TECHNISCHE HOCHSCHULE NÜRNBERG
GEORG SIMON OHM

Fakultät Informatik

Betrachtung aktueller Grenzen von Progressive Web Apps im Gegensatz zu Native Apps

Bachelorarbeit im Studiengang Medieninformatik

vorgelegt von

Mahja Sarschar

Matrikelnummer 320 1818

Erstgutachter:	Prof. Dr. Mathias Teßmann
Zweitgutachter:	Prof. Dr. Christian Schiedemeier

Hinweis: Diese Erklärung ist in alle Exemplare der Abschlussarbeit fest einzubinden. (Keine Spiralbindung)

TECHNISCHE HOCHSCHULE NÜRNBERG
GEORG SIMON OHM

Prüfungsrechtliche Erklärung der/des Studierenden

Angaben des bzw. der Studierenden:

Name: Sarschar Vorname: Mahja Matrikel-Nr.: 3201818

Fakultät: Informatik Studiengang: Medieninformatik

Semester: Sommersemester 2021

Titel der Abschlussarbeit:

Betrachtung aktueller Chancen von Progressive Web Apps im Gegensatz zu Native Apps

Ich versichere, dass ich die Arbeit selbständig verfasst, nicht anderweitig für Prüfungszwecke vorgelegt, alle benutzten Quellen und Hilfsmittel angegeben sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Ort, Datum, Unterschrift Studierende/Studierender

Erklärung zur Veröffentlichung der vorstehend bezeichneten Abschlussarbeit

Die Entscheidung über die vollständige oder auszugsweise Veröffentlichung der Abschlussarbeit liegt grundsätzlich erst einmal allein in der Zuständigkeit der/des studentischen Verfasserin/Verfassers. Nach dem Urheberrechtsgesetz (UrhG) erwirbt die Verfasserin/der Verfasser einer Abschlussarbeit mit Anfertigung ihrer/seiner Arbeit das alleinige Urheberrecht und grundsätzlich auch die hieraus resultierenden Nutzungsrechte wie z.B. Erstveröffentlichung (§ 12 UrhG), Verbreitung (§ 17 UrhG), Vervielfältigung (§ 16 UrhG), Online-Nutzung usw., also alle Rechte, die die nicht-kommerzielle oder kommerzielle Verwertung betreffen.

Die Hochschule und deren Beschäftigte werden Abschlussarbeiten oder Teile davon nicht ohne Zustimmung der/des studentischen Verfasserin/Verfassers veröffentlichen, insbesondere nicht öffentlich zugänglich in die Bibliothek der Hochschule einstellen.

Hiermit ☐ genehmige ich, wenn und soweit keine entgegenstehenden Vereinbarungen mit Dritten getroffen worden sind,

☐ genehmige ich nicht,

dass die oben genannte Abschlussarbeit durch die Technische Hochschule Nürnberg Georg Simon Ohm, ggf. nach Ablauf einer mittels eines auf der Abschlussarbeit aufgebrachten Sperrvermerks kenntlich gemachten Sperrfrist

von Jahren (0 - 5 Jahren ab Datum der Abgabe der Arbeit),

der Öffentlichkeit zugänglich gemacht wird. Im Falle der Genehmigung erfolgt diese unwiderruflich; hierzu wird der Abschlussarbeit ein Exemplar im digitalisierten PDF-Format auf einem Datenträger beigefügt. Bestimmungen der jeweils geltenden Studien- und Prüfungsordnung über Art und Umfang der im Rahmen der Arbeit abzugebenden Exemplare und Materialien werden hierdurch nicht berührt.

Ort, Datum, Unterschrift Studierende/Studierender

Formular drucken

Datenschutz: Die Antragstellung ist regelmäßig mit der Speicherung und Verarbeitung der von Ihnen mitgeteilten Daten durch die Technische Hochschule Nürnberg Georg Simon Ohm verbunden. Weitere Informationen zum Umgang der Technischen Hochschule Nürnberg mit Ihren personenbezogenen Daten sind unter nachfolgendem Link abrufbar: **https://www.th-nuernberg.de/datenschutz/**

Kurzdarstellung

Kurze Zusammenfassung der Arbeit, höchstens halbe Seite. Deutsche Fassung auch nötig, wenn die Arbeit auf Englisch angefertigt wird.

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Inhaltsverzeichnis

Kapitel 1.

Einleitung

You can also write footnotes.[1] Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

1.1. Zielsetzung

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

1.2. Umfeld

Das Thema der Arbeit wird in Zusammenarbeit mit dem IT-Consulting Unternehmen OPITZ CONSULTING bearbeitet. Die Betreuung findet durch Senior Consultant Michael Müller statt. Das Unternehmen entwickelt Lösungen für seine Kunden in den Bereichen

[1] Footnotes will be positioned automatically.

Applications, Analytics, Infrastructure und Integration. Da sich die Software- und Webentwicklungsbranche rasant verändert und weiterentwickelt, ist es für die Firma unerlässlich, neue Technologien und Innovationen zu erkennen und Know-How in diesen zu entwickeln, um potenziellen Kunden eine solche Lösung anzubieten. Besonders interessant sind hierbei diejenigen Innovationen, die mit wenig Aufwand viele Vorteile mit sich bringen.

1.3. Aufbau der Arbeit

Im Rahmen dieser Arbeit soll zuerst auf die Grundlagen von mobilen Anwendungen sowie der JavaScript-Bibliothek React und dem Framework React Native eingegangen werden. Im dritten Kapitel wird ein Kriterienkatalog festlegen, anhand dessen die implementierten Anwendungen später beurteilt werden. Demnach sollen Entwicklungsaufwand, Kompatibilität und Funktionalitäten, die mit dem jeweiligen Ansatz umgesetzt werden können, gegeneinander abgewägt werden. Im Anschluss wird die Programmierung der zwei Applikationen vorgestellt und auf spezielle Vorgehensweisen eingegangen. Sinn der Apps ist es, die aktuellen Covid-19 Fallzahlen der offiziellen Datenbank des Robert-Koch-Instituts darzustellen und weiterführende Funktionalitäten wie unter anderem Offline-Betrieb, Standortzugriff oder Benachrichtigungen zu unterstützen. Zuletzt sollen die Anwendungen anhand des Kriterienkatalogs bewertet und verglichen werden.

Kapitel 2.

Grundlagen

In diesem Kapitel werden die für diese Arbeit notwendigen Grundlagen aufgezeigt. Diese sollen für ein einheitliches Verständnis des Themas sorgen und unterschiedliche Vorstellungen von Fachbegriffen angleichen.
Generell lassen sich mobile Anwendungen in drei Kategorien einteilen: Native, Hybrid und Web Applikationen. Im Folgenden wird einerseits genauer auf Native und Hybride Applikationen und anderseits auf eine spezielle Form von Web Apps, genannt Progressive Web Apps, eingegangen.

2.1. Native Applikationen

Als native Applikationen bezeichnet man Anwendungen, die plattformspezifisch – ergo speziell für ein Betriebssystem – implementiert werden. Das wird dadurch ermöglicht, dass sie mit dem Software Development Kit (SDK) des Plattformherstellers entwickelt werden und somit kompletten Zugriff auf jegliche Funktionalitäten der Geräte besitzen. Außerdem entsteht durch Verwendung der plattformspezifischen UI-Komponenten[1], sogenannten Views, eine einheitliche Benutzerschnittstelle, die sich in allen Anwendungen des Betriebssystems widerspiegelt. Um solche Apps mit ihrem SDKs zu entwickeln, wird die zugehörige Programmiersprache verwendet. Die bekanntesten Programmiersprachen sind Java oder Kotlin für Android Geräte, Objective-C und Swift für iOS und .Net für Windows Phone [1]. Eine aktuelle Statistik von statcounter macht deutlich, dass die am meisten verbreiteten Betriebssysteme Android und iOS sind, weswegen im Verlauf dieser Arbeit ausschließlich auf ebendiese eingegangen wird [2].
Um Applikationen im jeweiligen App-Store veröffentlichen zu können, müssen diese eindeutig identifizierbar sein. Das erfolgt durch den Prozess des sogenannten *Signings*, der ebenfalls plattformabhängig durchgeführt wird.

[1] User Interface Komponenten, z. Dt. Benutzeroberflächenkomponenten

2.2. Cross-platform Applikationen

Unter cross-platform (z. Dt. plattformunabhängig) Applikationen versteht man Anwendungen, die auf einer Code-Basis aufbauen und zur Laufzeit zu mehreren Anwendungen für unterschiedliche Endgeräte kompiliert werden. Außerdem können sie in herkömmlichen Quelltext-Editoren entwickelt werden, da sie im Gegensatz zu nativen Anwendungen unabhängig von bestimmten Betriebssystemen oder SDKs sind. Der Vorteil solcher Applikationen ist, dass sie, obwohl sie auf demselben Code basieren, sich komplett den plattformspezifischen Stil anpassen. Somit verringern sich auch die Entwicklungskosten, die bei Implementierung von jeweils einer Anwendung pro Betriebssystem anfallen würden. Klar abzugrenzen sind plattformunabhängige Anwendungen von hybriden Anwendungen. Das Endprodukt bei letzterem ist eine Web Applikation, die sich durch eine WebView in einem nativen Container dem Kontext (z. B. Betriebssystem, Auflösung) des Geräts, in dem sie aufgerufen wird, anpasst [3]. Gängige Frameworks zur plattformunabhängigen Entwicklung sind Electron, Ionic und React Native [4]. Diese stellen meist eine begrenzte Anzahl an vorprogrammierten UI-Komponenten und Funktionalitäten zur Verfügung. Es gibt jedoch eine Vielzahl von Community Lösungen, die für wiederkehrende Problematiken eine Lösung bieten.

2.3. Progressive Web Apps

Progressive Web Apps (PWAs) sind in erster Linie Web Applikationen, also Anwendungen, die mit den üblichen Web Technologien wie HTML, CSS und ECMAScript (JavaScript) implementiert sind. Laut Mozilla Developer Network zeichnen sie sich dadurch aus, dass zusätzlich folgende technische Voraussetzungen erfüllt sind: Sie müssen auf einer sicheren Verbindung aufbauen, eine oder mehrere Service Worker besitzen und über ein App Manifest verfügen [5]. Ersteres wird gewährleistet durch eine HTTPS (Hypertext Transfer Protocol Secure) Verbindung, während Service Worker JavaScript-Skripte sind, die im Hintergrund auf dem Client und unabhängig von der Anwendung selbst ausgeführt werden [6]. Sie sind die wichtigste Komponente hinter den meisten Funktionalitäten, die eine PWA bietet, weswegen im Laufe dieses Kapitels genauer auf ebendiese eingegangen wird. Die letzte benötigte Komponente ist das App Manifest. Dabei handelt es sich um eine JSON-Datei, in der Einzelheiten zur Anwendung und deren Installation angegeben werden [7]. Da es sich dabei, trotz der zusätzlichen nativen Funktionalitäten, um eine Web Applikation handelt, lassen sich PWAs über eine URL im Browser aufrufen und sind somit zunächst unabhängig von dem Gerät, auf dem sie aufgerufen werden. Üblicherweise werden Progressive Web Apps als Single Page Application entwickelt. Das bedeutet, dass die Anwendung aus einem einzigen HTML-Dokument besteht und Inhalte dynamisch geladen werden. Ein Vorteil davon ist, dass sich dadurch auch die Anzahl der Anfragen des Clients an den Server verringert, da nicht nach jeder Anfrage die Seite komplett neu angefordert werden muss. Der Nachteil von

SPAs ist, dass im Gegensatz zu klassischen Webanwendungen Funktionalitäten wie Deep-Linking und die Navigation durch die Browser-Steuerelemente eigenständig implementiert werden müssen[8]. Bislang gibt es keinen offiziell definierten Web Standard für diese spezielle Art von Webanwendung.
Ein bedeutsames Konzept hinter Progressive Web Apps ist das „Progressive Enhancement“. Dieses verlangt, dass PWAs auf allen Endgeräten grundlegend funktionieren sollen, und schrittweise – falls der Browser und das Gerät dies unterstützt – in ihrer Funktionalität erweitert werden können [9]. Die Prüfung, ob der Browser die Anforderungen erfüllt, findet meist durch das Window-Objekt statt, dass das aktuelle Browser Fenster repräsentiert. Dieses enthält unter anderem das Navigator Property, das Informationen über den Browser besitzt und die Unterstützung der Service Worker API[2] signalisiert. Dadurch kann wiederum geprüft werden, ob andere Schnittstellen wie die *geolocation* in dem aktuellen Browser unterstützt werden.
Zur Evaluierung von PWAs bietet sich *Lighthouse* an, ein vorinstalliertes DevTools Add-On im Google Chrome Browser. Dort wird per Mausklick ein Testbericht zur aktuellen Anwendung erstellt, in der unter anderem auch Installierbarkeit und PWA-Optimierung geprüft werden. Dabei wird sich an die von `https://web.dev/pwa-checklist/` definierten Kernfunktionalitäten von PWAs orientiert, die lauten: „starts fast, stays fast“, „works in any browser“, „responsive to any screen size“, „provides a custom offline page“, und „is installable“. Ferner werden noch Kriterien angegeben, die die PWA optimal machen, beispielsweise „can be discovered through search“oder „provides context for permission requests“[9].

2.3.1. Service Worker API

Wie bereits erwähnt, ermöglicht der Service Worker Funktionalitäten, die Progressive Web Apps zur Konkurrenz von nativen Anwendungen machen. Dennoch kann die Service Worker API grundsätzlich in jeder Anwendung implementiert werden und ist nicht auf PWAs beschränkt, da es sich bei einem Service Worker um einen Web Worker handelt. Er fungiert dabei als eine Art Proxy Server, der zwischen der Anwendung, dem Browser und dem Netzwerk platziert ist und somit Zugriff auf Netzwerkanfragen besitzt [7]. Durch seine Position hat der Service Worker jedoch keinen Zugriff auf das Document Object Model (DOM). Die Voraussetzung für die Nutzung eines Service Workers ist, dass der verwendete Browser diese unterstützt und die Verbindung über HTTPS läuft [10]. Letzteres wird damit begründet, dass Sicherheitsprobleme wie Man-in-the-middle-Angriffe, die durch die Stellung des Service Workers als Proxy begünstigt werden, vermieden werden können.
Ein Service Worker besitzt die Lifecycle-Events *install*, *activate* und unter anderem das funktionale Event *fetch*. Diese ermöglichen es, auf bestimmten Ereignissen zur Laufzeit einer Anwendung zu reagieren. Der Service Worker wird üblicherweise beim ersten Aufrufen

[2] API ist kurz für Application Programming Interface, z.Dt. Programmierschnittstelle.

der Website registriert. Anschließend ist der Service Worker jederzeit verfügbar und läuft selbstständig im Hintergrund der Anwendung. Während des *Install*-Events sollten diejenigen Dateien in den Cachespeicher aufgenommen werden, die sich im Laufe der Anwendung nicht ändern. Das betrifft beispielsweise Styling Sheets, das App Manifest und Bild Dateien [11]. Außerdem kann auch die index.html, die als Eintrittspunkt von Single-Page-Webanwendungen dient, im Cache abgelegt werden. Das *Active*-Event wird dafür genutzt, veraltete Cachespeicherinhalte zu bereinigen und vorherige Service Worker Registrierungen zu entfernen. Um auf Netzwerkanfragen zu reagieren, gibt es das *Fetch*-Event. Hier ist es möglich, angeforderte Ressourcen ebenfalls im Cache abzulegen. Dafür gibt es verschiedene Strategien, zwischen denen je nach Anwendungsfall der Applikation abgewägt werden kann. Gängige Strategien sind der *Cache First*- oder der *Cache then network*-Ansatz [12]. Durch weitere funktionale Events wie *push*, *notificationclick* und *sync* und die Nutzung von alten und neuen Programmierschnittstellen ermöglicht der Service Worker das moderne, native-like Web. Jede dieser APIs sollten ebenfalls im Sinne von *progressive enhancement* eingebunden werden. Im Laufe dieser Arbeit wird genauer auf die Cache API, die Notification API, die Push API sowie die Geolocation API eingegangen.
Für alle der genannten Programmierschnittstellen ist es nötig, die Erlaubnis des Nutzers zu erfragen. Zugriff und Verwaltung aller erteilten Erlaubnisse bietet die Permissions API. Diese verfügt über ein Permission Registry, das Permissions für Schnittstellen wie *geolocation*, *bluetooth*, *speaker* und *device-info* enthält. Laut dessen W3 Spezifikation gibt es drei Status der Erlaubnis: *granted*, *denied* und *prompt*. Außerdem wird aufgrund des hohen Einflusses von Permissions unterschieden zwischen Funktionalitäten, die in unsicheren Kontexten (HTTPS) und jenen, die nur im sicheren Kontext verwendet werden können [13]. Mittlerweile bietet Google, unter anderem zur vereinfachten Implementierung und Verwaltung von Service Workern, das Tool *Workbox* an. Es handelt sich dabei um eine Bibliothek, die die gängigsten Funktionalitäten von Service Workern zur Verfügung stellt, wodurch wiederkehrende Prozesse eliminiert werden.

2.4. React

React ist eine von Facebook entwickelte, open-source JavaScript-Bibliothek, die seit 2013 publiziert ist. Sie zeichnet sich dadurch aus, dass sie in erster Linie zum Erstellen von User Interfaces entwickelt wurde. Durch ReactDOM-Bibliothek, wird die Anwendung um das Rendern dieser Benutzeroberflächen der erweitert. Daran lässt sich auch erklären, warum React im Gegensatz zu Vue oder Angular eigentlich kein Framework ist. Legt man Projekte mit diesen Frameworks an, erhält man eine Vielzahl von eingebauten Werkzeugen zum entwickeln von skalierbaren, mächtigen Webanwendungen. Im Gegensatz dazu ist React als UI-Bibliothek leichtgewichtig und ermöglicht individuelle Erweiterung zur Anpassung an die Anforderungen des Projekts [14].

Die einfachste Möglichkeit React in einem Projekt zu nutzen ist, es über eine CDN[3] einzubinden. Dabei ist es ein Anliegen der Entwickler, dass nur so wenig React genutzt werden kann, wie benötigt. Ferner ist es möglich React über Package-Manager wie npm in ein Projekt zu importieren oder durch Toolchains[4] wie Create-React-App über die CLI[5] eine Single Page Application zu erstellen [15]. Im Codeausschnitt 2.1 ist ein Beispiel zu sehen, wie eine React Anwendung in ihrer kleinsten Form aussehen kann. Zuerst müssen die Module React und ReactDom importiert werden. Dann wird per Aufruf der *React.createElement(...)*-Funktion mit den Übergabeparametern HTML-Element, Attribute und Inhalt ein React Element erstellt. Zuletzt wird in Zeile 9 die *ReactDOM.render(...)*-Funktion genutzt, um das erstellte Element einem anderen Element zuzuordnen und damit eine Hierarchie zu erzeugen. In diesem Fall konkret dem HTML-Element mit der *id* root. Die *render*-Funktionen kann als weiteren Übergabeparameter die *Properties* eines Elements oder einer Komponente enthalten, worauf im Laufe des Kapitels genauer eingegangen wird.

```
1 import React from 'react';
2 import ReactDOM from 'reactdom';
3
4 var element = React.createElement(
5     'h1',
6     { className: 'greeting},
7 ␣␣␣'Hello world.'
8 );
9 ReactDOM.render(element,␣document.getElementById('root');
```

Listing 2.1: Schlichtes Beispiel der index.js einer React Applikation

Eines der Argumente zur Nutzung eines Programmiergerüsts wie React ist dessen Implementierung des Virtual Document Object Model. Dieses baut auf dem normalen DOM auf und ermöglicht es, dass nur diejenigen UI-Elemente neu gerendert werden, deren Daten sich verändert haben. Des weiteren bietet sich durch die Abkapselung in Komponenten ein hohes Maß an Wiederverwendbarkeit. Außerdem ist React wie bereits erwähnt leichtgewichtig, da es sich bei der Hauptbibliothek nur um die Implementierung der wichtigsten Bestandteile handelt. Weitere Funktionalitäten wie der React Router zur Programmierung von der Navigation in einer SPA oder anderen Bibliotheken können nach Bedarf importiert werden. Ebenso gibt es UI-Bibliotheken wie MaterialUI oder PrimeReact für React, die häufig implementierte Komponenten im modernen Design anbieten. Generell lassen sich Progressive Web Apps jedoch mit jedem Framework oder auch mit einer einfachen Vanilla-JavaScript Implementierung verwirklichen.
Kritik erlangt React vor allem wegen des Vorwurfs, dass gegen das Entwurfsprinzip der

[3] Content Delivery Network
[4] Eine Sammlung von Werkzeugen, die zum unkomplizierten Aufsetzen eines Produkts dienen.
[5] Command Line Interface, z. Dt. Kommandozeile

Trennung der Verantwortlichkeiten (Separation of Concerns) verstößt. Dies wird in der Webentwicklung so umgesetzt, dass verschiedene Technologien wie HTML, CSS und JavaScript jeweils in eigenen Dateien modelliert oder programmiert werden. Im Gegensatz dazu steht jedoch Reacts Syntax Erweiterung JSX. Diese ermöglicht die drei genannten Technologien innerhalb von JavaScript zu entwickeln. Ein Beispiel dafür ist in 2.2 zu sehen. Wichtig ist hier jedoch, dass die Zeile 8 auch in JavaScript geschrieben werden kann, da es sich hierbei letztendlich um den Aufruf der React.createElement(component, props, ...children)-Funktion handelt [16].

```
1  import React from 'react';
2  import ReactDOM from 'reactdom';
3
4  const Example = (props) => {
5      const greeting = 'Hello world'
6
7      return (
8          <h1>{{ greeting }}</h1>
9      )
10 }
```

Listing 2.2: Nutzung von JSX

Außerdem gibt es einige Änderungen, die sich durch diese Art zu programmieren ergeben. Beispielsweise kann in JSX auf das Semikolon am Ende einer Zeile verzichtet werden und die CSS-Klasse *class* nennt sich *className*. Letzteres ist eines von mehreren Syntaxänderungen bei JSX. Das liegt dem zugrunde, dass jeder JSX-Code in JavaScript-Code umkompiliert wird. Das Schlüsselwort *class* ist dabei in JavaScript ein reserviertes Wort für Klassen und nicht für eine CSS-Klasse. Ein weiteres Beispiel ist *htmlFor* statt *for*. Durch diese Syntax bietet React eine inklusive Dateistruktur. Die Entwickler begründen das damit, dass es hier im Gegensatz zu anderen JS-Frameworks, in denen es pro Komponente jeweils eine getrennte HTML-, CSS- und JavaScript-Datei gibt, lediglich um eine Trennung der Technologien, nicht aber der Verantwortlichkeiten, handelt. Diese Syntax hingegen verbinden die Render Logik enger mit den Benutzeroberfläche und führt bei einer korrekten Aufteilung in Komponenten zu einer starken Kohäsion. Das bietet dem Entwicklern mehr Übersichtlichkeit und Verständnis für Zusammengehörigkeit. Auf der anderen Seite werden komplexe Komponenten jedoch durch diese inklusive Struktur schnell unübersichtlich, weshalb es sinnvoll ist, eine Aufteilung in Unterkomponenten angelehnt an deren Funktionalität vorzunehmen. Wichtig ist hierbei auch eine organisierte Ordnerstruktur aufrecht zu erhalten, damit die Anwendung wartbar bleibt.

Im Folgenden wird genauer auf einige Grundkonzepte von React eingegangen, wobei React-spezifische Begriffe bewusst nicht übersetzt werden.

2.4.1. Komponenten

Wenn Teile des Codes abgekapselt und wiederverwendet werden sollen, wird eine Komponente erstellt, die meist in einer Datei mit demselben Namen implementiert wird. React unterscheidet dabei zwischen zustandslosen und klassenbasierten Komponenten. Ersteres bezeichnete ehemals Komponenten, die nur zur Darstellung von zusammengehörigen UI-Elementen genutzt wird. Sie sind schlank, wiederverwendbar und leicht zu warten. Klassenbasierte Komponenten hingegen basieren auf herkömmlichen E6-Klassen und bieten sogenannte States, die lokale Daten einer Komponente verwalten, und einen Lifecycle. Die Hooks API bietet jedoch seit React 16.8 ein neues Konzept an, um States in sogenannten funktionalen Komponenten zu organisieren und somit die Vorteile von zustandslosen und klassenbasieren Komponenten zu vereinen. Im nächsten Kapitel wird genauer auf die Funktionsweise von Hooks eingegangen.
Zur Kommunikation und Datenaustausch zwischen Eltern- und Kind-Komponenten werden Properties und Callback-Methoden genutzt. Ein Property ist eine Art Übergabeparameter, die von der Eltern- an die Kindkomponente weitergegeben wird. Das Kind kann mit diesen Daten die eigene UI und Funktionalität entwickeln oder wiederum der eigene Kindkomponente geben. Wichtig ist dabei, dass übergebene Informationen lediglich gelesen, nicht aber verändert werden sollen. Die Kommunikation nach außen funktioniert über Events. In der Kindkomponente wird dafür eine Callback-Methode aufgerufen und somit signalisiert, dass die Eltern denjenigen Code ausführen sollen, der als Reaktion auf die Veränderung in der Kindkomponente dient.

2.4.2. Hooks

In Version 16.8 erfolgte die Einführung von Hooks. Diesen bieten die Möglichkeit, States und andere React Funktionalitäten zu implementieren, ohne eine JavaScript Klasse deklarieren zu müssen. Dadurch vereinfachen sich Komponenten, die ehemals von der *Component*-Klasse abgeleitet wurden, um in Konstruktoren Zustände definieren zu können. Der Rückgabewert dieser funktionalen Komponenten ist die UI-Deklaration selbst [17].
Es gibt unterschiedliche Arten von Hooks, die gängigsten sind der *useState*- und der *useEffect*-Hook. Sie werden erstmalig direkt nach dem Rendern der Komponente ausgeführt. Außerdem gibt es die Möglichkeit, eigene Hooks zu definieren.
Die useState-Hook dient zur Deklaration eines lokalen States. In Zeile 4 des folgenden Beispiels 2.3 wird der State *counter* in der Komponente Example initialisiert. Dieser erhält den Standardwert 0 und verfügt über einen Getter – hier genannt *counter* – und den Setter,

genannt *setCounter()*, über diese Funktion der Zustand geändert werden kann.

```
1 import React, { useState } from "react";
2
3 const Example = (props) => {
4     const [counter, setCounter] = useState(0);
5 }
```

Listing 2.3: Beispiel der Nutzung von useState

Der useEffect-Hook ersetzt die Lifecycle-Methoden *componentDidUpdate*, *componentDidMount* und *componentWillUnmount* der klassenbasierten Komponenten. Es bietet sich deshalb an, Ressourcenanfragen hier zu behandeln. Das Beispiel in 2.4 zeigt einen einfachen Effect welcher die frühere *componentDidMount*-Funktion und *componentWillUnmount()*-Funktion ersetzt. Der erste Übergabeparameter des useEffect-Aufrufs ist eine Funktion, die ausgeführt werden soll und der Zweite ein Array, genannt *dependency array*. Wenn das Array leer ist, bedeutet das, dass der Effect nur einmal nach dem Rendern der Komponente ausgeführt werden soll. Durch jedes Element, das diesem Array hinzugefügt wird, startet erneut diejenige Funktion, die als erster Übergabeparameter übergeben wurde.

```
1 import React, { useEffect } from "react";
2
3 const Example = (props) => {
4     const [counter, setCounter] = useState(0);
5
6     useEffect(() => {
7         console.log('Component did render!);
8 ␣␣␣}, ␣[]);
9
10 ␣␣␣useEffect(()␣=>␣{
11 ␣␣␣␣␣␣␣␣console.log('Counter was updated!);
12     }, [counter]);
13
14     // setCounter() is called somewhere in the component
15 }
```

Listing 2.4: Beispiel für Nutzung der useEffect-Hook

Durch das *dependency array* haben Effects den Vorteil, dass sie sehr fallspezifisch auf Änderungen der Daten reagieren können und somit umso mehr den dynamischen Gedanken der Single Page Applikation realisieren.

2.5. React Native

React Native ist eines der meist genutzten Frameworks zur Entwicklung von Native Apps [2]. Es basiert auf React und ist ebenfalls von Facebook entwickelt und veröffentlicht worden. Bekannte Apps, die auf React Native basieren, sind selbstverständlich die Facebook und Instagram App, aber auch die Unterkunftbuchungsapp Airbnb oder die Lieferdienstapp UberEats [18].
Wie bereits im Kapitel Grundlagen erklärt, werden native Applikationen üblicherweise mit der dafür vorgesehenen Programmiersprache implementiert. React Native nutzt die Möglichkeit, mit JavaScript Schnittstellen und Views von Native Anwendungen anzusprechen und das Ganze somit als native Code zu rendern. Zur Laufzeit werden dabei aus den *Core Components* von React Native die jeweils korrespondierenden *Native Components* von Android, iOS und anderen Betriebssystemen erstellt[19]. Das wird ermöglicht durch die Architektur von React Native, die in drei Komponenten unterteilt ist: eine Laufzeitumgebung für JavaScript auf der Zielplattform, eine Bridge und ein Native Module. Ersteres bietet in iOS JavaScriptCore und in Android eine Lösung, die von React Native zur Verfügung gestellt wird. Dabei gibt es in React Native plattformübergreifende Abstraktionen wie *View*, *Image* oder *NetInfo*, die korrespondierende Native Komponenten besitzen. Die Bridge ist für die Kommunikation zwischen Native Module und JS-Umgebung verantwortlich. Sie erhält von ersterem Events wie Toucheingaben oder Netzwerkanfragen und vom Native Module Anweisungen für das UI oder APIs. Der Austausch findet dabei asynchron über JSON[6] statt. Der letzte Teil der Architektur ist das Native Module. Dies ist die Anwendung, die der Nutzer verwendet [20]. Es handelt sich somit bei Apps, welche mit React Native programmiert wurden, nicht um hybride Apps, da sie nicht in einem Native Container gerendert, sondern tatsächlich in nativen Code umgewandelt werden. Die Entwicklung solcher Apps ist dennoch plattformunabhängig, da die Implementierung in JavaScript erfolgt und die Umwandlung erst zur Laufzeit geschieht. React Native Applikationen lassen sich dadurch schwierig in die bereits erklärten Kategorien von mobilen Anwendungen einordnen.
Zur Nutzung von React Native Apps benötigt das Gerät mindestens iOS 11.0 oder Android 5.0 [17]. Genau wie andere Native Apps können sie in den plattform-spezifischen App Store veröffentlicht und dort vom Nutzer heruntergeladen, installiert und aktualisiert werden. Dabei ist der Signing-Prozess derselbe wie bei Native Applikationen. Zur Implementierung von Komponenten und Funktionalitäten in React Native gibt es mehrere Optionen. Die erste Option ist das Nutzen der bereits erwähnten Core Components. Weiterhin kann auf Community Lösungen zugegriffen werden. Sollten diese nicht ausreichen, ist dies meist die effizienteste Wahl, da viele Funktionalitäten bereits zufriedenstellend und stabil entwickelt und veröffentlicht wurden. Unter `https://reactnative.directory/` sind diese auffindbar und durch einen errechneten Directory Score nach mehreren Kriterien wie Beliebtheitskur-

[6] JavaScript Object Notation, ein programmiersprachen-unabhängiges Datenformat

ve, Sterne auf GitHub oder Anzahl der Downloads bewertet. Die letzte Option ist es, Native Modules selbstständig zu programmieren.

Kapitel 3.

Kriterienkatalog zum Vergleich der Anwendungen

Nachdem die Grundlagen erläutert wurden, soll nun auf die Kriterien eingegangen werden, nach denen die Anwendungen im Verlauf der Arbeit verglichen werden. Durch eine Literaturrecherche hat sich ergeben, dass dies die elementaren Bausteine einer erfolgreichen Anwendung sind.

3.1. Funktionalität

Applikationen werden entwickelt, um Nutzern einen Mehrwert in ihrem Leben zu bieten. Jeder mehr Funktionalität eine Anwendung unterstützt, desto mehr Nutzen ist meist gegeben. Egal ob in der Web- oder Appentwicklung, wenn die Anwendung keinen Mehrwert bietet, wird sie nicht genutzt und wird dadurch vom Markt verdrängt.

3.1.1. Installierbarkeit und Aktualisierungen

Damit Apps, die der Nutzer oft verwendet, leichter erreichbar sind, wird es bevorzugt diese auf dem Gerät zu installieren. Falls nun Aktualisierungen des Herstellers verfügbar sind, möchte der Nutzer diese auch erhalten und durchführen können. Für das verwendete Gerät bedeutet das Installieren, dass es Teile seines Speichers der Applikation zur Verfügung stellen muss.

3.1.2. Offline-Betrieb

Eine wichtige Funktion von mobilen Anwendungen ist der Offline-Betrieb. Das bedeutet, dass die Anwendung auch ohne oder unter schlechter Internetverbindung verwenden werden kann. Dabei muss natürlich unterschieden werden zwischen Anwendungen, die generell keinen Zugriff auf das Internet benötigen und jenen, die ihre Funktionalität im Offline-Betrieb einschränken müssen. Meistens bieten Applikationen eine Mischung aus beiden Optionen an. Um aus technischer Sicht eine Unabhängigkeit von der Netzwerkverbindung zu schaffen,

müssen Daten lokal im Speicher des Geräts abgelegt werden. Das ermöglicht dem Nutzer seine Daten jederzeit verfügbar zu haben und somit unabhängig von äußeren Umständen zu sein. Vorteilhaft für den Nutzer ist dabei auch, wenn die Prozesse, die im Offline-Betrieb angestoßen wurden, gehalten werden, bis das Gerät wieder eine stabile Internetverbindung hat. Auf diesem Wege wird gewährleistet, dass jegliche Aktivitäten erfolgreich durchgeführt werden. Nachteil des Abspeichern der Daten ist zum einen erhöhten Speicherverbrauch und bei einer großen Anzahl auch eine Steigerung des Batterieverbrauchs.

3.1.3. Standortzugriff

Um mobile Anwendungen auf die eigenen Bedürfnisse anzupassen, ist der Standortzugriff eine Option. Dabei greift die Anwendung durch das Global Positioning System (GPS) auf den Standort des Nutzers zu und kann diese weiterverarbeiten, um standortabhängige Informationen darzustellen. Der Zugriff bezieht sie dabei auf den aktuellen Standort sowie auf Bewegungen des Nutzers. Meist muss bereits während des Installationsprozesses der Applikation die Zustimmung des Nutzers für das Nutzen von standortbezogenen Inhalten eingeholt werden. Wird dies genehmigt, ist es der Anwendung auch möglich im Hintergrund auf GPS-Daten zuzugreifen. Außerdem sind diese Daten sensibel.
Gängige Use Cases sind die Abfrage des Standorts für Wetterinformationen, Navigation oder Anzeige von Dienstleistungen in der Nähe.

3.1.4. Benachrichtigung

Neben dem Standortzugriff spielen auch Benachrichtigungen eine große Rolle bei der Interaktion mit Nutzern. Diese lassen sich aus technischer Sicht in zwei Kategorien unterteilen: persistent und nicht-persistente Benachrichtigungen. Sie unterscheiden sich darin, dass erstere nur erscheinen, wenn die Anwendung in dem Moment in Benutzung ist, wohingegen das Empfangen von persistenten Benachrichtigungen jederzeit erfolgen kann. Sie bieten den Vorteil, dass die Nutzer immer wieder auf die Anwendung aufmerksam werden und ihn somit motiviert diese öfters zu nutzen. Wichtig ist hierbei die Zahl der Benachrichtigungen pro Tag oder Woche zu planen. Denn wenn der Nutzer zu viele Nachrichten erhält, kann das als störend empfunden werden und im schlimmsten Fall zur Deaktivierung der Benachrichtigungen oder Deinstallation der Anwendung führen. Push Benachrichtigung können kurze Neuigkeiten durch Texte, Bilder oder Buttons beinhalten. Letzteres ist besonders wichtig, da der Nutzer darüber aus dem Menü heraus mit der Anwendung interagieren und auch auf diese weitergeleitet werden kann.

3.1.5. User Idle Detection oder Kontaktzugriff oder Geofencing oder !?

3.2. Plattformunabhängigkeit und Kompatibilität

Durch die große Auswahl von Betriebssystemen und Geräten ist es aufwendig Anwendungen zu implementieren, die allen Ansprüchen genügen. Unternehmen und Entwickler müssen deshalb im Voraus genau abwägen, auf welchen Systeme die Apps später verfügbar sein sollen.

3.3. Entwicklungsaufwand

Bei der nativen Entwicklung müssen etwaige Änderungen des Konzepts und der Logik der App in die Implementierung der Android und iOS App umgesetzt werden. In Webanwendungen muss in der Regel auf die Darstellung in verschiedenen Browsern geachtet werden. Gerade wenn schnell reagiert werden soll – zum Beispiel in der COVID-19-Krise – ist es praktisch, wenn der Einarbeitungsgrad angemessen und die Entwicklungsaufwand günstig ist, um effektiv und schnell Menschen zu unterstützen. Dabei sind auch Faktoren wie die Komplexität der Anwendung, das Design und die Anzahl der Funktionalitäten ausschlaggebend. , jedoch lässt sich durch Betrachtung der durchschnittlichen Gehälter einer App- oder Webentwicklers eine ungefähr Kostensumme feststellen. Ein Senior Entwickler in Nordamerika verdient durchschnittlich 101-120 die Stunde, während ein App Entwickler xxx verdient.

Kapitel 4.

Implementierung

Um die zwei Ansätze zur Implementierung einer Applikation zu testen, wurde eine App zur Darstellung der aktuellen COVID-19 Fallzahlen programmiert. Diese bezieht ihre Daten von dem offiziellen API des Robert-Koch-Instituts, die täglich aktualisiert werden Beide Applikationen sind mit dem Quelltext-Editor Visual Studio Code und weitestgehend mit JavaScript programmiert.

4.1. Progressive Web App mit React

Offline-Betrieb Zur Ermöglichung des Offline Betriebs der PWA wird ein Service Worker implementiert. Dieser kann Netzwerkabfragen durch das Abfangen des Fetch-Vorgangs im lokalen Speicher zwischenspeichern. Der Ansatz, der hierfür ausgewählt wurde ist Der "Cache then networkAnsatz. Das bedeutet, dass beim Ausführen der Anwendungen der Service Worker zuerst prüft, ob PWA: Offline Betrieb wird durch Service Worker ermöglicht. Fetch Event sollte abgefangen und gecached werden, sodass die Seite auch ohne Internet arbeitet, im Notfall (wenn nichts gecached wird) sollte trotzdem eine individuelle Seite angezeigt werden (um Nutzer eine besser / engere Erfahrung zu ermöglichen). Installierbarkeit

Standortzugriff Navigator.getCurrentPosition um Koordinaten zu bekommen, Google Geolocation / Geocaching API um aus Koordinaten den LK rauszubekommen || Freie API zum Reverse Geocaching, Permission für das Abfragen des Standorts. Funktioniert bei Apple Safari und Chrome, Android Chrome Navigator window objekt erklären Benachrichtigungen Um das Ganze einfach zu halten, Wie bereits erwähnt k Funktioniert nicht bei iOS ?! Firebase Cloud Messaging[21] Hier wird der BaaS Firebase genutzt, der von Google zur schnellen Implementation eines Backendsys-tems zur Verfügung gestellt wird. Speziell wurde das Firebase Clould Messaging (FCM) und die Real Time Database genutzt. Über dessen Konsole können einmalige sowie regelmäßige Benachrichtigungen an alle registrieren Nutzer versendet werden. Fettes Bild und Konzept erklären - Offline Funktionalität https://web.dev/offline-cookbook/ - Lokation: API Abfrage und navigator.geolocation -

PWA: Installierbarkeit wird auch durch Service Worker ermöglicht. Zusätzliche Anforderungen sind HTTPS und ein App Manifest. Bei Chrome gibt es nach öfter aufrufen der Seite einen Prompt, der fragt, ob die App installiert werden soll. Bei Safari / Chrome muss das beforein-stallprompt implementiert werden. (Nur Safari?) Installieren kann man die PWA direkt aus dem Browser.

4.2. Native App mit React Native

Kapitel 5.

Ergebnisse und Diskussion

Die Evaluierung der beiden Entwicklungsansätzen erfolgt über den bereits erläuterten Kriterienkatalog. Die Ergebnisse der Implementierung sind durch das Testen beider Anwendungen auf einem Apple IPhone 12 und einem Samsung S10 entstanden.

• Wars möglich alles umzusetzen? • Gab es irgendwo irgendwelche Beschränkungen? • Nochmal Kriterienkatalog dafür, ob alle Funktionaltäten passen • Tabelle für Vergleich insgesamt ? Keine Bewertungen, wenn bei Funktionalität bei beiden alles geht • Unterschiede zwischen Implementierungen • Vielleicht noch was reinbringen, was bei pwas nicht funktioniert • Was funktioniert und was nicht bei PWAs und RN?

5.1. Funktionalität

In beidem gut umsetzbar. Bei der React Native App konnten die geforderten Funktionalitäten sowohl mit Expo als auch mit reinem RN einfach implementiert werden. Der plattformspezifische Code bedarf jedoch wiederum genauer Auseinandersetzung mit der jeweiligen Programmierung der Funktionalitäten.

Installierbarkeit Offline-Betrieb Standortzugriff Benachrichtigungen

5.2. Plattformunabhängigkeit

Hier das Testing anbringen? PWA - iOS: 1. Öffnen der PWA im Safari Browser. 2. Auf das Teilen Icon klicken 3. „Zum Home-Bildschirm“. getestet auf einem Iphone 12, iOS 14.4. Iphone mit Chrome funktioniert es nicht. - Android: 1. Öffnen der PWA im Chrome Browser 2. Auf 3 Punkte (Einstellungen) klicken 3. „App installieren“ 4. Im Prompt auf Installieren klicken“ 1. Öffnen der PWA im „eingebauten“ Browser 2. Prompt kommt || „Hinzufügen zu. . . “ Aus-wahlmöglichkeit zwischen Lesezeichen, Browser-Startseite und Telefon Startbildschirm || In-formationen über Website - Getestet auf einem Huawei P10 Lite, Android 10. Nach dem Installieren ist die PWA durch den Home Screen erreichbar. Jedoch gibt es keine

au-tomatischen Aktualisierungen der Seite, falls eine neue Version veröffentlicht wird (?). RN - Eigentlich allgemein Plattformunabhängig - Durch ejecten von expo bleibt es immer noch plattformunabhängig, aber es lassen sich auch jeweils (im gleichen Projekt!) android oder ios spezifische Implementierungen vornehmen

Der besondere Vorteile bei PWA ist, dass sie im Gegensatz zu nativen Anwendungen, selbst wenn diese drauf ausgelegt sind plattformunabhängig zu sein, dennoch unbeschränkteren Zugriff anbieten.

5.3. Entwicklungsaufwand

Um die beiden Ansätze besser vergleichen zu können, wurde einerseits die Dauer der Einarbeitung und Recherchen gemessen und andererseits die reine Implementierungsdauer. Die Anwendungen wurden mit einem soliden Grundwissen in HTML, CSS und JavaScript entwickelt. Spezifische Kenntnisse über die Frameworks waren nicht gegeben. In der folgenden Grafik wird die Auflistung der Zeit dargestellt: PWA: Funktionalität Dauer - Recherche Dauer - Implementierung Grundlegendes Setup 1 Stunde 25min Installierbarkeit 2 Stunden 30min Standortabfrage 1 Stunde 15min Benachrichtigung 5 Stunden 120min Gesamt 9 Stunden 190min

RN: Funktionalität Dauer - Recherche Dauer - Implementierung Grundlegendes Setup 4 Stunde 60min Installierbarkeit Standortabfrage Benachrichtigung Hierbei wird deutlich, dass für die Einfindung und Implementierung der React Native App im Gegensatz zu React PWA deutlich mehr Zeit beansprucht wurde, obwohl Teile des Codes wiederverwendet werden konnten. Besonders ausschlaggebend war dabei die Einarbeitung in die Technologie, vor allem als es um das Programmieren von plattformspezifischem Code ging. Bei Betracht der im Kriterienkatalog dargelegten Gehälter eines Entwicklers, kann man ableiten, dass es für die Anforderungen an diese Anwendung kostengünstiger ist eine PWA zu entwickeln.

5.4. Vorteile von PWAs gegenüber Native Apps und Grenzen, Diskussion

https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Introduction - Schnelles installieren, weniger Speicherverbrauch - Aktualisierungen müssen nicht extra installiert werden aka die ganze Applikation muss nicht neu heruntergeladen werden - Deep Linking auf bestimmte Seiten (ist das auch möglich in normalen Apps?) - Bestehende Webseiten / Webapplikationen können mit wenigen Schritten in PWAs umgewan-delt werden (Installierbarkeit und Offline Modus) Der Vorteil der Installation einer PWA gegenüber die einer Native App ist, dass sie nutzerfreundlicher ist. Bei Native App wird der Nutzer muss

der Nutzer überzeugt werden, dass ihm die App den Speicherplatz wert ist, auch wenn er die Anwendung eventuell nicht oft nutzt. Bei PWAs ist der Vorteil, dass sie ein-fach und schnell bei vermehrter Nutzung der App installiert werden können und weniger Bandweite verbrauchen [6]. Zusammenfassend lässt sich sagen, das PWAs immer wichtiger werdende Konkurrenten für Native Apps sind, denn mittlerweile fehlen nur noch ein paar Funktionalitäten, um in Zukunft komplett auf Native Apps verzichten zu können. Dennoch sind die fehlenden Ausstattungsmerkmale ausschlaggebend, denn Apps ohne beispielsweise Zugriff auf Kontakte oder die Funktion des Geofencings, sind in vielen Fällen nicht konkurrenzfähig. Durch die Schnelligkeit und einfach Umsetzung einer Webb Anwendung mit er-weiterter Grundfunktionalität ist eine Progressive Web App eine solide Wahl, um schnell ans Ziel zu kommen. Dadurch dass React PWAs und React Native Apps in ihrer Grundlage auf dem gleichen Prinzip beruhen, nämlich React, ist es durch React Native möglich, eine bereits bestehende React PWA durch Übertragung in React Native zu einer Native App umzuprogrammieren, die alle Funktionalitäten einer Native App. Dies ist besonders vorteilhaft, da die Entwicklung nicht von Anfang an gemacht werden muss und sich somit viel Zeit sparen lässt. Wenn man lediglich bei dem „bare“ Workflow von React Native bleibt, hat man schnell eine Anwendung, die meist ohne weitere Konfiguration sowohl auf Android und iOS läuft. Soll die App nun noch plattformspezifische Elemente beinhalten, kann sogar durch das Verwerfen der Expo CLI komplett nativer Code entwickelt werden. Diese schrittweise Erweiterung der Möglichkeiten bietet dem Team eine enorme Flexibilität und optimiert den Entwicklungsaufwand.

5.4.1. Grenzen von Progressive Web Apps

Obwohl es mittlerweile viele Funktionalitäten gibt, die PWAs umsetzen können, gibt es immer noch einige Einschränkungen. Unter der Seite https://whatwebcando.today/ ist übersichtlich dargestellt, um welche konkreten Features es sich dabei handelt. Im Bereich „Native Behaviors“ fehlt nach Ansicht des Autors nur noch die „User Idle Detection“. Gravierend ist jedoch, dass bislang nicht auf die Kontakte oder SMS / MMS zugegriffen werden kann. Hierfür ist bereits ein inoffizieller Entwurfsvorschlag bei der W3C eingegangen [22]. Weitere fehlende Funktionalitäten sind das Geofencing, Shape Detection bei Kameraaufnahmen und NFC Übertragung.
Wichtig ist dabei, dass diese Features nicht immer in allen Browsern gleichermaßen verfügbar sind. Beispielsweise ist die automatische Aufforderung zur Installation der PWA, bezeichnet als "BeforeInstallPromptEvent API", bisher nur Android Geräten und den meisten Chromium-basierten Browsern vorenthalten []. Weitere Beispiele mit unterschiedlichen Browser Support sind Background Sync, Bluetooth und Push Benachrichtigungen. Vorteil ist hier, dass das Versprechen, dass bei Crossplatform Anwendungen genannt wird "Code once, run everywhere"gehalten wird. Einbüßungen sind dabei aber die teilweise fehlenden

Funktionalitäten. Aktuell sind aber Entwürfe für x, x und x beim W3C eingereicht, weswegen in Zukunft mehr Funktionalitäten auch im Web zur Verfügung stehen und somit wirklich nur in JavaScript entwickelt werden muss.

5.4.2. Grenzen von Cross-platform Apps

eigentlich nicht immer so einfach der write once run everywhere ansatz, da oft trotzdem code extra geschrieben werden muss für bestimmte betriebssysteme If you hire a JavaScript developer to work on your React Native project, expect that they will have to write native code to bridge any gaps in functionality.

Kapitel 6.

Fazit und Ausblick

Klärung der wissenschaftlichen Frage, Bezug auf Einleitung, Future Work Kritische Hinterfragung: Moderne Webanwendungen haben mittlerweile viele der Funktionen inbegriffen und es bedarf deshalb keine zusätzlichen Entwicklungsaufwand. Der Begriff PWA ist deshalb eher als Buzzword zu sehen, als der Gedanke des „nativen“ Webs entstanden ist, bezeichnet aber mittlerweile „nur“ Webanwendungen. Für anknüpfende Arbeiten könnten die Analyse von PWAs und React Native Apps intensiviert werden, indem genauer auf die verschiedenen Funktionalitäten eingegangen wird, die mit diesen beiden Vorge-hensweisen umsetzbar sind. Beispiele hierfür für die Weiterentwicklung bei der Webanwendung wären die Einbindung der Background Sync API, Payment Request API oder Web Share Target API.

Anhang A.

Supplemental Information

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literaturverzeichnis

[1] W. Jobe, "Native apps vs. mobile web apps," *International Journal of Interactive Mobile Technologies (iJIM)*, vol. 7, no. 4, p. 27, 2013.

[2] o. V., "Mobile operating system market share worldwide: Apr 2010 - apr 2021," 2021.

[3] M. Schickler, M. Reichert, and R. Pryss, *Entwicklung mobiler Apps: Konzepte, Anwendungsbausteine und Werkzeuge im Business und E-Health.* eXamen.press, Berlin: Springer Vieweg, 2015.

[4] o. V., "Mobile and desktop," 2020.

[5] MDN contributors, "Progressive web apps (pwas)."

[6] M. Gaunt, "Service workers: an introduction."

[7] MDN contributors, "Service worker api."

[8] M. Wenzel, G. Warren, Y. Victor, P. Marcano, S. Ghosh, D. Pine, and S. Smith, "Choose between traditional web apps and single page apps (spas)," 2020.

[9] S. Richard and P. LePage, "What makes a good progressive web app?."

[10] A. Russell, J. Song, J. Archibald, and M. Kruisselbring, "Service workers nightly," 2021.

[11] MDN contributors, "Web app manifest."

[12] C. Liebel, *Progressive Web Apps: Das Praxisbuch.* Rheinwerk Computing, Bonn: Rheinwerk Verlag, 1. auflage ed., 2019.

[13] M. Lamouri, M. Cáceres, and J. Yasskin, "Permissions," 2020.

[14] R. Barger, "Is react a library or a framework? here's why it matters," 2021.

[15] Facebook, "React without jsx."

[16] Facebook, "Who's using react native."

[17] Facebook, "Introducing hooks."

[18] Facebook, "React native."

[19] Facebook, "Core components and native components."

[20] E. Behrends, *React Native: Native Apps parallel für Android und iOS entwickeln.* Heidelberg: O'Reilly, 1. auflage ed., 2018.

[21] S. Hyun, J. Cho, G. Cho, and H. Kim, "Design and analysis of push notification-based malware on android," *Security and Communication Networks*, vol. 2018, pp. 1–12, 2018.

[22] P. Beverloo and Rayan Kanso, "Contact picker api," 2021.

Glossar

library A suite of reusable code inside of a programming language for software development. i

shell Terminal of a Linux/Unix system for entering commands. i