Seite zuständig ist
- Stellt sicher dass nur eine Version der Webseite gleichzeitig ausgeführt wird

Der Lebenszyklus eines Server Workers ist von der Webseite getrennt. In der Installationsphase werden die benötigten statischen Dateien zwischengespeichert und erst nach diesem Vorgang ist der Server Worker installiert. Die Installation erfolgt über die JavaScript-Funktion wie in Listing 3.4:

```
1 navigator.serviceWorker.register
```

Listing 3.4: Service Worker Navigator [8]

Danach folgt die Aktivierungsphase. In dieser Phase werden alte Cache-Inhalte verwaltet und aktualisiert.

Um die neuen Seiten zu steuern muss der Server Worker erneut geladen werden. In der Abbildung 3.6 ist eine vereinfachte Erstinstallation zu sehen.

Um den Service Worker zu registrieren muss folgender JS-Code in das Projekt unter `/app/src/js/app.js` integriert werden.

```
1 if ('serviceWorker' in navigator) {  
2   console.log('Service Worker and Notification is supported')  
3   navigator.serviceWorker.register('/sw.js')  
4     .then(reg => {  
5       console.log('Service worker registered!', reg);  
6     })  
7     .catch(err => console.log(err));  
8 }  
9 }
```

Listing 3.5: Service Worker Register [4]

In Zeile eins wird im Listing 3.5 die Unterstützung durch den Browser geprüft, bevor in der dritten Zeile der Server Worker über die `navigator.serviceWorker.register(<ServiceWorker Name>')`-Funktion, wie in Abbildung 3.4 zu sehen ist, aufgerufen wird.

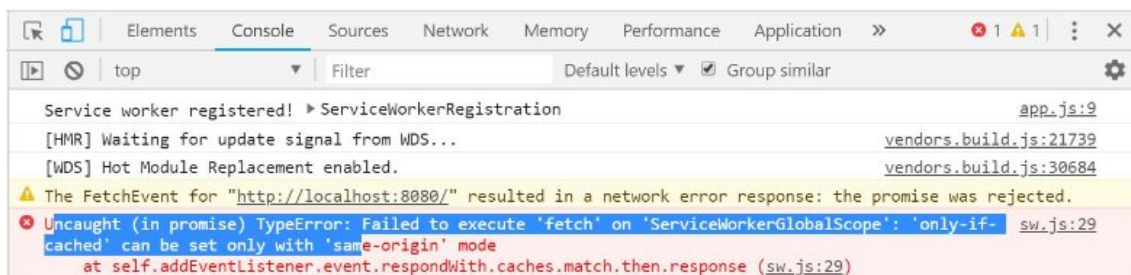


Abbildung 3.4: Registrierung Service Worker

In der Abbildung 3.5 ist zu sehen wie die Installation erst bei erneutem Laden der App startet. Dabei cached der Server Worker die zum Cache hinzugefügten Files, bevor die Installation fertig ist. Nach der Installation wird der Server Worker aktiviert und steht damit der Applikation zur Verfügung.

Wie in Abbildung 3.6 zu sehen ist kann der Server Worker nach der Übernahme der Steuerung zwei Zustände übernehmen, entweder dieser wird beendet oder er übernimmt die Verwaltung der Netzwerkanfragen und der Nachrichten [4]. Die Server Worker API stellt eine Cache-Schnittstelle zum Speichern von Daten auf dem Browser, im Browser-cache, zur Verfügung. Die API wurde ursprünglich für den Server Worker entwickelt,

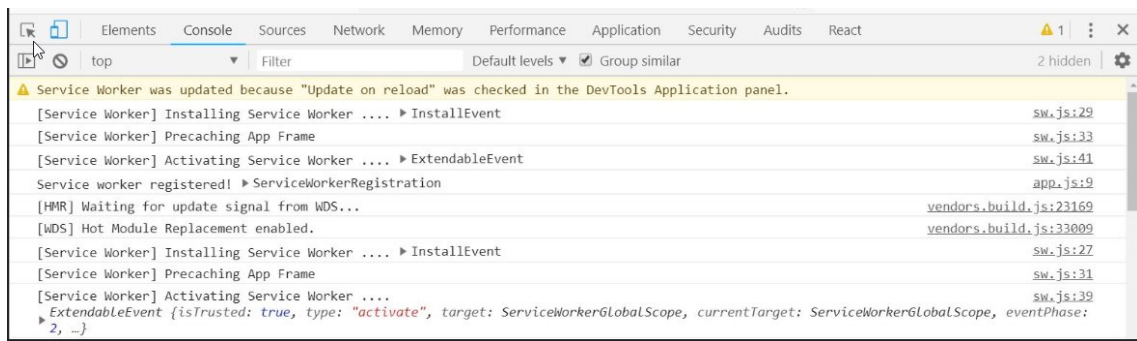


Abbildung 3.5: Registrierung Service Worker

diese kann aber von jedem anderen Script verwendet werden. Wie die API gestaltet wird, hängt von den Anforderungen der Applikation ab. Der Einstiegspunkt ist 'cache' wie man im folgenden Codebeispiel (Listing ??) sehr gut erkennen kann.

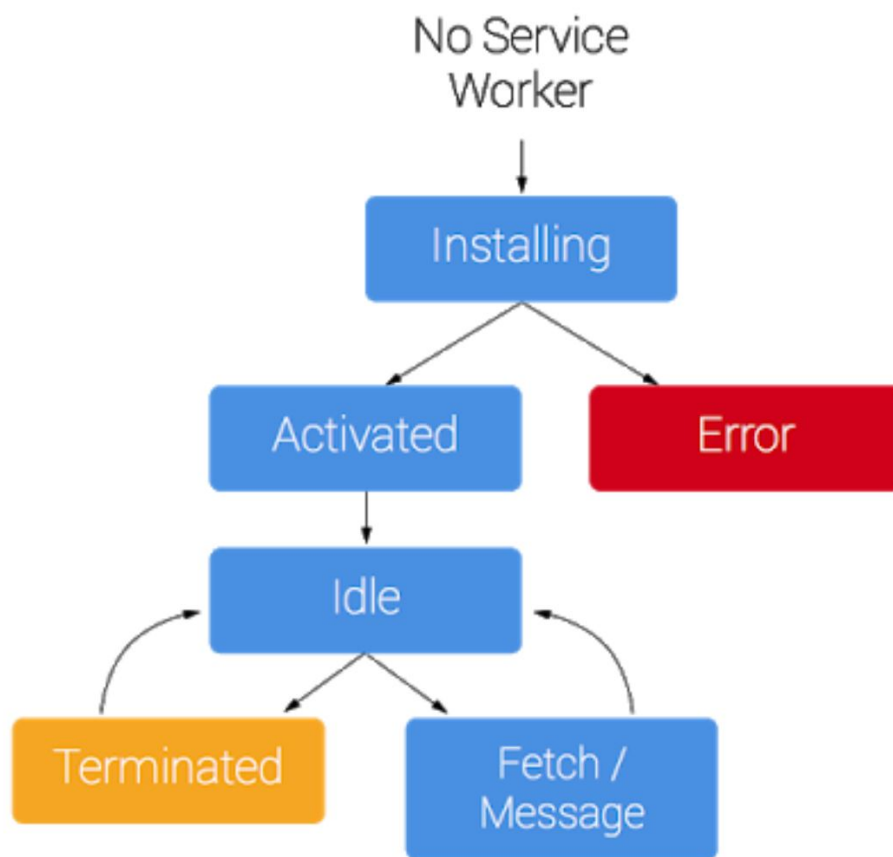


Abbildung 3.6: Erstinstallation Service Worker [4]

In der Abbildung 3.7 sieht man die Browserkompabilität des Service Workers zum Stand Juli 2018.

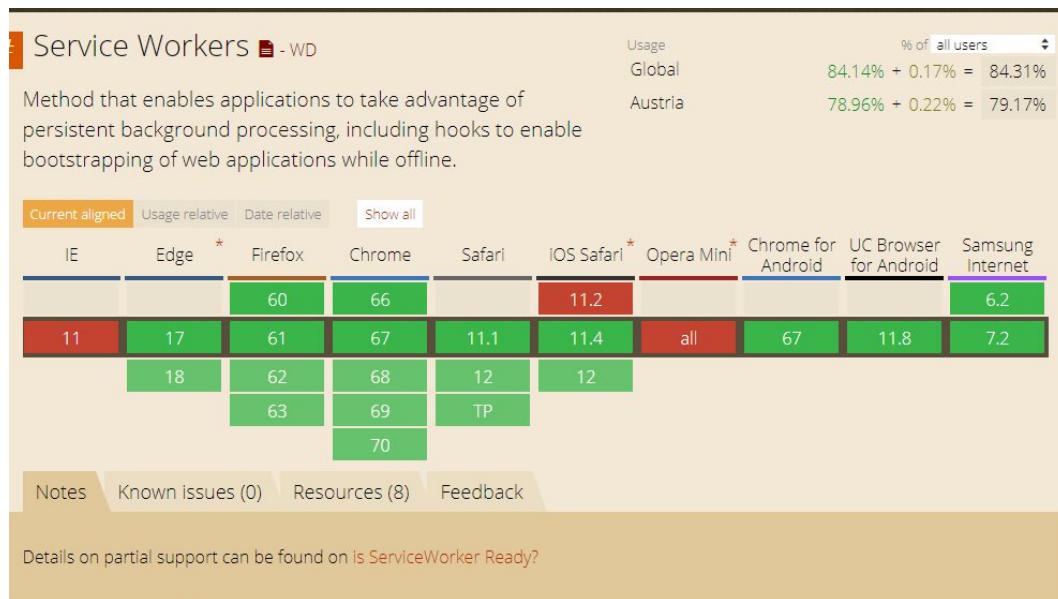


Abbildung 3.7: Kompabilität Service Worker [2]

3.6 Push Notifikation

Um dem User bei einer Server Worker das Gefühl von Nativen Applikationen aufkommen zu lassen ist die Push Funktion unablässig. Erst diese Funktion in Kombination mit dem Server Worker gibt den Web Applikationen ein individuelles Verhalten, dass bis jetzt den Native Apps vorbehalten war. Unternehmen bekommen, durch Push APIs und Notification APIs, neue Möglichkeiten mit dem Nutzer in Kontakt zu treten und ihn mit personalisierten, relevanten Inhalten zu versorgen. Zur Benachrichtigung oder für Push werden zwei Technologien eingesetzt. Mit Hilfe von Push werden vom Server Informationen an den Service Worker gesendet, um Informationen vom Service Worker zum Nutzer zu senden werden Benachrichtigungen verwendet. Die Technologien nutzen für diese Datenübertragung sich ergänzende APIs [23].

Im Kapitel 4 wird die praktische Verwendung der Benachrichtigungen genauer beschrieben.

Hier wird wie in Kapitel 3.5 die Browserunterstützung vom Service Worker und den Push Benachrichtigungen überprüft [23].

In der Abbildung 3.8 sieht man die Browserkompatibilität der Push API zum Stand Juli 2018.

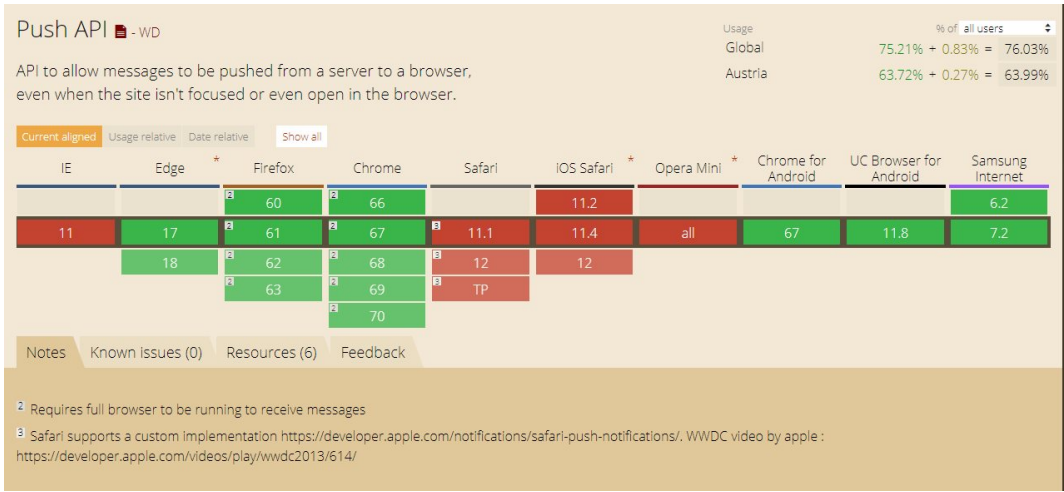


Abbildung 3.8: Kompatibilität Push Notifikation [2]

3.7 Geolocation API

Die Geolocation API kann nach Zustimmung des Benutzers den Standort bestimmen. Diese Funktion wird verwendet um den Benutzern zusätzlichen Nutzen zu bringen, wie z.B.: Optimierung von Benutzeranfragen, bestimmen des Standortes und Backendaufnahmen von Standortdaten für Datensammlung. Allerdings sind bei der Verwendung der Geolocation-API zwei wichtige Punkte zu beachten:

- sollte nur verwendet werden, wenn es für den Benutzer Vorteile bringt
- Fordern einer Erlaubnis durch den Benutzer

In der Abbildung 3.9 sieht man die Browserkompatibilität der Geolocation zum Stand Juli 2018.

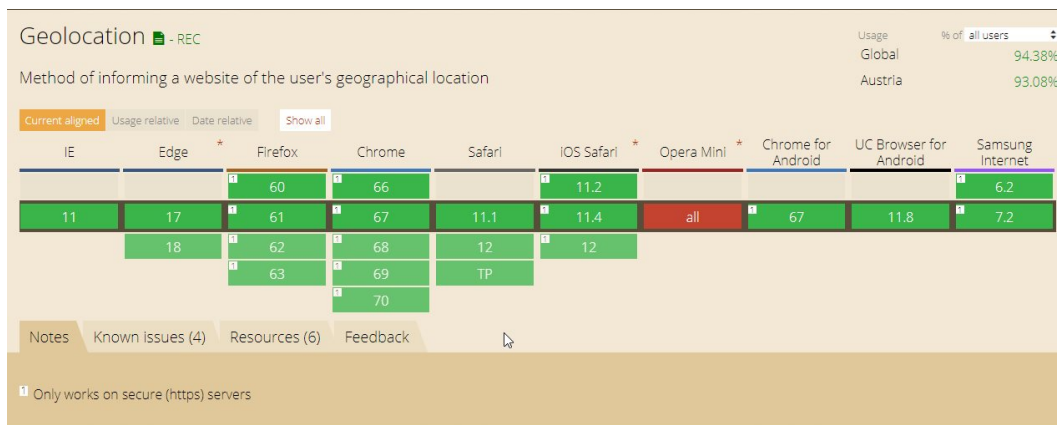


Abbildung 3.9: Kompabilität Geolocation [2]

4 Implementierung

In diesem Kapitel wird die Umsetzung der Applikation beschrieben.

4.1 Anforderungsanalyse

Die zu entwickelnde Smart Home Applikation muss das Verhalten einer PWA aufweisen. Das heißt es müssen die Attribute aus Kapitel 2.5 eingebaut werden und danach getestet werden siehe Kapitel 5. Außerdem sollen auch APIs entwickelt werden die es möglich machen mit Services von Drittanbietern die Temperatur zu regeln, mit Hilfe von Geolocation API den Standort zu ermitteln um das Garagentor über GPS automatisch öffnen zu können. Ebenso soll das Steuern der Beleuchtung möglich sein. Diese Applikation soll Offline so weit es geht verwendbar sein. Die Anwendung muss Plattformunabhängig und responsive sein.

4.2 Umsetzung der Anforderungen

Zur Umsetzung der Anforderungen aus Kapitel 4.1 wurden für User Interfac ReactJS¹ und als CSS Framework Semantic-UI² verwendet. Semantic-UI soll sicherstellen das die Applikation responsives Verhalten aufweist und für alle Bildschirmgrößen geeignet ist. Um die Daten zu versenden, aufzurufen und zu speichern wurde das JSON Key/Value Format, die Fetch API und der Browser Cache verwendet. Als Browser diente der Google Chrome Version 67. Die nicht fertigen Funktionen wurden mit Mockups dargestellt, um einen Eindruck zu vermitteln wie die App in Zukunft aussehen soll.

4.3 Ausgewählte Programmiersprache und IDE

Als Programmiersprache wurde JavaScript (JS) ausgewählt. Als Entwicklungsumgebung wurde Webstorm (Version 2018.2) von JetBrains verwendet. Weitere verwendete Tools und Frameworks wurden in Kapitel 4.2 beschrieben.

¹<https://reactjs.org/docs/getting-started.html>

²<https://react.semantic-ui.com/introduction>

4.4 Ordnerstruktur

Die zwei wichtigsten Dateien befinden sich wie in Abbildung 4.1 zu sehen ist, im app Verzeichnis. Ebenfalls wichtig ist das app.js File das unter `/app/src/js/app.js` zu finden ist.

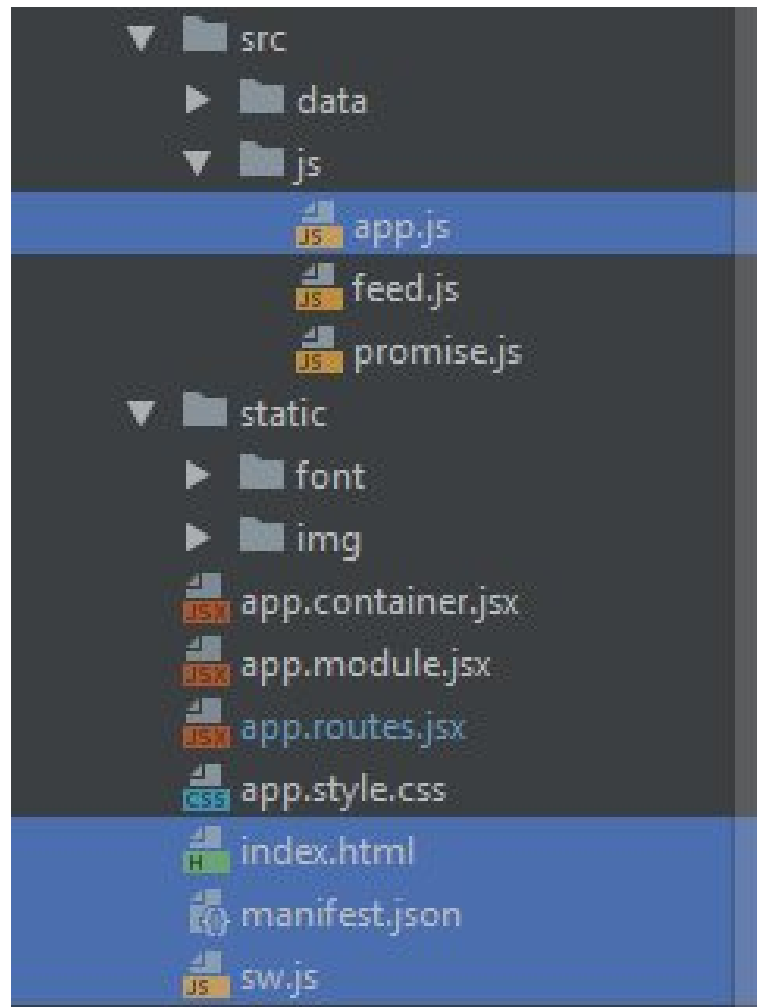


Abbildung 4.1: Ordner Struktur

4.5 Manifest

Das Manifest wird im Root Folder eingefügt und ist mit der Endung JSON deklariert. Der genau Pfad ist `/app/manifest.json`. Wie in Kapitel 3.3 schon beschrieben wurde, definiert das Manifest File das Aussehen des Icons am Startbildschirm, z.B. den Einstiegspunkt der App und den Namen. Die Listing 4.1 zeigt weitere wichtige Eigenschaften:

```
1 {
2   {
3     "name": "PWA Smart Home",
4     "short_name": "PWA_SHL_RMJ",
5     "start_url": "./",
6     "scope": ".",
7     "display": "standalone",
8     "background_color": "#003399",
9     "theme_color": "#3F51C5",
10    "description": "Keep running with PWA",
11    "dir": "ltr",
12    "lang": "de-DE",
13    "orientation": "portrait-primary",
14    "icons": [
15      {
16        "src": "./static/img/light48.png",
17        "type": "image/png",
18        "sizes": "48x48"
19      },
20      {
21        "src": "./static/img/light512.png",
22        "type": "image/png",
23        "sizes": "512x512"
24      }
25    ]
26  }
27 }
```

Listing 4.1: Manifest in das Projekt implementieren [6]

4.6 Add to Homescreen

Damit der Add to Homescreen Banner erscheint müssen die Bedingungen aus dem Kapitel 3.4 erfüllt werden. Nach Erfüllung dieser Forderungen wird der `beforeinstallprompt` Event wie in Listing 4.2 aufgerufen und in die `deferredPrompt` variable gespeichert.

Verzeichnis: `/app/src/js/app.js`

```
1 var deferredPrompt;
2
3 window.addEventListener('beforeinstallprompt', event => {
4     event.preventDefault();
5     deferredPrompt = event;
6     return false;
7 });
8 %
```

Listing 4.2: Add to Homescreen Funktion

Diese Callbackfunktion wird in die `app.js` Datei implementiert.

4.7 Service Worker und Cache API

Das Herzstück der Applikation ist der im Kapitel ?? beschriebene Service Worker. Als erstes muss validiert werden ob der Service Worker vom Browser unterstützt wird und danach kann dieser registriert, installiert und aktiviert werden. In den folgenden Listings 4.3, 4.4 und 4.5 sind die Funktionen, die für den **SW!** von Bedeutung sind aufgeführt und die wichtigsten Teile beschrieben.

Verzeichnis: `/app/src/js/app.js`

```
1 //Registrierung und Validierung vom Service Worker
2 if ('serviceWorker' in navigator && 'Notification' in window) {
3     console.log('Service Worker and Notification is supported')
4     navigator.serviceWorker.register('/sw.js')
5         .then(reg => {
6             console.log('Service worker registered!', reg);
7
8         })
9     .catch(err => console.log(err));
10 }
```

Listing 4.3: Registrierung

Die Registriermethode *register()* bekommt als Parameter die Service Worker Datei mitgegeben.

Verzeichnis: */app/sw.js*

```
1 self.addEventListener('install', event => {
2     console.log('[Service Worker] Installing Service Worker ....'
3         , event);
4     event.waitUntil(
5         caches.open(cacheName)
6             .then(cache => {
7                 console.log('[Service Worker] Precaching App
8                     Frame');
9                 cache.addAll(filesToCache);
10             })
11     );
12 });
```

Listing 4.4: Installation

Nach dem erneuten Laden der Anwendung wird im Listing 4.4 der Installations Eventlistener aufgerufen. Dieser installiert den Service Worker und cached die angegebenen Dateien um die App offline verwenden zu können.

Durch die Funktion *waitUntil()* wird mit der Installation gewartet, bis die Dateien die dieser Funktion als Parameter mitgegeben wurden, gecacht wurden. Durch die Cachemethode *cache.All()* werden alle angegebenen Dateien aufgerufen und es müssen nicht alle Dateien einzeln eingeben werden. Nach der Installation wird der Service Worker aktiviert und kann dann vom Browser verwendet werden.

Verzeichnis: */app/sw.js*

```
1 self.addEventListener('activate', event => {
2     console.log('[Service Worker] Activating Service Worker ....'
3         , event);
4     return self.clients.claim();
5 });
```

Listing 4.5: Aktivierung

Durch die *self.clients.claim()* Methode in Zeile 3 wird sichergestellt, dass der Service Worker nur installiert wird wenn alle Bedingungen erfüllt wurden.

In der Callback-Funktion *respondWith()* werden die Daten aufgerufen die *match*-Methode überprüft ob die Daten sich im Cache befinden. Um den Event aufzurufen werden Promises für asynchrone Aufrufe verwendet . Um Daten aus dem Netzwerk aufzurufen die nicht im Cachespeicher vorhanden sind wird über den *fetch* Event aufgerufen und überprüft wie in Abbildung 4.6 zu sehen.

Verzeichnis: */app/sw.js*

```
1 self.addEventListener('fetch', event => {
2     event.respondWith(
3         caches.match(event.request)
4         .then(response => {
5             if (response) {
6                 return response;
7             } else {
8                 return fetch(event.request);
9             }
10        })
11    );
12 });
```

Listing 4.6: fetch Callbackfunktion

4.8 Offline Modus

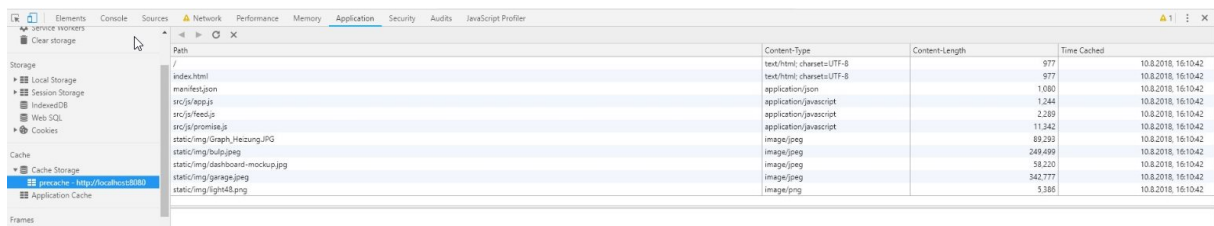
Eine der wichtigsten Aufgaben des Caches vom Service Worker ist der Offlinemodus oder das Arbeiten bei schlechter Internetverbindung. Der Cache beinhaltet im Grunde das Index.html File CSS und Bilder oder Icons. Bei der Entwicklung der Anwendung wurden statische Dateien wie im Listing 4.7 verwendet. Die benötigten Dateien wurden in die *let fileToCache* Variable gespeichert und im Listing 4.4 und der *caches.addAll()*-Methode als Parameter mitgegeben.

```

1 let cacheName = 'precache';
2 let filesToCache = [
3     '/',
4     '/index.html',
5     '/src/js/app.js',
6     '/static/img/light48.png',
7     '/static/img/dashboard-mockup.jpg',
8     '/static/img/bulb.jpeg',
9     '/static/img/garage.jpeg',
10    '/static/img/Graph_Heizung.JPG',
11    '/manifest.json'
12 ];

```

Listing 4.7: Cache



Path	Content-Type	Content-Length	Time Cached
/	text/html; charset=UTF-8	977	10.8.2018, 16:10:42
index.html	text/html; charset=UTF-8	977	10.8.2018, 16:10:42
manifest.json	application/json	1,080	10.8.2018, 16:10:42
src/js/app.js	application/javascript	1,244	10.8.2018, 16:10:42
src/js/feed.js	application/javascript	2,289	10.8.2018, 16:10:42
src/js/promise.js	application/javascript	11,342	10.8.2018, 16:10:42
static/img/Graph_Heizung.JPG	image/jpeg	69,283	10.8.2018, 16:10:42
static/img/bulb.jpeg	image/jpeg	269,499	10.8.2018, 16:10:42
static/img/dashboard-mockup.jpg	image/jpeg	58,220	10.8.2018, 16:10:42
static/img/garage.jpeg	image/jpeg	342,777	10.8.2018, 16:10:42
static/img/light48.png	image/png	5,386	10.8.2018, 16:10:42

Abbildung 4.2: Cache

Name	Method	Status	Type	L...	Size	▲ T...	Waterfall
semantic.min.css	GET	200	stylesheet	li...	(from ServiceWorker)	3...	
icon?family=Material+icons	GET	200	stylesheet	li...	(from ServiceWorker)	2...	
jquery-1.10.1.min.js	GET	200	script	li...	(from ServiceWorker)	0...	
app.js	GET	200	script	li...	(from ServiceWorker)	2...	
promise.js	GET	200	script	li...	(from ServiceWorker)	2...	
feed.js	GET	200	script	li...	(from ServiceWorker)	2...	
semantic.min.css	GET	200	text/css	\$...	(from disk cache)	1...	
icon?family=Material+icons	GET	200	fetch	\$...	(from disk cache)	6...	
css?family=Lato:400,700,400itali...	GET	200	stylesheet	li...	(from ServiceWorker)	0...	
S6uyw4BMUTPHjx4wXWtPCc.w...	GET	200	font	li...	(from ServiceWorker)	0...	
S6u9w4BMUTPHh6UVSwPGQ3...	GET	200	font	li...	(from ServiceWorker)	0...	
icons.woff2	GET	200	font	li...	(from ServiceWorker)	0...	
data:application/x-...	GET	200	font	li...	(from memory cache)	0...	
bulp.jpeg	GET	200	jpeg	Y...	(from ServiceWorker)	1...	
manifest.build.js	GET	200	script	li...	(from ServiceWorker)	3...	
vendors.build.js	GET	200	script	li...	(from ServiceWorker)	4...	
app.build.js	GET	200	script	li...	(from ServiceWorker)	3...	
semantic.min.css	GET	307	\$...	\$...	0 B	1...	
info?t=1533212962212	GET	200	xhr	Y...	(from ServiceWorker)	6...	

Abbildung 4.3: Datenaufwurf Service Worker

In der Abbildung 4.3 und 4.2 sind die gecachten Files vom Service Worker im Netzwerk und im Cache zu erkennen.

4.9 Push Notifications

Die Push Benachrichtigungen dienen dem Nutzer dazu, bei Änderungen im Haushalt benachrichtigt zu werden, z.B.: wenn das Licht defekt ist oder die Verbindung zur Heizung ausfällt.

Die Benachrichtigungen wurden in diesem Projekt nicht vom Server aus aufgerufen sondern sind über folgendes Listing im Pfad: `/app/src/app.js` eingetragen. Nach der Überprüfung im Listing 4.3 ob die Pushfunktion im Browserscope vorhanden ist, wird in Listing 4.8 der Eventlistener über eine Callbackfunktion aufgerufen. Bei diesem Prototypen sind die Benachrichtigungen im JSON-File (Verzeichnis: `/app/src/data/notification.json`) und werden über eine fetch-API aufgerufen. Durch die `openWindow()`-Funktion in Zeile 8 wird die Seite mit dem Benachrichtigungstext geöffnet.

```
1 self.addEventListener('notificationclick', e => {
2     let notification = e.notification;
3     let action = e.action;
4
5     if (action === 'close') {
6         notification.close();
7     } else {
8         clients.openWindow('http://localhost:8080/
          benachrichtigungen');
9         e.notification.close()
10
11     }
12 });
```

Listing 4.8: Geolocation Support [9]

4.10 Geolocation API

Um zu zeigen, dass der Zugriff auf die Geräte-APIs möglich ist wurde in der Applikation eine Funktion hinzugefügt, die den genauen Standpunkt über die Geolocation-API ermittelt. Diese Funktion könnte als Garagenöffner oder zum Einschalten der Heizung nützlich sein. Wenn sich der Nutzer dem Haus nähert könnte über die GPS Daten das Tor geöffnet werden ohne, dass der Benutzer über eine HCI eingreifen muss. Als erstes wird im Listing 4.9 der Support des Browsers überprüft.

Verzeichnis: */app/sw.js*

```
1  if (navigator.geolocation) {  
2    console.log('Geolocation is supported!');  
3  }  
4  else {  
5    console.log('Geolocation is not supported for this Browser/OS.'  
6              );  
6  }
```

Listing 4.9: Geolocation Support [10]

In der Abbildung 4.4 sieht man, dass der Chrome Browser diese Geolocation Funktion unterstützt und man sieht auch die genauen Koordinaten.

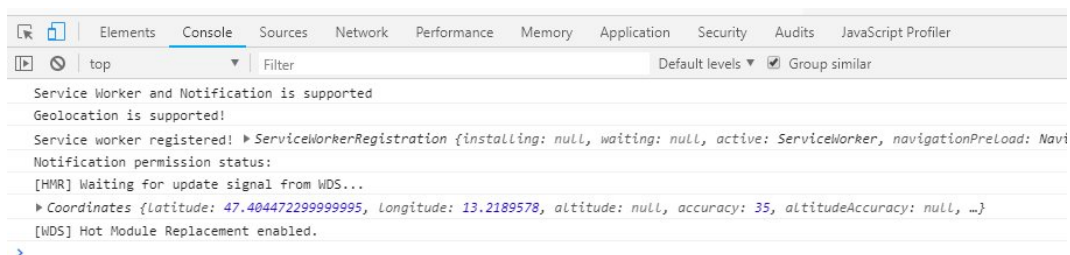


Abbildung 4.4: Konsolenmeldung Geolocation

5 Funktionstest/Validierung

5.1 Ausgangsbedingung und Ausgrenzung

Getestet wurden die in Kapitel 3 und 4 beschriebenen PWA-Features. Dies wurde zum einen über die DevTools vom Chrome Browser sowie über das Chrome PlugIN Lighthouse getestet. Als mobiles Testgerät wurde das Nexus X5 mit der Android 8.1.0 Software verwendet. Weiters kann der Emulator von Android Studio verwendet werden um den Test ohne Androidgerät darzustellen. Die Applikation selbst wurde hier nicht behandelt.

5.2 Testen auf Mobilen Geräten und Android Studio Emulator

Um auf dem Mobilen Smartphone testen zu können muss der Developer Modus auf dem Gerät eingeschaltet werden. Dies wird durch das Aktivieren der Entwicklertools und das Freischalten der USB-Debugging Funktion wie in Abbildung 5.1 und 5.2 zu sehen ist erreicht.

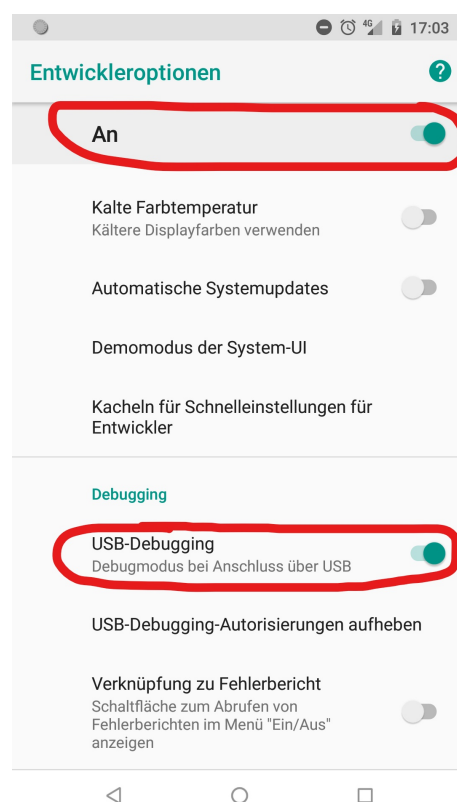


Abbildung 5.1: Aktivieren der Entwicklertools auf Android 8.1.0

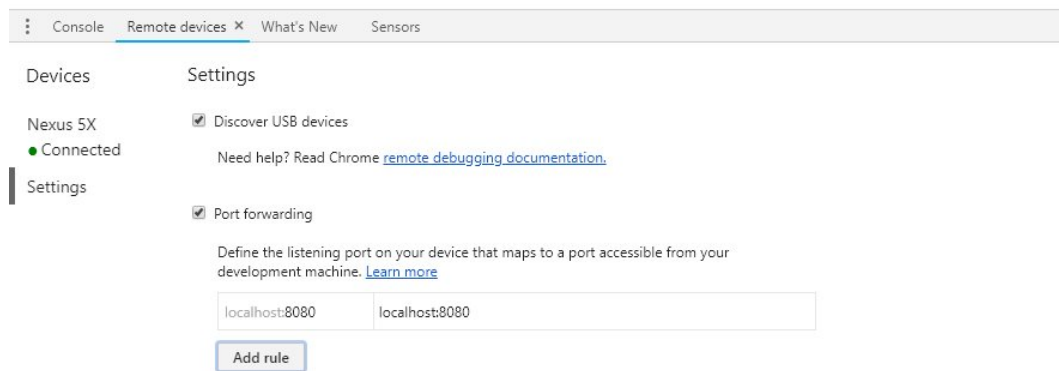


Abbildung 5.2: Anzeige der Verbindung auf Google Chrome 67

Falls kein Android Gerät zur Verfügung steht ist der von Android Studio¹ angebotene Emulator eine große Hilfe. Durch den integrierten Emulator lassen sich verschiedene Softwareversionen von Android darstellen. Sie helfen bei der Entwicklung und beim Testen der PWA.

5.3 Lighthouse

Lighthouse ist ein open-source Tool von Google und unterstützt den Entwickler bei der Verbesserung und Transformation der Applikation zu einer vollwertigen PWA. Man kann Lighthouse über 3 Wege verwenden:

- in Chrome DevTools
- über die Kommandozeile
- oder im Continuous Integration Prozess als Node Module

Jeder dieser Workflows benötigt den Google Chrome Browser [24]. Der Einsatz von Lighthouse über den Browser ist einfach. Nach eingabe der URL kann das Tool über das Chrome PlugIn, wie in Abbildung 5.3 zu sehen ist, gestartet werden. Lighthouse führt einen Debuggingtest (Abbildung 5.4) aus und erstellt einen Bericht. In Abbildung 5.5 ist der Überblick der Applikation zu sehen und in Abbildung 5.6 werden die fehlenden PWA-Features angezeigt.

¹<https://developer.android.com/studio/>

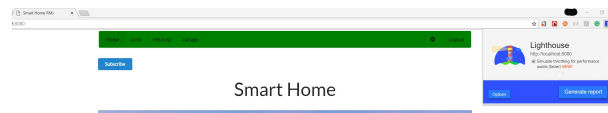


Abbildung 5.3: Lighthouse Plugin Chrome 67

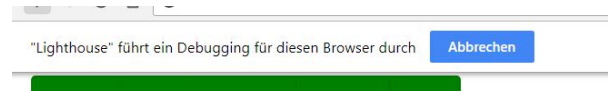


Abbildung 5.4: Debugging

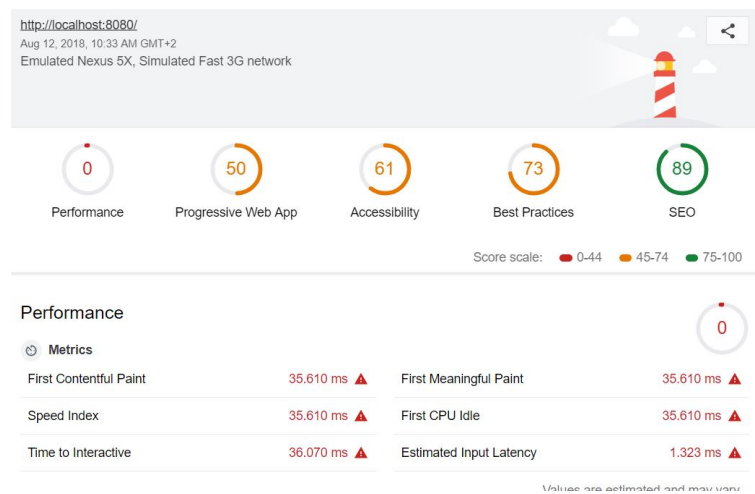


Abbildung 5.5: Lighthouse Überblick

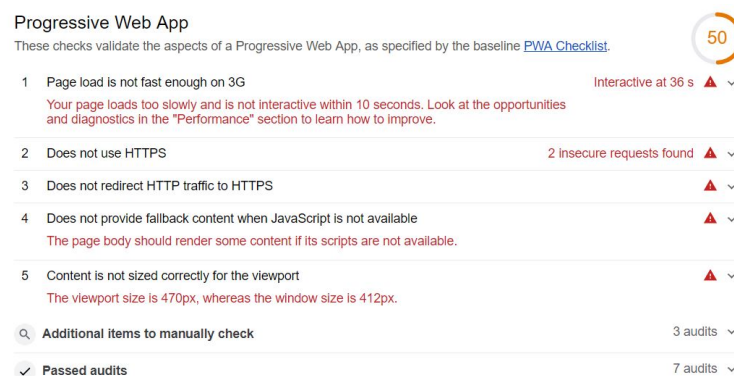


Abbildung 5.6: Lighthouse fehlende PWA-Features

In beiden Abbildungen 5.5 und 5.6 erkennt man, dass die Applikation nur zur Hälfte die PWA Features enthält. Dies kommt daher, weil die PWA bei diesem Test auf dem Localhost läuft und nicht wie von Google gefordert über das HTTPS-Protokoll. Das Projekt wurde für diese Arbeit nicht Live gestellt.

5.4 Add to Homescreen

Durch dieses Feature sollte die PWA dem Benutzer das Gefühl einer Native App geben. Doch leider funktioniert diese Funktion nicht immer wie sie sollte. Das Feature ist zurzeit nur auf Android und Chrome Browsern verfügbar und ist damit weit entfernt von den Native Apps. Startet oft nicht automatisch, ist aber durch das Menu aufrufbar wie in Abbildung 5.7 zu sehen ist.

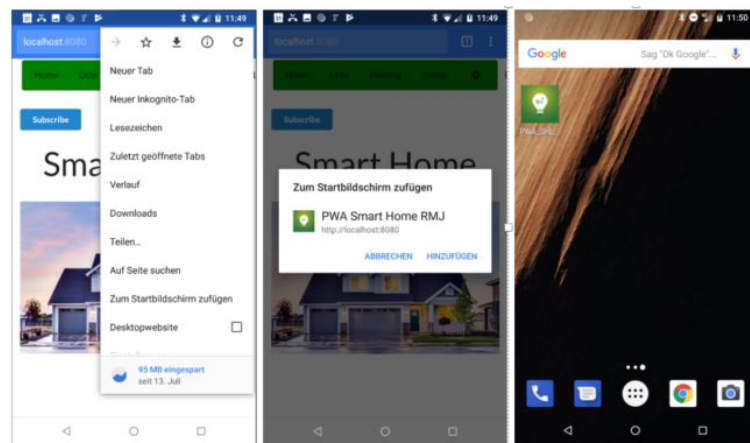


Abbildung 5.7: Add to Homescreen

5.5 Service Worker

Die Status des Service Workers wird durch die DevTools geprüft. Wie in Abbildung 5.8 gezeigt ist pro PWA nur ein Service Worker aktiv bei drei offenen Tabs.

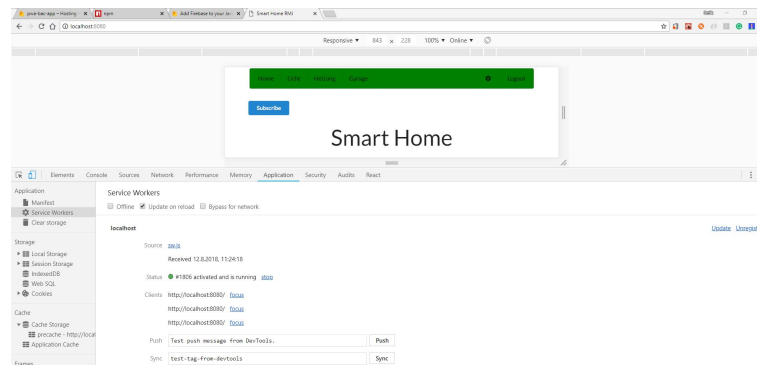


Abbildung 5.8: Service Worker Status

Weiters wurde die Offline Funktion der App getestet. Abbildung 5.9 zeigt, dass die Applikation offline funktioniert.

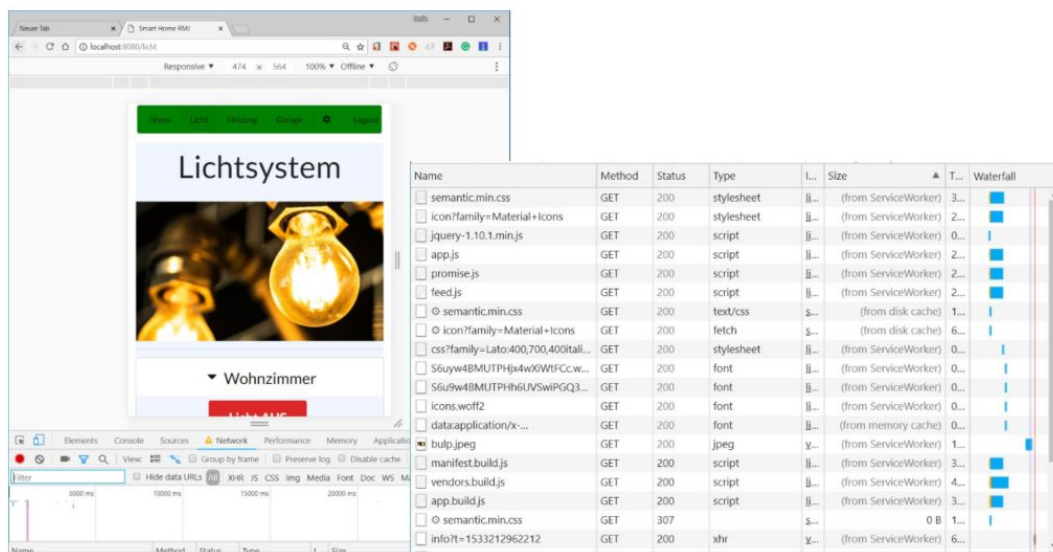


Abbildung 5.9: Offline

5.6 Push Notifikation

Um die Nachrichten erhalten zu können wird die Berechtigung (Abbildung 5.12) als Erstes abgefragt und dann werden die Benachrichtigungen an den User erst verschickt siehe Abbildung 5.11.

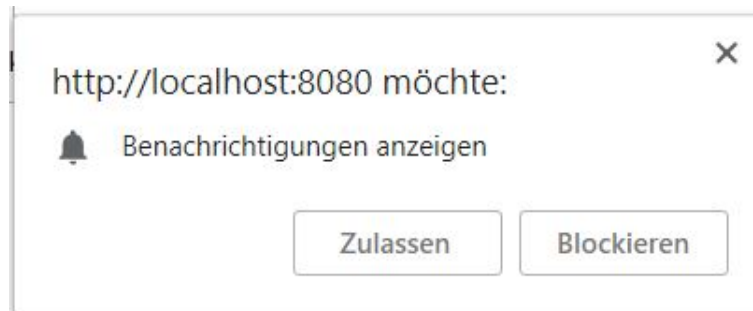


Abbildung 5.10: Registrierung Push Notification

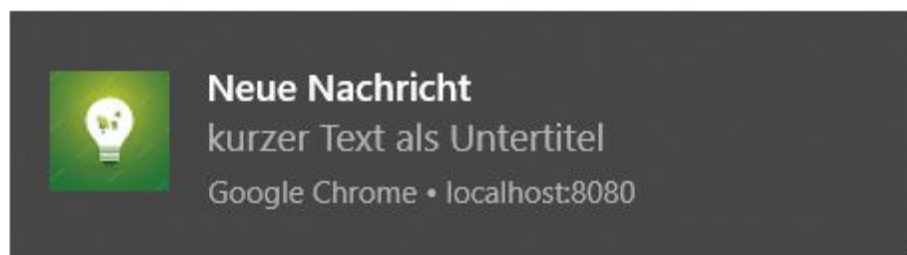


Abbildung 5.11: Push Notification

5.7 Geolocation

Die Geolocation Funktion muss den Nutzer immer fragen ob dieser seinen Standort bestimmt haben will. Wie in Abbildung 5.12 zu sehen ist, wird das in dem Prototypen umgesetzt. In der Abbildung 5.13 sieht man wie der Standort über die Geolocation-API ermittelt worden ist.

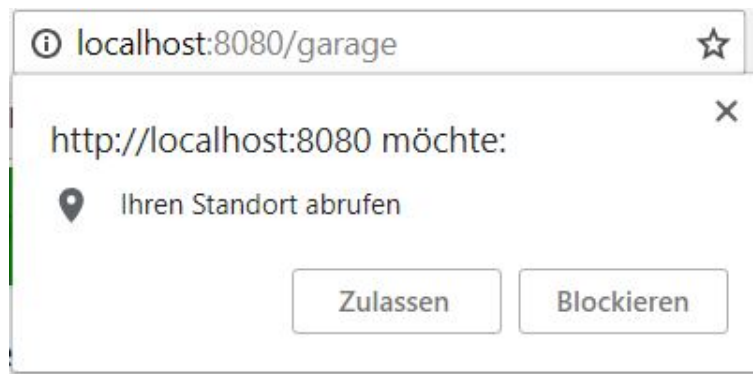


Abbildung 5.12: Registrierung Geolocation

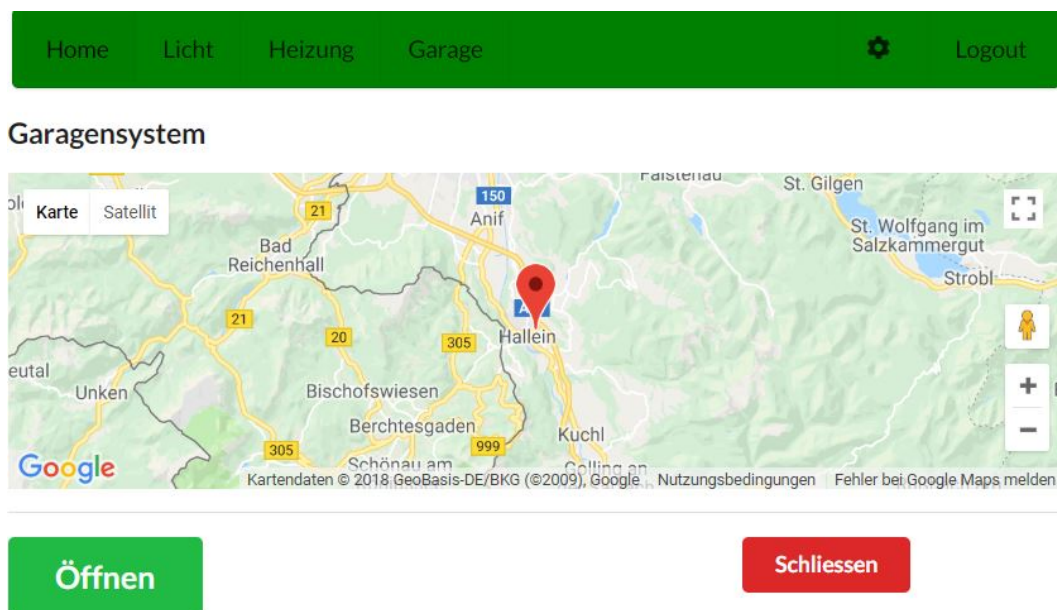


Abbildung 5.13: Standortermittlung

5.8 Vergleich mit Native App

Im Vergleich zur Native App bietet die Progressive Web Applikation einige Vorteile, aber auch Nachteile. Die PWAs sind responsive, müssen nicht heruntergeladen oder installiert werden und können zudem auf Geräte-APIs zugreifen. Durch die Möglichkeit benachrichtigt zu werden, wenn sich was ändert ist sie auch eine gute Alternative zu den Native Apps. Dank des Service Workers findet sich auch die Möglichkeit Applikationen offline oder bei niedriger Datenrate zu betreiben. Der Home Screen Banner verkürzt den Zugriff auf die App wesentlich, wodurch eine höhere Kundenbindung erreicht wird. Durch die Entwicklung der PWA pushed Google das verschlüsselte Versenden der Nachrichten über das HTTPS-Protokoll. Es sind aber auch noch einige Nachteile vorhanden, z.B.: funktionieren die oben genannten Features derzeit mehrheitlich nur auf Android-Geräten und dadurch ist diese Technologie nicht komplett Betriebssystemunabhängig. Die Browser unterstützen noch nicht alle Features, insbesondere hinkt der Internet Explorer von Microsoft hinterher, das ist deshalb ein Problem da Unternehmen noch auf diesen Browser setzen. Durch diese Tatsachen ist die Native App für eine angepasste Anwendung die bessere Wahl. Vorstellbar ist, dass die Technologie sich bei temporär benötigten Webseiten oder Nachrichtendienstseiten durchsetzen kann. Die PWA wird die Native Apps wahrscheinlich nie ersetzen, aber sie haben sicher das Potenzial eine feste Strategie bei der Entwicklung von Web-Apps zu werden.

6 Zusammenfassung und Ausblick

Literaturverzeichnis

- [1] Mindshare, *Über welche der folgenden Geräte nutzen Sie das Internet?*,
Gerätenutzung(2018).
 - [2] Fyrd, Lensco, *Can I Use*, <https://caniuse.com> (25.06.2018).
 - [3] A. Luntovskyy, „Advanced software-technological approaches for mobile apps development,” in *2018 14th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)*, Feb 2018, S. 113–118.
 - [4] Google Developers, *Your First Progressive Web App*,
<https://developers.google.com/web/fundamentals/primers/service-workers>
(2018).
 - [5] Robert, *Progressive Web Apps (PWA)*
Was ist das überhaupt und wie nutzt man sie?,
<https://apptooltester.com/de/progressive-web-apps> (12.03.2018).
 - [6] Matt Gaunt, Paul Kinlan, *The Web App Manifest*,
<https://developers.google.com/web/fundamentals/web-app-manifest>
(02.07.2018).
 - [7] Pete LePage, *Add to Home Screen*,
<https://developers.google.com/web/fundamentals/app-install-banners>
(17.07.2018).
 - [8] Jeff Posnick, *Service Worker Registration*,
<https://developers.google.com/web/fundamentals/primers/service-workers/registration> (02.07.2018).
 - [9] Google Developers, *Introduction to Push Notifications*,
<https://developers.google.com/web/ilt/pwa/introduction-to-push-notifications>
(2018).
 - [10] Paul Kinlan, *User Location*,
<https://developers.google.com/web/fundamentals/native-hardware/user-location>
(02.07.2018).
 - [11] Google Developers, *Progressive Web Apps*,
<https://developers.google.com/web/progressive-web-apps> (28.06.2018).
 - [12] Microsoft Corporation, *Microsoft Fast Facts*,
<https://news.microsoft.com/de-de/fast-facts> (2018).
 - [13] SAP SE, *SAP: 46 Jahre Innovation*,
<https://www.sap.com/corporate/de/company/history.html> (2018).
-

- [14] App Entwickler Verzeichnis, *Native Apps vs. Web Apps - Unterschiede und Vorteile*, <https://app-entwickler-verzeichnis.de/faq-app-entwicklung/11-definitionen/586-unterschiede-und-vergleich-native-apps-vs-web-apps-2> (2018).
- [15] Margaret Rouse, Alexander Gillis, *DEFINITION native App*, <https://searchsoftwarequality.techtarget.com/definition/native-application-native-app> (2013).
- [16] Stephan Augsten, *Defintion „Webanwendung“ Was ist eine Web App?*, <https://www.dev-insider.de/was-ist-eine-web-app-a-596814> (20.04.2017).
- [17] Anton Shaleynikov, *Top 5 Most Popular CSS Frameworks that You Should Pay Attention to in 2017*, <https://hackernoon.com/top-5-most-popular-css-frameworks-that-you-should-pay-attention-to-in-2017-344a8b67fba1> (2018).
- [18] Beratung FLYACTS, *Hybrid-Apps – Definition, Eigenschaften, Einsatzorte, Vorteile und Beispiele*, <https://www.flyacts.com/hybrid-apps-definition-eigenschaften-einsatzorte-vorteile-und-beispiele> (03.12.2013).
- [19] Google Developers, *Your First Progressive Web App*, <https://codelabs.developers.google.com/codelabs/your-first-pwapp/index.html> (2018).
- [20] D. Fortunato und J. Bernardino, „Progressive web apps: An alternative to the native mobile Apps,” in *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*, June 2018, S. 1–6.
- [21] Dennis Sterzenbach, *Worker*, <https://developer.mozilla.org/de/docs/Web/API/Worker> (21.12.2017).
- [22] Bitbruder, TobiDo, Heniz, *Service Worker API*, https://developer.mozilla.org/de/docs/Web/API/Service_Worker_API (30.01.2018).
- [23] Google Developers, *4.3.1 Einführung in Push-Benachrichtigungen im Web und Benachrichtigungen*, <https://support.google.com/partners/answer/7336533> (2018).
- [24] Google Developers, *Lighthouse*, <https://developers.google.com/web/tools/lighthouse> (09.04.2018).