Hochschule
Ravensburg-Weingarten
Technik | Wirtschaft | Sozialwesen

Bachelorarbeit

Progressive Web Apps – die Zukunft nativer Applikationen?
Konzeption und Realisierung einer Progressive Web App zum Einsehen von Teilnehmerdaten eines Events

Studiengang: Wirtschaftsinformatik und E-Business

Sascha Metz

Alte Dorfstraße 70

88662 Überlingen

Matrikelnummer: 26421

Email: sascha.metz@hs-weingarten.de

Erstprüfer: Prof. Dr. Marius Hofmeister

Zweitprüfer: Prof. Dr. Sebastian Mauser

Abgabedatum: 31.08.2019

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Problemstellung

In den letzten Jahren haben viele neue Technologien die Entwicklung von Webanwendungen stark verändert. Hierzu gehören unter anderem auch Single-Page Applications, die bereits bei vielen bekannten Seiten wie Netflix, Airbnb oder Shopify eingesetzt werden. Durch diese Art von Anwendung wird dem User eine performante und flüssige Bedienung der Webanwendung ermöglicht und die Usability deutlich angenehmer gestaltet als bei “klassischen” Webanwendungen.

Eine sich neue etablierende Technologie oder vielmehr ein Konzept stellen nun sogenannte Progressive Web Apps dar. Eine Progressive Web App kann als eine Mischung aus einer Responsive Website (nicht zwingend eine Single-Page Application) und einer nativen App betrachtet werden. Diese Anwendungen können dabei wie normale Webanwendungen im Browser benutzt werden mit dem großen Unterschied, dass diese auch als App auf dem Homescreen gespeichert/installiert werden können. Eine Progressive Web App hat dabei viele Gemeinsamkeiten mit einer nativen App. Es ist beispielsweise möglich, mit einer Progressive Web App Push Nachrichten an das Smartphone zu senden, Daten offline zur Verfügung zu stellen oder sogar die Benutzung der Kamera zu ermöglichen. Diese “Features” sind durch sogenannte Service Workers möglich, die in modernen Browsern durch JavaScript eingesetzt werden können. Durch Progressive Web Apps ist es also möglich, Webanwendungen zu bauen, die auf Smartphones als normale App verwendet werden können.

Dies würde vor allem für Unternehmen eine enorme Kosten- und Zeitersparnis bedeuten, da nicht mehr für jedes Betriebssystem eine eigene App entwickelt werden muss, sondern die Webapp sowohl auf dem Computer als auch auf dem Smartphone läuft. Ebenfalls entfällt der Schritt, dass die App extra aus dem App Store geladen werden muss, was die Absprungrate von potenziellen Kunden deutlich verringern könnte.

1.2 Zielsetzung

Das Konzept von Progressive Web Apps ist im Internet ein relativ neues, aber momentan aufstrebendes Thema. Da selbst der Internet-Gigant Google großes Potential in dieser Technologie sieht, lässt sich hinsichtlich Progressive Web Apps positiv in die Zukunft blicken. Vor allem in Unternehmen ist es wichtig, den Aufwand und

die Kosten bei der Entwicklung gering zu halten, aber keinen Verzicht auf Qualität hinsichtlich Usability und Funktionalität zu machen.

Das Ziel dieser Arbeit ist deshalb die Untersuchung, ob Progressive Web Apps eine Alternative oder einen Mehrwert gegenüber nativen Apps darstellen. Hierzu werden zuallererst die technischen und funktionalen Merkmale von Progressiven Web Apps aufgestellt. Weiterhin erfolgt eine Ist-Analyse von bestehenden Progressive Web Apps, die von Unternehmen bereits realisiert wurden.

Durch die praktische Umsetzung einer Progressive Web App werden die einzelnen Aspekte wie Aufwand der Programmierung, Usability und Funktionalität untersucht. Die praktische Umsetzung soll dabei in Form einer Single-Page Application erfolgen, die programmiert wird und danach eine Migration zu einer Progressive Web App erfolgt. Zur Evaluierung der Progressive Web App wird ein Kriterienkatalog hinsichtlich des Entwicklungsaufwands und der User Experience angefertigt.

Weiterhin erfolgt eine Qualitätsprüfung und Optimierung mit Hilfe von Tools wie Lighthouse.

Bei der Progressive Web App handelt es sich um eine App für Veranstalter der Plattform Lumis.ai, die durch diese Anwendung die Teilnehmer für ihre Events einsehen können. Dabei muss die Web App über eine Authentifizierung verfügen, da User nur Teilnehmer von Events einsehen dürfen, die auch von ihnen erstellt wurden.

2 Grundlagen

2.1 Begriffsdefinitionen

2.1.1 Event

Unter einem Event versteht man grundsätzlich eine Veranstaltung jeglicher Art. Ein Event ist dabei ein Zusammenkommen von Personen, die meist einer bestimmten Zielgruppe angehören, die ein bestimmtes Interessengebiet teilen. Das Event selbst ist dabei ein zeitlich begrenztes Ereignis, das durch einen Veranstalter organisiert und durchgeführt wird. Ein Veranstalter kann dabei eine Person, eine Personengruppe, eine Firma etc. sein. Ein Event kann aufgrund verschiedener Ziele durchgeführt werden. Unter anderem gibt es Events, bei denen das Erlebnis und der Unterhaltungsfaktor im Vordergrund stehen und andere Events werden wiederum veranstaltet, um Informationen zu vermitteln oder Produkte zu präsentieren. Dabei gibt

es zahlreiche verschiedene Arten von Veranstaltungen - beispielsweise Festivals, Messen oder Konferenzen. Einige bekannte Events sind zum Beispiel das Oktoberfest in München oder die Eurobike Messe in Friedrichshafen, aber auch eine private Firmenfeier stellt ein Event dar. Ein Event steht dabei nicht zwingend in Verbindung mit einer monetären Gegenleistung. Events können sowohl kostenpflichtig als auch kostenlos sein.[1]

2.1.2 Web App

Eine Web App oder auch Webanwendung genannt ist eine Anwendung, die über das Client-Server-Modell umgesetzt wird. Beim Client-Server-Modell wird dabei die Anwendung durch einen Web-Server bereitgestellt, wodurch der User diese Anwendung nicht lokal auf dem Computer installieren muss. Der Client wird dabei durch den Webbrowser repräsentiert. Die Funktionsweise in einem Client-Server-Modell basiert darauf, dass der Client eine Anfrage an den Web-Server stellt (Request). Der Server verarbeitet diese Anfrage und gibt eine Antwort (Response) zurück. Diese Antwort ist beispielsweise ein HTML Dokument, das im Browser dargestellt werden kann, wodurch der User diese Web App benutzen kann. Die Kommunikation zwischen Client und Server erfolgt dabei über das Hypertext Transfer Protocol (HTTP). Da die Datenübertragung in diesem Protokoll jedoch nicht verschlüsselt ist, gibt es das ergänzende Protokoll HTTPS (Hypertext Transfer Protocol Secure), was eine verschlüsselte Übertragung zwischen Client und Server gewährleistet. [2] [3] [4]

2.1.3 Single-Page Application

Im Gegensatz zu einer klassischen Webanwendung besteht eine Single-Page Applikation nur aus einem einzigen HTML Dokument. Ändert sich der Inhalt in einer Single-Page Application, wird dieser dynamisch geladen und es erfolgt kein neuer Aufruf der Seite im Browser, was oftmals zu langen Ladezeiten führt. Der Vorteil dabei besteht vor allem in der reduzierten Kommunikation zwischen Client und Server, da nicht die komplette Seite nochmals übertragen werden muss, sondern nur Teile der Applikation, die sich geändert haben. Single-Page Applications werden dabei mit JavaScript, HTML und CSS umgesetzt. Viele bekannte Webseiten wie

[1] Vgl. (Rück, 2018)
[2] Vgl. (Srocke & Karlstetter, 2017)
[3] Vgl. (Client-Server — e-teaching, 2015)
[4] Vgl. (Elektronik Kompendium, 2019)

Netflix, Airbnb, Instagram oder Shopify verwenden diese Art der Webanwendung bereits erfolgreich. [5]

2.1.4 Progressive Web App

Unter einer Progressive Web App versteht man eine Mischung aus einer normalen Website und einer App. Im Grunde gesehen ist eine Progressive Web App nichts anderes als eine Webseite, die über den Browser aufgerufen werden kann. Der große Unterschied hierbei ist, dass diese Seiten Eigenschaften von Smartphone Apps aufweisen. Diese „Webseiten“ können als App auf dem Smartphone gespeichert werden ohne einen Installationsprozess zu durchlaufen wie bei herkömmlichen Apps aus dem App Store. Verwendet man diese App nun, ruft man im Grunde gesehen eine Webseite über den Browser auf, jedoch merkt der Nutzer nichts davon, da bei Progressive Web Apps die Browserleiste, über die man normalerweise die URL eingibt, nicht angezeigt wird. Progressive Web Apps können wie herkömmliche Apps auch offline verwendet werden. Solche Funktionalitäten werden durch sogenannte Service Workers realisiert. [6] [7]

2.1.5 Native Applikation

Eine native Applikation zeichnet sich dadurch aus, dass sie in einer Programmiersprache entwickelt wurde, die plattformspezifisch ist. Wird die App in iOS entwickelt, handelt es sich bei der Programmiersprache um Swift oder Objective-C. Android Apps hingegen werden mit Java oder Kotlin entwickelt. Die Entwicklung dieser plattformspezifischen Apps erfolgt mit Hilfe eine Software Development Kit (SDK), das vom jeweiligen Hersteller des Betriebssystems bereitgestellt wird (iOS = Apple, Android = Google). Diese SDKs unterstützen den Entwickler beim Umsetzen der App durch die Bereitstellung verschiedenster Programmierwerkzeuge. [8] [9]

Da native Apps für eine spezifische Plattform entwickelt werden, funktionieren diese auch nur dort. Eine App, die speziell für Android entwickelt wurde, kann nicht auf einem iOS Betriebssystem laufen. Diese Plattformbeschränkung bringt sowohl Vorteile als auch Nachteile mit sich. Beispielsweise ist eine native App perfekt mit der

[5] Vgl. (Domin, 2018)
[6] Vgl. (Searchmetrics, 2017)
[7] Vgl. (Google Developers - Progressive Web Apps , 2019)
[8] Vgl. (Mobile Zeitgeist, 2017)
[9] Vgl. (Dev Insider, 2018)

jeweiligen Plattform kompatibel, wodurch die bestmöglichste Performance erzielt werden kann. [10]

3 Progressive Web App

3.1 Architektur

Für das Rendering einer Webseite gibt es im Grundlegenden zwei Ansätze.

Server-side rendering (SSR) bedeutet, dass die Webseite auf dem Server gerendert wird. Dadurch lädt die Seite zwar beim ersten Aufruf im Gegensatz zum zweiten Ansatz sehr schnell, jedoch stellt jede weitere Seite, zu der navigiert wird, eine neue Anfrage an den Server. Die neue Anfrage zu der Seite muss den gesamten Prozess nun nochmals durchlaufen - auch wenn sich der Inhalt nur minimal verändert hat.

Client-side rendering (CSR) hingegen lädt den Inhalt einer Website komplett auf der Client Seite. Der erste Seitenaufruf dauert länger als beim Server-side rendering, da beim ersten Aufruf die benötigten Dateien „gedownloadet“ werden müssen und auf der Client Seite die Seite gerendert werden muss. Der Vorteil, der sich hieraus ergibt ist, sobald man zu einer neuen Seite navigiert: Es wird nicht die komplette Seite nochmals neu geladen, sondern nur die Teile der Seite, die sich tatsächlich verändert haben – das passiert auf der Client Seite. Dadurch wird eine enorme Geschwindigkeit erzielt, wodurch der Benutzer sehr geringe Wartezeiten hat.

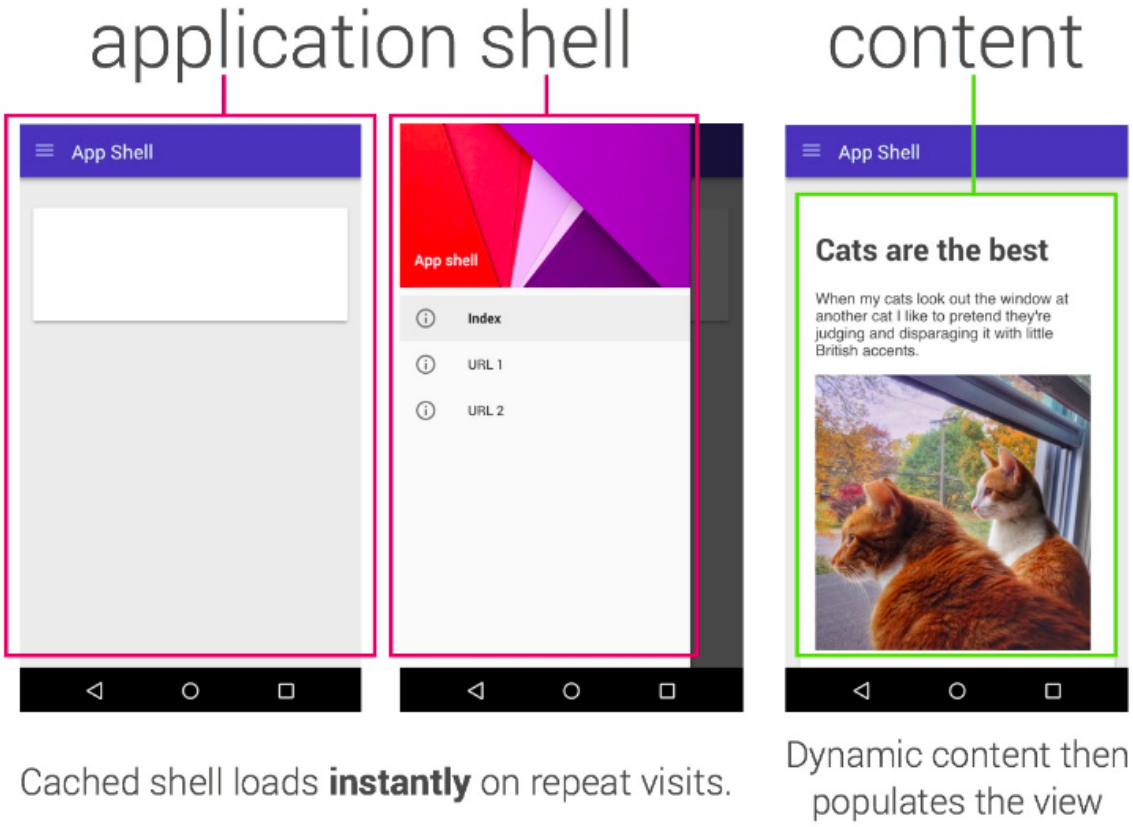
Beide Ansätze haben ihre Vor- und Nachteile. Vereint man jedoch SSR und CSR, kann dies zu einem bestmöglichen Ergebnis führen. Die Vereinigung von SSR und CSR sieht dabei folgendermaßen aus: Beim ersten Aufruf der Seite wird diese auf dem Server gerendert, wodurch der initiale Seitenaufruf sehr schnell verarbeitet wird. Sobald die Seite das erste Mal aufgerufen wird, erfolgt jede weitere Aktion auf der Client Seite. Navigiert man also nun innerhalb der Seite auf eine weitere Seite, wird der Inhalt auf der Client Seite gerendert, wodurch sich die Ladezeit enorm verringert. [11] [12] [13]

Progressive Web Apps können prinzipiell alle Ansätze verwenden, jedoch ist der beliebteste Ansatz das Konzept der sogenannten App Shell (Application Shell). Dieses Konzept beschreibt im Grunde die zuvor beschriebene Mischung aus Server-

[10] Vgl. (1&1 IONOS Digital Guide, 2019)
[11] Vgl. (Mozilla Developer Network, 2019)
[12] Vgl. (Vega, 2017)
[13] Vgl. (Google Developers - Progressive Web App Architectures, 2019)

side rendering und Client-side rendering. Die App Shell Architektur besteht aus zentralen Elementen, die erforderlich sind, damit die Anwendung ohne Internetverbindung funktionieren kann. Die App Shell bezieht sich praktisch auf die lokalen Ressourcen, die für das Skelett der Benutzeroberfläche benötigt wird. [14] [15]

Abbildung 1: Application Shell



Quelle: (Google Developers - Progressive Web App Architectures, 2019)

3.2 Features

Die Features, die eine Progressive Web App bietet, stellen zugleich auch die Voraussetzungen, die erfüllt werden müssen, um der Bezeichnung Progressive Web App gerecht zu werden. Eine Progressive Web App ist keine spezielle Technologie, sondern vielmehr ein Begriff, der die Verwendung spezifischer moderner Plattformfunktionen beschreibt. Um eine Web App als Progressive Web App bezeichnen zu können, müssen folgende Punkte erfüllt sein:

[14] Vgl. (Google Developers - Progressive Web App Architectures, 2019)
[15] Vgl. (Love, Progressive Web Application Development by Example, 2018, S. PWA technical requirements)

3.2.1 Responsiveness

Ein Responsive Web-Design passt sich der Screen-Größe eines Geräts automatisch an. Dadurch wird gewährleistet, dass egal von welchem Endgerät eine Webseite aufgerufen wird, die Benutzerfreundlichkeit nicht verloren geht. Ein Responsive Design wird dabei meist über sogenannte Media-Queries realisiert, die in CSS3 zur Verfügung stehen. [16]

Beispielsweise sieht eine Media-Query folgendermaßen aus (In diesem Beispiel wird ein Mobile-First Ansatz benutzt, was bedeutet, dass zuerst die mobile Version umgesetzt wird und danach die Version für größere Screens):

```css
.header {
  position: sticky;
  top: 0;
  left: 0;
  width: 100%;
  height: 50px;
  background-color: royalblue;
}

@media (min-width: 1200px) {
  .header {
    position: fixed;
    width: 300px;
    height: 100%;
  }
}
```

Hier wird für mobile Versionen ein schmaler Header am oberen Bildschirmrand angezeigt. Sobald die Breite des Bildschirms mindestens 1200px hat, wird der Header fixiert an der Seite angezeigt.

3.2.2 Discoverable (Auffindbarkeit)

Progressive Web Apps sind im Grunde gesehen nichts anderes als Websites, jedoch mit einigen extra Features. Deshalb müssen sie auffindbar sein, wofür mehrere Vorrausetzungen erforderlich sind. Zum einen muss die Web App eine URL besitzen, damit Suchmaschinen wie Google die Website indexieren können. Zum

[16] Vgl. (wendweb, 2019)

anderen muss erkennbar sein, dass es sich hierbei um eine Progressive Web App handelt und nicht um eine gewöhnliche Website. [17] [18]

Hierzu gibt es das sogenannte Web App Manifest. Dabei handelt es sich um eine Datei im JSON Format, die eine der ersten Komponenten der Progressive Web App darstellt. In dieser Datei erfolgt eine Art Konfiguration, in der festgelegt werden kann, wie der Name der App lautet, welche Icons verwendet werden sollen, ob ein Splash-screen gesetzt werden soll etc. [19]

Auf diese Manifest Datei wird im Head Teil des HTML Dokuments mit dem folgenden Meta Tag verwiesen: [20]

```
<head>
   ...
  <link rel="manifest" href="manifest.json">
</head>
```

Eine Manifest Datei sieht beispielsweise folgendermaßen aus: [21]

```
{
  "short_name": "Messenger",
  "name": "Facebook Messenger",
  "icons": [
    {
      "src": "/assets/icons/icon-192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
    {
      "src": "/assets/icons/icon-512.png",
      "type": "image/png",
      "sizes": "512x512"
    }
  ],
  "start_url": "/",
  "display": "standalone",
  "theme_color": "#3b5997"
}
```

[17] Vgl. (Seobility, 2019)
[18] Vgl. (WADI, Progressive Web Apps HQ - Discoverable, 2019)
[19] Vgl. (Liebel, Progressive Web Apps - Das Praxisbuch, 2018, S. 118ff)
[20] Vgl. (Love, Progressive Web Application Development by Example, 2018, S. The web manifest specification)
[21] Vgl. (Gaunt & Kinlan, The Web App Manifest, 2019)

Der **short_name** wird an Stellen verwendet, an denen der Platz sehr begrenzt ist - beispielsweise auf dem Homescreen für die Anzeige des Namens der App. Ist hingegen viel Platz für Text vorhanden, wird die Eigenschaft **name** verwendet.

Die **icons** Eigenschaft stellt ein Array von Icons in verschiedensten Größen dar, die für den Homescreen verwendet werden. Je nach Plattform wird eine unterschiedliche Größe benötigt, weshalb zu empfehlen ist, möglichst viele Größen zur Verfügung zu stellen.

Die **start_url** ist der Einstiegspunkt der App, die aufgerufen wird, sobald man diese öffnet. Diese wurde eingeführt, um zu vermeiden, dass ein verschachtelter Link als Einstiegspunkt verwendet wird, wenn man die PWA beispielsweise von einer Unterseite zum Homescreen hinzufügt.

Die **display** Eigenschaft legt fest, in welchem „Modus" die App gestartet werden soll. Die Option *fullscreen* startet die App beispielsweise im Fullscreen Modus. Wird hingegen *standalone* verwendet, wird die App ebenfalls im Fullscreen Modus gestartet mit dem Unterschied, dass einige Browser Elemente wie die Statusbar angezeigt werden.

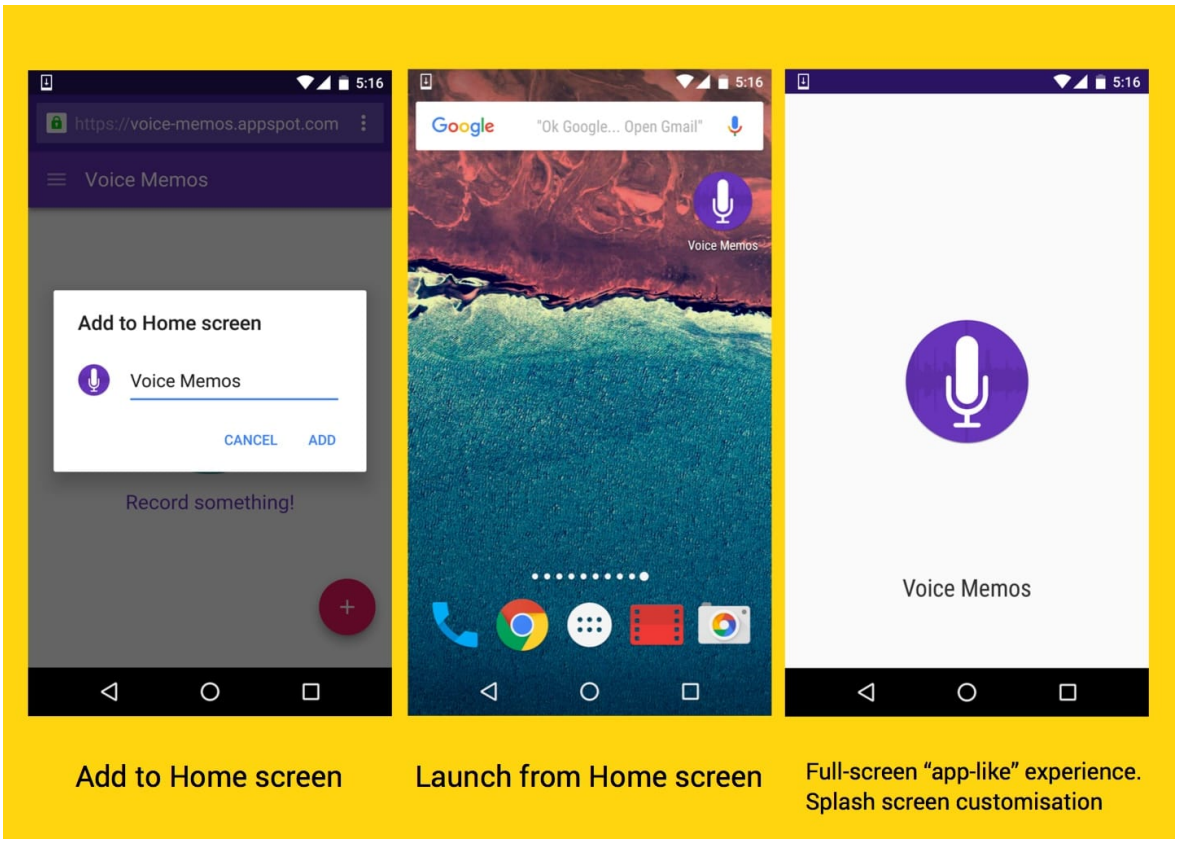
theme_color wird je nach Betriebssystem unterschiedlich verwendet und definiert das Farbschema der App.

3.2.3 Installable (Installierbarkeit)

Der Begriff Installation ist bei Progressive Web Apps nicht gleichzusetzen mit der Installation einer App aus dem Appstore. Die Installation bei Progressive Web Apps bedeutet, dass die PWA auf dem Homescreen abgespeichert wird, wodurch lediglich eine Art Verknüpfung erfolgt. Dabei erfolgt jedoch kein Download, auf den der User warten muss, wie es bei „normalen" Apps der Fall ist. Hierdurch wird dann kein zusätzlicher Speicherplatz eingenommen. Um diese Installation auszuführen wird das zuvor beschriebene Manifest benötigt. [22] [23]

[22] Vgl. (Liebel, Progressive Web Apps - Das Praxisbuch, 2018, S. 120)
[23] Vgl. (WADI, Progressive Web Apps HQ - Installable, 2019)

Abbildung 2: Installation einer Progressive Web App



Quelle: (Osmani, 2015)

3.2.4 Linkable (Verlinkbarkeit)

Linkable bedeutet, dass die Progressive Web App über eine URL verfügen muss. Dadurch kann diese einfach geteilt werden und es wird kein App Store benötigt. [24]

3.2.5 Progressiveness

Die Web App sollte für jeden Nutzer funktionieren, egal welcher Browser verwendet wird. Dabei sollte die Progressive Web App grundlegende Funktionalität anbieten, die plattformübergreifend verwendet werden kann. Wird beispielsweise der Internet-Explorer von Microsoft verwendet, der keine Service Worker unterstützt, fehlt eine wichtige Komponente, die für die volle Funktionalität einer Progressive Web App benötigt wird. Anstatt jedoch gar nicht zu funktionieren werden lediglich die grundlegenden Funktionen angeboten. Wird hingegen ein moderner Browser verwendet, der Progressive Web Apps unterstützt, wird dem User eine erweiterte Funktionalität angeboten. [25] [26]

[24] Vgl. (Liebel, Progressive Web Apps - Das Praxisbuch, 2018, S. 126f)
[25] Vgl. (Liebel, Progressive Web Apps - Das Praxisbuch, 2018, S. 100ff)
[26] Vgl. (Osmani, 2015)

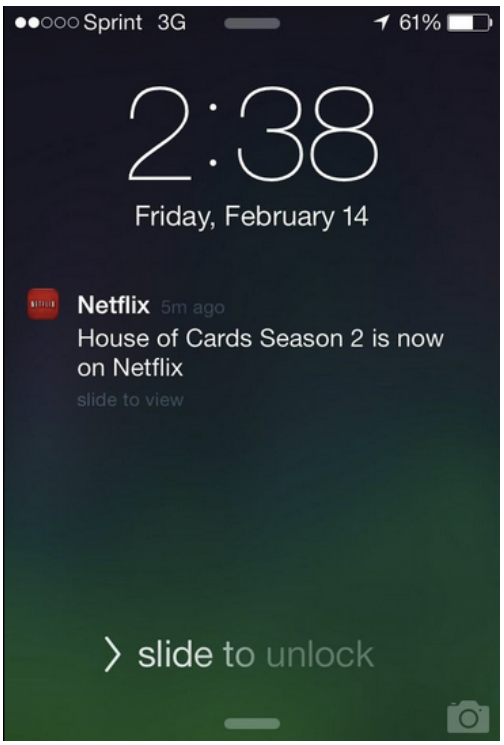
3.2.6 Network independent (Netzwerkunabhängigkeit)

Die Unabhängigkeit von einer Internetverbindung ist ein wichtiges Feature einer Progressive Web App, die durch einen sogenannten Service Worker realisiert werden kann. Dieser wird durch eine JavaScript Datei implementiert und kann durch Caching Inhalte nach wie vor korrekt anzeigen, auch wenn momentan keine Internetverbindung besteht. [27]

3.2.7 Re-engageable

Re-engageable zielt darauf ab, den Nutzer zur Wiederverwendung der App zu bewegen. Dies ist möglich durch die Verwendung von Push-Benachrichtigungen. Durch diese Art der Benachrichtigung hat der Anwender einen Anreiz, die App zu öffnen. Am Beispiel von Netflix kann diese Re-engageable Funktionalität demonstriert werden. Netflix sendet dem User eine Push-Benachrichtigung sobald eine neue Staffel der Lieblingsserien vorhanden ist. Dadurch wird der Nutzer dazu verleitet auf diese Benachrichtigung zu klicken, wodurch er die App verwendet. [28] [29]

Abbildung 3: Push-Benachrichtigung von Netflix



Quelle: (Marchick, 2015)

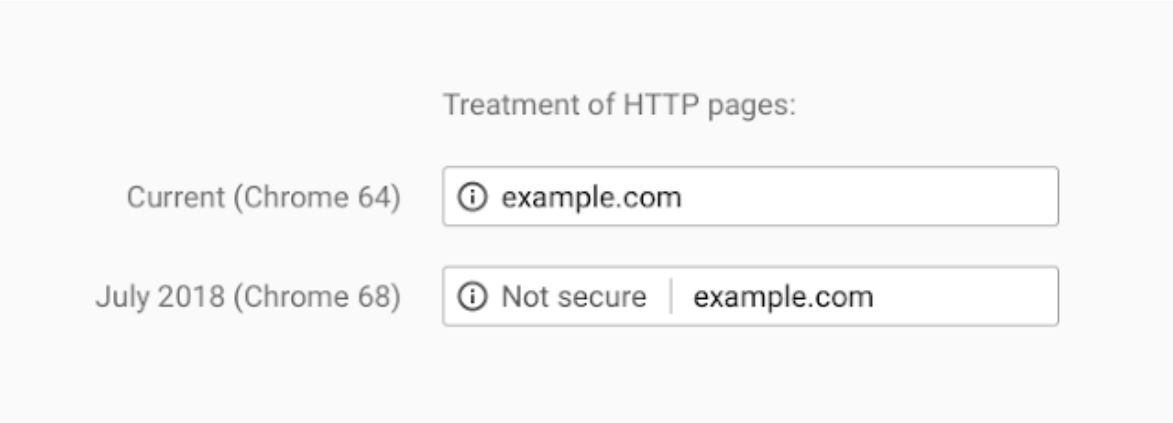
[27] Vgl. (Liebel, Progressive Web Apps - Das Praxisbuch, 2018, S. 106ff)
[28] Vgl. (Marchick, 2015)
[29] Vgl. (Liebel, Progressive Web Apps - Das Praxisbuch, 2018, S. 123ff)

3.2.8 Sicherheit

Progressive Web Apps müssen eine verschlüsselte Übertragung durch die Verwendung von HTTPS (Hypertext Transfer Protocol Secure) garantieren. Da Service Workers im Hintergrund ausgeführt werden und teilweise für Aufgaben verantwortlich sind, die mit sensiblen Daten handhaben, ist es wichtig, die Übertragung von Daten durch HTTPS zu schützen. In der heutigen Zeit ist es relativ einfach, eine Seite mit HTTPS zu schützen. Dank der Zertifizierungsstelle Let's Encrypt gibt es HTTPS Zertifikate mittlerweile sogar kostenfrei. [30] [31]

Um die Sicherheit des Webs zu fördern, haben Browser über die letzten Jahre die URL Leiste so angepasst, damit der User sofort weiß, ob man auf dieser Webseite sicher unterwegs ist oder nicht. Beispielsweise hat Google Chrome mit der Version 68 im Juli 2018 folgende Änderung eingeführt: [32]

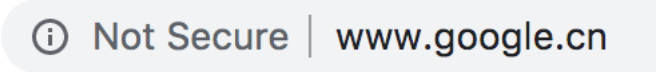
Abbildung 4: Behandlung von HTTP Seiten in Google Chrome



Quelle: (A secure web is here to stay, 2018)

Webseiten ohne HTTPS werden dabei mit einer Nachricht „Not secure" markiert, was sich negativ auf die User Experience auswirken kann. Google geht diesen Schritt, um sich für ein sicheres Internet einzusetzen.

Abbildung 5: Unsichere Seite mit HTTP in Google Chrome



[30] Vgl. (Let's Encrypt, 2019)
[31] Vgl. (Love, Progressive Web Application Development by Example, 2018, S. Making Your Website Secure)
[32] Vgl. (A secure web is here to stay, 2018)

3.3 Browserkompatibilität

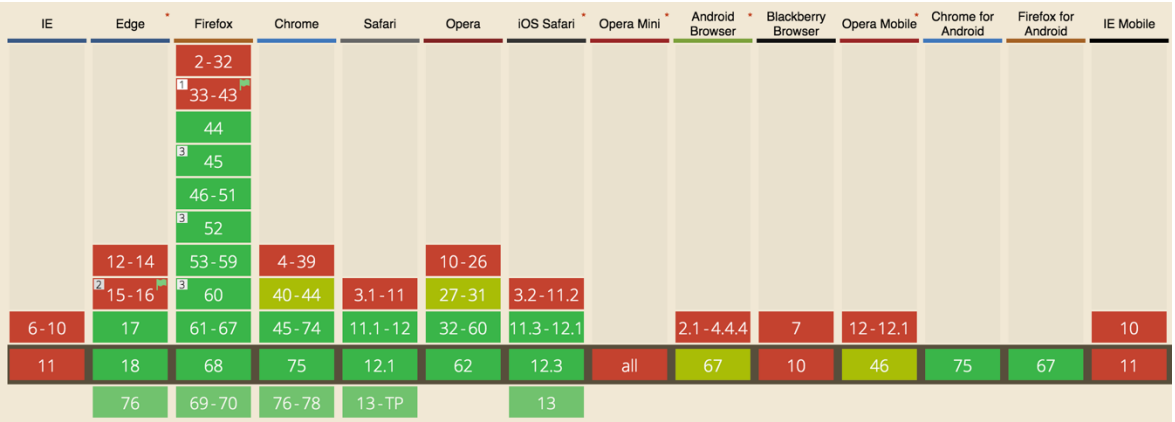
Für Progressive Web Apps dient ein Browser als Anwendungsplattform. Auch wenn die App auf dem Homescreen installiert/gespeichert wird, läuft die eigentliche Anwendung nach wie vor im Browser des Endgeräts. Dies ist für den User jedoch nicht erkennbar, da mit der entsprechenden Konfiguration in der Manifest Datei keine browserspezifischen Elemente vorhanden sind, wie zum Beispiel die URL Leiste. [33]

Progressive Web Apps zeichnen sich durch viele verschiedenen Funktionalitäten aus. Einige dieser Funktionalitäten wie die Offline Verfügbarkeit oder Push Benachrichtigungen sind nur möglich, wenn der Browser dies auch unterstützt. Wird eine Funktion nicht unterstützt, läuft die Progressive Web App dennoch weiterhin, da diese abwärtskompatibel entwickelt wird, wodurch zwar nicht das volle Potenzial der Progressive App genutzt werden kann, jedoch keine Einschränkung in der Grundfunktionalität der App erfolgt.

Progressive Web Apps werden grundlegend von allen modernen Browsern unterstützt, sofern es sich um die neueste Version handelt. [34]

Eine wichtige Technologie für Progressive Web Apps sind dabei die sogenannten Service Workers, die vom Browser unterstützt werden müssen. Der Browserunterstützung hierfür sieht derzeit folgendermaßen aus:

Abbildung 6: Browserunterstützung für Service Workers



Quelle: (Can I use, 2019)

[33] Vgl. (Love, Progressive Web Application Development by Example, 2018, S. What are PWAs?)
[34] Vgl. (Can I use, 2019)

Wie in der Grafik zu sehen ist, werden Service Workers von den modernen Browsern Edge, Firefox, Chrome, Safari und Opera vollständig unterstützt. Der Internet Explorer in der neuesten Version hingegen bietet hierzu keine Unterstützung mehr.

Da die Entwicklung des Internet Explorers jedoch seit 2015 von Microsoft zugunsten des neusten Browsers Edge eingestellt wurde, wird es auch in Zukunft keine Unterstützung für Funktionalitäten für Progressive Web Apps geben. [35]

3.4 Einsatz von Progressive Web Apps

Der Einsatz von Progressive Web Apps ist sehr vielfältig und letztlich sollte es möglich sein, jede native App mit einer Progressive Web App zu ersetzen.

Da es bei Progressive Web Apps keinen App Store benötigt, um eine App zu installieren, gibt es mehrere Seiten, die eine Sammlung von Progressive Web Apps anbieten. Hierzu gehört die Seite Appscope (https://appsco.pe), worüber viele bekannte Progressive Web Apps installiert werden können.

Zu den Apps, die installiert werden können, gehören unter anderem viele große Unternehmen wie Twitter, Uber, Lyft, Instagram, Google Maps und Pinterest. [36]

Einsatz der Progressive Web App anhand von Twitter:

Von den monatlich 326 Millionen Nutzern nutzen 80% Twitter über das Smartphone. Das Ziel von Twitter war es, mit der Entwicklung der Progressive Web App „Twitter Light“ eine bessere User Experience zu schaffen sowie verkürzte Ladezeiten und ein besseres User Engagement zu erzielen. [37]

Die Schwierigkeit der Nutzerbindung im mobilen Web löste Twitter durch die „Zum Homescreen hinzufügen“ Funktionalität von Progressive Web Apps, die je nach Browser automatisch empfiehlt, die App zu installieren. Nach Implementierung dieser Funktion konnte Twitter 250.000 Unique Users gewinnen, die im Schnitt 4-mal täglich die Progressive Web App über den Homescreen starteten.

Weiterhin gibt es enorme Einsparungen beim Datenkonsum. Die Progressive Web App ist nur 600KB groß, die native App für Android hingegen 23,5MB.

Da viele Twitter Nutzer die App auch über 2G oder 3G nutzen, ist eine geringer Datenverbrauch enorm wichtig, um die App schnell zu laden. Durch die Progressive

[35] Vgl. (Mühlroth, 2019)
[36] Vgl. (Appscope, 2019)
[37] Vgl. (Twitter by the Numbers, 2019)

Web App lädt die Seite nahezu sofort aufgrund der Verwendung von Caching durch Service Workers. Die Geschwindigkeit hat sich durch die Progressive Web App dabei um 30% verbessert. [38] [39]

3.5 Zusammenspiel von Progressive Web App und Single-Page Application

Sämtliche zuvor genannten Kriterien müssen erfüllt werden, um als Progressive Web App zu gelten. Google definiert unter anderem dem Begriff **Engaging** als eines der Kriterien. Unter diesem Kriterium werden mehrere Eigenschaften zusammengefasst wie Installierbarkeit und Re-engageable. Wichtig zu beachten ist auch, wie Google das **Engaging** Kriterium beschreibt: [40]

„Eine Progressive Web App sollte sich beim Bedienen wie eine normale App anfühlen und ein großartiges Nutzererlebnis bieten“. [41]

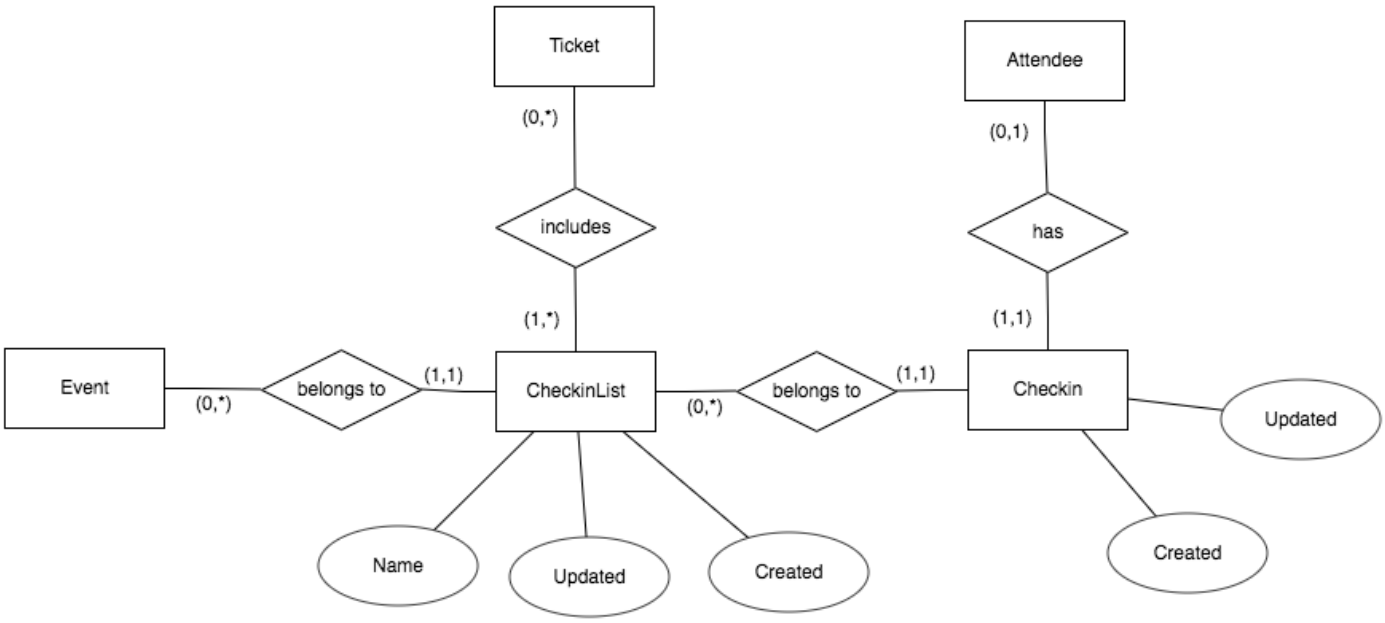
Um dieses Nutzererlebnis zu erschaffen, das eine native App bietet, ist die Grundlage einer Progressive Web App oftmals eine Single-Page Application, da die Bedienung sehr flüssig und vor allem schnell läuft. Jedoch ist zu erwähnen, dass nicht jede Progressive Web App automatische eine Single-Page Application ist, da dies kein zwingendes Kriterium darstellt. Gleichzeitig ist auch nicht jede Single-Page Application eine Progressive Web App. Eine Single-Page Application hat sehr gute Voraussetzungen für die Migration zu einer Progressive Web App. Erst nach Erfüllung der notwendigen Kriterien ist eine Single-Page Application eine Progressive Web App. [42] [43]

[38] Vgl. (Google Case Studies, 2017)
[39] Vgl. (Gallagher, 2017)
[40] Vgl. (Progressive Web Apps | Google Developers, 2019)
[41] (Progressive Web Apps | Google Developers, 2019)
[42] Vgl. (Love, Why Progressive Web Applications Are Not Single Page Applications, 2018)
[43] Vgl. (Kölpin, 2017)

4 Anforderungsanalyse / Planung

4.1 Datenbankmodell (ER-Modell)

Abbildung 7: Datenbankmodell der Checkin Liste



Das Datenbankmodell für die Umsetzung der Checkin App ist als Entity-Relationship Modell dargestellt unter der Verwendung der Min/Max Notation zur Beschränkung der Kardinalität einer Beziehung zwischen Entitätstypen. Die Entitäten, die dabei eine Rolle spielen, sind Event, Ticket, Attendee, CheckinList und Checkin.

Tabelle 1: Beschreibung der Entitäten des ER-Modells

Entität	**Beschreibung**
Event	Veranstaltung jeglicher Art durch den Veranstalter (Organizer).
Ticket	Eintrittskarte für die Veranstaltung, die gekauft werden kann.
Attendee	Teilnehmer, der ein Ticket erworben hat.
CheckinList	Eine Checkin Liste für eine Veranstaltung, die einschränkt, welche Tickets über diese Liste eingecheckt werden können.
Checkin	Ein Eintrag in der Checkin Liste, der erstellt wird sobald ein Teilnehmer eingecheckt wird.

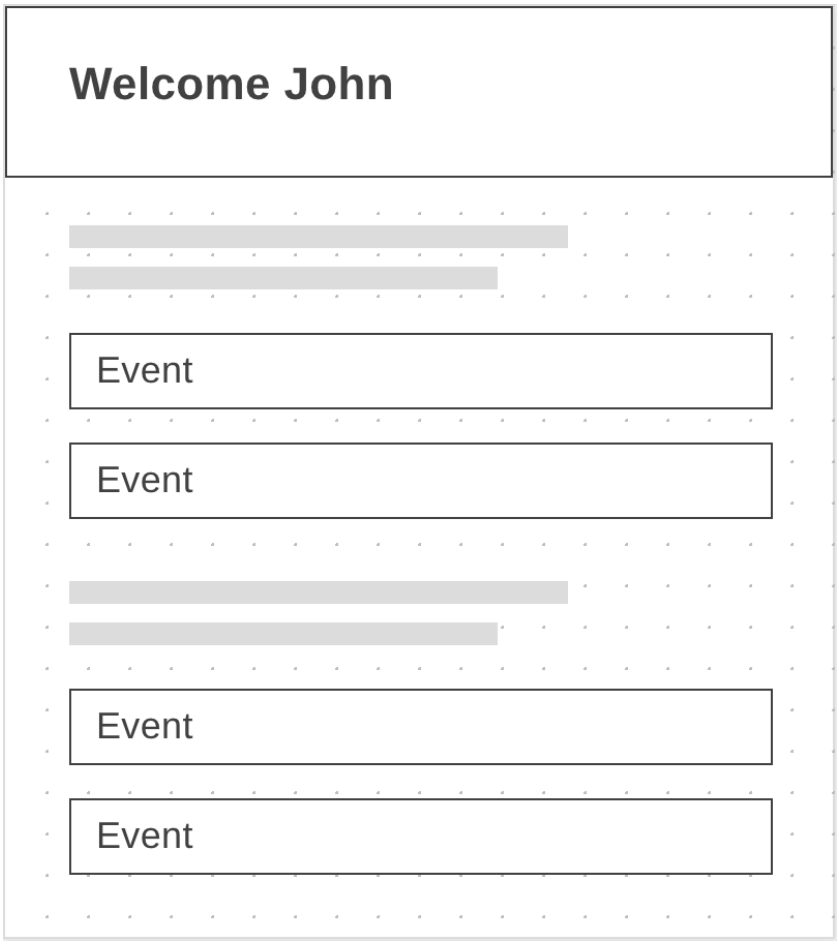
Eine Checkin Liste (CheckinList) stellt dabei eine Liste von Teilnehmern (Attendee) dar, die über diese Liste eingecheckt werden können. Welche Teilnehmer

tatsächlich auf dieser Liste stehen hängt davon ab, welche Tickets für die Checkin Liste gewählt wurden. Bei einer Checkin Liste hat man die Auswahl, welche Ticket Typen man einbeziehen will. Nur Teilnehmer, die eines der gewählten Tickets gekauft haben, können dann über diese Liste auch eingecheckt werden.

Sobald ein Teilnehmer eingecheckt wird erstellt sich ein Checkin Eintrag in der Datenbank, der über einen Fremdschlüssel mit dem Teilnehmer verbunden ist. Ebenfalls hat dieser Checkin Eintrag eine Beziehung zu der Checkin Liste, zu der dieser Eintrag gehört. Dadurch weiß man, ob ein Teilnehmer bereits eingecheckt wurde oder nicht.

4.2 Mockup des User Interface

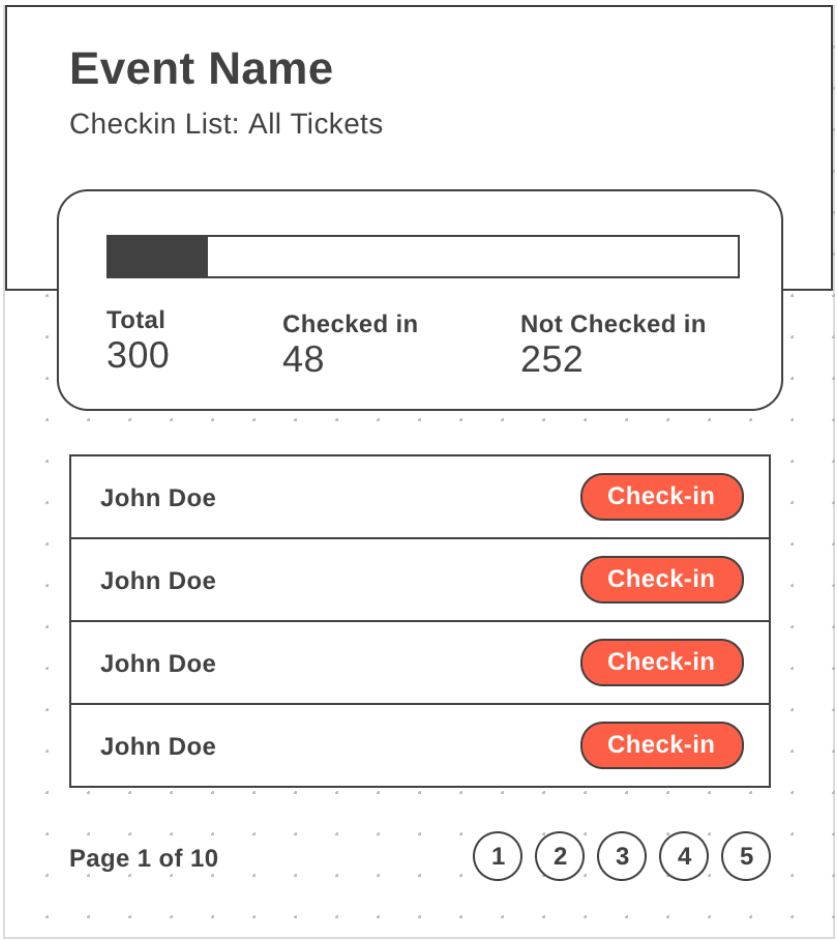
Abbildung 8: Mockup Dashboard



Das Dashboard zeigt alle erstellten Veranstalter des Users an sowie die jeweils dazugehörigen Events. Dabei ist jedes Event anklickbar, worauf sich eine Liste für das Event öffnen soll, die alle vorhandenen Checkin Listen anzeigt. Es ist auch möglich, dass für ein Event noch keine Checkin Liste erstellt wurde. Für diesen Fall soll ein sogenannter „Empty State“ erscheinen, der darüber informiert, dass noch keine

Checkin Listen vorhanden sind. Klickt man auf eine vorhandene Liste, öffnet sich die Detailansicht dieser Checkin Liste.

Abbildung 9: Mockup Detailseite einer Checkin Liste



Die Detailseite einer Checkin Liste baut sich in 3 Sektionen auf. In der obersten Sektion befinden sich Infos wie der Event Name sowie der Name der ausgewählten Checkliste. Darunter befindet sich eine Sektion, die Auskunft über die aktuellen Statistiken gibt. Hierzu gehören die gesamte Anzahl der Teilnehmer, die Anzahl der Teilnehmer, die bereits eingecheckt wurden sowie die Anzahl der Teilnehmer, die noch nicht eingecheckt wurden. Ein Fortschrittsbalken visualisiert dabei, wie viele Teilnehmer bereits eingecheckt wurden.

In der letzten Sektion befindet sich die Liste der Teilnehmer. Für jeden Teilnehmer wird der Name angezeigt sowie die Möglichkeit, diesen einzuchecken. Sobald der Button „Check-in“ geklickt wurde, wird der Teilnehmer eingecheckt und der Status entsprechend angepasst - sowohl im Frontend als auch in der Datenbank. Zusätzlich gibt es noch die Möglichkeit zwischen verschiedenen Seiten zu navigieren, falls mehr als 15 Teilnehmer für das Event registriert sind.

4.3 Einsatz von Technologien

Die Progressive Web App besteht prinzipiell aus zwei verschiedenen Teilen. Die Progressive Web App an sich, die hauptsächlich aus JavaScript besteht, sowie die API, über die Daten für die Progressive Web App über eine Schnittstelle bereitgestellt werden.

Dabei werden folgende Technologien eingesetzt:

Tabelle 2: Eingesetzte Technologien und deren Verwendung

Technologie	**Verwendung**
Javascript ES6 (ECMAScript 6 oder auch ECMAScript 2015)	Zur Umsetzung der Hauptkomponenten einer Progressive Web App wie Service Workers.
HTML5 (Hypertext Markup Language)	Zur Darstellung und Strukturierung von Inhalten, die der User sieht.
CSS3 (Cascading Style Sheets Level 3)	Mit Hilfe von CSS erfolgt die Gestaltung der Web App.
SASS/SCSS (Syntactically Awesome Stylesheets mit SCSS-Syntax (Sassy CSS))	Vereinfacht die Erstellung von CSS Dateien durch die Bereitstellung von Variablen, Funktionen etc. SCSS Dateien werden zu CSS Dateien kompiliert.
Python 3.7	Wird zur Umsetzung der API verwendet.

4.4 Einsatz von Frameworks

Für die Umsetzung des Frontends wird das JavaScript Framework Vue.js benutzt. Weiterhin wird das Framework Nuxt.js verwendet, das speziell für Vue.js entwickelt wurde.

Im Backend, dass für die Bereitstellung der Schnittstelle (API) verantwortlich ist, wird Django zusammen mit dem Django REST Framework verwendet.

5 Realisierung / Umsetzung

Die folgend beschriebene Realisierung kann unter anderem live getestet werden. Die API und die Progressive Web App befinden sich dabei auf zwei unterschiedlichen Servern. Bei den Servern handelt es sich um eine kostenfreie Variante, die sich nach 30 Minuten Inaktivität automatisch abschalten um Kosten zu sparen, weshalb der erste Aufruf der Seite etwas länger dauern kann.

Für den Live Test sind folgende Schritte notwendig:

1. API Server aktivieren durch aufrufen folgender URL:
   https://lumis-staging-pr-137.herokuapp.com/health/?token=qVjsGW
   Erscheint der Text „OK“ kann fortgefahren werden.

2. Nun kann die Progressive Web App unter folgender URL aufgerufen werden:
   https://lumis-cobe.herokuapp.com/admin/

   Email: sascha@lumis.ai

   Passwort: progressive7

5.1 Back-End

5.1.1 Implementierung des Datenbankmodells

Das zuvor erstelle ER-Diagram wird mit Hilfe von sogenannten Django Modellen realisiert. Ein Modell repräsentiert hierbei eine Tabelle in einer Datenbank. Dieses Modell besitzt alle Informationen, die benötigt werden, um die Tabelle in der Datenbank abzubilden. Dazu gehören die einzelnen Felder sowie Verhaltensweisen der Daten. [44]

Die Basis beider Modelle stellt ein sogenanntes abstraktes Modell dar. Ein abstraktes Modell erstellt keine Datenbanktabelle selbst, sondern muss mit einem Modell kombiniert werden. Dieses abstrakte Modell stellt Felder zur Verfügung, die dann automatisch in das Modell eingebracht werden. Das folgende abstrakte Modell mit dem Namen “TimeStampedModel” stellt die Felder „created“ und „updated“ zu Verfügung, die von fast jedem Modell benötigt werden. „created“ gibt dabei an, wann der Eintrag in der Datenbank erstellt wurde (dieser Wert ändert sich nach initialer

[44] (Django, 2019)

Eintragung nie mehr). „updated“ gibt an, wann das Modell zum letzten Mal verändert wurde.

```
class TimeStampedModel(models.Model):
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)

    class Meta:
        abstract = True
```

Das folgende “CheckinList” Modell repräsentiert eine Checkin Liste in der Datenbank. Dieses Modell hat zwei Felder: „event“ und „tickets“. Das „event“ Feld stellt eine Verknpüfung zum „Event“ Modell her, da es sich hierbei um eine Fremdschlüsselbeziehung handelt, denn jede Checkin Liste gehört exakt zu einem Event und ein Event kann beliebig viele Checkin Listen besitzen. Das „tickets“ Feld stellt eine Many-to-Many Beziehung mit dem „Ticket“ Modell dar. Eine Many-to-Many Beziehung sorgt automatisch dafür, dass in der Datenbank automatisch eine weitere Tabelle erstellt wird, die man jedoch nicht explizit benennen muss. Diese Tabelle besteht aus 3 Feldern: einem Primärschlüssel, einem Fremdschlüssel zu einem Ticket Eintrag und einem Fremdschlüssel zu einer Checkin Liste. Die Kombination von Ticket Eintrag und Checkin Liste muss dabei einzigartig sein. Es können also nicht dieselben Einträge mehrfach aufkommen. Durch diese Beziehungsart ist es möglich, dass eine Checkin Liste bestimmte Tickets inkludiert.

```
class CheckinList(TimeStampedModel):
    event = models.ForeignKey(
        Event, on_delete=models.CASCADE, related_name='checkin_lists'
    )
    tickets = models.ManyToManyField(Ticket, related_name='checkin_lists')
    name = models.CharField(max_length=50, verbose_name=_("Name"))

    def __repr__(self):
        return "{}: {}".format(self.__class__.__name__, self)

    def __str__(self):
        return self.name
```

Das „Checkin“ Modell repräsentiert einen Eintrag in einer Checkin Liste. Sobald ein Teilnehmer (Attendee) eingecheckt wird, erfolgt ein Eintrag in der Datenbanktabelle, den dieses Modell abbildet.

```
class Checkin(TimeStampedModel):
    attendee = models.ForeignKey(
        Attendee, related_name='checkins', on_delete=models.CASCADE
    )
    checkin_list = models.ForeignKey(
        CheckinList, on_delete=models.CASCADE, related_name='checkins'
    )

    def __repr__(self):
        return "{}: {}".format(self.__class__.__name__, self)

    def __str__(self):
        return self.checkin_list.name
```

5.1.2 Implementierung des Application Programming Interface (API)

Für die Implementierung der API wird das Django REST Framework verwendet. Das Framework bietet ein leistungsstarkes Toolkit an, wodurch die Erstellung von REST APIs erleichtert wird. Dieses Framework wird unter anderem auch von bekannten Unternehmen verwendet wie Eventbrite, Heroku oder Mozilla. [45]

Da man in Django mit sogenannten Apps arbeitet, wird für die API ebenfalls eine eigene App erstellt mit der folgenden Struktur:

```
api/
|__ pagination.py
|__ urls.py
|__ serializers/
|   |__________ accounts.py
|   |__________ checkin.py
|   |__________ events.py
|   |__________ organizers.py
|__ views/
    |____ accounts.py
    |____ checkin.py
    |____ events.py
    |____ organizers.py
```

Zur vereinfachten Darstellung werden im Folgenden nur kleinere Codeausschnitte dargestellt.

Die **urls.py** Datei ist dafür zuständig, die Endpunkte für die API festzulegen.

[45] Vgl. (Django REST framework, 2019)

```
event_router = DefaultRouter(trailing_slash=False)
event_router.register('checkin', checkin.CheckinListViewSet)

app_name = 'api'
raw_patterns = [
    path(
        'api/',
        include(
            [
                path(
                    'organizers/<slug:organizer>/events/<slug:event>/',
                    include(event_router.urls),
                ),
                # ...
            ]
        ),
    ),
]
```

In diesem Beispiel werden die Endpunkte für die Checkin Liste registriert. Die Basis dieses Endpunkts stellt die Event URL dar, da eine Checkin Liste immer zu einem Event gehört. Die Event URL wird aus dem dem Slug des Organizers und dem Slug des Events zusammengesetzt:

```
'organizers/<slug:organizer>/events/<slug:event>/'
```

An diesen Pfad werden alle Endpunkte der Checkin Liste angehängt mit dem zuvor definierten Ausgangspfad „checkin/“.

```
event_router.register('checkin', checkin.CheckinListViewSet)
```

Da ein Endpunkt oftmals die CRUD-Operationen unterstützt, werden hierfür einige weitere URLs benötigt. Durch die Verwendung von sogenannten ViewSets, die das Django REST Framework bereitstellt, werden diese Endpunkte automatisch erstellt, wie später in der **views.py** Sektion zu sehen ist. [46] [47]

Die Basis folgender Endpunkte stellt dieser Pfad dar:

```
'organizers/<slug:organizer>/events/<slug:event>/'
```

[46] Vgl. (Routers - Django REST framework, 2019)
[47] Vgl. (Viewsets - Django REST framework, 2019)

<int:pk> stellt in dieser Liste einen URL Parameter dar für einen Primarschlüssel (Primary Key = pk) von Datentyp Integer (int).

Tabelle 3: API Endpunkte

Endpunkt	**HTTP Methode**	**Beschreibung**
checkin	GET	Auflistung aller Checkin Listen für das Event.
checkin	POST	Erstellung einer neuen Checkin Liste.
checkin/<int:pk>	GET	Informationen einer einzelnen Checkin Liste aufrufen.
checkin/<int:pk>	PUT	Bearbeitung einer vorhanden Checkin Liste.
checkin/<int:pk>	DELETE	Löschen einer vorhanden Checkin Liste.

Der **serializers** Ordner enthält Dateien, die für die Serialisierung von einzelnen Modellinstanzen verantwortlich sind. Durch Serialisierung wird die Konvertierung komplexer Daten wie beispielsweise eine Modellinstanz oder auch ein sogenanntes Queryset (SQL Abfrage durch Django) in native Python-Datentypen ermöglicht, die dann in JSON gerendert werden können. [48]

Der folgende Code wird beispielsweise dazu verwendet, die Serialisierung der Checkin Liste durchzuführen:

```
class TicketRelatedField(serializers.PrimaryKeyRelatedField):
    def get_queryset(self):
        return self.context['event'].tickets.all()


class CheckinListSerializer(serializers.ModelSerializer):
    tickets = TicketRelatedField(
        queryset=Ticket.objects.none(),
        many=True,
    )

    class Meta:
        model = CheckinList
        fields = ('id', 'event', 'tickets', 'name')
```

[48] Vgl. (Serializers - Django REST framework, 2019)

Die Klasse `CheckinListSerializer` erbt dabei von der Klasse `ModelSerializer`. Diese Klasse wird durch das Django REST Framework bereitgestellt und muss bei jedem Serializer angegeben werden. Die innere Klasse `Meta` definiert das Modell, das serialisiert werden soll. Die `fields` Variable gibt an, welche Felder in die Serialisierung einbezogen werden sollen. Dadurch, dass die Datentypen für diese Felder bereits bei der Erstellung des Modells definiert wurden, weiß das Django REST Framework automatisch, welcher Serializer für das jeweilige Feld verwendet werden soll. Das Feld `id`, das einen Primärschlüssel repräsentiert, wird dabei vom Django REST Framework automatisch auf „read-only“ gestellt, da dieses Feld in keiner Operation geändert werden darf.

Für das Feld `tickets` wurde ein neuer Feldtyp erstellt durch die Klasse `TicketRelatedField`. Da es sich bei diesem Feld um eine Many-to-Many Beziehung handelt, erbt dieses Feld von der Serializer Klasse `PrimaryKeyRelatedField`. Der Grund für die Erstellung eines neuen Feldtyps ist, dass dieses Feld ohne Modifikation alle vorhandenen Tickets in der Datenbank auflisten würde. Da jedoch nicht jedes Ticket in der Datenbank zu dem gleichen Event gehört, muss diese Auswahl eingeschränkt werden. Diese Einschränkung erfolgt, indem die `get_queryset` Methode überschrieben wird, die einfach gesagt ein SQL Query in der Datenbank ausführt. Die `context[event]` Variable wird später in der View übergeben, da erst in der Request bekannt ist, um welches Event es sich handelt. Die ausgeführte Query listet dann alle Tickets für das Event auf, das in der Request übergeben wurde.

Der **views** Ordner enthält Dateien, die beispielsweise dafür verantwortlich sind, eine Modellinstanz dem Client als HTTP Response darzustellen. Hier werden die zuvor erstellen Serializer eingesetzt. [49]

[49] Vgl. (Views - Django REST framework, 2019)

Der folgende Code ist für die View der Checkin Liste verantwortlich:

```
class CheckinListViewSet(viewsets.ModelViewSet):
    serializer_class = checkin.CheckinListSerializer
    queryset = CheckinList.objects.none()

    def get_queryset(self):
        return self.request.event.checkin_lists.all()

    def get_serializer_context(self):
        context = super().get_serializer_context()
        context['event'] = self.request.event
        return context

    def perform_create(self, serializer):
        serializer.save(event=self.request.event)

    @action(detail=True)
    def stats(self, request, **kwargs):
        # ...

    @action(detail=True)
    def attendees(self, request, **kwargs):
        # ...
```

Die `CheckinListViewSet` Klasse erbt von der `ModelViewSet` Klasse, die aus dem Django REST Framework stammt. Zusammen mit der definierten Serializer Klasse `CheckinListSerializer`, die im Schritt zuvor erstellt wurde, können so sämtliche CRUD Operationen automatisch erstellt werden.

Für die Auflistung aller Checkin Listen wird die Methode `get_queryset` vewendet. Diese wird überschrieben, da sonst alle Einträge in der Datenbank zurückgegeben werden und nicht beachtet wird, ob die Checkin Liste auch wirklich zu dem Event gehört, das in der Request übergeben wurde.

Die `get_serializer_context` Methode übergibt Context „Variablen" an den Serializer, die im Schritt zuvor erwähnt wurden. Die `perform_create` Methode ist dafür verantwortlich, eine neue Checkin Liste zu erstellen. In dieser Methode wird das Event explizit gesetzt durch `event=self.request.event`, da dieses Feld vom Serializer benötigt wird, jedoch nicht in den Daten der eingehenden Request vorhanden ist. [50]

[50] Vgl. (Generic views - Django REST framework, 2019)

Die `CheckinListViewSet` Klasse hat zudem zwei weitere Methoden, die mit dem `@action` Decorator markiert wurden. Dadurch ist es möglich, zusätzliche Aktionen für das Routing zu definieren. Mit dem Parameter `detail` wird festgelegt, ob die Aktion auf ein einzelnes Objekt oder eine Sammlung von Objekten angewendet werden kann. [51]

```
class CheckinListViewSet(viewsets.ModelViewSet):

    # ...

    @action(detail=True)
    def stats(self, request, **kwargs):
        checkin_list = self.get_object()

        total_attendees = self.request.event.attendees.active().fil-
ter(
            ticket__in=checkin_list.tickets.all()
        ).count()
        checked_in = Checkin.objects.filter(
            checkin_list=checkin_list
        ).count()
        response = {
            'total_attendees': total_attendees,
            'checked_in': checked_in,
        }
        return Response(response)
```

In diesem Beispiel wurde die Methode `stats` als Aktion definiert, die für ein einzelnes Objekt verfügbar ist. Möchte man diese Aktion verwenden, kann eine Anfrage an den folgenden Endpunkt gestellt werden:

```
'organizers/<slug:organizer>/events/<slug:event>/checkin/<int:pk>/stats'
```

5.2 Front-End

5.2.1 Authentifizierung

Die Authentifizierung innerhalb der Single Page Application erfolgt über einen sogenannten JSON Web Token (JWT). Dabei handelt es sich um einen Access-Token, das Informationen sicher im JSON Format zwischen zwei Parteien übertragen kann. [52] [53]

[51] Vgl. (Marking extra actions for routing - Django REST framework, 2019)
[52] Vgl. (JSON Web Tokens, 2019) (Kastner, 2019)
[53] Vgl. (Kastner, 2019)

Folgend ein beispielhafter JSON Web Token:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFt-
ZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWUsImp0aSI6Ijg5NDVi-
ZjFhLTA2YjUtNGE0ZC04NWE2LTBiNDM5NjExMzYyMiIsImlhdCI6MTU2NTg1NzEx-
NywiZXhwIjoxNTY1ODYwNzE3fQ.yLNjAMg_HpCx1Fq_-eGDJMpeoikY4O5NdDC-F54Xy2g
```

Der Aufbau des Tokens ist in drei Abschnitte gegliedert, die durch einen Punkt getrennt werden.

`header.payload.signature`

Wird der Token dekodiert, erhält man folgende Infos:

Header

```
{
 "typ": "JWT",
 "alg": "HS256"
}
```

Payload

```
{
 "sub": "1234567890",
 "name": "John Doe",
 "admin": true,
 "jti": "8945bf1a-06b5-4a4d-85a6-0b4396113622",
 "iat": 1565857117,
 "exp": 1565860717
}
```

Signatur mit HS256 (Verwendeter Secret: „secret“)

```
yLNjAMg_HpCx1Fq_-eGDJMpeoikY4O5NdDC-F54Xy2g
```

JWT codiert und signiert die Informationen lediglich, verschlüsselt diese jedoch nicht. JWT ist nicht dafür gedacht, Daten zu verstecken, deshalb sollten keine sensiblen Informationen in einem Token vorhanden sein. Der Sinn von JWT ist die Verifizierung, dass es sich um einen gültigen Token handelt und dieser nicht modifiziert wurde. Deshalb wird jeder Token mit einem sogenannten Secret erstellt. Dieser Secret ist privat und ist dem Client nicht bekannt. Der Secret wird dazu verwendet, die Authentizität der Datenquelle zu überprüfen. Einem Angreifer ist es also nicht

möglich, einen validen JSON Web Token zu erstellen, da dies nur möglich ist, wenn der Secret bekannt ist. [54] [55]

Da die Entwicklung der Progressive Web App auf Basis einer Single Page Application fundiert, wird für die Authentifizierung eines Benutzers eine REST API Schnittstelle verwendet. Da eine REST API zustandslos ist, müssen in jeder Request alle Informationen vorhanden sein, um den User zu authentifizieren. Diese Informationen werden mit Hilfe des JSON Web Tokens übertragen. [56]

5.2.2 Umwandlung zu einer Progressive Web App

In diesem Abschnitt wird auf die konkrete Umsetzung der Progressive Web App eingegangen.

5.2.2.1 Installation

Für die Installierbarkeit der Progressive Web App wird folgende Manifest Datei verwendet:

```
{
  "short_name": "Checkin",
  "name": "Lumis Checkin",
  "icons": [
    {
      "src": "/icons/icon-192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
    {
      "src": "/icons/icon-512.png",
      "type": "image/png",
      "sizes": "512x512"
    }
  ],
  "start_url": "/admin",
  "display": "fullscreen",
  "orientation": "portrait",
  "theme_color": "#335EEA",
  "background_color": "#335EEA"
}
```

Bei Smartphones von Apple mit dem Betriebssystem iOS ist zu beachten, dass zwar Manifest Dateien unterstützt werden, wodurch die Installierbarkeit möglich ist, jedoch werden die festgelegten Icons in der Manifest Datei ignoriert. Um ein Icon für

[54] Vgl. (Stecky-Efantis, 2016)
[55] Vgl. (Hiwarale, 2018)
[56] Vgl. (Sachchithananthan, 2019)

iOS Geräte festzulegen, muss der folgende HTML Tag im Header gesetzt werden.[57]

```
<link rel="apple-touch-icon" size="180x180" href="/icons/apple-touch-icon.png">
```

Nach Erstellung und Verlinkung der Manifest Datei durch den entsprechenden HTML Tag ist es nun möglich, die App auf dem Homescreen zu installieren.

Abbildung 10: Installierung der PWA

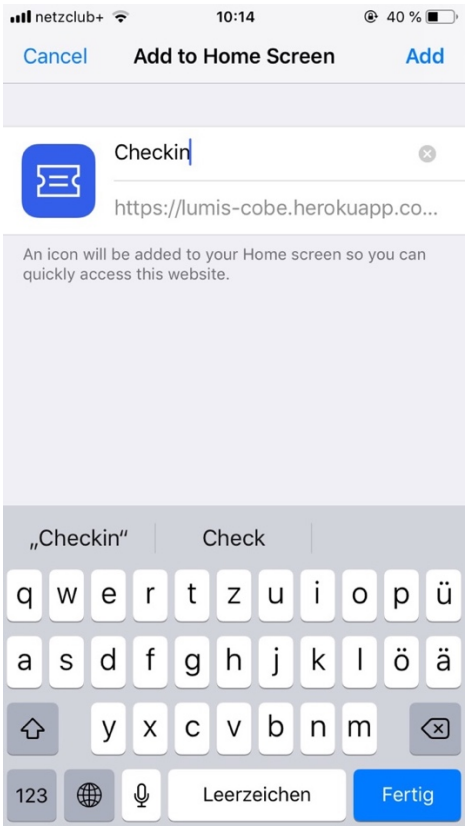
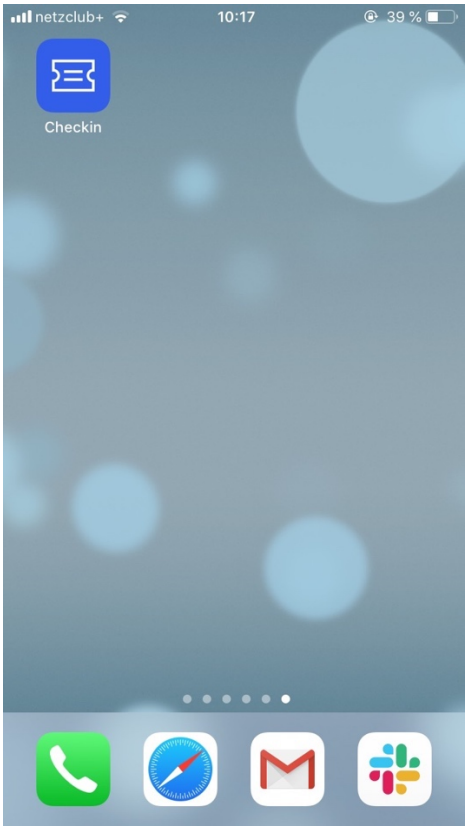


Abbildung 11: Installierte PWA auf dem Homescreen



Die Installierbarkeit beschränkt sich dabei nicht nur auf das Smartphone. Selbst über den Browser kann die Progressive Web App installiert werden und verhält sich dann wie ein normales Programm, das man am Computer öffnet. In Google Chrome gibt es hierfür zum Beispiel eine kleines **+** Symbol in der URL Leiste, was ein Indikator dafür ist, dass es sich um eine installierbare Web App handelt.

[57] Vgl. (Apple, 2016)

Abbildung 12: Installation der Progressive Web App über den Browser

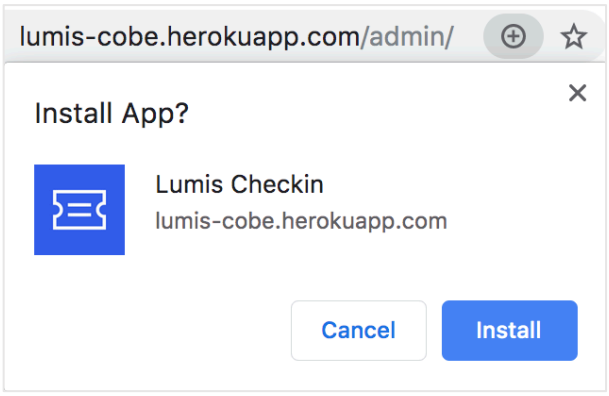
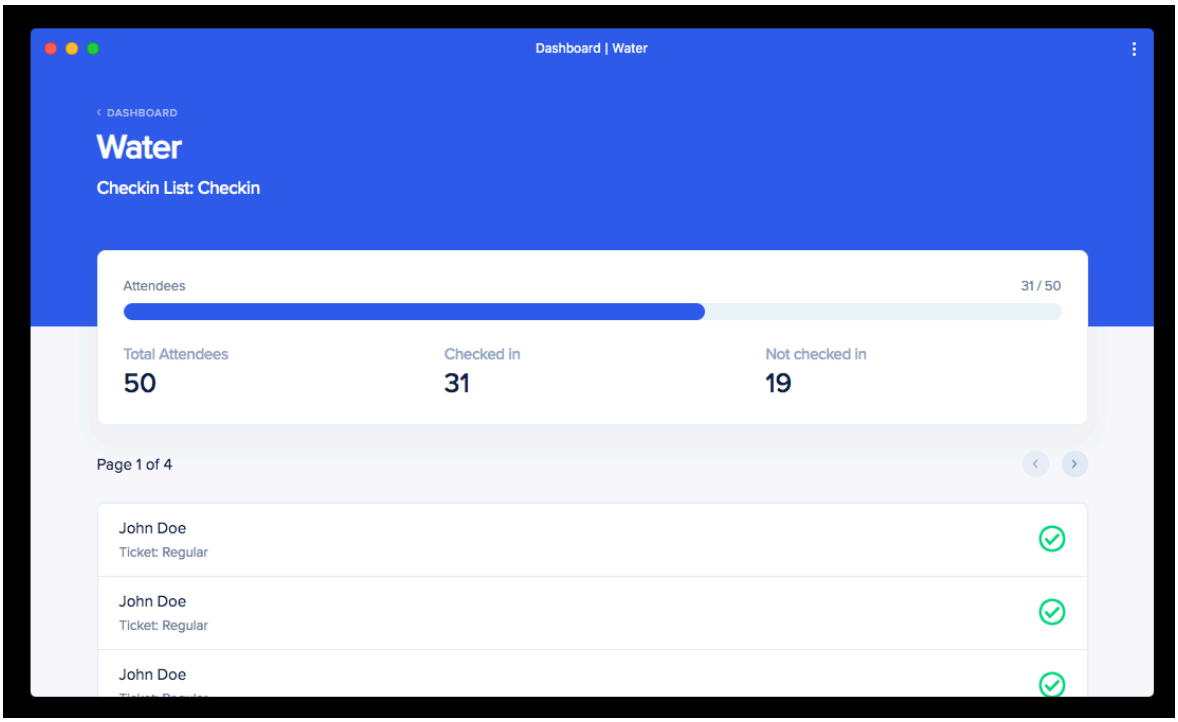


Abbildung 13: Installierte Progressive Web App auf dem Computer



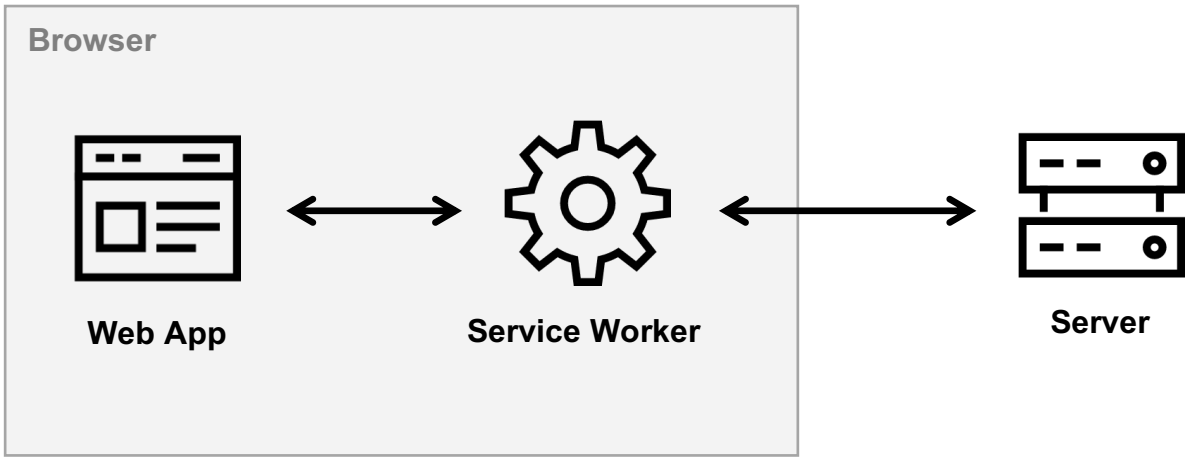
5.2.2.2 Offline Funktionalität

Einer der wichtigsten Funktionalitäten von Progressive Web Apps ist die Möglichkeit, die App auch offline verwenden zu können. Diese Netzwerkunabhängigkeit verschafft auch das Gefühl, eine native App zu verwenden.

Für die Umsetzung wird ein sogenannter Service Worker verwendet, der eine Kerntechnologie von Progressive Web Apps darstellt. Der Service Worker befindet sich dabei zwischen Browser und Netzwerk und fungiert als eine Art Proxy-Server. Er wird von der Progressive Web App im Browser registriert und ist nach der Registrierung nicht mehr abhängig von der Verfügbarkeit der Seite. Das bedeutet, dass der Service Worker unabhängig von der Webseite ausgeführt werden kann. Somit können zum Beispiel Push Benachrichtigungen empfangen werden, auch wenn die

Website gar nicht aufgerufen wird oder der Browser geöffnet ist. Da der Service Worker in einem separaten Thread läuft, hat dieser kein Zugriff auf das Document Object Model (DOM). [58] [59] [60]

Abbildung 14: Service Worker als Proxy

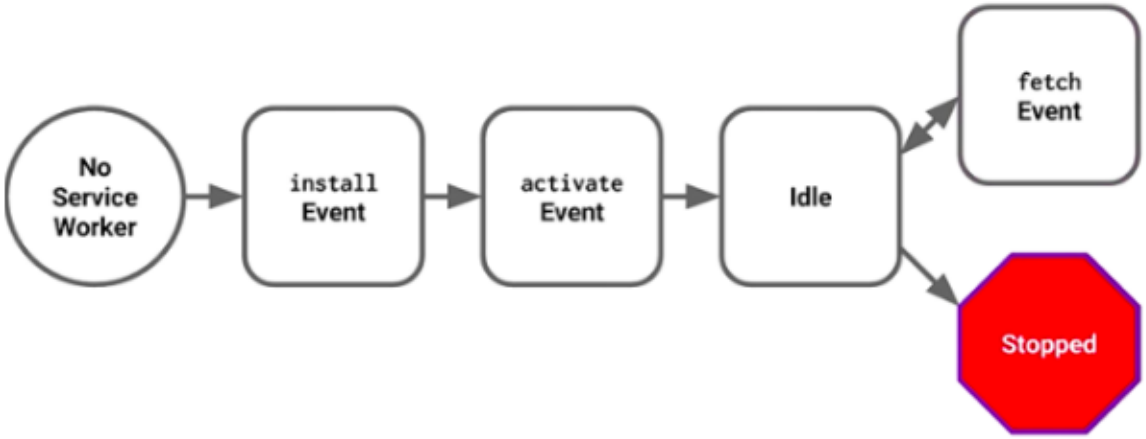


In der Darstellung ist zu sehen, dass der Service Worker zwischen der Web App und dem Server „sitzt“ und im Browser registriert wird. Dadurch ist es dem Service Worker möglich, die Details einer HTTPS-Anfrage auszulesen und sogar zu verändern. Durch die Funktion als Proxy ist es auch möglich, die Offline Funktionalität bereitzustellen. Der Service Worker bestimmt darüber, ob eine Anfrage über das Netz ausgeführt wird oder ob dieser die Anfrage selbst zusammensetzt. Bei letzterem ist der Einsatz der Cache API sehr hilfreich, da dadurch ermöglicht wird, Anfragen und Antworten im lokalen Speicher zu hinterlegen. [61] [62]

Ein Service Worker ist im Prinzip eine einzelne JavaScript Datei, die der Browser unabhängig von der Web App im Hintergrund ausführt. Bei der Verwendung eines Service Workers durchläuft dieser mehrere Phasen: [63]

[58] Vgl. (Service Worker API - MDN Web Docs, 2019)
[59] Vgl. (Liebel, Progressive Web Apps - Das Praxisbuch, 2018, S. 171)
[60] Vgl. (Love, Progressive Web Application Development by Example, 2018, S. Service Workers)
[61] Vgl. (Liebel, Progressive Web Apps - Das Praxisbuch, 2018, S. 177)
[62] Vgl. (Keklik, 2018)
[63] Vgl. (Gaunt, Service Workers: an Introduction - Google Developers, 2019)

Abbildung 15: Lebenszyklus eines Service Workers



Quelle: (Keklik, 2018)

Der Lebenszyklus eines Service Workers, der oberhalb abgebildet ist, besteht aus den Schritten Registrierung, Installation und Aktivierung, um vollständig zu funktionieren. Zusätzlich umfasst auch er die Aktualisierung sowie die Abmeldung eines Service Workers.

Für die **Registrierung** einen Service Workers wird folgender Code verwendet und in die Web Application eingefügt.

```
if ('serviceWorker' in navigator) {
  window.addEventListener('load', function() {
    navigator.serviceWorker.register('/sw.js').then(
      function(registration) {
        // Registration was successful
      }
    );
  });
}
```

Die if-Abfrage, ob das `navigator` Objekt das Attribut `serviceworker` aufweist, dient zur Sicherstellung, ob der Browser Service Workers unterstützt. Die `register` Funktion wird ausgeführt, sobald die Seite geladen ist und setzt den Pfad der Datei fest, die den Service Worker repräsentiert (in diesem Fall „sw.js“). Dabei ist es egal wie oft dieser Code ausgeführt wird. Der Browser führt die Registrierung nur durch,

sofern der Service Worker noch nicht zuvor registriert wurde oder eine aktualisierte Version des Service Workers vorliegt. [64] [65]

Der eigentliche Service Worker, der sich in diesem Beispiel in der JavaScript Datei „sw.js“ befindet, ist folgendermaßen aufgebaut:

```
self.addEventListener('install', function(event) {
 // ...
});

self.addEventListener('activate', function(event) {
 // ...
});

self.addEventListener('fetch', function(event) {
 // ...
});
```

Diese Grundstruktur ist ein Abbild des Lebenszyklus (ohne Registrierung, da diese bereits erfolgt ist). Die Event Listener in dem Service Workers „warten“ darauf, bis das entsprechende Event eintritt und führen dann den Code aus, der innerhalb des Listeners platziert wurde.

Das Event `install`, das die **Installation** des Service Workers repräsentiert, ist für die Vorbereitung sehr gut geeignet. Hier kann ein Cache initialisiert und statische Dateien zwischengespeichert werden. Die **Installation** erfolgt jedoch nur, wenn der Service Worker neu ist oder eine veraltete Version installiert ist.

Nach erfolgreicher **Registrierung** und **Installation** erfolgt die **Aktivierung** durch das Event `activate`. Ab diesem Schritt ist es dem Service Worker möglich, mit erneuten Seitenaufrufen zu arbeiten. Der Service Worker ist daher nur hilfreich, wenn die Seite erneut geladen wird. Beim ersten Aufruf erfolgt lediglich die **Registrierung**. Im Schritt der **Aktivierung** ist es deshalb sinnvoll, beispielsweise alte Dateien aus dem Cache zu löschen, die nicht mehr benötigt werden.

Das Event `fetch` wird ausgelöst, sobald eine Ressource im Netzwerk angefordert wird. Hier ist es möglich erst im Cache „nachzuschauen“, ob diese Anfrage bereits vorhanden ist. Sofern dies der Fall ist setzt der Service Worker die Anfrage selbst

[64] Vgl. (Gaunt, Service Workers: an Introduction - Google Developers, 2019)
[65] Vgl. (Copes, 2018)

zusammen und führt keine weitere Anfrage aus. Eine Ressource, die angefragt werden kann, ist dabei zum Beispiel eine CSS-Datei, ein Bild, eine API Antwort etc. Durch diesen Mechanismus wird die Offline Funktionalität gewährleistet. [66] [67] [68]

Die oben gezeigten Codebeispiele zeigen die direkte Implementierung von Service Workern. Jedoch gibt es eine Alternative, die von Google bereitgestellt wird, namens Workbox. Dies ist ein Tool aus einer Reihe von Bibliotheken, die das Arbeiten mit Service Workern erleichtert. Vor allem konzentriert sich das Tool auf den Caching Aspekt von Progressive Web Apps, um die Offline Funktionalität und eine schnelle Performance zu gewährleisten. [69] [70]

Durch ein Command Line Interface, das Workbox anbietet, wird ein sogenannter Wizard bereitgestellt, der durch die Erstellung einer Konfigurationsdatei führt, die später dazu dient, die Service Worker Datei zu generieren. Der interaktive Wizard stellt dem User dabei verschiedene Fragen wie „Welche Dateitypen möchte man speichern?“. Die Konfigurationsdatei, die dabei erstellt wird, kann folgendermaßen aussehen:

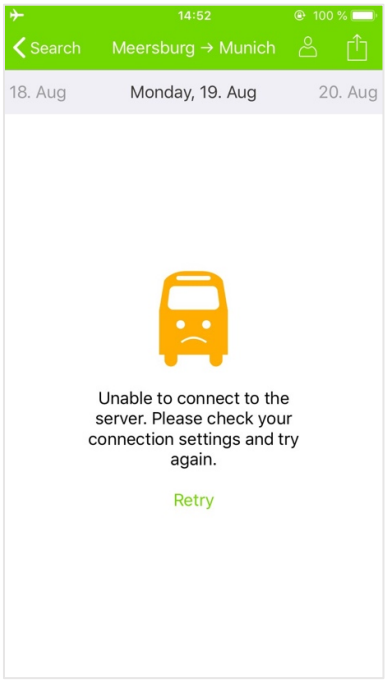
```
module.exports = {
  "globDirectory": "src/",
  "globPatterns": [
    "**/*.{css,html,js,txt}"
  ],
  "swDest": "src/sw.js"
};
```

Diese Datei ist die Grundlage für die Erstellung des Service Workers. Das Workbox Command Line Interface Tool verfügt über ein Kommando namens `generateSW`, das dafür zuständig ist, die Service Worker Datei anhand der Konfigurationsdatei zu erstellen. Die Registrierung des generierten Service Workers erfolgt nach wie vor manuell, wie anfänglich in diesem Kapitel dargestellt. Nun kann die Web App bereits offline verwendet werden. [71] [72]

[66] Vgl. (Irish, 2016)
[67] Vgl. (Copes, 2018)
[68] Vgl. (Ater, 2017, S. 45ff)
[69] Vgl. (Workbox - Google Developers, 2019)
[70] Vgl. (Liebel, Progressive Web Apps - Das Praxisbuch, 2018, S. 261)
[71] Vgl. (Liebel, Progressive Web Apps - Das Praxisbuch, 2018, S. 262)
[72] Vgl. (Sopheaktra, 2019)

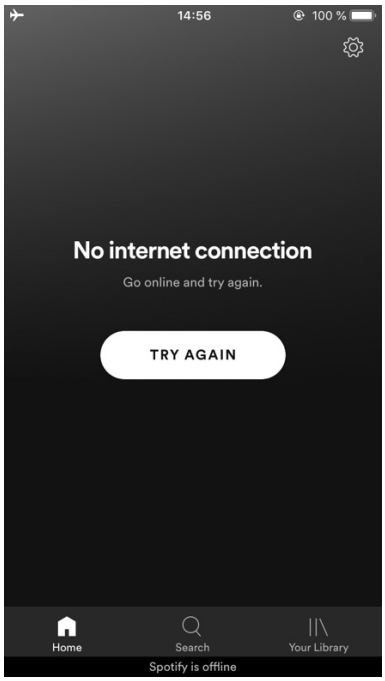
Sobald das Smartphone keine Internetverbindung hat werden nicht mehr alle Funktionalitäten bereitgestellt, die auch im Online Modus verfügbar sind. Dies ist nicht auf Progressive Web Apps beschränkt, sondern auch viele native Apps benutzen diesen Umgang mit dem Offline Modus. Der Grund hierfür ist schnell ersichtlich, da für viele Dinge eine Internetverbindung zwingend notwendig ist. Der große Vorteil der Offline Funktionalität ist jedoch, dass man nach wie vor die App öffnen kann und grundlegende Dinge nach wie vor funktionieren und der User nicht einfach eine Fehlermeldung bekommt, wie es bei normalen Webseiten der Fall wäre.

Abbildung 16: Flixbus App im Offline Modus



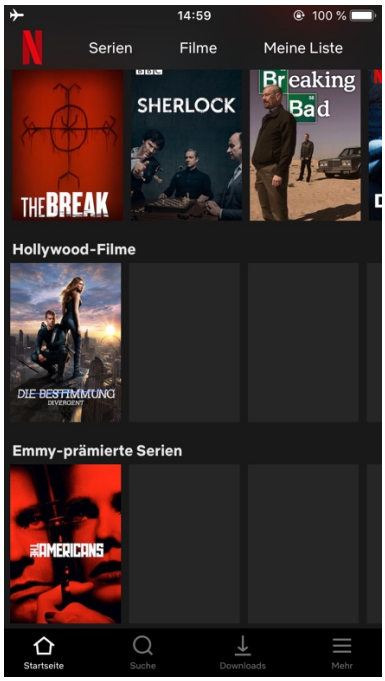
Quelle: (Flixbus, 2019)

Abbildung 17: Spotify App im Offline Modus



Quelle: (Spotify, 2019)

Abbildung 18: Netflix App im Offline Modus



Quelle: (Netflix, 2019)

Die obigen Beispiele zeigen, wie populäre native Apps mit einer fehlenden Internetverbindung umgehen. Die Flixbus App (links) bietet beispielsweise nach wie vor die Funktionalität an, gekaufte Tickets einzusehen, jedoch wird nach einer Online Verbindung verlangt, sobald versucht wird nach einer Busverbindung zu suchen.

Spotify (mitte) bietet nahezu keine Funktionalität im Offline Modus an, sofern man keine gekaufte Version hat, in der es möglich ist Offline Musik zu hören.

Die Netflix App (rechts) zeigt vereinzelt das Cover einiger Filme an, verlangt jedoch ebenfalls nach einer Online Verbindung nach Aufrufen eines Filmes, sofern dieser nicht heruntergeladen wurde.

Diese Beispiele zeigen, dass viele native Apps ohne Internetverbindung zwar aufrufbar sind, jedoch viele Funktionalitäten nach einer Online Verbindung verlangen. Dieses Verhalten kann ebenfalls auf Progressive Web Apps projiziert werden. Der Offlinemodus der App muss mindestens das reibungslöse Öffnen der App bereitstellen sowie grundlegende Funktionalitäten, die im Offline Modus Sinn für den User machen.

Abbildung 19: Dashboard im Online Modus

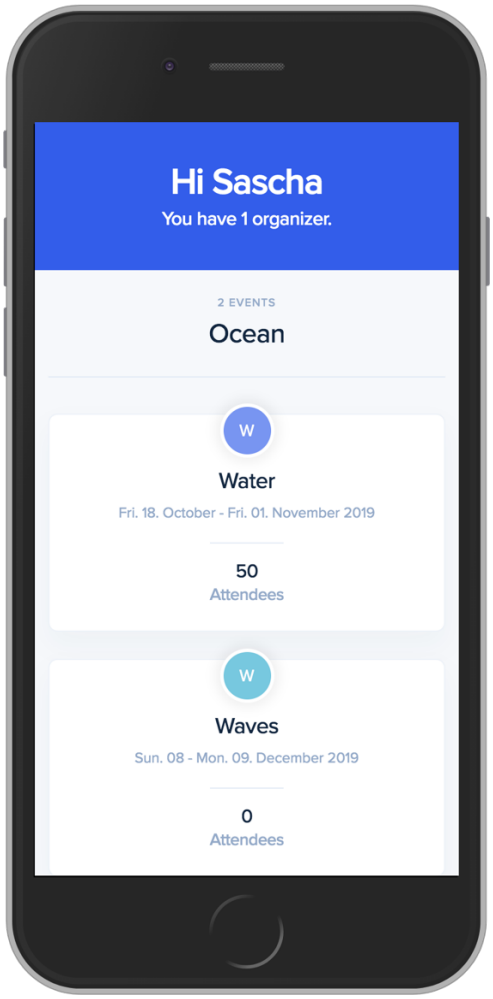
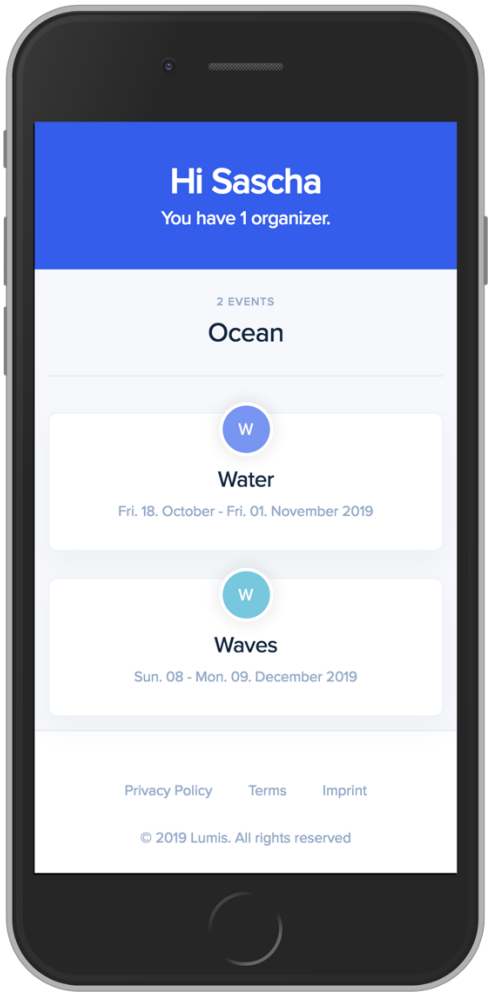


Abbildung 20: Dashboard im Offline Modus



In den Screenshots ist das Dashboard in der Online Variante (links) und der Offline Variante (rechts) zu sehen. Durch die Verwendung von Caching ist es möglich nahezu exakt den gleichen Inhalt in der Offline Variante darzustellen. Die Anzahl der Teilnehmer wurde im Offline bewusst ausgelassen, da sich diese Anzahl schnell

verändern kann und keine falschen Informationen angezeigt werden sollen. Sobald wieder eine Verbindung mit dem Internet hergestellt wird, erscheint diese Anzeige wieder.

Abbildung 16: Checkin Seite im Online Modus

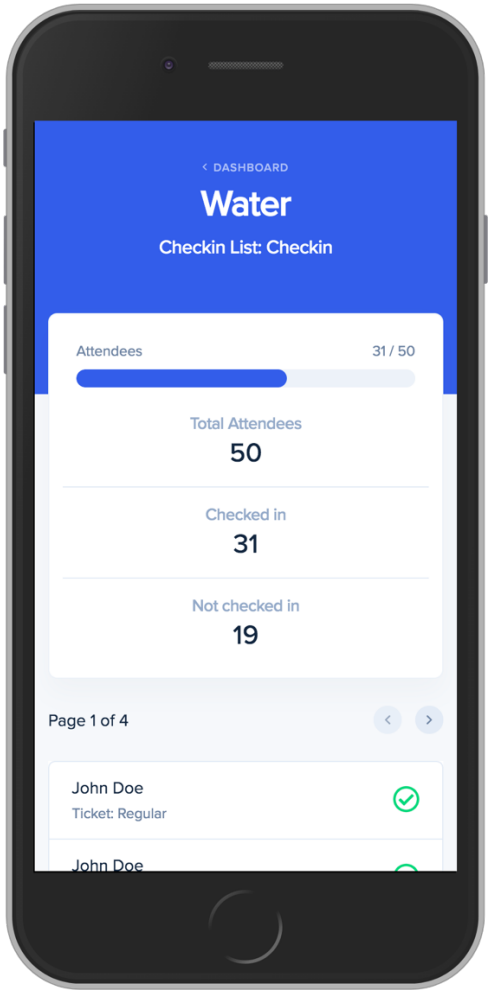
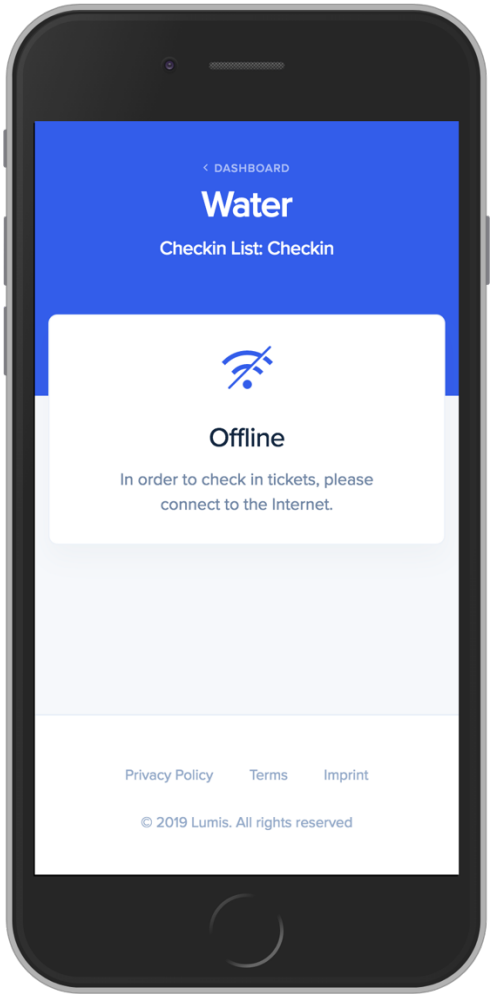


Abbildung 17: Checkin Seite im Offline Modus



Eine Checkin Liste ist nur im Online Modus voll funktionsfähig (links), jedoch wird dies dem User deutlich gemacht durch eine Meldung, die angezeigt wird, sobald sich das Gerät im Offline Modus befindet (rechts).

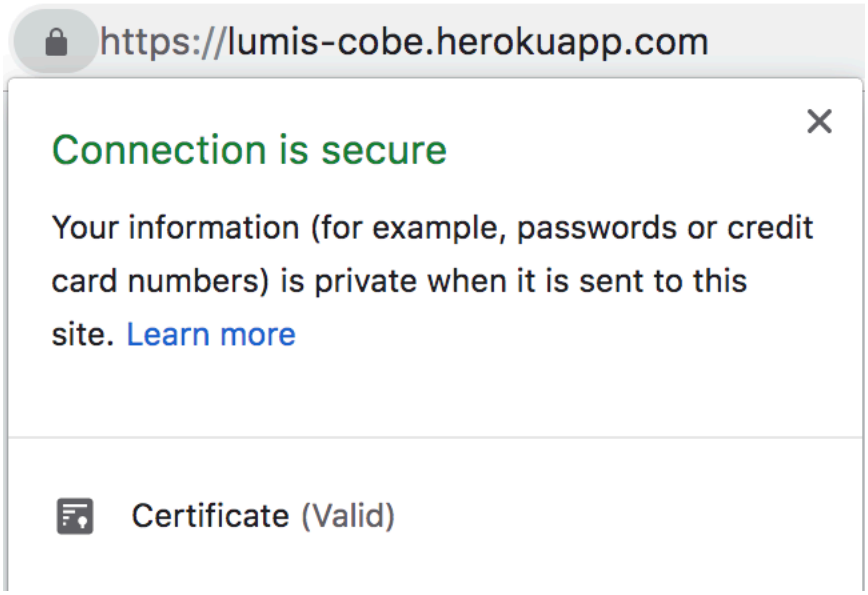
5.2.2.3 HTTPS Unterstützung

Für das Hosting der Anwendung wird die Plattform Heroku verwendet. Heroku ist eine containerbasierte Cloud Plattform, die es ermöglicht, moderne Webanwendungen bereitzustellen sowie zu verwalten. [73]

[73] Vgl. (What is Heroku?, 2019)

Durch die Verwendung dieser Plattform wird der Applikation automatisch ein gültiges SSL Zertifikat ausgestellt. Für die Zertifikatausstellung wird Let's Encrypt verwendet, wodurch es möglich ist, diese Zertifikate kostenfrei bereitzustellen. [74]

Abbildung 21: SSL Zertifikat Bestätigung im Browser



Quelle: (Google Chrome Browser, Version 75)

5.2.2.4 Verwendung von Push Benachrichtigungen

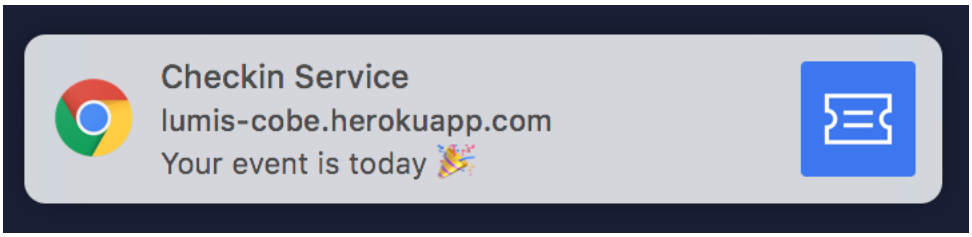
Push Benachrichtigungen sind ein wichtiger Teil einer App, da dadurch die Interaktion zwischen User und App gesteigert werden kann. Der große Vorteil von Push Benachrichtigungen ist, dass die entsprechende App dafür nicht geöffnet sein muss. Dies funktioniert sowohl in der mobilen App als auch am Computer. Beispielsweise kann am Tag der Veranstaltung eine Benachrichtigung an den User gesendet werden, um ihm mittzuteilen, dass diese heute stattfindet. Dadurch wird er auf die Checkin App nochmals aufmerksam gemacht. [75]

Google zeigt anhand des Beispiels Beyond the Rack (Online-Einkaufsunternehmen) wie wichtig Push Benachrichtigungen sind. Durch den Einsatz dieser verlängerte sich die Zeit, die ein User in der App verbrachte um 72%, sofern dieser über eine Push Benachrichtigung die App geöffnet hat. Weiterhin gab es einen Anstieg von 50% der wiederkehrenden Besuche. [76]

[74] Vgl. (Automated Certificate Management, 2019)
[75] Vgl. (Love, Progressive Web Application Development by Example, 2018, S. Using push notifications)
[76] Vgl. (Beyond the Rack - Google Developers, 2015)

Die Implementierung von Push Benachrichtigungen erfordert einen serverbasierten Dienst wie zum Beispiel den Cloud-basierten Dienst Google Cloud Messenger. Für die Umsetzung in dieser Arbeit wird der Drittanbieter OneSignal verwendet. OneSignal ist Marktführer im Bereich Kundenbindung bezogen auf Push Benachrichtigungen. Sowohl auf mobilen Plattformen als auch im Browser werden Push Benachrichtigungen unterstützt. Durch OneSignal können Benachrichtigungen entweder manuell verschickt werden oder über eine API. [77]

Abbildung 22: Push-Benachrichtigung



6 Bewertung

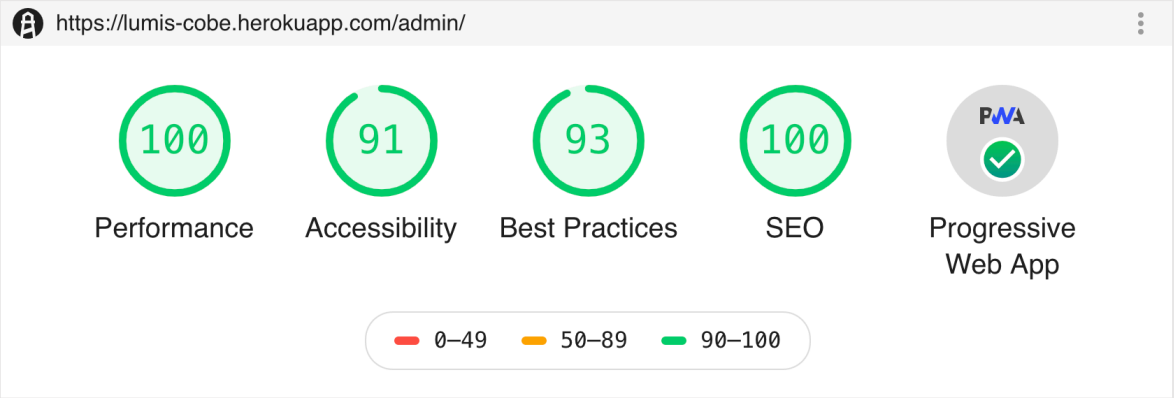
6.1 Qualitätsprüfung

Zur Qualitätsprüfung wird das Tool „Lighthouse“ verwendet. Dabei handelt es um ein von Google entwickeltes Tool zur Überprüfung verschiedenster Kriterien. Hierzu gehören Performance, Accessiblility, Best Practices, SEO und Progressive Web App. Dieses Tool ist bereits in Google Chrome integriert und kann in den Entwicklertools unter dem Reiter “Audits” aufgerufen werden. Das Tool ist nicht ausschließlich auf Progressive Web Apps begrenzt. Jede Webseite kann damit getestet werden, jedoch kann man auch explizit Progressive Web Apps testen auf ihre Performance und Nutzerfreundlichkeit. [78]

Der Lighthouse Report der entwickelten Progressive Web App ist unten detailiert dargestellt. Der Report kann unter anderem auch unter der folgenden URL abgerufen werden: https://bit.ly/2KzmQwY

[77] Vgl. (OneSignal, 2019)
[78] Vgl. (Lighthouse, 2019)

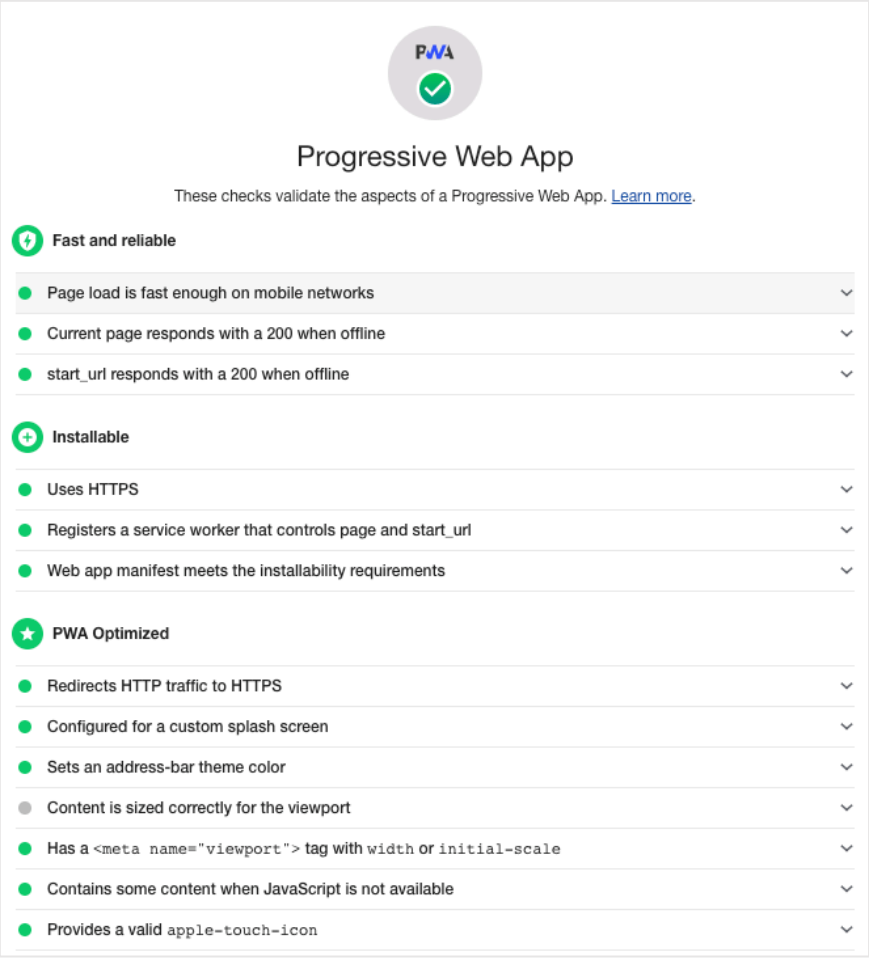
Abbildung 23: Google Lighthouse Report Ergebnisse



Quelle: (Google Lighthouse Report, 2019)

Das Ergebnis zeigt in allen 4 Kategorien nahezu die volle Punktzahl. Bei dem Progressive Web App Icon ist ein grüner Haken zu sehen, was bestätigt, dass die Kriterien einer Progressive Web App erfüllt werden. Folgend ist der detaillierte Bericht der Progressive Web App Tests zu sehen, die erfolgreich bestanden wurden.

Abbildung 24: Google Lighthouse Report (Progressive Web App)



Quelle: (Google Lighthouse Report, 2019)

6.2 Kriterienkatalog

Zur Bewertung, ob Progressive Web Apps eine Alternative zu nativen Apps darstellen, werden verschiedene messbare Kriterien festgelegt, anhand deren die Einschätzung darüber erfolgen kann. Da in dieser Bachelorarbeit eine Progressive Web App entwickelt wurde, jedoch keine native App, werden zur Bewertung vermehrt bereits bestehende Fallstudien einbezogen, die einen zuverlässigen Vergleich von nativen Apps und Progressive Web Apps gewährleisten.

Tabelle 4: Kriterienkatalog

Kriterium	**Beschreibung**
Entwicklungsaufwand	Wie gestaltet sich die Entwicklung der App und der damit verbundene Aufwand?
Nutzung und Conversion Rate	Wie kann die App benutzt werden und wie verhält sich die Conversion Rate?
Funktionsumfang	Welche Funktionalität kann die App anbieten?
Sicherheit	Wie sicher ist die App und wie sicher ist das Übertragen von benutzerspezifischen Daten?
Geschwindigkeit	Performance der App bezogen auf die Ladezeiten

6.2.1 Entwicklungsaufwand

Um eine möglichst große Nutzergruppe zu erreichen ist es wichtig, dass eine App auf verschiedensten Plattformen funktioniert. Bei einer nativen App ist der Entwicklungsaufwand verhältnismäßig deutlich größer, da für jede Plattform eine individuelle Version erstellt werden muss. So muss zum Beispiel eine Android App in der Sprache Java entwickelt werden und eine App für iOS in Objective-C/Swift. Nach der Entwicklung der App stellt die Veröffentlichung der App ebenfalls noch eine Hürde dar. Die App muss im App Store registriert werden und zusätzlich noch akzeptiert werden. Dies kann im Google Play Store mehrere Stunden dauern, im App Store von Apple sogar mehrere Tage. [79] [80]

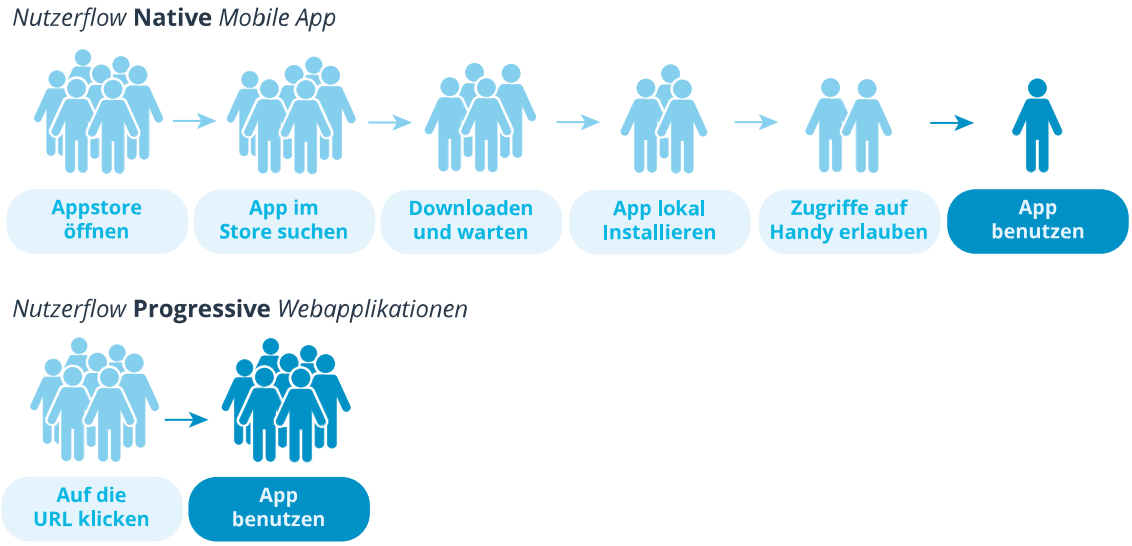
[79] Vgl. (Chekalin & Gritsay , 2018)
[80] Vgl. (Hedemann, 2018)

Bei Progressive Web Apps hingegen ist die einzige Plattform der Browser selbst. Dadurch muss nur eine einzige Version entwickelt werden, die sowohl am Computer als auch dem Smartphone verwendet werden kann. Da eine Progressive Web App keinen App Store benötigt, fällt dieser Schritt weg, was sowohl Kosten und Zeit spart.

Ein weiterer Aspekt bei der Entwicklung von Apps stellen Updates dar. Bei nativen Apps muss der User selbst aktiv werden und Updates manuell installieren. Bei Progressive Web Apps hingegen ist keine zusätzliche Interaktion nötig, da Updates sofort installiert werden, ohne dass der Nutzer etwas davon mitbekommt. [81]

6.2.2 Nutzung und Conversion Rate

Abbildung 25: Vergleich der notwendigen Schritte zur Nutzung der App



Quelle: (Progressive Web App - Die Zukunft von Webapplikationen, 2019)

Bevor eine native App genutzt werden kann fallen 5 Schritte an. Zuallererst muss hierfür der App Store geöffnet werden und die entsprechende App gesucht werden. Nachdem die App gefunden wurde, muss diese heruntergeladen werden, wodurch der User unter Umständen einige Zeit warten muss. Hinzu kommt, dass für das Herunterladen der App eine gute Internetverbindung nötig ist, ansonsten kann der Download viel Zeit in Anspruch nehmen. Nach dem erfolgreichen Download erfolgt die Installation und die App kann geöffnet werden. Nach dem Öffnen der App müssen unter Umständen Berechtigungen erteilt werden, um die App dann vollständig verwenden zu können. Bei einer Progressive Web App fällt ein einziger Schritt an,

[81] Vgl. (Hedemann, 2018)

um die App verwenden zu können. Hierzu muss lediglich die URL aufgerufen werden und die Progressive Web App kann verwendet werden. [82]

Durch Progressive Web Apps wird zudem nachweislich die Conversion Rate gesteigert. Grundlegend umfasst die Conversation Rate den Prozentsatz der Benutzer, die eine gewünschte Aktion abgeschlossen haben. Auf welche Aktion die Conversion Rate genau abzielt ist dabei je nach App unterschiedlich. [83]

Das größtes Ticketingunternehmen Indiens BookMyShow mit über 50 Millionen Besuchern pro Monat hat durch die Einführung einer Progressive Web App die Conversion Rate um **80%** gesteigert. Die Aktion, die dabei gemessen wurde, ist das Kaufen von Tickets für eine Veranstaltung. [84]

Aliba, die größte B2B-Handelsplattform der Welt, hat eine Steigerung der Conversion Rate um **76%** erzielt. Die Progressive Web App hat dabei die mobile Erfahrung hinsichtlich Schnelligkeit und Nutzerfreundlichkeit verbessert. Die Aktion verbunden mit der gemessenen Conversion Rate umfasste dabei ein Besuch der Seite sowie die direkte Kontaktaufnahme mit einem Lieferanten. [85]

Das afrikanisches Onlinehandelsunternehmen Jumia entwickelte für ihre Plattform ebenfalls eine Progressive Web App. Das Hauptaugenmerk lag dabei auf dem Feature „Push Notifications“, das durch Progressive Web Apps möglich ist. Push Notifications wurden dabei eingesetzt, um potenzielle Kunden zu benachrichtigen, falls diese ihren Kauf abgebrochen haben. Zuvor wurden die Kunden per Email benachrichtigt. Durch den Einsatz dieser Progressive Web App Technologie stieg die Conversion Rate für abgebrochene Warenkörbe um das 9-fache. Das bedeutet, dass mehr Kunden wieder dazu animiert wurden, den abgebrochenen Warenkorb wieder aufzunehmen. [86] Eine relativ ähnliche Strategie bezüglich der Push Benachrichtigungen für abgebrochene Warenkörbe hat die Kosmetikmarke Lancôme in ihrem Online Shop angewendet. Die neu entwickelte Progressive Web App hat die

[82] Vgl. (Silva, 2017)
[83] Vgl. (Adjust, 2018)
[84] Vgl. (Developers, BookMyShow’s new Progressive Web App drives an 80% increase in conversions , 2017)
[85] Vgl. (Developers, Alibaba - Case Studies, 2016)
[86] Vgl. (Developers, Case Studies - Jumia, 2016)

Conversion Rate um 12% gesteigert durch das Senden von Push Benachrichtigungen für abgebrochene Warenkörbe. [87]

MakeMyTrip.com ist ein führendes Reiseunternehmen in Indien mit knapp 8 Millionen Besuchern pro Monat. Viele potenzielle Kunden besuchen die Seite mit mobilen Endgeräten, die nur langsames Internet haben, was oftmals dazu führt, dass die native App nicht heruntergeladen wird, was sich negativ auf die Kundenakquisition auswirkte. Durch die Entwicklung einer Progressive Web App stieg die Conversion Rate um das 3-fache. Die Conversion Rate ist dabei auf Erstkäufer bezogen, die durch die Progressive Web App 3-mal mehr konvertieren als Erstbesucher der nativen App. [88]

6.2.3 Funktionsumfang

Ein wichtiger Vergleich von Progressiven Web Apps und nativen Apps stellt der Funktionsumfang dar, den Progressive Web Apps anbieten können. Zum jetzigen Zeitpunkt können Progressive Web Apps nicht dieselben Funktionen anbieten, die native Apps unterstützen. Je nach App benötigt man verschiedenen Funktionalitäten des Smartphones. Beispielsweise benötigt WhatsApp Zugriff auf die Kontakte, was bisher durch Progressive Web Apps nicht möglich ist. Teilweise gibt es auch Funktionen, die auf Android zwar unterstützt werden, jedoch nicht auf iOS. Hierzu gehören zum Beispiel Push Benachrichtigungen, die seitens Android zwar funktionieren, jedoch nicht auf iOS. [89]

Folgend werden einige wichtige Funktionen aufgelistet, die von Android und iOS unterstützt bzw. nicht unterstützt werden. Native Apps haben dabei die volle Unterstützung sämtlicher Funktionen. Für den Vergleich wurde die zu den jetzigen Zeitpunkten aktuellste Android (Android 9 - Pie) sowie iOS (12.4) Version benutzt.

[87] Vgl. (Developers, Lancôme rebuilds their mobile website as a PWA, increases conversions 17%, 2017)
[88] Vgl. (Developers, MakeMyTrip.com's new PWA delivers 3X improvement in conversion rates , 2017)
[89] Vgl. (Chekalin & Gritsay , 2018)

Tabelle 5: Vergleich der unterstützten Funktionen von Android und iOS

Funktion	**Android** Browser: Chrome	**iOS** Browser: Safari
Audio- und Videoaufzeichnung	✓	✓
Erweiterte Kamerasteuerung	✓	✗
Aufzeichnungsmedien	✓	✗
Bluetooth	✓	✗
Online Status	✓	✓
Vibration	✓	✗
Akkustand	✓	✗
Push-Nachrichten	✓	✗
Offline Speicher	✓	✓
Dateizugriff	✓	✓
Kontakte	✗	✗
SMS	✗	✗
Touch-Gesten	✓	✓
Spracherkennung	✓	✗
Offline Modus	✓	✓
Hintergrund-Synchronisation	✓	✗
Zahlungen	✓	✓

Geolokalisierung	✓	✓
Geräteposition	✓	✓
Gerätebewegung	✓	✓

[90] [91]

In der Auflistung ist klar zu sehen, dass Android eine deutlich bessere Unterstützung für Progressive Web Apps hat. Apple ist generell in der Unterstützung von Features für Progressive Web Apps sehr langsam im Vergleich zum Chrome Browser, der unter Android verwendet wird. [92]

Mit iOS 11.3 gab es die erste Unterstützung für Progressive Web Apps, dessen Implementierung jedoch nicht sehr ausgereift war. So wird zum Beispiel der Cache gelöscht, wenn die App einige Zeit nicht verwendet wird, wodurch die Offline Funktionalität verloren ging. Apple macht zwar erste Schritte für die Unterstützung von Progressive Web Apps, geht jedoch nur sehr kleine Schritte voran. [93] [94]

6.2.4 Sicherheit

Tabelle 6: Vergleich der unterstützten Sicherheitsfunktionen von Progressive Web Apps und Nativen Apps

Funktion	**Progressive Web App**	**Native App**
Sicherheitsprüfung durch App Store	✗	✓
Automatische Sicherheitsupdates	✓	✗
Zugriff auf Sicherheitsfeatures (Face ID, Touch ID)	✗	✓
HTTPS Pflicht	✓	✗

[90] Vgl. (Chekalin & Gritsay , 2018)
[91] Vgl. (Bar, 2019)
[92] Vgl. (Love, 2018)
[93] Vgl. (Rixecker, 2018)
[94] Vgl. (Firtman, 2019)

Ein wichtiger Aspekt, der bei Progressive Web Apps verloren geht, ist die Sicherheitsprüfung durch den App Store. Jede App, die im App Store eingereicht wird, wird durch Reviews geprüft und muss bestimmte Eigenschaften erfüllen, um im App Store veröffentlicht zu werden. Dieser Prozess sorgt für mehr Sicherheit der Apps und vermeidet Apps mit schadhaftem Code. Wenn eine App im App Store erscheint, bedeutet dies, dass die App die Sicherheits- und Leistungsstandards bestanden hat. Natürlich kann nicht komplett ausgeschlossen werden, dass eine App unsicher ist, jedoch ist diese Sicherheitsprüfung ein Schritt mehr, den Progressive Web Apps nicht durchlaufen müssen. [95] [96] [97]

Ein entscheidender Vorteil von Progressive Web Apps gegenüber von nativen Apps ist die Update-Fähigkeit ohne Zustimmung des Benutzers. Wenn eine App eine Sicherheitslücke aufweist, ist es Progressive Web Apps möglich, ein Update zu veröffentlichen, das sofort jedem Benutzer zur Verfügung steht, ohne dessen Zustimmung zu benötigen. Bei nativen Apps hingegen muss der Benutzer selbst aktiv werden und das Updated Installieren, wodurch kritische Sicherheitslücken nach wie vor auf vielen Geräten vorhanden sein können, sofern diese kein Update durchführen. [98]

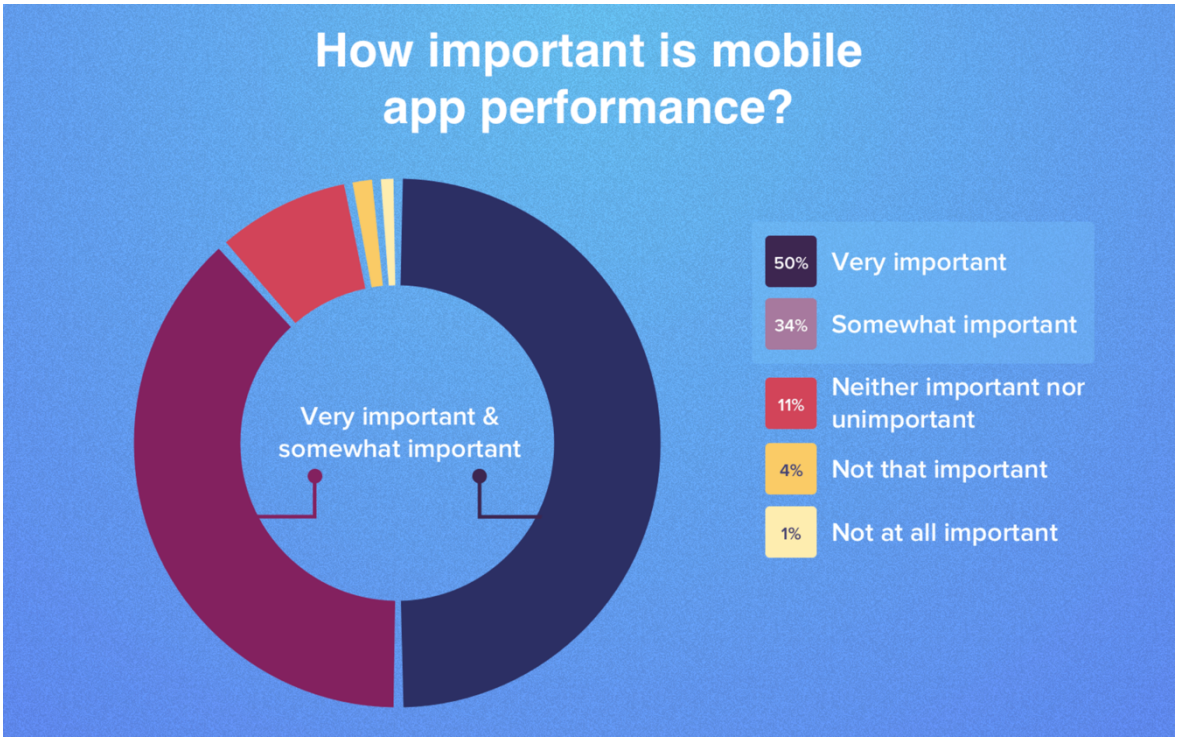
Der Zugriff auf Sicherheitsfunktionen, die im Smartphone verbaut sind, wie Touch ID oder Face ID, bleiben Progressive Web Apps derzeit verwehrt, wodurch ein wichtiges Feature nicht verfügbar ist, dass beispielsweise bei Finanz Apps oftmals verwendet wird, um eine hohe Sicherheit zu garantieren. [99]

Da Service Workers, die von Progressive Web Apps verwendet werden, HTTPS als Pflicht haben und ohne gar nicht funktionieren können, wird gewährleistet, dass eine sichere Verbindung besteht und Informationen geschützt sind und nicht manipuliert werden können. [100] [101]

[95] Vgl. (Media, 2018)
[96] Vgl. (Modi, 2018)
[97] Vgl. (Raczynski, 2018)
[98] Vgl. (Media, 2018, S. 5)
[99] Vgl. (Zibreg, 2018)
[100] Vgl. (Media, 2018)
[101] Vgl. (Liebel, Faktencheck zu Progressive Web Apps, Teil 2: Sicherheit und Privatsphäre, 2019)

6.2.5 Geschwindigkeit

Wie wichtig die Geschwindigkeit einer App für User ist zeigt eine von Dynatrace durchgeführte Studie. Dabei sind über 80% der Befragten der Ansicht, dass Geschwindigkeit ein wichtiges Kriterium für sie ist, wenn sie eine App verwenden.



Quelle: (Kuprenko, 2018)

Bezüglich Progressive Web Apps zeigen viele Fallstudien, dass durch den Einsatz von Progressive Web Apps teilweise sehr hohe Verbesserungen bei der Geschwindigkeit erzielt werden. Jedoch sind vielen dieser Fallstudien auf den Vergleich von Progressive Web Apps und der Webseite zuvor bezogen. [102]

Betrachtet man den Vergleich der Geschwindigkeit einer Progressive Web App und einer nativen App, ist die native Apps in den meisten Fällen schneller. [103] [104]

Der Grund, dass native Apps schneller sind als Progressive Web Apps ist, da Progressive Web Apps eine zusätzliche „Ebene“ brauchen um zu funktionieren, nämlich den Browser. Dieser dient als Vermittler zwischen dem Betriebssystem und der App, wodurch die Geschwindigkeit verlangsamt wird. [105] [106]

[102] Vgl. (Four, 2019)
[103] Vgl. (Developers, Twitter Lite PWA Significantly Increases Engagement and Reduces Data Usage, 2017)
[104] Vgl. (Miller, 2018)
[105] Vgl. (Kuprenko, 2018)
[106] Vgl. (Miller, 2018)

7 Schlussbetrachtung

7.1 Fazit

In dieser Bachelorarbeit setzte man sich zuerst mit den grundlegenden Funktionalitäten einer Progressive Web App auseinander und der Grundlage, was eine Progressive Web App auszeichnet gegenüber eine normalen Web Applikation. Dabei wurde deutlich, dass eine Progressive Web App viele Funktionalitäten unterstützt, die bisher nur nativen Applikationen vorbehalten waren wie beispielsweise Push Benachrichtigungen, Offline Funktionalität oder die Installation auf dem Endgerät.

Im Anschluss erfolgten die Planung sowie die Umsetzung einer Progressive Web App, die dafür verantwortlich ist, Teilnehmer am Tag des Events einchecken zu können. Hierfür wurde sowohl eine REST API entwickelt, die für die Lese- und Schreiboperationen verantwortlich ist, als auch die Progressive Web Apps selbst, die zuerst als Single-Page Application entwickelt wurde und danach zu einer Progressive Web App migriert wurde. Bei der Entwicklung wurde deutlich, dass viele Funktionalitäten, die eine Progressive Web App benötigt, durch bereits existierende Tools wie Workbox stark vereinfacht wird und dadurch schnell Ergebnisse erzielt werden können.

Letztlich schaffte die Evaluierung von Progressive Web Apps und nativen Apps Klarheit darüber, in welchen Kriterien die jeweiligen Apps ihre Stärken und Schwächen haben. Hier ist als Fazit klar zu sagen, dass derzeit nicht alle nativen Applikationen durch Progressive Web Apps ersetzt werden können. Der Grund hierfür liegt daran, dass Progressive Web App nicht alle Funktionalitäten unterstützen wie es eine native App tut. So ist es zum Beispiel nicht möglich auf Kontakte zuzugreifen. Ein durchaus schwerwiegender Punkt stellt dabei das Betriebssystem iOS dar, bei dem im Verhältnis zu Android sehr wenige Funktionen unterstützt werden. So verwehrt iOS beispielsweise Progressive Web Apps den Zugriff auf Aufzeichnungsmedien wie das Mikrofon, den Vibrationsalarm, Push Benachrichtigungen und vieles mehr. Durch diese fehlenden Unterstützungen ist es unmöglich, bestimmte native Applikationen als Progressive Web App zu entwickeln. In dieser Arbeit wurde jedoch noch ein ganz anderer Punkt deutlich, der zuvor nicht in Betracht gezogen wurde. Durch Progressive Web Apps ist es vielen Webseitenbetreiber plötzlich möglich, dem User sehr einfach eine mobile Applikation bereitzustellen. Wie viele Fallstudien gezeigt haben, konnten durch den Einsatz der Funktionalitäten, die Progressive Web Apps anbieten, die Conversion Rates teilweise enorm gesteigert werden. Es

kommt letztlich auf die Art der Applikation an, die umgesetzt werden soll und welche Funktionalitäten diese benötigt.

7.2 Ausblick

Progressive Web Apps stellen ein großes Potenzial dar, das bereits in vielen Bereichen eingesetzt werden kann. Vor allem für Webseiten, die bisher keine native App anbieten, öffnen Progressive Web Apps eine neue Möglichkeit, die Kunden besser zu erreichen und den Web-Auftritt zu verbessern. Die Zukunft von Pogressive Web Apps ist jedoch stark abhängig von der Browserentwicklung für die jeweiligen Endgeräte. iOS blockiert den Fortschritt von Progressive Web Apps derzeit durch die fehlende Unterstützung vieler wichtiger Funktionen, wodurch mit Sicherheit viele von der Entwicklung einer Progressive Web App abgeschreckt werden, da eine App, die nur auf manchen Endgeräten funktioniert, wenig ansprechend ist.

Sofern Google und Apple die Entwicklung weiter vorantreiben, haben Progressive Web Apps eine durchaus vielversprechende Zukunft, wodurch viele native Apps durch Progressive Web Apps ersetzt werden können. Ob Progressive Web Apps jemals alle nativen Apps ersetzen werden ist jedoch zu bezweifeln, da native Apps näher am Betriebssystem sind, wodurch Vorteile erlangt werden, die Progressive Web Apps nicht nutzen können. Ein weiterer Bereich, der die Zukunft von Progressive Web Apps spannend bleiben lässt, ist die Entwicklung für Apps auf dem Desktop. Bereits jetzt ist es möglich, Progressive Web Apps auf dem Desktop zu nutzen. Durch den Vorteil der Nutzbarkeit auf vielen Endgeräten (Computer, Tablet, Smartphone) schaffen Progressive Web Apps eine universale Lösung, die durch die Weiterentwicklung ein enormes Potenzial darstellen - sowohl für den Nutzer als auch für Entwickler.

8 Literaturverzeichnis

(07. 03 2019). Abgerufen am 10. 06 2019 von 1&1 IONOS Digital Guide: https://www.ionos.de/digitalguide/websites/web-entwicklung/verschiedene-app-formate-was-ist-eine-web-app/

A secure web is here to stay. (02 2018). Abgerufen am 22. 06 2019 von https://security.googleblog.com/2018/02/a-secure-web-is-here-to-stay.html

Adjust. (23. 11 2018). *Conversion Rate*. Abgerufen am 16. 08 2019 von https://www.adjust.com/glossary/conversion-rate/

AltexSoft. (10. 05 2018). Abgerufen am 17. 06 2019 von https://www.altexsoft.com/blog/engineering/progressive-web-apps/

Apple. (12. 12 2016). *Configuring Web Applications*. Abgerufen am 15. 07 2019 von https://developer.apple.com/library/archive/documentation/AppleApplications/Reference/SafariWebContent/ConfiguringWebApplications/ConfiguringWebApplications.html#//apple_ref/doc/uid/TP40002051-CH3-SW1

Appscope. (2019). Abgerufen am 05. 07 2019 von https://appsco.pe

Ater, T. (2017). *Building Progressive Web Apps.* O'Reilly UK Ltd.

Automated Certificate Management. (2019). Abgerufen am 18. 07 2019 von https://devcenter.heroku.com/articles/automated-certificate-management

Bar, A. (2019). *What Web Can Do Today*. Abgerufen am 17. 08 2019 von https://whatwebcando.today/

Beyond the Rack - Google Developers. (2015). Von https://developers.google.com/web/showcase/2015/beyond-the-rack abgerufen

Can I use. (2019). Abgerufen am 03. 07 2019 von https://caniuse.com/#search=service%20worker

Chekalin, D., & Gritsay , K. (2018). *Progressive Web Apps vs Native Apps*. Abgerufen am 16. 08 2019 von https://www.codica.com/blog/progressive-web-apps-vs-native/

Client-Server — e-teaching. (21. 07 2015). Abgerufen am 07. 06 2019 von https://www.e-teaching.org/technik/vernetzung/architektur/client-server

Copes, F. (17. 02 2018). *Service Workers explained*. Abgerufen am 07. 08 2019 von https://flaviocopes.com/service-workers/

Dev Insider. (03. 08 2018). Abgerufen am 10. 06 2019 von https://www.dev-insider.de/was-ist-ein-sdk-a-730921/

Developers, G. (11 2016). *Alibaba - Case Studies*. Abgerufen am 16. 08 2019 von https://developers.google.com/web/showcase/2016/alibaba

Developers, G. (2019. 05 2016). *Case Studies - Jumia*. Abgerufen am 16 08 von https://developers.google.com/web/showcase/2016/jumia

Developers, G. (10 2017). *BookMyShow's new Progressive Web App drives an 80% increase in conversions* . Abgerufen am 16. 08 2019 von https://developers.google.com/web/showcase/2017/bookmyshow

Developers, G. (05 2017). *Lancôme rebuilds their mobile website as a PWA, increases conversions 17%*. Abgerufen am 16. 08 2019 von https://developers.google.com/web/showcase/2017/lancome

Developers, G. (05 2017). *MakeMyTrip.com's new PWA delivers 3X improvement in conversion rates* . Abgerufen am 16. 08 2019 von https://developers.google.com/web/showcase/2017/make-my-trip

Developers, G. (2017). *Twitter Lite PWA Significantly Increases Engagement and Reduces Data Usage*. Abgerufen am 18. 08 2019 von https://developers.google.com/web/showcase/2017/twitter

Django. (2019). *Models | Django documentation | Django*. Abgerufen am 25. 06 2019 von https://docs.djangoproject.com/en/2.2/topics/db/models/

Django REST framework. (2019). Abgerufen am 09. 07 2019 von https://www.django-rest-framework.org/

Domin, A. (25. 04 2018). *t3n – digital pioneers*. Abgerufen am 07. 06 2019 von https://t3n.de/news/single-page-webanwendung-1023623/

Elektronik Kompendium. (21. 05 2019). Abgerufen am 07. 06 2019 von https://www.elektronik-kompendium.de/sites/net/2101151.htm

Firtman, M. (26. 03 2019). *What's new on iOS 12.2 for Progressive Web Apps*. Abgerufen am 17. 08 2019 von https://medium.com/@firt/whats-new-on-ios-12-2-for-progressive-web-apps-75c348f8e945

Flixbus. (2019). Von https://apps.apple.com/de/app/flixbus-fernbus-durch-europa/id778437357 abgerufen

Four, C. (2019). *A community-driven list of stats and news related to Progressive Web Apps*. Abgerufen am 18. 08 2019 von https://www.pwastats.com/

Gallagher, N. (06. 04 2017). *How we built Twitter Lite*. Abgerufen am 06. 07 2019 von https://blog.twitter.com/engineering/en_us/topics/open-source/2017/how-we-built-twitter-lite.html

Gaunt, M. (2019). *Service Workers: an Introduction - Google Developers*. Abgerufen am 19. 07 2019 von https://developers.google.com/web/fundamentals/primers/service-workers/

Gaunt, M., & Kinlan, P. (2019). *The Web App Manifest*. Abgerufen am 17. 06 2019 von https://developers.google.com/web/fundamentals/web-app-manifest/

Generic views - Django REST framework. (2019). Abgerufen am 11. 07 2019 von https://www.django-rest-framework.org/api-guide/generic-views/

Google Case Studies. (2017). Abgerufen am 06. 07 2019 von https://developers.google.com/web/showcase/2017/twitter

Google Developers - Progressive Web App Architectures. (29. 05 2019). Abgerufen am 12. 06 2019 von https://developers.google.com/web/ilt/pwa/introduction-to-progressive-web-app-architectures

Google Developers - Progressive Web Apps . (2019). Abgerufen am 09. 06 2019 von https://developers.google.com/web/progressive-web-apps/

Google Lighthouse Report. (14. 08 2019). Von https://googlechrome.github.io/lighthouse/viewer/?gist=384efd12ffe671cdbd5344291509120f abgerufen

Hedemann, F. (27. 10 2018). *Progressive Web Apps: Wenn Web und App verschmelzen*. Abgerufen am 16. 08 2019 von https://dmexco.com/de/stories/progressive-web-apps-wenn-web-und-app-verschmelzen/

Hiwarale, U. (15. 04 2018). *So what the heck is JWT or JSON Web Token?* Abgerufen am 03. 08 2019 von https://itnext.io/so-what-the-heck-is-jwt-or-json-web-token-dca8bcb719a6

Irish, P. (2016). *Basic Service Worker Sample* . Von https://googlechrome.github.io/samples/service-worker/basic/ abgerufen

JSON Web Tokens. (2019). Abgerufen am 03. 08 2019 von https://jwt.io/

Kölpin, S. (03. 07 2017). *Auf dem Weg zur SPA*. Abgerufen am 06. 08 2019 von https://jaxenter.de/enterprisetales-auf-dem-weg-zur-spa-58941

Kastner, V. (23. 05 2019). Abgerufen am 03. 08 2019 von https://www.scip.ch/?labs.20190523

Keklik, C. (16. 03 2018). Abgerufen am 19. 07 2019 von https://medium.com/commencis/what-is-service-worker-4f8dc478f0b9

Kuprenko, V. (10. 05 2018). *Native Apps vs Progressive Web Apps: Who is Going to Win This Battle?* . Abgerufen am 18. 08 2019 von https://www.cleveroad.com/blog/progressive-web-apps-vs-native-apps-who-is-going-to-win-this-battle

Let's Encrypt. (2019). Abgerufen am 21. 06 2019 von https://letsencrypt.org/

Liebel, C. (2018). *Progressive Web Apps - Das Praxisbuch.* Rheinwerk Computing.

Liebel, C. (24. 01 2019). *Faktencheck zu Progressive Web Apps, Teil 2: Sicherheit und Privatsphäre*. Abgerufen am 17. 08 2019 von https://www.heise.de/developer/artikel/Faktencheck-zu-Progressive-Web-Apps-Teil-2-Sicherheit-und-Privatsphaere-4259191.html

Lighthouse. (2019). Abgerufen am 14. 08 2019 von https://developers.google.com/web/tools/lighthouse/

Love, C. (03. 05 2018). Abgerufen am 17. 08 2019 von https://love2dev.com/blog/what-browsers-support-service-workers/

Love, C. (2018). *Progressive Web Application Development by Example.* Packt Publishing .

Love, C. (28. 10 2018). *Why Progressive Web Applications Are Not Single Page Applications*. Von https://love2dev.com/blog/pwa-spa/ abgerufen

Mühlroth, A. (11. 02 2019). *Microsoft warnt vor dem eigenen Browser*. Abgerufen am 03. 07 2019 von https://www.techbook.de/apps/software/microsoft-warnt-vor-ie

Marchick, A. (12. 01 2015). *Segment*. Abgerufen am 21. 06 2019 von 5 Ways to Craft Push Notifications That Users Actually Want : https://segment.com/blog/push-notifications-users-want-kahuna/

Marking extra actions for routing - Django REST framework. (2019). Abgerufen am 11. 07 2019 von https://www.django-rest-framework.org/api-guide/viewsets/#marking-extra-actions-for-routing

Media, T. (2018). *Web vs Native app*. Abgerufen am 17. 08 2019 von http://www.newformat.se/documents/twixl-media/twixl-media-white-papers/newformat-ab-sweden-twixl-publisher-pwa-vs-native-apps-white-paper-2018-02-22.pdf

Miller, P. (11. 04 2018). *Web apps are only getting better*. Abgerufen am 18. 08 2019 von https://www.theverge.com/circuitbreaker/2018/4/11/17207964/web-apps-quality-pwa-webassembly-houdini

Mobile Zeitgeist. (26. 09 2017). Abgerufen am 10. 06 2019 von https://www.mobile-zeitgeist.com/app-entwicklung-101-nativ-web-hybrid-oder-cross-plattform/?cookie-state-change=1561375496721

Modi, A. (28. 07 2018). *Progressive Web Apps - Blurring the Lines between Web and Mobile Apps*. Abgerufen am 17. 08 2019 von https://dev.to/topsinfosol/progressive-web-apps---blurring-the-lines-between-web-and-mobile-apps-53i

Mozilla Developer Network. (04. 06 2019). Abgerufen am 12. 06 2019 von Progressive web app structure: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/App_structure

Netflix. (2019). Von https://apps.apple.com/de/app/netflix/id363590051 abgerufen

OneSignal. (2019). Abgerufen am 07. 08 2019 von https://onesignal.com/

Osmani, A. (12 2015). *Progressive Web Apps - Google Developers*. Abgerufen am 18. 06 2019 von https://developers.google.com/web/updates/2015/12/getting-started-pwa

Progressive Web App - Die Zukunft von Webapplikationen. (2019). Von https://atmina.de/was-wir-tun/pwa/ abgerufen

Progressive Web Apps | Google Developers. (2019). Abgerufen am 06. 08 2019 von https://developers.google.com/web/progressive-web-apps/

Rück, H. (19. 02 2018). Abgerufen am 26. 04 2019 von Gabler Wirtschaftslexikon: https://wirtschaftslexikon.gabler.de/definition/event-34760/version-258256

Raczynski, M. (06. 04 2018). *Native App or a Progressive Web App (PWA) – Which to Choose and When?* Abgerufen am 17. 08 2019 von https://insanelab.com/blog/web-development/native-app-vs-progressive-web-app-pwa/

Rixecker, K. (09. 02 2018). *Progressive-Web-Apps unter iOS 11.3: Apple macht es Entwicklern nicht einfach*. Abgerufen am 17. 08 2019 von https://t3n.de/news/progressive-web-apps-ios-safari-942769/

Routers - Django REST framework. (2019). Abgerufen am 10. 07 2019 von https://www.django-rest-framework.org/api-guide/routers/

Sachchithananthan, R. (2019). *Einsatz von JSON Web Token für die Authentifikation in Webanwendungen* . Abgerufen am 03. 08 2019 von https://rathes.me/blog/de/json-web-token/

Searchmetrics. (21. 08 2017). Abgerufen am 09. 06 2019 von https://www.searchmetrics.com/de/glossar/progressive-web-apps/

Seobility. (2019). Abgerufen am 17. 06 2019 von https://www.seobility.net/de/wiki/Progressive_Web_Apps

Serializers - Django REST framework. (2019). Abgerufen am 10. 07 2019 von https://www.django-rest-framework.org/api-guide/serializers/

Service Worker API - MDN Web Docs. (2019). Abgerufen am 19. 07 2019 von https://developer.mozilla.org/de/docs/Web/API/Service_Worker_API

Silva, K. d. (15. 08 2017). *Progressive Web Apps: The Next Mobile Experience?* . Abgerufen am 16. 08 2019 von https://www.biznessapps.com/blog/progressive-web-apps/

Sopheaktra, Y. (23. 01 2019). *Build a PWA using Workbox*. Abgerufen am 08. 08 2019 von https://medium.com/the-web-tub/build-a-pwa-using-workbox-2eda1ef51d88

Spotify. (2019). Von https://apps.apple.com/de/app/spotify-musik-und-podcasts/id324684580 abgerufen

Srocke, D., & Karlstetter, F. (02. 10 2017). *Cloud Computing Insider*. Abgerufen am 07. 06 2019 von https://www.cloudcomputing-insider.de/was-ist-eine-web-app-a-644292/

Stecky-Efantis, M. (16. 05 2016). *5 Easy Steps to Understanding JSON Web Tokens (JWT)* . Abgerufen am 03. 08 2019 von https://medium.com/vandium-software/5-easy-steps-to-understanding-json-web-tokens-jwt-1164c0adfcec

Twitter by the Numbers. (06. 01 2019). Abgerufen am 06. 07 2019 von https://www.omnicoreagency.com/twitter-statistics/

Vega, J. (28. 02 2017). *freeCodeCamp*. Abgerufen am 12. 06 2019 von https://www.freecodecamp.org/news/what-exactly-is-client-side-rendering-and-hows-it-different-from-server-side-rendering-bd5c786b340d/

Views - Django REST framework. (2019). Abgerufen am 10. 07 2019 von https://www.django-rest-framework.org/api-guide/views/

Viewsets - Django REST framework. (2019). Abgerufen am 10. 07 2019 von https://www.django-rest-framework.org/api-guide/viewsets/

WADI, M. (2019). *Progressive Web Apps HQ - Discoverable*. Abgerufen am 17. 06 2019 von https://mozilla.github.io/progressive-apps-hq/features/discoverable/

WADI, M. (2019). *Progressive Web Apps HQ - Installable*. Abgerufen am 18. 06 2019 von https://mozilla.github.io/progressive-apps-hq/features/installable/

wendweb. (2019). Abgerufen am 13. 06 2019 von https://www.responsive-webdesign.mobi/was-ist-responsive-webdesign/

What is Heroku? (2019). Abgerufen am 18. 07 2019 von https://www.heroku.com/about

Workbox - Google Developers. (2019). Von https://developers.google.com/web/tools/workbox/ abgerufen

Zibreg, C. (19. 07 2018). *What Progressive Web Apps are and how to install and use them on iPhone and iPad*. Abgerufen am 17. 08 2019 von https://www.idownloadblog.com/2018/07/19/howto-progressive-web-apps-iphone-ipad/

9 Ehrenwörtliche Erklärung

Ich versichere, dass ich die beiliegende Bachelorarbeit mit dem Thema „Progressive Web Apps – die Zukunft nativer Applikationen? Konzeption und Realisierung einer Progressive Web App zum Einsehen von Teilnehmerdaten eines Events“ selbständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie alle wörtlich oder sinngemäß übernommenen Stellen in der Arbeit gekennzeichnet habe.

29.08.2019 _____

Datum

Unterschrift