Masterarbeit

Progressive Web Apps als Alternative zu Webapplikationen und Ersatz zu nativen Apps

von
Nicolas Sander
nicolas_sander@gmx.de
6659939

und
Eugen Ackermann
eugenack@mail.uni-paderborn.de
6682159

vorgelegt bei
Prof. Dr. Gerd Szwillus
Universität Paderborn
Mensch-Computer-Interaktion, Softwaretechnologie
(Erstprüfer)

und
Prof. Dr. Eric Bodden
Universität Paderborn
Softwaretechnik
(Zweitprüfer)

29. März 2018

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Paderborn, 29. März 2018

(Nicolas Sander)

(Eugen Ackermann)

Aufteilung

Die folgende Auflistung gibt an, welcher Teil der Arbeit von wem erstellt wurde.

Abschnitt der Arbeit	**Bearbeiter**
Abstract	Eugen Ackermann und Nicolas Sander
Kapitel 1	Eugen Ackermann und Nicolas Sander
Kapitel 2 Einführung	Eugen Ackermann und Nicolas Sander
Kapitel 2.1	Eugen Ackermann und Nicolas Sander
Kapitel 2.2.1	Nicolas Sander
Kapitel 2.2.2	Nicolas Sander
Kapitel 2.2.3	Eugen Ackermann und Nicolas Sander
Kapitel 2.2.4	Nicolas Sander
Kapitel 3	Eugen Ackermann
Kapitel 4 Einführung	Eugen Ackermann und Nicolas Sander
Kapitel 4.1	Nicolas Sander
Kapitel 4.2.1	Eugen Ackermann
Kapitel 4.2.2	Eugen Ackermann und Nicolas Sander
Kapitel 4.3	Nicolas Sander
Kapitel 4.4	Nicolas Sander
Kapitel 4.5	Eugen Ackermann und Nicolas Sander
Kapitel 5	Eugen Ackermann und Nicolas Sander
Programmcode	Eugen Ackermann
Programmcode Frontend Optimierung	Nicolas Sander
Dokumentarbeiten (LaTeX)	Eugen Ackermann und Nicolas Sander
Dokumente CD	Eugen Ackermann

Abstract - deutsch

Die vorliegende Masterarbeit beschäftigt sich mit Progressiven Web Apps, einer neuen Form der Web App, die auch einige bisher nur von Nativen Apps bekannte Eigenschaften unterstützt. Dabei wurde insbesondere untersucht, ob Progressive Web Apps für Nutzer und Unternehmen einen Mehrwert gegenüber Web Apps und Nativen Apps bieten können. Für den Vergleich der verschiedenen Entwicklungsansätze wurde zum einen eine Literaturrecherche durchgeführt. Zum anderen wurden ein für diese Arbeit entwickeltes Newsportal und ein Videoportal zu einer Progressive Web App migriert. Hierdurch kann ein Einblick in den benötigten Aufwand und mögliche auftretende Schwierigkeiten der Entwicklung gewährt werden. Auf Grundlage dieser Untersuchungen wird deutlich, dass Progressive Web Apps eine gute Erweiterung der Web Apps sind. Als Ersatz zu Nativen Apps fungieren die Progressive Web Apps jedoch noch nicht, da diese derzeit nur auf Android Geräten vollständig nutzbar sind. Mit steigender Unterstützung seitens Apple haben die Progressiven Web Apps jedoch das Potential, in Zukunft einen Teil der Nativen Apps abzulösen.

Schlagworte: Progressive Web Apps, PWA, Web Technologien, Push Benachrichtigungen, Service Worker, HTML5, Plattformübergreifende Entwicklung, Browserunterstützung, Web App, Native App, Softwareentwicklung für mobile Endgeräte

Abstract - english

This master thesis deals with progressive web apps, an advanced form of web app that also support some of the features that are only known from native apps. In particular, it will examine whether progressive web apps can offer users and companies additional value in comparison to standard web apps and native apps. To define and compare the different development approaches, on the one hand, a literature search is conducted. On the other hand, a news portal that has been designed for this work and an additional video portal will be migrated to a progressive web app in order to evaluate the effort and difficulties that arise during the development. Based on this, it can be seen that progressive web apps are a good extension of standard web apps. However, the progressive web apps are currently not working as a replacement for native apps, as they are actually only available on Android devices. With growing support from Apple, however, progressive web apps have the potential to replace some native apps in the future.

Keywords: progressive web apps, pwa, web technologies, push messages, service worker, HTML5, cross-platform development, browser support, web app, native app, software development for mobile devices

Inhaltsverzeichnis

Abbildungsverzeichnis iv

Tabellenverzeichnis vi

Quellcodeverzeichnis viii

Abkürzungsverzeichnis ix

1. Einführung 1

1.1. Problemstellung 1

1.2. Zielsetzung 3

1.3. Methodik 3

1.4. Aufbau der Arbeit 4

2. Stand der Technik und Beschreibung von Progressiven Web Apps 5

2.1. Übersicht und Beschreibung bisher genutzter Entwicklungskonzepte . 5

2.1.1. Native Apps 5

2.1.2. Cross Plattform Apps (Generated Apps) 8

2.1.3. Web Apps 9

2.1.4. Hybrid Apps 10

2.2. Progressive Web Apps 12

2.2.1. Allgemeine Beschreibung und Features 12

2.2.2. Technischer Hintergrund 16

2.2.3. Browser- und Betriebssystemunterstützung 25

2.2.4. PWA Beispiele 35

3. Konzeption der Anwendungsfälle 40

3.1. Entwicklung eines Nachrichtenportals 42

3.1.1. Aufwand 44

3.1.2. Software-Architektur 45

3.1.3. Datenbankschema 48

3.2. Entwicklung eines Videoportals 49

3.2.1. Aufwand 50

3.2.2. Software-Architektur 51

3.2.3. Datenbankschema 52

3.3. Migration der entwickelten Web Apps zu Progressiven Web Apps . . 52

3.3.1. Installation im Startbildschirm 53

3.3.2. Offline Support 55

3.3.3. Push Benachrichtigungen 58

3.3.4. HTTPS Unterstützung 60

3.3.5. Protokollierung des Aufwands 60

3.3.6. Änderungen der Architektur 61

4. Evaluierung 62

4.1. Vergleich zwischen Web App, Nativer App und PWA auf Basis der verwendeten Literatur 63

4.1.1. Funktionalität 63

4.1.2. Sicherheit 70

4.1.3. Entwicklung 74

4.1.4. Energieverbrauch 78

4.1.5. Performance und Usability 79

4.1.6. Monetarisierung 83

4.2. Evaluation der Entwicklung der konzipierten Anwendungsfälle 88

4.2.1. Aufwand und Schwierigkeiten 88

4.2.2. Hilfreiche Programme und Werkzeuge 91

4.3. Vergleich zwischen Web App und PWA auf Basis der Anwendungsfälle 94

4.3.1. Offline Funktionalität 94

4.3.2. Natives Verhalten 99
4.3.3. Performance 104
4.4. Verbesserungen durch PWA Entwicklung aus der Praxis 108
4.5. Ergebniszusammenfassung für Unternehmen und Nutzer 111
4.5.1. Unternehmen 111
4.5.2. Nutzer 114

5. Fazit und Ausblick **116**
5.1. Fazit 116
5.2. Ausblick 117

Literaturverzeichnis **118**

A. Ladezeiten aller Testdurchläufe **134**

B. Mockups des Videoportals **137**

C. Quellcodauszüge des Videoportals **140**

D. Änderungen an der Softwarearchitektur **143**

Abbildungsverzeichnis

2.1. Übersicht mobile Betriebssysteme 6
2.2. Übersicht der PWA Eigenschaften 14
2.3. Service Worker Funktionsweise 20
2.4. App Shell Beispiel 22
2.5. App Manifest Beispiel 23

3.1. Google Trends ausgewählter PHP-Frameworks im Jahre 2017 42
3.2. Mockup der Startseite des Nachrichtenportals 43
3.3. Mockup der Detailseite einer Nachricht im Nachrichtenportal 44
3.4. Lebenszyklus einer HTTP Anfrage 46
3.5. Software-Architektur des Nachrichtenportals 48
3.6. ER-Diagramm des Nachrichtenportals 49
3.7. ER-Diagramm des Videoportals 52

4.1. Installationsschritte PWA und Native App 82
4.2. Newsportal Lighthouse PWA Überprüfung 92
4.3. Ergebnis Test 1: Offline Funktionalität erstmaliger Seitenaufruf 95
4.4. Ergebnis Test 2: Offline Funktionalität erneuter Seitenaufruf 97
4.5. Ergebnis Test 3: Offline Funktionalität Videos 98
4.6. Ergebnis Test 4: Natives Verhalten des Anzeigebereichs 100
4.7. Ergebnis Test 5: Hinzufügen Homescreenverknüpfung 101
4.8. Ergebnis Test 6: Entfernen Homescreenverknüpfung 102
4.9. Ergebnis Test 7: Benachrichtigungen 104

B.1. Mockup der Startseite des Videoportals 137
B.2. Mockup der Detailseite eines Videos im Videoportal 138
B.3. Software-Architektur des Videoportals 139

D.1. ER-Diagramm des PWA Nachrichtenportals 143
D.2. Softwarearchitektur des PWA Nachrichtenportals 144

Tabellenverzeichnis

2.1. Features und Technologie Übersicht 17
2.2. Browserunterstützung: Natives Verhalten 28
2.3. Browserunterstützung: Umgebung 29
2.4. Browserunterstützung: Gerätefunktionen 29
2.5. Browserunterstützung: Mikrofon & Kamera 30
2.6. Browserunterstützung: Betriebssystem 31
2.7. Browserunterstützung: Eingabe 32
2.8. Browserunterstützung: Nahtlose Erfahrung 32
2.9. Browserunterstützung: Standort & Position 33
2.10. Browserunterstützung: Bildschirm und Ausgabe 34

3.1. Aufwandsprotokoll des Nachrichtenportals 45
3.2. Aufwandsprotokoll des Videoportals 51
3.3. Aufwandsprotokoll der Migration 61

4.1. Vergleich: Funktionalität 64
4.2. Vergleich: Sicherheit 70
4.3. Vergleich: Entwicklung 75
4.4. Vergleich: Monetarisierung 84
4.5. Service Worker Funktionalität im PWABuilder 93
4.6. Vorgehen Test 1: Offline Funktionalität erstmaliger Seitenaufruf . . . 94
4.7. Vorgehen Test 2: Offline Funktionalität erneuter Seitenaufruf 96
4.8. Vorgehen Test 3: Offline Funktionalität Videos 98
4.9. Vorgehen Test 4: Natives Verhalten des Anzeigebereichs 99
4.10. Vorgehen Test 5: Hinzufügen Homescreenverknüpfung 101
4.11. Vorgehen Test 6: Entfernen Homescreenverknüpfung 102
4.12. Vorgehen Test 7: Benachrichtigungen 103

4.13. Ergebnis Test 8: Ladezeit erstmaliger Seitenaufruf 105
4.14. Ergebnis Test 9: Ladezeit erneuter Seitenaufruf 107
4.15. PWA Verbesserungen aus der Praxis 109

A.1. Newsportal Web App erstmaliger Seitenaufruf 134
A.2. Newsportal PWA erstmaliger Seitenaufruf 134
A.3. Videoportal Web App erstmaliger Seitenaufruf 135
A.4. Videoportal PWA erstmaliger Seitenaufruf 135
A.5. Newsportal Web App erneuter Seitenaufruf 135
A.6. Newsportal PWA erneuter Seitenaufruf 136
A.7. Videoportal Web App erneuter Seitenaufruf 136
A.8. Videoportal PWA erneuter Seitenaufruf 136

Quellcodeverzeichnis

3.1. Manifest.json aus der VideoPortal PWA 54
3.2. HTML-Auszug um eine manifest.json Datei zu laden 55
3.3. Auszug aus der webpack.mix.js Datei des PWA Videoportals 56
3.4. Beispielimplementierung des Laufzeitcachings 57
3.5. Beispiel-Code zur Registrierung eines Service Workers 58
3.6. Beispiel eines Subscription-Objektes 59

C.1. Service-worker.js aus dem PWA Videoportal 141
C.2. Auszug aus dem Service Worker des Videoportals zur Erstellung eines Subscribe-Objektes 142

Abkürzungsverzeichnis

API	Application programming interface
B2B	Business to Business
CEO	Chief Executive Officer
CSS	Cascading Style Sheets
GPS	Global Positioning System
HTTPS	Hypertext Transfer Protocol Secure
NFC	Near Field Communication
NP	Newsportal
OS	Operating System
PWA	Progressive Web App
SW	Service Worker
UI	User Interface
URL	Uniform resource locator
VP	Videoportal

1. Einführung

1.1. Problemstellung

Die steigende Zahl von Smartphone Nutzern und die steigende Intensität der Nutzung von mobilen Endgeräten sorgt für einen stetig wachsenden Datenverbrauch des mobilen Datennetzwerkes. Durch die Nutzung von mobilen Endgeräten wie Smartphones, Tablets oder Laptops können Nutzer immer mehr Dinge von unterwegs erledigen. Die steigende Belastung des Datennetzes, mangelnde Empfangsqualität aufgrund schlecht ausgebauter Netze oder begrenzte Datenvolumen der Tarife dämpfen die Attraktivität des mobilen Internets und verhindern stellenweise den Einsatz des mobilen Endgerätes in bestimmten Situationen. Aus diesem Grund kann es sinnvoll sein, Internetseiten mit einem möglichst geringen Datenverkehr zu gestalten und Informationen durch neue Technologien offline zur Verfügung zu stellen.
Ebenso mangelt es Websites an nützlichen Features wie Push Benachrichtigungen oder den Gebrauch der Kamera aus dem aktiven Fenster, die dem Nutzer einen Mehrwert bieten. Bisher waren diese Features und Vorteile nur den Nativen Apps zugänglich, weshalb der Fokus bei vielen Unternehmen auf der Entwicklung dieser Apps lag. Allerdings ist die Entwicklung von Nativen Apps aufgrund der verschiedenen mobilen Betriebssysteme sehr aufwendig, da meist für jedes Betriebssystem ein komplettes Entwicklungsteam engagiert werden muss. Zwar gibt es einige Wrapper-Frameworks, die es ermöglichen, Apps in nur einer Programmiersprache zu entwickeln, jedoch nicht immer zur vollständigen Zufriedenheit der Unternehmen. Dies liegt daran, dass in der Regel trotzdem spezielle Änderungen für die jeweiligen Betriebssysteme nötig sind oder die erstellten Apps nicht performant genug sind.
Eine Abhilfe der oben genannten Probleme sollen sogenannte Progressive Web Apps (PWAs) schaffen. Diese sind zwar in erster Linie auch nur über den Browser aufgerufene Websites, bieten aber darüber hinaus weitere Features. Mittels Services

Workers, die mit Javascript realisiert werden, können Daten im Hintergrund im Cache gespeichert werden. Dadurch ist eine teilweise Nutzung der Webanwendung ohne bestehende Internetverbindung möglich, was für normale Webseiten undenkbar ist. Außerdem sollen Progressive Web Apps um einiges schneller sein als bisherige Websites und Web Apps. Im Normalfall klickt der Nutzer auf einen Button und die neue Website wird komplett geladen. In diesem Prozess gibt es für eine gewisse Zeit keine Reaktion von der Website, da sie serverseitig gestaltet und erst dann übermittelt wird. Entsprechend der Internetverbindung kann der Nutzer erst zum Schluss eine Reaktion wahrnehmen. In diesem Punkt möchten PWAs ebenfalls entgegenwirken, denn hier wird die Webseite nicht auf einmal geladen, sondern progressiv gestaltet und an die jeweilige zur Verfügung stehende Browserumgebung angepasst. Hierfür sprechen einige Vorteile. Erstens hat der Nutzer von Beginn an eine Reaktion und zweitens ist das User Interface dadurch flüssiger zu bedienen. Damit versprechen sich Unternehmen eine geringere Bouncerate und eine verbesserte Conversion-Rate. Bei den PWAs werden nicht nur die bisherigen Features von Websites verbessert. Wie zuvor erwähnt, ist es auch möglich Features zu nutzen, die bisher nur Nativen Apps zugänglich waren. Laut Google können PWAs Gerätefunktionen, wie z.B. Push Benachrichtigungen, Standortbestimmungen, die Kamera sowie andere Gerätesensoren nutzen. Durch diese Leistungen können nun auch Webentwickler Apps für jedes mobile Betriebssystem entwickeln, was wiederum eine beachtliche Kosten- und Zeitersparnis für die Unternehmen bedeuten könnte. Unter anderem aus diesen Gründen sieht Google großes Potential in der Verbesserung des Webs durch Progressive Web Apps, welche Webanwendungen die Möglichkeiten einer Nativen App bieten. Wenn ein Big Player wie Google so viel Zeit und Energie in die Entwicklung und Vermarktung von Progressiven Web Apps steckt, ist dies ein Grund, Progressive Web Apps genauer zu analysieren, um sich ein eigenes Bild dieser neuen Technologie zu machen.

1.2. Zielsetzung

Der Begriff Progressive Web App ist ein relativ neues Thema im Web. Aus diesem Grund gibt es noch wenige wissenschaftliche Untersuchungen in diesem Bereich. Da es sehr viele positive Versprechungen gibt, das Web mit dieser Technologie besser zu gestalten und einige bekannte Unternehmen wie Twitter schon eine PWA entwickelt haben, ist das Ziel dieser Arbeit, herauszufinden, ob Progressive Web Apps wirklich einen Mehrwert gegenüber Webapplikationen und Nativen Apps bieten können. Hierfür werden die Progressiven Web Apps zunächst genau bezüglich der Features und technischen Voraussetzungen beschrieben. Danach folgt eine Betrachtung verschiedener Perspektiven und ein Vergleich mit bisher angewandten Entwicklungskonzepten im mobilen Bereich. Zum einen sollen Änderungen für den Nutzer analysiert werden, wie z.B. die Nutzungsqualität. Hierzu gehören unter anderem schnelle Ladezeiten, ein gutes Offline Verhalten und eine flüssige Bedienung. Zum anderen soll auch die Sicht der Unternehmen und Entwickler betrachtet werden. In diesem Fall vor allem der Entwicklungsaufwand und die Monitarisierung der Anwendung. Mit Hilfe zweier Anwendungsfälle und der vorhandenen Literatur soll der Aufwand der Erstellung einer Progressiven Web App, bzw. der Migration von bestehenden Websites zu einer Progressiven Web App aufgezeigt werden. Hierdurch soll verdeutlicht werden, ob sich der Aufwand für die Unternehmen lohnen kann.

1.3. Methodik

Um im Evaluationkapitel dieser Arbeit die Progressiven Web Apps mit Web Apps und Nativen Apps genauer vergleichen zu können, werden zuerst die bisher im mobilen Bereich verwendeten Entwicklungskonzepte für ein besseres Verständnis kurz beschrieben und voneinander abgegrenzt. Anschließend werden die Progressiven Web Apps ausführlich dargestellt und die möglichen Features erläutert, da dies für das Verständnis der gesamten Arbeit wichtig ist. Als Literatur dieser Arbeit werden zu einem großen Teil auch Online Quellen und Ausarbeitungen der Industrie hinzugezogen, da es im Bereich der Progressiven Web Apps zum Zeitpunkt der Aus-

arbeitung dieser Arbeit nur sehr wenige wissenschaftliche Publikationen gibt. Auf diesen Umstand weisen auch Biørn-Hansen, Majchrzak und Grønli (2017) hin. Da in der Praxis die meisten Unternehmen schon eine Webpräsenz vorweisen können, wird für die Analyse der Entwicklung nicht die komplette Neuentwicklung einer PWA untersucht, sondern die Migration von zwei klassischer Websites bzw. Web Apps zu einer PWA. Hierfür werden zwei Web Apps als Ausgangsbasis implementiert und zu einer PWA migriert. Um die Vor- und Nachteile einer PWA an diesen Beispielen genauer untersuchen zu können, werden unterschiedliche Eigenschaften, wie z.B. der Datenverbrauch der Anwendung, abgebildet.

1.4. Aufbau der Arbeit

Die vorliegende Arbeit ist in fünf Kapitel gegliedert. Nach einer Hinführung zum Thema in Kapitel 1, folgt in Kapitel 2 eine Beschreibung der bisher im mobilen Bereich üblicherweise genutzten Entwicklungskonzepte. Im Anschluss daran werden die Progressiven Web Apps, welche einen neuen Ansatz darstellen, in Bezug auf vorhandene Features, technische Hintergründe und Beispiele aus der Praxis vorgestellt. Im darauffolgenden Kapitel 3 werden die für diese Arbeit konzipierten Anwendungsfälle aufgeführt und der Programmcode beschrieben. Diese wurden entwickelt, um sie anschließend zu einer Progressive Web App zu migrieren. Dabei wird vor allem der Entwicklungsaufwand protokolliert und auftretende Schwierigkeiten erläutert. Die Evaluation des Entwicklungs- und Migrationsaufwands folgt in Kapitel 4. Des Weiteren werden zuvor die Vor- und Nachteile der verschiedenen Entwicklungsansätze in verschiedenen Bereichen verglichen und bewertet. Insbesondere wird evaluiert, ob die Progressiven Web Apps aus Sicht der Nutzer und Unternehmen als Alternative zu Web Apps bestehen können und die Nativen Apps auf Dauer ersetzen können. Im letzten Kapitel 5 wird ein Fazit gezogen und es findet ein kurzer Ausblick auf mögliche zukünftige Änderungen und Entwicklungen im Bereich der Progressiven Web Apps statt.

2. Stand der Technik und Beschreibung von Progressiven Web Apps

In diesem Kapitel werden zunächst in Abschnitt 2.1 die bisher üblicherweise im mobilen Bereich eingesetzten Entwicklungsansätze beschrieben. Dazu gehören Native Apps, Hybrid Apps, Cross Plattform Apps und Web Apps. Anschließend folgt in Abschnitt 2.2 eine Erläuterung der Progressiven Web Apps. Die Betrachtung und der Einsatz der Technologie der Progressiven Web Apps ist, wie der Titel dieser Masterarbeit andeutet, Hauptbestandteil der Arbeit. Daher findet in diesem Kapitel eine genaue Beschreibung der Funktionsweise und der Eigenschaften dieses neuen Entwicklungsansatzes statt. Des Weiteren wird aufgeführt, inwieweit die Funktionen von PWAs bzw. der HTML5 APIs auf Android und iOS in der Kombination mit den Browsern Google Chrome, Microsoft Edge, Mozilla Firefox und Apple Safari unterstützt werden. Der Ansatz der Hybrid Apps und der ähnlichen Cross Plattform Apps wird beschrieben, um die Progressiven Web Apps besser einordnen zu können. Bei der späteren Evaluation der Entwicklungsansätze werden die Hybrid Apps und Cross Plattform Apps nicht weiter betrachtet. Ein Vergleich findet nur zwischen den Nativen App, Web Apps und Progressiven Web Apps statt.

2.1. Übersicht und Beschreibung bisher genutzter Entwicklungskonzepte

2.1.1. Native Apps

Native Apps sind Applikationen auf mobilen Endgeräten, die laut Majchrzak, Biørn-Hansen und Grønli (2018) mit Hilfe eines Software Development Kit (SDK) des jeweiligen Anbieters der mobilen Plattform entwickelt werden. In der Regel werden

Native Apps daher über einen App Store wie z.B. den Google Play oder den Apple App Store installiert. Somit werden diese „stets für eine bestimmte Plattform entwickelt und entsprechen üblicherweise auch den Konventionen der Benutzeroberfläche dieser Plattform“ (Nielsen und Budiu, 2013, S. 58). IBM (2012) beschreibt, dass eine Installation alternativ auch ohne App Store durchgeführt werden kann. Dabei erfolgt die Installation direkt über die Installationsdatei. Hierbei können allerdings Sicherheitsrisiken bestehen. Bei Android beispielsweise muss in den Einstellungen des Systems der Installation aus unsicheren Quellen zugestimmt werden. Neben der Installation werden bei Nativen Apps auch die App Updates über den App Store des jeweiligen Betriebssystems veröffentlicht und bezogen. Aufgrund der verschiedenen Plattformen bzw. Betriebssysteme werden Native Apps in unterschiedlichen Programmiersprachen geschrieben und müssen für jedes System individuell entwickelt werden. „In order to create true, native applications, the Java programming language must be used for Android, the Objective C programming language for iOS, and the .NET framework for Windows Phone“ (Jobe, 2013). Ergänzend zu der Java Programmiersprache wird nach Shafirov (2017) auf Android seit 2017 auch die Programmiersprache Kotlin unterstützt. Für iOS Systeme wurde laut Apple (o.J.) die Programmiersprache Swift hinzugefügt. Abbildung 2.1 gibt einen Überblick über die verschiedenen Plattformen und den dazugehörenden Entwicklungseigenschaften.

	Apple iOS	Android	Blackberry OS	Windows Phone
Languages	Objective-C, C, C++	Java (some C, C++)	Java	C#, VB.NET and more
Tools	Xcode	Android SDK	BB Java Eclipse Plug-in	Visual Studio, Windows Phone development tools
Packaging format	.app	.apk	.cod	.xap
App stores	Apple App Store	Google Play	Blackberry App World	Windows Phone Marketplace

Abbildung 2.1.: Übersicht der bekanntesten mobilen Betriebssysteme (IBM, 2012).

Dabei ist jedoch zu beachten, dass die Betriebssysteme Windows Phone und Blackberry OS laut Marktforschungsergebnissen von Gartner (2017) aktuell kaum noch Marktanteile besitzen. Aus diesem Grund werden im weiteren Verlauf dieser Arbeit nur die Systeme iOS und Android weiter betrachtet.

Die direkte Verbundenheit der Nativen App mit der Plattform sorgt laut Jobe (2013) dafür, dass die Applikation so die gesamte zur Verfügung stehende Hardware und alle Funktionen des mobilen Gerätes ungehindert nutzen kann. Dies sorgt für eine gute Nutzer Erfahrung in Bezug auf die nahtlose Integration der App mit dem Betriebssystem und der daraus resultierenden guten Performance. „From the end user perspective, native apps provide the richest user experience. Source code is efficient, with fast performance, consistent look and feel and full access to the underlying platform hardware and data“ (Xanthopoulos und Xinogalos, 2013). Allerdings hat diese starke Integration in das Betriebssystem auch Nachteile in Bezug auf Aufwand und Kosten der Entwicklung. So beschreibt IBM (2012), dass jedes OS z.B. andere UI Komponenten nutzt und somit sich die Entwickler mit den verschiedenen Designs der unterschiedlichen Betriebssysteme auskennen müssen, falls eine Entwicklung für mehr als eine Plattform gewünscht ist. Zu diesen UI Komponenten gehören unter anderem „[...] buttons, input fields, sliders, menus, tab bars, dialog boxes and so on“ (IBM, 2012).

Gestartet wird eine Native App direkt über ein Icon auf dem Homescreen oder App-Drawer. „Außerdem können [Native Apps] den Speicherplatz des verwendeten Geräts nutzen“ (1&1-DigitalGuide, 2017b). Durch diese Nutzung des Gerätespeichers können viele Daten und der Programmcode gespeichert werden und müssen somit nicht erst bei jedem Start aus dem Internet geladen werden. Dies ist besonders bei großen Apps wie z.b. Spielen eine wichtige Eigenschaft. Des Weiteren gibt es standardisierte Möglichkeiten, um über den App Store des jeweiligen Betriebssystems mit Nativen Apps Geld zu verdienen. Hierbei bekommt der App Store Betreiber jedoch einen gewissen Teil des Gewinns von jeder verkauften App für die Bereitstellung dieser Vertriebsplattform. Der Zwang zum App Store sorgt auch dafür, dass die App an die Richtlinien des Store Betreibers angepasst werden muss, was eventuell zu Einschränkungen oder zusätzlichen Entwicklungsaufwand führen kann. „Developers have to follow mobile app stores' [...] requirements to distribute their apps to end users“ (Joorabchi, Mesbah und Kruchten, 2013).

2.1.2. Cross Plattform Apps (Generated Apps)

„Die Disziplin der Cross-Platform-Entwicklung beschäftigt sich damit denselben Code auf verschiedenen Plattformen nutzen zu können und so dieselbe App in den verschiedenen Stores anbieten zu können“ (Feiner, Krajnc und Niederl, 2015). Xanthopoulos und Xinogalos (2013) beschreiben, dass hierdurch Zeit und Aufwand eingespart werden kann. Im Kontext dieser Arbeit werden mit dem Begriff der Cross Plattform Apps die Anwendungen bezeichnet, die mit Hilfe von nativer Code Generierung entwickelt werden. In der Literatur können auch die Hybrid Apps unter diesen Begriff fallen, die hier aber gesondert in Kapitel 2.1.4 betrachtet werden. Das Vorgehen der Code Generierung grenzt sich durch den nativen Code von den im späteren Verlauf beschriebenen Hybrid Apps, Web Apps und PWAs ab, da diese die Cross Plattform Nutzung durch den Einsatz von Webtechnologien ermöglichen. Die Grundidee ist, „[...] dass der von den Entwicklern erzeugte gemeinsame Code fur jede Plattform als Input für das Generierungstool dient, dieses Tool den Code in nativen Code umwandelt, der von der jeweiligen Runtime der Plattform verstanden wird“ (Feiner et. al., 2015). Hierdurch wirkt die App wie eine extra für jede Plattform entwickelte Native App. Jedoch weisen Feiner et. al. (2015) darauf hin, dass dieser Ansatz sehr generisch ist und somit einige Nachteile im Vergleich zu richtigen Nativen Apps enstehen. So ist der Code durch die automatische Erzeugung nicht optimal und aufgeblasen. „Fast immer wirkt sich das auf die Große [sic] der App aus“ (Feiner et. al., 2015). Des Weiteren merken Xanthopoulos und Xinogalos (2013) an, dass durch die automatisch generierte Codestruktur in der Praxis die Wartung und Benutzung des Codes schwieriger ist. Außerdem ist die Funktionalität der App abhängig von den unterstützten Funktionen des genutzten Code Generators. Neben der Aufwandserleichterung und dem Einsparen von Kosten sorgt dieser Ansatz jedoch auch für Vorteile bei der Performance. Im Gegensatz zu den auf Webtechnologien basierenden Cross Plattform Entwicklungen erreichen die nativ generierten Apps laut Xanthopoulos und Xinogalos (2013) eine gute Performance, ähnlich der eigens für jede Plattform entwickelten Nativen Apps. Das Niveau von Nativen Apps werden die generierten Apps durch den nicht optimalen Code jedoch nicht erreichen. Allerdings schreitet die Entwicklung der Code Generatoren weiter vorran und

die Wahl des richtigen Code Generators ist entscheident, um ein gutes Ergebnis zu erzielen. Angulo und Ferre (2014) haben vier verschiedene Code Generatoren als mögliche Kandidaten für ihre Studie über Cross Plattform Frameworks untersucht und nur ein Tool konnte die benötigten Eigenschaften für die Studie erfüllen. Für die Auswahl des richtigen Frameworks muss also zuvor etwas Aufwand erbracht werden.

2.1.3. Web Apps

Majchrzak et. al. (2018) beschreiben, dass es sich bei dem Begriff der Web Apps um einen nicht eindeutig definierten Begriff handelt. Innerhalb dieser Arbeit wird mit dem Begriff der Web Apps ein Entwicklungskonzept bzw. eine Anwendungsart für mobile Geräte bezeichnet, die mit Hilfe von Webtechnologien wie HTML, CSS und JavaScript realisiert wird. Web Apps werden im Browser ausgeführt, sind genauso wie Websites über eine URL erreichbar und basieren auf einem Client Server Modell. Im 1&1-DigitalGuide (2017b) ist beschrieben, dass die Web App hierbei von einem Webserver geladen wird und eine Installation somit nicht erforderlich ist. Dies ist ein Grund, wieso eine Web App nahezu plattformunabhängig ist. Web Apps „[...] do not require adaptation to any operating environment, they are platform-independent, and they are easy and quick to launch“ (Delia, Galdamez, Thomas, Corbalan und Pesado, 2015). Zur Nutzung einer Web App auf den unterschiedlichsten Betriebssystemen wird also lediglich ein aktueller Browser benötigt. Diese Tatsache sorgt auch für eine plattformunabhänige Entwicklung und somit für eine Reduktion des Aufwands und der Kosten. Allerdings hat der Bezug zum Browser auch zur Folge, dass nicht alle Funktionen des Endgerätes genutzt werden können. „Also, security restrictions imposed by the execution of the code through a browser limit the access that these applications have to all the features offered by the device“ (Delia et. al., 2015). Auch Nielsen und Budiu (2013) beschreiben, dass einige Handyfunktionen nur eingeschränkt genutzt werden können. „Ihr Look-and-Feel entspricht dem der nativen Apps, allerdings nutzen sie die Funktionalität des Handys nur in vergleichsweise eingeschränktem Maße (beispielsweise unterstützen sie nicht so viele Fingergesten)“ (Nielsen und Budiu, 2013, S. 58). Außerdem beschreibt

Jobe (2013), dass die User Experience unter der eingeschränkten Integration in das Betriebssystem leidet. Allerdings merken Xanthopoulos und Xinogalos (2013) an, dass durch einige Software Bibliotheken und die HTML5 Unterstützung der Browser die Hardware Unterstützung steigt. „Moreover, possibilities and performance have been improved“ (Majchrzak et. al., 2018). Die Browserabhängigkeit sorgt laut Majchrzak et. al. (2018) jedoch auch dafür, dass es bei unterschiedlichen Browsern zu verschiedenen Darstellungen der Web App kommen kann. Ein Vorteil ist jedoch, dass durch die genutzten Webtechnologien neue Versionen der Web App dem Nutzer direkt zur Verfügung gestellt werden können. Ein Update über einen App Store oder Ähnlichem entfällt hierdurch. Allerdings ist für diese noch nicht zwischengespeicherten Daten und Programmcode eine Internetverbindung nötig, um die Web App nutzen zu können. Auch gibt es laut Jobe (2013) keine universelle Strategie, um mit einer Web App oder einem darüber angebotenen Service Profit zu machen. Um die Vermarktung der Web App bzw. der darüber angebotenen Services muss sich der Betreiber selbst kümmern. Ein anderes Problem ist die Abhängigkeit der Funktionalität von JavaScript, das unter bestimmten Bedingungen von einigen Nutzern deaktiviert wurde. „Also, disabling JavaScript is a possibility in most modern browsers; Thus, effectively disabling the logic layer and often the user interface layer of modern Webapps results in apps rendering blank pages“ (Majchrzak et. al., 2018). Dies ist aber ein allgemeines Problem von Entwicklungskonzepten, die Javascript nutzen.

2.1.4. Hybrid Apps

Hybrid Apps sind, wie der Name schon verrät, eine Kombination aus Nativen Apps und Webinhalten. Delia et. al. (2015) und Xanthopoulos und Xinogalos (2013) beschreiben, dass Hybrid Apps, ähnlich wie Web Apps, Technologien wie HTML, Javascript und CSS verwenden. „Hybrid apps are primarily built using HTML5 and JavaScript, and a detailed knowledge of the target platform is not required“ (Xanthopoulos und Xinogalos, 2013). Der Unterschied zu Web Apps liegt darin, dass die Hybrid Apps nicht direkt über einen Browser aufgerufen werden, sondern auf dem

Gerät wie eine Native App über einen App Store installiert und über diese Native App gestartet werden. Innerhalb der Nativen App können dann die Webtechnologien genutzt werden. Laut Delia et. al. (2015) wird hierfür ein Container genutzt, wodurch der Webcode eine native Umgebung bekommt. Diese Kombination aus Nativen Eigenschaften und Web Eigenschaften ermöglicht laut Hird (2011) und Delia et. al. (2015) den Zugriff auf die Hardware des Gerätes, obwohl Webtechnologien eingesetzt werden. „Like web apps, the source code is still executed by a browser that is part of the final application and can be packaged with the application“ (Xanthopoulos und Xinogalos, 2013). Die Entwicklung einer Hybrid App verschiebt sich durch diese Vorgehensweise in Richtung plattformunabhängige Entwicklung. Hierdurch lässt sich Entwicklungsaufwand einsparen, da viele Inhalte nur einmal geschaffen werden müssen, unabhängig vom genutzten Betriebssystem. Rickard (2016) beschreibt, dass bis zu 80% des Codes so plattformübergreifend genutzt werden kann. Der restliche Code muss laut Feiner et. al. (2015) jedoch trotzdem noch für jedes Betriebssystem nativ entwickelt werden, „[...] um auf [...] Geräte-Spezifika zugreifen zu können“ (Feiner et. al., 2015). Feiner et. al. (2015) fügen weiter hinzu, dass hierfür Cross-Platform-Frameworks Abhilfe schaffen können. Somit kann eine Hybrid App als Einstieg in den App Store betrachtet werden. „Viele Unternehmen nutzen hybride Apps als Container für bereits existierende Websites, in der Hoffnung, auf diese Art [‚ohne übermäßigen Aufwand,] eine Präsenz im App Store zu realisieren“ (Nielsen und Budiu, 2013, S. 58). Des Weiteren weist Rickard (2016) darauf hin, dass Updates der nativen Anwendungen über den App Store erfolgen, Updates des Inhalts können bei den genutzten Webtechnologien jedoch auch ohne den Weg über den App Store durchgeführt werden. Jedoch beschreiben Delia et. al. (2015), dass durch nicht Nutzen von nativen Komponenten des Interfaces die Nutzer Experience beeinträchtigt werden kann. Außerdem kann die Performance unter der Web Container Lösung leiden.

2.2. Progressive Web Apps

Im diesem Abschnitt wird die Technologie der Progressiven Web Apps beschrieben. Dazu gehören eine allgemeine Beschreibung, die möglichen Features und Eigenschaften, Erläuterung zur Funktionsweise einiger Teile dieses Entwicklungskonzeptes und eine Auflistung der nötigen Voraussetzungen, um alle Funktionalitäten nutzen zu können. Am Ende dieses Kapitels werden PWA Beispiele aufgeführt und es wird erläutert, inwiefern hier der Einsatz dieser Technologie für das Unternehmen und dessen Kunden einen positiven Effekt erzeugt. Eingeordnet werden PWAs bei der Verteilung zwischen Web Anwendungen und nativen Anwendungen neben den zuvor beschriebenen Web Apps und Hybrid Apps.

2.2.1. Allgemeine Beschreibung und Features

Progressive Web Apps sind laut Griffith (2017) das erste Mal im Jahr 2015 von Google vorgestellt worden. Danach hat sich der Begriff der Progressiven Web Apps schnell verbreitet und zu einem starken Schlagwort im Jahr 2016 und 2017 entwickelt. Wolber (2018) beschreibt, dass es keine Statistiken zur Verteilung von PWAs gibt. Die Menge der verfügbaren PWAs scheint jedoch aktuell noch überschaubar zu sein. Deutlich wird dies dadurch, dass beim alltäglichen Surfen im Web nur bei sehr wenigen Websites der Hinweis auf die Installationsmöglichkeit der Web App auf den Homescreen des Smartphones auftaucht. „One way to discover a PWA is to visit a site that works as a PWA. You may see a notification that you can add the app to your Home screen“ (Wolber, 2018). Jedoch beschäftigen sich immer mehr Entwickler und Unternehmen mit dieser Thematik und stellen ihre eigene Website auf eine PWA um. Verdeutlicht werden kann dies mit Hilfe von Outweb[1] und pwa.rocks[2]. Immer mehr Progressive Web Apps werden dort gelistet und die Portale selber sind auch eine PWA. Außerdem steigt das Interesse an dem Begriff der Progressiven Web Apps laut Google-Trends (2018) im zeitlichen Verlauf in den Google Suchanfragen seit der Vorstellung des Begriffs Ende November 2015. Des Weiteren

[1] https://outweb.io
[2] https://pwa.rocks

beschreibt d'Heureuse (2017), dass immer mehr Entwickler und Unternehmen Interesse an PWAs zeigen, da die Entwickler oft bereits schon Kenntnisse in den bei PWAs genutzten Technologien wie CSS, HTML und JavaScript haben. Somit lassen sich mit bekannten Technologien und einem bereits vorhandenen Entwicklerteam unmittelbare Verbesserungen der User Experience, vor allem im mobilen Bereich, erzielen.

Allgemein betrachtet sind PWAs Webauftritte, die auf den Web Apps aufbauen und diese um native Verhaltensweisen und Inhalte erweitern. Dies bedeutet, dass zum Beispiel teilweise auf die Gerätehardware zugegriffen werden kann und Push Benachrichtigungen verschickt werden können. Laut Gustafson (2017a) soll eine PWA für den Nutzer eine App-Like Experience bieten. Das Web rutscht hierdurch ein Stück weiter in Richtung Funktionalität der Nativen Apps. „[...] regular Web sites can to a larger extent than before act, feel and look as any other installed app“ (Majchrzak et. al., 2018). Dies geschieht vor allem durch das Nutzen von neuen HTML5 APIs. Da diese auch den Web Apps zur Verfügung stehen, folgt in dieser Arbeit eine Abgrenzung. Neue HTML5 Apis werden den Progressiven Web Apps zugeordnet.
PWAs werden im Vergleich zu Nativen Apps nicht über einen App Store, sondern über einen Webbrowser zur Verfügung gestellt. Dabei passt sich der Funktionsumfang an die Kompatibilität des genutzten Browser und Betriebssystem an. „A PWA is a Website running in a browser that will progressively add more features based on compatibility“ (Firtman, 2016, S. 15). Somit ist die PWA in einem geringeren Funktionsumfang auch auf Betriebssystemen und Browsern verfügbar, die die genutzten Technologien, bzw. APIs noch nicht unterstützen. In solch einem Fall verhält sich die PWA dann wie eine Web App bzw. Website.

Die Technologie, die hinter dem Begriff der Progressiven Web App steht, ist an sich nicht eine einzige neue Technologie. Der Begriff beschreibt vielmehr eine Sammlung von oft bereits bekannten Webtechnologien, die zusammen auf eine bestimmte Art und Weise genutzt werden. „On the whole, Progressive Web Apps describe a collection of technologies, design concepts, and Web APIs that work in tandem to provide

an app-like experience on the mobile web“ (Lynch, 2016). Zur genaueren Beschreibung der Eigenschaften und Möglichkeiten von PWAs, sind in Abbildung 2.2 acht Schlagworte dargestellt, die im Folgenden genauer erläutert werden.

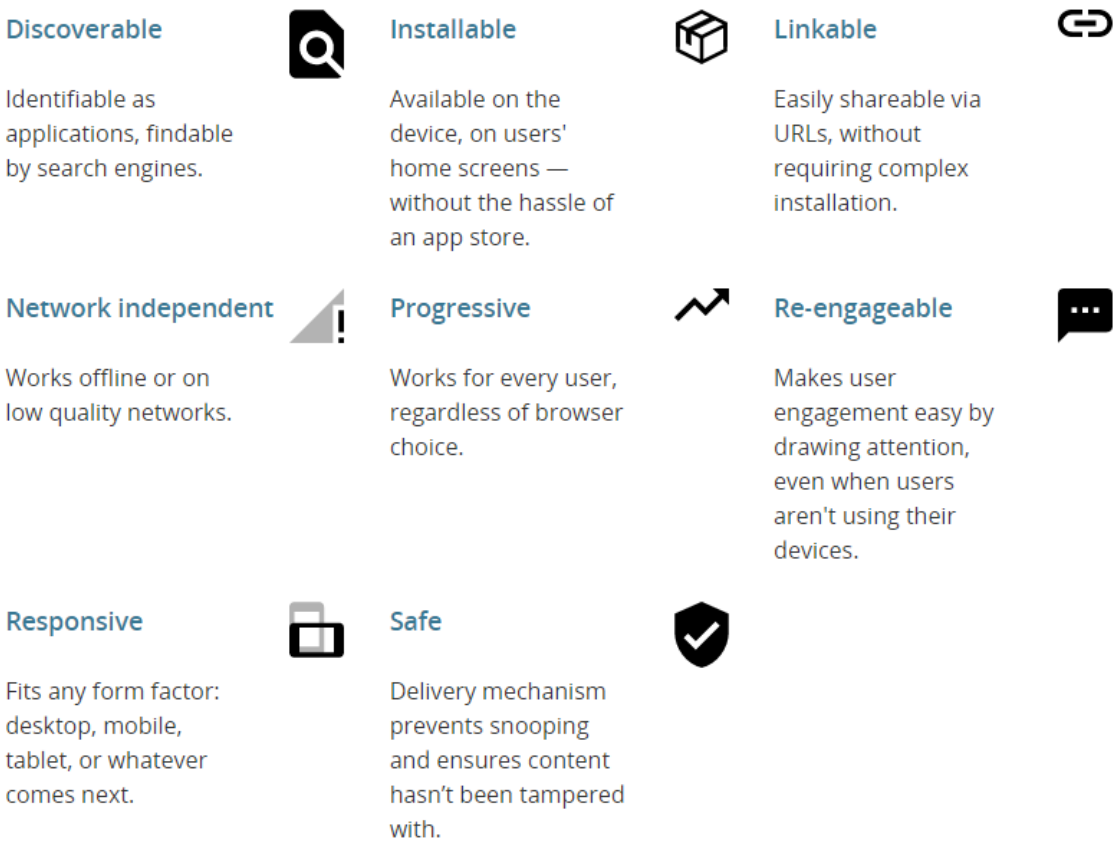


Abbildung 2.2.: Übersicht der PWA Eigenschaften (Mills und Hobson, 2016).

Eine wichtige Eigenschaft ist 'Discoverable'. Dies bedeutet, dass die PWA leicht gefunden werden kann, da die PWA eine erweiterte Form der Web App ist, die über eine URL im Browser aufgerufen werden kann. Somit wird eine PWA auch von Suchmaschinen indiziert und ist von der breiten Masse der Nutzer, z.B. über die Google Suchmaschine, auffindbar. Eine Suche und Installation der Applikation in einem App Store entfällt.
Weiter ist eine PWA 'Installable'. PWAs können bei Interesse auch als App installiert, bzw. auf dem Homescreen des Nutzers abgelegt werden. Dabei wird wie bei Nativen Apps ein Icon in der App Übersicht und im App-Drawer angelegt. Somit ist die PWA bei erneuter Verwendung leicht zugänglich, auch ohne zuvor den Browser

aufzurufen um die URL anzusteuern.
'Linkable' bedeutet, dass die Progressive Web App und der Zugang zu dieser leicht verbreitet werden kann. „Zum Teilen der App brauchen die Benutzer nur die URL, diese können sie dann direkt ihren Freunden versenden“ (d'Heureuse, 2017).
'Network independent' ist eine weitere Eigenschaft der PWA's und beschreibt, dass PWAs teilweise auch Offline funktionieren, sobald eine vorherige Nutzung stattfand und der Cache dadurch gefüllt ist. „No more blank connectivity error pages nor dinosaurs running through the desert“ (Mozilla-WADI-initiative, 2016). Auch eine Nutzung über eine geringe zur Verfügung stehende Internetverbindungen ist somit angenehmer. Dies ist vor allem nützlich in Gebieten mit einer schlechten Netzabdeckung oder bei aufgebrauchtem Datenvolumen. „A clear separation between UI and content in addition to offline caches and service workers allow you to store the application layout for future use“ (Mozilla-WADI-initiative, 2016). Eine genauere Beschreibung der Funktionsweise der Offline Funktionalität erfolgt in einem späteren Abschnitt.
Eine weitere Eigenschaft ist 'Progressive'. Das Wort Progressive steckt bereits in dem Begriff Progressive Web App und bedeutet, dass sich die PWA an die vorliegenden Umstände anpasst. „Progressive Enhancement ist der Grundsatz nach dem Progressive Web Apps gebaut werden [...]“ (Tandler, 2017). Hierbei verhindert die Nutzung eines unpassenden Browsers oder Endgerät nicht die gesamte Nutzung der PWA. Je nach nach Browser- und Betriebssystemunterstützung werden dann nicht alle Funktionen zur Verfügung gestellt. Somit kann die PWA dann ohne die neuen Funktionalitäten genutzt werden und fühlt sich dann wie eine Standard Website bzw. Web App an. „Modern web apps can be developed to provide a super cool experience to fully capable browsers, and an acceptable (although not quite as shiny) experience to less capable browsers“ (Mills, 2016).
'Re-engageable' ist ein weiterer Aspekt. Durch die Möglichkeit, Push Benachrichtigungen zu nutzen, ist es möglich, die Aufmerksamkein des Nutzer zurückzugewinnen. „Progressive Web Apps [...] can even re-engage users with web push notifications“ (Google, 2016b). Somit kann der Nutzer nun auch bei einer auf den ersten Blick normalen Website per Push über Benachrichtigungen von Updates und neuen Inhalten

informiert werden, auch wenn das Browserfenster nicht aktiv geöffnet ist oder die Seite aufgerufen ist.
Des Weiteren ist eine PWA 'Responsive'. Somit ist es egal, ob der Nutzer per Laptop, Smartphone oder Tablet auf die Seite zugreift. Die Darstellung ist je nach Eigenschaften des Endgerätes, wie z.B. die Bildschirmgröße, angepasst. „Schlussendlich sind PWAs ja nur Webseiten mit erweiterter Funktionalität und können somit mit den bestehenden responsiven Möglichkeiten entwickelt werden“ (d'Heureuse, 2017). Des Weiteren müssen PWAs über eine sichere HTTPS Verbindung laufen. Daher wird der Begriff 'Safe' verwendet. „Because a progressive web app has a more intimate user experience and because all network requests can be intercepted through service workers, it is imperative that the app be hosted over HTTPS“ (Farrugia, 2016).

2.2.2. Technischer Hintergrund

Im Folgenden wird erläutert, durch welche technischen Aspekte die wichtigsten Features der Progressiven Web Apps realisiert werden. Wie bereits zuvor erwähnt, basieren PWAs vor allem auf bereits bekannten Web Technologien. „Eine PWA basiert ausschließlich auf offenen Webstandards und wird – wie eine konventionelle Web-App – prinzipiell in den Sprachen HTML, CSS und JavaScript geschrieben“ (1&1-DigitalGuide, 2017a). Durch diese Web Technologien, besonders neue HTML5 APIs, können bestimmte technische Komponenten zur Verfügung gestellt werden. Zur Veranschaulichung, welche Komponenten für welche Features genutzt werden, bietet Tabelle 2.1 eine Übersicht der wichtigsten Eigenschaften und Features von Progressiven Web Apps und die dafür nötigen technischen Hintergründe.

> „From a technical perspective, three conditions must hold for considering a mobile web app as a PWA, namely: (i) it is served over HTTPS for security reasons, (ii) it comes with a web app manifest for declaring app metadata [...] and (iii) it uses at least a service worker“ (Malavolta et. al., 2017).

Eigenschaften und Features	**ermöglicht durch**
Keine Installation von Updates aus dem App Store nötig.	Service Worker
Keine Suche im App Store nötig.	URL Aufrufbarkeit
Über Suchmaschinen findbar.	URL Aufrufbarkeit
Auf Homescreen installierbar.	App Manifest
Gute Performance und Usability.	Service Worker und App Shell.
Kann plattformübergreifend entwickelt werden.	Webtechnologien - Browser.
Offline Funktionalität.	Service Worker
Push Benachrichtigungen.	Service Worker

Tabelle 2.1.: Technischer Hintergrund der PWA Features.

Durch den Einsatz von Service Workern ist es unter anderem möglich, Updates zur Verfügung zu stellen, ohne dass der Nutzer aktiv werden muss. Tandler (2017) beschreibt, dass durch Service Worker die Möglichkeit besteht, Inhalte automatisch im Hintergrund zu aktualisieren. Ein manuelles oder automatisches Update über einen App Store entfällt somit. Des Weiteren ist es laut d'Heureuse (2017) und Tandler (2017) durch das Aufrufen der PWA über eine URL im Browser nicht nötig, diese in einem App Store zu suchen. Stattdessen findet eine Suche genauso wie bei Web Apps über eine Suchmaschinen statt. Ein weiterer Aspekt bei PWAs ist die Installierbarkeit auf dem Homescreen des Endgerätes. Diese Eigenschaft wird nach Google (2016b) durch das App Manifest ermöglicht und sorgt laut Liebel (2017b) dafür, dass die PWA nicht mehr von einer Nativen App unterscheidbar ist und sich gleichzeitig so von normalen Webseiten unterscheidbar macht. Das Icon der PWA fügt sich zwischen den anderen App Icons auf dem Homescreen ein und ist auch im App-Drawer des Smartphones aufgeführt. Hierbei ist laut Farrugia (2016) das Web

App Manifest eine einfache JSON Datei, welche einige Einstellungsmöglichkeiten bietet, die im weiteren Verlauf dieses Kapitels genauer erläutert werden. „With it, it is possible to run the web app in full-screen mode as a standalone application, to assign an icon [...], and to assign a theme and background color to the app“ (Farrugia, 2016).

Ein weiterer technischer Aspekt ist die App Shell. So sorgen Service Worker und App Shell für eine Verbesserung der Performance und Usability. Der App Shell Ansatz sorgt laut Lynch (2016) für eine Trennung von User Interface und Content. So werden Ladezeiten beim Aufruf der App reduziert, da nicht der gesamte Inhalt bereits beim Start geladen werden muss. „When launched from the user's home screen, service workers enable a Progressive Web App to load instantly, regardless of the network state“ (Google, 2016b). Auch bei der Benutzung der App fühlt sich laut Google (2016b) alles flüssig an und die PWA reagiert schnell auf Interaktionen der Nutzer, was sich auch positiv auf die Usability auswirkt. Des Weiteren ist es laut Majchrzak et. al. (2018) durch die Nutzung von Webtechnologien möglich, PWAs plattformübergreifend zu entwickeln, da diese im Browser ausgeführt werden. Außerdem sind PWAs, wie bereits zuvor erwähnt, teilweise auch Offline nutzbar. Google (2016b) und Liebel (2017a) führen auf, dass diese Möglichkeit aufgrund der Service Worker zur Verfügung steht. „Der Service Worker kann [...] entscheiden, ob er eine Anfrage aus seinem Cache beantworten kann oder die Anfrage an das Netzwerk weiterleitet. Auf genau diese Art lässt sich die Offlinefähigkeit realisieren“ (Liebel, 2017a). Bereits vorhandene Inhalte können so ohne Internetverbindung dargestellt werden. Zusätzlich lassen sich durch die Service Worker und Push APIs auch Push Benachrichtigungen steuern. „Push is based on service workers because service workers operate in the background“ (Medley, 2018). So kann der Nutzer Meldungen erhalten, ohne dass der Browser geöffnet ist. Bisher war dies, wie bereits zuvor erwähnt, nur bei Nativen Apps möglich. „The Push API and Notification API open a whole new set of possibilities for you to re-engage with your users“ (Medley, 2018).

Im Folgenden werden die Begriffe Service Worker, App Shell und App Manifest

weiter erläutert.

Service Worker Die Service Worker sind die wohl wichtigste Änderung der PWAs im Vergleich zu den Web Apps. „Der Service Worker agiert wie ein Hintergrunddienst, der auch dann läuft, wenn die Webanwendung nicht geöffnet ist [...]“ (Liebel, 2017a). Dabei geht es laut Gaunt (2018) vor allem darum, Netzwerkanfragen zu unterbrechen und zu bearbeiten. Hierzu gehört auch die Kontrolle über einen Cache, mit dessen Hilfe Anfragen an das Netzwerk ohne Internetverbindung beantwortet werden können. Zwar konnte man auch bisher mit Hilfe des Application Cache Inhalte zwischenspeichern, jedoch ist vom W3C (2018) in der Service Worker Dokumentation und von Gaunt (2018) beschrieben, dass bisherige Cache Varianten nicht besonders gut funktionierten. „Web developers familiar with previous attempts to solve the offline problem have reported a deficit of flexibility in those solutions“ (W3C, 2018). Ohne Service Worker wären, wie zuvor beschrieben, einige Features der Progressiven Web Apps, wie z.B. Push Benachrichtigungen, nicht realisierbar. „A PWA cannot properly work in browsers without Service Worker support“ (Biørn-Hansen et. al., 2017). Somit gelten die Service Worker als wichtiger Bestandteil im Konzept der Progressiven Web Apps.

Geschrieben wird ein Service Worker in JavaScript. „According to the standard, a service worker is a special case of web worker, it is implemented in a dedicated JavaScript file and runs in a separate thread with respect to the main JavaScript thread“ (Malavolta et. al., 2017). Der Service Worker läuft somit unabhängig von der Hauptseite und die Interaktion erfolgt über Events wie z.B. fetch und push. „Once a service worker is in control, it will be in one of two states: either the service worker will be terminated to save memory, or it will handle fetch and message events that occur when a network request or message is made from your page“ (Gaunt, 2018). Dabei steht dem Service Worker neben dem Cache auch die *IndexedDB API* zur Verfügung. „If there is information that you need to persist and reuse across restarts, service workers do have access to the IndexedDB API“ (Gaunt, 2018). Somit können Daten dauerhaft gespeichert werden, damit diese beim erneuten Start

des Service Worker bzw. der PWA wieder zur Verfügung stehen, auch wenn der Nutzer Offline ist. Alternativ kann laut Farrugia (2016) dem Nutzer auch eine zuvor definierte Offlineseite bzw. die App Shell mit dem Hinweis angezeigt werden, dass derzeit keine Internetverbindung zur Verfügung steht. Die Service Worker können auf unterschiedlichste Weise eingesetzt werden und ermöglichen so verschiedene Stufen der Offlinefähigkeit der Anwendung. Diese werden in Kapitel 4.2.2 genauer dargestellt. Abbildung 2.3 verdeutlicht einen möglichen Ablauf von einem Service Worker Einsatz.

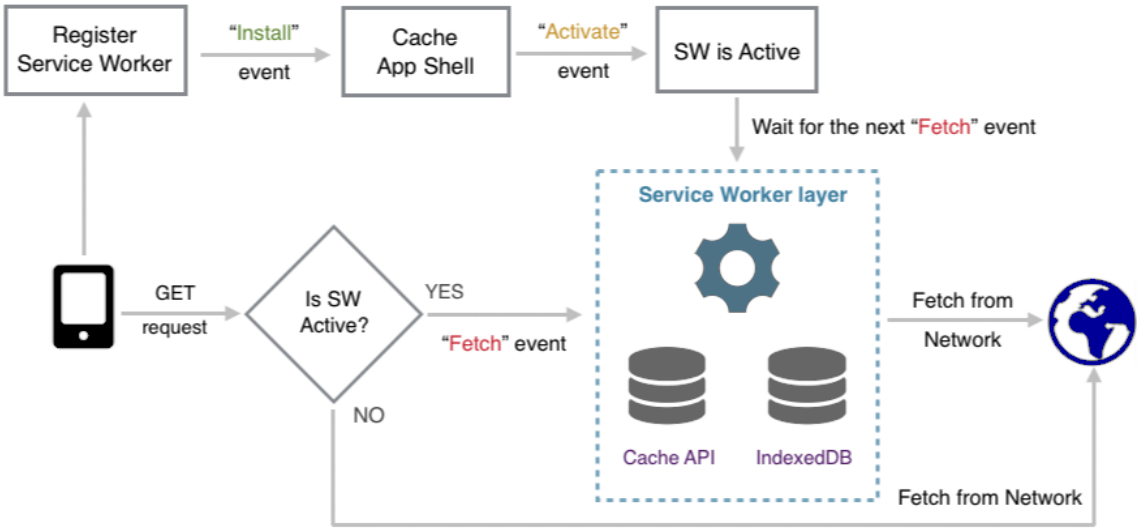


Abbildung 2.3.: Service Worker Funktionsweise bei GET-Request (Semenov, 2017).

Beim ersten Aufruf der Website wird laut 1&1-DigitalGuide (2017a) und Biørn-Hansen et. al. (2017) der Service Worker vom PWA-Server geladen und installiert. Gaunt (2018) beschreibt, dass dies aber nur bei Websites mit HTTPS funktioniert. „Nach der erfolgreichen Installation ist der Service Worker bei jedem Neustart der App einsatzbereit und wird über jegliche Netzwerkabfrage auf der betreffenden Domain informiert“ (1&1-DigitalGuide, 2017a). Sobald vom Nutzer ein *request* geschickt wird, wird geprüft, ob ein Service Worker aktiv ist. Ist dies der Fall, so wird ein *fetch* Event an den Service Worker geleitet. Andernfalls wird der *request* direkt an das Netzwerk geleitet. „Findet der Service Worker einen passenden Eintrag in seinem Cache für den angegebenen Request, beantwortet er die Anfrage, ohne

sie tatsächlich über das Netzwerk zu senden“ (Liebel, 2017a). Eine aktive Internetverbindung ist in dieser Situation nicht nötig. Falls im Cache des aktiven Service Worker jedoch keine passende Antwort verfügbar ist, wird der *request* zum Netzwerk durchgeleitet und die darauf folgende Antwort dem Nutzer übermittelt. In diesem Fall ist dann jedoch eine aktive Internetverbindung nötig.

Entwickler müssen bei der Vorgehensweise der Service Worker beachten, dass diese im Laufe der Zeit sehr oft beendet und neugestartet werden. „Developers are advised to keep in mind that service workers may be started and killed many times a second“ (W3C, 2018). Die Vorgehensweise im Programmcode wird in Kapitel 3 anhand von Codebeispielen der entwickelten Anwendungsfälle erläutert. Der Umfang der Service Worker Implemetierung ist davon abhängig, welche Offline Funktionalität erreicht werden soll.

App Shell „The App Shell model is a simple design concept whereby the initial load of a mobile web app provides a basic shell of a app UI, and the content for the app is loaded after“ (Lynch, 2016). Laut Osmani (2018) handelt es sich bei der genannten App Shell um den für die Darstellung der Hauptseite bzw. des User Interface der Applikation mindestens benötigten HTML, CSS und JavaScript Code. Als Neuheit gilt dieses Design Konzept allerdings nicht, denn Osmani (2018) beschreibt, dass auch einige moderne JavaScript Frameworks und Bibliotheken den Inhalt von der UI trennen.

Beim erstmaligen Besuch einer PWA können die statischen Ressourcen direkt im Cache des Service Workers gespeichert werden und so beim erneuten Besuch der PWA direkt aus diesem geladen werden. „Ideally, our App Shell is cached such that it loads as quickly as possible when a user visits and returns at a later date“ (Lynch, 2016). In der Abbildung 2.4 ist auf der linken Seite die App Shell durch einen roten Rahmen hervorgehoben. Hierbei wird deutlich, dass es sich bei der App Shell um die Struktur und Navigation der Anwendung handelt, die direkt beim Start der PWA

geladen werden. Auf dem rechten Screenshot ist der Inhalt der Anwendung, der erst nach der App Shell geladen wird, durch einen grünen Rahmen hervorgehoben. Durch das verzögerte Laden des Inhalts wird der Start der PWA beschleunigt, was die gefühlte Performance verbessert und sich so positiv auf die Usability auswirkt. Geeignet ist die Trennung von UI und Content vor allem für „[...] apps and sites with relatively unchanging navigation but changing content“ (Osmani, 2018).

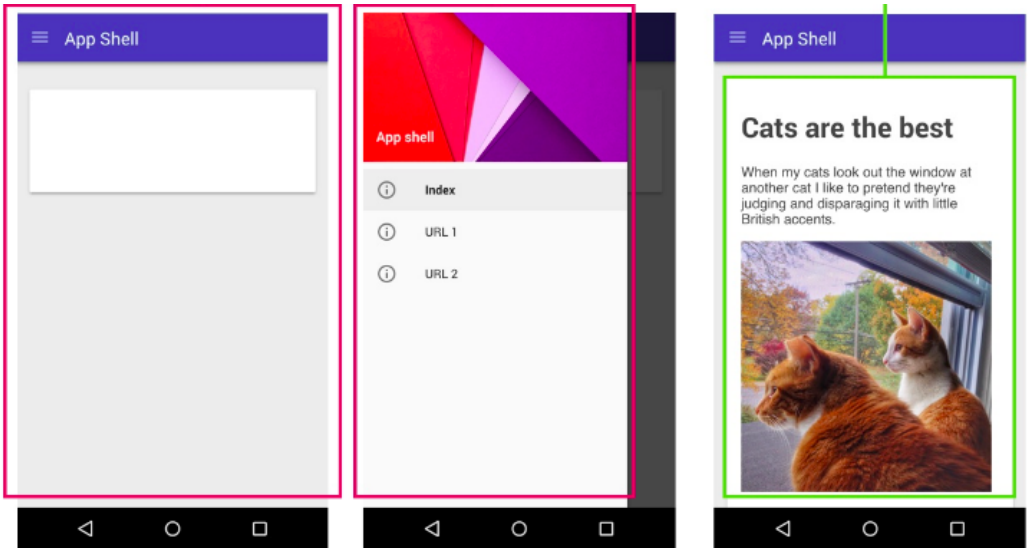


Abbildung 2.4.: Trennung von App Shell und Content (Osmani, 2018).

App Manifest Beim App Manifest handelt es sich um eine wichtige Datei, die dabei helfen kann, dass sich Progressive Web Apps mehr und mehr wie Native Apps anfühlen und verhalten.

> „The web app manifest is a simple JSON file that gives you, the developer, the ability to control how your app appears to the user in areas where they would expect to see apps (for example, a mobile device's home screen), direct what the user can launch, and define its appearance at launch“ (Gaunt und Kinlan, 2018).

Osmani (2018) beschreibt, dass der genutzte Webbrowser durch das Vorhandensein dieser Datei weiß, dass es sich bei dieser Website um eine PWA handelt und ein Icon

dieser somit auf dem Homescreen abgelegt werden kann. Liebel (2017b) erläutert, dass Browser ohne Unterstützung für solch ein App Manifest dieses einfach ignorieren, um die PWA auch ohne die durch das App Manifest ermöglichten Features nutzen zu können. Die PWA wird dann wie eine Web App bzw. Website behandelt.

Der Inhalt einer beispielhafte App Manifest Datei mit den wichtigsten Startparametern ist in der Abbildung 2.5 zu sehen.

```
{
  "lang": "en",
  "dir": "ltr",
  "name": "Donate App",
  "description": "This app helps you donate to worthy causes.",
  "short_name": "Donate",
  "icons": [{
    "src": "icon/lowres.webp",
    "sizes": "64x64",
    "type": "image/webp"
  },{
    "src": "icon/lowres.png",
    "sizes": "64x64"
  }, {
    "src": "icon/hd_hi",
    "sizes": "128x128"
  }],
  "scope": "/racer/",
  "start_url": "/racer/start.html",
  "display": "fullscreen",
  "orientation": "landscape",
  "theme_color": "aliceblue",
  "background_color": "red",
}
```

Abbildung 2.5.: Ausschnitt aus einer App Manifest Datei im JSON Format (Caceres, Christiansen, Lamouri, Kostiainen und Dolin, 2017).

Zu Beginn der Datei wird hier durch 'en' im *lang* Parameter die Sprache, hier Englisch, festgelegt. Laut Caceres et. al. (2017) bezieht sich diese Angabe auf die Sprache des App-Namens, der App-Beschreibung und des Kurznamens. Des Weiteren

wird über den *dir* Parameter zwischen 'ltr' (left-to-right text) und 'rtl' (right-to-left text) unterschieden. Eine automatische Bestimmung dieses Wertes ist durch *auto* möglich. Durch *name* wird der vollständige Name der PWA definiert und über *description* weiter erläutert. „The description member allows the developer to describe the purpose of the web application“ (Caceres et. al., 2017). Zusätzlich wird noch ein *short_name* aufgeführt, welcher bei Platzmangel anstelle des normalen Namens zum Einsatz kommt.
Unter *Icon* wird auf das Icon verwiesen, welches auf dem Homescreen des Endgerätes bei dieser PWA angezeigt werden soll. Hierbei ist es möglich, Icons für unterschiedliche Auflösungen aufzuführen, um so die Darstellung für verschiedene Displays zu optimieren. Bei dem vorliegenden Beispiel von Caceres et. al. (2017) wurden zwei Icons mit derselben Größe aufgeführt. „One is explicitly marked as WebP through the type member. If the user agent doesn't support WebP, it falls back to the second icon of the same size“ (Caceres et. al., 2017). Über den Eintrag Scope kann laut MDNwebdocs (2017) der Navigationsbereich der Webanwendung definiert werden. „Wenn der Benutzer die Anwendung außerhalb des Gültigkeitsbereichs navigiert, kehrt er zu einer normalen Webseite zurück“ (MDNwebdocs, 2017). Durch den *start-url* Parameter wird definiert, welche URL beim Start der Website bzw. PWA aufgerufen werden soll. Des Weiteren ist es über das 'fullscreen' im Punkt *display* möglich, die komplette UI des Browsers und der PWA auszublenden, damit der komplette Bildschirm für den Inhalt genutzt werden kann. Anstelle von 'fullscreen' kann auch 'standalone', 'browser' und 'minimal-ui' gewählt werden. Soll sich eine PWA wie eine Native App anfühlen, so ist 'fullscreen' und 'standalone' zu empfehlen. „In [the standalone] mode, the user agent will exclude UI elements for controlling navigation, but can include other UI elements such as a status bar“ (MDNwebdocs, 2017). So fühlt sich die PWA mehr wie eine Native App an und es fällt dem Nutzer durch das fehlende Browser Menü nicht mehr auf, dass die PWA im Browser läuft. Soll die Darstellung aber im Browser erfolgen, so wird der Parameter einfach auf 'browser' gesetzt. Außerdem sind eventuell nicht alle Modi bei allen Browsern verfügbar. „Each display mode, except browser, has a fallback display mode, which is the display mode that the user agent can try to use if it doesn't support a particular

display mode“ (Caceres et. al., 2017). Ein ähnlicher Parameter ist die Einstellung der Display Ausrichtung beim Start der PWA vom Homescreen. Diese Bestimmung der Ausrichtung beim Start ist auch von Nativen Apps bekannt. Zur Wahl stehen hier 'landscape' und 'portrait'.
Des Weiteren kann die *theme_color* festgelegt werden, „[...] welche die Plattform zum Beispiel zur Steuerung der Farbe der Statuszeile heranziehen kann“ (Liebel, 2017b). Passend dazu kann auch die *background_color* definiert werden. Hierbei beschreibt MDNwebdocs (2017), dass diese zwar bereits vorher im Stylesheet definiert wurde und somit schon vorhanden ist. Allerdings ist diese hiermit auch schon bekannt, bevor die Dokumente geladen wurden und steht somit auch schon beim Start der PWA zur Verfügung. „The background_color member is only meant to improve the user experience while a web application is loading and must not be used by the user agent as the background color when the web application's stylesheet is available“ (MDNwebdocs, 2017).

2.2.3. Browser- und Betriebssystemunterstützung

Im diesem Kapitel wird die Browser- und Betriebssystemunterstützung genauer beschrieben. Um als Anwender die verschiedenen Funktionalitäten der Progressiven Web Apps auf dem eigenen Endgerät nutzen zu können, müssen bestimmte Browser und Betriebssystem Voraussetzungen erfüllt sein. Hierzu muss insbesondere eine möglichst große HTML5 API Unterstützung gewährleistet sein. Im Folgenden wird aufgeführt, in welchem Umfang die von PWAs genutzten HTML5 APIs in den jeweiligen Browser und Betriebssystem Kombinationen zum jetzigen Zeitpunkt unterstützt werden. Nach Steiner (2018) muss jedoch beachtet werden, dass eine API Implementierung nicht immer eine Garantie für ein funktionierendes Feature bietet. „[...] the pure presence of an api object or method does not necessarily guarantee that the feature will then actually work“ (Steiner, 2018).

Grundsätzlich ist hervorzuheben, dass die Progressiven Web Apps vor allem vom Chrome Browser in Kombination mit Android unterstützt werden. Dies liegt dar-

an, dass, wie bereits in Abschnitt 2.2.1 erwähnt, Google die Technologie der PWAs entwickelt hat und diese nun verbreitet. Somit ist es naheliegend, dass der Browser aus dem eigenen Unternehmen in der Unterstützung der PWA Features führend ist. Daher wird in den nachfolgenden Kapiteln dieser Arbeit bei den Progressiven Web Apps der Chrome Browser in Kombination mit Android verwendet. Vor allem beim Vergleich mit den Nativen Apps und Web Apps wird so gewährleistet, dass diese mit dem aktuellen technischen Stand von Progressiven Web Apps verglichen werden.
Neben dem Chrome Browser unterstützen auch der Edge Browser von Microsoft und der Firefox Browser auf Android viele der aktuellen APIs. Auf Apple Systemen sieht dies zum jetzigen Zeitpunkt jedoch weniger gut aus. Apple startet nur sehr zögerlich mit der Unterstützung der PWA Features auf dem iOS Betriebssystem und dem Safari Browser. Resultierend daraus können neben dem Safari Browser auch die Browser der anderen Browserhersteller auf iOS Systemen keine bessere Unterstützung der APIs liefern.

> „Für einige Teilaspekte der Progressive Web Apps stellt Apple zwar proprietäre (Apple Pay JS) oder veraltete (Application Cache) Schnittstellen zur Verfügung. Doch die Service Workers lassen sich derzeit überhaupt nicht nutzen. Das gilt zugleich auch für alle Ausgaben von 'Fremdbrowsern' unter iOS, denn auch unter der Haube des Chrome oder Firefox für iOS tickt letztlich ein abgespeckter Safari“ (Liebel, 2017c).

Verdeutlicht werden diese Aussagen durch die Tabellen 2.2 - 2.10, die im weiteren Verlauf dieses Abschnittes beschrieben werden und detailliert darstellen, welche Funktionen von den Browsern bereits unterstützt werden. Zur Erstellung des Inhalts der Tabellen wurde die Website whatwebcando.today[3], welche gleichzeitig auch eine PWA ist, auf einem Huawei P9lite mit Android Version 7 und auf einem Iphone 8 mit iOS Version 11.2.1 genutzt. Die Browser Chrome, Firefox, Edge und Safari wurden ausgewählt, da diese laut W3Counter (2017) den größten Marktanteil besitzen. Verwendet werden die Browser in den folgenden, zum Zeitpunkt der Verfassung

[3]https://whatwebcando.today/

dieses Kapitels aktuellen Versionen:

- Android: Chrome 63 | Firefox 58.0.2 | Edge 1.0
- iOS: Chrome 63 | Firefox 10.04 | Edge 41.8 | Safari 11.0.2

Die Versionsnummern der Browser sind zwischen Android und iOS nicht vergleichbar und somit lassen sich aus den Versionsnummern keine Rückschlüsse auf den Stand der API Unterstützung ableiten. Die durch die HTML5 APIs ermöglichte Funktionalität ist in die folgenden, von whatwebcando.today übernommenen, Themengebiete gegliedert.

- Natives Verhalten
- Umgebung
- Gerätefunktionen
- Mikrofon & Kamera
- Betriebssystem
- Eingabe
- Nahtlose Erfahrung
- Standort & Position
- Bildschirm & Ausgabe

Derzeit noch experimentelle API Implementierungen werden genauso wie gar nicht vorhandene Implementierungen als nicht Verfügbar (-) aufgeführt. Die gebotene Kompatibilität von Chrome und Edge wird identisch sein, da laut Lyndersay (2017) der Edge Browser für Android auf Chromium basiert.

Tabelle 2.2 listet Features auf, die für ein natives Verhalten der Anwendung nötig sind. Wichtig sind dabei vor allem die APIs für Push Benachrichtigungen und für die Homescreen Installierbarkeit. Hinter der Homescreen Installierbarkeit verbirgt sich das zuvor beschriebene Web App Manifest. Diese beiden wichtigen Eigenschaften waren bisher nicht durch Web Apps realisierbar. Wie in der Tabelle ersichtlich, sind auf Android bei Chrome, Firefox und Edge alle aufgeführten Funktionen möglich. Auf dem iOS System wird momentan nur die Vordergrund Erkennung unterstützt,

hinter der sich die *Page Visibility API* verbirgt. „The Page Visibility API defines a means to programmatically determine the visibility state of a top level browsing context, and to be notified if the visibility state changes“ (W3C, 2017). Alle anderen Aspekte beim nativen Verhalten sind jedoch auf iOS Endgeräten noch nicht nutzbar.

Natives Verhalten	Android			iOS			
Features \ Browser							
Lokale Benachrichtigungen	+	+	+	-	-	-	-
Push Benachrichtigungen	+	+	+	-	-	-	-
Homescreen Installierbarkeit	+	+	+	-	-	-	-
Vordergrunderkennung	+	+	+	+	+	+	+
Berechtigungen	+	+	+	-	-	-	-

Tabelle 2.2.: Browserunterstützung 'Natives Verhalten'.

Als nächstes wird in Tabelle 2.3 die Kommunikation mit der Umgebung des Endgerätes betrachtet. Hierbei ist bereits auf den ersten Blick eine deutlich schlechtere Verfügbarkeit ersichtlich. Zu den Umgebungsfunktionen gehören unter anderem die Bluetooth, USB und NFC Schnittstellen. Unter Chrome und Edge auf Android ist die Nutzung von Bluetooth und USB möglich. Eine Verwendung von NFC nach aktueller Spezifikation ist jedoch derzeit bei keinem Browser implementiert. Des Weiteren ist der Zugriff auf den Lichtsensor des Endgerätes nur beim Firefox möglich. Auf iOS Systemen steht bisher keines der in dieser Tabelle aufgeführten Features zur Verfügung.

Umgebung	Android			iOS			
Features \ Browser							
Bluetooth	+	-	+	-	-	-	-
USB	+	-	+	-	-	-	-
NFC	-	-	-	-	-	-	-
Umgebungslicht	-	+	-	-	-	-	-

Tabelle 2.3.: Browserunterstützung 'Umgebung'.

Tabelle 2.4 zeigt die Browserkompatibilität der direkten Funktionen des Endgeräts bzw. den Zugriff auf diese.

Gerätefunktionen	Android			iOS			
Features \ Browser							
Netzwerktyp- & Geschwindigkeit	+	+	+	-	-	-	-
Onlinezustand	+	+	+	+	+	+	+
Vibration	+	+	+	-	-	-	-
Batteriestatus	+	-	+	-	-	-	-
Arbeitsspeicher	+	-	+	-	-	-	-

Tabelle 2.4.: Browserunterstützung 'Gerätefunktionen'.

Hierzu gehört z.B. der Zugriff auf die Vibration, die momentanen Netzwerkeigenschaften und der Online Zustand. Diese drei Funktionen werden auf Android von allen drei Browsern unterstützt. Auf iOS Systemen ist nur der Zugriff auf den Onlinezustand möglich. Weiter ist bei Chrome und Edge auf Android der Zugriff auf Eigenschaften des Arbeitsspeichers und den Batteriestatus möglich. „The Device Memory API allows Web applications to assess the class of the device by the size

of the RAM memory installed“ (whatwebcando.today, 2017b). Hierdurch kann laut whatwebcando.today (2017b) die Website an die Geräteleistung angepasst werden, um die Performance zu steigern.

Ein anderer wichtiger Aspekt ist die Unterstützung der Mikrofon und Kamera APIs. Beim ersten Blick auf Tabelle 2.5 ist direkt ersichtlich, dass hier die API Unterstützung der Browser weit fortgeschritten ist. So sind bis auf die erweiterten Kamerafunktionen beim Firefox alle gelisteten Features bei den drei betrachteten Browsern auf Android Systemen verfügbar. Auf dem iOS System wird nur die Echtzeitkommunikation von allen Browsern unterstützt. „Real-Time Communication in the Web [...] is a set of APIs allowing Web applications to send and receive streaming real-time video, audio and data to/from remote peers, without relying it through the centralized server“ (whatwebcando.today, 2017d). Neben der WebRTC Unterstützung ist auch die API für Audio- & Videoaufnahmen beim Safari Browser auf iOS implementiert.

Mikrofon und Kamera	Android			iOS			
Features \ Browser							
Audio- & Videoaufnahme	+	+	+	-	-	-	+
Erweiterte Kamerafunktionen	+	-	+	-	-	-	-
Medienaufnahme	+	+	+	-	-	-	-
Echtzeitkommunikation	+	+	+	+	+	+	+

Tabelle 2.5.: Browserunterstützung 'Mikrofon und Kamera'.

Als nächstes werden die Features des Betriebssystems aufgeführt. Hierbei wird der Offlinespeicher und der Dateizugriff von allen Browsern auf Android und iOS unterstützt. Weitere APIs sind für den Zugriff auf die Kontakte, SMS und das Task Scheduling zuständig. „The [Task Scheduler API's] idea is to act as a wrapper for the underlying operating system scheduler and allow authorized Web applications

implementing scenarios such as alarms, reminders or periodic data synchronization“ (whatwebcando.today, 2017e). Diese werden derzeit jedoch noch von keiner Browser und Betriebssystem Kombinationen unterstützt. Ein weiterer Punkt sind die Speicherkontingente, die nur von Chrome und Edge auf Android implementiert sind. Hierbei geht es um Einsicht in den momentan genutzten Speicher und wie viel davon der Applikation zur Verfügung steht.

Betriebssystem	Android			iOS			
Features \ Browser							
Offlinespeicher	+	+	+	+	+	+	+
Dateizugriff	+	+	+	+	+	+	+
Kontakte	-	-	-	-	-	-	-
SMS	-	-	-	-	-	-	-
Speicherkontingente	+	-	+	-	-	-	-
Task Scheduler	-	-	-	-	-	-	-

Tabelle 2.6.: Browserunterstützung 'Betriebssystem'.

In Tabelle 2.7 sind vier Features in Bezug auf Eingabemöglichkeiten des Endgeräts aufgeführt. Hierbei ist ersichtlich, dass Berührungsgesten und die Zwischenablage durch alle Browser auf iOS und Android unterstützt werden. Die Berührungsgesten sind besonders wichtig, da die meisten mobilen Endgeräte durch Touch bedient werden. Des Weiteren ist es auf Chrome und Edge für Android möglich, die Stimme als Eingabemethode und zur Steuerung zu verwenden. Außerdem sind Anpassungen der Web Anwendung an das genutzte Zeigegerät möglich. „The Interaction Media part of CSS4 specification defines the media queries allowing Web applications to alter its layout and user interface depending on the way the user is supposed to interact with the application“ (whatwebcando.today, 2017c).

Eingabe	Android			iOS			
Features \ Browser							
Berührungsgesten	+	+	+	+	+	+	+
Spracherkennung	+	-	+	-	-	-	-
Zwischenablage	+	+	+	+	+	+	+
Zeigegerät Anpassungen	+	-	+	+	+	+	+

Tabelle 2.7.: Browserunterstützung 'Eingabe'.

Weiter sind in der folgenden Tabelle 2.8 Features aufgelistet, die sich auf das Benutzererlebnis auswirken.

Benutzererlebnis	Android			iOS			
Features \ Browser							
Offline Modus	+	+	+	-	-	-	-
Hintergrundsynchronisation	+	-	+	-	-	-	-
Inter-App Kommunikation	+	-	+	-	-	-	-
Zahlungen	+	-	+	+	+	+	+
Zugangsdaten Management	+	-	+	-	-	-	-

Tabelle 2.8.: Browserunterstützung 'Nahtlose Erfahrung'.

Ein wichtiger Aspekt ist hier der Offline Modus, hinter dem sich die bereits zuvor beschriebenen Service Worker verbergen, welche Hauptbestandteil von PWAs sind und unter anderem den bisher im Web genutzten *Application Cache* ersetzen. Implementiert ist die Service Worker Unterstützung jedoch bisher nur auf Android Browsern. Auf Apple Geräten ist die Nutzung noch nicht möglich. Ähnlich zum Offline Modus ist die Hintergrundsynchronisation, welche auch mit den Service Workern zu tun

hat. „The Background Sync API allows authorized Web applications to not rely on having stable internet connection and defer network-related operations to the moment the connection is available“ (whatwebcando.today, 2017a). Genauer betrachtet ist es laut Archibald (2015) durch diese API möglich, der Anwendung mitzuteilen, dass die Internetverbindung wieder zur Verfügung steht und so aufgeschobene Aktionen nun durchgeführt werden können. Nutzbar ist die API im Chrome und Edge Browser auf Android. In den gerade erwähnten Browsern ebenso nutzbar ist eine API für Inter-App Kommunikation (*Web Share API*), eine Zahlungs API (*Payment Request API*) und einer API für das Management von Zugangsdaten (*Credential Management API*). Die API für die Weiterleitung des Zahlungsprozesses an das Betriebssystem, ist auf iOS Geräten bei allen Browsern nutzbar. Die anderen Features sind jedoch noch nicht bei iOS Webbrowsern verfügbar.

Ein weiterer wichtiger Aspekt im mobilen Bereich ist die Standortlokalisierung und Gerätepositionierung. Hierfür zeigt Tabelle 2.9 fünf Funktionalitäten.

Standort und Position	Android			iOS			
Features \ Browser							
Geolocation	+	+	+	+	+	+	+
Geofancing	-	-	-	-	-	-	-
Geräteorientierung	+	+	+	+	+	+	+
Gerätebewegung	+	+	+	+	+	+	+
Näherungssensor	-	+	-	-	-	-	-

Tabelle 2.9.: Browserunterstützung ’Standort und Position’.

Die *Geolocation API* ermöglicht es der Web Anwendung, auf den Standort des Gerätes zuzugreifen. Diese Funktion ist bei allen Browser und Betriebssystem Kombinationen implementiert. Außerdem überall verfügbar sind APIs für die Geräteorien-

tierung und die Gerätebewegung. Noch nicht implementiert ist das Geofancing, an dessen Implementierung laut whatwebcando.today (2016) derzeit auch nicht aktiv gearbeitet wird. Ein weiterer Sensor ist der Näherungssensor. Auf diesen ist ein Zugriff nur über den Firefox Browser auf Android möglich.

Die Features in der letzten Tabelle handeln vom Gerätebildschirm und der Ausgabe.

Bildschirm und Ausgabe	Android			iOS			
Features \ Browser							
Vollbild	+	+	+	-	-	-	-
Bildschirmausrichtung	+	+	+	-	-	-	-
Wake Lock	-	-	-	-	-	-	-
Zugriff auf externe Displays	+	-	+	-	-	-	-

Tabelle 2.10.: Browserunterstützung 'Bildschirm und Ausgabe'.

So ist es auf Android mit allen drei Browsern möglich, Web Anwendungen in Vollbild anzuzeigen. Des Weitern besteht ein Zugriff auf die Bildschirmausrichtung des Gerätes. Die *Wake Lock API* ist jedoch von keinem Hersteller bisher implementiert worden. „The Wake Lock API allows Web applications to prevent the resource such as the screen or system from becoming unavailable as long as the application holds a lock for that resource“ (whatwebcando.today, 2017f). Eine weiter API ist die *Presentation API*. Diese ermöglicht es, über Web Anwendungen auf externe Präsentation Geräte wie z.B. Beamer zuzugreifen. Verfügbar ist dies in Chrome und Edge auf Android. Auf iOS Systemen ist keine der vier hier aufgeführten Features derzeit in Webanwendungen realisierbar.
Für die Nutzung vieler Features einer PWA ist aus Sicht des Nutzers also momentan auf Grund der mangelnden allgemeinen Cross-Browser Unterstützung die Verwendung von Chrome oder Edge empfehlenswert. Die Android Version an sich ist laut Tests von Steiner (2018) nicht ausschlaggebend. Dies ist wichtig, da die Updatepo-

litik der Smartphone Hersteller nicht sehr gut ist. Des Weiteren wird der Ausbau der PWA Unterstützung im Laufe der Zeit weiter voranschreiten. „Die Browser der meisten Anbieter (Google, Mozilla, Microsoft, Opera) werden viel daran setzen, dass eine bestmögliche Kompatibilität des offenen PWA-Formats schnell Einzug in ihre Produkte findet“ (1&1-DigitalGuide, 2017a). Und auch Apple scheint in Zukunft für eine bessere PWA Unterstützung beim Safari Browser zu sorgen. Laut Claburn (2017) hat Apple Anfang August 2017 mit der Implementierung der Service Worker Unterstützung begonnen. Dies ist auch auf der Webkit[4] Website ersichtlich. „WebKit is the web browser engine used by Safari, Mail, App Store, and many other apps on macOS, iOS, and Linux“ (WebKit, 2018). Dort ist das Service Worker Feature nun mit 'Supported In Preview' gekennzeichnet. „Now that WebKit is implementing Service Workers, the technology can be expected to show up in future versions of Apple's Safari browser“ (Claburn, 2017). Somit wird es wahrscheinlicher, dass auch weitere PWA Features in Zukunft von Apple Geräten nutzbar werden. Des Weiteren bietet eine PWA auch auf Apple Systemen Vorteile und kann alles, was eine Web App auch kann. „Despite the fact that iOS doesn't support the full feature set of Progressive Web Apps, early evidence indicates that Progressive Web Apps perform better on iOS than the sites they replace“ (Grigsby, 2016). Eine PWA kann sich somit auch für iOS Geräte lohnen. Trotzdem wird die derzeit noch schlechte Unterstützung seitens Apple in die Bewertung der PWAs einfließen.

2.2.4. PWA Beispiele

Um zu verdeutlichen, wie der Einsatz einer PWA aussehen kann und welche unterschiedlichen Gründe für die Erstellung einer PWA existieren, werden im Folgenden die PWA Anwendungen von fünf Unternehmen kurz vorgestellt. Des Weiteren sammeln z.B. PWAStats (2018) und Google (2018e) die Ergebnisse von vielen unterschiedlichen Unternehmen, die eine PWA entwickelt haben. Hierbei werden Beispiele aus verschiedensten Branchen zusammengetragen.

[4] https://webkit.org/status

Twitter Lite: Twitter ist ein von Millionen Menschen weltweit genutzter Kurznachrichtendienst. Zusätzlich zur Nativen App und Website wurde nun auch eine Progressive Web App vom Twitter Team entwickelt. Diese wird als Twitter Lite bezeichnet gilt als eine der bekanntesten PWAs. Traughber (2017) erläutert, dass in dieser Version des Kurznachrichtendienstes der Datenverbrauch und der benötigte Speicherplatz auf dem Gerät des Kunden verringert wird und trotzdem alle Hauptfeatures nutzbar sind. Außerdem verringert sich die Startzeit und Inhalte laden auch bei langsamer und unzuverlässiger Internetverbindung schneller und stabiler. Hierdurch wird auch den Menschen der Zugang zu Twitter ermöglicht, die in Regionen leben, in denen eine schnelle und dauerhaft zuverlässige Datenverbindung nicht selbstverständlich ist. Realisiert werden diese Verbesserungen durch die Nutzung des PWA Ansatzes und den damit verbundenen Features. „We have been able to achieve speed and reliability [...] by using the new capabilities of modern browsers on Android (e.g., Google Chrome) which include Service Worker, IndexedDB, Web App Install Banners, and Web Push Notifications. “ (Gallagher, 2017).

Airberlin: Als erste Airline weltweit hat die inzwischen insolvente Fluggesellschaft Airberlin eine Progressive Web App für seine Kunden entwickelt. Vorgestellt wurde diese innovative App auf der Entwicklerkonferenz Google I/O im Mai 2016 in San Francisco. „Mit der Progressive Web App Technologie können Airberlin Fluggäste nach einem einmaligen Web Check-in jederzeit auf der Welt ortsunabhängig und ohne Internetverbindung ihre persönliche Bordkarte und weiterführende Reiseinformationen zu ihrer Zieldestination abrufen“ (Airberlin, 2016). Hier wird laut Airberlin (2016) die Eigenschaft genutzt, dass bei einer PWA auch Informationen Offline im Browser zur Verfügung gestellt werden können, sobald die Inhalte einmal geladen wurden. Dies ist ein klarer Vorteil, da in Flughafengebäuden teilweise keine mobilen Datenverbindungen möglich sind und auch ein kostenloses Wifi nicht immer zur Verfügung steht. Des Weiteren haben Nutzer oft keinen passenden Datentarif, um im Ausland eine mobile Datenverbindung nutzen zu können. Eine Native App, welche erst vom Kunden installiert werden muss, ist so nicht mehr nötig. Denn alle gewohnten Funktionen konnten nun über den Browser angeboten werden. Dies dürf-

te für Kunden, die selten oder nur einmalig mit Airberlin fliegen, ein Vorteil sein, da sie nun nicht extra die App herunterladen müssen.

Alibaba: Alibaba ist laut Google (2016a) die weltweit größte B2B Handelsplattform im Internet. Für diesen Erfolg macht das Unternehmen unter anderem eine gute Unternehmenspräsenz im Web verantwortlich, die den Kunden eine gute User Experience bietet. „Building great mobile experiences is an indispensable part of Alibaba.com's success. The mobile web is their primary platform for discovery on mobile, so they've always focused on design and functionality“ (Google, 2016a). Bisher wurde allerdings aufgrund der begrenzten Möglichkeiten der Web Entwicklung versucht, möglichst viele Nutzer mit Hilfe der Website zur Nutzung der Nativen App von Alibaba zu überzeugen. Nun soll jedoch den Nativen App Nutzern und den Website Nutzern eine positive Nutzererfahrung geboten werden. „They wanted to deliver a great user experience for both first-time visitors [...] and repeat visitors“ (Google, 2016a). Laut Case Study von Google (2016a) hat das Web Team von Alibaba daher eine PWA entwickelt, die schnell und effektiv ist und eine zuverlässige mobile Web Erfahrung bietet. Für Android sind über den Browser Push Benachrichtigungen möglich. Außerdem besteht auf Android die Möglichkeit, sich ein Icon auf den Homescreen abzulegen. „Which resulted in users re-engaging four times more often than other mobile users“ (Google, 2016a).

MakeMyTrip: MakeMyTrip ist laut Google (2017e) Indiens größtes Reiseunternehmen, über dessen Website und Native App der Kunde unter anderem Flüge und Hotelübernachtungen buchen kann. Im Monat besuchen etwa 8 Millionen Menschen die Website, wovon zwei Drittel über die mobile Version zugreifen. Auch die Native App ist mit 23 Millionen Downloads sehr beliebt. „But in a market of over a billion people, many potential users only have intermittent connectivity and visit the site from low-end phones with data limitations. These two factors make it difficult to drive users to download a native app“ (Google, 2017e). Um den Kunden aus Gebieten, in denen keine gute Internetverbindung herrscht und die Menschen keine guten Smartphones besitzen trotzdem ein gutes mobiles Nutzererlebnis zu bieten, wurde

von MakeMyTrip eine PWA entwickelt. Diese löst laut Google (2017e) die bisherige mobile Website ab, soll jedoch die Native App nicht ersetzen.

The Weather Channel: The Weather Channel ist laut Google (2016c) ein zu IBM gehörendes Unternehmen, das Wettervorhersagen erstellt und diese den Nutzern in vielen Ländern auf verschiedenen Wegen zur Verfügung stellt. Mögliche Wege sind unter anderem eine Native App für mobile Endgeräte und eine Website, welche von zu Hause auf Desktop Computern genauso abrufbar ist, wie unterwegs vom Smartphone aus. „Weather is volatile, and delivering important data when people need it isn't always easy, so the team was interested in improving its mobile web experience“ (Google, 2016c). Da laut Google (2016c) die Native App mit den möglichen Push Benachrichtigungen nicht für jedermann zugänglich ist und die Hälfte der Kunden eh die Website nutzt, möchte das Unternehmen das Nutzererlebnis im mobilen Web verbessern. Hierzu wurde eine PWA entwickelt. „Being a site that focuses on immediacy, especially related to real-time weather notifications in severe situations, the first area the team tackled was to create browser push notifications when severe weather hits“ (Google, 2016c). Diese Push Benachrichtigungen stehen den Nutzern von Android Endgeräten zur Verfügung.

Konga: Konga ist laut Google (2017c) Case Study eine führende E-Commerce Plattform in Nigeria. Im Vergleich zu anderen Kontinenten wurde hier der Desktop Bereich übersprungen und die meisten Nutzer starten direkt mit mobilen Endgeräten. „Poor connectivity, the prevalence of low-end devices, and several other obstacles hindered Konga's ability to grow“ (Google, 2017c). So beschreibt Google (2017c) weiter, dass die Internetkosten für die Bürger in Bezug zum Einkommen sehr teuer sind. Aufgrund dieser Tatsachen zögern viele Nutzer von Konga, die Native App herunterzuladen. Um den Nutzern trotzdem die Vorteile einer Nativen App zu ermöglichen, wurde eine PWA entwickelt. „Konga wanted to provide all of their web users with the benefits of their app, including performance, the ability to work offline, and re-engage without the data cost“ (Google, 2017c). Durch die Features der Progressive Web App und dem damit verbundenen besserem Offline Handling

und geringerem Speicherverbrauch ist dies möglich.

3. Konzeption der Anwendungsfälle

Wie bereits erwähnt ist das Ziel dieser Arbeit eine genaue Betrachtung von Progressiven Web Apps sowie die Analyse, ob die Migration einer Webseite zu einer PWA für Nutzer und Unternehmen einen Mehrwert bieten kann. Um dies festzustellen, sollen zwei Anwendungsfälle realisiert werden. Dazu werden im ersten Schritt beide Use Cases mit Standardtechnologien implementiert, die von einem Großteil der Webentwickler genutzt werden. Im Anschluss folgt eine Migration der Use Cases zu Progessiven Web Apps mit einer umfangreichen Erläuterung der Umsetzung. Für die Ermittlung des Nutzermehrwertes werden die Benutzerfunktionen der verschiedenen Ausführungen verglichen. Der Wertzuwachs der Unternehmen wird durch die Abwägung des Implementierungsaufwands erarbeitet.
Insgesamt werden zwei unterschiedliche Use Cases entwickelt. Zum einen soll ein Nachrichtenportal implementiert werden, das den Nutzer über das Portal erstellte Nachrichten informiert. Zum anderen soll im zweiten Anwendungsfall ein Videoportal entwickelt werden, welches Ähnlichkeiten zu einem primitiven youtube.com Abbild aufweist. Die Auswahl der Anwendungsfälle unterlag der Unterscheidung durch mindestens ein Kriterium. Damit soll die Portabilität von PWA's beurteilt und vorhandene Muster erkannt werden. In dieser Ausarbeitung werden zwei Websites entwickelt, die in einem Fall wenige Daten und im anderen Fall viele Daten übertragen.
Um die Entwicklung realitätsnah durchzuführen, werden Technologien genutzt, die die von *vielen* Entwicklern angewendet werden. Die Maßeinheit *viel* bezieht sich in diesem Punkt auf die Entscheidung einer Technologie, die beispielsweise nicht von einer kleinen Minderheit genutzt wird.
Im TIOBE Programming Community Index (September 2017)[1] belegen die Progam-

[1] https://www.tiobe.com/tiobe-index

miersprachen *Java*, *C*, *C++*, *C#* und *Python* die ersten fünf Plätze. Der TIOBE Index gibt laut TIOBE Software B.V. (2017) die Popularität einer Programmiersprache, auf Grundlage von Suchanfragen, Entwicklern und Kursen, an. Eine genaue Berechnung wird nicht angegeben, weshalb dieser Index nur ein Gespür für die Häufigkeit der Verwendung geben sollte. Diese Programmiersprachen werden zweifellos oft genutzt, jedoch ist es fraglich, ob diese auch häufig in der Webentwicklung verwendet werden. Laut einer bei Wikipedia (2018) zusammen getragenen Übersicht werden viele unterschiedliche Programmiersprachen, wie beispielsweise JavaScript, Java, PHP oder Python zur Entwicklung einer Webseite angewendet. Diese angewendeten Programmiersprachen spiegeln sich ebenfalls im oberen Ranking des TIOBE Indexes wieder. Bei einer Umfrage von Stack Overflow (2017) wurden Entwickler gefragt welche Technologien sie für die Entwicklung der Bereiche wie Back-End, Front-End oder Full-Stack verwenden. Laut dieser Umfrage von Stack Overflow (2017) nutzen die meisten Entwickler JavaScript im Backend- und Frontend-Bereich, dicht gefolgt von C# und PHP im Backend-Bereich. Aufgrund der Quellen, die ein Indiz dafür geben, welche Programmiersprachen häufig genutzt werden, wird für die Entwicklung der Use Cases die Programmiersprache PHP eingesetzt. Diese Programmiersprache hat zwar nicht den ersten Platz erreicht, ist jedoch ebenfalls im oberen Ranking. Außerdem sind die Fähigkeiten der Autoren in PHP besser als in JavaScript. Bei der Protokollierung des Aufwands wäre PHP ebenfalls realitätsnäher, da keine zusätzliche Einarbeitung in JavaScript erfolgen muss und Unternehmen sich ebenfalls für die Technologien entscheiden könnten, die sie bereits beherrschen, um das Produkt schneller zu veröffentlichen.

Zusätzlich zu der Entscheidung einer Programmiersprache kann es sinnvoll sein, ein Entwickler-Framework zu verwenden. Gemäß Symfony™ (2018) bieten Frameworks den Entwicklern generische Module an. Bei Verwendung der Module kann der Entwickler sich auf andere Areale konzentrieren und spart dadurch Zeit im Entwicklungsprozess. Darüber hinaus wird der Entwickler gezwungen, eine bestimmte Struktur beim Implementieren einzuhalten, wodurch andere Entwickler sich schneller in das System einarbeiten können und die Wartbarkeit damit ebenfalls erhöht werden kann.

In einem Beitrag zeigt Elitech Systems (2017) einige PHP-Frameworks mit ihren Features auf, die nach der Meinung des Autors die weitverbreitetsten Frameworks sind. Darin aufgelistet werden Laravel[2], CodeIgniter[3], Symfony[4], Yii[5] sowie nette Framework[6]. Ein Blick auf die Google Trends, siehe Abbildung 3.1, zeigt einen massiven Popularitätsvorsprung des Laravel Frameworks gegenüber anderer Frameworks. Die Popularität muss nicht unbedingt die Qualität des Frameworks wiederspiegeln. Andererseits wirft dies jedoch die Vermutung auf, dass es dadurch mehr Ressourcen an Anleitungen und Dokumentation gibt als bei den restlichen Frameworks. Infolgedessen wird für die Entwicklung das PHP-Framework Laravel verwendet.

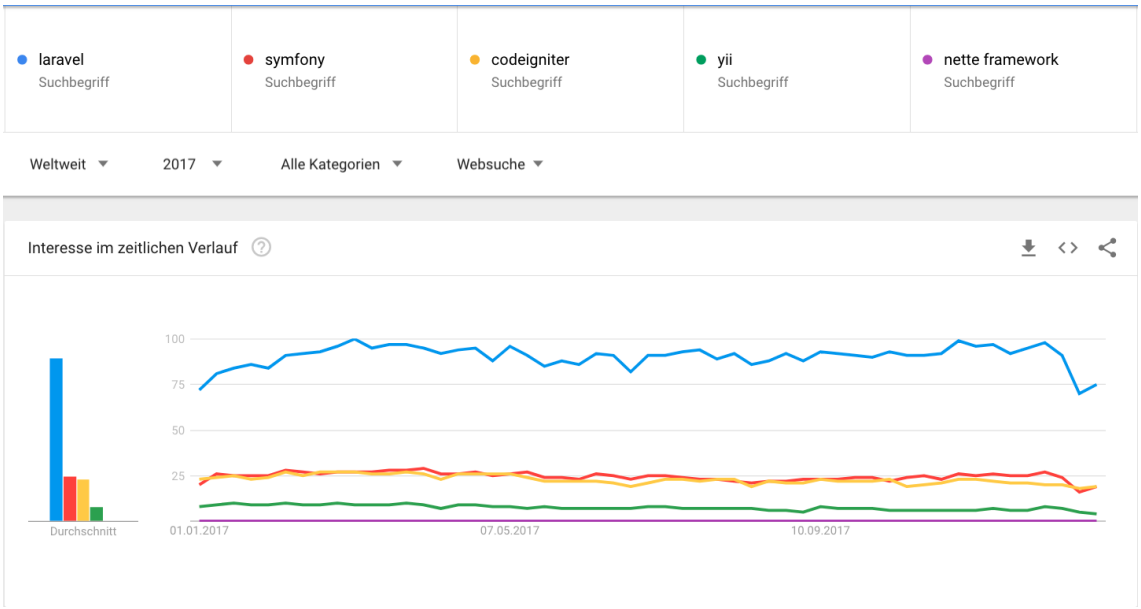


Abbildung 3.1.: Google Trends ausgewählter PHP-Frameworks im Jahre 2017 (Google, 2018b)

3.1. Entwicklung eines Nachrichtenportals

Im ersten Schritt werden die Anforderungen des ersten Anwendungsfalles, eines Nachrichtenportals, definiert und anhand von Mockups veranschaulicht. Die Start-

[2]http://laravel.com
[3]https://www.codeigniter.com
[4]https://symfony.com
[5]http://www.yiiframework.com
[6]https://nette.org

seite des Nachrichtenportals soll in erster Linie veröffentlichte Nachrichten, wie in Abbildung 3.2 zu sehen, darstellen, welche automatisch nach Datum absteigend sortiert sind. Dabei können Nachrichten aus reinem Text, oder aber auch Bildern, bestehen.

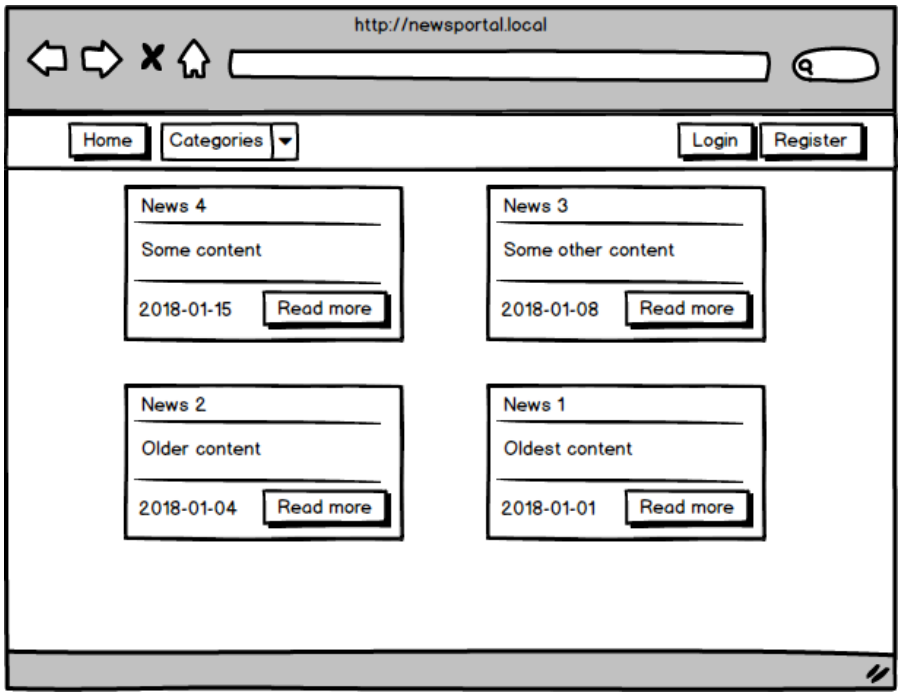


Abbildung 3.2.: Mockup der Startseite des Nachrichtenportals (eigene Darstellung)

Um die Übersicht zu wahren, wird der Inhalt der Nachricht, wie in Abbildung 3.3 ersichtlich, in verkürzter Form auf der Startseite angezeigt. Des Weiteren soll das Portal elementare Nutzerfunktionen anbieten. Die Besucher sind in der Lage ein Nutzerkonto zu erstellen, wodurch weitere Funktionen, wie das Kommentieren von Nachrichten, freigeschaltet werden.

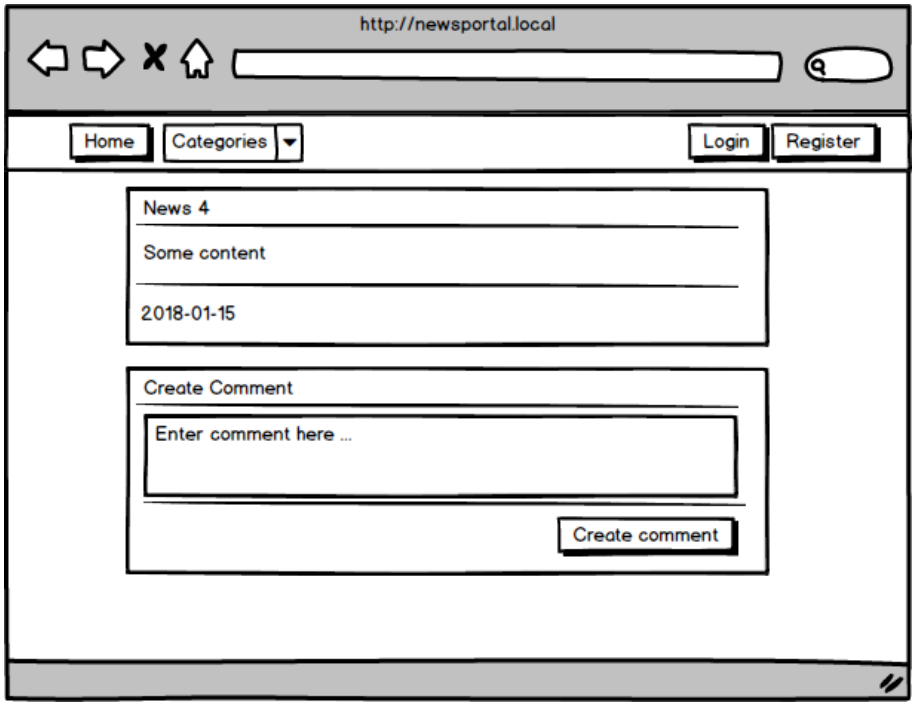


Abbildung 3.3.: Mockup der Detailseite einer Nachricht im Nachrichtenportal (eigene Darstellung)

Neben normalen Nutzerkonten existieren Administratorkonten mit erweiterten Rechten. Sie beinhalten alle Funktionen eines normalen Nutzerkontos inklusive weiterer Funktionen wie das Löschen von Kommentaren anderer Nutzer. Außerdem können Administratoren mittels eines WYSIWYG-Editors Nachrichten erstellen, bearbeiten oder löschen.

3.1.1. Aufwand

Um den Nutzen für ein Unternehmen besser analysieren und quantifizieren zu können, soll der Aufwand, wie in Kapitel 3 erwähnt, protokolliert werden. Der Aufwand wird anhand von *Arbeitstagen* gemessen. Ein *Arbeitstag* umfasst dabei *8* Stunden. Die Tabelle 3.1 gibt eine Übersicht über die Funktionen und deren Aufwand. Hinsichtlich der Übersichtlichkeit wurden einzelne Funktionen in Meilensteine gruppiert. Im ersten Schritt wurde das System zur Entwicklung vorbereitet. Dazu wurden die benötigten Programme, unter anderem virtuelle Maschinen, installiert und konfigu-

riert. Im nächsten Schritt wurden die ersten Funktionen, wie beispielsweise elementare Nutzerfunktionen, implementiert. Schritt für Schritt wurden weitere Meilensteine bearbeitet. Während und nach der Entwicklung der Funktionen wurden manuelle Tests durchgeführt, um Fehler zu beheben und Optimierungen durchzuführen. Der Gesamtaufwand aller Meilensteine beträgt 9,5 Arbeitstage, was 76 Stunden entspricht.

Meilenstein	**Arbeitstage**
Systemvorbereitung	0.5
Nutzerfunktionen (Registrierung, Authentifizierung)	1,5
Admininistratorfunktionen (Rollenbasierte Nutzer)	0,5
Nachrichten (Erstellen, Lesen, Bearbeiten, Löschen)	2
Kategorien (Erstellen, Lesen, Bearbeiten, Löschen)	1
Fehlerbehebungen/Optimierungen	4
Gesamt	**9,5**

Tabelle 3.1.: Aufwandsprotokoll des Nachrichtenportals

3.1.2. Software-Architektur

Im Folgenden soll die Software-Architektur erläutert werden. Wie bereits in Kapitel 3 erwähnt, wird in der Entwicklung das PHP-Framework Laravel verwendet. Die Konzipierung von Laravel ist so gestaltet, dass es vom Entwickler fordert, das MVC (Model View Controller) Prinzip von Grund auf anzuwenden. Neben der MVC-Architektur bietet das Framework noch weitere Eigenschaften, die den Entwickler dabei helfen, eine solide Webseite zu entwickeln. Eine grobe Übersicht bietet die Abbildung 3.4 in Form einer Beispiel HTTP-Anfrage eines Nutzers. Laut Abbildung 3.4 sendet der Nutzer im ersten Schritt eine Anfrage mit seinem Browser an den

Server. Das Laravel Framework nimmt die Anfrage an und übergibt sie der *Middleware* Schicht. Diese überprüft anhand von vordefinierten Regeln, ob diese Anfrage zulässig ist und leitet diese weiter. Beispielsweise kann diese Schicht kontrollieren, ob der aktuelle Nutzer die nötigen Rechte hat, diese Webseite zu besuchen. Im nächsten Schritt wird die Anfrage an den *Router* weitergeleitet. Der *Router* entscheidet, welcher *Controller* für welche Unterseite verantwortlich ist und übermittelt dann die Anfrage an den zuständigen *Controller*. Der *Controller* enthält die Kernfunktionalitäten der Applikation. Dieser verarbeitet die Anfrage und ruft die zuständigen Funktionen auf. Falls Daten persistent gespeichert werden sollen, können die Daten mit den *Model-Instanzen* abgerufen, erstellt, bearbeitet oder gelöscht werden. Die einzelnen *Model-Instanzen* sind nämlich mit der Datenbank verbunden. Änderungen an den Objekten werden dementsprechend in der Datenbank gespeichert.

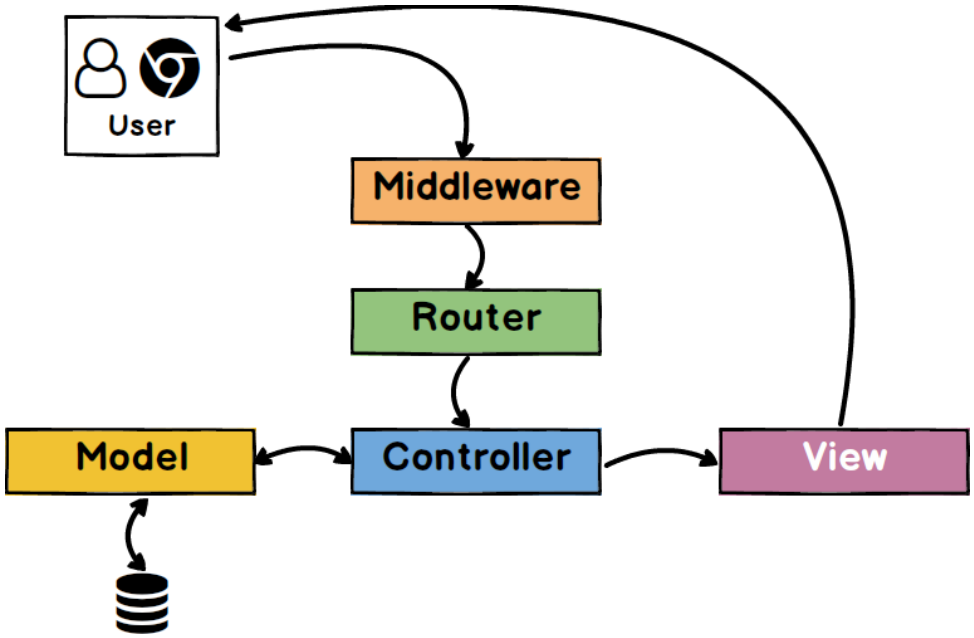


Abbildung 3.4.: Lebenszyklus einer HTTP Anfrage (eigene Darstellung)

Bei der Implementierung wird viel Wert auf die Erweiterbarkeit und Wartbarkeit gelegt. Aus diesem Grund wurde bei der Konzeption darauf geachtet, die Empfehlungen des PHP-Frameworks einzuhalten. Wie in Kapitel 3.1 beschrieben, soll das Nachrichtenportal die Funktionen, Nachrichten speichern und darstellen, sowie elementare Nutzerfunktionen, wie Kommentieren und nach Kategorien filtern, anbieten. Als Grundlage zur Entwicklung dient die MVC-Architektur.
Die Abbildung 3.5 beinhaltet einen Ausschnitt der wichtigsten Klassen der Web App. Die Controller enthalten in der Regel die Kernfunktionalitäten. In diesem Fall existieren fünf Controller. Der *SiteController* ist dafür zuständig, die Startseite korrekt darzustellen. Dazu werden die einzelnen Nachrichten mit Hilfe des Models *Post* aus der Datenbank gelesen und in der korrekten Sicht angezeigt. Die Controller *PostController*, *CommentController* und *CategoryController* sind vom Funktionsumfang bedeutsamer. Sie regeln die benötigten Funktionen um Nachrichten, Kategorien und Kommentare zu lesen, bearbeiten, erstellen oder zu löschen. Die Wahrscheinlichkeit ist hoch, dass diese wichtigen Funktionen in der Zukunft auch von anderen Klassen benötigt werden, beispielsweise bei der Entwicklung einer API-Schnittstelle. Aus diesem Grund wurden diese Funktionen in *Job-Klassen* isoliert. Falls beispielsweise eine Useranfrage zur Erstellung einer Nachricht existiert, wird diese Anfrage im ersten Schritt an den *PostController* übergeben. Nach erfolgreicher Validierung der Inhalte wird der Auftrag *CreatePostJob* gestartet. Dieser enthält die komplette Logik, um eine Nachricht zu erstellen. Sobald der Prozess abgeschlossen ist, zeigt der *PostController* das Resultat an. Das gleiche Verfahren wird bei den restlichen Funktionen (Bearbeiten und Löschen), wie ebenfalls bei den Kommentaren und Kategorien mit den zugehörigen *Jobs* ausgeführt. Primitive Operationen, wie beispielsweise das Anzeigen aller Kategorien, werden direkt im Controller ausgeführt. Eine Auslagerung würde in diesem Fall den Quellcode verlängern und verkomplizieren.

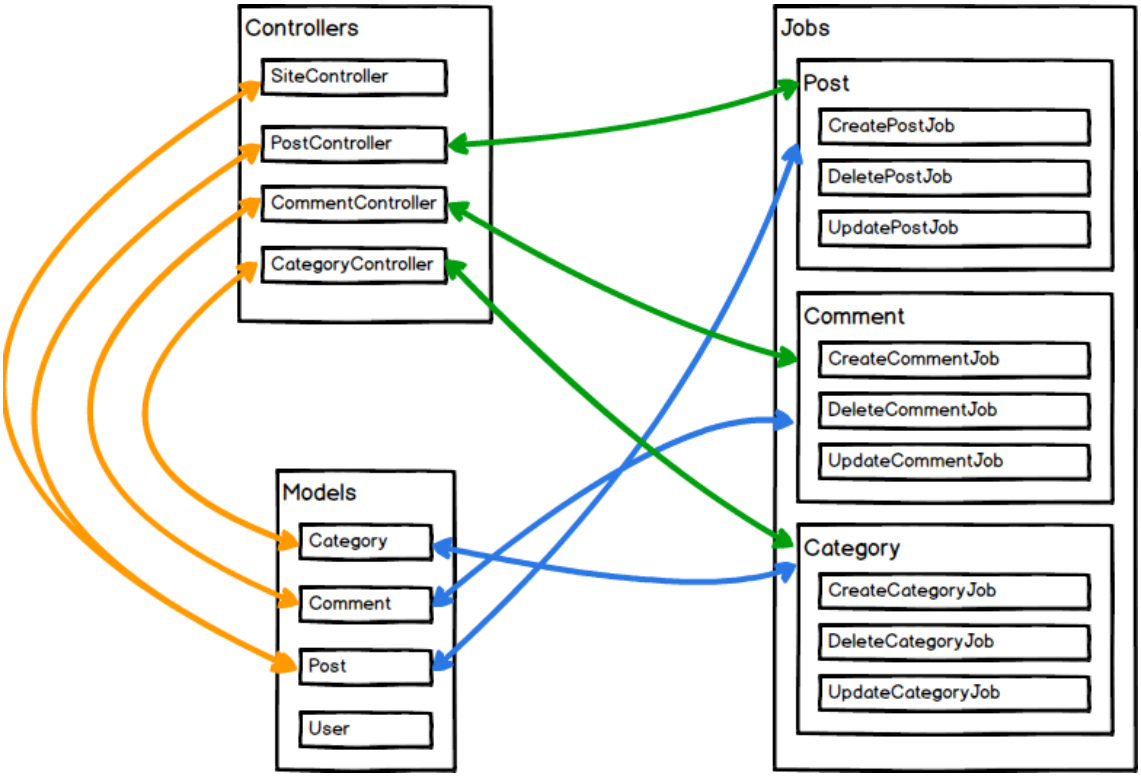


Abbildung 3.5.: Software-Architektur des Nachrichtenportals (eigene Darstellung)

3.1.3. Datenbankschema

Zur Speicherung der Daten bietet Laravel laut Otwell (2018a) Schnittstellen zu vier Datenbanken an. Diese sind MySQL, PostgreSQL, SQLite und SQL Server. In diesem Anwendungsfall wurde MySQL verwendet, da bereits positive Erfahrungen der Datenbank gesammelt werden konnten und die Auswahl der Datenbank in diesem Anwendungsfall kein ausschlaggebendes Kriterium ist.

Die Abbildung 3.6 beschreibt die Datenbankstruktur des Nachrichtenportals. Hauptaugenmerk ist die Tabelle *posts*. Diese enthält die wichtigen Informationen einer Nachricht, wie den Titel, Inhalt und Autor. Jede Nachricht muss mit einem Autor verbunden sein. Ein Autor ist ein Benutzer, der in der Tabelle *users* gespeichert wird und Informationen wie beispielsweise den Namen, die E-Mail und das Passwort enthält. Des Weiteren kann jede Nachricht beliebig viele Kommentare, zu sehen in der Tabelle *comments*, enthalten. Ein Kommentar kann jedoch nur zu einer Nachricht gehören. Außerdem kann eine Nachricht beliebig vielen Kategorien angehörig

sein, welche in der Tabelle *categories* zu finden sind. Andersherum kann eine Kategorie auch beliebig viele Nachrichten enthalten. Die Verbindung wird mit einer *Many-to-Many* Relation realisiert.

Falls ein Nutzer sein Passwort vergisst, kann er dies zurücksetzen. Beim Prozess des Zurücksetzens wird ein Token in der Tabelle *password_resets* generiert, welcher, wie auch die E-Mail, Voraussetzung für das Setzen eines neuen Passwortes ist. Die Tabelle *migrations* enthält eine Liste von *Migrationen*, die bereits durchgeführt wurden. Eine Migration ist gemäß Otwell (2018b) eine Art Versionskontrolle der Datenbank. Sie speichert alle Änderungen der Datenbankstruktur, damit diese für alle Entwickler nachvollziehbar sind und das ganze Entwicklerteam die Struktur nachbauen kann.

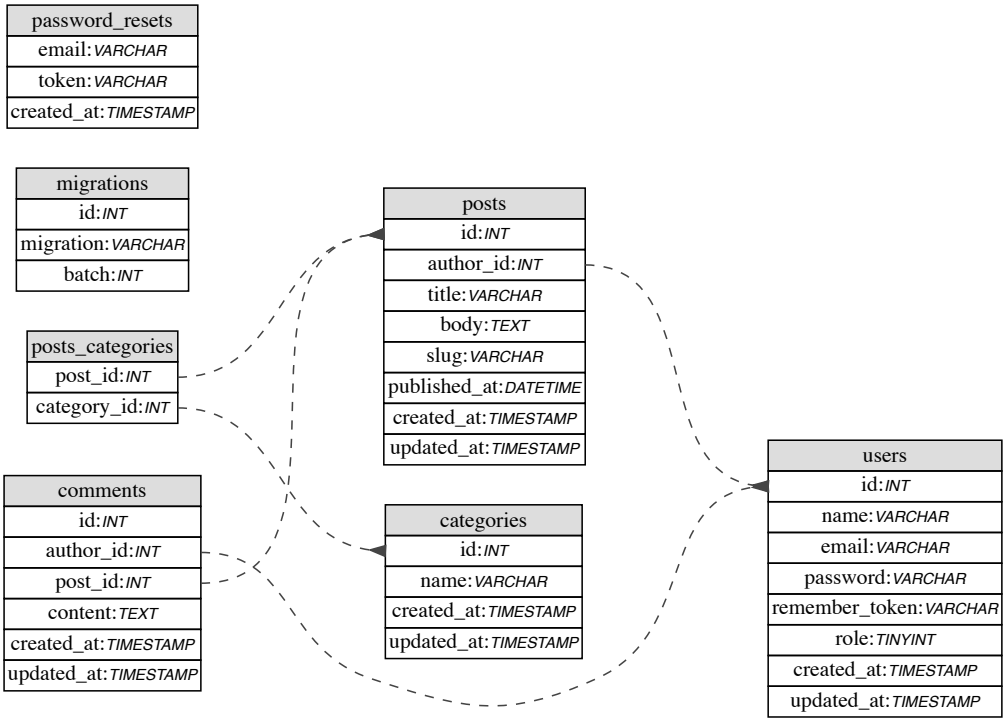


Abbildung 3.6.: ER-Diagramm des Nachrichtenportals (eigene Darstellung)

3.2. Entwicklung eines Videoportals

Wie auch beim Nachrichtenportal sollen die Anforderungen des Videoportals erarbeitet und mittels Mockups veranschaulicht werden. Hauptbestandteil des Portals

ist das Abspielen und Durchstöbern von auf diesem Portal hochgeladenen Videos. Die Startseite des Videoportals listet, wie in Abbildung B.1 zu sehen, die aktuellsten Videos chronologisch, nach Datum absteigend sortiert, auf. Jedes Video-Objekt enthält dabei einen Titel, welches auf 255 Zeichen beschränkt ist, eine Videodatei und eine Beschreibung, die auf 4000 Zeichen beschränkt ist. Die Eingrenzung der Zeichen soll den Ersteller zwingen, die Beschreibung präzise zu formulieren, da der Fokus auf den Videoinhalten liegt. Um die Übersicht der Startseite zu wahren, werden auf der Seite nur die Titel und das Video selbst angezeigt. Die Detailansicht eines Videos ist über den Button *Read more* zu erreichen. Wie in Abbildung B.2 zu sehen, soll in dieser Ansicht zusätzlich die Beschreibung des Videos angezeigt werden. Additional soll der Nutzer die Möglichkeit haben, Videos nach Kategorien zu filtern, siehe Abbildung B.1. Wie auch im Nachrichtenportal, soll es elementare Nutzerfunktionen, wie das Erstellen eines Nutzerkontos, geben. Durch die Erstellung eines Nutzerkontos soll der entsprechende Nutzer in der Lage sein, wie in der Abbildung B.2 zu sehen, Kommentare zu den Videos zu veröffentlichen. Im Gegensatz zum normalen Nutzerkonto existiert auch hier eine Administratorrolle, welches die Privilegien besitzt, Videos und Kommentare zu bearbeiten und zu löschen, was dem normalen Nutzern untersagt ist.

3.2.1. Aufwand

Wie auch im Nachrichtenportal wurde der Aufwand der Entwicklung des Videoportals protokolliert. Die Tabelle 3.2 zeigt, wie auch in Kapitel 3.1, eine Übersicht der Funktionsgruppen und deren Aufwand in Arbeitstagen an. Angesichts der Ähnlichkeit der beiden Portale unterscheidet sich die Tabelle kaum im Vergleich zu der Aufwandstabelle 3.1. Die Implementierung gleicher Meilensteine hat in der Entwicklung des Videoportals weniger Arbeit in Anspruch genommen, da einige Funktionen aus dem Nachrichtenportal übernommen werden konnten. Da die Entwicklung des Videoportals auf Grundlage neuer Erfahrungen implementiert wurde, enthält die Tabelle 3.2 die Arbeitstage des tatsächlichen Aufwands in Klammern, sowie den theoretischen Aufwand, bei identischem Vorwissen, des Nachrichtenportals. Dadurch

soll die Entwicklung der Portale mit der Entwicklung der Migrationen leichter verglichen werden. Die tatsächliche Gesamtarbeitszeit beträgt hier 7,5 Arbeitstage (60 Stunden), welches eine Ersparnis von 2 Arbeitstagen gegenüber des Nachrichtenportals aufweist.

Meilenstein	**Arbeitstage**
Systemvorbereitung	0.5 (0,25)
Nutzerfunktionen (Registrierung, Authentifizierung)	1,5 (1)
Administrator (Rollenbasierte Nutzer)	0,5 (0,25)
Videos (Erstellen, Lesen, Bearbeiten, Löschen)	2 (1)
Kategorien (Erstellen, Lesen, Bearbeiten, Löschen)	1 (1)
Fehlerbehebungen/Optimierungen	4 (4)
Gesamt	**9,5 (7,5)**

Tabelle 3.2.: Aufwandsprotokoll des Videoportals (Tatsächlicher Aufwand in Klammern) (eigene Darstellung)

3.2.2. Software-Architektur

In diesem Abschnitt wird die Architektur des Videoportals erläutert. Ein Ausschnitt der Software-Architektur bietet die Abbildung B.3. Analog zum Nachrichtenportal wird die Wiederverwendbarkeit und Wartbarkeit großgeschrieben. Für die einzelnen Aufgaben zur Erstellung und Bearbeitung von Datensätzen werden ebenfalls eigenständige *Jobs* erstellt, die von verschiedenen Controllern ausgeführt werden können. Für die Verarbeitung eines Video-Objekts ist der *VideoController*, siehe Abbildung B.3, zuständig. Dieser startet den benötigten Auftrag um die Nutzeranfrage erfolgreich zu bearbeiten. Die Abwicklung der restlichen Funktionen, wie beispielsweise die Verarbeitung von Kommentaren und Kategorien, wird wie im Nachrichtenportal

auch von dem *CommentController* und *CategoryController* bewerkstelligt. Insgesamt entspricht die Software-Architektur des Videoportals nahezu der Architektur des Nachrichtenportals.

3.2.3. Datenbankschema

Das Videoportal verwendet, sowie das Nachrichtenportal eine MySQL Datenbank. Das Schema ist nahezu eine 1:1 Kopie des Nachrichtenportals, siehe Abbildung 3.7. Die Tabelle *posts* wurde durch die Tabelle *videos* ersetzt. In dieser Tabelle wird der Pfad der Videodatei und eine Beschreibung dazu gespeichert. Der Rest des Schemas ist mit dem des Nachrichtenportals nahezu identisch.

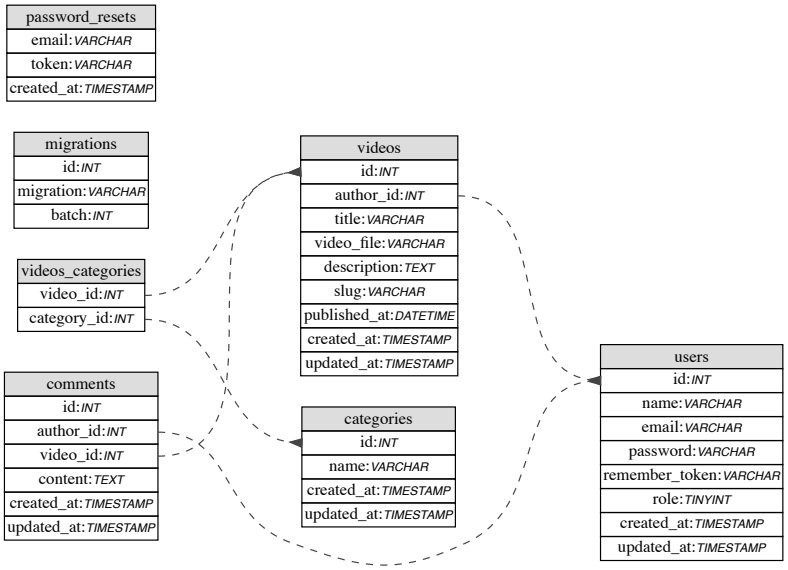


Abbildung 3.7.: ER-Diagramm des Videoportals (eigene Darstellung)

3.3. Migration der entwickelten Web Apps zu Progressiven Web Apps

Die aktuelle Grundlage ist eine Web App, die mit dem PHP Framework Laravel programmiert wurde. Ziel ist es nun, aus den Portalen jeweils eine Progressive Web App zu erstellen. Diese soll die folgenden Eigenschaften bieten:

- Integration der PWA in den Homescreen des Smartphones
- Bereits besuchte Inhalte offline verfügbar zu machen
- Push Benachrichtigung bei Veröffentlichung neuer Einträge
- HTTPS

Im Folgenden werden die einzelnen Schritte, um aus eine Website um die zuvor beschriebenen Eigenschaften zu erweitern, erläutert. Gemäß Google (2018a) stellt Google eine hauseigene Plattform zu Verfügung, die eine Sammlung von Schritt für Schritt Anleitungen bietet, um verschiedenste Applikationen zu entwickeln. Unter anderem finden sich dort ebenfalls Anleitungen zur Erstellung von Progressiven Web Apps. Als grobe Hilfestellung dient daher die von Google bereitgestellte Anleitung *Migrate your site to a Progressive Web App*[7] zur Migration von Websites zu einer Progressiven Web App.

3.3.1. Installation im Startbildschirm

Der erste und einfachste Schritt, eine Webseite zu einer Progressiven Web App zu machen, ist die Bereitstellung der Möglichkeit, die Web App in den Startbildschirm zu integrieren. Dazu wird zunächst eine *manifest.json* Datei erstellt. Diese enthält wie zuvor bereits erwähnt, Informationen darüber, wie die zukünftige Progressive Web App auf dem Startbildschirm angezeigt werden soll. Hierzu gehören beispielsweise der Name, die Hintergrundfarben oder die Icons. Im Quellcode 3.1 ist ein Auszug der *manifest.json* Datei aus dem VideoPortal zu sehen. Eine genaue Beschreibung von weiteren möglichen Aspekten einer *manifest.json* Datei wurde bereits in Kapitel 2.2.2 durchgeführt.

[7] https://codelabs.developers.google.com/codelabs/migrate-to-progressive-web-apps

```
{
  "name": "VideoPortal",
  "short_name": "VideoPortal",
  "start_url": "/",
  "display": "standalone",
  "background_color": "#f5f8fa",
  "theme_color": "#f5f8fa",
  "orientation": "portrait",
  "description": "A simple VideoPortal app.",
  "icons": [{
    "src": "images/touch/icon48.png",
    "sizes": "48x48",
    "type": "image/png"
  }, { ... } ]
}
```

Quellcode 3.1: Manifest.json aus der VideoPortal PWA

Diese Datei wird in diesem Fall in den *public* Ordner angelegt, da dies das Root-Verzeichnis beim Aufrufen der Webseite ist. Zum Schluss wird die *manifest.json* Datei, wie im Quellcode 3.2 zu sehen, im *<head>*-Bereich des HTML-Quellcodes geladen.

```
<!DOCTYPE html>
<html>
<head>
  ...
  <link rel="manifest" href="/manifest.json" />
</head>
<body>
  ...
</body>
</html>
```

Quellcode 3.2: HTML-Auszug um eine manifest.json Datei zu laden

3.3.2. Offline Support

Ein sehr wichtiges Merkmal von Progressiven Web Apps ist die Bereitstellung der Inhalte, auch wenn das Gerät offline ist. Wie in Kapitel 2.2.2 beschrieben, wird die Offline-Unterstützung von *Service Workern* realisiert, die im Hintergrund ihren Dienst erweisen. *Service Worker* können auf verschiedene Art und Weise implementiert werden. Auf der einen Seite können sie manuell und auf der anderen Seite mit Hilfe von Werkzeugen, wie Workbox[8], entwickelt werden. Laut Google (2018d) ist *Workbox* eine Sammlung an Bibliotheken, die den Prozess der Entwicklung von optimalen Service Workern vereinfachen soll. Workbox bietet dem Entwickler verschiedene Konfigurationen an und generiert aus diesen Einstellungen einen Service Worker. Beispielsweise kann die Art und Weise des Cachings damit eingerichtet werden. In diesen Anwendungsfällen wurde die Verwendung von *Workbox* herangezogen, da gemäß Google (2018d) eine manuelle Entwicklung von Service Workern nicht zu empfehlen ist.

Da Laravel den Bundler *webpack*[9] verwendet, um JavaScript-Dateien zusammenzufügen, ist es, gemäß Google (2018d) sinnvoll, mit Hilfe des Bundlers Workbox zu

[8] https://developers.google.com/web/tools/workbox
[9] https://webpack.js.org/

implementieren. Im ersten Schritt wird das Workbox-Plugin in die Web App integriert. Dazu wird in der *webpack.mix.js* das Workbox-Plugin geladen, siehe Quellcode 3.3.

```
new workboxPlugin({
  globDirectory: "/public",
  globPatterns:
  ↪ ['**/*.{css,eot,svg,ttf,woff,woff2,js,html,mp4}'],
  globIgnores: ['**\/mix-manifest\.json', '**\/manifest\.json',
  ↪ '**\/service-worker\.js', '**\/\.map$'],
  swSrc: './resources/assets/js/service-worker.js',
  swDest: path.join("/public", 'service-worker.js')
})
```

Quellcode 3.3: Auszug aus der webpack.mix.js Datei des PWA Videoportals

Die in JSON-Format geschriebenen Key-Value Paare spiegeln die Konfigurationsmöglichkeiten des Workbox-Plugins wieder. *GlobDirectory* enthält den Pfad, dessen Dateien in den Cache gespeichert werden sollen. *GlobPatterns* besteht aus einem Suchmuster, anhand dessen nach Dateien im *globDirectory* Ordner gesucht wird. Alle gefundenen Dateien werden in eine Liste eingefügt, bei der das Precaching angewendet werden soll. Gemäß Google (2018d) werden alle Dateien in dieser Liste im Hintergrund automatisch in den Cache des Browsers geladen. Diese sind im Anschluss somit sofort offline verfügbar. Die *globIgnores* Variable enthält eine Liste von Dateien, die nicht in die Precaching-Liste hinzugefügt werden sollen. In *swSrc* befindet sich der Pfad von benutzerdefiniertem Code, der in den Service Worker integriert werden soll. Beispielsweise wäre es sinnvoll, eine Push-Integration in dieser Datei auszulagern, da das Workbox-Plugin ausschließlich das Caching verarbeitet. Im letzten Schritt wird der Wert von *swDest* erwartet. Dieser enthält den Pfad der generierten Service Worker Datei, die aus dem benutzerdefinierten und automatisch generierten Code besteht. Als Ausgabedatei kommt ein voll funktionsfähiger Ser-

vice Worker heraus, welcher im Anhang C.1 zu sehen ist. Damit ist das Precaching fertiggestellt.

Zusätzlich ist es noch sinnvoll, ein Laufzeit-Caching zu implementieren. Dabei werden die Seiten im Cache gespeichert, sobald der Benutzer die Seite angesteuert hat. Befindet sich der Benutzer im Offline-Modus, soll die Seite aus dem Cache geladen werden. Falls sich die Seite nicht im Cache befindet, wird eine Offline-Seite angezeigt, welche durch das Precaching bereits im Cache gespeichert wurde. Die Offline-Seite weist den Nutzer darauf hin, dass er sich im Offline-Modus und die Seite sich nicht im Cache befindet. Die Realisierung ist durch die Verwendung des Workbox-Plugins vereinfacht worden. Ein Beispiel ist im Auszug 3.4 zu sehen.

```
1  var networkFirstHandler = workbox.strategies.networkFirst({
2    cacheName: 'default-cache'
3  });
4  workbox.router.registerRoute(
5    /.*/,
6    function(args) {
7      return
       ↪  networkFirstHandler.handle(args).then(function(response)
       ↪  {
8        if (typeof response == 'undefined') {
9          // Offline
10         return caches.match('offline');
11       }
12      return response;
13    });
14   }
15   );
```

Quellcode 3.4: Beispielimplementierung des Laufzeitcachings

Im ersten Abschnitt wird ein Cache-Speicher festgelegt und die Strategie dazu definiert. In diesem Beispiel wird der *Network-First* Ansatz verfolgt. Dabei wird im ersten Schritt versucht, die Seite aus dem Internet zu laden. Falls dies nicht erfolgreich ist, wird die Seite aus dem Cache geladen.
Der zweite Abschnitt verarbeitet den Fall, dass die Seite nicht im Cache vorhanden ist. Tritt dieser Fall ein, wird die Offline-Seite angezeigt. Mit den oben aufgeführten Schritten ist der Nutzer nun in der Lage, die Progressive Web App im Offline-Modus zu bedienen, sofern die Seiten sich im Cache befinden.
Abschließend muss der Service Worker noch im Browser registriert werden. Der Code für die Registrierung wird in eine JavaScript Datei eingefügt, die sich im *<body>*-Tag befindet. Der folgende Code überprüft, ob der Browser die Funktionen des Service Workers unterstützt. Ist dies der Fall, installiert der Browser den Service Worker aus der *service-worker.js* Datei.

```
1 navigator.serviceWorker &&
  ↪ navigator.serviceWorker.register('/service-worker.js')
2   .then(function(registration) {
3     console.log('SW registered: ', registration);
4   }).catch(registrationError => {
5     console.log('SW failed: ', registrationError);
6   });
```

Quellcode 3.5: Beispiel-Code zur Registrierung eines Service Workers

3.3.3. Push Benachrichtigungen

Ein weiterer wichtiger Bestandteil der Progessiven Web App sind die Push Benachrichtigungen. Die Funktion der Push Benachrichtigung besteht aus der Verwaltung aus Client- und Serversicht. Die Serversicht wird aus Gründen der Komplexität, des Aufwands und der Empfehlung von Google (2018c) nicht im Detail erläutert. Das bewährte Verfahren ist die Verwendung von Push API Services, die die Serversicht

verwalten. Aus Clientsicht verwaltet laut Google (2018c) jeder Browser die technologischen Details der Push Benachrichtigungen auf seine eigene Art und Weise - der Ablauf ist jedoch immer der gleiche. Der Benutzer wird vom Browser gefragt, ob dieser Push Benachrichtigungen von der Seite zulassen möchte. Bei Zustimmung wird ein Subscription-Objekt erstellt. Dieses Objekt besteht aus einer Endpoint-URL und einem Schlüssel. Ein Beispielobjekt ist im Auszug 3.6 zu sehen.

```
1 {
2   "endpoint":
    ↪  "https://fcm.googleapis.com/fcm/send/d6uHLHjVbMg:APAb...",
3   "key": "BNAIZfrPYGN/9VJ/Cc+o9wCcdW...",
4   "authSecret":"6q5sybcqtZZOIWLrt9VeOg=="
5 }
```

Quellcode 3.6: Beispiel eines Subscription-Objektes

Mit der Endpoint-URL ist es möglich, den User zu identifizieren. Nachdem das Objekt generiert wurde, sendet der Client das Objekt an den Server. Der Server speichert diese Informationen in seinem Speicher. Soll eine Push Benachrichtigung an den Nutzer geschickt werden, müssen zuerst die Daten vom Server erstellt werden, wie zum Beispiel Titel und Text der Push Benachrichtigung. Daraufhin verschlüsselt der Server die Daten mit dem Schlüssel aus dem Subscription-Objekt und sendet das Resultat an die Endpoint-URL. Anschließend wird die Nachricht mit Hilfe der API-Services an das Gerät des Benutzers übermittelt. Das Gerät nimmt die Nachricht an und zeigt es dem Benutzer auf dem Bildschirm an.
Die Implementierung der Push Benachrichtigungen folgt genau diesem Schema. Im JavaScript-Teil wird mittels *pushManager.getSubscription()* überprüft, ob der Browser Push Benachrichtigungen unterstützt und fragt den Benutzer nach dessen Erlaubnis, diese zu versenden. Falls der Benutzer zustimmt, wird mit der Methode *pushManager.subscribe* ein Subscription-Objekt erzeugt (die detaillierte Erzeugung des Subscription-Objektes kann im Anhang C.2 nachgesehen werden). Anschließend

wird via AJAX das Objekt an den Server über eine API-Schnittstelle übergeben. Der Server speichert die Informationen in der MySQL-Datenbank. Sobald ein neuer Beitrag oder ein neues Video im System hochgeladen wird, sendet der Server an alle in der Datenbank befindlichen Subscription-Objekte eine Push Benachrichtigung. Dazu wird der Inhalt mit dem Schlüssel in der Datenbank verschlüsselt und via HTTPS an die Endpoint-URL übergeben. Anschließend erhält der Benutzer eine Benachrichtigung auf seinem Gerät.

3.3.4. HTTPS Unterstützung

Damit die Progressive Web App im Browser auch als eine PWA wahrgenommen wird, fehlt noch, wie in Kapitel 2.2.2 beschrieben, die HTTPS-Unterstützung. Ein sehr simpler Weg die HTTPS-Unterstützung zu realisieren ist die Verwendung eines Reverse-Proxies, wie https://cloudflare.com. Bei Verwendung von Cloudflare dienen ihre Server gemäß Cloudflare Inc (2018) als Proxy. Bei HTTPS-Anfragen wird somit zuerst der Cloudflare-Server angesteuert, welcher die Anfrage an den Server weiterleitet, der die Portale betreibt. Da die Cloudflare-Server ein SSL-Zertifikat verwenden und als Proxy dienen, erhalten die entwickelten Portale ebenfalls aus der Sicht des Endbenutzers eine HTTPS-Unterstützung.
Alternativ kann eine HTTPS-Unterstützung durch die Installation eines SSL-Zertifikates ebenfalls erreicht werden.

3.3.5. Protokollierung des Aufwands

Da aus Unternehmenssicht der Aufwand eine wichtige Rolle spielt, wird auch der Aufwand der Migration, wie in Kapitel 3.1, protokolliert. Die Aufwandstabelle 3.3 zeigt die Arbeit der Migration (detailliert aufgeteilt nach Eigenschaften) und die benötigten Arbeitstage an.

PWA Eigenschaft	**Arbeitstage**
SSL Unterstützung	0,5
Installation in den Startbildschirm	2
Offline Support	6
Push Benachrichtigung	5
Fehlerbehebungen	3
Gesamt	**16,5**

Tabelle 3.3.: Aufwandsprotokoll der Migration

Insgesamt hat die Migration 16,5 Arbeitstage (132 Stunden) benötigt. Im Vergleich zur reinen Implementation der Portale hat die Migration um knapp die Hälfte mehr Zeit erfordert als die Erstellung eines Portals.

3.3.6. Änderungen der Architektur

Mit der Implementierung der Features zur Umwandlung in eine Progressive Web App haben sich einige Änderungen in der Software-Architektur ergeben. In erster Linie sind der *Service Worker* und die *manifest.json* Datei hinzugekommen, die von dem Browser anschließend geladen und verarbeitet werden. Auf der Serverseite wurde eine neue Tabelle namens *subscriptions* hinzugefügt, siehe Anhang D.1, welche Informationen zu den Subscription-Objekten enthält. Zusätzlich wurde ein neuer Job angelegt, siehe Anhang D.2, der eine Push-Benachrichtigung an die Benutzer versendet. Grundsätzlich lässt sich somit sagen, dass mit ein paar Veränderungen der Software-Architektur erste Features einer Progressiven Web App implementiert werden können. Nähere Informationen zur Einschätzung des Aufwands werden im nächsten Kapitel 4 erläutert.

4. Evaluierung

In diesem Kapitel wird die Technologie der Progressiven Web Apps evaluiert. Hierbei soll vor allem geklärt werden, ob sich zum jetzigen Zeitpunkt die Erstellung einer Progressiven Web App als Alternative zur einer Web App oder als Ersatz zu einer Nativen App für Unternehmen lohnen kann. Von Bedeutung sind dabei auch die Änderungen und Auswirkungen, die sich hierbei für die Kunden des Unternehmens, bzw. der Nutzer der PWA ergeben.
Dafür werden im ersten Teil dieses Kapitels auf Basis der aktuell verfügbaren Literatur die Funktionen und Eigenschaften der verschiedenen in Kapitel 2.1 vorgestellten Entwicklungsansätze untereinander verglichen und evaluiert. Im darauffolgenden Kapitel 4.2 folgt eine Bewertung der Aufwände der Entwicklung bzw. Migration der erstellten Web Apps zu Progressiven Web Apps. Des Weiteren werden die bei der Migration auftretenden Schwierigkeiten, welche im Kapitel 3 beschrieben wurden, beurteilt. Weiter wird aufgeführt, durch welche verfügbaren Programme und Werkzeuge die Schwierigkeiten und Aufwände der PWA Entwicklung verringert werden können. Außerdem wird anhand des entwickelten News- und Videoportals untersucht, inwieweit sich eine PWA im Vergleich zur Web App in den Bereichen Offline Funktionalität, natives Verhalten und Performance als vorteilhaft erweist. Zusätzlich wird darauffolgend ein Einblick in die in der Praxis durch eine PWA Entwicklung erzielten Verbesserungen gewährt.
Auf Grundlage dieser Ausarbeitungen wird anschließend argumentiert, wann und ob sich zum jetzigen Zeitpunkt die Entwicklung einer Progressive Web App für Unternehmen und Nutzer lohnen kann. Dabei wird auch zwischen kleinen und großen Unternehmen unterschieden um herauszufinden, ob je nach Unternehmensgröße und Zweck durchaus verschiedene Gründe für die Entscheidung zwischen den Entwicklungsansätzen von Bedeutung sind.

4.1. Vergleich zwischen Web App, Nativer App und PWA auf Basis der verwendeten Literatur

Der Vergleich zwischen Web Apps, Nativen Apps und Progressiven Web Apps wird hier in sechs verschiedene Bereiche gegliedert, die sich bei der Literaturrecherche als sinnvolle Einteilung ergeben. Dabei werden in jedem Bereich die für die Evaluation als interessant und wichtig eingestuften Eigenschaften und Umstände aufgeführt und erläutert.

- Funktionalität
- Sicherheit
- Entwicklung
- Energieverbrauch
- Performance & Usability
- Monetarisierung

Im Bereich der Entwicklung werden in diesem Abschnitt nur allgemeine Aspekte betrachtet. Später folgt in Kapitel 4.2 eine genauere Aufwandsbetrachtung anhand der eigens entwickelten Anwendungsfälle aus Kapitel 3. Dabei werden vor allem die Aufwände der Migration von Web App zu Progressive Web App betrachtet und die auftretenden Schwierigkeiten bewertet.

4.1.1. Funktionalität

Ein für viele Entwickler und Nutzer wichtiger Entscheidungsbereich zwischen Nativer App, Web App und Progressiver Web App ist der Vergleich des gebotenen Funktionsumfangs. Denn eine Verbesserung der Funktionalität kann dazu motivieren, sich mit einem neuen Entwicklungsansatz zu beschäftigen und somit von bereits bekannten Entwicklungsansätzen abzuweichen. Dieser Vergleich der Funktionalität auf Basis der betrachteten Literatur wird im Folgenden durchgeführt. Tabelle 4.1 zeigt hierfür einen Überblick der für den Vergleich hinzugezogenen Eigenschaften und Funktionen.

Eigenschaft / Ansatz	Native App	PWA	Web App
Push Benachrichtigungen	ja	ja	nein
Gute Offline Funktionalität	ja	ja	nein
Nutzung aller Gerätefunkionen	ja	nein	nein
Suchmaschinenindizierung	nein	ja	ja
Aufruf über URL, Verknüpfbarkeit	nein	ja	ja
Verfügbarkeit auf Desktop Systemen	nein	ja	ja
Updateinstallation durch Nutzer nötig	ja	nein	nein
Auf Homescreen installierbar	ja	ja	nein
Nutzung von Wearables etc. möglich	ja	nein	nein

Tabelle 4.1.: Vergleich der Funktionalität der Entwicklungsansätze.

Ein in der Webentwicklung oft vermisstes Feature von Web Apps ist die Möglichkeit von direkten Push Benachrichtigungen auf das Smartphone des Nutzers, auch wenn das Browserfenster nicht geöffnet ist. „If you ask a room of developers what mobile device features are missing from the web, push notifications are always high on the list“ (Medley, 2018). Zwar sind laut Steyer (2016) bei einigen Browsern Benachrichtigungen innerhalb des aktiv geöffneten Browserfensters bereits seit einiger Zeit möglich, die direkten Push Mitteilungen außerhalb der geöffneten App bisher jedoch nicht. Nun ist dieses Feature durch den Einsatz einer Progressive Web App auch mit Webtechnologien realisierbar und nicht mehr nur auf Native Apps begrenzt. „Progressive Web Apps [...] can even re-engage users with web push notifications“ (Google, 2016b). Somit haben Native Apps bezüglich der Push Funktionalität keinen Vorteil mehr gegenüber einer auf Webtechnologien basierenden Progressiven Web App. „Progressive web app notifications have a completely native feel and are indistinguishable from native app notifications“ (Ater, 2017, S. 6). Weiter beschreibt Ater (2017), dass genauso wie bei den Nativen Apps erst die Berechtigung hierfür beim User eingeholt werden muss. Web Apps können diese Funktionalität unter an-

derem aufgrund der fehlenden Service Worker nicht bieten und sind hierbei somit deutlich im Nachteil. Geprüft werden die Push Eigenschaften auch später am entwickelten Newsportal in Kapitel 4.3.2.

Ein weiterer wichtiger Aspekt ist die Offlinefähigkeit der Anwendungen, bzw. eine bessere Nutzung der Anwendung im Falle einer nicht dauerhaft zur Verfügung stehenden Internetverbindung. Bei Nativen Apps ist laut dem 1&1-DigitalGuide (2017b) meistens keine aktive Internetverbindung notwendig. Denn der Code der Anwendung wird bei der Installation geladen und ist danach auf dem Gerät dauerhaft gespeichert. Beim Start der App muss der Programmcode nicht erneut heruntergeladen werden. Jedoch wird für Interaktionen mit dem Server trotzdem eine Internetverbindung benötigt. Wie oft dies nötig ist, hängt von der Art der Anwendung ab. Bei der bisherigen Nutzung von Webtechnologien hingegen wird gerade beim Start und der aktiven Nutzung eine Internetverbindung benötigt, um Code und Inhalt der Seite zu laden. Weiter wird im 1&1-DigitalGuide (2017b) beschrieben, dass durch die Verwendung von HTML5 bei Web Apps diese lokal zwischengespeichert werden können, wodurch eine Art Offline Verwendung ermöglicht wird. Dies ist allerdings nicht sehr oft der Fall und beschränkt sich auf die Funktionalität, die aktuell geladen wurde. „[...] unlike native apps, our web apps remained completely dependent on the server. Any disruption in connectivity would cause the app to fail completely“ (Ater, 2017, S. 64). Bei Progressiven Web Apps wird durch die Verwendung der Service Worker versucht, die eingeschränkte Offline Fähigkeit der Web Apps und Websites auszubessern. „[...] we use a Service Worker to enable temporary offline browsing and near-instant loading on repeat visits, regardless of the network conditions“ (Gallagher, 2017). Die Offline Fähigkeit bedeutet laut Griffith (2017) und Mozilla-WADI-initiative (2016) vor allem auch, dass dem Nutzer keine Fehlerseiten mehr angezeigt werden. „No more blank connectivity error pages nor dinosaurs running through the desert“ (Mozilla-WADI-initiative, 2016). Griffith (2017) beschreibt, dass die PWA in diesem Fall einen angepassten Text anzeigen kann. Bei einem solchen Text kann z.B. darauf hingewiesen werden, dass bei erneut zur Verfügung stehender Internetverbindung eine Aktualisierung erfolgt und

Benutzeraktionen bis dahin auch weiterhin durchgeführt werden können. „A modern messaging PWA can cache its interface and logic locally [...] [and] then display the full interface along with the last cached content to the user. [T]he content may be a bit stale (and it is important to communicate this to the user), but it can still be useful“ (Ater, 2017, S. 65). Aktionen, die während einer Offline Phase vom Nutzer getätigt wurden, könnten dann bei erneuter Internetverfügbarkeit ausgeführt werden. Somit haben die PWAs hier einen Vorteil gegenüber den Web Apps und rücken bei der Offline Fähigkeit an die Nativen Apps heran.

Für viele Anwendungen und Einsatzzwecke ist die Interaktion der App mit der Gerätehardware des Endgeräts von großer Bedeutung. Hierzu gehören z.B. der Gerätespeicher, Sensoren, die Kamera oder Kommunikationsschnittstellen wie NFC und Bluetooth. Charland und Leroux (2011) und IBM (2012) beschreiben, dass solch eine Interaktion bei Nativen Apps kein Problem darstellt, da diese, die nötigen Berechtigungen vorausgesetzt, Zugriff auf die APIs des Endgeräts haben und eng mit diesem verbunden sind. Problematischer hingegen ist dies bei Webtechnologien. „Web technology lives in a sandbox, which is also a jail from lower-level APIs that native code can access—APIs that can access device storage, sensors, and data“ (Charland und Leroux, 2011). Dieser mangelnde Zugriff ist ein großer Nachteil von Web Apps. Allerdings weisen Charland und Leroux (2011) darauf hin, dass immer mehr APIs auch über Browser verfügbar gemacht werden und PWAs diese nutzen. „PWAs have the ability to connect with the operating system (and, thereby, its users) on a deeper level through installation and APIs offering capabilities like notifications, access to the address book, and more“ (Gustafson, 2017b). Weiter erläutert Malavolta (2016), dass abhängig vom verwendeten Browser, auch die Kamera, das Mikrofon und GPS bei PWAs genutzt werden können. Allerdings wird es noch einige Zeit dauern, bis PWAs beim Hardwarezugriff auf demselben Stand wie Native Apps sind, falls dieser Zeitpunkt überhaupt kommen wird. „PWAs have limited capabilities when it comes to offering integration with a smartphone or tablet's hardware features such as NFC [...] functionality as well as the device's sensor package (accelerometer, fingerprint sensor, etc.)“ (Zolciak, 2017). Somit sind die Nativen Apps bei dieser Thematik

gegenüber Webtechnologien trotz der Verbesserungen durch die Progressiven Web Apps und die neuen HTML5 APIs weiterhin deutlich im Vorteil. Eine aktuelle Übersicht der Unterstützung von Gerätefunktionen wurde in Kapitel 2.2.3 aufgeführt.

Auch die Aufrufbarkeit einer App über eine URL und die damit verbundene Indizierung der Anwendung durch Suchmaschinen ist eine nicht zu unterschätzende Eigenschaft. Denn die wenigsten Nutzer kennen für ein Problem direkt den Anbieter einer Lösung und dessen Namen, um direkt die passende App aus einem App Store zu laden. „75% der Internetnutzer sehen Suchmaschinen als zentralen Anlaufpunkt bei der Suche nach Informationen, bestimmten Themen, Produkten und Dienstleistungen“ (Heinrich, 2010, S. 32). Auch Nielsen und Budiu (2013) merken an, dass die Probanden in durchgeführten Studien zur Mobile Usability für neue Aufgaben und Aktivitäten fast immer als Erstes eine Suchmaschine aufgerufen haben. Von Web Apps und Websites ist bekannt, dass diese über eine URL aufgerufen werden und somit von Suchmaschinen indiziert werden. Da Progressive Web Apps aber auch auf Webtechnologien basieren und die Web Apps erweitern, können auch diese über eine URL aufgerufen werden. „Progressive Web Apps are also both discoverable and linkable, meaning that both users and search engines can discover the content within the app“ (Griffith, 2017). Im Gegensatz dazu sind Native Apps, wie zuvor beschrieben, an einen App Store gebunden und können in der Regel nur über diesen installiert werden. Eine App Suche ist zwar auch über die Google Suchmaschine möglich, jedoch wird hierbei nur eine Verlinkung der App Store Ergebnisse erstellt. Durch die Bindung der App an das System und den App Store ist es nicht möglich, Inhalte der Anwendung über eine Suchmaschine zu finden. Auch der Aufruf der App über eine URL und die Verlinkung von Inhalten ist bei Nativen Apps schwieriger und eingeschränkt. Bei Android-Developers (2017a) ist beschrieben, wie sich dies mittels Deep Links realisieren lässt. Hierbei muss aber die jeweilige App schon auf dem Gerät vorhanden sein, damit eine bestimmte Stelle innerhalb der App geöffnet werden kann. Weiter kann über Depp Links laut Android-Developers (2017b) auch eine direkte Verknüpfung zum Store Eintrag der App erstellt werden. An die Auffindbarkeit von Webinhalten kommt dies aber nicht heran. „[...] lack of discove-

rability is one clear disadvantage of native apps“ (Griffith, 2017). Native Apps sind bei dieser Thematik somit durch die Bindung an den App Store im Nachteil.

Zusätzlich zur URL Aufrufbarkeit sollen PWAs in Zukunft auch in einigen App Stores zu finden sein. So erzählt Burtoft (2017), Principal Program Manager von Microsoft, in einem Interview, dass Progressive Web Apps ab dem kommenden Windows 10 Update im Frühjahr 2018 auch im Microsoft Store gelistet werden und über diesen installiert werden können. Somit ist die Installation von PWAs bald nicht mehr nur auf Android Systemen möglich, sondern auch auf Desktop Systemen. Aber auch jetzt schon sind PWAs über den Browser auf Desktop Systemen nutzbar. Auch Google (2017b) schreibt in einer E-Mail an die Chrome Apps Entwickler, dass die Progressiven Web Apps ab Sommer 2018 auch auf Desktop Systemen verfügbar sein werden. Bei Nativen Android Apps hingegen ist die Nutzung auf Desktop Systemen nur mit Hilfe von Tools wie BlueStacks[1] möglich. Web Apps sind genauso wie PWAs auf Desktop Systemen im Browser nutzbar.

Ein weiterer Aspekt ist die Aktualität der Daten bzw. des Codes der App. So ist im 1&1-DigitalGuide (2017b) beschrieben, dass es bei Nativen Apps erforderlich ist, die Anwendung über den jeweiligen App Store zu aktualisieren, um die neueste Version der App zu erhalten. Bei PWAs und Web Apps hingegen ist dies nicht der Fall, da eine Aktualisierung automatisch durch den Anbieter erfolgt und spätestens bei erneutem Aufruf der URL stattfindet. „[...] Updates via App-Store sind nicht notwendig, so wird eine Progressive Web App mit Hilfe eines Service Workers automatisch auf dem neusten Stand gehalten“ (Tandler, 2017). Bei der Aktualität sind die auf Webtechnologien basierenden Entwicklungskonzepte somit im Vorteil gegenüber einer nativen Entwicklung. Durch die Service Worker sind die PWAs hierbei auch den Web Apps überlegen.

Ein anderer wichtiger Punkt ist die Installierbarkeit der Anwendung auf dem Homescreen. Krebs (2016) beschreibt, dass ein Icon der Anwendung auf dem Homescreen

[1] https://www.bluestacks.com/de/index.html

und im App-Drawer bisher nur bei Nativen Apps üblich war. Bei Web Apps und Websites konnte dies laut Liebel (2017b) bisher nur über das Anlegen von Lesezeichen durch den Nutzer teilweise realisiert werden. Somit „[...] konnten mobile Web-Apps nicht installiert werden wie Apps und hatten daher auch kein Icon auf dem Homescreen“ (Krebs, 2016). Ein natives Feeling können die Web Apps so beim Nutzer nicht herbeirufen. Mit Progressiven Web Apps ändert sich dies nun. So führt Google (2016b) auf, dass sich PWAs auf den Homescreen des Nutzers ablegen lassen und somit immer gut erreichbar sind, auch ohne den Umweg über einen zuvor aufgerufenen Browser. Hierdurch stehen die PWAs den Nativen Apps in dieser Thematik in Nichts nach und lassen sich wie Native Apps über ein Icon aufrufen. „Progressive web apps launched from the homescreen can have an entirely native, app-like look. They can have a splash screen as they are loading. They can launch in full-screen mode, without the browser and phone UI around them“ (Ater, 2017, S. 8). Web Apps hingegen sind bei der Installierbarkeit den Nativen Apps und PWAs deutlich unterlegen und können dem Nutzer kein natives Feeling bieten. Ein dazu passender Vergleich wird in Kapitel 4.3 durchgeführt.

Neben dem oben genannten Aspekt der unterschiedlichen Hardwareunterstützung, ist als Folge daraus auch die Nutzung von Wearables, wie z.B. Smartwatches oder Fitess-Trackern, nicht bei allen Entwicklungsansätzen gegeben. Da Native Apps, wie zuvor beschrieben, keine Einschränkungen in Bezug auf die Nutzung der Sensoren und anderer Gerätehardware haben, sind Smartwatches und sonstige Accessories bei diesen nutzbar. Bei der Nutzung von Web Apps und Progressiven Web Apps hingegen muss z.B. auf Smartwatches verzichtet werden. „[...] limited [...] integration with a smartphone or tablet's hardware features [...] prevents PWAs from being developed for use with mobile accessories and „wearables“ [...]“ (Zolciak, 2017). Beim jetzigen Entwicklungsstand der PWAs entfallen somit Entwicklungkonzepte auf Basis von Webtechnologien, falls die Anwendung mit Wearables interagieren soll. Hierbei sind die Nativen Apps derzeit noch deutlich im Vorteil.

4.1.2. Sicherheit

Das Thema Sicherheit rückt in der heutigen Zeit immer mehr in den Fokus und wird deshalb auch hier in die Diskussion mit einbezogen. Betrachtet werden keine in bestimmten Technologien existierenden Sicherheitslücken, sondern allgemeine Sicherheitsaspekte, die bei der Recherche zu dieser Arbeit als erwähnenswert eingestuft wurden.

In Tabelle 4.2 sind die in dieser Diskussion behandelten Aspekte aufgeführt. Einer der momentan noch größten Nachteile der PWAs und Web Apps gegenüber Nativen Apps ist, wie bereits zuvor angesprochen, der mangelnde Zugriff auf bestimmte Hardwareteile des Smartphones. Zu diesen Hardwareteilen können auch Bauteile gehören, die die Sicherheit verbessern können. Eines dieser Hardwareteile ist in manchen Smartphones zum Beispiel der Fingerabdrucksensor, der nicht nur zum Entsperren des Smartphones, sondern auch zum Login in bestimmte geschützte Anwendungen genutzt werden kann.

Eigenschaft \ Ansatz	Native App	PWA	Web App
Zugriff auf Sicherheitshardware	ja	nein	nein
HTTPS Pflicht	nein	ja	nein
Datenspeicherung im Browserspeicher	nein	ja	ja
Datenspeicherung im Gerätespeicher	ja	nein	nein
Sicherheit durch App Store	ja	nein	nein
Sicherheitsupdates sofort vorhanden	nein	ja	ja

Tabelle 4.2.: Vergleich der Sicherheit der Entwicklungsansätze.

Zum Beispiel nutzen einige Hersteller von Überwachungskameras den Fingerabdruck als Login in die App, um auf die Systeme zuzugreifen. „If you are using a mobile device that supports a Touch ID® fingerprint identity sensor, you are prompted to

enable Touch ID or skip“ (NETGEAR, 2016). Mit einer PWA könnte diese Funktion derzeit also nicht realisiert werden.
Außerdem wird sich auch das Bezahlen mit Smartphone und Fingerabdruck in Zukunft weiterverbreiten. „Fingerprint is the natural way to provide security to the mobile payment. With the near field communication (NFC) technology available, mobile payment will be increased significantly“ (Gao et. al., 2014). Ohne den Zugriff auf den Fingerabdrucksensor und den NFC Chip muss auf diese zusätzliche Sicherheitsprüfung bzw. neue Bezahlmethode verzichtet werden. Ein Login in geschützte Bereiche ist bei PWAs also nur mit Passwort bzw. PIN möglich. Hierdurch herrschen wieder die üblichen durch die Nutzer hervorgerufenen Sicherheitsprobleme von Passwörtern. Der richtige Umgang mit Passwörtern sorgt jedoch auch ohne Fingerabdrucksensor für Sicherheit. Allerdings ist das Entsperren mit Fingerabdruck für private Anwender meist deutlich angenehmer als die Verwendung von Passwörtern, da sich die Nutzer laut Fondsbörse-Deutschland (o.J.) im Zuge der Digitalisierung immer mehr Zugangsdaten merken müssen. „Users prefer using fingerprint [...] or Iris recognition capabilities [...] over account specific passwords“ (Lindemann, 2016). Eine in Zukunft wahrscheinlich auch häufiger vorkommende Entsperrmöglichkeit bei Smartphones und Apps ist die Iris- und Gesichtserkennung, welche unter anderem Apple und Samsung in neueren Modellen einbauen. Durchgeführt wird dies und unter anderem mit Hilfe der Kamera des Smartphones. Da eine PWA Zugriff auf die Kamera bekommen kann, könnten einige dieser auf Gesichtserkennung basierenden Sicherheitsüberprüfungen also auch bei einer PWA genutzt werden. In Bezug auf diese Thematik ist jedoch auch erwähnenswert, dass die Anzahl der Nativen Apps, die mit Fingerabdruck oder Passwort geschützt werden müssen, sehr gering ist. Bei vielen Apps wird auf eine Zugriffsbeschränkung verzichtet, denn dies stört oft die Usability. „The design of any password authentication system requires a tradeoff between security and usability“ (Mehler und Skiena, 2006). Oft ist bei Smartphone Nutzern nur der Zugriff auf das Gerät an sich durch PIN oder Fingerabdruck beschränkt. Somit betrifft dieser Nachteil der PWAs und Web Apps gegenüber den Nativen Apps in Bezug auf Sicherheitssensoren nur einen geringen Teil von Anwendungsfällen.

Ein weiterer Aspekt ist, wie zuvor erwähnt, die HTTPS Pflicht bei Datenübertragungen. Die Nutzung von HTTPs wird im Gegensatz zu den Web Apps und den Nativen Apps bei Progressiven Web Apps vorausgesetzt und sorgt so für mehr Sicherheit beim Transport aller Daten über das Internet. Wichtig ist dies laut Farrugia (2016) vor allem, da die bei PWAs verwendeten Service Worker Netzwerkanfragen unterbrechen können. Um das Mitlesen oder eine Manipulation des Datenverkehrs zu verhindern, wird daher HTTPS vorgeschrieben. Neben dem Sicherheitsaspekt kann es auch noch weitere Gründe für HTTPS geben. So hat die Nutzung von HTTPS laut Ait Bahajji und Illyes (2014) einen positiven Effekt auf das Ranking der Websites in der Google Suchmaschine. Des Weiteren wird von Schechter (2017) im Chromium Blog aufgeführt, dass seit Oktober 2017 verstärkt Websites ohne HTTPS als unsicher markiert werden. Allerdings weisen Selvarajah et. al. (2013) darauf hin, dass Native Apps generell etwas sicherer sind als Web Apps. „Web Apps are generally less secure than Native Apps as they require constant connection with the server where private information could potentially be compromised“ (Selvarajah et. al., 2013). Diese Problematik betrifft die PWAs jedoch nicht, da die oben beschriebenen HTTPS Pflicht besteht und die Verbindung somit geschützt ist.

Auch die Speicherung von Daten ist bei den drei Entwicklungsansätzen unterschiedlich. So wird bei Nativen Apps für die Speicherung der Daten der Gerätespeicher genutzt. Bei Web Apps bzw. Progressiven Web Apps hingegen „[...] ist es erst einmal so, dass die Daten einer PWA irgendwo im Speicher des Browsers liegen und dort verhältnismäßig schlecht geschützt sind“ (Otto, 2017). Inwieweit dies ein Sicherheitsrisiko darstellt, wird hier nicht weiter betrachtet.

Ein weiterer Aspekt ist die Sicherheit durch eine App Store Umgebung. Native Apps werden wie zuvor beschrieben im Gegensatz zu PWAs und Web Apps über einen App Store installiert. In den meisten Fällen ist dies der Google Play Store oder der App Store von Apple. Dies hat den Vorteil, dass dort angebotene Anwendungen durch die Store Betreiber überprüft werden können und somit die Verteilung von Schadcode erschwert wird. Google hat z.B. hierfür den Bouncer entwickelt, „[...] which

provides automated scanning of Android Market for potentially malicious software without disrupting the user experience of Android Market“ (Lockheimer, 2012). In einem Bericht über den Kampf gegen schädliche Apps im Jahr 2017 beschreibt der Product Manager Ahn (2018), dass Google die Verfahren gegen schädliche Apps weiter verbessern konnte und über 700.000 Apps entfernt hat. „In fact, 99% of apps with abusive contents were identified and rejected before anyone could install them“ (Ahn, 2018). Außerdem verlangen Apple und Google die Signierung der Apps mit einem Zertifikat um sicherzustellen, dass die Inhalte der App auch vom Herausgeber stammen und nicht von Fremden. „Code signing your app assures [...] the app hasn't been modified since it was last signed. Before your app can [...] be submitted to the App Store, it must be signed with a certificate issued by Apple“ (Apple, 2017). Des Weiteren müssen sich laut Schonschek (2013) App Entwickler erst registrieren, bevor diese eine App in die App Stores laden können. Bei Google erfolgt dies über das Google Konto und bei Apple über die Apple ID. Außerdem haben Nutzer die Möglichkeit, die Bewertungen anderer Nutzer zu den jeweiligen Apps zu lesen. „Anonyme, nicht nachvollziehbare Bewertungen werden im App Store von Apple dadurch vermieden, dass sich der Nutzer zuerst mit seiner Apple ID anmelden muss. Auch Google Play fordert eine Anmeldung für Erfahrungsberichte über Apps“ (Schonschek, 2013). PWAs und Web Apps hingegen werden über eine URL im Browser aufgerufen und unterliegen somit keiner Kontrollinstanz. Somit können hier manipulierte URLs oder andere Sicherheitslücken im Web eine Gefahr darstellen. Auch als Entwickler ist hier keine Registrierung nötig. In Bezug auf diese Kontrollinstanz sind Native Apps auf Grundlage der hier vorliegenden Argumentation sicherer als PWAs und Web Apps. Eine absolute Sicherheit gibt es jedoch auch in App Stores nicht, denn wie Schirrmacher (2018) berichtet, können auch schädliche Apps den Weg in die App Stores schaffen. „Offensichtlich haben die Sicherheitsmechanismen von Google Play versagt und 27 mit Schadcode verseuchte Spiele-Apps durchgelassen“ (Schirrmacher, 2018).

Eine weitere Eigenschaft ist die Verfügbarkeit von Sicherheitsupdates. Wie bereits zuvor erwähnt, stehen bei Web Apps und PWAs Updates nahezu sofort zur Ver-

fügung. „As there aren't any apps to download [...], PWAs will keep everyone concurrently up to date. As a result, the end user can immediately benefit from the latest updates without having to download the app again“ (Zola, 2017). Bei Nativen Apps hingegen ist in der Regel erst ein App Update über den App Store erforderlich, das viele Nutzer erst nach einiger Zeit oder gar nicht installieren. Des Weiteren ist aus den Daten von Shiny-Development (2018) ersichtlich, dass bei Apple die Freigabe von Updates einige Tage dauern kann. Zwar gab es laut Rusovsky (2017) auch die Möglichkeit von Live Updates von App Store Apps durch bestimmte Dienste, wie z.B. denen von Rollout.io[2]. Jedoch weist Miller (2017) darauf hin, dass Apple damit nicht einverstanden ist, da dieses Vorgehen gegen App Store Richtlinien verstößt. Eine Kontrolle des Updates Inhalts ist so nicht gewährleistet und dies würde die zuvor angesprochene Sicherheit durch App Stores beeinträchtigen. Daher steht diese Möglichkeit nun nicht mehr zur Verfügung. Security Updates können also bei Web Apps und PWAs schneller zur Verfügung gestellt werden als bei Nativen Apps. Dieses Vorgehen steigert die Sicherheit und damit sind auf Webtechnologien basierende Entwicklungen bei der Updategeschwindigkeit im Vorteil.

4.1.3. Entwicklung

Die eigentliche Entwicklung einer App und der damit verbundene Aufwand und weitere Begleitumstände sind wichtige Aspekte bei der Entscheidung zwischen Nativer App, Web App und Progressiver Web App. Im Folgenden werden verschiedene Eigenschaften, die eventuell für die Entscheidung von Bedeutung sind, aufgeführt und erläutert.

Eine wichtige Eigenschaft ist die Möglichkeit der plattformübergreifenden Entwicklung, da diese die Produktivität steigert und weitere Vorteile mit sich bringt. „Cross-platform development approaches [...] [allow] developers to implement their apps in one step for a range of platforms, avoiding repetition and increasing productivity“ (Heitkötter, Hanschke und Majchrzak, 2012). IBM (2012) und Jobe (2013) beschrei-

[2]https://rollout.io

ben, dass dies bei Web Apps der Fall ist und somit viel Zeit und Kosten bei der Entwicklung eingespart werden können. Des Weiteren ermöglichen auch Progressive Web Apps eine plattformübergreifende Entwicklung, denn diese erweitern die Web Apps und basieren ebenso auf Webtechnologien. „The fact that PWAs use web technology means they work cross-platform, on most browsers, taking the operating system out of the equation“ (de Silva, 2017). Bei Nativen Apps hingegen ist keine richtige plattformübergreifende Entwicklung möglich. „[...] code written for one mobile platform cannot be used on another, making the development [...] of native apps for multiple OSs a very long and expensive undertaking“ (Apple, 2017). Aus diesem Grund können sehr viele Ressourcen eingespart werden, wenn als Entwicklungskonzept eine Web App bzw. Progressive Web App gewählt wird.

Eigenschaft / Ansatz	Native App	PWA	Web App
Plattformübergreifende Entwicklung	nein	ja	ja
Plattformübergreifende Wartung	nein	ja	ja
Entwicklung leistungsstarker Apps	ja	teils	teils
Möglicherweise Updatepflicht	ja	nein	nein
Designelemente frei wählbar	nein	ja	ja

Tabelle 4.3.: Vergleich der Entwicklung der Entwicklungsansätze.

Von den vier in Kapitel 2.1.1 aufgeführten Betriebssystemen konnten sich, wie zuvor bereits erwähnt, lediglich iOS und Android im mobilen Markt durchsetzen. Allerdings kann sich eine plattformübergreifende Entwicklung auch schon bei nur ein oder zwei zu unterstützenden Systemen lohnen. „[...] native development is not necessary when implementing mobile applications. Even if only a single platform is to be supported, a cross-platform approach may prove as the most efficient method due to its low entry barriers“ (Heitkötter et. al., 2012). Daher sind die Web Apps und PWAs hier den Nativen Apps deutlich überlegen.

Neben der plattformübergreifenden Entwicklung ist auch die plattformübergreifende Wartung nicht zu vernachlässigen. „[...] code written for one mobile platform cannot be used on another, making the development and maintenance of native apps for multiple OSs a very long and expensive undertaking“ (IBM, 2012). Im Gegensatz dazu ist bei auf Webtechnologien basierenden Entwicklungen die Wartungsarbeit deutlich geringer, ähnlich wie der zuvor beschriebene geringere Entwicklungsaufwand. „[...] developers and webmasters do not have to maintain multiple versions of the app or website to accommodate everyone and users all get an equivalent experience“ (Kintan, 2017). Somit sind auch hier die Nativen Apps im Nachteil gegenüber den Web Apps und Progressiven Web Apps.

Ein weiterer Aspekt ist die Eignung der Ansätze für leistungsstarke Anwendungen, wie z.B. Spiele. Jobe (2013) und Huy et. al. (2012) erläutern, dass Native Apps für hardwareintensive Anwendungen in der Regel die beste Alternative sind, da diese die Nutzung der Gerätehardware vollständig unterstützen und damit die benötigte Rechenleistung abrufen können. „By using native app paradigm, developers can make use the capability of the phone: hardware, built-in APIs, etc to create interesting and sophisticated games“ (Huy et. al., 2012). Jedoch bedeutet dies nicht, dass es unmöglich ist, mit Webtechnologien leistungsstarke Anwendungen für mobile Endgeräte zu erstellen. Für „[...] komplexe Spielerlebnisse eignen sich [PWAs] eher weniger, sind aber selbstverständlich durchaus möglich“ (Tandler, 2017). Eine weitere Betrachtung der Performance folgt in Abschnitt 4.1.5.

Ein anderer Punkt ist eine mögliche Updatepflicht, die der App Store Betreiber bzw. Betriebssystem Hersteller dem Entwickler auferlegen kann. In solch einem Fall muss der Entwickler dann für bereits vorhandene Apps ein Update bereitstellen, das die Neuerungen eines Betriebssystem Updates unterstützt, um weiterhin im Store mit der jeweiligen App präsent zu sein. Dies ist bei Updates zur Verbesserung der Sicherheit sinnvoll. Allerdings führt dies zu zusätzlichem Entwicklungsaufwand und kann so Entwickler unter Druck setzen. So erklärt der Android Produkt Manager Cunningham (2017), dass z.B. bis Ende 2018, alte und neue Apps ein Update be-

kommen müssen, um das momentan neuste API Level zu unterstützen. „[...] Play will require that new apps and app updates target a recent Android API level. This will be required for new apps in August 2018, and for updates to existing apps in November 2018“ (Cunningham, 2017). Des Weiteren zwingt auch Apple die Entwickler durch iOS Updates zur Handlung. So berichtet Becker (2017), dass mit dem Update auf iOS 11 die 32-Bit Unterstützung von Apple gestrichen wird und hierdurch viele Apps bis zu einem Update des Entwicklers nicht mehr nutzbar sind. Dies hat auch einen Datenverlust für in den betroffenen Apps gespeicherten Daten zur Folge und nicht mehr weiter entwickelte Apps werden dauerhaft unbrauchbar. Das ist besonders dann ärgerlich, wenn Nutzer hierfür Geld bezahlt haben. Gerade kleine Entwickler können so Probleme bekommen. Web Apps und Progressive Web Apps müssen zwar auch gelegentlich an neue Browser Versionen angepasst werden, jedoch hätte der Verzicht auf ein Update nicht direkt einen kompletten Stillstand zur Folge, da es oft nur zu Darstellungs- und Funktionsproblemen kommt. Zur Umstellung auf neue Technologien wie HTML5 gibt es laut Braun (2011) z.B. keinen Zwang, denn die Browser können auch weiterhin Websites darstellen, die nicht auf den neusten Entwicklungen aufbauen. Für kleine Unternehmen oder einzelne Entwickler können PWAs und Web Apps somit die bessere Entscheidung sein, falls ein plötzlicher Mehraufwand vermieden werden soll.

Ferner ist die freie Wahl von Designelementen ein Vorteil der webbasierten Anwendungen. „Das Web App Design muss keinen Design-Guidelines a la Apple und Co. entsprechen, sie können optimal auf die Marke abgestimmt werden“ (Koroll, 2016). Somit können Unternehmen das bei den Kunden bekannte Corporate Design des Unternehmens auch im App Bereich umsetzen. Das Corporate Design „[...] sorgt für ein einheitliches Erscheinungsbild und trägt so zur Identität einer Marke bei“ (Siebert, 2017). Native Apps sind daher bei der Freiheit der Design Erstellung im Nachteil.

4.1.4. Energieverbrauch

Der Energieverbrauch wird betrachtet, da dieser im mobilen Bereich aufgrund begrenzter Akkukapazitäten ein nicht zu unterschätzendes Thema ist. „Energy consumption has become an increasingly important topic in software development, especially due to the ubiquity of mobile devices, and the choice of programming language can directly impact battery life“ (Oliveira, Torres, Castor und Ximenes, 2016). Die durchgeführte Literaturrecherche und auch Oliveira et. al. (2016) bestätigen allerdings, dass es bisher wenige Untersuchungen im direkten Vergleich zwischen einem Web basierten Entwicklungsansatz und dem nativen Ansatz gibt. Daher sind keine eindeutigen Aussagen möglich. „To the best of our knowledge, no previous work has analyzed the impact of different app development approaches on energy consumption“ (Oliveira et. al., 2016). Aus diesen Gründen wird dieses Thema hier nur kurz betrachtet, indem zwei für die Diskussion relevante Untersuchungen vorgestellt werden, die sich mit dem Energieverbrauch befassen. Die erste Untersuchung beschäftigt sich mit dem Energieverhalten der Service Worker von PWAs. Bei der zweiten Untersuchung wird analysiert, ob die Nutzung von Javascript oder Java vorteilhafter für den Energieverbrauch des Smartphones ist.

Service Worker Energieverbrauchsuntersuchung Malavolta, Procaccianti, Noorland und Vukmirović (2017) haben untersucht, ob der Einsatz von Service Worker bei PWAs zu einer Änderung im Energieverbrauch der Anwendung führt. Dafür wurden sieben verschiedene PWAs auf einem LG G2 und einem Nexus 6P getestet. Dabei sind Malavolta et. al. (2017) zu dem Ergebnis gekommen, dass die Service Worker keinen Einfluss auf den Energieverbrauch haben. „Our results show that service workers do not have a significant impact over the energy consumption of the two devices, regardless of the network conditions“ (Malavolta et. al., 2017). Da der Einsatz von Service Worker der größte Unterschied zwischen Progressiven Web Apps und Web Apps ist, scheinen somit die Progressiven Web Apps auf Grundlage der Untersuchung von Malavolta et. al. (2017) keinen Vor- oder Nachteil im Energieverbrauch gegenüber den Web Apps zu haben.

Java und JavaScript Energieverbrauchsuntersuchung auf Android Oliveira et. al. (2016) haben untersucht, ob es energieeffizienter ist, auf Android einen webbasierten Entwicklungsansatz oder den nativen Entwicklungsansatz zu verfolgen. „[...] comparing the performance and energy consumption of 33 different benchmarks in the two main programming languages employed in Android development: Java and JavaScript“ (Oliveira et. al., 2016). Javascript steht dabei für einen webbasierten Ansatz und Java für einen nativen Ansatz. Dabei kamen sie zu dem Ergebnis, dass die JavaScript Versionen meistens weniger Energie verbrauchen als die nativen Versionen. Jedoch war dies nicht immer der Fall und somit lässt sich nicht eindeutig bestimmen, ob ein Entwicklungsansatz energieeffizienter ist als der andere. Je nach Anwendungsfall unterscheidet sich dies. JavaScript scheint aber im Durchschnitt effizienter zu sein, zumindest bei den getesteten Benchmark Anwendungen. Laut Oliveira et. al. (2016) muss allerdings beachtet werden, dass Benchmark Anwendungen nicht alle Arten von Anwendungen repräsentieren „and for that reason it is not possible to extrapolate the results for all applications, since applications are usually much more IO intensive“ (Oliveira et. al., 2016).

4.1.5. Performance und Usability

Im Folgenden werden Unterschiede in der Performance und Usability bzw. Nutzererfahrung dargestellt, die aus der genutzten Literatur und den verschiedenen Eigenschaften der drei betrachteten Entwicklungsansätze ersichtlich werden. Ein genauer Vergleich der Performance ist abhängig von den eingesetzten mobilen Endgeräten und kann gerade bei einer PWA oder Web App auch abhängig von der zur Verfügung stehenden Internetverbindung und des Browsers sein. Somit ist die Performance mit Hilfe der Literatur schwer vergleichbar. „Performance is dependent on JavaScript rendering and mobile web browsers“ (Jobe, 2013). Die kaum zur Verfügung stehende Literatur wirkt sich auch auf Entscheidungsschwierigkeiten der Entwickler und Unternehmen bei der Wahl des Entwicklungsansatzes aus. Oliveira et. al. (2016) beschreibt, dass Entwickler kaum Informationen über die verschiedenen Performance-Eigenschaften der unterschiedlichen Entwicklungsansätze haben. In Kapitel 4.3.3

wird zwischen den zwei für diese Arbeit entwickelten Web Apps und PWAs ein Vergleich der Seitenladezeiten durchgeführt. Hierduch soll festgestellt werden, ob sich die Migration von einer bereits vorhandenen Web App zu einer PWA in dem in dieser Arbeit geleisteten Entwicklungsaufwand aus Performance Sicht für Unternehmen lohnen kann.

Wie bereits in Kapitel 4.1.3 angedeutet, sind die Nativen Apps den Web Apps und PWAs im Bereich der Hardware derzeit überlegen und können somit eine bessere Performance in Bezug auf Rechenleistung bieten. „If developers want to make an app that requires accelerated graphic processing (e.g. high-end gaming apps) and access to hardware features, they should develop a native app“ (Huy et. al., 2012). Positiv auf die Performance wirkt sich bei Nativen Apps auch aus, dass Parallelisierung z.B. bei der auf Android genutzten Programmiersprache Java einfacher zu handhaben ist als bei Webtechnologien. „[...] it is easier to leverage parallelism in Java, which contributes positively to the performance of the Java versions of the benchmarks“ (Oliveira et. al., 2016). Bezüglich der nutzbaren Hardware Features verbessert sich die Situation für die Webtechnologien jedoch, da, wie bereits zuvor beschrieben, durch die Progressiven Web Apps und die damit verbundene Nutzung neuer HTML5 APIs ein immer größerer Teile der Gerätehardware nutzbar wird. In der weiteren Entwicklung der PWAs wird sich zeigen, inwieweit dies zu einer besseren Performance bei leistungsstarken Anwendungen führt.

In Bezug auf die benötigte Datenmenge, die über eine Internetverbindung gesendet werden muss, sind die PWAs durch den Einsatz der Service Worker und der App Shell den Web Apps gegenüber deutlich im Vorteil. Ein Vergleich mit Nativen Apps ist schwieriger, da sich die anfallenden Datenmengen während der Nutzung einer Nativen App von Grund auf von der Datenmenge einer PWA unterscheiden. Bei Nativen Apps werden, wie bereits zuvor beschrieben, viele Daten durch die Installation dauerhaft auf dem Gerät gespeichert und müssen so bei der Nutzung der App nicht mehr über das Netzwerk geladen werden. Farrugia (2016) beschreibt, dass durch die Service Worker und dem damit verbundenen Caching bei Progressiven Web Apps

weniger Daten versendet werden müssen als bei den Web Apps.

> „Comparing our website to itself before service workers were implemented, before we were retrieving over 200 KB upon page load; that is now reduced to 13 KB. On a regular 3G network, the page would have taken 3.5 seconds to load; now it takes 500 milliseconds“ (Farrugia, 2016).

Jedoch muss darauf hingewiesen werden, dass es sich bei der in dem Beispiel betrachteten PWA um eine Applikation mit geringerem Funktionsumfang handelt. Des Weiteren hat passend zum Einsatz der Service Worker auch der Einsatz einer App Shell bei PWAs einen positiven Einfluss auf die Performance, bzw. die Wahrnehmung der Performance durch den Nutzer. „Having the shell and the content load separately theoretically improves the user's perception of the performance and usability of the app“ (Lynch, 2016). Dabei muss laut Osmani (2018) die App Shell nicht jedes Mal erneut geladen werden. Weiter beschreibt Osmani (2018), dass dies bei Nativen Apps ähnlich ist. Somit rücken die Webtechnologien bei der Performance durch die PWAs in der Theorie weiter in Richtung Native Apps und erzielen eine Verbesserung gegenüber den Web Apps. Dies führt zu einer besseren Usability, da der Nutzer eine flüssigere Bedienung der App wahrnimmt. Performance Verbesserungen aus der Praxis werden später in Kapitel 4.4 aufgeführt.

Ein weiterer interessanter Aspekt beim Vergleich der Nutzererfahrung ist die Anzahl der vom Nutzer durchzuführenden Schritte bis zur ersten Nutzung der Anwendung. Abbildung 4.1 stellt den Unterschied im Ablauf der Installation grafisch dar.
Bei Nativen Apps muss der Nutzer zuerst den App Store öffnen, um dort nach einer App zu suchen. Anschließend muss diese dann vor der ersten Nutzung heruntergeladen werden. Dies bedeutet für den Nutzer abhängig von der Internetverbindung etwas Wartezeit. Bei der darauffolgenden Installation verstreicht weitere Zeit und der Nutzer muss den Berechtigungen der App zustimmen. Erst dann kann die App genutzt werden. Bis zur ersten Nutzung fallen also fünf Schritte an, die der Nutzer wahrscheinlich gerne vermeiden würde. „Consumers find it tedious having to find

the app in the app store, wait for it to download, and worry that it'll use up all their storage space before they can start using the app“ (de Silva, 2017). Dazu beschreibt Ater (2017), dass dies bei beliebten Apps wie Twitter oder Facebook zwar kein Problem darstellt, bei unbekannten Apps in der Regel aber laut Studien bei jedem Installationsschritt 20% der Nutzer abbrechen. Im Gegensatz dazu fällt bei PWAs und Web Apps nur ein Schritt vor der Nutzung an. Der Nutzer muss nur auf eine URL klicken und kann nach einer kurzen Seitenladezeit die Anwendung über den Browser nutzen. „You can share it on social media, email, text message, online ad, or link it to a QR code, and the user will instantly be immersed into an app“ (de Silva, 2017). Die PWAs sind somit bei der Usability der Installation den Nativen Apps überlegen. Für eine mehrmalige Nutzung der App kann die PWA, wie bereits zuvor beschrieben, auch auf dem Homescreen abgelegt werden und ist somit genauso wie die Nativen Apps über einen Klick auf das Icon aufrufbar.

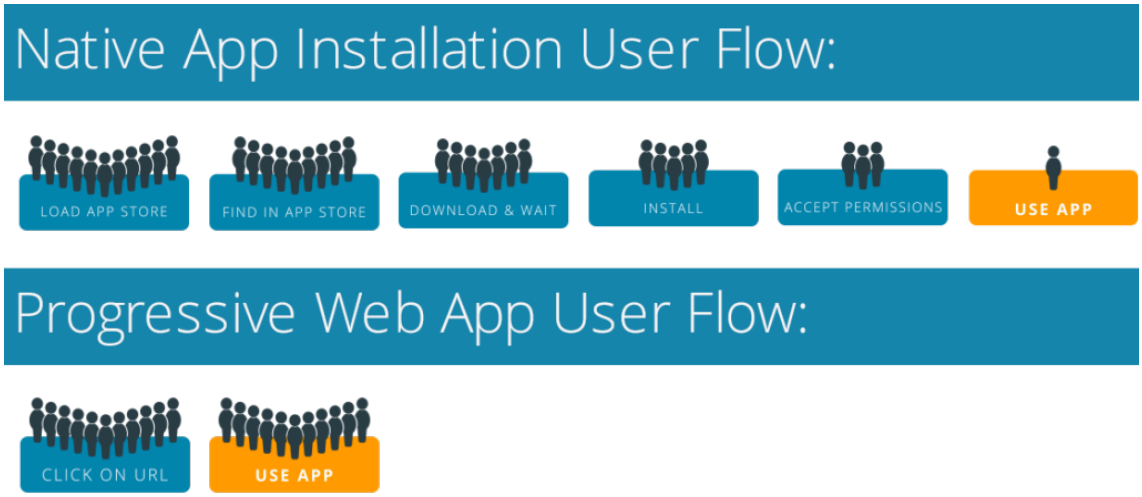


Abbildung 4.1.: Gegenüberstellung der Installationsschritte von Nativen Apps und Progressiven Web Apps (de Silva, 2017).

Außerdem hat der Nutzer die Möglichkeit, die PWA vor der Installation ausführlich zu testen. Somit kann dieser bereits vor dem Installationsprozess entscheiden, ob er diese Anwendung dauerhaft nutzen möchte oder nicht. Bei einer Nativen App ist dies erst nach der Installation möglich. Bei Nichtgefallen muss der Nutzer die Native App dann wieder deinstallieren. In Regionen mit schlechter Internetverbindung,

nicht verfügbarem Wifi und teuren Datentarifen kann dies für Nutzer sehr kostspielig werden. „Instead of forcing users to download an app [...], they can experience the product in their web browser as a regular website before installing it via the banner“ (Biørn-Hansen et. al., 2017). Ater (2017) merkt passend dazu an, dass viele Websites dem Nutzer die Installation der Nativen App geradezu aufdrängen. „The info is right there, within your grasp, but then a banner pops up on your screen blocking the very content you want. It asks you if you would like to install an app instead of reading the content that is already right in front of your eyes“ (Ater, 2017, S. 4).

4.1.6. Monetarisierung

Für die Entwicklung von Nativen Apps, Web Apps oder Progressiven Web Apps entstehen teilweise erhebliche Kosten. Daher ist die Monetarisierung dieser Entwicklung aus Unternehmenssicht ein wichtiges Thema. Dies gilt zumindest dann, wenn die Native App oder der Internetauftritt einen Service anbietet, welcher dem eigentlichen Geschäftsmodell entspricht und die App somit direkt zum Erzielen von Gewinnen entwickelt wird. Bei einigen Unternehmen stehen die Profitmöglichkeiten der entwickelten Anwendung nicht unbedingt im Vordergrund, da der Profit mit anderen Unternehmensteilen generiert wird. Die entwickelte Applikation dient dann z.B. nur als zusätzlicher Service, Hilfsmittel für Mitarbeiter oder „Marketinginstrument, mit dessen Hilfe sie ihr Image stärken und sich gegenüber den Wettbewerbern behaupten. Die Web-Auftritte großer Konzerne fallen beispielsweise in diese Kategorie“ (Anderl, März, Schumann, Wangenheim und Ackermann, 2012). Für diese Fälle bilden die Aspekte der Monetarisierung keine wichtige Entscheidungsgrundlage.
Tabelle 4.4 führt die Aspekte auf, die in diesem Abschnitt zum Vergleich der Entwicklungsansätze herangezogen werden.

So haben Native Apps im Gegensatz zu den browserbasierten Anwendungskonzepten ein vorhandenes System, „[...] that allows developers to sell and distribute their products to actors within one or more multisided software platform ecosystems“ (Jansen

und Bloemendal, 2013). Dieses vorgegebene Bezahlsystem stellt somit einen einfachen und bequemen Weg da, um die Anwendung zu vermarkten. Allerdings wird durch dieses System eine Provision für den App Store Betreiber fällig. Wie bereits in Kapitel 2.1.1 beschrieben, beträgt diese im Play Store von Google und im App Store von Apple bis zu 30% des Verkaufspreises. Somit verdient der App Entwickler bzw. das Unternehmen dementsprechend weniger Geld mit dem Verkauf der Anwendung. Bei Progressiven Web Apps und Web Apps entfallen diese Provisionskosten. „Herausgeber wie die Financial Times [...] setzen schon heute Web-Apps ein, [...] aus geschäftlichen Überlegungen heraus. Allen voran das Verlagswesen ist nicht mehr bereit, den App Stores gezwungenermaßen einen riesen Teil der Einnahmen [...] zu überlassen“ (Nielsen und Budiu, 2013). Jedoch müssen sich die Unternehmen und Entwickler dann selber um die Vermarktung ihrer PWA oder Web App und den damit angebotenen Service kümmern. Hierfür fallen in der Regel einige Kosten für Entwicklung und Pflege der Systeme an. Ob ein vorhandenes Profitsystem bei einer Nativen App auf Provisionsbasis im App Store oder die eigene Vermarktung einer PWA oder Web App die bessere Variante für ein Unternehmen ist, kann allgemein nicht beantwortet werden. Aus Aufwandssicht ist eine Native App jedoch leichter zu monetarisieren als eine PWA oder Web App.

Eigenschaft / Ansatz	Native App	PWA	Web App
Vorhandenes Profitsystem	ja	nein	nein
Provision für App Store Betreiber	ja	nein	nein
Werbeeinnahmen möglich	ja	ja	ja
Größte Kundenreichweite	nein	ja	ja
Einschränkung des Geschäftsmodells	ja	nein	nein
Inhalts- und Verkaufseinschränkung	ja	nein	nein

Tabelle 4.4.: Vergleich der Monetarisierung der Entwicklungsansätze.

Neben der Möglichkeit des Appverkaufs stellt das Schalten von Werbung bei allen der drei Varianten eine weitere Einnahmemöglichkeit dar. Diese Variante ist je nach Verkaufsstrategie für manche Unternehmen deutlich wichtiger als der Erlös durch den reinen Verkaufspreis. Dies liegt daran, dass die wenigsten Nutzer bereit sind, für mobile Inhalte Geld zu bezahlen. „Not surprisingly free apps are the most popular especially among younger users educated by Internet where 'everything is free'" (Kimbler, 2010). Um die Nutzer anzusprechen, muss der gebotene Service also meist kostenlos zur Verfügung gestellt werden und das Unternehmen muss sich auf die mobile Werbung als Einnahmequelle fokussieren. „[...] popular ad networks [...] allow developers to define which kind of ads are embedded, how they are delivered (push/pull techniques) and how often they are refreshed" (Vallina-Rodriguez, Shah, Finamore, Grunenberger, Papagiannaki, Haddadi und Crowcroft, 2012). Da die Möglichkeit der Werbeeinnahmen durch solche Werbenetzwerke bei allen drei hier betrachteten Entwicklungskonzepten zur Verfügung steht, kann in Bezug auf Werbeeinnahmen mit der verwendeten Literatur kein Vorteil für eines der drei Entwicklungskonzepte aufgezeigt werden.

Allerdings bieten die verschiedenen Entwicklungskonzepte eine unterschiedliche Kundenreichweite und somit eine unterschiedliche Zahl potentieller Kunden. So beschreibt Tandler (2017), dass die enorme Reichweite von Web Diensten einer der größten Vorteil der Webtechnologien gegenüber den Nativen Apps in den verschiedenen App Stores ist. „[...] Progressive Web Apps lassen sich so deutlich besser und einfacher promoten" (Tandler, 2017). Somit sind Web Apps und Progressive Web Apps bei dieser Thematik den Nativen Apps überlegen. Denn ein Browser kann auf jedem System installiert werden oder ist meist sowieso schon vorhanden. Die Apps in den verschiedenen App Stores hingegen erreichen immer nur die Kunden des jeweiligen Betriebssystems. Unter anderem aus Kosten- und Aufwandsgründen ist nicht jede Native App auf jedem Betriebssystem verfügbar und somit nicht in jedem App Store vorhanden. Jedoch weist Tiongson (2015) darauf hin, dass viele Nutzer den App Store durchscrollen, um neue Apps zu suchen. „In fact, 40% of smartphone users browse for apps in app stores" (Tiongson, 2015). Diese Kundenschicht entfällt

bei PWAs und Web Apps, falls die Nutzer nicht durch andere Wege auf die Anwendung aufmerksam werden. Des Weiteren ist es laut der App Agentur APP3null (2015) einfacher, im App Store durch App Store Optimierung ein gutes Ranking zu erzielen als mit Search Engine Optimization im Web. Dies liegt daran, dass „[...] die Suchalgorithmen der App Stores [...] noch nicht ausgereift [sind] und [...] sich relativ einfach beeinflussen [lassen]“ (APP3null, 2015). Wie bereits in Kapitel 4.1.1 beschrieben, werden PWAs jedoch in Zukunft z.B. auch im Microsoft Store gelistet. Dies würde bedeuten, dass PWAs auch in App Stores gefunden werden können und der Nutzer kann sich aussuchen, ob er die PWA über das Web oder in diesem Falle über den Microsoft Store sucht. In Summe ist die Kundenreichweite bei Web Apps und PWAs somit besser als bei Nativen Apps. Dies führt theoretisch zu mehr Einnahmen, da mehr Nutzer erreicht werden und somit die Zahl der Seitenaufrufe und Werbeklicks steigt.

Ein weiterer Faktor ist eine mögliche Einschränkung des eigenen Geschäftsmodells. Bahr et. al. (2012) beschreiben, dass bei geschlossenen Ökosystemen wie App Stores die Betreiber die Regeln festlegen. Somit gibt es keine Sicherheit, dass ein durch die App eines Unternehmens zur Verfügung gestellter Dienst auch in Zukunft noch den App Store Richtlinien entspricht und somit gestattet wird. „Innovative Inhaltsangebote können so auch nachträglich durch Plattformbetreiber eingeschränkt oder ausgesperrt werden, was Investitionen in ebensolche Angebote risikoreicher macht“ (Bahr et. al., 2012). Dieses Risiko besteht vor allem dann, wenn das Geschäftsmodell eines Unternehmens größtenteils auf dem Vertrieb von Diensten über Native Apps in App Stores basiert. „Apples App-Prüfungsprozess gilt als willkürlich und inkonsistent“ (Becker, 2018). Bei Web Angeboten gibt es zwar auch keinen Schutz des Geschäftsmodells, jedoch entfällt das Risiko von unvorteilhaften Entscheidungen für das Unternehmen durch die App Store Betreiber, solange der zur Verfügung gestellte Service nichts mit App Stores zu tun hat. Als Beispiel für solch eine Situation gilt z.B. das zuvor erwähnte Unternehmen Rollout.io. Der CEO Rusovsky (2017) beschreibt in einem Statement, dass der gebotene Service wegen Apples App Store Richtlinien nicht weiter angeboten werden kann und sich das Unternehmen daher

neuen Produkten widmen muss.

Neben den möglichen Einschränkungen beim Geschäftsmodell kann es auch zu anderen Inhalts- und Verkaufseinschränkungen kommen. Auch dies betrifft vor allem die Nativen Apps. Web Apps und PWAs sind hiervon nicht betroffen, da diese Einschränkungen auf den jeweiligen App Store Richtlinien der Betreiber beruhen. Durch diese wird unter anderem das Publizieren von bestimmten Inhalten in Nativen Apps verboten. Hierdurch kommt es laut Nielsen und Budiu (2013) zu einer Zensur des Internets. Außerdem beschreiben Bahr et. al. (2012), dass es vorkommen kann, dass z.B. Apps für den Apple App Store abgelehnt werden, die externe Links zu Mechanismen für Käufe oder Abonnements enthalten. Als Beispiel wird hier die 'c't Viewer' App vom Heise Verlag genannt. „Dem Verlag wurde stattdessen nahegelegt, Kauf- oder Abo-Funktionen in Form von „In-App Purchases" einzubauen, an denen Apple mitverdient – obwohl die Links gar nicht direkt auf Kaufoptionen oder Abo-Seiten verwiesen" (Bahr et. al., 2012). Die Betreiber von App Stores haben somit eine nicht zu unterschätzende Zensurmacht innerhalb der App Store Ökosysteme. „Zensurfreiheit und die Möglichkeit, die eigenen Einnahmen vollumfänglich selbst behalten zu können, sind gute Gründe, sich für das freie Internet und gegen die «umzäunten Gärten» der App Stores zu entscheiden" (Nielsen und Budiu, 2013). Soll also jederzeit die Möglichkeit bestehen, Anwendungen ohne eine mögliche Zensur des Inhalts oder der Verkaufswege zu vermarkten, so ist eine webbasierte Entwicklung die bessere Wahl.

4.2. Evaluation der Entwicklung der konzipierten Anwendungsfälle

Im Folgenden wird die Entwicklungsarbeit der für diese Arbeit konzipierten Anwendungsfälle evaluiert. Hierdurch findet ein Vergleich zwischen Web Apps und Progressiven Web Apps statt. Da viele Unternehmen bereits einen Webauftritt besitzen, liegt der Fokus der Aufwandsbewertung vor allem auf der Migration einer Web App zu einer PWA. Ein Vergleich mit der Entwicklung einer Nativen App entfällt in diesem Kapitel, da aus Aufwands- und Zeitgründen im Zuge dieser Arbeit keine Native App entwickelt wird. Als Erstes werden der Aufwand und die aufgetretenen Schwierigkeiten betrachtet. Anschließend folgt ein Einblick in nützliche Programme und Werkzeuge, die bei der Entwicklung einer PWA hilfreich sein können.

4.2.1. Aufwand und Schwierigkeiten

Eine erste Idee, ob sich der Aufwand lohnen kann, die eigene Internetpräsenz in eine Progessive Web App zu migrieren, kann mit der Tabelle 3.3 erschlossen werden. In diesen Anwendungsfällen hat die Migration drei Arbeitstage mehr benötigt als die eigentliche Implementierung der Web Apps. Im ersten Moment kann dies sehr irreführend wirken. Es gibt jedoch einige Gründe, weshalb dieses Ergebnis erzielt wurde. Zum Einen ist das Vorwissen sehr heterogen. Es gab viel mehr Hintergrundwissen bei der Erstellung einer normalen Web App als bei der Erstellung einer Progressiven Web App. Aus diesem Grund war die Einarbeitungszeit bei der Migration höher als bei der Web App. Zum Anderen haben die Portale eine überschaubare Anzahl an Funktionen. Hätten die Portale vergleichbar viele Funktionen wie Youtube.com o. ä., wäre der Aufwand der Implementation natürlich höchstwahrscheinlich höher als die Migration der Grundprinzipien einer Progressiven Web App.
Ein weiterer Grund für den hohen Aufwand der Migration sind die auftretenden Schwierigkeiten, die den Prozess verlangsamen. Google bietet zwar mit ihrer Plattform Codelaps[3] eine große Sammlung an Schritt für Schritt Anleitungen an, wie

[3]https://codelabs.developers.google.com/

beispielsweise die Migration einer Webseite zu einer Progressiven Web App[4], jedoch verwenden diese nur primitive Beispiele, die meistens nur eine einzige Seite behandeln. Anleitungen mit umfangreicheren Beispielen und deren Best Practices in einem Produktivsystem werden vergebens gesucht. Google selbst bietet Tools wie workbox[5] oder Service Worker Precache[6] an, die für größere Projekte gedacht sind und die Programmierung vereinfachen sollen, jedoch können sie den Entwicklungsprozess auch stark verlangsamen. Diese Tools befinden sich nach eigener Erfahrung in einem starken Entwicklungsstadium, der noch einige Fehler enthält. Diese Fehler kosten dem Entwickler sehr viel Zeit, da nicht in jedem Fall direkt klar wird ob es ein eigener Fehler oder ein Fehler der Bibliothek ist. Außerdem werden einem bei der Entwicklung von Service Workern viele Steine in den Weg gelegt. Der Cache im Browser ist meist sehr aggressiv, sodass Quelltextänderungen nicht beachtet werden, obwohl Einstellungen wie *update on reload* verwendet werden, die dies verhindern sollen. Ferner muss darauf geachtet werden, den Service Worker selbst nicht in den Cache zu speichern. Eine aufwendige Lösung ist die Löschung des Browsercaches nach jeder Änderung, welches in manchen Fällen mit einem Browserneustart kombiniert werden muss. Daneben sind je nach Browser die Fehlerbehebungen auf Basis der Fehlermeldungen ebenfalls schwierig. Trotz des gleichen Codes kann es vorkommen, dass zwei unterschiedliche Browser wie Chrome und Firefox andere Fehlermeldungen ausgeben, was die Suche nach dem Fehler erschwert.

Außerdem kann das Caching bei einer Web App viele Probleme bereiten. In diesem Fall werden die Inhalte der Web App beim Server verarbeitet und an den Browser weitergegeben, welcher sie direkt dem Nutzer anzeigt. Es werden hier keine Inhalte asynchron geladen. Normalerweise wird in dem Service Worker eine konstante Liste mit URLs eingetragen, die er im Hintergrund Schritt für Schritt abarbeitet und im Cache speichert. Liegt beispielsweise die URL eines Artikels im Cache, kann sofort die Seite vom Cache abgerufen werden ohne sie erneut aus dem Internet zu laden. Bei der Web App besitzen die Artikel/Videos dynamische URLs, somit kann keine konstante Liste an den Service Worker übergeben werden. Damit gestaltet sich das

[4]https://codelabs.developers.google.com/codelabs/migrate-to-progressive-web-apps
[5]https://developers.google.com/web/tools/workbox/
[6]https://github.com/GoogleChromeLabs/sw-precache

Vorausladen der Inhalte schwierig. Um die Seiten trotzdem im Cache zu speichern, wurde das *Laufzeitcaching* angewendet. Dabei wurde eine Regel erstellt, dass jede Seite die aufgerufen wird, im Cache gespeichert wird. Damit ist jede aufgerufene Seite im Offline-Modus trotzdem zugänglich. Eine Alternative wäre die Verwendung einer Single Page Application. Dabei wird die Seite initial ohne Inhalt geladen. Diese werden erst im nächsten Schritt asynchron nachgeladen und angezeigt. Dazu ruft der Browser eine API Schnittstelle auf und lädt darüber die Inhalte herunter. Die API-Endpunkte können einfach in die Liste des Service Workers eingetragen werden und das Precaching wäre bereits realisiert. In diesen Anwendungsfällen müsste jedoch zuerst eine API Schnittstelle bereitgestellt werden und aus Anwendersicht weitere Funktionen zur Verwendung der API und Darstellung der Webseite implementiert werden. Anschließend könnte eine Migration stattfinden. Dieser Gedanke wäre jedoch zeitlich kaum realisierbar gewesen.

Obwohl die Migration in diesem Fall einen höheren Aufwand ergab und es viele Stolpersteine geben kann, sollte die Entscheidung der Migration nicht sofort verworfen werden. Der Aufwand der Anwendungsfälle muss nicht für die Allgemeinheit gelten, da jeder Entwickler ein anderes Vorwissen besitzt. Außerdem geht die Entwicklung von Hilfsbibliotheken stetig voran. Progressive Web Apps enthalten viele wichtige Funktionen, die je nach Web App für ein Unternehmen einen großen Mehrwert geben können. Durch eine geschickte Priorisierung der zu entwickelnden Funktionen können jedoch schnell gute Ergebnisse erzielt werden, was ebenfalls einen Mehrwert für den Nutzer bieten kann. Ein Blick auf die erzielten Ergebnisse von Unternehmen in 4.4 zeigt, wie bedeutend dieser Mehrwert sein kann. Somit lässt sich sagen, dass sich eine Migration einer Web App für ein Unternehmen bei Betrachtung des Aufwandes durchaus lohnen könnte, wenn einige Faktoren wie Vorwissen und zu entwickelnde Funktionen abgewogen werden. Weiterhin können andere Entwicklungen, wie die Erstellung einer Nativen App, welche hier aus Zeitgründen ausgelassen wurde, einen höheren Aufwand mit sich bringen. Je nach benötigter Funktionalität würde dies nicht unbedingt auch einen Mehrwert gegenüber einer Progressiven Web App bedeuten.

4.2.2. Hilfreiche Programme und Werkzeuge

Die Entwicklungen rund um das Thema Progressive Web App schreiten weiter voran. Daher gibt es neben den in Kapitel 3 und 4.2.1 erwähnten und bei der Entwicklung genutzten Hilfen, auch weitere Programme und Werkzeuge, die die Entwickler unterstützen können. Hierdurch kann der zuvor beschriebenen Aufwand und das Auftreten von Schwierigkeiten eventuell reduziert werden. Google stellt für die Entwickler unter anderem die 'Progressive Web App Checklist' und das Tool 'Lighthouse' zur Verfügung. Damit kann überprüft werden, ob die eigene Entwicklung alle PWA Eigenschaften umsetzt. Des Weiteren hat Microsoft das Open-Source-Projekt 'PWABuilder' gegründet. Durch dieses Tool können Teile der PWA automatisiert erstellt werden.

Progressive Web App Checklist und Lighthouse

Die Progressive Web App Checkliste von Google führt grundlegende Aspekte auf, die bei einer Website bzw. Web App vorhanden sein müssen, damit diese als PWA betrachtet wird. „To help teams create the best possible experiences we've put together this checklist which breaks down all the things we think it takes to be a Baseline PWA [...]“ (Google, 2017f). Neben den Aspekten für eine grundlegende PWA, werden darüber hinaus auch noch weitere Aspekte in den Kategorien Indexability & social, User experience, Performance, Caching und Push notifications aufgeführt. Des Weiteren beschreibt Google (2017f), dass einige dieser Aspekte manuell überprüft werden müssen. Viele Punkte der einfachen Checklist können jedoch durch das Tool Lighthouse automatisch überprüft werden. „Lighthouse is an open-source, automated tool for improving the quality of web pages. [...]. It has audits for performance, accessibility, progressive web apps, and more“ (Google, 2017d). Dabei werden im Bereich der Progressiven Web Apps elf Punkte der PWA Checkliste validiert. Zu diesen Punkten gehört unter anderem die Prüfung auf erfolgreiche Registrierung eines Service Worker, die Verwendung von HTTPS und die Installationsfähigkeit der PWA. Die folgende Abbildung 4.2 zeigt das Ergebnis einer solchen Auswertung am Beispiel des entwickelten Newsportals. Genutzt wird in diesem Fall Lighthouse über

Chrome DevTools.

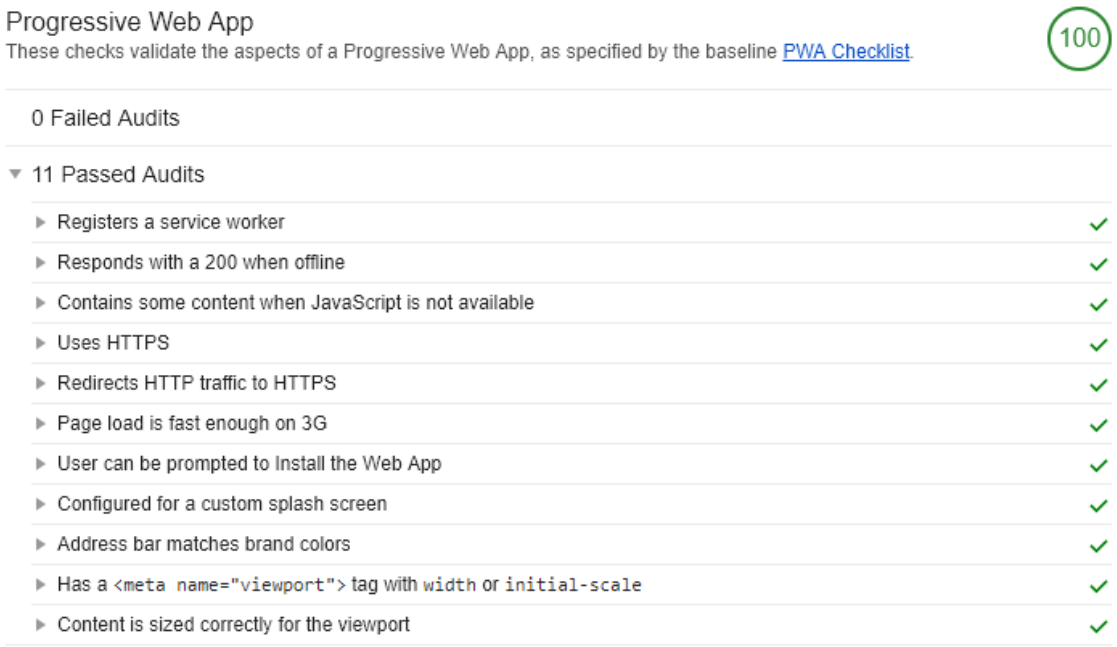


Abbildung 4.2.: Überprüfung des Newsportals durch Lighthouse.

Wie aus Abbildung 4.2 ersichtlich, sind alle elf automatisch durchgeführten Prüfungen durch Lighthouse erfolgreich verlaufen.

PWABuilder

PWABuilder ist ein Tool, mit dessen Hilfe das Web App Manifest und die Service Worker automatisch erstellt werden können. „PWABuilder was founded by Microsoft as a community guided, open source project to help move PWA adoption forward“ (Microsoft-Corporation, 2018). Das Tool kann webbasiert genutzt werden und steht zusätzlich auch als Download für die Kommandozeile zur Verfügung. Nach der Benutzung können der Quellcode bzw. die erstellten Dateien heruntergeladen werden, um sie im eigenen Code einzubauen. Hierdurch wird der Entwicklungsaufwand für eine PWA reduziert. Für die Erstellung des Web App Manifests durch den PWA-Builder wird nur die URL des Webauftritts benötigt. Hieraus werden dann die nötigen Einträge für die App Manifest Datei generiert, die dann im nächsten Schritt

weiter angepasst werden können. Bei der Erstellung der Service Worker wird beim PWABuilder derzeit zwischen vier verschiedene Stufen der Offline Funktionalität unterschieden. Die Auswahlmöglichkeiten sind in der folgenden Tabelle 2.2 aufgeführt.

Stufe	Beschreibung
1	Eine zuvor definierte Offline Datei wird geladen, falls keine Internetverbindung vorhanden ist.
2	Sobald eine Seite der PWA aufgerufen wird, wird eine Kopie im Cache abgelegt. Diese Kopie wird dann bei fehlender Internetverbindung geladen.
3	Stufe 2 mit Stufe 1 als Backup.
4	Bestimmte Seiten der PWA werden beim Start geladen und im Cache abgelegt. Bei welche Inhalten dies geschehen soll, wird zuvor festgelegt.

Tabelle 4.5.: Stufen der Service Worker Funktionalität im PWABuilder. Eigene Darstellung, Inhalt von: Microsoft-Corporation (2018).

Durch diese Auswahl wird deutlich, welche verschiedenen Möglichkeiten Service Worker bieten. Eine weitere Stufe, mit der noch weitere Möglichkeiten der Service Worker ausgeschöpft werden, ist in Arbeit. Wie gut dieser automatisch erstellte Service Worker Code in der Praxis funktioniert, wird hier nicht weiter untersucht.

4.3. Vergleich zwischen Web App und PWA auf Basis der Anwendungsfälle

Im Folgenden wird geprüft, welche Verbeserungen bzw. Änderungen durch den Umbau des Newsportals (NP) und des Videoportals (VP) zu einer PWA erzielt werden. Hierbei wird insbesondere die Offline Funktionalität, das native Verhalten und die Performance untersucht. Für alle hierfür durchgeführten Tests wird der Chrome Browser auf einem Huawei P9lite mit Android 7.0 genutzt. Bei den Tests zur Offline Funktionalität wird zur Simulation einer nicht vorhandenen Internetverbindung der Flugmodus des Smartphones verwendet.

4.3.1. Offline Funktionalität

Test 1: Erstmaliger Seitenaufruf

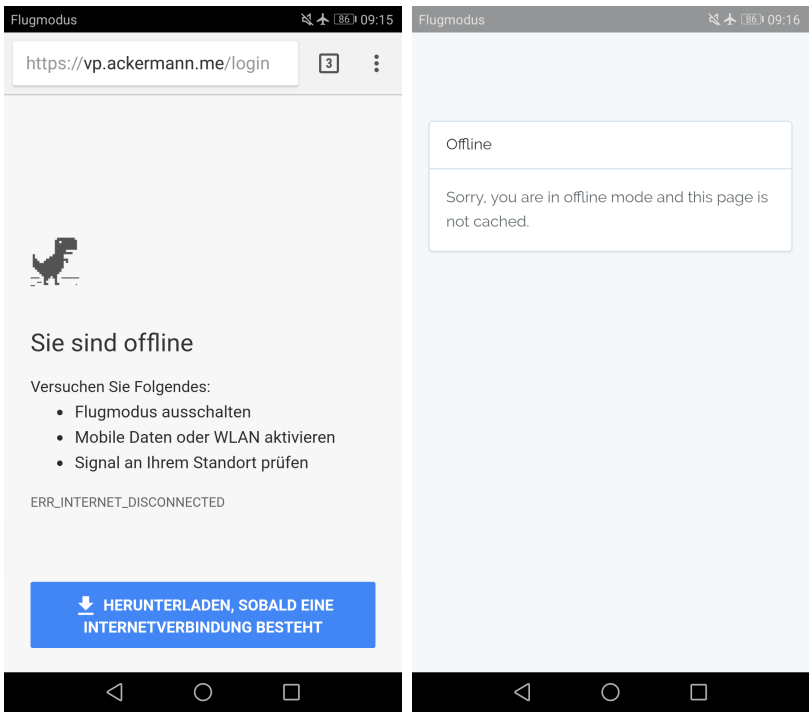
Bei diesem Test wird das Verhalten der Anwendung geprüft, wenn eine zuvor noch nicht aufgerufene Unterseite ohne aktive Internetverbindung aufgerufen wird.

Test 1	Web App (NP und VP)	PWA (NP und VP)
Ausgangslage	Browserdaten gelöscht	Browserdaten gelöscht
Vorgehen	1. Aufruf über URL 2. Flugmodus einschalten 3. Beliebige Unterseite aufrufen	1. Aufruf über URL oder Icon 2. Flugmodus einschalten 3. Beliebige Unterseite aufrufen

Tabelle 4.6.: Vorgehen beim Test zur Offline Funktionalität bei erstmaligem Seitenaufruf.

Der Test wurde jeweils mit dem Newsportal und Videoportal durchgeführt, welche beide das gleiche Ergebnis zeigen. Aufgerufen wurde die PWA über das Icon auf dem Homescreen. Abbildung 4.3 zeigt das Ergebnis des Videoportals. Der linke Teil (a) der Abbildung zeigt das Verhalten der Web App während der rechte Teil (b) das

Verhalten der dazugehörigen PWA aufweist.



(a) Web App (VP) (b) PWA (VP)

Abbildung 4.3.: Testergebnis der Offline Funktionalität bei erstmaligem Seitenaufruf.

Es wird deutlich, dass die PWA in der hier vorliegenden Situation ein besseres Offline Verhalten bieten kann als die Web App. Ist die derzeit geöffnete Seite zuvor noch nicht aufgerufen worden und somit noch nicht zwischengespeichert, so wird dem Nutzer anstelle der üblichen Browser Fehlermeldung eine vom Entwickler anpassbare Mitteilung angezeigt. Hierdurch kann der Nutzer der PWA eine aussagekräftigere Zustands- und Fehlerbeschreibung erhalten. Dem Nutzer könnte z.B. mitgeteilt werden, welche Handlungen auch Offline ausführbar sind, falls einer derartige Funktionalität implementiert ist. Bei der Web App hingegen ist dies nicht möglich, daher erscheint hier nur die übliche Fehlermeldung des Browers.

Test 2: Erneuter Seitenaufruf

Im Folgenden wird geprüft, ob ein bereits zuvor einmal aufgerufener Abschnitt der Web App und PWA auch bei erneutem Aufruf, ohne vorhandene Internetverbindung, angezeigt wird. Tabelle 4.7 zeigt das Vorgehen des Tests.

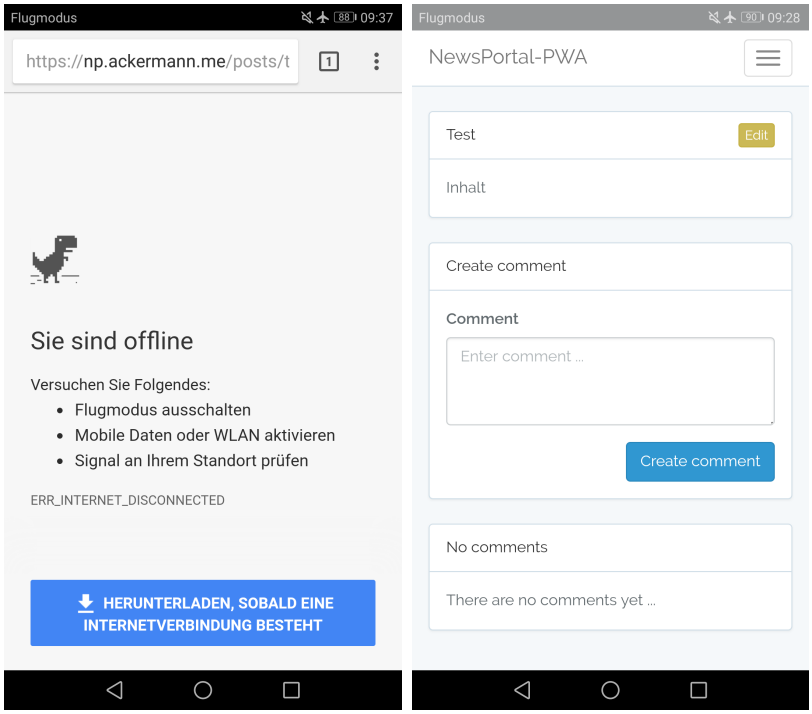
Test 2	Web App (NP und VP)	PWA (NP und VP)
Ausgangslage	Browserdaten gelöscht	Browserdaten gelöscht
Vorgehen	1. Aufruf über URL 2. Unterseite aufrufen 3. Zurück zur Hauptseite 4. Flugmodus einschalten 5. Unterseite erneut aufrufen	1. Aufruf über URL oder Icon 2. Unterseite aufrufen 3. Zurück zur Hauptseite 4. Flugmodus einschalten 5. Unterseite erneut aufrufen

Tabelle 4.7.: Vorgehen beim Test zur Offline Funktionalität bei erneutem Seitenaufruf.

Der Test wurde jeweils mit dem Newsportal und Videoportal durchgeführt. Das Ergebnis ist wie zu erwarten bei beiden das gleiche. Daher werden hier in Abbildung 4.4 nur Screenshots des Newsportals aufgeführt. Der Aufruf der PWA wurde über das Icon auf dem Homescreen getätigt.

Der linke Teil (a) der Abbildung zeigt das Verhalten der Web App. Der rechte Teil (b) zeigt das Verhalten der PWA. Aus den zwei dargestellten Screenshots wird deutlich, dass bei nicht vorhandener Internetverbindung die PWA den zuvor bereits einmal aufgerufenen Inhalt darstellen kann. Bei der Web App hingegen wird in diesem Fall nur die übliche Fehlermeldung des Browsers angezeigt. Somit liegt bei der PWA ein besseres Offline Verhalten vor, da es durch die Service Worker bessere und intelligentere Möglichkeiten gibt, Daten zwischenzuspeichern, um sie dann auch im Offline Zustand dem Nutzer zur Verfügung zu stellen. Eine einfache Zwischenspeicherung

der Web App kann jedoch auch durch den Browser erfolgen und ist nicht unmöglich.



(a) Web App (NP) (b) PWA (NP)

Abbildung 4.4.: Testergebnis der Offline Funktionalität bei erneutem Seitenaufruf.

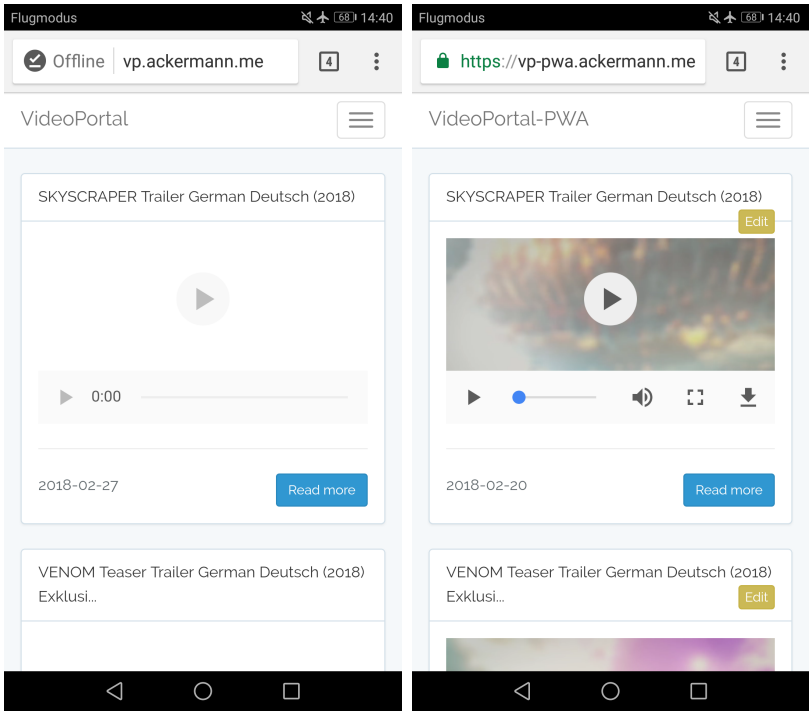
Test 3: Videos

Im Folgenden wird untersucht, ob sich die bessere Offline Funktionalität der PWA auch auf das Abspielen von Videos auswirkt. Hierzu wird geprüft, ob sich die Videos auf der Hauptseite der Web App und PWA auch im Offline Modus nach erneutem Aufruf der URL abspielen lassen.

Test 3	Web App (VP)	PWA (VP)
Ausgangslage	Browserdaten gelöscht	Browserdaten gelöscht
Vorgehen	1. Aufruf über URL 2. Flugmodus einschalten 3. URL erneut aufrufen	1. Aufruf über URL oder Icon 2. Flugmodus einschalten 3. URL erneut aufrufen

Tabelle 4.8.: Vorgehen beim Test zur Offline Funktionalität von vorhandenen Videos.

Durchgeführt wird der Test mit dem Videoportal. Aufgerufen wurde die PWA hier über die URL im Browser. Die Abbildung 4.5 verdeutlicht das Ergebnis.



(a) Web App (VP) (b) PWA (VP)

Abbildung 4.5.: Testergebnis der Offline Funktionalität von Videos.

Es ist ersichtlich, dass bei der Web App die Videos bei fehlender Internetverbindung nicht mehr abgespielt werden können, wenn die Seite erneut aufgerufen wird.

Die Darstellung der Web App erfolgt aus dem Cache des Browsers. Bei der PWA hingegen können die Videos auch dann abgespielt werden. Dies funktioniert bis zu dem Zeitpunkt im Video, an dem das Zwischenspeichern durch den Internetausfall unterbrochen wurde.

4.3.2. Natives Verhalten

Test 4: Anzeigebereich

Dieser Test untersucht den Anzeigebereich der Anwendung. Zu einem nativen Verhalten gehört unter anderem auch eine Vollbilddarstellung der Anwendung. Hierbei werden die Navigations- und Benachrichtigungsleiste ausgeblendet.

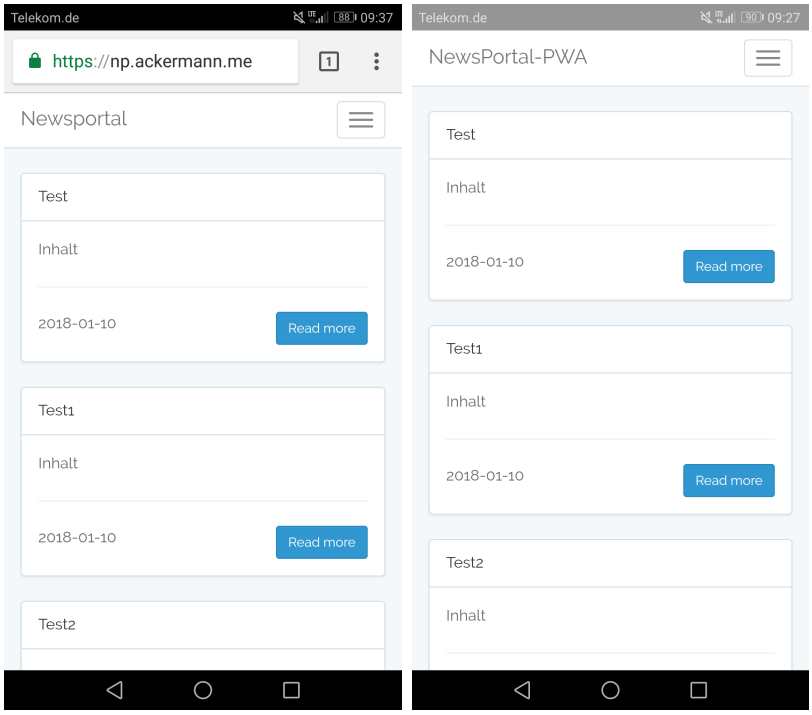
Test 4	Web App (NP und VP)	PWA (NP und VP)
Ausgangslage	Browserdaten gelöscht	Browserdaten gelöscht
Vorgehen	Aufruf über URL	Aufruf über Icon

Tabelle 4.9.: Vorgehen beim Test des nativen Verhalten des Anzeigebereichs.

Auch dieser Test wurde jeweils mit dem Newsportal und Videoportal durchgeführt. Die Durchführung führt auch hier bei beiden zum gleichen Ergebnis. Allerdings ist es in diesem Fall wichtig, dass die PWA über das Icon auf dem Homescreen gestartet wird, da ansonsten kein Vollbild Modus genutzt wird.

Wie in Abbildung 4.6 ersichtlich, ist der Anzeigebereich der PWA deutlich größer als der Anzeigebereich der Web App. Dies liegt daran, dass die PWA im Vollbildmodus ausgeführt wird. In dieser Situation bemerkt der Nutzer somit nicht, dass er sich eigentlich im Browser befindet. Bei einer Web App und beim Aufruf der PWA über die URL werden die UI Elemente des Browsers nicht ausgeblendet. Erst wenn der Nutzer auf der Website nach unten scrollt, blendet der Browser diese aus. Die Nutzung einer auf dem Homescreen abgelegten PWA verbessert somit das native

Verhalten von in Browsern laufenden Anwendungen.



(a) Web App (NP) (b) PWA (NP)

Abbildung 4.6.: Testergebnis des nativen Verhalten des Anzeigebereichs.

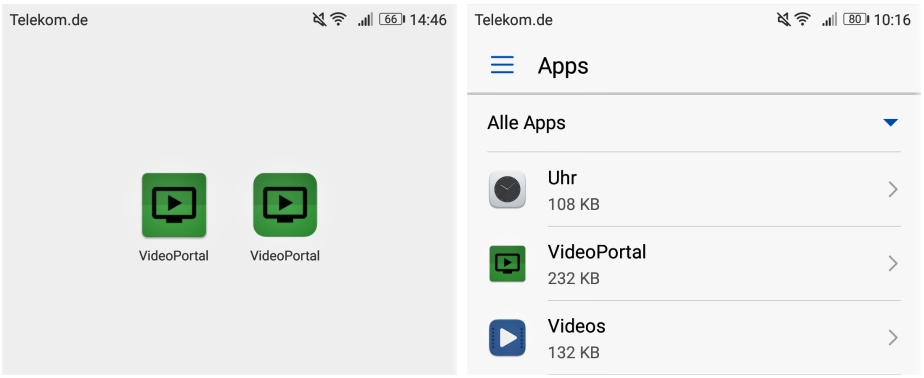
Test 5,6: Homescreenverknüpfung und Systemintegration

Im Folgenden wird überprüft, ob sich das Hinzufügen einer Web App Verknüpfung zum Homescreen von der Installation einer PWA auf dem Homescreen unterscheidet. Wie bereits am Vorgehen in Tabelle 4.10 zu erkennen, muss bei der Web App zuerst das Browsermenü geöffnet werden, um eine Verknüpfung der Web App zum Homescreen hinzuzufügen. Bei der PWA hingegen genügt ein Klick im unteren Bereich des Browserfensters, da dort beim erstmaligen Aufruf ein Hinweis erscheint, dass die PWA zum Homescreen hinzugefügt werden kann.

Test 5	Web App (NP und VP)	PWA (NP und VP)
Ausgangslage	keine vorherige Nutzung	keine vorherige Nutzung
Vorgehen	1. Aufruf über URL 2. Browsermenü öffnen 3. Zum Startbildschirm zufügen	1. Aufruf über URL 2. Zum Startbildschirm zufügen

Tabelle 4.10.: Vorgehen beim Test des Hinzufügens zum Homescreen.

Anhand von Abbildung 4.7 (a) wird deutlich, dass es zwischen der Web App und der PWA kaum einen sichtbaren visuellen Unterschied der Icons auf dem Homescreen gibt.



(a) Homescreen (b) App-Drawer

Abbildung 4.7.: Testergebnis des Hinzufügen zum Homescreen.

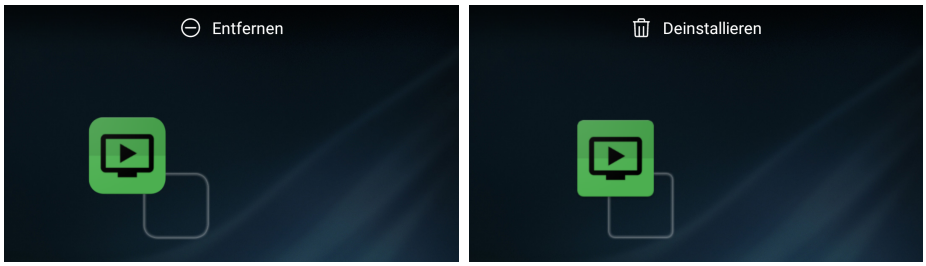
Beide Icons sind in diesem Fall nicht von einem Icon einer nativen Anwendung unterscheidbar. Eine PWA auf dem Homescreen wirkt vor der Nutzung also nicht nativer als eine auf dem Homescreen abgelegte Verknüpfung der Web App. Allerdings wird in (b) deutlich, dass sich die PWA im Gegensatz zu der Web App in den App-Drawer des Smartphones einfügt. Im App-Drawer sind die PWAs nicht von den dort aufgeführten Nativen Apps unterscheidbar. Der Nutzer wird hierdurch die PWA als vollwertige Anwendung wahrnehmen.

Weiter wird untersucht, ob sich das Entfernen der Web App Verknüpfung vom Homescreen von der Entfernung der PWA vom Homescreen unterscheidet und einen unterschiedlich starken nativen Eindruck vermittelt.

Test 6	Web App (NP und VP)	PWA (NP und VP)
Ausgangslage	Verknüpfung auf Homescreen	Auf Homescreen installiert
Vorgehen	1. Icon markieren und halten 2. Icon nach oben ziehen	1. Icon markieren und halten 2. Icon nach oben ziehen

Tabelle 4.11.: Vorgehen beim Test der Entfernung der Homescreenverknüpfung.

Sobald die Anwendung entfernt wird, wird deutlich, dass die PWA im Gegensatz zur Web App ähnlich wie eine Native App im System integriert ist. In Abbildung 4.8 ist jeweils im oberen Bereich der Screenshots der Unterschied sichtbar.



(a) Web App entfernen (b) PWA entfernen

Abbildung 4.8.: Testergebnis der Entfernung der Homescreenverknüpfung.

Während bei der Web App nur *Entfernen* steht, heißt es bei der Web App *Deinstallieren*, genauso wie bei nativen Apps. Diese Wortwahl kann sich jedoch je nach Version des Android Betriebssystems unterscheiden. Des Weiteren wird bei der PWA noch einmal nachgefragt, ob eine Deinstallation wirklich durchgeführt werden soll. Bei der Entfernung der Web App Verknüpfung ist dies nicht der Fall.

Test 7: Benachrichtigungen

Im Folgenden werden die Push Benachrichtigungen der PWA getestet. Dabei wird die Push Funktioalität jedoch nicht direkt mit der entwickelten Web App Version verglichen, da dort die Web Benachrichtigungen nicht implementiert sind. Nach der in dieser Arbeit erfolgten Abgrenzung von Web App und PWA wird die *Push API* den Progressiven Web Apps zugeordnet. Des Weiteren werden für eine sinnvolle Push Funktionalität die Service Worker benötigt, die bei Web Apps nicht vorhanden sind.

Tabelle 4.12 beschreibt die Durchführung des Tests der Push Benachrichtigungen.

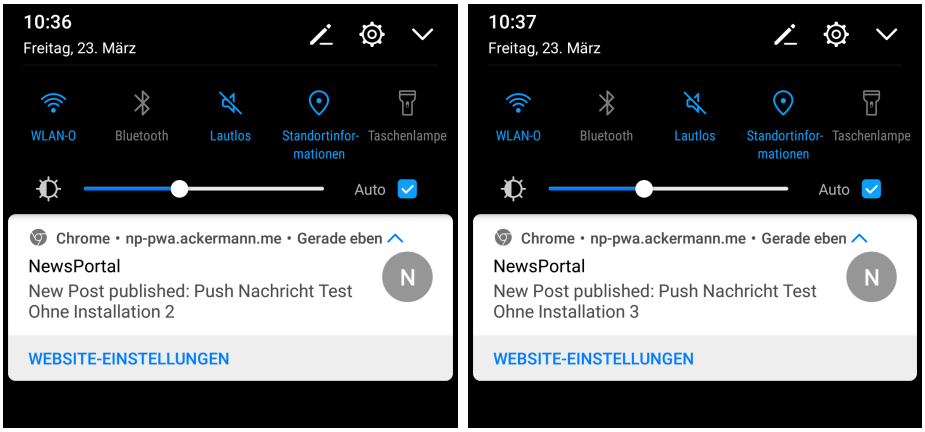
Test 7	PWA (NP und VP)	PWA (NP und VP)	PWA (NP und VP)
Ausgangslage	Push zugelassen Chrome geöffnet	Push zugelassen Chrome geschlossen	Push zugelassen Chrome geschlossen PWA installiert
Vorgehen	Eintrag erstellen mit anderem Gerät	Eintrag erstellen mit anderem Gerät	Eintrag erstellen mit anderem Gerät

Tabelle 4.12.: Vorgehen beim Test der Benachrichtigungen.

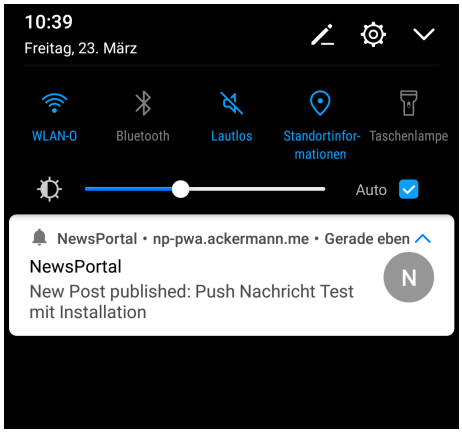
Der jeweilige News Eintrag zum Testen der Benachrichtigungen wird über die PWA Website von einem anderen Gerät aus erstellt. In Abbildung 4.9 ist das Ergebnis dieses Tests ersichtlich.

Die Push Benachrichtigungen funktionieren ohne PWA Installation unabhängig davon, ob Chrome im Hintergrund ausgeführt wird oder geöffnet oder geschlossen ist. Des Weiteren funktionieren die Push Meldungen, wenn Chrome geschlossen ist und die PWA auf dem Homescreen installiert ist. Hierbei wird als Herausgeber der Push Meldung nun nicht mehr Chrome genannt, sondern das NewsPortal. Jedoch fehlt dann aber die Möglichkeit, direkt zu den Website, bzw. Push Einstellungen zu gelangen. Part (c) zeigt, dass es zu keinen doppelten Push Benachrichtigungen kommt.

Es wird also deutlich, dass sich die Push Benachrichtigungen der PWAs nicht von denen einer nativen App unterscheiden.



(a) ohne Installation, Chrome geöffnet (b) ohne Installation, Chrome geschl.



(c) mit Installation, Chrome geschl.

Abbildung 4.9.: Testergebnis der Benachrichtigungen.

4.3.3. Performance

In diesem Abschnitt werden die Ladezeiten der erstellten Web Apps mit den Ladezeiten der erstellten PWAs verglichen. Hierdurch wird überprüft, ob eine Verbesserung durch die Migration des Newsportals und Videoportals erfolgt ist. Durchgeführt werden die Tests mit Hilfe von Webpagetest[7]. „WebPageTest is the leading web performance tool in the world today. It [...] provides the performance metrics that

[7]https://www.webpagetest.org/

matter, and is pioneering new ways to measure the actual user experience that websites deliver“ (Viscomi, Davies und Duran, 2015, S. viii). Für die Tests stehen laut WebPagetest-Dokumentation (2018) einige Einstellungsmöglichkeiten zur Verfügung. So kann z.B. festgelegt werden, von welchem Serverstandort und mit welchem Browser auf die Website zugegriffen werden soll. In den folgenden Tests wird Frankfurt als Serverstandort und Chrome als Browser gewählt. Des Weiteren können verschiedene Übertragungsgeschwindigkeiten simuliert und die Anzahl der Testdurchläufe festgelegt werden. Da PWAs vor allem bei schlechten Verbindungsgeschwindigkeiten Vorteile bringen sollen, wird in den folgenden Tests auch eine geringe Geschwindigkeit berücksichtigt. Unterschieden wird zwischen 2G (280 Kbit/s), 3G (1.6 Mbit/s) und LTE (12 Mbit/s). Außerdem wird die Anzahl der Testdurchläufe pro Website und Geschwindigkeit auf fünf festgelegt, um eine Verfälschung durch Ausreißer zu vermeiden. Auf Grundlage der fünf Messwerte wird der Mittelwert gebildet, der in den folgenden Tabellen aufgeführt ist. Die einzelnen gemessenen Werte sind im Anhang A zu finden.

Test 8: Erstmaliger Seitenaufruf

Im Folgenden werden die PWAs und Web Apps beim erstmaligen Seitenaufruf getestet. Dies bedeutet, dass Webpagetest die Seite mit gelöschten Browserdaten aufruft.

Ladezeiten Test 8	2G (280 Kbit/s)	3G (1.6 Mbit/s)	LTE (12 Mbit/s)
Newsportal Web App	18,892	5,641	1,853
Newsportal PWA	20,974	5,645	1,637
Videoportal Web App	52,057	7,010	2,447
Videoportal PWA	55,528	5,942	2,890

Tabelle 4.13.: Übersicht der Ladezeiten in Sekunden bei erstmaligen Seitenaufruf.

Anhand der Messwerte wird deutlich, dass in Bezug auf den erstmaligen Seiten-

aufruf keine eindeutige Verbesserung der Ladezeiten durch die Migration zu einer PWA erfolgt ist. Bei der 2G Geschwindigkeit ist die PWA beim Newsportal und Videoportal langsamer als die Web App. Beim 3G Netz ist die Videoportal PWA schneller geladen. Beim Newsportal ist die Ladezeiten identisch mit der Ladezeit der Web App. Bei LTE Geschwindigkeit ist die PWA beim Newsportal etwas schneller, beim Videoportal etwas langsamer.
Microsofts Pläne, die PWAs auch in ihren App Store zu listen, könnte allerdings auch dazu führen, dass die PWAs, genauso wie die Nativen Apps, über App Stores vertrieben werden und somit in Zukunft den Einschränkungen dieser unterliegen. Dies ist bei der Art und Weise, wie das Newsportal und das Videoportal hier implementiert sind, jedoch nicht so. Bei der PWA muss deutlich mehr JavaScript geladen und ausgeführt werden, was zu Zeitverzögerungen führt. Weiter ist beim erstmaligen Aufruf der Seite noch kein Caching durch die Service Worker erfolgt. Mögliche Vorteile durch besseres Caching der Service Worker im Vergleich zum Standard Browser Caching der Web Apps werden, falls vorhanden, somit erst im nächsten Test ersichtlich. Eine Verbesserung der Ergebnisse könnte eine Single Page Application bewirken. Der Vorteil liegt im asynchronen Verhalten der Applikation. Die Internetseite kann in einem Rohzustand ohne Inhalte geladen werden und gibt den Nutzer direkt ein Feedback. Anschließend kann der Inhalt nachgeladen werden. Ein weiterer Schritt wäre das im Hintergrund durchgeführte Vorausladen in den Cache weiterer Inhalte, die der Nutzer auf der aktuellen Seite aufrufen kann. Die Wahrnehmung des Nutzers wäre dadurch eine schnelle Single Page Application. Google verwendet im Codelab in der Regel Single Page Applications als Beispiele in ihren Progessive Web Apps Anleitungen. Eine These wäre dadurch ein größerer Mehrwert der PWA in der Ladezeit.

Test 9: Erneuter Seitenaufruf

Bei diesem Test werden die PWAs und Web Apps beim erneuten Seitenaufruf getestet. Dies bedeutet, dass die Testgeräte von Webpagetest die Seite zuvor schon aufgerufen haben und die Browserdaten nicht gelöscht sind. Bei diesen Tests ist

durch das bessere Caching der PWAs zu erwarten, dass diese eine bessere Performance aufweisen können als die Web Apps. Die Mittelwerte der Messergebnisse sind in Tabelle 4.14 aufgeführt.

Ladezeiten Test 9	2G (280 Kbit/s)	3G (1.6 Mbit/s)	LTE (12 Mbit/s)
Newsportal Web App	3,164	1,311	0,708
Newsportal PWA	3,969	1,411	0,684
Videoportal Web App	8,487	3,727	0,836
Videoportal PWA	8,477	4,145	0,967

Tabelle 4.14.: Übersicht der Ladezeiten in Sekunden bei erneutem Seitenaufruf.

Bei 2G Geschwindigkeit ist die Newsportal Web App schneller als die PWA und die Ladezeiten beim Videoportal unterscheiden sich nicht zwischen PWA und Web App. Beim 3G Netz zeigen die Messwerte auch keine Verbesserung der Ladezeiten der PWA. Gleiches gilt für die LTE Messungen, denn auch hier ist die Web App schneller als die PWA geladen. Laut zuvor vorgestellter Literatur soll die PWA jedoch gerade bei niedrigen Geschwindigkeiten Vorteile bringen.
Eine Ursache weshalb es nicht zu eine signifikanten Verbesserung der Zeit des Seitenaufbaus kam, könnte an der Verwendung von JavaScript liegen. In der Web App Version werden die Inhalte alle bereits vom Server bearbeitet und an den Browser übergeben. Es findet danach keine weitere Bearbeitung durch JavaScript statt. Im Gegensatz dazu findet zwar in der PWA auch die Bearbeitung der Inhalte auf der Serverseite statt, jedoch laufen noch einige Hintergrundprozesse, wie der Service Worker und der Push-Handshake, ab. Diese werden benötigt, bevor die Seite als geladen gilt. Des Weiteren beziehen sich Verbesserungen oft auf die Tatsache, dass Inhalte der PWA Offline durchgeschaut werden können und eine Internetverbindung bei bestimmten Service Worker Einsatz erst bei der Nutzung bestimmter Funktionen, wie z.B. ein Checkout beim Online Shopping, benötigt werden.

4.4. Verbesserungen durch PWA Entwicklung aus der Praxis

Im Folgenden werden einige Unternehmen aufgelistet, dessen Webauftritt zu einer Progressiven Web App weiterentwickelt wurde. Dabei werden, falls vorhanden, die erzielten Verbesserungen in der Conversion Rate, Bounce Rate und der Seitenladezeit aufgeführt.
„The conversion rate reflects the interaction between a website and its consumers' purchase choices and is defined as the percentage of users purchasing a product out of the total unique visitors entering a website“ (McDowell, Wilson und Kile Jr, 2016). Eine Steigerung dieser kann somit zu einem höheren Umsatz führen, da mehr Kunden eine Bestellung aufgeben. Die Bounce Rate gibt laut Kent, Carr, Husted und Pop (2011) den Teil der Nutzer an, die die Seite direkt wieder verlassen und somit nur einen Seitenaufruf erzeugen. Eine Verringerung dieser Rate deutet somit darauf hin, dass den Nutzern bzw. potentiellen Kunden der Webauftritt besser als zuvor gefällt. „[...] high bounce rates may indicate that users are dissatisfied with page content or layout, or that the page is not well aligned to their original query“ (Sculley, Malkin, Basu und Bayardo, 2009). Weiter beschreiben Sculley et. al. (2009), dass Nutzer, die den Webauftritt direkt wieder verlassen wahrscheinlich keinen Kauf durchführen. Somit kann sich eine Verringerung der Bounce Rate auch positiv auf den Unternehmenserfolg auswirken.

In Tabelle 4.15 sind elf Unternehmen und die durch die PWA Entwicklung erzielten Verbesserungen in den zuvor beschriebenen Messgrößen aufgeführt. Falls kein Wert vorliegt, so ist dieser Eintrag mit (-) gekennzeichnet. Durch die von den Unternehmen veröffentlichten Zahlen wird deutlich, dass sich der Umstieg auf eine PWA lohnen kann. So konnte die Conversion Rate bei einigen der aufgelisteten Webauftritte deutlich gesteigert werden. Beim Reiseunternehmen Wego hat sich diese um +95% verbessert. Und auch bei der Handelsplattform Alibaba, der Ticketverkaufsplattform BookMyShow und dem Video-on-Demand-Dienst Voot stieg die Conversion Rate um über 76%. Des Weiteren konnte bei vielen Unternehmen auch die Bounce Rate und die Seitenladezeit deutlich verringert werden. So konnten The Weather

Channel und Voot die Seitenladezeit gegenüber dem vorherigen Webauftritt um bis zu 80% verkürzen. Des Weiteren wurde auch die Bounce Rate bei den meisten um mindestens 20% verringert.

Unternehmen / Website	Conversion Rate	Bounce Rate	Seitenladezeit
Alibaba	+76%	-	-
BookMyShow	+80%	-	-
George	+31%	2x kleiner	3,8x schneller
Housing	+38%	-40%	-30%
MakeMyTrip	3x höher	-20%	-38%
Mynet	-	-24%	4x schneller
The Weather Channel	-	-	-80%
Twitter	-	-20%	-
Voot	+77%	-19%	-80%
Wego	+95%	-20%	-
5miles	+30%	-50%	-

Tabelle 4.15.: Erzielte Verbesserungen aus der Praxis. Eigene Darstellung, Daten von: Google (2018e), Google (2017g) und Google (2018f).

Neben den Verbesserungen dieser drei Kennzahlen kommt es laut Google (2018e) auch zu anderen Verbesserungen, wie teilweise bereits in Kapitel 2.2.4 angedeutet. Hierzu gehört zum Beispiel beim Kurznachrichtendienst Twitter eine Steigerung der Anzahl gesendeter Tweets um 75 % oder beim Nachrichtenportal Mynet ein längerer Seitenaufenthalt der Nutzer. BookMyShow konnte durch die Entwicklung einer PWA unter anderem den benötigten Speicherplatz deutlich reduzieren. „The size of the PWA is 54x smaller than the Android app and 180x smaller than the iOS app“ (Google, 2017a). Außerdem stieg laut Google (2016a) bei Alibaba die Zahl

der aktiven Nutzer auf Android nach der Einführung der PWA um 30% und auf iOS trotz fehlender Unterstützung vieler PWA Features um 15%. Des Weiteren beschreibt Google (2017e), dass sich beim Reiseunternehmen Makemytrip beim Nutzen der PWA 30% mehr Kunden spontan für eine Reise entscheiden als über die Native App. Auch das für den nigerianischen Markt ausgelegte E-Commerce Unternehmen Konga konnte Verbesserungen erzielen. „Compared with their native app, their Progressive Web App uses 92% less data for initial load and 82% less data to complete the first transaction“ (Google, 2017c). Im Vergleich zur vorherigen Website von Konga sind die Verbesserungen ebenfalls ähnlich.

Anhand dieser Praxis Erfahrungen von PWA Entwicklungen wird deutlich, dass sich eine PWA Migration in verschiedenen Bereichen auf unterschiedliche Art und Weise positiv bemerkbar machen kann. Entwickelt wurden die Progressiven Web Apps jedoch immer zusätzlich zum Webauftritt und Nativer App, da diese bereits schon vorhanden sind.

4.5. Ergebniszusammenfassung für Unternehmen und Nutzer

Im Folgenden wird zusammengefasst, welche Gründe für und gegen eine Progressive Web App als Alternative zur Web App und Ersatz zur Nativen App sprechen und inwieweit für Unternehmen und Nutzer ein Mehrwert entsteht.

4.5.1. Unternehmen

Für Unternehmen kann sich die Entwicklung einer PWA anstelle einer Web App oder Nativen App durchaus lohnen.
So sind PWAs im Vergleich zu einer Nativen App vor allem dann die bessere Wahl, wenn über die Anwendung Inhalte und Produkte präsentiert werden sollen. Hierzu gehören zum Beispiel Nachrichtenportale, Videoportale und E-Commerce Websites. Dies liegt daran, dass alle Inhalte über Suchmaschinen gefunden werden können. Somit können potentielle Kunden besser durch Suchmaschinen zum Webauftritt des Unternehmens geführt werden. Bei einer Nativen Anwendung ist dies nicht möglich, da der Inhalt dieser nicht über Suchmaschinen indiziert wird. Außerdem sind Unternehmen bei einer PWA oder Web App bezüglich der Inhalte und deren Vermarktung von den Richtlinien der App Store Betreiber unabhängig.
Des Weiteren ermöglichen PWAs auch immer mehr Features, die bisher nur von Nativen Apps bekannt sind. Besonders die Push Benachrichtigungen und die Homescreen Installation sind schnell realisierbar. So können Nutzer über Neuigkeiten informiert und die Kundenbindung gestärkt werden. Bei Web Apps war diese Möglichkeit bisher nur im geöffneten Browserfenster möglich und ein großer Nachteil webbasierter Anwendungen. Somit wird nicht mehr zwingend eine aufwendig für jedes Betriebssystem zu entwickelnde Native App benötigt. Trotz besserer Unterstützung bestimmter Gerätefunktionen kann es jedoch passieren, dass eine auf jeden Fall benötigte Gerätefunktion, wie z.B. der Zugriff auf Wearables, nicht unterstützt wird. In diesem Fall kommt eine PWA Entwicklung dann nicht als Alternative in Frage und es muss weiterhin auf die Entwicklung einer Nativen App zurückgegriffen werden.

Für kleine Unternehmen, wie z.B. Startups, kann die plattformunabhängige Entwicklung einer PWA eine kostengünstige Möglichkeit sein, um einen leichten Einstieg in den Markt zu finden. Der Mehraufwand für eine PWA gegenüber einer Web App ist im Vergleich zu den gewonnenen Nativen Features akzeptabel. Die Service Worker benötigen zwar einen nicht zu unterschätzenden Aufwand an Entwicklungsarbeit, jedoch können diese nach und nach verbessert werden und zum Markteinstieg reicht wahrscheinlich eine grundlegende Implementierung. Außerdem gibt es hilfreiche Tools, die bei der Erstellung einer PWA helfen. Eine Webpräsenz muss oftmals sowieso erstellt werden, um als Unternehmen Erfolg zu haben. Im Gegensatz zu den Nativen Apps in den App Stores bieten PWAs genauso wie Web Apps zwar kein vorhandenes Bezahlmodell, jedoch ist dies für Start-Ups oft nicht wichtig. Denn häufig kann bei Markteintritt für eine App sowieso kein Geld verlangt werden, da die Nutzer kostenlose Apps bevorzugen und die Nutzerzahlen erst einmal durch eine kostenlose Einführungsphase gesteigert werden müssen. Zu einem späteren Zeitpunkt kann dann ein eigenes Bezahlsystem implementiert werden oder die Einnahmen werden auf Werbeeinnahmen beschränkt.
Für große, bereits auf dem Markt etablierte Unternehmen kann die Entscheidungsgrundlage anders ausfallen. Zwar ist eine plattformübergreifende Entwicklung aus Kosten- und Aufwandsicht gut, jedoch ist dies nicht unbedingt im Fokus, da solche Unternehmen oft schon auf allen Plattformen mit einer eigenen App vertreten sind. Oftmals ist es für große Konzerne einfach nur wichtig, überall präsent zu sein, um die Marke ins Gedächtnis des Kunden zu rufen. Somit wird eine PWA hier wahrscheinlich keine Native App ersetzen, die meist bereits schon vorhanden ist. Jedoch kann es durchaus sinnvoll sein, den Webauftritt zu einer PWA zu migrieren. Für das vorhandene Entwicklungsteam sollte dies machbar sein und somit wird der Webauftritt um nützliche Features erweitert.

Des Weiteren versprechen die in Kapitel 4.4 vorgestellten Verbesserungen der Conversion Rate, Bounce Rate und Seitenladezeit, dass sich die Entwicklung einer PWA nicht nur aufgrund neuer Funktionen und der Möglichkeit der plattformübergreifenden Entwicklung lohnen kann, sondern auch wegen einer besseren Performance und

Nutzererfahrung. Allerdings können die Verbesserungen der Seitenladezeit durch die in dieser Arbeit erstellten und entwickelten Anwendungsfälle nicht bestätigt werden. Die Gründe hierfür wurden bereits zuvor beschrieben. Um die Performance zu steigern, scheint also auch bei einer PWA einiges an Entwicklungsaufwand nötig und gegebenfalls muss der Webauftritt komplett neu aufgebaut werden. Denn für eine Performance Steigerung müsste wahrscheinlich eine Single Page Application entwickelt werden. Da allerdings vor allem der Aufwand der Migration einer Web App zu einer PWA abgebildet werden sollte, wurde dies hier nicht durchgeführt.

Ein weiterer Aspekt ist der geringere Speicherverbrauch von PWAs. Dies ist für Unternehmen von Bedeutung, da Nutzer so wahrscheinlich eher bereit sind, eine Anwendung auf dem Endgerät zu installieren. Hierdurch ist das Unternehmen auch ohne Native App jederzeit auf dem Homescreen im Blickfeld des Kunden. Der geringere Speicher- und Datenverbrauch ist von Vorteil, wenn Märkte in Regionen mit schlechter Netzqualität, teurer Internetverbindung oder weniger leistungsstarken Endgeräten bedient werden sollen. Eine PWA kann hier ein guter Ersatz zur Nativen App sein, die erst aus dem App Store geladen werden muss. In solchen Regionen sind auch die Smartphones oft mit wenig Speicher ausgestattet. Ein geringer Speicherverbrauch der Anwendung ist dann ein großer Vorteil, um Kunden auch ohne Native App eine gute User Experience zu bieten.
Auch das gute Offline Verhalten kann dafür sprechen, auf eine PWA zurückzugreifen. Im Laufe dieser Arbeit wird zwar deutlich, dass eine gute Service Worker Funktionalität mit einigem Aufwand verbunden ist. Jedoch sind bei etablierten Unternehmen ab einer bestimmten Größe oft genug Ressourcen vorhanden, um sich länger mit einem neuen Entwicklungsansatz zu beschäftigen und alle Optimierungen auszuschöpfen. Im Falle der PWA kann mit genug Aufwand eine sehr gute Nutzung der Service Worker, und somit ein gutes Offline Verhalten, erreicht werden. Die verbesserte Offline Nutzung kann so das Nutzererlebnis deutlich steigern. Die Nativen Apps haben zwar auch oft ein gutes Offline Verhalten, da sich der Code nach der Installation auf dem Smartphone befindet. Jedoch benötigen viele native Anwendungen während der Nutzung trotzdem genauso wie die Web Anwendungen eine

Internetverbindung.
Da derzeit die meisten PWA Features von Apple noch nicht unterstützt werden, beschränkt sich der Mehrgewinn der PWAs jedoch vorerst noch größtenteils auf die Android Plattform. Dies ist ein Problem, da je nach gewünschter Funktionalität eine zusätzliche Native App für iOS Systeme entwickelt werden muss. Dies bedeutet wieder eine deutliche Steigerung des Entwicklungsaufwands. Außerdem hat nach Zahlen von Gartner (2018) iOS weltweit einen Marktanteil an den 2017 verkauften Smartphones von 14%. Dies scheint im Vergleich zu Android mit fast 86% wenig, jedoch gelten Käufer von iOS Geräten als deutlich zahlungskräftiger. „Of those purchases made on mobile devices, 24.5% were placed on devices running Google's Android operating system and 75.2% were made on devices running Apple's iOS“ (Buras, 2016).
Für zukünftig geplante Apps des Unternehmens kann eine PWA als Alternative zur Nativen App sinnvoll sein. Nutzen z.B. alle für eine interne App relevanten Mitarbeiter des Unternehmens ein Android Smartphone, so stellt die derzeitige mangelnde Unterstützung von Apple kein Problem dar. Für unternehmensinterne Zwecke kann so schnell für bestimmte Aufgabengebiete eine schlanke Anwendung entwickelt werden.

4.5.2. Nutzer

Auch für die Nutzer der Anwendungen können Gründe für den Umstieg auf eine PWA sprechen. So kann der Nutzer im Gegensatz zur Nativen App eine PWA ausgiebig vor der ersten Nutzung testen. Gefällt die Anwendung nicht, so entfällt die Installation und die anschließende Deinstallation. Des Weiteren hat der Nutzer durch den geringen Speicherverbrauch einer PWA mehr Speicherplatz für Videos oder andere Medien zur Verfügung. Dies ist besonders für Smartphone Besitzer mit sehr geringem Speicher interessant und bei selten genutzten Apps ist eine hohe Speichernutzung oft ein Ärgernis für den Nutzer. Auch ein manuelles Update muss nicht mehr durchgeführt werden, da sich Web Apps und PWAs ohne Zutun des Nutzers aktualisieren. Des Weiteren können Inhalte einfacher mit anderen geteilt werden.

Außerdem entfällt das Problem, wenn eine bestimmte Anwendung nur für das Betriebssystem zur Verfügung steht, das der Nutzer derzeit nicht nutzt.
Allerdings kann es zu Beginn ungewohnt sein, die App nicht mehr aus einem App Store zu beziehen. Ein App Store schafft eventuell Vertrauen, da dort verfügbare Apps geprüft werden. Des Weiteren unterscheidet sich eine schlichte App Suche im Store gegenüber einer umfangreichen Suchmaschinensuche. Auch für den Nutzer spürbare Performance Verbesserungen sind von Fall zu Fall unterschiedlich. In den in dieser Arbeit durchgeführten kleinen Use Cases ist wie bereits zuvor beschrieben keine Verbesserung bemerkbar. Die in Kapitel 4.4 aufgeführte Beispiele zeigen jedoch, dass durchaus eine Steigerung der Performance möglich ist. Besonders die bessere Offline Funktionalität kann die Nutzererfahrung steigern. Auch neue Features, wie die Push Benachrichtigungen, die dem Nutzer sonst nur von Nativen Apps bekannt waren, sind auch für die Nutzer interessant. Gegenüber der Web App sind für den Nutzer keine Nachteile feststellbar. Jemand, der die Web App ohne neue Funktionen nutzen möchte, lehnt eine Installation und die Berechtigung zu Push einfach ab.

5. Fazit und Ausblick

5.1. Fazit

In dieser Arbeit wurden die Progressiven Web Apps, anhand der zur Verfügung stehenen Literatur und den entwickelten Use Cases, untersucht. Dabei wurden die Eigenschaften und Möglichkeiten dieses neuen Entwicklungsansatzes aufgezeigt. Anschließend wurden die PWAs dann mit den Web Apps und Nativen Apps verglichen. Hierbei wird deutlich, dass sich die Migration einer Web App zu einer PWA trotz des Entwicklungsaufwands lohnen kann, da die Vorteile einer PWA, wie z.b. die Push Benachrichtigungen und die Installationsmöglichkeit, nicht unerheblich sind. Die Web Apps werden um zusätzliche Funktionen erweitert, die vorher nur bei Nativen Apps möglich waren.
Des Weiteren gibt es durchaus auch Anwendungsfälle, in denen eine Progressive Web App eine Native App ersetzen kann. Ob dies der Fall ist, hängt jedoch von dem gewünschten Funktionsumfang der Anwendung und anderen Gegebenheiten der Entwickler und Unternehmen ab. In manchen Situation können die PWAs nicht den ganzen benötigten Funktionsumfang bedienen. Außerdem sorgt die derzeit noch mangelnde Unterstützung seitens Apple dafür, dass für iOS Geräte trotzdem noch eine Native App entwickelt werden muss. Somit haben auch die Nativen Apps weiterhin eine Daseinsberechtigung.
Eine klare Empfehlung für einen nativen- oder webbasierten Entwicklungsansatz kann also auf Basis dieser Arbeit nicht aufgezeigt werden, da es von den jeweiligen Anforderungen abhängig ist.

5.2. Ausblick

Die weiteren Entwicklungen im Bereich der Progressiven Web Apps werden zeigen, ob sich diese Art von Applikation als plattformübergreifendes Entwicklungskonzept bei Unternehmen, Entwicklern und Nutzern durchsetzen kann. Vor allem wird erst in einiger Zeit deutlich werden, ob die PWAs die Nativen Apps und damit auch die App Stores ersetzen können. Die Grundlagen sind vielversprechend und da auch Apple andeutet, die PWAs in Zukunft zu unterstützen, spricht einiges dafür. Im Zuge dessen wird das Smartphone in Zukunft weiter in Richtung offenes Web ausgerichtet werden, da durch die PWAs eine Alternative zu den, über App Stores vertriebenen, Nativen Apps vorhanden ist. Hierfür muss allerdings noch Entwicklungsaufwand betrieben werden. Interessant klingt auch die Möglichkeit, die PWAs auf Desktop Systemen nutzbar zu machen, wie Microsoft und Google angekündigt haben. Dies würde die Progressiven Web Apps zu einem Entwicklungskonzept erweitern, das nicht nur verschiedene mobile Systeme, sondern gleichzeitig auch Desktop Systeme abdeckt. Microsofts Pläne, die PWAs auch in ihren App Store zu listen, könnte allerdings auch dazu führen, dass die PWAs, genauso wie die Nativen Apps, über App Stores vertrieben werden und somit in Zukunft den Einschränkungen dieser unterliegen. Eine weitere Beobachtung der Entwicklungen in diesem Bereich kann sich demnach lohnen und ermöglicht weitere Forschungsfragen im Bereich der Progressiven Web Apps und der Cross-Platform Entwicklung. Insbesondere in Bezug auf das Offline Verhalten der PWAs können Untersuchungen angestrebt werden, um herauszufinden, wie die Service Worker am Besten eingesetzt werden können, um eine gute Offline Erfahrung des Nutzers zu ermöglichen.

Literaturverzeichnis

[1&1-DigitalGuide 2017a] 1&1-DigitalGuide: *Progressive Web-Apps: Was versprechen die progressiven Apps?* 2017. – URL https://hosting.1und1.de/digitalguide/websites/web-entwicklung/progressive-web-apps-welche-vorteile-bieten-sie/. – [Abruf: 10.08.2017]

[1&1-DigitalGuide 2017b] 1&1-DigitalGuide: *Was ist eine Web-App? Definition und Web-App-Beispiele.* 2017. – URL https://hosting.1und1.de/digitalguide/websites/web-entwicklung/verschiedene-app-formate-was-ist-eine-web-app/. – [Abruf: 07.09.2017]

[Ahn 2018] Ahn, Andrew: *How we fought bad apps and malicious developers in 2017.* 2018. – URL https://android-developers.googleblog.com/2018/01/how-we-fought-bad-apps-and-malicious.html. – [Abruf: 31.01.2017]

[Airberlin 2016] Airberlin: *Progressive Web App von airberlin - Präsentation auf der Google I/O.* 2016. – URL https://flights.airberlin.com/de-DE/progressive-web-app#informationen. – [Abruf: 18.07.2017]

[Ait Bahajji und Illyes 2014] Ait Bahajji, Zineb ; Illyes, Gary: *HTTPS as a ranking signal.* 2014. – URL https://webmasters.googleblog.com/2014/08/https-as-ranking-signal.html. – [Abruf: 17.10.2017]

[Anderl et. al. 2012] Anderl, E ; März, A ; Schumann, JH ; Wangenheim, F v. ; Ackermann, S: Kann man mit kostenfreien Dienstleistungen Geld verdienen? (2012)

[Android-Developers 2017a] Android-Developers: *Create Deep Links to App Content.* 2017. – URL https://developer.android.com/training/app-links/deep-linking.html. – [Abruf: 13.03.2018]

[Android-Developers 2017b] Android-Developers: *Verknüpfungen mit Google Play*. 2017. – URL https://developer.android.com/distribute/marketing-tools/linking-to-google-play.html. – [Abruf: 13.03.2018]

[Angulo und Ferre 2014] Angulo, Esteban ; Ferre, Xavier: A case study on cross-platform development frameworks for mobile applications and ux. In: *Proceedings of the XV International Conference on Human Computer Interaction* ACM (Veranst.), 2014, S. 27

[APP3null 2015] APP3null: *Apps Monitarisieren - Mit Apps Geld verdienen*. 2015. – URL https://app3null.com/apps-monetarisieren/. – [Abruf: 20.12.2017]

[Apple 2017] Apple: *Code Signing*. 2017. – URL https://developer.apple.com/support/code-signing/. – [Abruf: 08.11.2017]

[Apple o.J.] Apple: *Swift. Eine leistungsstarke offene Sprache, mit der jeder großartige Apps erstellen kann.* o.J.. – URL https://www.apple.com/de/swift/. – [Abruf: 08.03.2017]

[Archibald 2015] Archibald, Jake: *Introducing Background Sync*. 2015. – URL https://developers.google.com/web/updates/2015/12/background-sync. – [Abruf: 20.01.2017]

[Ater 2017] Ater, Tal: *Building progressive web apps: bringing the power of native to the browser*. Ö'Reilly Media, Inc.", 2017

[Bahr et. al. 2012] Bahr, Fabian ; Dapp, Thomas F. ; Dobusch, Leonhard ; Grzegorzek, Marcin ; Kerst, Valentina ; Meinberg, Ralf ; Rehse, Mario ; Sänger, Jessica ; Schauf, Thomas ; Tillmann, Henning: Schönes neues Internet? Chancen und Risiken für Innovation in digitalen Ökosystemen. In: *Policy Brief. Stiftung Neue Verantwortung. Berlin* (2012)

[Becker 2017] Becker, Leo: *Achtung, Datenverlust: iOS 11 öffnet alte Apps nicht mehr*. 2017. – URL https://www.heise.de/mac-and-i/meldung/Achtung-Datenverlust-iOS-11-oeffnet-alte-Apps-nicht-mehr-3823602.html. – [Abruf: 10.11.2017]

[Becker 2018] Becker, Leo: *Apple stört sich an Emoji-Verwendung in Apps*. 2018. – URL https://www.heise.de/mac-and-i/meldung/Apple-stoert-sich-an-Emoji-Verwendung-in-Apps-3960787.html. – [Abruf: 05.02.2018]

[Biørn-Hansen et. al. 2017] Biørn-Hansen, Andreas ; Majchrzak, Tim A. ; Grønli, Tor-Morten: Progressive web apps: The possible web-native unifier for mobile development. In: *Proceedings of the 13th International Conference on Web Information Systems and Technologies*, 2017, S. 344–351

[Braun 2011] Braun, Herbert: *FAQ: HTML5*. 2011. – URL https://www.heise.de/ct/hotline/FAQ-HTML5-1284742.html. – [Abruf: 21.12.2017]

[Buras 2016] Buras, Maddie: *Custora E-Commerce Pulse Report: Q1 2016*. 2016. – URL http://www.custora.com/2016/05/custora-e-commerce-pulse-report-q1-2016/. – [Abruf: 28.02.2017]

[Burtoft 2017] Burtoft, Jeff: *Microsoft's Bold Plan to Bring PWAs to Windows 10*. 2017. – URL https://www.thurrott.com/windows/windows-10/142649/microsofts-bold-plan-bring-pwas-windows-10. – [Abruf: 11.12.2017]

[Caceres et. al. 2017] Caceres, Marcos ; Christiansen, Kenneth R. ; Lamouri, Mounir ; Kostiainen, Anssi ; Dolin, Rob: *Web App Manifest*. 2017. – URL https://www.w3.org/TR/appmanifest/#dfn-manifest. – [Abruf: 26.10.2017]

[Charland und Leroux 2011] Charland, Andre ; Leroux, Brian: Mobile application development: web vs. native. In: *Communications of the ACM* 54 (2011), Nr. 5, S. 49–53

[Claburn 2017] Claburn, Thomas: *Apple signals it's willing to let next-gen web apps compete with iOS apps*. 2017. – URL http://www.theregister.co.uk/2017/08/04/service_workers_land_in_webkit_heralding_apple_acceptance/. – [Abruf: 10.08.2017]

[Cloudflare Inc 2018] Cloudflare Inc: *Secure Socket Layer (SSL)/TLS | Cloudflare*. 2018. – URL https://www.cloudflare.com/de/ssl/. – [Abruf: 10.03.2018]

[Cunningham 2017] Cunningham, Edward: *Improving app security and performance on Google Play for years to come*. 2017. – URL https://android-developers.googleblog.com/2017/12/improving-app-security-and-performance.html. – [Abruf: 20.12.2017]

[Delia et. al. 2015] Delia, Lisandro ; Galdamez, Nicolas ; Thomas, Pablo ; Corbalan, Leonardo ; Pesado, Patricia: Multi-platform mobile application development analysis. In: *Research Challenges in Information Science (RCIS), 2015 IEEE 9th International Conference on* IEEE (Veranst.), 2015, S. 181–186

[d'Heureuse 2017] d'Heureuse, Severin: *Progressive Web Apps - PWA*. 2017. – URL http://www.opten.ch/blog/2017/01/18/progressive-web-apps-pwa/. – [Abruf: 18.07.2017]

[Elitech Systems 2017] Elitech Systems: *The most popular PHP-frameworks in 2017*. 2017. – URL https://medium.com/@elitechsystems/the-most-popular-php-frameworks-in-2017-a90a1189405e. – [Abruf: 06.12.2017]

[Farrugia 2016] Farrugia, Kevin: *A Beginner's Guide To Progressive Web Apps*. 2016. – URL https://www.smashingmagazine.com/2016/08/a-beginners-guide-to-progressive-web-apps/. – [Abruf: 19.08.2017]

[Feiner et. al. 2015] Feiner, Johannes ; Krajnc, Elmar ; Niederl, Franz: 7 Der Weg zur eigenen App-Herausforderungen im Mobile App Development. In: *KMU GOES MOBILE* (2015), S. 67

[Firtman 2016] Firtman, Maximiliano: *High Performance Mobile Web: Best Practices for Optimizing Mobile Web Apps*. O'Reilly Media, Inc., 2016. – URL https://books.google.de/books?hl=de&lr=&id=OOYPDQAAQBAJ&oi=fnd&pg=PR2&dq=High+Performance+Mobile+Web:+Best+Practices+for+Optimizing+Mobile+Web+Apps&ots=rtude1EAZD&sig=UMdYWIhSqnVpumNUurtE1_uHEXQ&redir_esc=y#v=onepage&q=High%20Performance%20Mobile%20Web%3A%20Best%20Practices%20for%20Optimizing%20Mobile%20Web%20Apps&f=false

[Fondsbörse-Deutschland o.J.] Fondsbörse-Deutschland: *Biometrie statt Passwort*. o.J.. – URL https://www.fondsboerse-deutschland.de/biometrie-statt-passwort. – [Abruf: 07.02.2018]

[Gallagher 2017] Gallagher, Nicolas: *How we built Twitter Lite*. 2017. – URL https://blog.twitter.com/engineering/en_us/topics/open-source/2017/how-we-built-twitter-lite.html. – [Abruf: 08.12.2017]

[Gao et. al. 2014] Gao, Ming ; Hu, Xihong ; Cao, Bo ; Li, Dianxin: Fingerprint sensors in mobile devices. In: *Industrial Electronics and Applications (ICIEA), 2014 IEEE 9th Conference on* IEEE (Veranst.), 2014, S. 1437–1440

[Gartner 2017] Gartner: *Gartner Says Worldwide Sales of Smartphones Grew 9 Percent in First Quarter of 2017*. 2017. – URL https://www.gartner.com/newsroom/id/3725117. – [Abruf: 07.11.2017]

[Gartner 2018] Gartner: *Gartner Says Worldwide Sales of Smartphones Recorded First Ever Decline During the Fourth Quarter of 2017*. 2018. – URL https://www.gartner.com/newsroom/id/3859963. – [Abruf: 28.02.2017]

[Gaunt 2018] Gaunt, Matt: *Service Workers: an Introduction*. 2018. – URL https://developers.google.com/web/fundamentals/primers/service-workers/. – [Abruf: 20.03.2018]

[Gaunt und Kinlan 2018] Gaunt, Matt ; Kinlan, Paul: *The Web App Manifest*. 2018. – URL https://developers.google.com/web/fundamentals/engage-and-retain/web-app-manifest/. – [Abruf: 20.03.2018]

[Google 2016a] Google: *Alibaba.com increases conversions on the mobile web by 76Progressive Web App*. 2016. – URL https://developers.google.com/web/showcase/2016/alibaba. – [Abruf: 25.07.2017]

[Google 2016b] Google: *Progressive Web Apps - A new way to deliver amazing user experiences on the web*. 2016. – URL https://developers.google.com/web/progressive-web-apps/?hl=en. – [Abruf: 18.07.2017]

[Google 2016c] Google: *The Weather Channel sees mobile web success with Progressive Web App, launching in 62 languages to 178 countries*. 2016. – URL https://developers.google.com/web/showcase/2016/weather-channel. – [Abruf: 25.07.2017]

[Google 2017a] Google: *BookMyShow's new Progressive Web App drives an 80in conversions*. 2017. – URL https://developers.google.com/web/showcase/2017/bookmyshow. – [Abruf: 20.03.2018]

[Google 2017b] Google: *Changes to the Chrome App Deprecation Schedule*. 2017. – URL https://plus.google.com/+NobleAckerson/posts/i8uLr9rpGwR. – [Abruf: 31.01.2017]

[Google 2017c] Google: *Konga*. 2017. – URL https://developers.google.com/web/showcase/2016/konga. – [Abruf: 28.02.2018]

[Google 2017d] Google: *Lighthouse*. 2017. – URL https://developers.google.com/web/tools/lighthouse/. – [Abruf: 09.02.2018]

[Google 2017e] Google: *MakeMyTrip.com's new PWA delivers 3X improvement in conversion rates*. 2017. – URL https://developers.google.com/web/showcase/2017/make-my-trip. – [Abruf: 26.07.2017]

[Google 2017f] Google: *Progressive Web App Checklist*. 2017. – URL https://developers.google.com/web/progressive-web-apps/checklist. – [Abruf: 20.03.2018]

[Google 2017g] Google: *Show Cases (2016)*. 2017. – URL https://developers.google.com/web/showcase/2016/. – [Abruf: 20.03.2018]

[Google 2018a] Google: *Google Codelabs*. 2018. – URL https://codelabs.developers.google.com/. – [Abruf: 10.03.2018]

[Google 2018b] Google: *Google Trends*. 2018. – URL https://trends.google.de/trends/. – [Abruf: 27.03.2018]

[Google 2018c] Google: *Introduction to Push Notifications | Web | Google Developers*. 2018. – URL https://developers.google.com/web/ilt/pwa/introduction-to-push-notifications. – [Abruf: 10.03.2018]

[Google 2018d] Google: *Overview | Tools for Web Developers | Google Developers*. 2018. – URL https://developers.google.com/web/tools/workbox/overview. – [Abruf: 10.03.2018]

[Google 2018e] Google: *Show Cases (2017)*. 2018. – URL https://developers.google.com/web/showcase/2017/. – [Abruf: 20.03.2018]

[Google 2018f] Google: *Show Cases (2018)*. 2018. – URL https://developers.google.com/web/showcase/2018/. – [Abruf:20.03.2018]

[Google-Trends 2018] Google-Trends: *Progressive Web Apps Suchhäufigkeit*. 2018. – URL https://trends.google.de/trends/explore?date=2015-08-01%202018-01-29&q=progressive%20web%20apps. – [Abruf: 29.01.2017]

[Griffith 2017] Griffith, C.: *Mobile App Development with Ionic 2: Cross-Platform Apps with Ionic 2, Angular 2, and Cordova*. O'Reilly Media, Incorporated, 2017. – ISBN 9781491937785

[Grigsby 2016] Grigsby, Jason: *iOS doesn't support Progressive Web Apps, so what?* 2016. – URL https://cloudfour.com/thinks/ios-doesnt-support-progressive-web-apps-so-what/. – [Abruf: 23.01.2017]

[Gustafson 2017a] Gustafson, Aaron: *Progressive web apps and the Windows ecosystem*. 2017. – URL https://channel9.msdn.com/Events/Build/2017/B8075. – [Abruf: 26.10.2017]

[Gustafson 2017b] Gustafson, Aaron: *Yes, That Web Project Should Be a PWA*. 2017. – URL https://alistapart.com/article/yes-that-web-project-should-be-a-pwa. – [Abruf: 13.12.2017]

[Heinrich 2010] Heinrich, Jana: *Suchmaschinen-Marketing als Teil des Online-Marketings*. diplom. de, 2010. – URL https://books.google.es/books?id=

rNFmDQAAQBAJ&printsec=frontcover&hl=de&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false

[Heitkötter et. al. 2012] Heitkötter, Henning ; Hanschke, Sebastian ; Majchrzak, Tim A.: Evaluating cross-platform development approaches for mobile applications. In: *International Conference on Web Information Systems and Technologies* Springer (Veranst.), 2012, S. 120–138

[Hird 2011] Hird, Jake: *The fight gets technical: mobile apps vs. mobile sites.* 2011. – URL https://econsultancy.com/blog/7832-the-fight-gets-technical-mobile-apps-vs-mobile-sites. – [Abruf: 12.09.2017]

[Huy et. al. 2012] Huy, Ngu P. et. al.: Selecting the right mobile app paradigms. In: *Service-Oriented Computing and Applications (SOCA), 2012 5th IEEE International Conference on* IEEE (Veranst.), 2012, S. 1–6

[IBM 2012] IBM: *Native, web or hybrid mobile-app development.* Whitepaper. 2012. – URL ftp://public.dhe.ibm.com/software/pdf/mobile-enterprise/WSW14182USEN.pdf. – [Abruf: 07.11.2017]

[Jansen und Bloemendal 2013] Jansen, Slinger ; Bloemendal, Ewoud: Defining app stores: The role of curated marketplaces in software ecosystems. In: *International Conference of Software Business* Springer (Veranst.), 2013, S. 195–206

[Jobe 2013] Jobe, William: Native apps vs. mobile web apps. In: *International Journal of Interactive Mobile Technologies (iJIM)* 7 (2013), Nr. 4, S. 27–32

[Joorabchi et. al. 2013] Joorabchi, Mona E. ; Mesbah, Ali ; Kruchten, Philippe: Real challenges in mobile app development. In: *Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium on* IEEE (Veranst.), 2013, S. 15–24

[Kent et. al. 2011] Kent, Michael L. ; Carr, Bryan J. ; Husted, Rebekah A. ; Pop, Rebeca A.: Learning web analytics: A tool for strategic communication. In: *Public Relations Review* 37 (2011), Nr. 5, S. 536–543

[Kimbler 2010] Kimbler, Kristofer: App store strategies for service providers. In: *Intelligence in Next Generation Networks (ICIN), 2010 14th International Conference on* IEEE (Veranst.), 2010, S. 1–5

[Kintan 2017] Kintan: *Does Your Business Need Progressive Web App?* 2017. – URL https://skilled.co/resources/business-need-progressive-web-app-infographic/. – [Abruf: 21.12.2017]

[Koroll 2016] Koroll, Julius: *5 Gründe, warum Sie auf eine Web App setzen sollten.* 2016. – URL https://desiign.de/2016/06/web-app-vorteile/. – [Abruf: 10.01.2017]

[Krebs 2016] Krebs, Simon: *3 GRÜNDE, WARUM PROGRESSIVE WEB APPS DIE BESSEREN WEB-APPS SIND!* 2016. – URL http://www.biz-factory.de/blog/2016/08/04/3-gruende-warum-progressive-web-apps-die-besseren-apps-sind/. – [Abruf: 12.12.2017]

[Liebel 2017a] Liebel, Christian: *Progressive Web Apps, Teil 2: Die Macht des Service Worker.* 2017. – URL https://www.heise.de/developer/artikel/Progressive-Web-Apps-Teil-2-Die-Macht-des-Service-Worker-3740464.html. – [Abruf: 22.08.2017]

[Liebel 2017b] Liebel, Christian: *Progressive Web Apps, Teil 3: Wie die Web-App zur App-App wird.* 2017. – URL https://www.heise.de/developer/artikel/Progressive-Web-Apps-Teil-3-Wie-die-Web-App-zur-App-App-wird-3464603.html. – [Abruf: 22.08.2017]

[Liebel 2017c] Liebel, Christian: *Progressive Web Apps, Teil 5: Das App-Modell der Zukunft?* 2017. – URL https://www.heise.de/developer/artikel/Progressive-Web-Apps-Teil-5-Das-App-Modell-der-Zukunft-3767383.html. – [Abruf: 10.08.2017]

[Lindemann 2016] Lindemann, Rolf: Usable Authentication. In: *Datenschutz und Datensicherheit-DuD* 40 (2016), Nr. 4, S. 229–230

[Lockheimer 2012] Lockheimer, Hiroshi: *Android and Security*. 2012. – URL http://googlemobile.blogspot.de/2012/02/android-and-security.html. – [Abruf: 08.11.2017]

[Lynch 2016] Lynch, Max: *What are Progressive Web Apps?* Mai 2016. – URL http://blog.ionic.io/what-is-a-progressive-web-app/. – [Abruf: 18.07.2017]

[Lyndersay 2017] Lyndersay, Sean: *Microsoft Edge for iOS and Android: What developers need to know*. 2017. – URL https://blogs.windows.com/msedgedev/2017/10/05/microsoft-edge-ios-android-developer/. – [Abruf: 07.02.2018]

[Majchrzak et. al. 2018] Majchrzak, Tim A. ; Biørn-Hansen, Andreas ; Grønli, Tor-Morten: Progressive Web Apps: the Definite Approach to Cross-Platform Development? In: *Proceedings of the 51st Hawaii International Conference on System Sciences*, 2018, S. 5735–5744

[Malavolta 2016] Malavolta, Ivano: Beyond Native Apps: Web Technologies to the Rescue! (Keynote). In: *Proceedings of the 1st International Workshop on Mobile Development*. New York, NY, USA : ACM, 2016 (Mobile! 2016), S. 1–2. – URL http://doi.acm.org/10.1145/3001854.3001863. – ISBN 978-1-4503-4643-6

[Malavolta et. al. 2017] Malavolta, Ivano ; Procaccianti, Giuseppe ; Noorland, Paul ; Vukmirović, Petar: Assessing the Impact of Service Workers on the Energy Efficiency of Progressive Web Apps. In: *Proceedings of the 4th International Conference on Mobile Software Engineering and Systems*. Piscataway, NJ, USA : IEEE Press, 2017 (MOBILESoft '17), S. 35–45. – URL https://doi.org/10.1109/MOBILESoft.2017.7. – ISBN 978-1-5386-2669-6

[McDowell et. al. 2016] McDowell, William C. ; Wilson, Rachel C. ; Kile Jr, Charles O.: An examination of retail website design and conversion rate. In: *Journal of Business Research* 69 (2016), Nr. 11, S. 4837–4842

[MDNwebdocs 2017] MDNwebdocs: *Web App Manifest*. 2017. – URL https://developer.mozilla.org/en-US/docs/Web/Manifest. – [Abruf: 29.08.2017]

[Medley 2018] Medley, Joseph: *Web Push Notifications: Timely, Relevant, and Precise*. 2018. – URL https://developers.google.com/web/fundamentals/engage-and-retain/push-notifications/. – [Abruf: 20.03.2018]

[Mehler und Skiena 2006] Mehler, Andrew ; Skiena, Steven: Improving usability through password-corrective hashing. In: *International Symposium on String Processing and Information Retrieval* Springer (Veranst.), 2006, S. 193–204

[Microsoft-Corporation 2018] Microsoft-Corporation: *Generate your Progressive Web App*. 2018. – URL https://www.pwabuilder.com/serviceworker. – [Abruf: 09.02.2018]

[Miller 2017] Miller, Chance: *Apple cracking down on developers who use SDKs like Rollout to update apps without App Store approval*. 2017. – URL https://9to5mac.com/2017/03/07/apple-cracks-down-on-hot-push-code-sdks/. – [Abruf: 04.01.2017]

[Mills 2016] Mills, Chris: *Progressive web app advantages*. 2016. – URL https://developer.mozilla.org/en-US/Apps/Progressive/Advantages. – [Abruf: 31.01..2018]

[Mills und Hobson 2016] Mills, Chris ; Hobson, Stephanie: *Progressive web apps*. 2016. – URL https://developer.mozilla.org/en/Apps/Progressive. – [Abruf: 18.07.2017]

[Mozilla-WADI-initiative 2016] Mozilla-WADI-initiative: *Progressive Web Apps HQ. The Web is the platform*. Mozilla Developer Websit. 2016. – URL https://mozilla.github.io/progressive-apps-hq/features/re-engageable/. – [Abruf: 31.01.2018]

[NETGEAR 2016] NETGEAR: *Arlo Wire-Free HD Security Camera System User Manual*. 2016. – URL https://www.arlo.com/en-us/images/Documents/Arlo_Wirefree/arlo_wirefree_um.pdf. – [Abruf: 14.12.2017]

[Nielsen und Budiu 2013] Nielsen, Jakob ; Budiu, Raluca: *Mobile usability*. MITP-Verlags GmbH & Co. KG, 2013

[Oliveira et. al. 2016] Oliveira, Wellington ; Torres, Weslley ; Castor, Fernando ; Ximenes, Bianca H.: Native or web? A preliminary study on the energy consumption of android development models. In: *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on* Bd. 1 IEEE (Veranst.), 2016, S. 589–593

[Osmani 2018] Osmani, Addy: *The App Shell Model*. 2018. – URL https://developers.google.com/web/fundamentals/architecture/app-shell. – [Abruf: 20.03.2018]

[Otto 2017] Otto, Hans-Christian: *Oft sind Progressive Web-Apps der effizientere Weg!* 2017. – URL https://react-days.de/blog/oft-sind-progressive-web-apps-der-effizientere-weg/. – [Abruf: 16.10.2017]

[Otwell 2018a] Otwell, Taylor: *Database: Getting Started*. 2018. – URL https://laravel.com/docs/5.6/database. – [Abruf: 08.03.2017]

[Otwell 2018b] Otwell, Taylor: *Database: Migrations*. 2018. – URL https://laravel.com/docs/5.6/migrations. – [Abruf: 08.03.2017]

[PWAStats 2018] PWAStats: *A community-driven list of stats and news related to Progressive Web Apps*. 2018. – URL https://www.pwastats.com/. – [Abruf: 03.03.2018]

[Rickard 2016] Rickard, Chris: *Choosing the right mobile app for your project: Native vs cross-platform vs hybrid*. 2016. – URL http://inoutput.io/articles/development/choosing-the-right-mobile-app-for-your-project-native-vs-cross-platform-vs-hybrid. – [Abruf: 12.09.2017]

[Rusovsky 2017] Rusovsky, Erez: *Rollout's Statement on Apple Guidelines*. 2017. – URL https://rollout.io/blog/rollout-statement-on-apple-guidelines/. – [Abruf: 04.01.2017]

[Schechter 2017] Schechter, Emily: *Next steps toward more connection security*. 2017. – URL https://blog.chromium.org/2017/04/next-steps-toward-more-connection.html. – [Abruf: 07.11.2017]

[Schirrmacher 2018] Schirrmacher, Dennis: *Trojaner-Apps mit 4,5 Millionen Downloads in Google Play entdeckt.* 2018. – URL https://www.heise.de/security/meldung/Trojaner-Apps-mit-4-5-Millionen-Downloads-in-Google-Play-entdeckt-3952145.html. – [Abruf: 31.01.2017]

[Schonschek 2013] Schonschek, Oliver: *Wie sicher sind die App-Stores?* 2013. – URL https://www.computerwoche.de/a/wie-sicher-sind-die-app-stores,2519593, 2. – [Abruf: 16.10.2017]

[Sculley et. al. 2009] Sculley, D ; Malkin, Robert G. ; Basu, Sugato ; Bayardo, Roberto J.: Predicting bounce rates in sponsored search advertisements. In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining* ACM (Veranst.), 2009, S. 1325–1334

[Selvarajah et. al. 2013] Selvarajah, Kirusnapillai ; Craven, Michael P. ; Massey, Adam ; Crowe, John ; Vedhara, Kavita ; Raine-Fenning, Nicholas: Native Apps versus Web Apps: which is best for healthcare applications? In: *International Conference on Human-Computer Interaction* Springer (Veranst.), 2013, S. 189–196

[Semenov 2017] Semenov, Alan: *How Progressive Web Apps make the Web great again.* 2017. – URL http://webagility.com/posts/how-progressive-web-apps-make-the-web-great-again. – [Abruf: 20.08.2017]

[Shafirov 2017] Shafirov, Maxim: *Kotlin on Android. Now official.* 2017. – URL https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official/. – [Abruf: 26.01.2017]

[Shiny-Development 2018] Shiny-Development: *iOS App Store - Raw Data.* 2018. – URL http://appreviewtimes.com/ios/raw-data. – [Abruf: 04.01.2017]

[Siebert 2017] Siebert, Jürgen: *Was ist eigentlich Corporate Design?* 2017. – URL https://page-online.de/branche-karriere/was-ist-eigentlich-corporate-design/. – [Abruf: 07.02.2018]

[de Silva 2017] Silva, Kimberly de: *Progressive Web Apps: The Next Mobile Experience?* 2017. – URL https://www.biznessapps.com/blog/progressive-web-apps/. – [Abruf: 14.12.2017]

[Stack Overflow 2017] Stack Overflow: *Developer Survey Results 2016*. 2017. – URL https://insights.stackoverflow.com/survey/2016#developer-profile. – [Abruf: 06.09.2017]

[Steiner 2018] Steiner, Thomas: What is in a Web View? An Analysis of Progressive Web App Features When the Means of Web Access is not a Web Browser. (2018)

[Steyer 2016] Steyer, Manfred: *Web-Apps mit Push Notifications durch Service Worker*. 2016. – URL https://entwickler.de/leseproben/web-apps-mit-push-notifications-255533.html. – [Abruf: 30.08.2017]

[Symfony™ 2018] Symfony™: *Why should I use a framework?* 2018. – URL https://symfony.com/why-use-a-framework. – [Abruf: 06.03.2018]

[Tandler 2017] Tandler, Marcus: *Progressive Web Apps sind cooler als native mobile Apps*. 2017. – URL https://www.deutsche-startups.de/2017/03/21/progressive-web-apps-sind-cooler-als-native-mobile-apps/

[TIOBE Software B.V. 2017] TIOBE Software B.V.: *TIOBE Index*. September 2017. – URL https://www.tiobe.com/tiobe-index. – [Abruf: 06.09.2017]

[Tiongson 2015] Tiongson, James: *Mobile App Marketing Insights: How Consumers Really Find and Use Your Apps*. 2015. – URL https://www.thinkwithgoogle.com/consumer-insights/mobile-app-marketing-insights/. – [Abruf: 10.01.2017]

[Traughber 2017] Traughber, Patrick: *Introducing Twitter Lite*. 2017. – URL https://blog.twitter.com/en_us/topics/product/2017/introducing-twitter-lite.html. – [Abruf: 08.12.2017]

[Vallina-Rodriguez et. al. 2012] Vallina-Rodriguez, Narseo ; Shah, Jay ; Finamore, Alessandro ; Grunenberger, Yan ; Papagiannaki, Konstantina ;

Haddadi, Hamed ; Crowcroft, Jon: Breaking for commercials: characterizing mobile advertising. In: *Proceedings of the 2012 ACM conference on Internet measurement conference* ACM (Veranst.), 2012, S. 343–356

[Viscomi et. al. 2015] Viscomi, Rick ; Davies, Andy ; Duran, Marcel: *Using WebPageTest: Web Performance Testing for Novices and Power Users*. Ö'Reilly Media, Inc.", 2015

[W3C 2017] W3C: *Page Visibility Level 2*. 2017. – URL https://www.w3.org/TR/page-visibility-2/. – [Abruf: 17.01.2017]

[W3C 2018] W3C: *Service Workers Nightly*. 2018. – URL https://w3c.github.io/ServiceWorker/. – [Abruf: 30.01.2017]

[W3Counter 2017] W3Counter: *Browser & Platform Market Share*. 2017. – URL https://www.w3counter.com/globalstats.php. – [Abruf: 15.01.2017]

[WebKit 2018] WebKit: *WebKit A fast, open source web browser engine.* 2018. – URL https://webkit.org/status/#?search=service. – [Abruf: 03.02.2018]

[WebPagetest-Dokumentation 2018] WebPagetest-Dokumentation: *WebPagetest Documentation*. 2018. – URL https://sites.google.com/a/webpagetest.org/docs/using-webpagetest/quick-start-quide. – [Abruf: 28.02.2017]

[whatwebcando.today 2016] whatwebcando.today: *What Web Can Do Today: Geofencing*. 2016. – URL https://whatwebcando.today/geofencing.html. – [Abruf: 18.01.2017]

[whatwebcando.today 2017a] whatwebcando.today: *What Web Can Do Today: Background Sync*. 2017. – URL https://whatwebcando.today/background-sync.html. – [Abruf: 18.01.2017]

[whatwebcando.today 2017b] whatwebcando.today: *What Web Can Do Today: Device Memory*. 2017. – URL https://whatwebcando.today/memory.html. – [Abruf: 14.02.2017]

[whatwebcando.today 2017c] WHATWEBCANDO.TODAY: *What Web Can Do Today: Pointer Adaptation*. 2017. – URL https://whatwebcando.today/pointer-adaptation.html. – [Abruf: 18.01.2017]

[whatwebcando.today 2017d] WHATWEBCANDO.TODAY: *What Web Can Do Today: Realtime*. 2017. – URL https://whatwebcando.today/realtime.html. – [Abruf: 18.01.2017]

[whatwebcando.today 2017e] WHATWEBCANDO.TODAY: *What Web Can Do Today: Scheduler*. 2017. – URL https://whatwebcando.today/scheduler.html. – [Abruf: 18.01.2017]

[whatwebcando.today 2017f] WHATWEBCANDO.TODAY: *What Web Can Do Today: Wake-Lock*. 2017. – URL https://whatwebcando.today/wake-lock.html. – [Abruf: 20.01.2017]

[Wikipedia 2018] WIKIPEDIA: *Programming languages used in most popular websites*. 2018. – URL https://en.wikipedia.org/wiki/Programming_languages_used_in_most_popular_websites. – [Abruf: 06.03.2018]

[Wolber 2018] WOLBER, Andy: *How to find and install Progressive Web Apps*. 2018. – URL https://www.techrepublic.com/article/how-to-find-and-install-progressive-web-apps/. – [Abruf: 29.01.2017]

[Xanthopoulos und Xinogalos 2013] XANTHOPOULOS, Spyros ; XINOGALOS, Stelios: A comparative analysis of cross-platform development approaches for mobile applications. In: *Proceedings of the 6th Balkan Conference in Informatics* ACM (Veranst.), 2013, S. 213–220

[Zola 2017] ZOLA, Andrew: *Will Progressive Web Apps Ever Replace Native Mobile Apps*. 2017. – URL https://intersog.com/blog/will-progressive-web-apps-ever-replace-native-mobile-apps/. – [Abruf: 03.01.2017]

[Zolciak 2017] ZOLCIAK, Antoni: *Progressive Web Applications: The Thing to Consider When Short on Resources*. 2017. – URL https://insanelab.com/blog/web-development/progressive-web-apps-pwa-vs-native/. – [Abruf: 16.10.2017]

Anhang A.

Ladezeiten aller Testdurchläufe

NP Web App	2G (280 Kbit/s)	3G (1.6 Mbit/s)	LTE (12 Mbit/s)
Testdurchlauf 1	17,200	5,311	1,661
Testdurchlauf 2	18,767	5,396	1,736
Testdurchlauf 3	20,678	5,632	1,836
Testdurchlauf 4	20,565	5,453	2,074
Testdurchlauf 5	17,254	6,416	1,962

Tabelle A.1.: Ladezeiten in Sekunden aller Testdurchläufe der Newsportal Web App bei erstmaligem Seitenaufruf

NP PWA	2G (280 Kbit/s)	3G (1.6 Mbit/s)	LTE (12 Mbit/s)
Testdurchlauf 1	24,443	5,623	1,839
Testdurchlauf 2	21,452	5,671	1,481
Testdurchlauf 3	21,537	5,763	1,523
Testdurchlauf 4	17,148	5,589	1,585
Testdurchlauf 5	20,292	5,583	1,758

Tabelle A.2.: Ladezeiten in Sekunden aller Testdurchläufe der Newsportal PWA bei erstmaligem Seitenaufruf

VP Web App	2G (280 Kbit/s)	3G (1.6 Mbit/s)	LTE (12 Mbit/s)
Testdurchlauf 1	50.850	7.591	2.044
Testdurchlauf 2	51.010	7.089	2.397
Testdurchlauf 3	53.498	6.133	2.094
Testdurchlauf 4	53.510	7.223	2.103
Testdurchlauf 5	51.417	7.016	3.598

Tabelle A.3.: Ladezeiten in Sekunden aller Testdurchläufe der Videoportal Web App bei erstmaligem Seitenaufruf

VP PWA	2G (280 Kbit/s)	3G (1.6 Mbit/s)	LTE (12 Mbit/s)
Testdurchlauf 1	57.551	7.012	1.999
Testdurchlauf 2	53.411	6.232	3.414
Testdurchlauf 3	54.140	6.976	3.458
Testdurchlauf 4	59.767	6.194	2.104
Testdurchlauf 5	52.773	3.297	3.478

Tabelle A.4.: Ladezeiten in Sekunden aller Testdurchläufe der Videoportal PWA bei erstmaligem Seitenaufruf

NP Web App	2G (280 Kbit/s)	3G (1.6 Mbit/s)	LTE (12 Mbit/s)
Testdurchlauf 1	3,179	1,335	0,723
Testdurchlauf 2	3,170	1,293	0,721
Testdurchlauf 3	3,155	1,298	0,671
Testdurchlauf 4	3,162	1,328	0,744
Testdurchlauf 5	3,157	1,301	0,684

Tabelle A.5.: Ladezeiten in Sekunden aller Testdurchläufe der Newsportal Web App bei erneutem Seitenaufruf

NP PWA	2G (280 Kbit/s)	3G (1.6 Mbit/s)	LTE (12 Mbit/s)
Testdurchlauf 1	3,943	1,397	0,684
Testdurchlauf 2	4,008	1,409	0,673
Testdurchlauf 3	3,983	1,399	0,684
Testdurchlauf 4	3,971	1,427	0,686
Testdurchlauf 5	3,944	1,426	0,696

Tabelle A.6.: Ladezeiten in Sekunden aller Testdurchläufe der Newsportal PWA bei erneutem Seitenaufruf

VP Web App	2G (280 Kbit/s)	3G (1.6 Mbit/s)	LTE (12 Mbit/s)
Testdurchlauf 1	8.857	4.634	0.878
Testdurchlauf 2	8.640	5.465	0.826
Testdurchlauf 3	8.840	1.539	0.861
Testdurchlauf 4	7.952	2.371	0.839
Testdurchlauf 5	8.149	4.627	0.780

Tabelle A.7.: Ladezeiten in Sekunden aller Testdurchläufe der Videoportal Web App bei erneutem Seitenaufruf

VP PWA	2G (280 Kbit/s)	3G (1.6 Mbit/s)	LTE (12 Mbit/s)
Testdurchlauf 1	8.182	4.637	1.003
Testdurchlauf 2	8.745	4.006	0.879
Testdurchlauf 3	7.470	4.634	0.995
Testdurchlauf 4	9.203	1.539	0.964
Testdurchlauf 5	8.787	5.909	0.998

Tabelle A.8.: Ladezeiten in Sekunden aller Testdurchläufe der Videoportal PWA bei erneutem Seitenaufruf

Anhang B.

Mockups des Videoportals

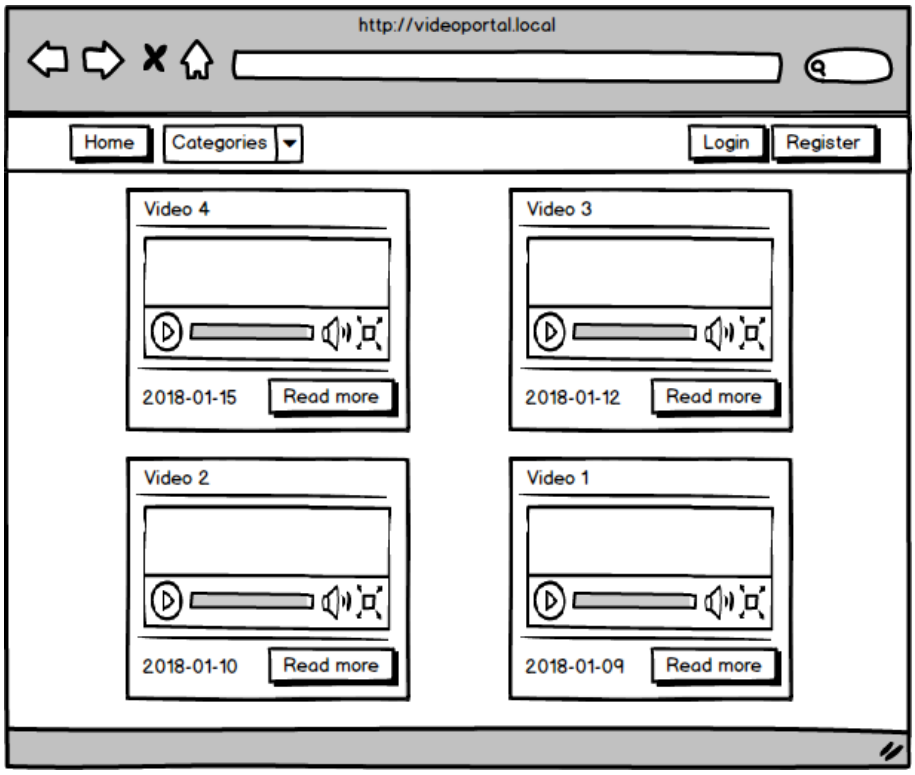


Abbildung B.1.: Mockup der Startseite des Videoportals (eigene Darstellung)

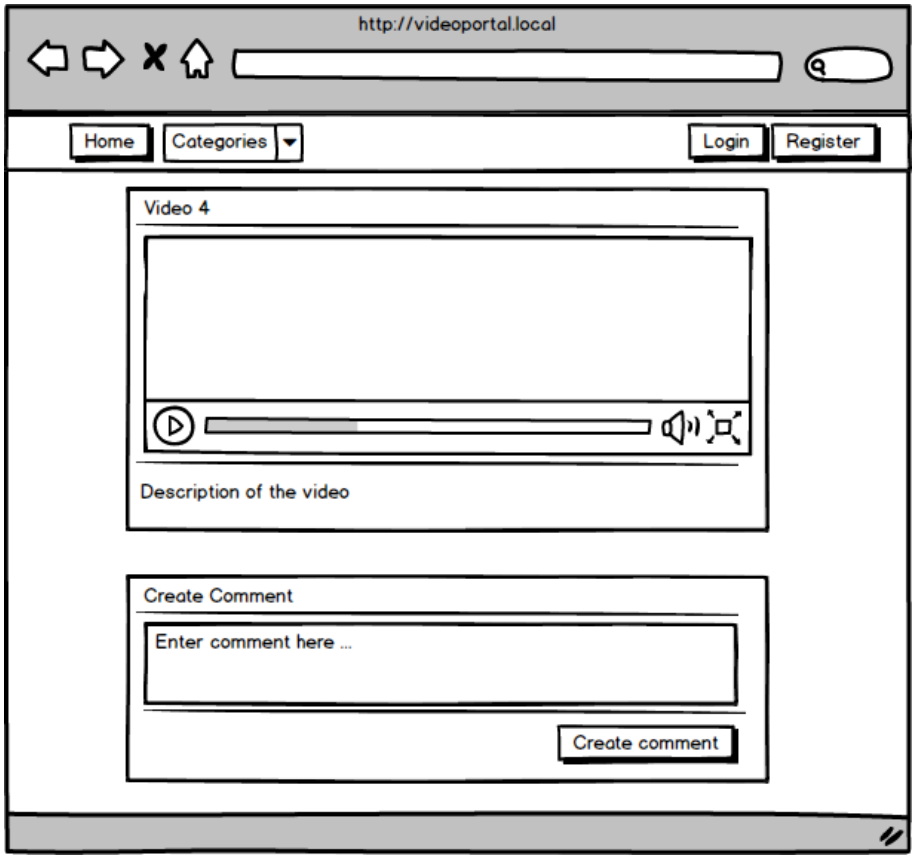


Abbildung B.2.: Mockup der Detailseite eines Videos im Videoportal (eigene Darstellung)

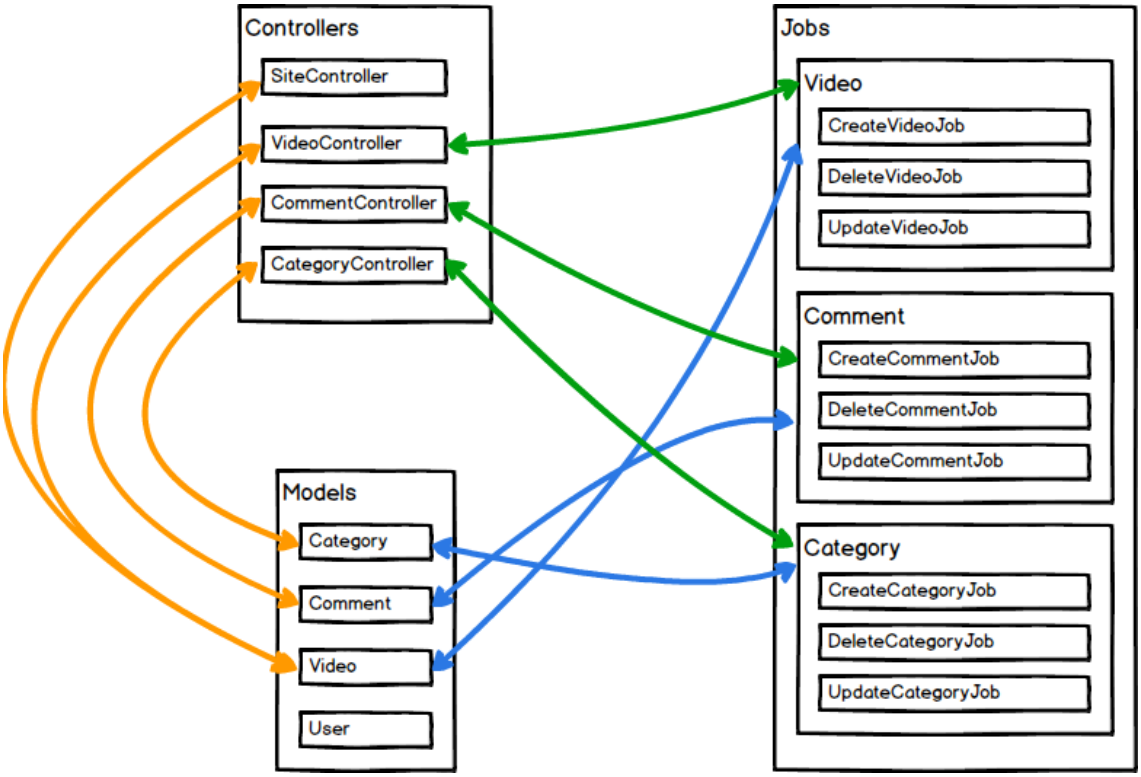


Abbildung B.3.: Software-Architektur des Videoportals (eigene Darstellung)

Anhang C.

Quellcodauszüge des Videoportals

```
importScripts('/workbox-sw.prod.v2.1.2.js');

// Note: Ignore the error that Glitch raises about WorkboxSW being
↪  undefined.
const workbox = new WorkboxSW({
  skipWaiting: true,
  clientsClaim: true
});

workbox.router.registerNavigationRoute("/");

var networkFirstHandler = workbox.strategies.networkFirst({
  cacheName: 'default-cache'
});

workbox.router.registerRoute(
  /.*/,
  function(args) {
    return networkFirstHandler.handle(args).then(function(response) {
      if (typeof response == 'undefined') {
        // Offline
        return caches.match('offline');
      }

      return response;
    });
  }
);
```

```
28
29
30  self.addEventListener('push', (event) => {
31    const title = 'VideoPortal';
32    const options = {
33      body: event.data.text()
34    };
35    event.waitUntil(self.registration.showNotification(title, options));
36  });
37
38
39  workbox.precache([]);
```

Quellcode C.1: Service-worker.js aus dem PWA Videoportal

```
1   navigator.serviceWorker &&
    ↪  navigator.serviceWorker.ready.then(function(serviceWorkerRegistration) {
2     serviceWorkerRegistration.pushManager.getSubscription()
3       .then(function(subscription) {
4         // subscription will be null or a PushSubscription
5         if (subscription) {
6           console.info('Got existing', subscription);
7           window.subscription = subscription;
8           return;  // got one, yay
9         }
10
11        const applicationServerKey = urlB64ToUint8Array(publicKey);
12        serviceWorkerRegistration.pushManager.subscribe({
13            userVisibleOnly: true,
14            applicationServerKey,
15        })
16          .then(function(subscription) {
17            console.info('Newly subscribed to push!', subscription);
18            window.subscription = subscription;
19            console.info('Subscription: ', subscription.endpoint);
20            console.log(subscription.getKey('auth'));
```

```
21
22          var endpoint;
23          var key;
24          var authSecret;
25
26          var rawKey = subscription.getKey ? subscription.getKey('p256dh') :
            ↪ '';
27          key = rawKey ?
28                btoa(String.fromCharCode.apply(null, new
                ↪ Uint8Array(rawKey))) :
29                '';
30          var rawAuthSecret = subscription.getKey ?
            ↪ subscription.getKey('auth') : '';
31          authSecret = rawAuthSecret ?
32                       btoa(String.fromCharCode.apply(null, new
                         ↪ Uint8Array(rawAuthSecret))) :
33                       '';
34
35          endpoint = subscription.endpoint;
36          fetch('/api/subscribe', {
37            method: 'post',
38            headers: {
39              'Content-type': 'application/json'
40            },
41            body: JSON.stringify({
42              endpoint: subscription.endpoint,
43              key: key,
44              authSecret: authSecret,
45            }),
46          });
47
48        });
49    });
50 });
```

Quellcode C.2: Auszug aus dem Service Worker des Videoportals zur Erstellung eines Subscribe-Objektes

Anhang D.

Änderungen an der Softwarearchitektur

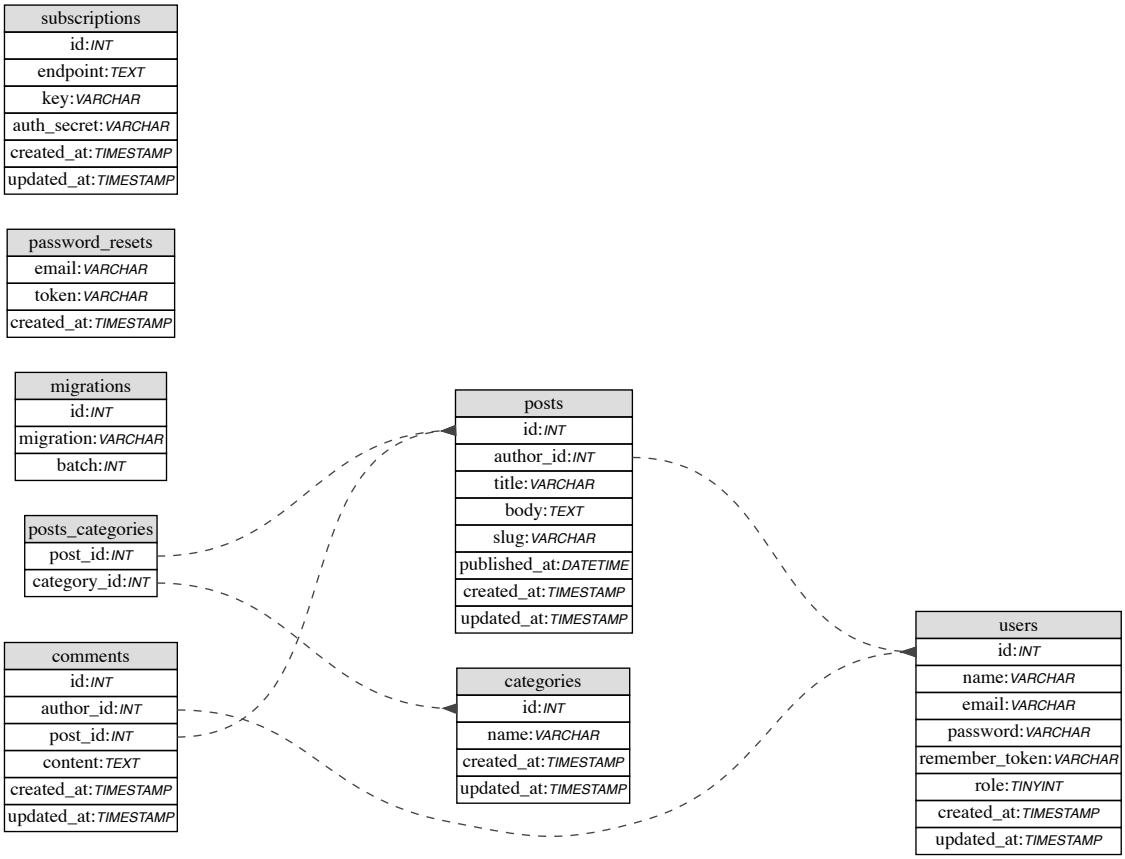


Abbildung D.1.: ER-Diagramm des PWA Nachrichtenportals (eigene Darstellung)

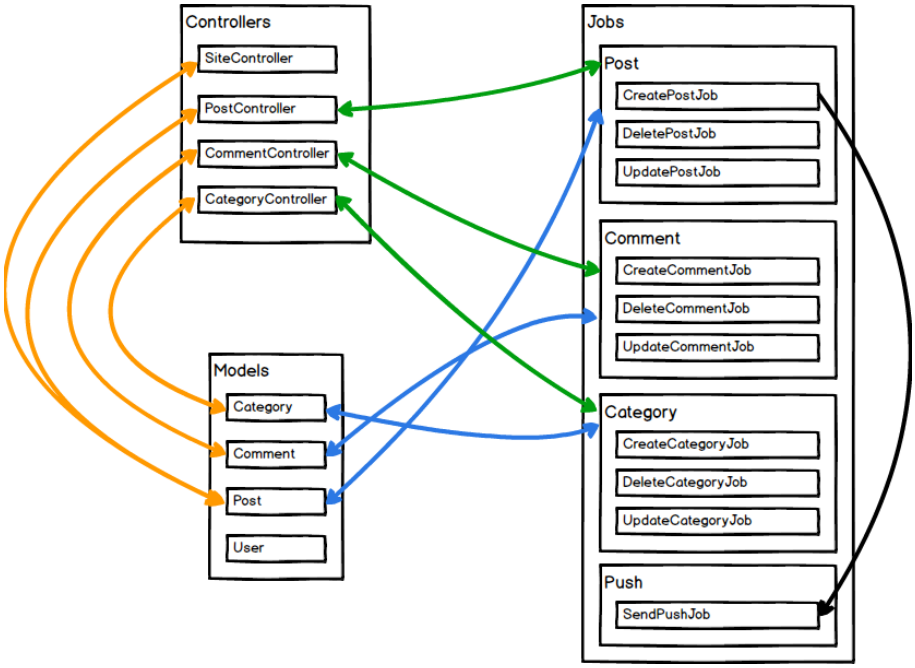


Abbildung D.2.: Softwarearchitektur des PWA Nachrichtenportals (eigene Darstellung)