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

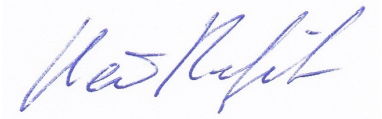
Studiengangsleiter:	FH-Prof. DI Dr. Gerhard Jöchtl
Betreuer:	DI Norbert Egger BSc

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:	PWA, Manifest, Service Workers, Push Notification, Cach API
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

PWA	Progressive Web Applikationen
SP	Smart Phone
JS	Java Script
SHP	Smart Home Prototypen
NA	Nativen Applikationen
WA	Web Applikationen
HTML5	Hypertext Markup Language
CSS	Cascading Style Sheets
HyApp	Hybrid Applikationen
SW	Service Worker

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Einleitung

Durch die Markteinführung des Smart Phones(SP) hat sich unser Leben gravierend geändert. Nicht nur unsere Kommunikation, sondern unser Leben im Allgemeinen, ist durch dieses kleine Wundergerät erleichtert worden. Wir haben ständig das SP im Einsatz, zum Organisieren, zum Spielen, zum Musik hören, um unsere Kontakte zu pflegen und ab und zu wird es auch zum Telefonieren verwendet. Das SP hat nicht nur unseren Alltag beeinflusst, sondern auch das Internet und die Entwicklung von Webapplikationen. 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. Das Ganze benötigt natürlich eigene Entwickler die sich auf die jeweiligen Plattformen spezialisieren. Dies führt zu höheren Entwicklungskosten, unter anderem auch um das Produkt auf verschiedenen Plattformen betreiben zu können. In den letzten Jahren wurden, durch die immer besseren werdenden Browser, die Web Applikationenen (WA) stetig weiter verbessert und durch erweiterte Technologien wie den Progressive Web Applikationens (PWA) sind diese heute schon in der Lage mit den Native Apps zu konkurrieren.

1.1 Motivation

Wie im vorigen Kapitel beschrieben werden native Applikationen für ein bestimmtes Betriebsystem optimiert. Diese haben dann den Vorteil die Hardware des Gerätes nutzen zu können und somit sind komplexere Anwendungen realisierbar. Doch diese sind relativ kostspielig und auf Grund der vielzahl der diversen Apps in den App Stores nicht sehr lukrativ. Durchschnittlich werden (genaue Prozentzahl ermitteln) monatlich pro Nutzer runtergeladen und benötigen viel Speicherplatz auf dem Gerät. Die Progressive Web Applikationen(PWA) vereint die Vorteile von native App und von Webanwendungen und gibt dem Nutzer ein Gefühl, dass man es mit einer auf das System angepassten Anwendung zu tun hat.

1.2 Zielsetzung

Das Ziel dieser Arbeit ist es, einen Smart Home Prototypen(SHP) zur Demonstration zu entwickeln. Dem Prototypen werden die PWA typischen Features wie das Hinzufügen auf dem Startbildschirm, Offline arbeiten, die Pushfunktionen, das Zugreifen auf Gerätefunktionen und das Chachen über eine Clientseitig integrierte Datenbank, hinzugefügt. An diesen Features sollen die Vorteile, Nachteile, Entwicklung, Betrieb und User Experience betrachtet werden. Basis Technologien der Webentwicklung und verwendete Frameworks (z.B.: Java Script(JS),ReactJS, NodeJS, Yarn,...) werden in dieser Arbeit nicht behandelt.

2 Grundlagen

Wie in Kapitel 1 beschrieben, hat der stetige Zuwachs von Smart Phones SPs [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 [3] da.
Die PWAs sollen den Spagat zwischen diesen beiden Anwendungen schaffen. Eventuell könnte diese neue Form der Appentwicklung die traditionelle 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 und 2.2 dargestellt, voran.

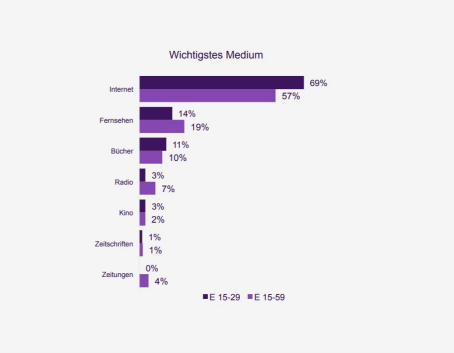


Abbildung 2.1: Internetnutzung [1]

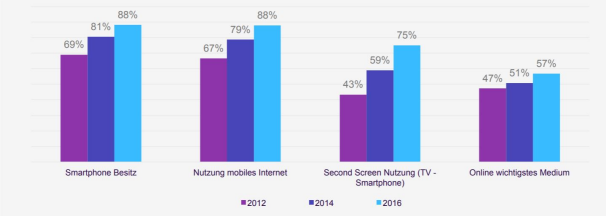


Abbildung 2.2: Smartphonenutzung [1]

2.1 Geschichte Softwareentwicklung

Um die Geschichte der Softwareentwicklung darstellen zu können, müssen wir als aller erstes die Frage stellen "Was ist Software?ünd wie ist eine Software definiert? Diese Frage stellen sich sicherlich alle mal die zum ersten Mal in ihrem Leben mit dieser Technologie in Berührung kommen. Eine genau Definition zu finden ist schwierig da die Software für die Gesamtheit eines Produktes steht . In [4] ist die Softwaretechnik wie folgt definiert:

> *„Zielorientierte Bereitstellung und systematische Verwendung von Prinzipien, Methoden und Werkzeugen für die arbeitsteilige, ingenieurmäßige Entwicklung und Anwendung von umfangreichen Softwaresystemen. Zielorientiert bedeutet die Berücksichtigung z.B. von Kosten, Zeit, Qualität.“([4] Seite 17).*

Im Laufe der Jahre wurden verschiedenste Softwaren 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 softwareorienttiere Unternehmen wie Microsoft oder SAP [5] [6].

2.2 Mobile Applikationen

Anfang des neuen Jahrtausends war die Vorstellung, dass das Mobiltelefon für uns sehr viele alltäglichen Aufgaben erledigt unvorstellbar, doch heute können wir uns das Leben ohne Mobiltelefon kaum vorstellen. Wir organisieren unser Leben damit und steuern unsere Haushaltsgeräte, unser Garagentor, verbinden uns mit unserem Auto usw. All diese Möglichkeiten werden durch Apps ermöglicht. Die Apps werden im Allgemeinen in 3 Kategorien aufgeteilt Native-, Web- und Hybridapps.

2.2.1 Native Apps

Nativen Applikationens (NA) (deutsch; angepasste Anwendung) 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 [7]. 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 [8].

2.2.2 Web Applikationen

Im Gegensatz zu den NAs sind Web Applikationen (WA) speziell Programmierte Webseiten [8]. WA funktionieren nach dem Server-Client Prinzip und werden vom Browser aufgerufen. In der Regel werden WAs auf der Basis von JS, CSS und HTML5 entwickelt. Die Verarbeitung erfolgt auf dem Webserver oder auf der Cloud. Client seitig werden die Ergebnisse der Datenverarbeitung angezeigt. Der größte Vorteil ist sicherlich der unkomplizierte Zugang im Gegensatz zu den NAs [9]. 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 [10].

2.2.3 Hybrid Applikationen

Hybrid Applikationen (HyApp) verbinden die Eigenschaften den in Kapitel 2.2.1 und 2.2.2 genannten Technologien. Zum einen verwenden sie die webbasierende Client-Server Technologie zum anderen kann man mit einer HyApp auf Gerätefunktionen wie Kamera und Kalender zugreifen [11].

2.2.4 Progressive Webapplikationen

Progressive Web Applikationenen sind im Grunde eine Weiterentwicklung von einer WA. 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 NA. Das Herausragende dabei ist, im Gegensatz zu einer HyApp, dass jede bestehende WA in eine PWA umgebaut werden kann. Durch Hinzufügen einer Manifest Datei und eines Service Worker (SW) werden Features hinzugefügt, die es ermöglichen offline zu arbeiten oder das Icon der App auf den Desktop oder Home-Bildschirm zu speichern [3]. Google definiert die PWA wie folgt:

- **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 NA
- **Aktuell** - Durch die Wartung des SW 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 Notfication erreicht
- **Installierbar** - Ermöglicht das Hinzufügen auf dem Startbildschirm
- **Verteilbar** - Einfache Freigabe über URL [12]

.

3 Basistechnologien

3.1 Aufbau PWA

In diesem Kapitel werden die Komponenten der Progressive Web Applikationen (PWA) allgemein erklärt. Weiter werden durch die Tabelle 3.2 die wichtigsten Punkte gegenübergestellt.

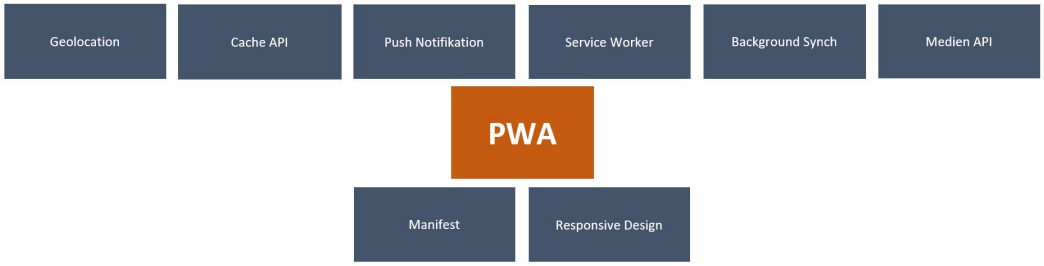


Abbildung 3.1: PWA Komponenten

3.2 PWA vs. Native Applikation vs Web Applikation

Folgende Punkte werden verglichen:

- **Veröffentlichung und Installation**
- **Zugriff**
- **Funktionen**

.

Kommentar: Tabelle für Funktionen fehlt noch!

	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 downgeloaded werden	Wird mit einem Klick auf dem Startbildschirm hinzugefügt	keine Funktion
Updates	über App-Store	Serverseitig	Serverseitig

Tabelle 3.1: Veröffentlichung und Installation [2]

	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	möglich	möglich	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 [2]

3.3 Web App Manifest

Das App Manifest ist eine JSON Datei die dem Browser verrät wie sich die WA bei der Installation auf dem Startbildschirm verhält. Im Manifest wird der Name, der Kurzname, die Größe, das Aussehen der Icons und weitere Eigenschaften definiert.

3.3.1 Bereitstellung des Web App Manifests

Die App Manifest.json Datei wird in die gleiche Ebene wie die Index.html Datei in das Projekt eingepflegt und und über den folgenden Link-Tag in der Index Datei bereitgestellt:

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

Listing 3.1: Manifest.json

Der Aufbau des Manifests ist wie in Listing ?? gezeigt aufgebaut:

```
{
  "name": "HackerWeb",
  "short_name": "HackerWeb",
  "start_url": ".",
  "display": "standalone",
  "background_color" : "#fff" ,
  "description": "Eine einfach lesbare Hacker News App
     .",
  "Icons": [{
    "src": "images/touch/homescreen48.png",
    "sizes": "48x48",
    "type": "image/png"
  }, {
    "src": "images/touch/homescreen72.png",
    "sizes": "72x72",
    "type": "image/png"
  }, {
    "src": "images/touch/homescreen96.png",
    "sizes": "96x96",
    "type": "image/png"
  }, {
    "src": "images/touch/homescreen144.png",
    "sizes": "144x144",
    "type": "image/png"
  }, {
    "src": "images/touch/homescreen168.png",
    "sizes": "168x168",
    "type": "image/png"
  }, {
    "src": "images/touch/homescreen192.png",
    "sizes": "192x192",
    "type": "image/png"
  }],
  "related_applications": [{
    "platform": "Web"
```

```
35    }, {
36      "platform": "play",
37      "url": "https://play.google.com/store/apps/details
           ?id=cheeaun.hackerweb"
38    }]
39  }
```

Listing 3.2: Manifest in das Projekt implementieren

[13]

3.3.2 Zum Startbildschirm hinzufügen

Um den Banner am mobilen Gerät anzuzeigen müssen folgende Kriterien erfüllt werden:

- 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

Wenn diese Punkte erfüllt sind startet Chrome ein beforinstallpromt wie in Listing 3.3

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

Listing 3.3: beforinstallprompt

3.4 Service Worker

Der Service Worker (SW) ist ein Script das der Browser im Hintergrund ausführt [14]. Mit der Hilfe des SW ist es möglich die WA offline zu betreiben, Push Notifikationen zu erhalten und gecachte Daten abzurufen. SW verhalten sich wie Proxy-Server, welche in einer Zwischenschicht vom Browser und dem Netzwerk sitzen. Ein SW wird von einem Worker-Kontext [15] ausgeführt, hat keinen DOM Zugriff und wird als Haupt-Java Script Thread verwendet [16].

3.4.1 Basis Architektur

Der Zyklus eines SWs ist von der Webseite getrennt. In der Installationsphase werden benötigte statische Dateien zwischengespeichert und erst danach ist der SW installiert. Die Installation erfolgt über die JavaScript-Funktion:

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

Listing 3.4: Service Worker Navigator

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

Abbildung 3.2: Basis Architektur Service Worker [17]

Um die neuen Seiten zu steuern muss der SW erneut geladen werden. In der Abbildung 3.3 ist eine vereinfachte Erstinstallation zu sehen:

3.4.2 Registrierung Service Worker

Um den SW zu registrieren muss folgender JS-Code in das Projekt im (genauen Pfad rausfinden) integriert werden.

```
1  if ('serviceWorker' in navigator) {
2    window.addEventListener('load', function() {
3      navigator.serviceWorker.register('/sw.js').then(function(
           registration) {
4        // Registration was successful
5        console.log('ServiceWorker registration successful with
             scope: ', registration.scope);
6      }, function(err) {
7        // registration failed :(
8        console.log('ServiceWorker registration failed: ', err);
9      });
10   });
11 }
```

Listing 3.5: Service Worker Register

Hier wird die Unterstützung durch den Browser geprüft.

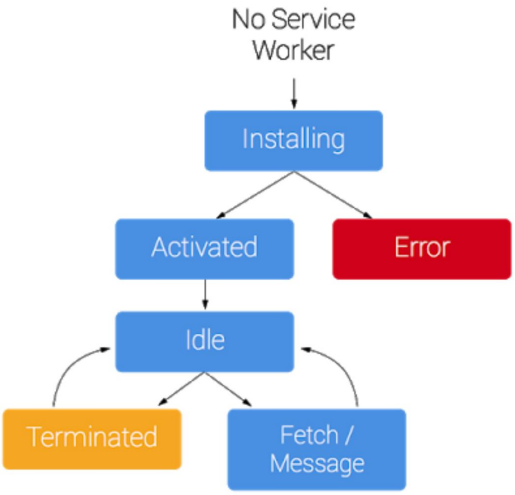


Abbildung 3.3: Erstinstallation Service Worker [14]

Der SW kann nach der Übernahme der Steuerung zwei Zustände übernehmen, entweder dieser wird beendet oder er übernimmt die Verwaltung der Netzwerkanfragen und der Nachrichten [14].

3.4.3 Cache API

Die SW API stellt eine Cache-Schnittstelle zum Speichern von Daten auf dem Browser, die über IndexDB [18] gespeichert und bei Bedarf aufgerufen werden können. Die API wurde ursprünglich für den SW entwickelt, diese kann aber von jedem Script verwendet werden. Wie die API gestaltet wird, hängt ganz von den Anforderungen der Applikation ab. Der Einstiegspunkt ist 'cache' wie man im folgenden Codebeispiel sehr gut erkennen kann.

```
1  self.addEventListener('install', function(event) {
2    event.waitUntil(
3      caches.open(cacheName).then(function(cache) {
4        return cache.addAll(
5          [
6            '/css/bootstrap.css',
7            '/css/main.css',
8            '/js/bootstrap.min.js',
9            '/js/jquery.min.js',
10           '/offline.html'
11         ]
12       );
13     })
14   );
15 });
```

Listing 3.6: Service Worker Cache

Wie imm Listing **??** zu sehen werden statische HTML, CSS und JS Dateien gecacht bevor der intall event des SW aufgerufen wird. Die Callbackfunktion ruft die Cache-API auf [19]. Um den Event aufzurufen werden Promises für Asynchrone Aufrufe verwendet [20]. Es gibt wie in [19] beschrieben noch andere Möglichkeiten um die Cache API nützlich in ein Projekt einzubauen.

3.5 Push Notifikation

Um dem User bei einer PWA das Gefühl einer Nativen Applikationen aufkommen zu lassen ist die Push Funktion unablässig. Erst diese Funktion in Kombination mit dem SW gibt den Web Applikationen die persönliche Nähe zum User [21].

3.5.1 Registrierung Push Notifikation

Um die Push Funktion zu integrieren muss die Registerfunktion des SW wie folgt erweitert werden:

```
1  if ('serviceWorker' in navigator && 'PushManager' in window) {
2    console.log('Service Worker and Push is supported');
3
4    navigator.serviceWorker.register('sw.js')
5    .then(function(swReg) {
6      console.log('Service Worker is registered', swReg);
7
8      swRegistration = swReg;
9    })
10   .catch(function(error) {
11     console.error('Service Worker Error', error);
12   });
13 } else {
14   console.warn('Push messaging is not supported');
15   pushButton.textContent = 'Push Not Supported';
16 }
```

Listing 3.7: Push Notifications

Hier wird wie in
Kapitel 3.4.2 die Browserunterstützung vom SW und den Push Benachrichtigungen überprüft. Bei fehlerlosem Durchlauf wird die SW.js Datei registriert [21].

3.5.2 Geolocation API

Die Geolocation API kann nach Zustimmung des Benutzers den Standort bestimmen. Diese Funktion kann verwendet werden um den Benutzer zusätzlichen Nutzen zu bringen, wie z.B.: Optimierung von Benutzeranfragen, bestimmen des Standortes und Backendaufnahmen von Standortdaten für Datensammlung.

3.5.3 Registrierung Geolocation API

Als erstes wird im Listing 3.8 der Support des Browsers überprüft werden.

```
1 // check for Geolocation support
2 if (navigator.geolocation) {
3   console.log('Geolocation is supported!');
4 }
5 else {
6   console.log('Geolocation is not supported for this Browser/OS.'
      );
7 }
```

Listing 3.8: Geolocation Support

```
1 window.onload = function() {
2   var startPos;
3   var geoSuccess = function(position) {
4     startPos = position;
5     document.getElementById('startLat').innerHTML = startPos.
        coords.latitude;
6     document.getElementById('startLon').innerHTML = startPos.
        coords.longitude;
7   };
8   navigator.geolocation.getCurrentPosition(geoSuccess);
9 };
```

Listing 3.9: Geolocation API

Wenn keine Errormeldung erscheint, wird die genaue Position im Listing 3.9 über die Methode `getCurrentPosition()` aufgerufen [22].

3.5.4 Camera API

3.5.5 Browser

4 Entwurf

In diesem Kapitel wird das Muster im Allgemeinen und die Anforderungen bzw. die Umsetzung der Progressive Web Applikationens betrachtet.

4.1 Model Native

4.2 Model PWA

4.3 Anforderungsanalyse

5 Implementierung

5.1 Umsetzung der Anforderungen

5.2 Ausgewählte Programmiersprache und IDE

5.3 Manifest

5.3.1 Aufbau

5.3.2 Implementierung

5.4 Aufbau Service Worker

5.4.1 Aufbau

5.4.2 Implementierung

5.5 Zugriff Cache API

5.5.1 Aufbau

5.5.2 Implementierung

5.6 Zugriff Device API

5.6.1 Aufbau

5.6.2 Implementierung

5.7 Fetch API

5.7.1 Aufbau

5.7.2 Implementierung

5.8 IndexDB

5.8.1 Aufbau

5.8.2 Implementierung

6 Funktionstest/Validierung

6.1 Ausgangsbedingung und Ausgrenzung

6.2 Komponententest

6.2.1 Add to Homescreen

6.2.2 Funktion Service Worker

6.2.3 Cache

6.2.4 Kamera

6.2.5 Geolocation

7 Fazit

Literaturverzeichnis

[1] Mindshare, *Über welche der folgenden Geräte nutzen Sie das Internet?*, https://de-statista-com.ezproxy.fh-salzburg.ac.at/statistik/daten/studie/742449/umfrage/umfrage-zur-internetnutzung-nach-geraetetyp-in-oesterreich-nach-alter/ (2018).

[2] Robert, *Progressive Web Apps (PWA) – Was ist das überhaupt und wie nutzt man sie?*, https://apptooltester.com/de/progressive-web-apps/ (12.03.2018).

[3] Google Developers, *Progressive Web Apps*, https://developers.google.com/web/progressive-web-apps/ (28.06.2018).

[4] H. Balzert, *Lehrbuch der Softwaretechnik: Softwaremanagement*, 2. Aufl. Heidelberg, Neckar: Springer Spektrum, 2008.

[5] Microsoft Corporation, *Microsoft Fast Facts*, https://news.microsoft.com/de-de/fast-facts/ (2018).

[6] SAP SE, *SAP: 46 Jahre Innovation*, https://www.sap.com/corporate/de/company/history.html (2018).

[7] 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).

[8] Margaret Rouse, Alexander Gillis, *DEFINITION native App*, https://searchsoftwarequality.techtarget.com/definition/native-application-native-app (2013).

[9] Stephan Augsten, *Defintion „Webanwendung“ Was ist eine Web App?*, https://www.dev-insider.de/was-ist-eine-web-app-a-596814/ (20.04.2017).

[10] 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).

[11] 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).

[12] Google Developers, *Your First Progressive Web App*, https://codelabs.developers.google.com/codelabs/your-first-pwapp/index.html#0 (2018).

[13] Lion Ralfs, *Bereitstellung eines Manifests*, https://developer.mozilla.org/de/docs/Web/Manifest (27.02.2018).

[14] Google Developers, *Your First Progressive Web App*, https://developers.google.com/web/fundamentals/primers/service-workers/ (2018).

[15] Dennis Sterzenbach, *Worker*, https://developer.mozilla.org/de/docs/Web/API/Worker (21.12.2017).

[16] Bitbruder, TobiDo, Heniz, *Service Worker API*, https://developer.mozilla.org/de/docs/Web/API/Service_Worker_API (30.01.2018).

[17] David Guan, *Using Service Workers*, https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers (03.07.2018).

[18] J Doose, *IndexedDB*, https://developer.mozilla.org/de/docs/IndexedDB (26.01.2018).

[19] Google Developers, *Caching Files with Service Worker*, https://developers.google.com/web/ilt/pwa/caching-files-with-service-worker (09.04.2018).

[20] J Nnkm, *Promise*, https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Promise (09.05.2018).

[21] Google Developers, *4.3.1 Einführung in Push-Benachrichtigungen im Web und Benachrichtigungen*, https://support.google.com/partners/answer/7336533?hl=de&ref_topic=7327985 (2018).

[22] Paul Kinlan, *User Location*, https://apptooltester.com/de/progressive-web-apps/ (02.07.2018).