MASTERARBEIT

MOBILE ANWENDUNGEN – PROGRESSIVE WEB APPLIKATIONEN ALS ERSATZ ZU NATIVEN ANWENDUNGEN

ausgeführt am

Studiengang

Informationstechnologien und Wirtschaftsinformatik

Von: Markus Saurer

Personenkennzeichen: 2010320012

Hartberg, am 15. Dezember 2021

Unterschrift

EHRENWÖRTLICHE ERKLÄRUNG

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die benutzten Quellen wörtlich zitiert sowie inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

.....

Unterschrift

DANKSAGUNG

An erster Stelle möchte ich mich bei allen Menschen bedanken, die mich über das ganze Masterstudium hinweg begleitet und gefördert haben. Dazu zählt ganz besonders meine Familie, die mir in jeder Zeit mit viel Zuspruch und Geduld beigestanden hat.

Daneben möchte ich mich noch bei meinen Freundinnen und Freunden bedanken, welche mir während dieser Zeit mit gutem Rat zur Seite gestanden sind und mich immer unterstützt haben.

Nicht zuletzt gebührt ein besonderer Dank auch meinem Betreuer Herrn DI. Christoph Pangerl. Bedanken möchte ich mich für das konstruktive Feedback, die freie Gestaltung und den großer Entscheidungsspielraum der Zusammenarbeit.

KURZFASSUNG

Mobile Anwendungen bekommen eine immer größere Relevanz zugesprochen. Wesentliche Gründe hierfür sind die Offline-Fähigkeit, uneingeschränkte Geräte-Zugriffe, App Stores zur Bereitstellung, intuitive Interaktionen und vieles mehr. Um diese Charakteristiken zu nutzen, hat sich bisher der native Ansatz durchgesetzt, wobei für jedes Betriebssystem eine eigener Quellcode geschrieben werden muss. Genau hier setzen Progressive Web-Anwendungen an. Diese Art von Technologie verspricht eine native Benutzererfahrung, mit dem entscheidenden Unterschied, dass nur eine einzige Quellcode-Basis notwendig ist. Somit kann der Entwicklungs- und Wartungsaufwand mit den umhergehenden Kosten intensiv verringert werden.

Diese Arbeit betrachtet aus diesem Grund die Möglichkeiten und Einschränkungen von Progressive Web-Anwendungen gegenüber nativen iOS- und Android-Anwendungen.

Um dieses Ziel zu erreichen, wird ein Benchmarking für die unterschiedlichen Konzepte durchgeführt. Die Basis bilden dabei eine Literaturanalyse und eine Implementierung einer Progressive Web-Anwendung mit den Mindestanforderungen. Im Anschluss werden Vergleichskriterien anhand der ISO 25010 aufgestellt, welche als Norm für die Qualitätskriterien von Software, IT-Systemen und Software-Engineering bekannt sind. Die Technologieansätze werden daraufhin mithilfe dieser Kriterien evaluiert und einander gegenübergestellt.

Das Resultat dieser Arbeit ist eine Entscheidungsmatrix, welche für jede Art von Entwicklungsprojekten mobiler Anwendungen anwendbar ist. Diese Matrix hilft, den optimalen Ansatz für vorliegende Projekt-Anforderungen zu bestimmen. Grundsätzlich konnte in dieser Master-Thesis ein aufsteigender Trend für Progressive Web-Anwendungen beim mobilen Chrome und Samsung Internet Browser evaluiert werden. Nichtsdestotrotz sind seitens Apple mit dem mobilen Safari Browser noch wesentliche Einschränkungen vorhanden, was den Ersatz durch Progressive Web-Anwendungen nicht in allen Fällen gewährleisten kann.

ABSTRACT

The importance of mobile applications has been increasing steadily, mainly due to their offline capability, unrestricted device access, availability on app stores, intuitive interactions, and more. Therefore, the native development approach has prevailed which involves writing a separate source code for each operating system. This is exactly where progressive web applications come into play. This type of technology promises a native user experience but only requires a single source code base. Thereby significantly decreasing the development and maintenance effort and the associated costs.

For this reason, this work examines the possibilities and limitations of progressive web applications compared to native iOS and Android applications.

In order to achieve this goal, a benchmarking was conducted for the different concepts based on a literature analysis and an implementation of a progressive web application with the minimum requirements. Comparison criteria were then established based on ISO 25010, the standard for the quality criteria of software, IT systems and software engineering. The technological approaches were then evaluated and compared with the help of these criteria.

The outcome of this work is a decision matrix that can be used to determine the optimal approach for the project requirements of any type of mobile application development project. The present results showed that progressive web applications are suitable for the mobile Chrome and Samsung Internet browsers, while significant limitations with the mobile Safari browser mean that progressive web applications cannot at present be deployed for all Apple use cases.

INHALTSVERZEICHNIS

1 EINFÜHRUNG

„ Progressive Web Apps - Websites that took all the right vitamins.‘ – PWA Founders of Google“

Eine Vielzahl an Studien veranschaulicht, dass die Anzahl der im Internet verbrachten Stunden auf mobilen Endgeräten von Jahr zu Jahr weiter zu nimmt. Vor allem die Trennung zwischen Webseiten, welche in einem mobilen Browser aufgerufen werden, und mobile Apps, welche direkt auf dem Endgerät installiert werden, wird immer deutlicher. Laut einer Studie von EMarketer (Yoram Wurmser 2020) werden die verbrachten Stunden zu ca. 90% in mobilen Anwendungen gegenüber mobilen Browsern verbracht. Die Wichtigkeit einer installierbaren Anwendung ist somit allgemeingültig. Dieser aufsteigende Trend zeigt sich auch in der Anzahl der verfügbaren mobilen Anwendungen am Markt. Im Juni 2020 wurden bis zu diesem Datum 2,96 Millionen Anwendungen im Google Play Store veröffentlicht (statista 2020a), beim Apple App Store sogar 4,4 Millionen (statista 2020b).

1.1 Problemstellung

Diese Fakten verdeutlichen nur einen Bruchteil des noch immer aufstrebenden Marktes von mobilen Anwendungen. Deshalb zeigen sich viele Möglichkeiten für potenzielle Geschäftsideen auf. Konnte eine Geschäftsidee und ein Businessplan in diesem Bereich bestimmt werden, wird sich früher oder später die Frage des gewünschten Technologieansatzes herausstellen. Im besten anzunehmenden Fall soll die gewählte Technologie alle Funktionalitäten abdecken, der Entwicklungsaufwand so gering wie möglich gehalten, die Sicherheits- und Performanceanforderungen jederzeit erreicht und eine gute User Experience geschaffen werden. In weiterer Folge soll es dann auch möglich sein, die Anwendung bestmöglich zu monetarisieren. Der Prozess der Technologieentscheidung legt somit einen entscheidenden Grundstein einer erfolgreichen mobilen Anwendung. Die populärsten Technologien reichen hier von Web, Native, Hybride, Cross-Plattform und Progressive Web-Anwendungen. Im Kapitel 2 werden diese näher betrachtet und gegenübergestellt.

1.2 Zielsetzung und Einschränkungen der Arbeit

Da der Markt und dessen Zielgruppe an installierbaren Anwendungen immer größer wird, untersucht diese Arbeit inwieweit es möglich ist, den „Alleskönner“ die native Anwendung durch Progressive Web-Anwendungen zu ersetzen bzw. analysiert die vorliegende Arbeit vorhandene Einschränkungen und Voraussetzungen. Da dieser Technologieansatz in den letzten Jahren einen immensen Aufschwung erhalten haben und immer mehr native Funktionen freigegeben werden, wird dieser Technologieansatz herangezogen.

Dabei kommt es zu Einschränkungen bei den Betriebssystemen und den mobilen Browseranbietern. Auf der Seite der nativen Anwendungen werden die Betriebssysteme IOS und Android betrachtet, da diese im Mai 2021 bereits 99% des Marktanteils von mobilen Betriebssystemen eingenommen haben (S. O'Dea 2021a). Bei Progressive Web-Anwendungen sollen der mobile Chrome Browser, der mobile Samsung Browser und der mobile Safari Browser die Grundlage bilden. Diese Einschränkung wurde getroffen, weil diese drei Browser knapp 97% im europäischen Raum und 95% weltweilt aller genutzten Browser einnehmen und die beiden Betriebssysteme IOS und Android abgreifen. Laut einer Statitisk von statcounter (Statcounter 2021) lag im Juni 2021 der Marktanteil von mobilen Browsern zu knapp 59% bei Chrome, zu ca. 30% bei Safari bzw. zu ca. 8% bei Samsung Internet. Um diese Studie so aktuell wie möglich zu gestalten, werden die letzten “stable versions” von den Browseranbietern verwendet. Bei Chrome ist das zum Zeitpunkt der Verfassung dieser Arbeit Version 95.x, beim Safari Browser die Version 15.x und beim Samsung Internet Browser die Version 15.x. Grundsätzlich könnte man auf jeder Plattform den Chrome Browser in Betracht ziehen, da dieser mit iOS und Android Geräten kompatibel ist. Jedoch würde man dabei die Präferenzen der potenziellen Endkundinnen und Endkunden einschränken. Darüber hinaus ist dies aus technischer Sicht nicht möglich, da der mobile Chrome Browser für IOS keine Möglichkeit zur Installation auf diesem Betriebssystem bietet (Google Support 2021f).

In erster Linie werden die zwei Entwicklungsansätze genauer betrachtet, inwiefern diese relevant in der mobilen Anwendungsentwicklung sind bzw. welche Anforderungen und Einschränkungen vorherrschen. Um vergleichbare Qualitätsmerkmale zu schaffen, wird als Grundlage die ISO Norm 25010 - System und Software-Engineering – Qualitätskriterien und Bewertung von System und Softwareprodukten (SQuaRE) verwendet (ISO/IEC 25010:2011). Diese Qualitätskriterien sind im Anschluss knapp zusammengefasst:

Funktionalität: Beim Punkt der Funktionalität ist es das Ziel der Entwicklerinnen und Entwickler, alle funktionalen Anforderungen abzudecken. Deshalb wird dieser Teil in dieser Arbeit aufgenommen. Prinzipiell wird hier die Vollständigkeit hinsichtlich von Software- und Hardwarezugriffen behandelt.

Performance: Darüber hinaus wird die Performance von nativen Anwendungen gegenüber Progressive Web-Anwendungen näher beleuchtet, wo es zu Einschränkungen der Reaktions- und Zugriffszeiten kommen kann bzw. wird untersucht, wie sich die Technologien auf den Energieverbrauch der Softwarelösungen auswirken.

Portierbarkeit: Dieses Merkmal betrachtet die Effektivität und Effizienz, mit der ein System bereitgestellt und installierbar gemacht werden kann. Des Weiteren wird betrachtet, ob und wie eine Software plattformunabhängig ist.

Wartung: Dieses Qualitätsmerkmal befasst sich damit, wie effizient eine Softwarelösung modifiziert, angepasst oder verbessert werden kann.

Sicherheit: Ein weiterer nicht wegzudenkender Punkt ist das Gewährleisten der Sicherheit. Aus diesem Grund befasst sich die ISO 25010 mit Sicherheitsaspekten, wie dem Zugriff auf Hardwarekomponenten, der HTTPS Pflicht etc., um die Software und die Benutzerinnen und Benutzer zu schützen.

Kompatibilität: Darüber hinaus wird die Kompatibilität als Qualitätsmerkmal nach ISO 25010 näher analysiert. Ziel dieses Merkmals ist es, eine Software zu entwickeln, welche es möglich macht, unterschiedliche Endgeräte wie auch Plattformen aufeinander bestmöglich abzustimmen.

Verlässlichkeit: Das Qualitätsmerkmal der Verlässlichkeit beschäftigt sich mit der Verfügbarkeit bei unvorhersehbaren Ereignissen, der Fehlertoleranz und weiteren Faktoren.

Usability: In einem weiteren Schritt werden die Technologien anhand ihres Designspektrums erläutert, um das gewünschte User Interface und die bestmögliche User Experience zu schaffen.

Um die erlangten Informationen zu den Qualitätsmerkmalen als Grundlage für Technologieentscheidungen nutzen zu können, soll als Ergebnis dieser Arbeit eine Entscheidungsmatrix geschaffen werden, welche auf Basis der acht Kernkriterien der ISO 25010 Norm die Technologieansätze der Progressive Web-Anwendungen und der nativen Anwendungen vergleicht.

1.3 Methodik

Für diese Masterarbeit wird eine Literaturanalyse durchgeführt und anhand dieser Informationen, wie bereits erwähnt, eine Entscheidungsmatrix nach ISO 25010 entwickelt, welche die Auswahl der gewünschten Technologie erleichtern soll.

Anhand der Literaturanalyse wird das „state of the field“ geklärt. Hier wird primär der Fokus auf die technologischen Aspekte, Voraussetzungen, Möglichkeiten und Einschränkungen der Entwicklungsansätze geklärt. Die Literaturanalyse wurde, wie in Abbildung 1 ersichtlich, nach dem Modell von Jan vom Brocke durchgeführt (Jan vom Brocke et al. 2009).

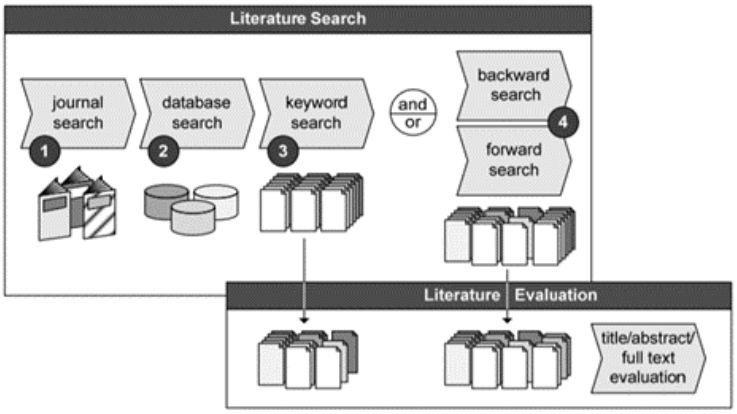

Abbildung 1: Literatursuche und Beurteilung nach Brocke (Jan vom Brocke, 2009)

Dabei wurde im ersten Schritt nach elektronischen Büchern und Journals in den Bibliotheken der FH Campus 02, Technische Universität Wien/Graz und Wirtschaftsuniversität Wien gesucht. Ergänzend wurde in verschiedensten Literaturdatenbanken nach relevanten Publikationen gesucht. Des Weiteren wurden qualitative Onlineressourcen aus Google Suchanfragen verwendet. Da es sich um eine techniknahe Arbeit handelt, wurden offizielle Online-

Dokumentation der Softwareplattformen verwendet. Um diesen Prozess so effizient wie möglich zu gestalten, wurde die Recherche mit inhaltsrelevanten Suchstrings durchgeführt. Nach Ablauf der Literatursuche wurde eine Menge an zitierwürdigen Schriften identifiziert. Nachdem die Kriterien nach ISO 25010 definiert und evaluiert wurden, verfolgt diese Arbeit das Ziel eine Entscheidungsmatrix anhand dieser Kriterien zu erstellen.

1.4 Aufbau der Arbeit

Die vorliegende Arbeit besteht aus fünf Kapiteln. Das erste Kapitel bildet die Einführung in die Masterthesis. Hier wird die konkrete Problemstellung, die Zielsetzung, die Methodik und der Aufbau der Arbeit betrachtet. Im nächsten Kapitel werden die Grundlagen der verschiedenen Technologien behandelt. Dabei wird in erster Linie eine Technologie-Übersicht über klassische Web-Anwendungen, Cross Plattform und Hybride Anwendungen gegeben. Da der Fokus dieser Arbeit bei nativen und Progressive Web-Anwendungen liegt, wird in weiterer Folge native Anwendungen im Bereich IOS und Android und Progressive Web-Anwendungen im Detail erläutert. Hier sollen Begriffe wie Service Worker, App Shell, App Manifest usw. geklärt werden. Darüber hinaus werden Codeausschnitte einer Implementierung Teil dieser Ausarbeitung sein. Nachdem die Grundlagen der Technologien geklärt sind, werden im Kapital 3 die Kriterien zur Gegenüberstellung definiert und eingeteilt. Im Kapitel 4 soll dabei analysiert werden, inwiefern und mit welchen Einschränkungen die Erfüllung der Kriterien möglich ist. Nachdem alle vordefinierten nicht-funktionalen und funktionalen Anforderungen gegenüber des Technologieansatzes gegenüberstellt wurden, werden die Ergebnisse ausgewertet und daraus im Anschluss eine Entscheidungsmatrix entwickelt. Diese soll als Basis dienen, um den richtigen Technologieansatz zu finden. Nachdem eine Matrix entworfen wurde, werden im letzten Kapitel die Ergebnisse zusammengefasst und ein Ausblick für weitere wissenschaftliche Beiträge gegeben.

2 GRUNDLAGEN

Nachdem im Kapitel 1 ein Überblick über die Arbeit gegeben wurde, werden in diesem Schritt die Grundlagen der verschiedenen Technologieansätze geklärt. Dabei werden Web-Anwendungen, Hybrid Anwendungen, Cross Plattform Anwendungen, Native Anwendungen und Progressive Web-Anwendungen analysiert und aufgeschlüsselt. Ein besonderer Schwerpunkt wird bei den Progressive Web-Anwendungen liegen, da der Fokus auf dieses Konzept gelegt wurde. Dabei werden die entscheidendsten Bestandteile, Anforderungen und Funktionalitäten dieser Technologie geklärt.

2.1 Web-Anwendungen

Die klassische Technologie stellt die Web-Anwendung dar. Diese ist eine vollständig browserbasierte Anwendung, welche mithilfe eines Webservers in der Regel Javascript Code ausführt und mittel HTML und CSS aufgebaut wird. Wird eine Web-Anwendung aufgerufen, ist das nur über einen Browser auf dem Endgerät möglich.

Dieser Ansatz ist nach dem Client-Server-Modell aufgebaut. Im Gegenzug zu klassischen Desktop-Anwendungen werden Web-Anwendungen nicht auf den lokalen Rechnern des Benutzers installiert, sondern der Datenaustausch findet mit einem Webserver statt, welcher auf dem lokalen oder einem entfernten Rechner aufgesetzt werden kann (Rossi 2008).

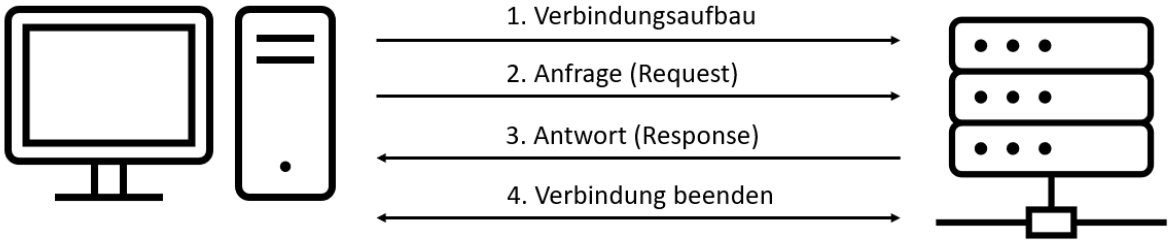

Abbildung 2: Client-Server-Modell (Eigene Darstellung, 2020)

Wie die Abbildung 2 vereinfacht dargestellt, wird in erster Linie eine Transmission Control Protocol (TCP) bzw. User Datagram Protocol (UDP) Verbindung zwischen den zwei Bestandteilen hergestellt. Nachdem eine Verbindung erfolgreich hergestellt wurde, stellen Benutzer und Benutzerinnen (Client) an den Server eine Anfrage (Request), um die gewünschten Daten, wie zum Beispiel Webseiteninhalte, zu erhalten. Der Server liefert dann die angefragten Datenpaketen (Response) an den Benutzer aus. Diese Kommunikation läuft dabei meist über das Hypertext Transfer-Protokoll (HTTP). Nachdem die gewünschten Datenpakete ausgeliefert worden sind, wird die Verbindung wieder geschlossen (Rossi 2008).

Besondere Charakteristiken dieser Technologie sind die einmalige plattformübergreifende Entwicklung und die einfache Erreichbarkeit, sofern das Endgerät einen gängigen Browser besitzt. Diese Tatsache stellt aber gleichzeitig auch einen wesentlichen Nachteil dar, da die verschiedenen am Markt befindlichen Browser spezifische Behandlung bedürfen. Somit stellt es in gewissen Aspekten einen wesentlichen Unterschied dar, ob eine Web-Anwendung in einen Chrome oder Safari Browser geöffnet wird. Darüber hinaus gibt es wesentliche Einschränkungen beim Zugriff auf native Funktionen des Endgerätes. Nichtsdestotrotz ist die neueste Generation mobiler Browser in der Lage, fortschrittlichere Webtechnologien zu verwenden, darunter Funktionen von HTML5, CSS 3, eine Reihe von umfangreichen JavaScript-APIs, die den Weg für webbasierte mobile Anwendungen ebnen. W3C hat hierbei eine detaillierte API-Beschreibungen der unterschiedlichen Funktionen. Bekannte Beispiele sind die Geolocation API, File API, Generic Sensor API und viele mehr. Ein weiterer Nachteil bei einer klassischen Web-Anwendung ist, dass es keine Möglichkeit der Installation bzw. der Offline Nutzung gibt (Rossi 2008).

2.2 Hybride Anwendungen

Im Gegensatz zu Web-Anwendung beruhen Hybride Anwendungen auf dem Ansatz, dass native Funktionen mit Webinhalten vereint werden und diese über die jeweiligen App Stores angeboten werden können. Dabei läuft diese Anwendung mit den Webtechnologien HTML, CSS und Javascript innerhalb eines Webview-Containers. Dieser Container besteht aus einem Browser mit einem nativen Wrapper, welcher es erlaubt, zusätzlich auf native API’s und Funktionen vom jeweiligen Betriebssystem zuzugreifen. Infolgedessen kann der Entwicklungsaufwand durch die plattformübergreifende Entwicklung optimiert werden. Jedoch kann es bei diesem Ansatz zu Performanceeinschränkungen kommen (Khandeparkar Anmol, Gupta Rashmi, B. Sindhya 2015).

2.3 Cross Plattform Anwendungen

Cross Plattform Anwendungen verfolgen einen ähnlichen Ansatz wie Hybride Applikationen. Anwendungen mit dieser Technologie werden auch mit den Technologien HTML, CSS und Javascript entwickelt. Jedoch unterscheiden sich Cross Plattform Applikationen dadurch, dass der Code asynchron mithilfe einer sogenannten Bridge mit den nativen Betriebssystem-Threads kommuniziert. Dabei ist es möglich, auf native UI-Elemente zuzugreifen und somit eine native User Experience zu erreichen. Zu Einschränkungen kann es hier wieder bei der Performance kommen (Biørn-Hansen et al. 2019).

2.4 Native Anwendungen

Im Gegenzug zu den oben genannten Anwendungen werden native Anwendungen speziell für ein Betriebssystem entwickelt und sind somit nur auf diesem lauffähig. Die Nutzung ist erst nach dem Downloaden und Installieren aus dem betriebssystem-spezifischen App Stores möglich. Dabei können auf alle Gerätefunktionen und Informationen zugegriffen und alle User-Interface-

Elemente des jeweiligen Betriebssystems verwendet werden. Native Anwendungen bieten den größten Umfang an Funktionen. Nichtsdestotrotz muss man die Entwicklung für jede Plattform unabhängig voneinander durchführen, was einen höheren Entwicklungsaufwand bedeutet (Lewis Shaun und Dunn Mike 2020).

Grundsätzlich wird in der Arbeit zwischen den Betriebssystemen IOS und Android unterschieden. In Tabelle 1 ist eine Gegenüberstellung der einzelnen Systeme dargestellt - in Bezug auf das dahinterstehende Unternehmen, die verwendete Programmiersprache, die angebotenen Integrierte Entwicklungsumgebung (IDE) und der App Stores.

	IOS	**Android**
Unternehmen	Apple	Google
Programmiersprache	Swift, Objective-C	Java, Kotlin
IDE	XCode	Android Studio
App Store	Apple App Store	Google Play Store

Tabelle 1: Vergleich IOS und Android

2.5 Progressive Web-Anwendung

Im Folgenden wird der Technologieansatz der Progressive Web-Anwendungen betrachtet. Im ersten Schritt dieses Kapitels werden ein allgemeiner Überblick über diesen Technologieansatz und den Aufbau erläutert. Daraufhin werden die notwendigen Core Elemente von Progressive Web-Anwendungen betrachtet, welche diesen Ansatz charakterisieren. Im nächsten Schritt wird die Kompatibilität mit verschiedenen Browsern und Betriebssystemen analysiert. Als letzten Schritt werden erfolgreiche und bekannte Implementierungen von Progressive Web-Anwendungen gezeigt. Um die Aufbereitung der Informationen praxisnah zu betrachten, wird eine solche Softwarelösung mit den Mindestanforderungen implementiert und der Quellcode beschrieben.

2.5.1 Überblick

Progressive Web-Anwendungen erlangen seit wenigen Jahren einen großen Anstieg an Beliebtheit und werden bereits von namhaften Softwareunternehmen eingesetzt. Im Allgemeinen sind Progressive Web-Anwendungen Webseiten, welche es ermöglichen, diese um native Funktionalitäten zu erweitern und als Anwendung, welche im Browser dargestellt wird, installierbar zu machen. Die Komponente baut auf einen sogenannten Service-Worker auf. Dieser zentrale Bestandteil ermöglicht es zum Beispiel Funktionalitäten offline durch Caching bereit zu stellen. Ein wesentlicher Vorteil ist die schnelle Entwicklung einer mobilen Anwendung anhand dieser Technologie, da eine Progressive Web-Anwendung mit allen gängigen Browsern betriebssystemübergreifend angezeigt werden kann. Im Gegenzug ist es jedoch nicht möglich auf alle nativen Funktionen der Betriebssysteme zuzugreifen. Um die Mindestanforderungen

solcher Softwarelösungen neben dem Service-Worker zu erlangen, sind die Core Elemente die App Shell und das App Manifest entscheidende Bestandteile dieses Technologieansatzes. In den nächsten Unterkapiteln werden diese im Detail betrachtet. (MDN Web Docs 2021m)

Bevor jedoch die Hauptbestandteile im Detail analysiert werden, werden die Schlüsselprinzipien, welche eine Progressive Web-Anwendung ausmachen und ihr diesen angesehenen Status verleihen, identifiziert. Mozilla hat dabei folgenden Charakteristiken identifiziert (MDN Web Docs 2021m):

Discoverable: Da es sich bei einer Progressive Web-App im Wesentlichen um eine Website handelt, sind diese einfach per Suchmaschinen auffindbar. Dies ist ein großer Vorteil gegenüber nativen Anwendungen, die in der Auffindbarkeit immer noch hinter Webseiten zurückbleiben. Dieser Vorteil entsteht, da Progressive Web-Anwendungen eine URL, wie eine klassische Webseite, besitzen. Somit ist es möglich, das Prinzip des Crawling und der Indexierung von Browsern anzuwenden.

Beim Crawling werden öffentliche Webseiten anhand einer dafür konzipierten Software identifiziert. Ein Beispiel für einen bekannten Crawler ist der Googlebot, der bei einer Anfrage in der Suchmaschine alle potenziellen Seiten durchsucht. Dabei ruft der Crawler die Webseiten auf und analysiert alle internen und externen Verlinkungen aus dem Quellcode, um die verschiedenen Webseiten einem Index zuzuweisen. Nachdem die Webseiten beim Crawling indexiert und den unterschiedlichen Suchmaschinen bereitgestellt wurden, wird bei einer Suchanfrage auf diesen Index zurückgegriffen. Anhand eines komplexen Algorithmus werden dabei die besten Ergebnisse mit einem Ranking geliefert. Die resultierende Reihenfolge ist ein Produkt aus unterschiedlichen Ranking-Faktoren (Google 2021).

Linkable: Einer der mächtigsten Vorteile gegenüber nativen Apps ist neben dem „Discoverable“ das Verlinken von Progressive Web-Anwendungen. Somit ist es nicht notwendig, einen App-Store zu besuchen, denn es ist ein einfacher Installationsprozess vorhanden.

Responsive: Die Benutzeroberfläche einer Anwendung soll stets dem Formfaktor und der Bildschirmgröße des Geräts entsprechen. Mit einem Responsive Design ist es somit möglich alle Bildschirmgrößen und Ausrichtungen abzudecken.

Connectivity-independent: Eine besondere Charakteristik von Progressive Web-Anwendung ist die Möglichkeit der offline Nutzung bzw. die Nutzung bei geringer Konnektivität. Dies ist aufgrund des Service Workers möglich, der mithilfe von einem Cache Inhalte ohne aktive Internet-Verbindung darstellen kann. Die detaillierte Funktionalität von Service Worker wird im anschließenden Kapitel erklärt.

Re-engageable: Benutzer von Progressive Web-Anwendungen verwenden ihre Apps zum Beispiel auf Grund von Push-Benachrichtigungen eher wieder.

Fresh: Wenn neue Inhalte veröffentlicht werden und der Benutzer mit dem Internet verbunden ist, sind diese Inhalte in der App sofort verfügbar. Somit ist es nicht notwendig, Updates direkt am Client auszuführen.

Installable: Auf dem Startbildschirm des Geräts kann eine progressive Web-App installiert werden, sodass diese sofort verfügbar ist. In Abbildung 3 ist das Installationsicon für PWAs im Chrome Browser in der Version 91 beispielhaft dargestellt - mit der Vorbedingung, dass diese Anwendung auch installierbar ist.

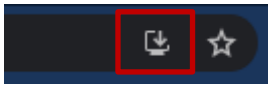

Abbildung 3: Chrome PWA Installations-Icon (Eigene Darstellung)

Safe: Da alle Netzwerkanforderungen durch den Service Worker abgefangen werden können, muss die App unbedingt über HTTPS gehostet werden, um Man-in-the-Middle-Angriffe zu verhindern.

Um in den folgenden Unterkapiteln die einzelnen Hauptelemente neben dem theoretischen Wissen auch mit Quellcode versehen zu können, wurde für diese Arbeit eine Progressive Web-Anwendung realisiert. Diese Anwendung ist eine klassische „Hello World“ Umsetzung, da der Fokus auf den Mindestanforderungen einer Progressive Web-Anwendung liegt und nicht in den Technologien HTML, CSS und Javascript. Die Anwendung sieht wie folgt aus:

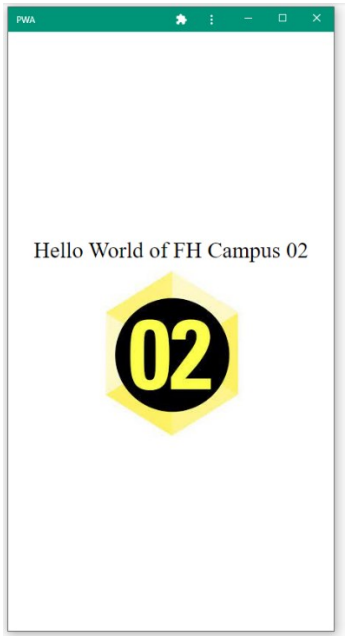

Abbildung 4. Hello World PWA (Eigene Darstellung)

Wie man der Abbildung 4 sieht, wird zur Veranschaulichung ein Bild mit einer Überschrift dargestellt, um die notwendigen Anforderungen zu erlangen.

2.5.2 Service-Worker

Wie bereits erwähnt ist eines der entscheidendsten Aspekte die Tatsache, dass Progressive Web-Anwendungen auch offline arbeiten können. Mit Service-Workern ist es möglich, Daten

anzuzeigen, die in früheren Sitzungen der Anwendungen abgerufen wurden und im sogenannten Cache gespeichert wurden. Sobald sich die Benutzerin oder der Benutzer mit dem Netzwerk verbindet, können die neuesten Daten vom Server wieder heruntergeladen werden. Diese Service-Worker-Architektur wird in Abbildung 5 systematisch dargestellt.

All diese genannten Funktionalitäten sind durch Service-Worker möglich, bei denen es sich um ereignisgesteuerte Skripte (in JavaScript geschrieben) handelt. Durch das Caching ist es möglich, statische Ressourcen zwischenzuspeichern, welche dadurch die Netzwerkanforderungen drastisch reduzieren und auch die Leistung erheblich verbessern können (Mozilla 2021a).

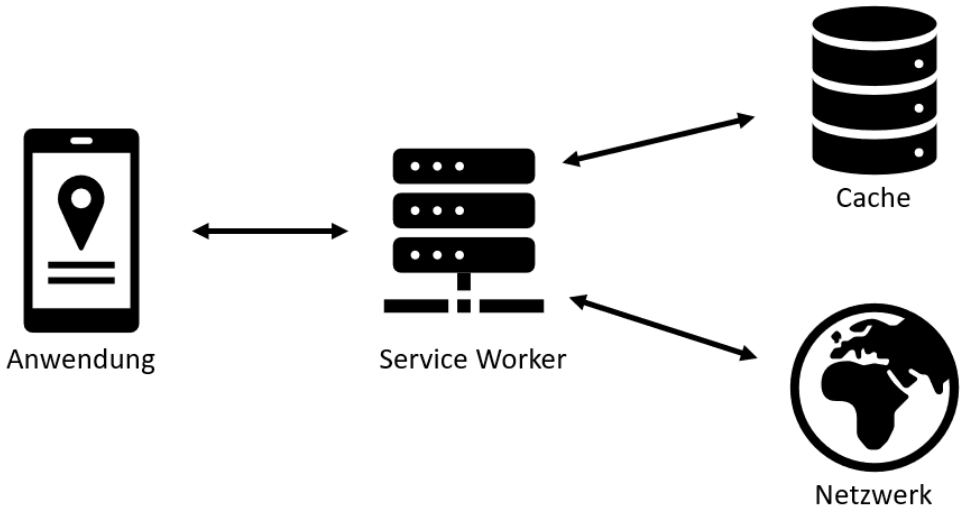

Abbildung 5: Service Worker Architektur (Eigene Darstellung)

In einem weiteren Schritt wird die praktische Umsetzung anhand des Quellcodes der Beispielanwendung beschrieben.

Um einen Service-Worker zu installieren, muss man diesen im Haupt-JavaScript-Code (index.js) registrieren. Die Registrierung teilt dem Browser mit, wo sich der Service-Worker befindet und dass dieser im Hintergrund mit der Installation beginnen kann.

```
9 lines (9 sloc)  290 Bytes
1  if ("serviceWorker" in navigator){
2      navigator.serviceWorker.register("sw.js").then(registration => {
3          console.log("SW Registered");
4          console.log(registration);
5      }).catch(error => {
6          console.log("SW Registered Failed");
7          console.log(error);
8      })
9  }
```

Abbildung 6: PWA Haupt Javascript Datei (Eigene Darstellung)

Der vorhandene Code in Abbildung 6 beginnt mit der Überprüfung der Browserunterstützung. Dabei wird das Navigator-Objekt, welches die Browserinformationen enthält, auf den

„serviceWorker“ untersucht. Damit wird dieser nur installiert, wenn es die vorliegende Browser-Version unterstützt. Die Service-Worker Datei (sw.js), welche in der Abbildung 7 darstellt ist, wird dann anhand der „navigator.serviceWorker.register“ Funktion registriert. Wenn der Service-Worker bereits installiert ist, wird das Registrierungsobjekt des derzeit aktiven Service-Worker zurück geliefert. Sobald der Browser den Service-Worker registriert hat, kann die Installation versucht werden. Dies tritt auf, wenn der Service-Worker vom Browser als neu angesehen wird, entweder weil die Webseite derzeit keinen registrierten Service-Worker hat oder weil sich dieser in der Zwischenzeit geändert hat (Google Developers 2021a).

18 lines (16 sloc) | 483 Bytes

```
self.addEventListener("install", e => {
    e.waitUntil(
        caches.open("static").then(cache => {
            return cache.addAll(["./", "./src/master.css", "./images/logo192.png"]);
        })
    )
    console.log("Install!");
})

self.addEventListener("fetch", e => {
    console.log("Fetch from Cache");
    e.respondWith(
        caches.match(e.request).then(response => {
            return response || fetch(e.request);
        })
    )

})
```

Abbildung 7: Service Worker Javascript File (Eigene Darstellung)

Wie im Quellcode der Abbildung 7 zu sehen ist, wird nach der Registrierung ein Installationsereignis im installierenden Service-Worker Javascript File (sw.js) ausgelöst. Während der Installation kann der Service Worker beispielsweise Teile einer Web-App im Voraus zwischenspeichern, sodass sie beim nächsten Öffnen sofort geladen wird. Im Beispiel Quellcode werden die Root Files index.html bzw. die master.css und das Logo in den Cache geladen. Nach diesem ersten Laden kann man dadurch die Ressourcen direkt aus dem Cache laden. In Abbildung 8 wird dieser anhand der Chrome Dev Tools das Cache Storage der Beispiel Anwendung veranschaulicht. Dabei sieht man auf der linken Seite den Cache mit dem Namen „static“ und rechts davon die zwischengespeicherten Files (Google Developers 2021a).

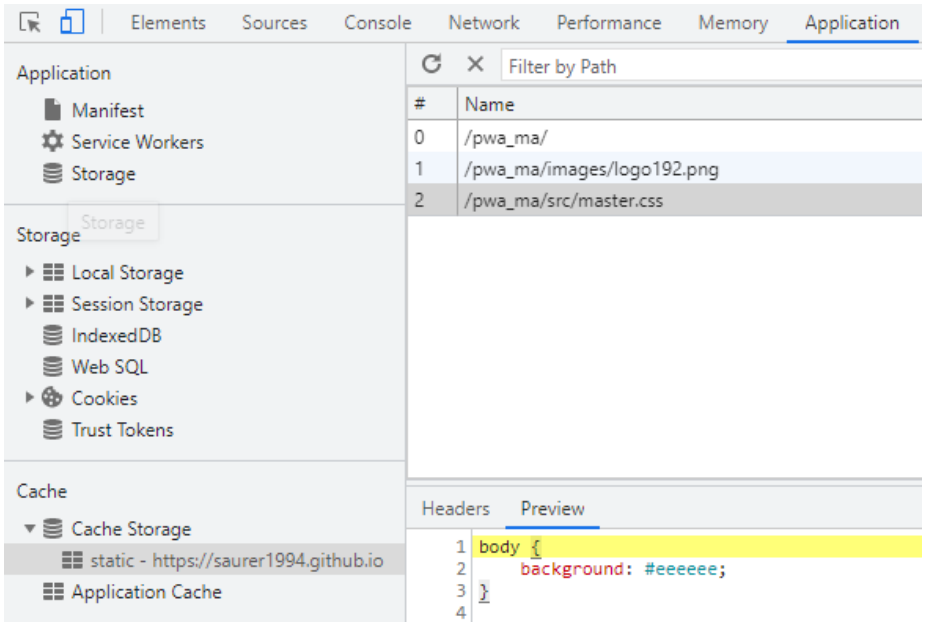

Abbildung 8: Service Worker Cache Storage (Eigene Darstellung)

In der nachfolgenden Abbildung 9 wurde die Anwendung vom Netzwerk getrennt und das Laden aus dem Cache mithilfe des Service-Workers simuliert. Nachdem die Progressive Web-Anwendung im Offline-Modus neu geladen wurde, wurden die davor in den Cache geladenen Dateien (Root Files, master.css, Icons) direkt daraus bezogen. In der Abbildung wurde es anhand der master.css Datei visualisiert, wobei man beim Punkt Status Code sehen kann, dass die Datei vom Service-Worker, also vom Cache, geladen wurde.

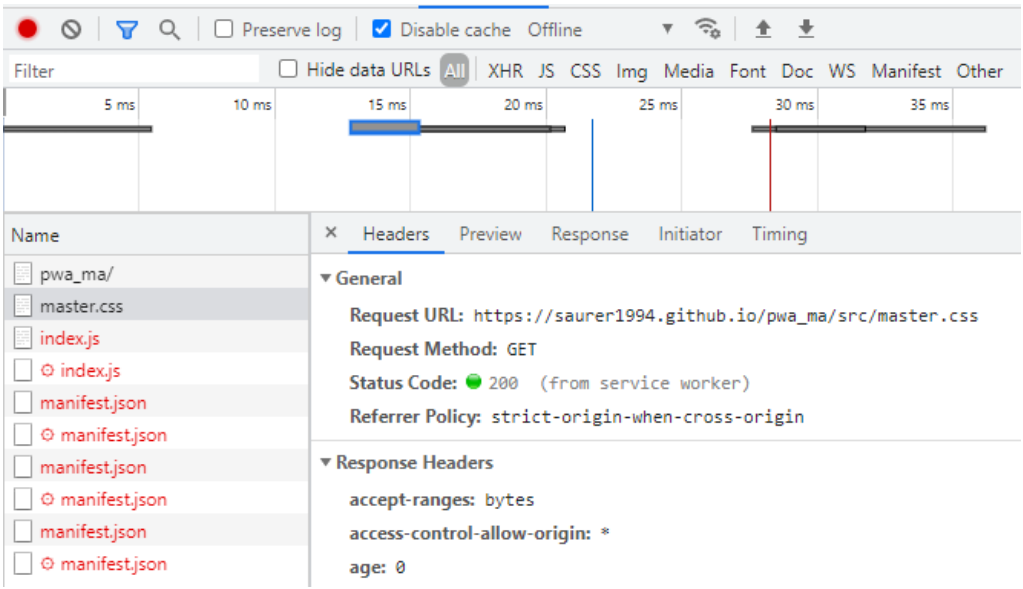

Abbildung 9: Service Worker Offline Funktionalität (Eigene Darstellung)

2.5.3 App Shell

Die App-Shell ist das Minimum an HTML, CSS und JavaScript, welches zur Umsetzung einer Anwendung erforderlich ist. Eine native mobile Anwendung enthält die App-Shell als Teil ihrer verteilbaren Inhalte, während Webseiten dies normalerweise über das Netzwerk anfordern. Progressive Web-Anwendungen schließen diese Lücke, indem sie die Ressourcen der App-Shell im Cache des Browsers speichern.

Das App-Shell-Konzept befasst sich somit damit, eine minimale Benutzeroberfläche so schnell wie möglich zu laden und diese dann zwischenzuspeichern, damit sie für spätere Besuche offline verfügbar ist, bevor dann alle Inhalte der App geladen werden. Auf diese Weise wird beim nächsten Besuch der App vom Gerät die Benutzeroberfläche sofort aus dem Cache geladen und alle neuen Inhalte werden vom Server angefordert (sofern sie nicht bereits im Cache verfügbar sind). Diese Struktur ist schnell und fühlt sich auch schnell an, da die Benutzerin bzw. der Benutzer auch offline Inhalte sieht, anstatt eine leere Seite zu laden. Sie ermöglicht dabei auch den Offlinezugriff auf die Website, wenn die Netzwerkverbindung nicht verfügbar ist. Somit fühlt sich die Website an wie eine native App mit sofortiger Interaktion und solider Leistung, während alle Vorteile von Webseiten erhalten bleiben (Google Developers 2021f).

2.5.4 App Manifest

Bei einem App Manifest handelt es sich um eine JSON-Datei, welche auf einen Webserver abgelegt, wie auch direkt in der Webanwendung eingebunden, wird.

20 lines (20 sloc) | 432 Bytes

```
{
  "name": "PWA Example",
  "short_name": "PWA",
  "start_url": "https://saurer1994.github.io/pwa_ma/",
  "background_color":"#6dcdb1",
  "theme_color": "#009578",
  "display": "standalone",
  "icons": [
    {
      "src": "images/logo192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "images/logo512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ]
}
```

Abbildung 10: App Manifest Quellcode (Eigene Darstellung)

Grundsätzlich hat das App-Manifest drei Hauptaufgaben (Mozilla 2021b):

1. **Unterscheidung zwischen klassischen Webseiten und Progressive Web-Anwendungen**: Das Vorhandensein dieser Datei ist ein entscheidendes Kriterium, um von Browser Crawlern richtig erkannt zu werden.
2. **Installierbarkeit und Erscheinungsbild**: Ist eine App-Manifest Datei vorhanden, stellt der Browser den Installationsbutton zur Anzeige. Dabei werden die erforderlichen Dateien mitinstalliert, um gewisse Funktionalitäten auch ohne Internetverbindung möglich zu machen. Des Weiteren ist es möglich das Erscheinungsbild anhand dieser Datei zu konfigurieren. Dabei können die Browserleiste, Farben und vieles mehr angepasst werden.
3. **Homeicon**: Um einen native App Eindruck zu erlangen, ist es mit der Manifest-Datei möglich, individuelle Icons und Namen zu vergeben.

Um im Anschluss die Konfigurationsmöglichkeiten zu betrachten, werden diese einzeln kurz beschrieben (Mozilla 2021b):

lang: Anhang von „lang“ kann die Sprache angegeben werden, in welcher die Anwendung verfasst wurde bzw. welche die Default-Sprache bei mehrsprachigen Progressive Web-Anwendungen ist.

short_name: Dies ist der Name, welcher in der Titelleiste der Anwendung angezeigt wird. Im vorhanden Quellcode der Softwarelösung wurde dabei “PWA” gewählt und wird somit wie folgt angezeigt (Abbildung 10):

Abbildung 11: Titelleiste short_name (Eigene Darstellung)

name: Dies ist der lesbare Name, welcher inklusive dem App Icon angezeigt wird. Dabei wurde dieser im Beispiel Quellcode mit “PWA Example” benannt und könnte wie folgt aussehen (Abbildung 11):

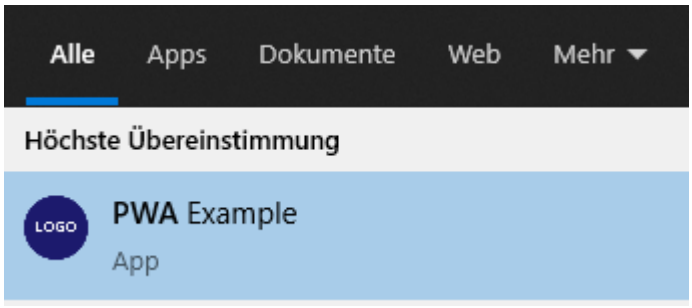

Abbildung 12: Name installierte Anwendung (Eigene Darstellung)

description: Dieser Parameter bietet eine allgemeine Beschreibung der Progressive Web-Anwendung.

icons: Hier kann eine Liste von Icons unterschiedlicher Größe bereitgestellt werden, welche dann je nach Anwendungszwecks als Anwendungsicon verwendet werden können (siehe Abbildung 12). Im vorliegenden Quellcode wurden zwei Icons mit unterschiedlichen Größen bereitgestellt.

Dies kann man der Abbildung 12 entnehmen, welche einen Auszug aus den Chrome Dev Tools darstellt.

Abbildung 13: Icon Chrome Dev Tools (Eigene Darstellung)

start_url: Dieser Parameter ist die Start-URL der Anwendung. Die Beispiel-Anwendung wurde auf Github Pages veröffentlicht, aus diesem Grund wurde diese start_url verwendet.

display: Hiermit kann der Standardanzeigemodus der Webanwendung angegeben werden. Dazu zählen:

- Vollbild
- Standalone
- Minimal-UI
- Browser

orientation: Die „orientation“ definiert die Standardausrichtung für die Webanwendung, welche entweder Hochformat oder Querformat ist.

theme_color: Dieser Parameter ist die Standarddesignfarbe für die Anwendung. Auf Android wird diese auch verwendet, um die Statusleiste einzufärben.

background_color: Hiermit wird die Hintergrundfarbe der Progressivee Web-Anwendung angegeben. In Chrome definiert es auch die Hintergrundfarbe des Begrüßungsbildschirms.

dir: Mit “dir” ist es möglich die Leserichtung zu bestimmen, ob von Links nach Rechts oder von Rechts nach Links gelesen werden soll.

related_applications: Dies wird verwendet, um native Anwendungsalternativen in den verschiedenen App Stores anzugeben.

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <meta name="theme-color" content="#009578">
        <title>PWA</title>
        <link rel="stylesheet" href="src/master.css">
        <link rel="manifest" href="manifest.json">
        <link rel="apple-touch-icon" href="images/logo192.png">
    </head>
    <body style="height: 100vh;">
        <div style="display: flex; flex-direction: column; height: 100%; justify-content: center; align-items: center; font-size: 30px;">
            Hello World of FH Campus 02
            <img style="margin-top: 10px;"
                 src="https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSHZEQagtR9bFw7xb-4IUwgbESZ2p2iVrwgc0repxyZY5AkPh_PtXScRDcQPsBX8tWu0Do&usqp=CAU"/>
        </div>
        <script src="src/index.js"></script>
    </body>
</html>
```

Abbildung 14: App Manifest im Quelltext hinzufügen (Eigene Darstellung)

Nachdem das App Manifest mit dem gewünschten Parameter konfiguriert wurde, muss die Datei nur mehr im Quellcode, wie in Abbildung 14 darstellt, eingefügt werden (Mozilla 2021b).

2.5.5 Unterstützung der Browser- und Betriebssysteme

Wie bereits erwähnt, hängen Progressive Web-Anwendungen nicht von einer einzelnen API ab, sondern verwenden verschiedene Schnittstellen, um alle Anforderungen abzudecken. Jedoch gibt es verschiedene Browseranbieterinnen und -anbieter, welche unterschiedliche Zugriffsfreiheiten bieten. Nichtsdestotrotz ist die wichtigste Voraussetzung die Unterstützung des Service-Workers. Mittlerweile werden Servicemitarbeiter aber bereits von allen gängigen Browsern auf Desktops und Mobilgeräten unterstützt (Mozilla 2021a). Bei weiteren Schnittstellen, Software- bzw. Hardwarezugriffen kann es jedoch Abhängigkeiten des gewählten Browsers geben. Dies und weitere Faktoren werden in der Evaluierungsphase dieser Arbeit näher betrachtet.

2.5.6 Anwendungsbeispiele

Immer mehr namhafte Unternehmen nutzen Progressive Web-Anwendungen, um ihre Anwendungen zu realisieren. PWA.BAR bietet hier einen Überblick über bekannte Beispiele, einige davon werden im Anschluss davon betrachtet (PWA.BAR 2021):

Starbucks: Mit dem Ziel, allen Kundinnen und Kunden eine zugängliche, benutzerfreundliche Online-Bestellung zu ermöglichen, hat Starbucks eine Bestellsystem-Anwendung erstellt, die eine ähnliche User Experience wie ihre vorhandene native Anwendung bietet. Mit anderen Worten, mit seiner Fähigkeit, im Offline-Modus zu laufen, ermöglicht Starbucks mithilfe von Progressive Web-Anwendung seinen Kundinnen und Kunden, das Menü zu durchsuchen, ihre Bestellungen anzupassen und Artikel in ihre Warenkörbe zu legen – alles ohne durchgehenden Zugang zum Internet. Online können sie ortsspezifische Preise einsehen und ihre Speise- und Getränkebestellung aufgeben. Da die meisten Progressive Web-Anwendungen ohne Netzwerkverbindung verfügbar sind, eignen sie sich hervorragend für Menschen, die unterwegs

sind und bei denen sich die Konnektivität den ganzen Tag über ein- und ausschaltet, oder zum Beispiel auch für ländliche Gemeinden, in denen die Verbindung weniger zuverlässig ist. Mit der Einführung der neuen Bestell-Anwendung hat Starbucks bereits bedeutende Ergebnisse erzielt. Die Progressive Web-Anwendung ist 99,84% kleiner als die bestehende iOS-App von Starbucks, was die Web-App zu einem Favoriten unter ihren Benutzerinnen und Benutzern macht. Infolgedessen verdoppelten sie die Zahl der Web-Benutzerinnen und Benutzer, die täglich Bestellungen aufgeben, wobei Desktop-Benutzerinnen und Benutzer ungefähr dieselbe Rate wie mobile bestellen.

Uber: Da das Unternehmen in neue Märkte expandiert, wurde die Webseite von Grund auf als Progressive Web-Anwendung aufgebaut, um ein vergleichbares Buchungserlebnis wie die native mobile Anwendung zu bieten. Die Uber-Software wurde entwickelt, um die Autobuchung in 2G-Netzen mit niedriger Geschwindigkeit möglich zu machen. Die Progressive Web-Anwendung basiert auf dem Konzept einer app-ähnlichen Erfahrung, auf die in allen modernen Browsern zugegriffen werden kann, und eignet sich hervorragend für Fahrerinnen und Fahrer auf älteren Geräten, die möglicherweise nicht mit der nativen Uber-App kompatibel sind. Durch die Integration der nativen Erfahrung in eine Web-Anwendung hat Uber eine schnelle Fahrtanfrage unabhängig von Standort, Netzwerkgeschwindigkeit und Gerät ermöglicht. Die Kern-App von nur 50kB ermöglicht es, innerhalb von 3 Sekunden in 2G-Netzwerken zu laden.

Pinterest: Mit Fokus auf internationales Wachstum startete Pinterest sein neues mobiles Web-Erlebnis von Grund auf als Progressive Web-Anwendung. Das soziale Netzwerk stellte fest, dass nur 1% seiner mobilen Benutzerinnen und Benutzer aufgrund der schlechten Leistung auf dem Handy zu Anmeldungen oder zu App-Installationen konvertierten. Als sie erkannten, dass die Möglichkeit zur Verbesserung der Konversion riesig war, bauten sie die mobile Webseite mit der Progressiv-Web-Anwendung-Technologie neu auf, was zu mehreren positiven Ergebnissen führte: Die aufgewendete Zeit ist im Vergleich zur vorherigen mobilen Webseite um 40% gestiegen bzw. stiegen die Einnahmen aus nutzergenerierten Anzeigen um 44%.

Spotify: Aufgrund einiger Meinungsverschiedenheiten zwischen Spotify und Apple bezüglich der 30%-igen App-Store-Provision von Apple fand Spotify es eine passende Gelegenheit, mit der Entwicklung einer Progressive-Web-Anwendung-Version ihrer App zu beginnen. Im Vergleich zu ihrem nativen App-Pendant ist die neue Version mit ihrer eigenen adaptiven Benutzeroberfläche, die ihren Hintergrund ändert, wenn der Benutzer durch die App voranschreitet, erheblich schneller. Wie bei vielen anderen Progressive Web-Anwendungen werden Benutzerinnen und Benutzer auch aufgefordert, die Softwarelösung zu ihrem Startbildschirm hinzuzufügen, wodurch die Anwendung zugänglicher und mit seinen anderen Versionen vergleichbar ist.

3 KRITERIENDEFINITION

Im Anschluss werden die Kriterien, welche die Technologien evaluieren, aufgestellt und in ihrer Bedeutsamkeit erläutert. Um die zwei Technologieansätze der nativen Anwendunden und Progressive Web-Anwendungen anhand eines Qualitätsstandard zu vergleichen, wurde die ISO 25000 Software engineering – Software product Quality Requirements and Evaluation (SQuaRE) verwendet, genauer betrachtet wurde dabei die ISO 25010 der Normenfamilie ISO 250xx. Ziel ist es, dem „System und Software-Qualitätsmodell“ und somit aller Qualitätskriterien dieser Norm gerecht zu werden. Diese wurden, wie bereits erwähnt, als Grundlage herangezogen, da sie für die Qualitätssicherung von Software, IT-Systemen und Software-Engineering stehen (ISO/IEC 25010:2011).

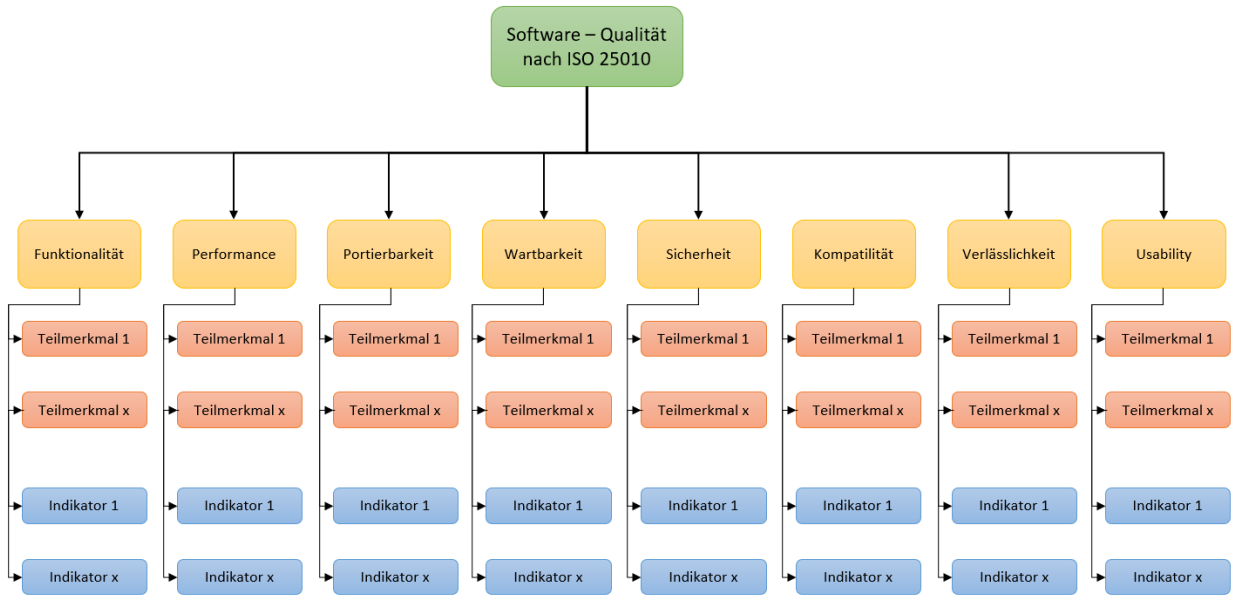

Abbildung 15: Qualitätsmodell Struktur (Eigene Darstellung)

Die Qualitätskriterien werden in unterschiedliche Abstraktionsebenen nach dem FCM-Modell von Böhm et. al., wie in Abbildung 15 dargestellt, eingeteilt. Die Abkürzung FCM kommt dabei aus dem Englischen und steht für „factors“, „criterions“ und „metrics“.

Die oberste Ebene bilden die Qualitätsmerkmale nach der ISO 25010. Diese sind Funktionalität, Performance, Wartbarkeit, Sicherheit, Kompatibilität, Verlässlichkeit, Usability und Portierbarkeit, welche im FCM-Modell als „factors“ bezeichnet werden.

Um daraufhin die Qualitätsmerkmale in separierbare und vergleichbare Charakteristiken zu teilen, werden diese in Qualitätsteilmerkmale aufgeteilt. Diese Teilmerkmale werden seitens des FCM-Modells als „criterions“ bezeichnet.

Am Ende werden diese dann mit Qualitätsindikatoren versehen, um die Merkmale mit einer messbaren Variablen, als „metrics“ bezeichnet, zu versehen. Natürlich bedarf es je nach Erfüllung der Anforderungen der Qualitätsteilmerkmale an unterschiedlichen Indikatoren. Dies reichen von metrischen Vergleichswerten für den Erfüllungsgrad bis hin zu kategorischen Metriken, die

definieren, ob die Anforderungen erfüllt sind oder nicht (B. W. Boehm, J. R. Brown, M. Lipow 1978).

Um über diese Qualitätsmerkmale nun eine fundierte Wissensbasis zu schaffen, werden diese im Anschluss einzeln ausgearbeitet.

3.1 Funktionalität

Im ersten Schritt wird das ISO 25010 Qualitätsmerkmal der Funktionalität betrachtet und die dazugehörigen Software-Features aufstellt. Primär werden hier für die Teilmerkmale die Zugriffe verschiedenster Hardware- und Softwarekomponenten betrachtet. Bevor diese im Kapitel 4 der Evaluierung unterzogen bzw. deren Einschränkungen verdeutlicht werden. Ziel ist es, die entscheidendsten Softwarebestandteile zu identifizieren, um die möglichen funktionalen Anforderungen bestmöglich aufzugreifen, da die resultierende Entscheidungsmatrix für diverse Softwareprojekte anwendbar sein soll. Im Anschluss werden diese hinsichtlich ihrer Relevanz und ihrer technologischen Prinzipien erklärt (ISO/IEC 25010:2011).

Push-Benachrichtigung: Push-Benachrichtigungen werden dazu genutzt, um Nachrichten bei Endgeräten sichtbar zu machen, selbst wenn die Benutzerin oder der Benutzer die Anwendung gerade nicht aktiv nutzt. Mit diesem Feature ist es folglich möglich, Informationen am Endgerät zu einer gewünschten Zeit mit administrierbarem Inhalt anzuzeigen. Push-Benachrichtigungen folgen dabei dem „publish-subscribe“ Nachrichten Pattern. Dieser Ansatz besteht prinzipiell aus drei wesentlichen Bestandteilen, welche für die Realisierung notwendig sind. Zum einen ist das ein Server, welcher Push-Nachrichten generiert und somit im Englischen als „publisher“ bezeichnet wird. Zum anderen gibt es einen Push-Notification-Service, welcher die Vermittlung an die drei Komponente schickt. Das mobile Endgerät registriert sich dabei beim Push-Noitification-Service, welcher dem „subscriber“ des Nachrichten Pattern entspricht. Wenn der zuständige Server eine Nachricht erstellt, wird diese an die registrierten Endgeräte verteilt (Chen et al. 2018; Warren et al. 2014).

Bluetooth: Darüber hinaus könnte der Datenübertragungsstandard Bluetooth eine wichtige funktionale Anforderung für mobile Anwendungen sein. Deswegen kann es eine Anforderung sein, Daten über kurze Distanzen (bis ca. 100 Meter) zu erhalten. Bluetooth wird seit den 1990ern als Industriestandard bei der Datenübertragung über kurzen Distanzen verwendet, welcher von der Bluetooth Special Interest Group (SIG) initiiert wurde (Bisdikian 2001). Grundsätzlich muss zwischen zwei Technologien unterschieden werden. Zum einen gibt es das klassische Bluetooth und zum andere das sogenannte „Bluetooth Low Energy“ (=BLE), welches ab der Version 4.0 als weiterer Industriestandard eingeführt wurde. Bluetooth und Bluetooth Low Energy werden prinzipiell für unterschiedliche Zwecke eingesetzt. Der klassische Übertragungsstandard Bluetooth steht für die Übermittlung von größeren Datenmengen, wie bei kabellosen Headsets oder bei der Dateien-Übertragung zwischen Endgeräten. BLE wird für Anwendungen verwendet, die nur kleine Datenmengen austauschen müssen und daher mit geringen Batteriestrom betrieben werden können, wie bei gesundheitsüberwachenden oder IOT Sensoren (Gomez et al. 2012). Somit können beide Standards relevant bei mobilen Anwendungsentwicklungen sein. Für

die Übertragung wird dabei der Frequenzbereich von 2,402 – 2,480 GHz genutzt. Dieser Bereich wurde gewählt, um Frequenzkollisionen zu vermeiden. Dabei wird das Frequenz-Hopping angewandt, wobei ca. 1600-mal pro Sekunde der Sender und der Empfänger in einen Kanal wechseln. Bei der klassischen Bluetooth Technologie gibt es hierfür 79 Kanäle und bei Bluetooth Low Energy 40. Dieser Datenübertragungsstandard wird daher als sehr stabil angesehen (Golmie et al. 2003).

NFC (Near Field Communication): In vielen Fällen kann es eine Anforderung sein, Daten nur über sehr kurze Distanzen (von ein Centimeter bis vier Centimeter) zu übertragen. Das kann mittels NFC geschehen. Der wesentliche Vorteil gegenüber Bluetooth ist, dass es schwieriger ist, eine willkürliche Verbindung mit einem nicht gewollten Gerät herzustellen. Aus diesem Grund ist es bei diesem Technologieansatz nicht notwendig, Authentifizierungsschritte einzuleiten. Die Anwendungsgebiete sind sehr umfangreich, angefangen vom mobilen kontaktlosen Bezahlen bis hin zum Austausch von Kontaktinformationen und vieles mehr. Die Kommunikation über NFC – Transmitter geschieht grundsätzlich über die Frequenz 13,56 kHz, welche im Gegenzug zur Bluetooth-Technologie einen fixen Frequenzbereich hat (Nishihara et al. 2018). Zur Datenübertragung stehen die zwei Protokolle NDEF (NFC Data Exchange Format) und SNEP (Simple NFC Exchange Format) zur Verfügung. In den Anfangsjahren gab es nur das NDEF Protokoll, welches nur den Lese- und Schreibmodus von NFC Tags ermöglichte. Das SNEP Protokoll bietet eine Erweiterung der vorhandenen Lösung. Damit können auch Daten nahtlos im Peer-to-Peer Modus übertragen werden (Wolfgang W. Osterhage 2018).

Vibration: Um bei definierten Interaktionen oder Ereignissen die Endbenutzerin bzw. den Endbenutzer mit einer Geste auf eine Statusänderung aufmerksam zu machen, ist es oftmals ein nützliches Hilfsmittel am Endgerät eine Vibration hervorzurufen. Aus diesem Grund wird dieses Feature als Teil der Funktionalitäten nach ISO 25010 in dieser Arbeit aufgenommen.

Kamera und Mikrofon: Bei der Planung eines Softwareentwicklungsprojektes wird man in manchen Fällen mit der Anforderung konfrontiert werden, dass Fotos, Videos bzw. Tonaufnahmen in der Laufzeit vom Endgerät verwendet werden sollen. Neben dem Hochladen aus dem Gerätespeicher, kann es in manchen Fällen zur Problemstellung kommen, dass der direkte Zugriff auf Kamera bzw. Mikrofon gewährt werden soll.

Dateien Zugriff: In vielen Fällen ist es eine funktionale Anforderung, eine Datei vom mobilen Endgerät bereitzustellen. Aus diesem Grund wird der Zugriff auf den lokalen Speicher in der Evaluierungsphase untersucht, inwiefern diese bei den Technologieansätzen erlaubt bzw. eingeschränkt sind.

Geolocation: In den Fällen, dass der aktuelle GPS-Standort in der Anwendung genutzt werden soll, ist es notwendig, diese per zur Verfügung stehenden Schnittstellen zu erlangen. Die Verwendung des Standortes beruht dabei auf Längen- und Breitegrade, welche der Softwarelösung bereitgestellt werden sollen.

Geofencing: Ist man daran interessiert, ob die Endbenutzerin bzw. der Endbenutzer gerade einen geologischen Bereich betreten oder verlassen hat, ist es notwendig, eine sogenannten „Geofence“ zu definieren. Dieser Technologieansatz versucht eine Beziehung zwischen der Position des mobilen Endgeräts und einem vordefinierten Ort zu erstellen. Somit ist es möglich,

bei der Übertretung der virtuellen „Geofence“-Grenzen eine Aktion auszuführen. Prinzipiell ist ein „Geofence“ ein virtueller, selbst definierter Radiu, um ein physisches Objekt oder einem Standort (Reclus und Drouard 2009).

Device Motion: Beim Qualitätsteilmerkmal der „Device Motion“ werden die Möglichkeiten der Beobachtung der Geräte-Bewegungen betrachtet. Um diese Bewegung messbar zu machen, gibt es drei wesentliche Sensoren, welche in einem mobilen Gerät verbaut sind (Debnath Bhattacharyya et al. 2009):

- **Accelerometer (Beschleunigungssensor):** Ein Accelerometer ist ein Sensor, der die Beschleunigung messen kann. Dieser Sensor erzeugt somit Beschleunigungswerte der Achsen X, Y und Z, welche mit der SI-Einheit m/s² gemessen werden (MDN Web Docs 2021c; Olli Särkkä et al. 2021).
- **Gyroscope (Geschwindigkeitssensor):** Der Gyroscope-Sensor wird im Gegenzug dazu verwendet, um Geschwindigkeitsinformationen der drei Achsen des mobilen Endgerätes zu erhalten. Ziel ist es dabei Messungen in der SI-Einheit (MDN Web Docs 2021h; Olli Särkkä et al. 2021).
- **Magnetometer (Magnetfeldsensor):** Um das vorherrschende Magnetfeld mithilfe eines mobilen Endgerätes zu messen, muss eine Magnetometer-Sensor verbaut sein. Dies kann beispielsweise für einen Kompass genutzt werden. Die Magnetfeld-Informationen werden dabei in der SI-Einheit Tesla erwartet (MDN Web Docs 2021n; Olli Särkkä et al. 2021).

Fingerabdruck/Gesichtserkennung: Um die Anmeldungen ohne Eingabe eines Passwortes zu ermöglichen wie auch eine Multi-Faktor Authentifikation zu gewährleisten, ist es notwendig, Zugriffe auf die biometrischen Sensoren des mobilen Endgeräts zu erhalten. Bei biometrischen Authentifizierungen wird ein physisches Körperteil verwendet, um die Identität zu gewährleisten. Die Basis kann ein Fingerabdruck, ein Gesicht oder andere physische Merkmale sein. Die Hardware für Fingerabdrucksensoren und die Funktionalitäten hinter einer Gesichtserkennung sind mittlerweile ein fixer Bestandteil von neueren mobilen Endgeräten. Nichtsdestotrotz muss der Zugriff aus einer Anwendungslogik heraus erlaubt sein. In welchem Ausmaß dies erlaubt ist, wird in der Evaluierungsphase im Detail betrachtet (Debnath Bhattacharyya et al. 2009).

3.2 Performance

Des Weiteren verfolgt die ISO 25010 das Qualitätsmerkmal der Performance, welche sich das Ziel setzt, eine Software zu entwickeln, die stets ein gutes Zeitverhalten hat, Ressourcen effektiv nutzt und die Kapazitäten so gut wie möglich schont (ISO/IEC 25010:2011). Um dieses angestrebte Ziel zu erreichen, werden die Technologieansätze der Progressive Web-Anwendungen und nativen Anwendungen in vier Qualitätsteilmerkmalen untersucht. Anzumerken ist jedoch, dass aufgrund der vielen Abhängigkeiten, welche die Performance beeinflussen können, nur sehr wenige Studien vorliegen. Der Vergleich ist somit nur sehr oberflächlich zu betrachten, da die Performance stark von den benutzten Geräten, der Internetverbindung, der

Zugriffsfreiheiten und weiteren Faktoren abhängig ist. Aus diesem Grund ist es schwierig eine global-gültige Aussage zu treffen. Jedoch können aufgrund der vorliegenden Literatur spezifische Anwendungsszenarien genauer betrachtet werden, welche es ermöglich können, Aussagen für allgemeine Softwareprojekte zu treffen.

Gleichbleibender Energieverbrauch durch Service-Worker: Da die Akkulaufzeit eine begrenzte Ressource bei mobilen Endgeräten darstellt, stellt der Energieverbrauch ein Vergleichskriterium dar. Sollte jedoch bereits eine Web-Anwendung bestehen und man hat das Ziel, daraus eine Progressive Web-Anwendung zu transformieren, möchte man die Performance der bestehenden Lösung nicht verschlechtern. Wie bereits erwähnt, ist der Service-Worker eine der essenziellen Bestandteile von Progressive Web-Anwendungen. Somit wird es auch Teil der Arbeit sein, den Energieverbrauch mit bzw. ohne eines Service-Workers zu vergleichen.

Energieverbrauch bei Interaktionen mit UI-Elementen im Bereich 2 ± 2 Joule: Hier werden unterschiedliche Interaktionen der Systeme evaluiert, um im Anschluss eine Bewertungsgrundlage zu erhalten. Die Bereiche des Energieverbrauchs wurde von 0 – 4 Joule auf Basis einer Studie von Huber et. al. gewählt. Dabei konnten verschiedenste UI-Interaktionen bezüglich des Energieverbrauchs gemessen werden. Für diese Studie wurden dabei mehrere Technologieansätze bezüglich ihrer Energiebilanz betrachtet. Als Grundlage wird der durchschnittliche Energieverbrauch von nativen Lösungen gewählt, da der Fokus dieser Arbeit auf der Gegenüberstellung der Progressive Web-Anwendungen und der nativen Anwendungen liegt. Dieser ist in dem genannten Forschungssetup von 0 – 4 Joule (Huber et al. 2021).

Reaktionszeit Kamera-/Standortzugriff unter 1 Sekunde: Um die Endbenutzerinnen und -benutzer eine Benutzeroberfläche zu bieten, welche stets flüssig wirkt, wird in dieser Arbeit die Reaktionszeiten bei Hardwarezugriffen untersucht. Der Umfang der Evaluierung wurde aufgrund der mangelnden Literatur dabei auf die Kamera- und Standortzugriffe beschränkt. Jakob Nielsen (1993) hat dabei in seiner Publikation „Usabilty Engineering“ die Reaktionszeiten in Stufen eingeteilt:

- < 0,1 Sekunden: Die erste Stufe sind Reaktionszeiten eine Produktes bis 0,1 Sekunden, wobei die Nutzerinnen und Nutzer das Gefühl haben, dass Änderungen unverzüglich passieren.
- < 1 Sekunde: Die zweite Stufe von einer Reaktionszeit kleiner 1 Sekunden ist die Schwelle für den Gedankenfluss der Nutzerinnen und Nutzer. Interaktionen wirken ununterbrochen, obwohl ein Ladevorgang erkannt wird.
- <10 Sekunden: Die dritte Stufe ist dabei die Schwelle, um die Aufmerksamkeit einer Benutzerin oder eines Benutzers aufrecht zu erhalten. Dies ist bis zu 10 Sekunden möglich, jedoch wünscht sich der Benutzende eine Ladeanzeige oder die Möglichkeit, andere Aktionen in der Zwischenzeit auszuführen.

Da ein Informationsfluss bis zu 1 Sekunde als ununterbrochen für Nutzerinnen und Nutzer wirkt, wird dies als Schwelle für die Untersuchung der Kamera- und Standortzugriffe gewählt.

3.3 Portierbarkeit

Die Portierbarkeit nach ISO 25010 beschäftigt sich mit dem Qualitätsziel, eine Anwendung ausführbar bzw. installierbar für mehrere Plattformen zu machen. Des Weiteren beschäftigt sich dieses Qualitätsmerkmal mit den Grad der Effektivität und Effizienz, mit der eine Softwarelösung installierbar gemacht werden kann (ISO/IEC 25010:2011).

Aus diesem Grund werden fünf Qualitätsteilmerkmale für die Evaluierung der Entwicklungsansätze aufgestellt. Diese werden im Anschluss einzeln erläutert.

Verfügbarkeit auf Google Play Store: Zunächst wird die Möglichkeit zur Veröffentlichung von verschiedenen Entwicklungsansätzen auf den Google Play Store betrachtet. Neben den Möglichkeiten und Einschränkungen wird auch der Prozess der Veröffentlichung analysiert.

Verfügbarkeit auf Apple App Store: Im Anschluss wird die Verfügbarkeit auf dem Apple App Store betrachtet. Der Kontext der Evaluierung wird derselbe sein wie beim Google Play Store, jedoch mit dem Unterschied, dass die Veröffentlichung den Fokus auf den Apple Store legt.

Verfügbarkeit auf Webseiten: Darüber hinaus soll die Veröffentlichung der Software auf einer Webseite betrachtet werden. Es stellt sich dabei die Frage, ob es möglich ist, eine installierbare Datei über einen Web-Auftritt zu veröffentlichen. Ziel ist es, die Anwendung zum Download anzubieten und am Endgerät zu installieren, mit der Voraussetzung, dass diese auf der jeweiligen Plattform zulässig ist.

Aufruf über URL: Des Weiteren kann es eine Anforderung sein, eine Anwendung ohne Installation über eine URL aufzurufen. Somit kann gewährleistet werden, dass die Anwendung keinen vorgeschriebenen Installationseinschränkungen folgen muss, sondern direkt in einem Browser aufgerufen werden kann.

3.4 Wartbarbeit

Im Anschluss wird das ISO 25010 Qualitätsmerkmal der Wartbarkeit betrachtet. Ziel ist es, dabei eine Softwarelösung mit einem geringen Grad an Aufwand zu haben, das Produkt bzw. das System zu verbessern, zu korrigieren oder an neue Anforderungen und Umgebungsbedingungen anzupassen (ISO/IEC 25010:2011). Um den Grad der Erfüllung dieses Ziels zu evaluieren, wurden sechs Qualitätsteilmerkmale aufgestellt, welche im Anschluss betrachtet werden.

Automatische Updates: Bei der Evaluierung dieses Qualitätsteilmerkmals werden die Möglichkeiten und Einschränkungen von automatischen Updates bei neuen Softwareversionen analysiert. Liegt eine neue Version der Anwendung vor, ist die Entwicklerin oder der Entwickler stets daran interessiert, dass die Software am Endgerät den aktuellen Stand vorweist. Aus diesem Grund gilt es zu evaluieren, inwiefern es möglich ist, Updates automatisch zu aktualisieren bzw. ob es notwendig ist, die Nutzerinnen und Nutzer mit einer eigenen Logik über Updates zu informieren.

Sicherheitsupdates: Bei diesem Qualitätsteilmerkmal wird die Frage beantwortet, ob es seitens der Software möglich ist, Sicherheitsupdates ohne Zustimmung der Nutzerinnen und Nutzer zu

veröffentlichen, um die Sicherheit zu jedem Zeitpunkt zu gewährleisten. Sollte dies nicht gestattet sein, wird darüber hinaus untersucht, ob es möglich ist die Nutzung der Software unzulässig zu machen, bis die Nutzerinnen und Nutzer das Update installiert haben.

Update von der Endbenutzerin bzw. dem Endbenutzer steuerbar: Dieses Qualitätsziel befasst sich mit der Frage, ob es möglich ist, Updates seitens der Nutzerinnen oder Nutzer zu steuern. Dies ist vor allem dann von Interesse, wenn ein relevantes Update vorliegt und die Endbenutzerin bzw. der Endbenutzer das Update verweigern will oder kann. Somit gilt es zu untersuchen, ob es auf der Seite der mobilen Endgeräte auch möglich ist, Updates nach eigenem Interesse zu verwalten. Dies ist vor allem für Entwicklerinnen und Entwickler von Relevanz, weil es somit möglich wäre, dass Nutzerinnen oder Nutzer eine nicht mehr unterstütze Version installiert haben.

Updates in Phasen veröffentlichen: Sollen Updates, bevor sie vollständig freigegeben werden, einer minimierten Anzahl von Nutzerinnen und Nutzer zur Verfügung gestellt werden, um eine neue Version zu testen, kann es eine Anforderung sein, die Updates in Phasen zu veröffentlichen. Bekommt somit nur ein eingeschränkter Kreis eine neue Version, können fehlerhafte Releases auf diese Personen eingeschränkt werden. Die Anzahl der Endgeräte, welche das Update erhalten, soll dann kontinuierlich erhöht werden, bis alle Benutzerinnen und Benutzer das Update erhalten haben.

Bewertungen und Rezension: Um den Bedürfnissen der Nutzerinnen und Nutzer gerecht zu werden, ist es notwendig, eine Möglichkeit zu schaffen, diese Anforderungen zu verwalten. Es kann daher ein endscheidender Erfolgsfaktor gegenüber anderen Anbieterinnen und Anbieter sein, wenn man seine Softwarelösung ständig an die Nutzerinnen und Nutzer anpasst. Dabei gilt es zu evaluieren, ob der Technologieansatz hierfür bereits Services bzw. Schnittstellen anbieten.

App-Performance Metriken: Da in den meisten Fällen die Entwicklerin bzw. der Entwickler daran interessiert ist, Umsätze zu steigern und/oder die Bedürfnisse ihres bzw. seines Kundenstamms zu befriedigen, ist es notwendig, den Erfolg der Anwendung mit Metriken zu versehen. Hier kann es relevant sein, Metriken über Installationen, Umsätzen, Abstürzen etc. zu erhalten. Aus diesem Grund sollen in der Evaluierungsphase die verschiedenen Ansätze untersucht werden, inwiefern es bereits inkludierte Services oder programmatische Möglichkeiten gibt.

3.5 Sicherheit

Des Weiteren wird das Qualitätsmerkmal der Sicherheit anhand von ISO 25010 betrachtet (ISO/IEC 25010:2011). Ein nicht mehr weg zu denkender Faktor in allen Softwareentwicklungsprojekten ist die Sicherstellung der Sicherheit gegenüber böswilligen Handlungen. Dabei können nicht nur personenbezogene Daten verloren gehen, sondern auch ganze IT-Infrastrukturen lahmgelegt werden. Darüber hinaus sind damit in den meisten Fällen auch hohe Kosten verbunden. Somit ist es ein endscheidender Erfolgsfaktor, das Softwareprodukt so sicher wie möglich zu gestaltet. Um diese Ziele zu gewährleisten, werden Progressive Web-Anwendungen und native Anwendungen in Bezug auf Sicherheit

gegenübergestellt. Als Vergleichsgrundlage wurden zwei sicherheitsrelevante Qualitätsteilmerkmale einer Anwendung aufgestellt:

Sicherheitsprüfung der Installationsdatei: Zum einen wird es Teil der Evaluierungsphase sein, die Installationsdatei auf ihre Sicherheit zu überprüfen. Eines der Hauptaugenmerke wird es sein festzustellen, ob es weitere Instanzen gibt, welche die Sicherheit der Anwendung überprüfen. Des Weiteren werden die potenziellen Schadsoftware-Quellen der verschiedenen Technologieansätze belichtet. Ferner wird evaluiert, ob es bereits Sicherheitsanforderungen gibt, welche eingehalten werden müssen, um überhaupt eine solche Anwendung veröffentlichen zu können.

Multi-Faktor Authentifizierung: Neben der Sicherheitsprüfung der Installationsdatei ist es ein Ziel dieser Arbeit, das Qualitätsteilmerkmal der Multi-Faktor Authentifizierung zu analysieren. Im Anschluss wird kurz die Relevanz von diesem Sicherheitskonzept betrachtet. Dabei ist es notwendig, mindestens zwei Beweisstücke aus unterschiedlichen Kategorien bereit zu stellen, um die digitalen Benutzerinnen- und Benutzer-Identität sicher zu stellen. Aus diesem Grund ist es für Hackerinnen und Hacker schwieriger, an eine gefälschte Identität zu kommen, da es mehr wie ein Beweisstück erfordert. Im Wesentlichen gibt es drei Kategorien, wovon zwei Kategorien erfüllt sein müssen. Die erste Kategorie ist das „Wissen“, welches ein PIN, Passwort etc. sein kann. Darüber hinaus gibt es die Kategorie „Eigenschaft“. Dieser Authentifizierungsfaktor steht eng im Einklang mit der sicherheitsrelevanten Hardware einer Anwendung. Er bezieht sich auf die biometrischen Merkmale einer Person. Die dritte und letzte Kategorie ist der „Besitz“. Beispiele dafür sind SmartCards und zugesandte Einmalpasswörter. Der Realisierungsgrad dieses Qualitätsteilmerkmales soll in der Evaluierungsphase näher betrachtet. Der Fokus wird dabei auf die Kategorie der Eigenschaften gelegt. Somit gilt zu untersuchen, ob biometrische Daten mit dem Endgerät auslesbar sind (Stephan Spitz 2021).

3.6 Kompatibilität

Daraufhin wird die Kompatibilität als Qualitätsmerkmal nach ISO 25010 näher betrachtet. Ziel dieses Merkmals ist es, eine Software zu entwickeln, welche es möglich macht, unterschiedliche Endgeräte wie auch Plattformen aufeinander bestmöglich abzustimmen (ISO/IEC 25010:2011).

Plattformunabhängigkeit: Das erste Qualitätsteilmerkmal ist die Plattformunabhängigkeit. Plattformunabhängigkeit bei Software bedeutet, dass eine Codebasis betriebssystemübergreifend auf mehrere Plattformen lauffähig ist und somit die Kompatibilität zu dem jeweiligen Endgerät mit der verwendeten Hardware und Software sicherstellt. Ziele der Evaluierungsphase wird es sein, herauszufinden, inwiefern es möglich ist, mit Progressive Web-Anwendungen und nativen Anwendungen eine Software für mehrere Plattformen zu entwickeln. Somit soll es möglich sein, mithilfe einer Quellcode-Basis die Kompatibilität zu den Endgeräten der potenziellen Nutzerinnen und Nutzer herzustellen (Hubert 2002).

Sicherstellung der Auf- und Abwärtskompatibilität: Wenn eine neue Software-Version veröffentlicht wird, welche bereits auf neu-implementierte Features der jeweiligen Technologieansätze setzt, ist es ein nicht wegzudenken Faktor, die installierten Versionen der

Nutzerinnen/Nutzer-Endgeräte zu berücksichtigen. Hier kann das Problem auftauchen, dass der neue Software-Release bereits Features nutzt, welche eine veraltete Geräte-Version nicht mehr in der Lage ist zu verwenden. Da in den meisten Fällen bei neuen Releases weitere Plattform-Funktionen implementiert und vorhandene Schnittstellen ersetzt oder entfernt werden, soll es trotzdem möglich sein, Features von älteren Geräte-Versionen zu nutzen. Somit kann es notwendig sein, eine Mindest-Version eines Betriebssystems bzw. Browsers festzulegen, um Inkompatibilitäten zu umgehen. Dabei spricht man von der Abwärtskompatibilität. Diese minimale Version stellt sicher, dass nur Endgeräte die Software benutzen können, welche mindestens diese Version aufweisen. Die Abwärtskompatibilität würde somit sichergestellt sein, wenn zum Beispiel ein von einem neuen Release erstelltes Dokument auch mit einer älteren Version geöffnet werden kann. Daneben gibt es auch die Aufwärtskompatibilität. Die Aufwärtskompatibilität kann sichergestellt werden, wenn man ein Endgerät mit einer älteren Version besitzt, aber trotzdem die Funktionen des neuen Anwendungs-Releases nutzen kann. In der Softwareentwicklung gestaltet sich die Gewährleistung der Aufwärtskompatibilität etwas schwieriger als die Abwärtskompatibilität, da bei der Erstellung einer neuen Version noch nicht bekannt ist, welche Features und Formate in Zukunft umgesetzt werden könnten. Diese Art von Kompatibilität ist sichergestellt, wenn man zum Beispiel mit einer alten Softwareversion ein Dokument erstellt und trotzdem mit einer neuen Version verwenden kann (Sangani 2011; Michael Kende 1994).

Da diese zwei Parameter Fehlerursachen minimieren und die Software-Qualität erhöhen können, werden diese in der Evaluierungsphase näher betrachtet.

Verschiedene Versionen veröffentlichen: Darüber hinaus wird das Qualitätsteilmerkmal betrachtet, welches es möglich macht, eine Softwarelösung mit mehreren Versionen zu veröffentlichen. Wie bereits erwähnt, kann eine Anwendung mit einer Mindest- und Höchst-Version versehen werden, um die Nutzerinnen bzw. Nutzer einzuschränken. Will man diese Einschränkungen jedoch nicht hinnehmen, kann es von Interesse sein, mehrere Versionen zu veröffentlichen. Diese können dann an die Anforderungen und Möglichkeiten von der bestehenden Hardware und Software der mobilen Geräte angepasst werden. Welche Möglichkeiten es gibt und welche Schritte notwendig sind, um dieses Qualitätsziel zu erreichen, werden ebenfalls in der Evaluierungsphase geklärt.

3.7 Verlässlichkeit

Im Anschluss wird das ISO 25010 Qualitätsmerkmal der Verlässlichkeit betrachtet. Dieses beschäftigt sich im Wesentlichen mit dem Ziel, eine Software zu entwickeln, welche stets die definierten Anforderungen erfüllt. Somit soll die Softwarelösung auch bei nicht geplanten Ereignissen und Einflüssen betriebsbereit und zugänglich sein. Um dieses Qualitätsziel mit den Technologieansätzen gegenüberzustellen, werden die folgenden Qualitätsteilmerkmale betrachtet (ISO/IEC 25010:2011):

Offline-Datenhaltung: Obwohl die Welt immer vernetzer wird und immer mehr Menschen Internet-Zugang haben, gibt es eine Vielzahl an Personen, welche nur sehr langsame oder

fehlerhafte Internetverbindungen vorweisen können. Dazu wurde von Simon O'Dea auf der Online-Plattform „statista" eine Statistik veröffentlicht, welche die Breitbandabonnements-Rate im Jahr 2020 nach Regionen betrachtet. Die Abbildung 16 zeigt die erlangten Ergebnisse. Dabei konnten die Erkenntnisse gewonnen werden, dass im Jahr 2020 weltweit ca. 75% eine aktive mobile Datenverbindung verfügt haben. Europa ist dabei der Vorreiter mit 99,9%. Nichtsdestotrotz gibt es im afrikanischen, arabischen und asiatischen Bereich verhältnismäßig niedrige Raten. Gründe hierfür können die schlechte Abdeckungen des Internetanbieters, Umwelteinflüsse, abgelegene geografische Standorte und viele mehr (S. O'Dea 2021b).

Um dabei die Anwendung trotzdem nutzen zu können, ist es notwendig diese so zu konzipieren, dass die Softwarelösung auch ohne eine aktive Internetverbindung bedienbar bleibt. Aus diesem Grund soll die Fähigkeit zur Offline-Datenhaltung für die unterschiedlichen Technologieansätze evaluiert werden. Bei der Verwendung eines Offline-Speichers kann ein Konzept generiert werden, um Daten nicht bei jeder Interaktion neu abfragen zu müssen. Sollte keine oder eine mangelhafte Netzwerkverbindung vorhanden sein, wird auf den lokalen Speicher des Gerätes zurückgegriffen.

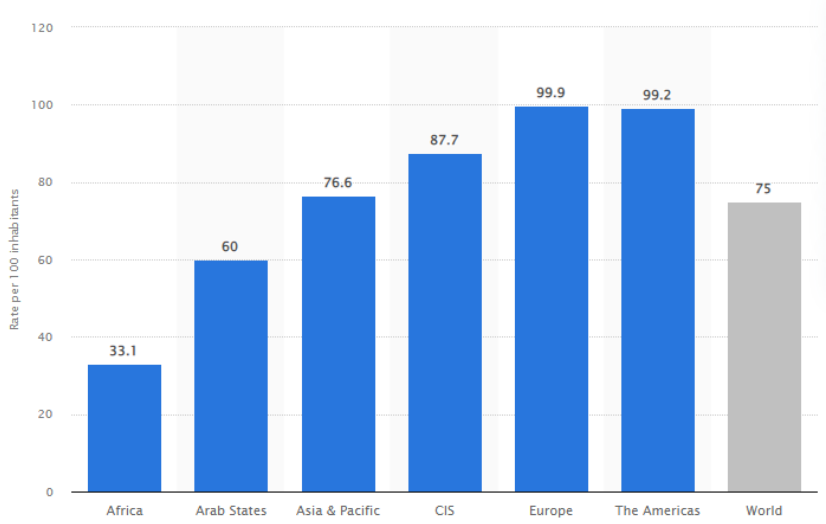

Abbildung 16: Mobile Breitband Anbindung im Jahr 2020 nach Regionen (S. O'Dea 2021b)

Hintergrund-Synchronisation: Darüber hinaus werden die Möglichkeiten betrachtet, wie Daten im Hintergrund geladen werden können, ohne die Benutzung der Anwendung einzuschränken bzw. auch wenn diese gar nicht geöffnet ist. Aus diesem Grund werden Progressive Web-Anwendungen gegenüber den nativen Varianten evaluiert und es soll festgestellt werden, ob diese Technologien die Möglichkeit bieten, Daten im Hintergrund mit einem Anwedungsserver auszutauschen. Kann dieses Feature nicht bereitgestellt werden, kann das Qualitätsziel der verlässlichen Nutzung der Anwendung nicht im vollen Umfang gewährleistet werden. Vor allem ist dies besonders relevant im Zusammenhang mit der Offline-Nutzung. Als Entwicklerin bzw. Entwickler und Benutzerin bzw. Benutzer wird man daran interessiert sein, dass Server-Anfragen im Offline-Betrieb nicht verloren gehen, sondern zwischengespeichert werden und bei aktiver Netzwerkverbindung im Hintergrund abgearbeitet werden.

3.8 Usability

Im letzten Schritt wird das Qualitätsmerkmal der Usability betrachtet. Dieser Teil der ISO 25010 verfolgt das Ziel, dass eine Anwendung stets den Bedürfnissen der Benutzerinnen und Benutzer entspricht. Des Weiteren wird mit diesem Qualitätsziel der Grad der Verfügbarkeit von Features betrachtet, welche eine einfache Bedienung und Steuerung ermöglichen. Ein zusätzlicher entscheidender Faktor ist die Ästhetik der Benutzeroberfläche, um für die Benutzerinnen und Benutzer eine zufriedenstellende Interaktion mit der Anwendung sicherzustellen. Da die Usability das Ausmaß betrachtet, in welchem der Benutzende Interaktionen durchführt, um definierte Ziele so effizient, effektiv und zufriedenstellend zu erlangen, fallen auch zuvor behandelte Qualitätsmerkmale in den Bereich von Usability. Eine zufriedenstellende Usability kann nur erreicht werden, wenn zum Beispiel die verlangten Zugrifffreiheiten erlaubt sind, die gewünschte Performance erreicht werden kann, eine Offline-Verfügbarkeit gegeben ist und vieles mehr. Diese Themen werden nicht nochmals explizit betrachtet, da diese bereits evaluiert wurden. Nichtsdestotrotz werden im Anschluss ergänzende Features und Funktionen betrachtet, welche für die Usability einer mobilen Anwendung relevant sein können (ISO/IEC 25010:2011):

Einheitlicher Look & Feel: Um Interaktionen für die Benutzerin bzw. den Benutzer so intuitiv wie möglich zu gestalten, kann es notwendig sein, eine Oberfläche zu schaffen, die den Style-Guidelines der jeweiligen Plattform entspricht. Somit soll in der Evaluierungsphase untersucht werden, inwiefern es möglich ist, eine von der Benutzerin bzw. dem Benutzer gewohnte Benutzeroberfläche zu schaffen. Darüber hinaus werden UI-Bibliotheken betrachtet, welche hier eine Hilfe bei Umsetzung sein können.

Inter-app Sharing: Des Weiteren wird das Qualitätsteilmerkmal des Inter-app Sharing betrachtet. Hier handelt es sich um den Datenaustausch innerhalb von unterschiedlichen Anwendungen, wie in Abbildung 15 dargestellt. Dies kann die Nutzung in Bezug auf die Usability deutlich verbessern, da Daten bequem zwischen Anwendungen importiert bzw. exportiert werden können und somit die Anzahl an notwendigen Interaktionsschritte deutlich verringern kann.

Abbildung 17: Inter-app Sharing IOS (Eigene Darstellung)

Schnellzugriffe per Kontextmenü: Der letzte Schritt der Evaluierungsphase bildet die Evaluierung über das Hinzufügen eines Kontext-Menüs beim Installationsicon. Damit soll die Möglichkeit geschaffen werden, häufig gebrauchte Funktionen zusammen zu fassen und ausführbar zu machen, ohne die Anwendung explizit zu öffnen. Dieses Hilfsmittel zielt besonders

auf die Effizienz und Effektivität der Benutzerinnen/Benutzer-Interaktionen ab. Die Abbildung 17 zeigt ein Kontext Menu bei einem mobilen Apple-Gerät.

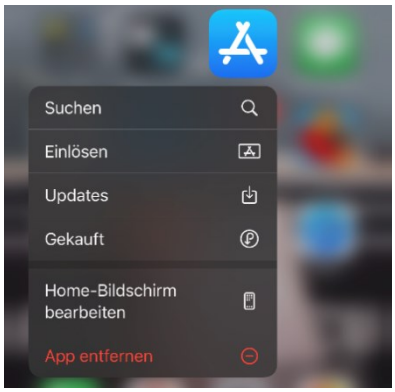

Abbildung 18: Schnellzugriffe per Kontextmenü (Eigene Darstellung)

4 EVALUIERUNG

Nachdem auf Basis der ISO 25010 in der Kriterien-Definitionsphase die Qualitätsmerkmale „factors“ und deren Teilmerkmale „criterions“ nach dem FCM-Modell spezifiziert wurden, werden diese im Anschluss mit Qualitätsindikatoren „metrics“ evaluiert (B. W. Boehm, J. R. Brown, M. Lipow 1978; ISO/IEC 25010:2011). Die Evaluierung wird, wie im Kapitel 1.2 Zielsetzung und Einschränkungen der Arbeit erläutert, seitens Progressive Web-Anwendungen mit dem mobilen Chrome, Safari und Samsung Internet Browser durchgeführt. Sollten sich diese Metriken nicht auf einen bestimmten Browser beziehen, werden diese nicht gesondert betrachtet. In Bezug auf nativen Anwendungen werden die Betriebssystem IOS und Android als Vergleichsgrundlage herangezogen.

Um die Abbildung 15, welche in der Kriterien-Defintionsphase generell nach ISO 25010 aufgestellt wurde, mit den ermittelten Kriterien zu ergänzen, wird im Anschluss die aktualisierte Abbildung dargestellt (Abbildung 18).

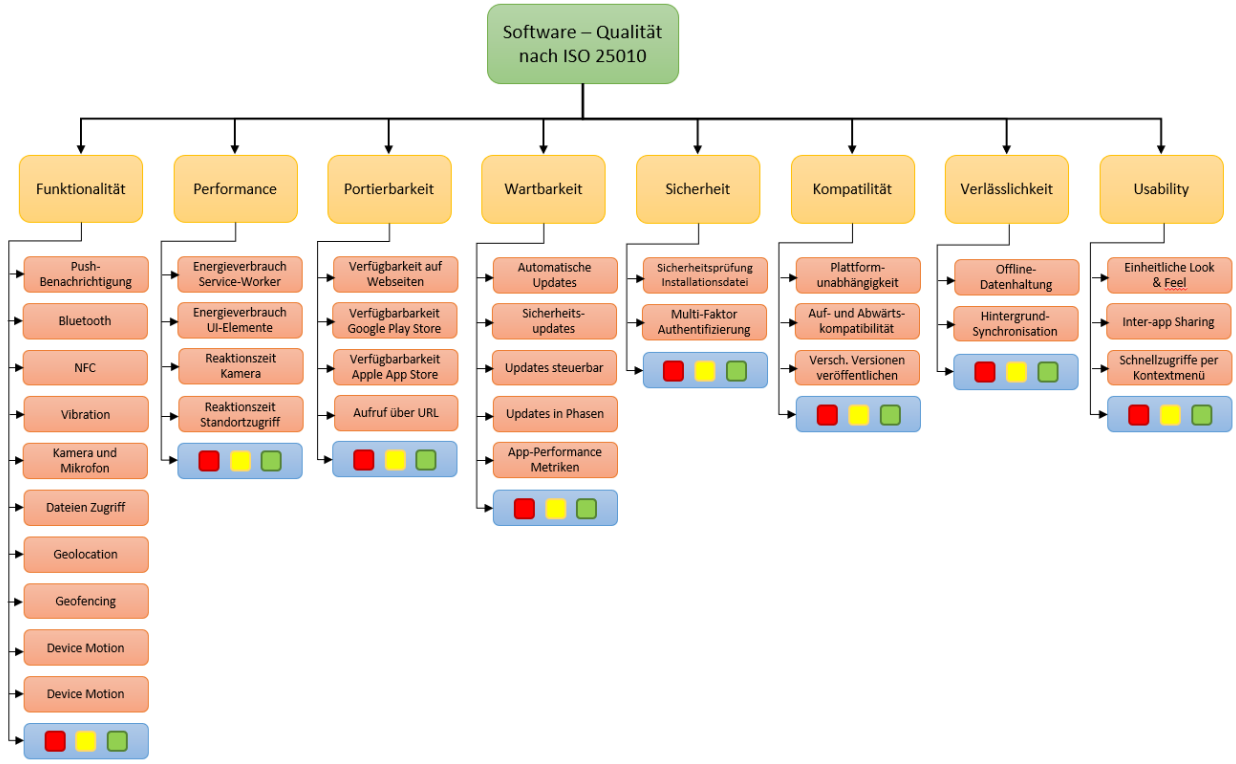

Abbildung 19: Aktualisiertes Qualitätsmodell (Eigene Darstellung)

Bei der Auswertung der Metriken werden diese so weit heruntergebrochen, um den Erfüllungsgrad einheitlich in drei Qualitätsindikatoren einteilen zu können:

- Qualitätsteilmerkmal erfüllt: Diese sind Grün in der Tabelle markiert.
- Qualitätsteilmerkmal erfüllt mit Einschränkungen: Diese sind Gelb in der Tabelle markiert mit dem Hinweis auf Einschränkungen, welche in der textuellen Beschreibung nachgelesen werden können.
- Qualitätsteilmerkmal nicht erfüllt: Diese sind Rot in der Tabelle markiert.

4.1 Analyse der Technologieansätze anhand der Kriterien

Im ersten Schritt wird wieder das ISO 25010 Qualitätsmerkmal (factors) der Funktionalität betrachtet. Hier werden die möglichen Teilmerkmale (criterions) verglichen bzw. deren Einschränkungen verdeutlicht (ISO/IEC 25010:2011).

4.1.1 Funktionalität

In erster Linie sollen die Zugriffsfreiheiten von diverser Software- und Hardwarekomponenten im Detail analysiert werden. Danach werden diese in einer Tabelle zusammengefasst. Diese Übersicht zeigt dabei, welche Features und Schnittstellen von Progressive Web-Anwendungen bzw. nativen Anwendungen zur Gänze, mit Einschränkung oder nicht erlaubt werden.

Push-Benachrichtigung: Push-Benachrichtigungen werden, wie bereits erwähnt, dazu genutzt, um kurze Nachrichten bei Endgeräten sichtbar zu machen, selbst wenn die Benutzerin bzw. der Benutzer die Anwendung gerade nicht aktiv nutzt (Warren et al. 2014). Ist dies eine Anforderung bei der Umsetzung einer mobilen Lösung, dann kann diese bei nativen Anwendungen mit vollem Umfang genutzt werden. Apple bietet hier das „Apple Push Notification Service“ an, welches die Vermittlung der generierten Nachrichten an die Endbenutzerinnen und -benutzer verwaltet (Apple Developer 2021a). Darüber hinaus gibt es eine Menge an Push-Notification-Server Anbieterinnen und Anbieter, welche im nativen Bereich für IOS und Android zur Verfügung stehen. Große bekannten Anbieterinnen und Anbieter bieten hier Lösungen an, wie zum Beispiel Google mit Firebase Cloud Messaging (Google Firebase 2021) oder Amazon mit deren „Simple Notification Service“ (Amazon AWS 2021). Somit kann auch seitens Android der volle Funktionsumfang benutzt werden.

Bei Progressive Web-Anwendungen ist das eingeschränkt. Hier gibt es Web Push API, welches es möglich macht, Push-Nachrichten in einer Browserumgebung zu realisieren. Push-Nachrichten werden dabei vom Server an den Service-Worker der PWA übermittelt, der ein Javascript-Event anstoßt, um den Geräte-Status zu aktualisieren und der Benutzerin bzw. dem Benutzer im Anschluss eine Benachrichtigung anzuzeigen (MDN Web Docs 2021r).

Beim Google Chrome und Samsung Internet Browser ist bei der Realisierung die einzige Einschränkung, dass ein Service-Worker registriert ist, welcher aber ohnehin die Grundvoraussetzung für eine PWA darstellt. Im Gegenzug dazu verweigert der IOS Safari Browser diese Web Push API (WHAT WEB CAN DO TODAY? 2021f).

Bluetooth: Für die Realisierung dieses Datenübertragungsstandards von Progressive Web-Anwendungen wird die Web Bluetooth API im Bereich von PWAs bereit gestellt (MDN Web Docs 2021v). Beim Safari Browser wird dieser Zugriff jedoch zur Zeit der Verfassung dieser Arbeit nicht unterstützt. Bei Chrome Android Browsern besteht im Gegenzug seit Version 56 die Möglichkeit, diese Schnittstelle einzusetzen (WHAT WEB CAN DO TODAY? 2021f). Hier gibt es jedoch auch eine Einschränkung (E1) bei der Verwendung der Bluetooth.getDevices() Funktion (MDN Web

Docs 2021v). Es besteht nicht die Möglichkeit, eine Liste gespeicherter Geräte abzufragen, ohne jedes Mal eine Berechtigungsaufforderung anzuzeigen. Hat die Endbenutzerin bzw. der Endbenutzer ein oder mehrere Geräte, welche sie bzw. er koppeln möchte, dann muss die Berechtigung bei jedem erneuten Öffnen erteilt werden (MDN Web Docs 2021e). Des Weiteren ist es bei der Web Bluetooth API nur möglich, BLE einzusetzen. Folglich ist es nicht möglich, große Datenmengen zu senden (E2) (MDN Web Docs 2021v).

Im Bereich der nativen Anwendungen von Android und IOS wird der volle Funktionsumfang gewährleistet. Es ist möglich, Geräte mit klassischem Bluetooth und dem Bluetooth Low Energy Übertragungsstandard zu koppeln und zu verwalten (Android Developers 2021g; Apple Developer 2021i).

Near-Field-Communication: Wenn Daten über kurze Distanzen übertragen werden müssen, kommt die NFC-Technologie zum Einsatz.

Soll dieser Technologieansatz Teil einer Progressive Web-Anwendung sein, gibt es die HTML 5 Web NFC API. Diese Schnittstelle ist jedoch limitiert auf das NDEF Protokoll, welches in der Kriterien-Definitionsphase erläutert wurde. Unabhängig von den Browseranbieter ist es somit nur möglich, die Technologie im Lese- und Schreibmodus zu verwenden. Aufgrund der Tatsache, dass das SNEP Protokoll nicht unterstützt wird, ist es nicht möglich, Daten direkt zwischen zwei NFC-Geräten auszutauschen (Google Github 2021).

Die Einschränkung wird größer bei der Betrachtung der Browserfunktionalitäten. Die Browser Safari und Samsung Internet unterstützen diese Technologie nicht. Nichtsdestotrotz unterstützt Google Chrome Browser diese API seit Version 95, mit genannter Limitierung auf das NDEF Protokoll (E3) (Can I use 2021d).

Im Bereich von nativen Android Anwendungen werden neben dem Lesen und Schreiben (NDEF Protokoll) eines NFC-Tags noch weitere Funktionalitäten geboten. Zum einen ist das der Peer-to-Peer Mode. Dieser erlaubt es, Daten innerhalb von zwei NFC Geräten auszutauschen (SNEP Protokoll). Diese Funktion wird zum Beispiel beim Feature „Android Beam“ verwendet. Darüber hinaus gibt es den „Host-Card-Emulation“ (HCE) Mode. In diesem Fall verhält sich das Mobilgerät wie eine NFC Karte. Diese emulierte NFC Karte kann dann von externen NFC-Lesegeräten erkannt werden, zum Beispiel bei „point-of-sales“ Terminals (Android Developers 2021r).

Im Gegenzug wird bei nativen IOS Anwendungen lediglich der Lese- und Schreibezugriff auf NFC Tags erlaubt. Einen Peer-to-Peer Mode mithilfe von NFC gibt es nicht (E4). Um dies zu gewährleisten, setzt Apple auf einen anderen Standard, genannt „AirDrop“, welcher als Basis eine WIFI-Verbindung zur Datenübertragung nutzt (Daniel Nations 2021). Darüber hinaus gewährt das Betriebssystem IOS nicht den HCE-Modus (E5), somit können Apple Endgeräte nicht als NFC–Karte simuliert werden (Apple Developer 2021n).

Vibration: Sollen definierte Interaktionen oder Ereignissen im Bereich von Progressive Web-Anwendungen mit einer Geste aufmerksam gemacht werden, wird die HTML 5 Vibration API bereitgestellt. Diese erlaubt es per Javascript, das Endgerät vibrieren zu lassen, sofern die dazugehörige Hardware vorhanden ist. Dabei ist es möglich, das mobile Endgerät nach eigenen Vibrations-Zeitintervallen zu steuern. Die mobilen Chrome und Samsung Internet Browser bieten

hier den vollen Funktionsumfang. Wobei der IOS Safari Browser den Zugriff zur Gänze nicht gewährt (Can I use 2021d; MDN Web Docs 2021t).

Wiederum gibt es hier bei nativen Anwendungen keine Einschränkung, um die zuständige Hardware zu steuern. Bei native IOS Anwendungen bietet Apple das Core Haptics Framework. Dies bietet der Entwicklerin bzw. dem Entwickler die Möglichkeit, neben dem Zeitintervall auch die Intensität und Empfindlichkeit per Parameter anzupassen. Dieselben Möglichkeiten bietet das Betriebssystem von Android (Apple Developer 2021k; Android Developers 2021ab).

Foto-, Ton-, Videoaufnahme: Eine weitere häufige Anforderung kann es sein, Foto-, Ton- oder Videoaufnahmen bereitzustellen. Aus diesem Grund, wie in der Kriterien-Definitions-Phase beschrieben, werden die Technologien auf diese Zugriffsfreiheiten betrachtet.

Seitens der Progressive Web-Anwendungen gibt es für die Anforderungen zum einen die HTML 5 Image Capture API, welche für Fotoaufnahmen konzipiert wurde, und zum anderen die Media Recorder API. Mit der zweitgenannten Schnittstelle ist es möglich, Ton- und Videomaterial aufzunehmen (WHAT WEB CAN DO TODAY? 2021a).

Die Image Capture API bietet die Möglichkeit, anhand der erweiterten Einstellungen, wie Zoom, Weißabgleich, ISO oder Fokuspunkte, Foto aufzunehmen. Bei der Verwendung der mobile Chrome und Samsung Internet Browser kann man den vollen Umfang dieser Schnittstelle genießen. Jedoch ist es seitens mobilen Safari Browsern nicht möglich, Fotoaufnahmen zu generieren (MDN Web Docs 2021j). Hier muss die HTML-Canvas-Methode gewählt werden, welche eingesetzt wurde bevor es die Image Capture API gegeben hat. Die exakte Umsetzung wird nicht in dieser Arbeit betrachtet. Jedoch kann angemerkt werden, dass die Methode Einschränkungen (E6) in der Auflösung aufweist, da keine echten 4k-Fotos aufgenommen werden. Auch die Zugriffszeiten sind im Verhältnis im Gegensatz zur HTML 5 Image Capture API sehr hoch (Rebecca Fransson 2017).

Die Media Recorder API wurde, wie bereits erwähnt, für Video- und Tonaufnahmen konzipiert. Im Gegenzug zu der Image Capture API ist der Zugriff neben dem Chrome und Samsung Internet auch beim Safari Browser unterstützt (MDN Web Docs 2021o).

Bei der Betrachtung der nativen IOS Anwendungen konnte festgestellt werden, dass es sowohl möglich ist, Fotos, Videos und Tonmaterial aufzunehmen (Apple Developer 2021g). Dasselbe gilt für Anwendungen, welche auf Geräten mit dem Betriebssystem Android installiert werden (Android Developers 2021f, 2021h).

Datei Zugriff: In vielen Fällen ist es eine funktionale Anforderung, eine lokale Datei vom mobilen Endgerät hochzuladen bzw. bereitzustellen.

Dafür gibt es auf der Seite von Progressive Web-Anwendung Browser File API. Der Zugriff ist auf allen untersuchten Plattformen von Progressive Web-Anwendungen und nativen Anwendungen erlaubt. Die hochgeladenen Dateien werden schreibgeschützt bereitgestellt und können somit nur im Read-Only-Modus verwendet werden (WHAT WEB CAN DO TODAY? 2021c).

Der Zugriff auf den Gerätespeicher, um Daten hochzuladen, wird dabei auch von beiden Plattformen IOS und Android unterstützt (Apple Developer 2021t; Android Developers 2021d).

Geolocation: In den Fällen, dass der aktuelle GPS-Standort in der Anwendung genutzt werden soll, ist es notwendig, diese per zur Verfügung stehenden Schnittstellen zu erlangen.

Um im Bereich der Browsertechnologien den aktuellen GPS-Standort zu nutzen, gibt es die HTML 5 Geolocation API. Nachdem die Endbenutzerin oder der Endbenutzer die Berechtigung zur Nutzung der GPS-Daten freigegeben hat, können alle untersuchten Browser diese Funktionalität nutzen (Can I use 2021a).

Darüber hinaus bieten beide nativen Anwendungsplattformen die Möglichkeit, den Standort des mobilen Endgerätes bereit zu stellen (Google Developers 2021d; Apple Developer 2021l).

Geofencing: Soll es dabei, wie im Kapitel der Kriteriendefinition erläutert, möglich sein, ein Betreten oder Verlassen eines definierten geologischen Bereichs zu erkennen, dann ist es notwendig einen „Geofence“ zu definieren. Aufgrund der Beziehung des aktuellen Standortes des mobilen Endgeräts und des vordefinierten Bereichs können spezielle Aktionen mit diesem Wissen ausgeführt werden.

Seitens der Progressive Web-Anwendungen gibt es die von Google noch in Entwicklung stehende Geofencing API. Dies ist aber noch nicht in der aktuellen Version von IOS Safari, Samsung Internet und Android Chrome bereitgestellt und somit nicht nutzbar (WHAT WEB CAN DO TODAY? 2021d).

Bei den nativen Plattformen ist es möglich, geografische Bereiche zu definieren und beim Betreten bzw. beim Verlassen eine Aktion auszuführen. Bei nativen Android als auch IOS Anwendungen wird sowohl beim Betreten eines geografischen Bereichs wie auch beim Verlassen ein Event ausgelöst, um mit gewünschten Aktionen darauf zu regieren. Android hat dabei ein Limit 100 Geofences pro Gerät. Im Gegenzug sind IOS Geräte auf 20 Bereiche pro Gerät eingeschränkt (Android Developers 2021j; Apple Developer 2021p).

Device Motion: Bei Betrachtung des Qualitätsteilmerkmal der Geräte-Bewegung konnten folgenden Informationen erfasst werden:

Im Bereich der Progressive Web-Anwendungen gibt es seit Mitte 2018 die Generic Sensor Browser-API, welche für die verschiedensten Sensoren spezifizierbar ist. Im Folgenden werden die unterschiedlichen Sensorentypen-Schnittstellen gegenüber der Zugriffsfreiheit der Browseranbieter betrachtet (WHAT WEB CAN DO TODAY? 2021b).

- **Accelerometer API:** Diese Schnittstelle, welche Informationen über die Beschleunigung der Achsen X, Y und Z liefert, wird im Chrome Browser ab Version 67 und im Samsung Safari Browser ab Version 9.0 unterstützt. Der Safari Browser auf der anderen Seite verweigert den Zugriff auf diesen Sensor (MDN Web Docs 2021c).
- **Gyroscope API:** Die Gyroscope API wird dazu verwendet, um Geschwindigkeitsinformationen aller drei Achsen des Gerätes zu erhalten. Beim Zugriff liegt dasselbe Verhalten der Browser wie bei der Accelerometer API vor (MDN Web Docs 2021h).
- **Magnetometer API:** Um das Magnetfeld des Endgeräts, zum Beispiel für einen Kompass, zu nutzen, ist diese Schnittstelle von Bedeutung. Im Gegensatz zu den davor genannten

Browserschnittstellen erlaubt hier nur der Chrome Browser den Zugriff auf den Magnetometer Sensor (MDN Web Docs 2021n).

Somit erlaubt der Chrome Browser alle, der Safari keine und der Samsung Browser mit Einschränkung auf den Beschleunigungs- und Geschwindigkeitssensor (E7) den Zugriff.

Im Anschluss werden die nativen Plattformen bezüglich der Messung von Geräte Bewegungen betrachtet. Beim Betriebssystem Android ist es möglich auf den Accelerometer, Gyroscope und Magnetometer Sensor zuzugreifen und die Informationen in der Anwendung zu verwenden (Android Developers 2021p). Seitens IOS sind die Zugriffsmöglichkeiten für die physikalischen Größen Beschleunigung, Geschwindigkeit und Magnetfeld gleichermaßen zur Verfügung (Apple Developer 2021m).

Fingerabdruck/Gesichtserkennung: Um eine passwortlose Authentifizierung und/oder eine sichere Zweifaktorauthentifizierung zu gewährleisten, ist der Zugriff zum Fingerabdruck und zum Gesichtserkennungs-Sensor notwendig. Deshalb werden die Technologieansätze bezüglich dieser Anforderungen betrachtet.

Bei Progressive Web-Anwendungen gibt es die Web Authentication API, welche eine Erweiterung der Credential Management API ist. Diese ermöglicht eine Authentifizierung mit Public-Key-Kryptografie. Diese kann jedoch nur beim Google Chrome und Apple Safari Browser genutzt werden. Bei Samsung Internet Browser wird der Zugriff zur Gänze verweigert (MDN Web Docs 2021u).

Die beiden nativen Varianten bieten im Gegenzug wieder den vollen Zugriff, sofern die notwendige Hardware im Endgerät verbaut ist (Apple Developer 2021d; Android Developers 2021z).

Tabelle 2 zeigt eine Übersicht über die oberhalb gewonnenen Erkenntnisse.

Qualitätsteilmerkmal	PWA			Native App	
	Safari	Chrome	Samsung	IOS	Android
Push Benachrichtigung					
Bluetooth		E1 + E2	E1 + E2		
NFC		E3		E4 + E5	
Vibration					
Kamera und Mikrofon	E6				
Dateien Zugriff					
Geolocation					
Geofencing					
Device Motion			E7		
Fingerabdruck/Gesichtserkennung					

Tabelle 2: Gegenüberstellung der Zugriffsfreiheiten von PWAs und nativen Anwendungen

4.1.2 Performance

Im nächsten Schritt wird das ISO 25010 Qualitätsmerkmal der Performance betrachtet, mit dem Ziel, eine Software zu entwickeln, welche ein gutes Zeitverhalten hat, Ressourcen effektiv nutzt und Kapazitäten so gut wie möglich schont (ISO/IEC 25010:2011).

Ein wichtiger Faktor, wie im Kapitel der Kriterien-Definition erläutert, ist dabei der Energieverbrauch der Anwendung, da die Akkulaufzeit eine begrenzte Ressource bei mobilen Endgeräten darstellt.

Gleichbleibender Energieverbrauch durch Service Worker: Sollte bereits eine klassische Web-Anwendung bestehen und man hat das Ziel daraus eine Progressive Web-Anwendung zu transformieren, möchte man nicht die Performance der bestehenden Lösung verschlechtern. Auf der Universität Vrije Amsterdam (2017) wurde dazu ein wissenschaftlicher Versuch veranlasst, welcher den Einfluss von Service-Workern auf die Energieeffizienz bei Web-Anwendungen betrachtet. Die Verfasser hatten sich das Ziel gesetzt, zu untersuchen, ob es einen signifikanten Unterschied zwischen mit bzw. ohne der Verwendung von Service-Workern gibt. Dabei wurden sieben etablierte, mit verschiedenen Features ausgeprägte Progressive Web-Anwendungen einmal mit und einmal ohne Service-Worker gemessen. Der Versuch wurde einmal mit einem 2G und einmal mit einem Wifi-Netz durchgeführt. Wie man der Abbildung 20 bzw. der Abbildung 21 entnehmen kann, konnte als Ergebnis dieser Studie die Erkenntnis gewonnen werden, dass unter diesen Voraussetzungen keine signifikanten Änderungen des Energieverbrauchs vorliegen. Dies kann anhand der Boxplot-Diagramme entnommen werden, da diese nur mit geringen Abweichungen im selben Energieverbrauchsbereich liegen. Somit müssen keine erwähnenswerten Einschränkungen bei der Migration einer bestehenden Webseite zu einer Progressive Web-Anwendung hingenommen werden (Ivano Malavolta et al. 2017). Aufgrund der Tatsache, dass die nativen Plattformen nicht auf Service-Workern aufbauen, wäre eine Analyse der Einschränkungen irrelevant und wird somit als gleichbleibende Energieeffizienz in der Übersichtstabelle deklariert.

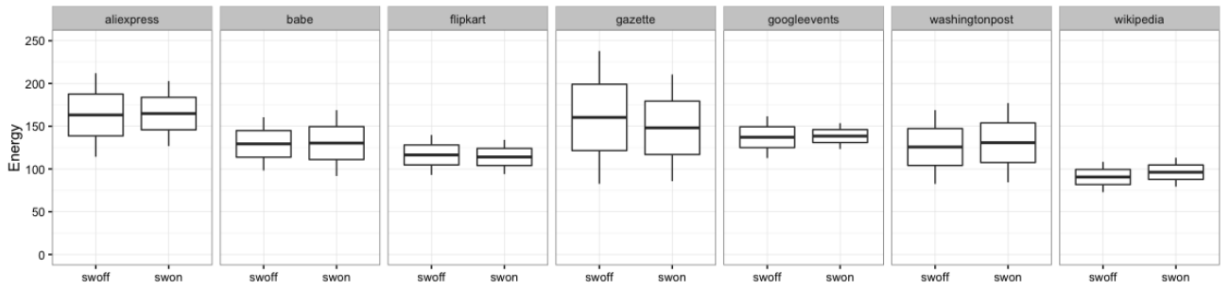

Abbildung 20: Energieverbrauchswerte mit ein- und ausgeschalteten Service-Workern unter einem 2G-Netz (in Joule) (Ivano Malavolta et al. 2017)

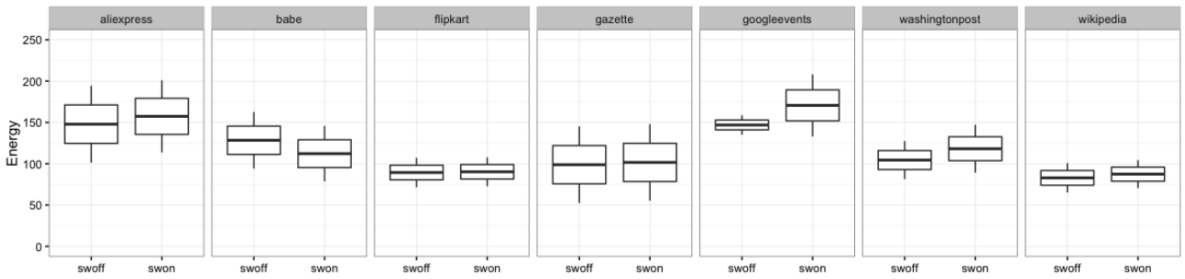

Abbildung 21. Energieverbrauchswerte mit ein- und ausgeschalteten Service-Worker unter einem WIFI-Netz (in Joule) (Ivano Malavolta et al. 2017)

Energieverbrauch bei Interaktionen mit UI-Elementen unter 4 Joule: Im Anschluss wird noch eine dritte Studie von Huber et. al (2021) betrachtet. Diese analysiert im Gegenzug zur davor genannten die Energieeffizienz von Interaktionen mit der Software. Dabei wurde die verbrauchte Energie bei Interaktion mit Menüs (Drawer), mit Listen, mit Schalter und bei der Eingabe von Formular-Daten in Joule gemessen. In dieser Studie wurden neben nativen Android und Progressive Web-Anwendungen auch weitere Technologien, wie Capacitor, React und Flutter, analysiert. Darüber hinaus wurden zwei Endgeräte bei dieser Studie verwendet, zum einen ein leistungsschwächeres Nexus X5 und zum anderen ein leistungsstarkes Samsung Galaxy S9. Wie bereits erwähnt, ist in Abbildung 22 zu sehen, dass dieser Forschungskontext nicht nur die Technologieansätze der Progressive Web-Anwendungen und nativen Anwendungen, sondern auch Cross-Plattform Technologien analysiert, welche aber in der vorliegenden Arbeit nicht betrachtet werden. Die Erkenntnisse dieser Arbeit sind, dass der Energieverbrauch bei Progressive Web-Anwendungen zwar signifikant höher ist, aber nur sehr gering im Mittel aller Messungen (Huber et al. 2021).

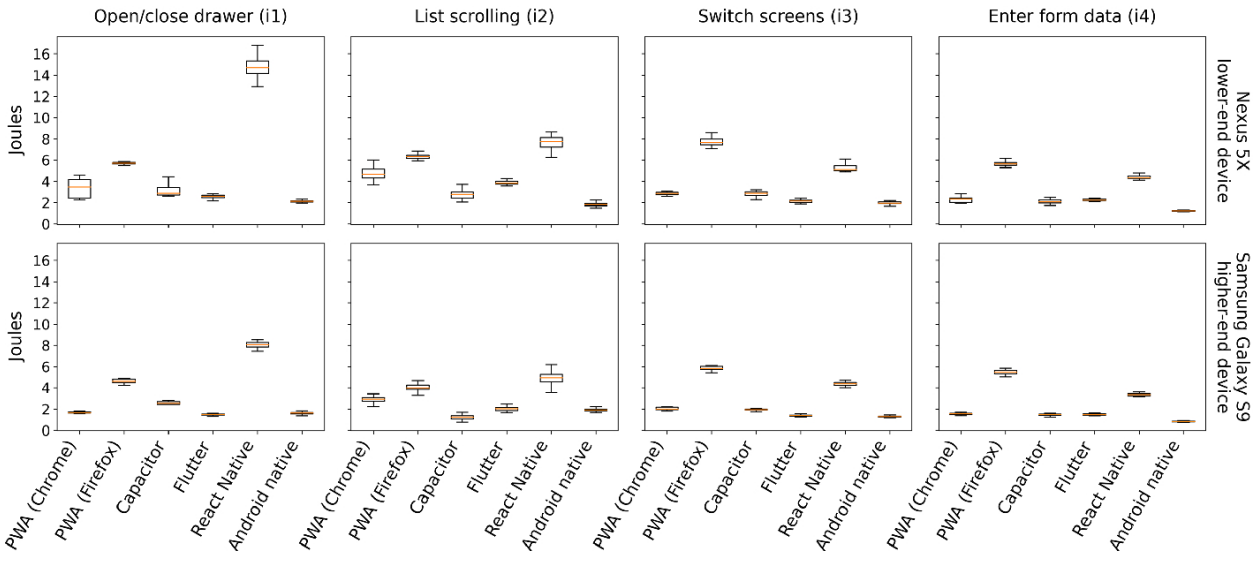

Abbildung 22: Energieeffizienz bei Anwedungsinteraktionen (Huber et al. 2021)

Reaktionszeit Kamerazugriff unter 1 Sekunde: Eine weitere Studie von Rebecca Fransson und Alexandre Driaguine (2017) hat sich mit einer ähnlichen Forschungsfrage beschäftigt. Dabei war das Ziel zu analysieren, ob Progressive Web-Anwendungen in der Leistung mit nativen

Anwendungen mithalten können, insbesondere in der Reaktionszeit beim Zugriff auf die Hardware des Gerätes. Diese Studie limitiert sich speziell auf Android Geräten mit dessen nativen Lösungsansatz bzw. im Bereich von Progressive Web-Anwendungen mit dem Google Chrome Browser. Im Forschungskontext wurden dabei Kamera- und Geolokalisierungszugriffe auf ihre Performance untersucht. Dabei wurde beschlossen, zwei PWA-Benchmarks für die Kamera-Analyse zu erstellen, einen mit der neueren Image Capture API und einen mit der klassischen Canvas-Methode. Die Canvas-Methode war die gängige Methode, um die Kamera einzubinden, bevor es die HTML 5 Image Capture API gab. Aufgrund der Tatsache, dass die Canvas-Methode keine echten 4K-Fotos liefern kann, wurde dieser Benchmark weiter in zwei Teile geteilt – zum einen mit einer 4K-Auflösung und zum anderen mit einer 720p-Auflösung. Die Intention dahinter war, wie sehr die Auflösung einen Einfluss auf die Performance beim Render-Prozess hat. In Abbildung 23 sind die Ergebnisse zusammengefasst. Dabei wurde 50 Messungen durchgeführt, wobei die Reaktionszeit in Millisekunden gemessen wurde. Dabei lässt sich erkennen, dass die klassische Canvas Methode mit einer 4k-Auflösung zu einer ca. 6-fachen Reaktionszeit führt gegenüber der Canvas-Methode mit einer 720p-Auflösung, der nativen Lösung und auch der Image Capture API. Grundsätzlich lässt sich aber sagen, dass die hohe Reaktionszeit bei Canvas 4k-Auflösungen keine Einschränkungen beim Chrome und Samsung Internet Browser bildet, da die Image Capture API bereits von diesen Browsern unterstützt wird. Der mobile Safari unterstützt, wie in der Evaluierung der Kamerazugriffe erwähnt, dabei diese Schnittstelle nicht, somit muss die Einschränkung (E1) einer niedrigeren Auflösung bei kurzer Reaktionszeit hingenommen werden. Diese API ermöglicht, wie bereits im Kapitel des Qualitätsmerkmal der Funktionalität erwähnt, die Steuerung von Kamerafunktionen wie Zoom, Helligkeit, Kontrast, ISO und Weißabgleich bzw. kann auf die volle Auflösungsfähigkeit jeder verfügbaren Gerätekamera oder Webcam zugegriffen werden (Rebecca Fransson 2017).

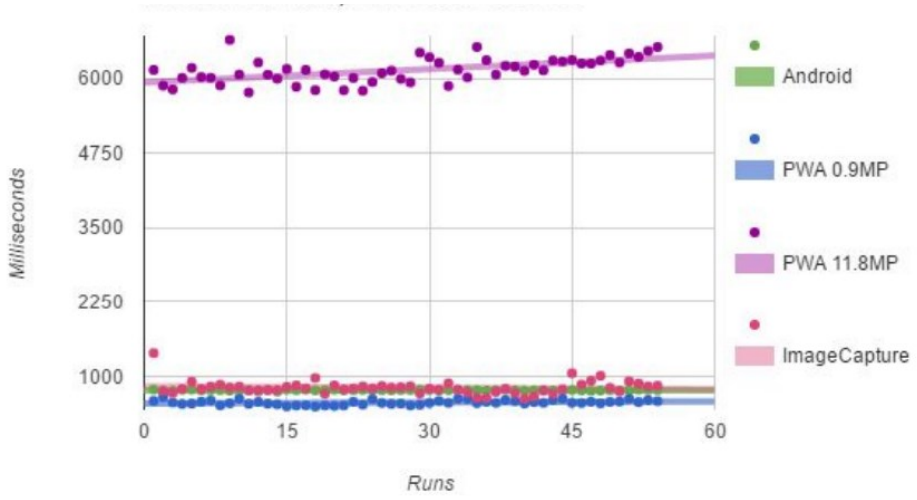

Abbildung 23: Reaktionszeit bei Kamera - Zugriffen (Rebecca Fransson 2017)

Reaktionszeit Standortzugriff unter 1 Sekunde: Des Weiteren wurden in dieser Studie die Zugriffe auf die geografischen Informationen mit messbaren Metriken hinterlegt. Dabei wurden die Google Maps API bzw. die im Einklang mit der bereits genannten Geolocation API im Bereich der Progressive Web-Anwendungen verwendet. Die Ergebnisse dieser Studie mit den

dazugehören Forschungsaufbau haben gezeigt, dass Progressive Web-Anwendungen im Schnitt geringere Reaktionszeiten wie native Android Anwendungen aufweisen. Diese Messungen werden in Abbildung 24 gezeigt. Des Weiteren wirkt sich der Caching-Mechanismus von Progressive Web-Anwendungen geringfügig positiv auf die Performance aus, da gewisse Google-Maps-Metadaten schon geladen sind (Rebecca Fransson 2017).

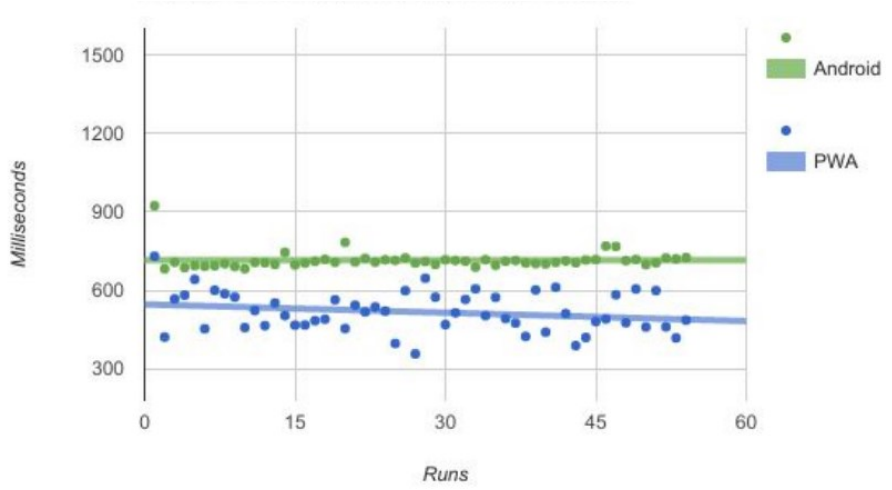

Abbildung 24: Reaktionszeit bei Geolokalisierung – Zugriffe (Rebecca Fransson 2017)

Zusammenfassend lässt sich sagen, dass der Energieverbrauch von Progressive Web-Anwendungen mit einem Service-Worker nur sehr gering höher ist als jener von nativen Anwendungen und somit keine Einschränkungen mit sich trägt. Des Weiteren lässt sich nur ein sehr geringer Unterschied bei den Zugriffszeiten erkennen. Je nach Hardware-Zugriff hat einmal die browserbasierte Technologie rund um Progressive Web-Anwendungen bessere Ergebnisse und einmal die der nativen Anwendungen. Die einzige Ausnahme bildet der IOS Safari Browser beim Kamera-Zugriff, da diese die Image Capture API nicht zulässt und somit die ineffizientere Canvas-Variante nutzen muss.

Tabelle 3 gibt einen Überblick über die Ergebnisse.

Qualitätsteilmerkmal	PWA			Native App	
	Safari	Chrome	Samsung	IOS	Android
Gleichbleibende Energieverbrauch durch Service-Worker					
Energieverbrauch bei Interaktionen mit UI-Elementen unter 4 Joule					
Reaktionszeit Kamerazugriff unter 1 Sekunde	E1				
Reaktionszeit Standortzugriff unter 1 Sekunde					

Tabelle 3: Gegenüberstellung der Performance von PWAs und nativen Anwendungen

4.1.3 Portierbarkeit

Im nächsten Schritt wird das Qualitätsmerkmal (factor) der Portierbarkeit im Detail erfasst. Grundsätzlich beschäftigt sich die Portierbarkeit mit dem Ziel, eine Software so anzupassen, dass diese auf anderen Plattformen lauffähig bzw. leicht zu installieren ist. Der Zweck dieser Evaluierung ist die in Kriterien-Definitionsphase erläuterten Qualitätsteilmerkmale zu analysieren. Diese Arbeit beschäftigt sich auf der einen Seite mit der Verfügbarkeit auf den unterschiedlichen App Stores. Auf der anderen Seite werden die Technologieansätze betrachtet, ob es möglich ist diese über eine URL aufzurufen.

Verfügbarkeiten: Im nächsten Schritt wird betrachtet, wie Progressive Web-Anwendungen und native Anwendungen zum Download verfügbar gemacht werden können. In Betracht werden dabei der Google Play Store, der Apple App Store und klassische Webseiten gezogen, welche die Installationsdateien zur Verfügung stellen.

Android: Wenn man die Installationsmöglichkeiten von Android Anwendungen betrachtet, gibt es mehrere Möglichkeiten, wie man diese Softwarelösungen veröffentlichen und zum Download anbieten kann. Zum einen kann eine Android Anwendung über dessen offiziellen App Store, den Google Play Store, veröffentlicht werden (Android Developers 2021u). Im Gegenzug dazu gibt es noch eine Menge von weiteren App Stores, welche aber verhältnismäßig eine viel geringere Anzahl an Kundinnen und Kunden haben, deshalb werde diese außer Acht gelassen werden (L. Ceci 2021). Da es sich bei einem Android-Package (APK), um eine Android spezifische Installations-Datei handelt, ist es nicht möglich, diese auf dem Apple App Store hochzuladen.

Außerdem gibt es noch die Möglichkeit, die Installationsdatei direkt über eine eigene Webseite bereitzustellen, jedoch hat man hier keinen Zugriff auf Entwicklertools, welcher der Google Play Store bietet (E2). Darüber hinaus ist beim Download verpflichtend die Installation zu bestätigen, weil solche Anwendungen vom Betriebssystem als nicht vertrauenswürdig eingestuft werden, da hier die Überprüfung seitens des Google Play Stores entfällt (E3). Aus diesem Grund ist das Qualitätsteilmerkmal „Verfügbarkeit auf Webseite“ nur mit Einschränkungen erfüllt (Android Developers 2021u).

Im Anschluss wird der Prozess zur Veröffentlichung einer Android Anwendung im Google Play Store betrachtet. Bevor man jedoch eine Anwendung bereitstellen kann, muss ein Google Developer Account erstellt werden, welcher einmalig 25 US-Dollar kostet, andernfalls ist es nicht möglich, Anwendungen auf dieser Plattform zu veröffentlichen. Um dann die Anwendung verfügbar zu machen, gibt es einen von Google vorgeschlagenen Prozess. Dieser besteht hauptsächlich aus zwei Aufgaben. Im ersten Schritt muss die Anwendung für die Veröffentlichung vorbereitet werden. Anhand von diesen vorgegeben Vorbereitungsschritten muss eine Release-Version der Anwendungen erstellt werden. Die zweite Hauptaufgabe ist die Veröffentlichung der aktuellen Release-Version. Da diese zwei Aufgaben eine Menge an Schritten mit sich bringen, werden diese im Anschluss betrachtet. Als Basis wird hierfür die offizielle Dokumentation des Google Play Store herangezogen (Android Developers 2021t).

Zu Beginn werden die Vorbereitungsschritte der Anwendung betrachtet. Die offizielle Android-Developer Plattform bietet hierfür eine Checkliste. Diese besteht im Wesentlichen aus drei

Schritten. Im ersten Schritt „Sammeln von Materialien und Ressourcen“ ist es auf der einen Seite notwendig, einen kryptografischen Schlüssel zu erstellen, um die Anwendung zu signieren. Zum anderen müssen die Anwendungsicons eingefügt werden. Über den Mindestanforderungen hinweg ist es noch möglich, eine Lizenzvereinbarung (End User License Agreement (EULA)), Wertetexte, Screenshots und weitere Beschreibungen hochzuladen. Im darauffolgenden Schritt „Konfigurieren der Anwendung für die Freigabe“ werden weitere Schritte von Google Play Store empfohlen. Dabei soll die Manifest-Datei, welches die essenziellen Informationen der Anwendung beschreibt, angepasst werden. Eine beispielhafte Manifest-Datei ist in Abbildung 25 veranschaulicht.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="string"
          android:sharedUserId="string"
          android:sharedUserLabel="string resource"
          android:versionCode="integer"
          android:versionName="string"
          android:installLocation=["auto" | "internalOnly" | "preferExternal"]
    . . .
</manifest>
```

Abbildung 25: Android Manifest File (Android Developers 2021a)

In diesem Schritt ist es von Notwendigkeit, einen eindeutigen Package-Namen, Versionscode und Versionsname zu vergeben. Die Versionsangaben sind in weiterer Folge bei Updates wichtig, welche im Qualitätsteilmerkmal „Wartung“ näher betrachtet werden. Ein weiterer entscheidender Faktor ist das Konfigurieren des Lizenz-Modells (Google-Play-Licensing-Service), sofern die Benutzung der Anwendung nur unter Bezahlung möglich ist.

Der nächste und dritte Vorbereitungsschritt ist das „Erstellen der Anwendung für die Veröffentlichung“. Ziel ist es, eine Android-Package Datei (APK) zu erzeugen, welche die Installationsdatei von Android-Anwendungen darstellt. Die Entwicklungs-IDE Android Studio von Google bietet dabei Unterstützung, eine APK Datei für den Release zu generieren. Im Zuge der Erstellung einer „release-ready“ APK ist es notwendig, die Anwendung mit dem zuvor generierten Schlüssel zu signieren. Diese Signierung muss gemacht werden, um die Autorin bzw. den Autor von der Anwendung eindeutig mit einem Zertifikat zu identifizieren. Dies dient als Sicherheitsmechanismus, damit bei Änderungen und Updates gewährleistet werden kann, dass es sich um die ursprüngliche Entwicklerin bzw. den ursprünglichen Entwickler dieser Anwendung handelt.

Wie bereits erwähnt, ist die zweite Hauptaufgabe die Veröffentlichung der Android Anwendung, nachdem diese erstellt und signiert wurde. Dabei wird die APK Datei im Administrationsportal, der Google Play Console, hochgeladen, welche einer Schadsoftware-Überprüfung unterzogen wird. Diese kann bis zu sieben Tage dauern. Ist diese Prüfung wiederum erfolgreich, steht die Anwendung ab diesem Zeitpunkt im Google Play Store zur Verfügung (Android Developers 2021t).

Native IOS Anwendungen: Im nächsten Schritt wird die Veröffentlichung von nativen IOS Anwendungen betrachtet. Im Gegensatz zum Android App Store gibt es hier nur den offiziellen

Apple App Store, um die Installationsdatei (.ipa) hochzuladen. Damit eine Anwendung auf der Plattform veröffentlich werden kann, werden bereits bei der Entwicklung zwei Dienste von Apple benötigt. Die Entwicklungsgrundlagen bilden zum einen das „Apple Developer Program“ und zum anderen der „iTunes Connect“ Dienst. Für die Veröffentlichung einer IOS Anwendung ist darüber hinaus zwingend ein Mac-Rechner mit einer aktiven Internetverbindung notwendig. Des Weiteren muss die offizielle IDE von Apple installiert sein, da nur aus dieser integrierten Entwicklungsumgebung eine Binärdatei für den App Store generiert werden kann. Ergänzend muss noch eine Mitgliedschaft mit einer einmaligen Apple-ID, um die Berechtigung zum Veröffentlichen zu erhalten, erworben werden. Diese kostet jährlich 99 US-Dollar. Wurden all diese Voraussetzungen erfüllt, kann eine IOS Anwendung anhand folgender vier Hauptschritte veröffentlicht werden (Ionos 2021):

Im ersten Schritt müssen ein Development- sowie ein Distribution-Zertifikat im „Apple Developer Program“ angelegt werden, um diese in weiterer Folge für die Signierung der Anwendung zu verwenden (Ionos 2021).

Die Veröffentlichung im App Store erfordert zudem noch das Anlegen einer App-ID der Anwendung. Diese stellt eine einmalige Identität der Anwendung dar, welche aus zwei Teilen besteht. Der erste Teil wird automatisch von Apple generiert, der zweite Teil ist von der Entwicklerin bzw. dem Entwickler eine wählbare Endung. Diese Endung kann eine „Explicit App ID“ oder „Wildcard App ID“ sein. Der Unterschied liegt in der Häufigkeit der Verwendung. Eine „Explicit App ID“ wird nur für eine Anwendung verwendet. Eine „Wildcard App ID“ wird für mehrere Anwendungen wiederverwendet. Da aber keine einmalige Erkennung vorhanden ist, ist es nicht möglich auf Push-Benachrichtigungen, In-App-Bezahlungen und weitere Features zurückzugreifen (Apple Developer 2021x).

Im dritten und letzten Schritt „Apple Developer Programm“ muss ein Provisioning-Profil erstellt werden. Ziel der Erstellung ist es, die davor generierten Zertifikate und die App ID zu einem einheitlichen Profil zusammen zu führen. Des Weiteren ist es notwendig, mithilfe von der IDE XCode die Anwendung dem Developement- und Distribution-Zertifikat zu signieren (Ionos 2021).

Im letzten Schritt muss ein iTunes-Connect Datensatz erstellt werden, welcher alle Informationen zur Verwaltung der Anwendung erfasst. Hier werden der Anwendungsname, die Sprachen, die Kategorisierungen, die Pricing Modelle und allgemeine Beschreibungen eingerichtet. In weitere Folge muss das Herzstück der Anwendung, die Build-Datei (.ipa), bereitgestellt werden. Dies geschieht mit der integrierten Entwicklungsumgebung XCode. Diese erstellt die Binärdatei und kann diese direkt im iTunes Connect hochladen. Nachdem die Anwendung erfolgreich hochgeladen wurde, wird sie von Apple auf Schadsoftware untersucht und in der Regel nach zwei Tagen veröffentlicht, wenn keine Auffälligkeiten vorhanden sind (Ionos 2021).

In Bezug auf die Verfügbarkeit lässt sich Folgendes sagen: Wie bereits erwähnt, gibt es für den Download von nativen IOS Anwendungen offiziell nur den Apple App Store. Auch die manuelle Installation einer Binärdatei wird im Gegensatz zu Android Anwendungen (.apk) von Apple nicht erlaubt. Somit ist die Verfügbarkeit einzig auf den Apple App Store beschränkt.

Progressive Web-Anwendungen: Bei Progressive Web-Anwendungen hebt sich der Prozess zur Veröffentlichung etwas von den nativen Varianten ab. Da Progressive Web-Anwendungen in

der Basis klassische Webseiten sind, ist der Prozess der Veröffentlichung auch derselbe. Mit der Voraussetzung, dass das HTTPS-Protokoll verwendet werden muss, um die Sicherheit des Service-Workers zu gewährleisten. Dieser Prozess wird im Anschluss dem nativen Varianten gegenüberstellt (MDN Web Docs 2021m).

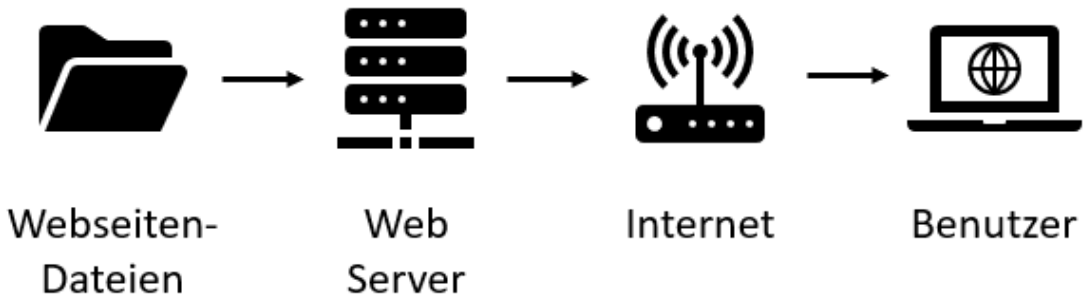

Abbildung 26: Hosting Progressive Web-Anwendung (Eigene Darstellung)

Prinzipiell kann der Prozess, wie in Abbildung 26 vereinfacht dargestellt, wie folgt zusammengefasst werden: Die Projekt-Files müssen auf einem Web-Server hochgeladen werden. Stellt dabei eine Endbenutzerin bzw. ein Endbenutzer eine Anfrage zur Auslieferung der Seite, wird diese mit einer aktive Netzwerkverbindung vom Web-Server ausgeliefert und im Browser darstellt (Rossi 2008).

Nach der Entwicklung der Anwendung ist es möglich, das Softwareprojekt sofort zu veröffentlichen, ohne es zu signieren. Dazu muss lediglich eine einmalige Domaine erworben werden, welche die URL darstellt, von der die Anwendung erreichbar ist. Um die Softwarelösung weltweit verfügbar zu machen, ist es darüber hinaus notwendig, diese auf einem Webserver auszuliefern. Dabei gibt es zwei Varianten – entweder man verwendet Hosting-Serviceanbieter oder man liefert die Anwendung direkt über einen eigenen Server aus. Für die Auswahl der richtigen Variante müssen die Vor- und Nachteile der unterschiedlichen Ansätze abgewogen werden, welche aber in dieser Arbeit nicht näher betrachtet werden (Lawrence 2007).

Das Qualitätsteilmerkmal der Verfügbarkeit auf Webseiten ist somit bei einer Progressive Web-Anwendung sichergestellt, da der Browser-URL-Aufruf bereits die Anwendung selbst bereitstellt. Wie in Abbildung 3 gezeigt, wird ein Installation-Icon in der Suchleiste dargestellt, sofern es sich um eine Progressive Web-Anwendung handelt. Somit kann die Installation am Homescreen des mobilen Gerätes vom Browser heraus gewährleistet werden.

Hat man die Anforderung, eine Progressive Web-Anwendung in einen der beiden offiziellen App Stores zu laden, kann sie beim Google Play Store mit wenigen Anpassung veröffentlicht werden (Chrome Developer 2021). Beim Google Play Store gibt es seit Anfang des Jahres 2020 die Möglichkeit, Progressive Web-Anwendungen zu veröffentlichen, da dort die Trusted Web Activities API (TWA) vorgestellt wurde. Dank dieser Technologie können Entwicklerinnen und Entwickler eine Play Store Installationsdatei (.apk) generieren, obwohl man das bestehende Progressive Web-Anwendung Projekt verwenden kann und ohne dafür einen nativen Code zu schreiben. Es wird dabei ein Android Package auf den Google Play Store geladen, welche nur

einen Link zu der Webseite mitliefert. Wie bereits bei Veröffentlichung von nativen Android Anwendungen beschrieben, ist es notwendig, das Projekt mit einem Zertifikat zu signieren, um eine einmalige Identität zu gewährleisten. Um die Signierung durchzuführen, muss eine sogenannte „Digital Asset Links“ Datei im Projekt hochgeladen werden. Für die Erstellung einer solchen Datei bietet Google einen Service, der den notwendigen Quellcode, wie Abbildung 27 dargestellt, erzeugt (Google Developers 2021e; Chrome Developer 2021).

```
[{
  "relation": ["delegate_permission/common.handle_all_urls"],
  "target" : { "namespace": "android_app", "package_name": "com.example.saurer",
               "sha256_cert_fingerprints": ["e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855"] }
}]
```

Abbildung 27: Auto-generierter Digital Asset Links Quellcode (Google Developers 2021e)

Wurden all diese Schritte erfüllt, kann die generierte Android Package Datei (.apk) mit denselben Schritten wie bei der nativen Variante zur Veröffentlichung durchgeführt werden. Somit ist es mit der Einschränkung der Verwendung von Trusted Web Activities (E4) möglich, Progressive Web-Anwendungen auf den Google Play Store zu laden.

Beim Apple App Store wird das komplett ausgeschlossen. Apple hat eine sehr eindeutige Ansicht zu Progressive Web-Anwendungen in seinem App Store. Das Unternehmen bietet für die Veröffentlichung in seinem Store die „App Store Review Guidelines“. Bei den minimalen Funktionalitäten wird dabei von Apple folgende Aussage getroffen, welche Progressive Web-Anwendungen nicht erfüllen können: „Your app should include features, content, and UI that elevate it beyond a repackaged website. If your app is not particularly useful, unique, or “app-like,” it doesn’t belong on the App Store.“ (Apple Developer 2021c).

Aufruf über URL: Da beide nativen Varianten eine plattformspezifische Installationsdatei (.apk und .ipa) zur Bereitstellung verlangen, ist es nicht möglich, diese im Browser per URL-Aufruf zu erlangen. Diese können folglich nicht in einem Browser lauffähig gemacht werden, da sie im Kern aus den Web-Technologien (HTML, CSS und Javascript) bestehen müssten (Droid Wiki 2021; fileformat 2021).

Im Gegensatz zu den nativen Varianten ist bei Progressive Web-Anwendungen der URL-Aufruf gewährleistet, aufgrund der Tatsache, dass diese im Ursprung eine Webseite darstellen und über eine URL im Browser erreichbar sind (Google web.dev 2021).

Tabelle 4 fasst die Ergebnisse in einer Tabelle zusammen.

Qualitätsteilmerkmal	PWA			Native App	
	Safari	Chrome	Samsung	IOS	Android
Verfügbarkeit auf Webseiten					E2 + E3
Verfügbarkeit auf Google Play Store	E4	E4	E4		
Verfügbarkeit auf Apple App Store					
Aufruf über URL					

Tabelle 4: Gegenüberstellung der Portierbarkeit von PWAs und nativen Anwendungen

4.1.4 Wartbarbeit

In Anschluss wird das ISO 25010 Qualitätsmerkmal der Wartbarkeit betrachtet. Ziel ist es, eine Softwarelösung mit einem geringen Grad an Aufwand zu verbessern, zu korrigieren oder an neue Anforderungen und Umgebungsbedingungen anzupassen (ISO/IEC 25010:2011).

Updates: Da sich, wie bereits in der Kriterien-Definitionsphase erwähnt, Anforderungen ändern, aber auch Fehler einschleichen können, ist es für Softwarelösungen unumgänglich, Updates über den gesamten Lifecycle hinweg auszuliefern. Hierfür werden folgende Qualitätsteilmerkmale betrachtet:

- Automatische Updates: Können Updates automatisch am Endgerät installierbar gemacht werden?
- Sicherheitsupdates: Kann ein sicherheitsrelevantes Update jederzeit installiert werden?
- Update von der Endbenutzerin bzw. vom Endbenutzer steuerbar: Kann die Endbentzerin bzw. der Endbenutzer die Updates selbst entscheiden?
- Updates in Phasen veröffentlichen: Lassen sich Updates in Phasen ausliefern, um Updates bei speziellen Benutzerinnen und Benutzer zu testen?

Im Anschluss werden die vier genannten Qualitätsteilmerkmale bei den unterschiedlichen Technologieansätzen evaluiert.

Progressive Web-Anwendungen: Seitens Progressive Web-Anwendungen ist es im Zuge eines Updates notwendig, den Service-Worker zu aktualisieren. Grundsätzlich steht man vor dem Problem, dass sich alle Browserfenster denselben aktiven Server-Worker teilen, da dieser ein einzelner Controller und Netzwerk-Proxy für alle geöffneten Anwendungen ist. Somit kann nicht nahtlos aktualisiert werden, solange es einen geöffneten Tab gibt. In manchen Fällen kann es auch erwünscht sein, dass ein Tab für immer geöffnet ist. In solchen Fällen würde sich der Service-Worker bzw. die dazugehörige Progressive Web-Anwendung nie aktualisieren. Des Weiteren reicht eine einfache Aktualisierung des Browser-Tabs nicht aus, um mögliche neue Inhalte zu laden. Hat man aus den genannten Gründen keine Update-Strategie, könnte es passieren, dass eine alte Version ewig läuft und die Endbenutzerin bzw. der Endbenutzer immer mit Inhalten aus dem Cache versorgt wird. Deshalb bieten Service-Worker die Möglichkeit, programmatisch die Kontrolle über die existierenden Clients zu übernehmen. Hierfür kann man die „self.skipWaiting()“ Funktion des Service-Workers aufrufen. Gibt es eine neue Version, wird der zuvor aktive Service-Worker gestoppt und der neue aktiviert, sodass die derzeit geöffneten Anwendungen mit der neuen Version aktualisiert werden. Würde man das „self.skipWaiting()“ willkürlich und sofort nach dem Eintreffen eines Updates aktualisieren, könnte es jedoch zu Problemen bei den Daten, zu Unterbrechungen bei Ladevorgängen bzw. Bezahlvorgängen und so weiter kommen. Um den Benutzerfluss nicht zu unterbrechen, muss eine Variante gefunden werden, um Updates durchzuführen. Hierfür gibt es mehrere Ansätze, welche von den Eigenschaften der Anwendung abhängig sind (Google Web 2021; WHAT WEB CAN DO TODAY? 2021e).

- Hat man die Möglichkeit zu erkennen, ob die Benutzerin bzw. der Benutzer inaktiv ist, kann man die Aktualisierung durchführen, während keine aktive Aktion ausgeführt wird. Jedoch ist diese Art der Erkennung oft sehr schwierig. Üblich ist es aber, diesen Ansatz zu wählen, um sicherheitsrelevante Updates automatisch ohne Zustimmung der Benutzerin bzw. des Benutzers bereit zu stellen.
- Ein weiterer Ansatz wäre es, beim Wechsel zwischen den Seiten einer Anwendung, diese zu aktualisieren. Dieser Ansatz ist bei Single-Page-Anwendungen jedoch nicht durchführbar bzw. kann es wieder zu Problemen kommen, wenn aktive Ladevorgänge unterbrochen werden.
- Darüber hinaus könnte man den Ansatz verfolgen, am Endgerät eine UI-Benachrichtigung anzuzeigen, um auf eine neue Version aufmerksam zu machen. Die Endbenutzerin bzw. der Endbenutzer muss dann aktiv das Update einleiten.

Da die letztgenannte Update-Strategie für den Großteil der Anwendungsbeispiele gültig ist bzw. programmatisch sehr den anderen Ansätzen ähnelt, wird dieser Prozess in Abbildung 28 anhand von Quellcode-Ausschnitten beschrieben.

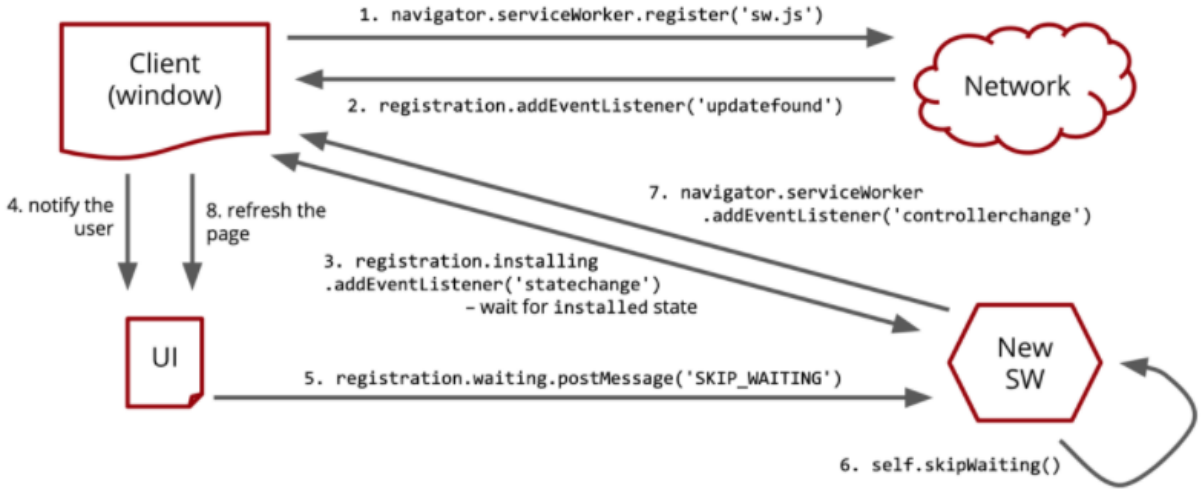

Abbildung 28: Progressive Web-Anwendung Updateprozess (WHAT WEB CAN DO TODAY? 2021e)

Im ersten Schritt wird der Service-Worker registriert. Danach wird im zweiten Schritt diese davor generierte Instanz genutzt, um einen Event-Listener mit dem „updatefound“ Event zu versehen. Wird dieses Event aufgerufen, ist eine neue Service-Worker-Version verfügbar. Prinzipiell wird eine Änderung des Service-Workers erkannt, wenn sich das Service-Worker-Javascript File ändert. Es wird dabei eine „byte-by-byte content comparison“ durchgeführt. Hat man zum Beispiel eine Konstante mit der Versionsnummer in diesem File und ändert diese sich bei einem Update, wird eine neue Version erkannt, weil diese bei der „byte-by-byte content comparison“ als Änderung erkannt werden. Darüber hinaus informiert das „statechange“ Event im Schritt 3 das System, dass der Service-Worker vollständig verfügbar ist. Im Schritt 4 wird dann die Endbenutzerin bzw. der Endbenutzer informiert, dass es eine neue Version gibt. Wird das Update akzeptiert, dann wird per „registration.waiting.postMessage(„SKIP_WAITING“)“ der Service-Worker darüber informiert. Die wirkliche Aktualisierung des Service-Worker erfolgt dann im Schritt

6 mit der „self.skipWaiting()“ Funktion. Nachdem daraufhin im Schritt 7 das „controllerChange“ Event des Service Workers die vollständige Aktualisierung bekannt gibt, wird im letzten Schritt die grafische Benutzeroberfläche aktualisiert (WHAT WEB CAN DO TODAY? 2021e).

Aus den erlangten Informationen kann man zusammenfassend sagen, dass automatische Updates bei Progressive Web-Anwendungen möglich sind, da durch das „updateFound“ Event und der „self.skipWaiting()“ Funktion Updates automatisiert erfolgen können. Dasselbe gilt für Sicherheitsupdates. In diesem Fall könnte man die Versionsnummer-Konstante mit einem Flag versehen, damit das System Sicherheitsupdates erkennt und automatisch ohne Rückfragen direkt durchführt. Die Erfüllung des Qualitätsteilmerkmals, dass das Update von der Endbenutzerin bzw. dem Endbenutzer sein soll, kann, wie die Abbildung 28 zeigt, programmatisch durchgeführt werden. Bei der Betrachtung des Qualitätsteilmerkmals der Updates in Phasen, ist es nicht ohne immensen Aufwand möglich eine neue Version gestaffelt auszuliefern, da die Entwicklerin bzw. der Entwickler die Update-Strategie selbst umsetzen muss.

Android: In einem weiteren Schritt wird der Updateprozess bei nativen Android-Anwendungen betrachtet. Wurde ein App-Bundle, wie bereits bei dem Qualitätsteilmerkmal Portierbarkeit erklärt, veröffentlicht, kommt es in weitere Folge bei Updates zu folgendem Prozess: Laut der Google Play Console-Hilfe (Google Support 2021b) ist es notwendig drei Schritte zu erledigen, um das Update vorzubereiten:

- Der eindeutige Package-Name der Manifest-Datei muss mit der momentan aktuellen Version übereinstimmen.
- Die aktualisierte Version muss mit demselben Prozess wie bei der initialen Installation signiert werden. Die Signaturen müssen somit ident sein.
- Die Versionsparameter der Anwendung müssen erhöht werden, damit die aktualisierte Version als Update der momentanen Version erkannt werden kann.

Da die ersten zwei Schritte keine explizite Erklärung aufgrund ihrer Trivialität benötigen, wird im Weiteren auf den dritten Schritt der Erhöhung der Versionsparameter eingegangen. Bei der Definierung der Versionsinformationen müssen die Werte in der Gradle-Build-Datei (build.gradle) definiert werden. Diese Datei beinhaltet grundsätzlich Informationen zum Build-, Test- und Veröffentlichungsprozess und somit auch die Versionsdaten. Wie jedoch in Abbildung 25 ersichtlich, gibt es in der Manifest-Datei auch Versionsinformationen. Die Android Developer Dokumentation weist darauf hin, dass die Versionsangaben im Gradle-Build-File gemacht werden sollen, weil in letzter Instanz die Manifest-Datei überschrieben wird. Hierfür gibt es fünf Versionsparameter (Android Developers 2021aa):

- versionCode
- versionName
- minSdkVersion
- maxSdkVersion
- targetSdkVersion

Der Versionscode soll eine positive, ganze Zahl sein, welche nur als interne Versionsinformation verwendet wird. Sie wird der Benutzerin bzw. dem Benutzer nicht angezeigt. Das Android Projekt verwendet diesen Versionscode lediglich zum Schutz vor Downgrades, damit keine niederen APK Dateien als die derzeitigen am Endgerät installiert werden (Android Developers 2021aa).

Der Versionsname wird im Gegensatz dazu den Benutzerinnen und Benutzer engezeigt. Hier empfiehlt Google, die Version anhand der Semantic-Versioning anzugeben. Diese ist wie folgt aufgebaut: - <MAJOR>.<MINOR>.<PATCH>. Wobei die MAJOR Nummer erhöht wird, wenn API inkompatible Veränderungen bereitgestellt werden. Die MINOR Nummer wird bei neuen Funktionalitäten und die PATCH Nummer bei Bugfixes erhöht (Semantic Versioning 2021).

Soll die Anwendung eine Mindestversion erfordern, ist es möglich, Version-Anforderungen im Gradle-Build-File anzugeben. Damit kann sichergestellt werden, dass die Anwendung nur auf Geräten mit einer kompatiblen Version installiert werden kann. Dafür gibt es die „minSdkVersion“, „maxSdkVersion“ und „targetSdkVersion“ Parameter. Ersterer ist die Mindestversion des Android Software-Developement-Kit. Somit können mobile Endgeräte mit niedrigerer Version solche Anwendungen im Google Play Store nicht downloaden. Bei der „maxSdkVersion“ ist es ähnlich, mit der Ausnahme, dass keine Geräte mit höherer Version diese Anwendung nicht installieren können. Die „targetSdkVersion“ stellt dabei im Gegensatz sicher, dass neue Funktionen erst nach Erhöhung dieses Parameters freigeschaltet werden, aber trotzdem von höheren Versionen nutzbar sind. Die Versionsinformationen werden im Kapitel 4.1.6 Kompatibilität näher betrachtet (Android Developers 2021aa, 2021b).

Wurde das Update mit den genannten Schritten vorbereitet, ist es notwendig, einen Release-Plan zu erstellen und zu veröffentlichen. Der Google Play Store bietet hier vier verschiedene Varianten (Google Support 2021d):

- Produktion: Das Release steht für alle Google Play Nutzerinnen und Nutzer zum Download bereit.
- Offene Tests: Die Anwendung ist für Testerinnen und Tester beim Google Play Store verfügbar.
- Geschlossene Tests: Dabei wird die Anwendung für selbst definierte Test-Personen freigeschaltet. Diese haben dann die Möglichkeit, die Anwendung zu testen und Feedback zu geben.
- Interne Tests: Die Verfügbarkeit der Anwendung ist bei einem internen Test auf 100 Personen beschränkt, welche selbst ausgewählt werden dürfen.

Bei der Veröffentlichung kann die Anwendung entweder sofort für Google Play Store Nutzerinnen und Nutzer bereitgestellt werden, sofern diese die Länder und Versionseinschränkungen erfüllen, oder man veröffentlicht die Updates in gestaffelter Form. Hier ist es möglich die Versionsänderungen im ersten Moment nur für einen Teil der Nutzerinnen und Nutzer, welche nach einem Zufallsprinzip ausgewählt werden, zu veröffentlichen. Der prozentuelle Anteil von den bereits heruntergeladenen Anwendungen, welche die Version erhalten sollen, muss händisch verwaltet werden. Somit muss der Prozentsatz in der Google Play Console bis auf 100% selbst aktualisiert werden (Google Support 2021d).

Das Update wird nicht in allen Fällen automatisch heruntergeladen. Der Grund dafür ist, dass die Benutzerin bzw. der Benutzer die Auswahl zwischen manuellen und automatischen Updates hat. Automatische Updates sind im klassischen Fall nur mit der Berechtigung der Benutzerin bzw. des Benutzers (E1) möglich. Sollte es sich jedoch um kritische Sicherheitsupdates handeln, gibt es die „In-app updates“ in der Google Play Store Bibliothek, welche es möglich macht, die Bneutzerin bzw. den Benutzer auf ein Update aufmerksam zu machen. Diese Bibliothek bietet dabei zwei Möglichkeiten. Zum einen ist das der flexible Update-Prozess, welcher die Benutzerin bzw. den Benutzer auf eine neue Version hinweist. Bei sicherheitskritischen Änderungen soll jedoch die Anwendung nicht weiter mit dem aktuellen Stand verwendet werden. Deshalb bietet die Bibliothek die zweite Variante des sofortigen Updates. Dabei ist die Benutzung der Anwendung so lange nicht gestattet bis das Update installiert wurde, um potenzielle Sicherheitsrisiken zu minimieren (Android Developers 2021m; Google Support 2021a).

Apple: Im folgenden Schritt wird der Update-Prozess beim Apple App Store betrachtet. Grundsätzlich wird der ursprüngliche Anwendungsdatensatz auf der iTunes Connect Plattform verwendet, um diesen mit einer neuen Version zu versehen. Die neu erstellte Version wird den Kundinnen bzw. Kunden kostenlos bereitgestellt, sofern die aktuelle Version bereits heruntergeladen bzw. gekauft wurde. Darüber hinaus ist es beim Apple App Store, wie beim Google Play Store, nicht möglich eine frühere Version zu verwenden, wenn einmal das Update durchgeführt wurde (App Store Connect Help 2021c).

Um den Prozess von einer neuen Version bis zur Veröffentlichung zu betrachten, werden im Anschluss die wesentlichsten Schritte des Updatevorganges zusammengefasst. Dabei ist es notwendig, bei iTunes Connect beim Anwendungsdatensatz eine neue Version, mit einem einfachen Klick auf einen Button, hinzuzufügen. Die Meta-Daten von der bereits existierenden Anwendung werden dabei übernommen. Im nächsten Schritt muss die aktualisierte Build-Version wie beim Installationsprozess hochgeladen werden - mit dem wesentlichen Unterschied, dass die Versionsinformationen angepasst werden müssen. Hier gibt es wieder zwei Angaben, welche die aktuelle Version beschreiben (App Store Connect Help 2021c). Zum einen ist das die Versionsnummer, welche eine sichtbare Zeichenfolge für die Benutzerin bzw. den Benutzer ist. Das vorgegebene Format ist eine Zeichenfolge mit drei durch Punkte getrennte numerische Ganzzahlen (<MAJOR>.<MINOR>.<PATCH>.). Dies folgt dem Semantic-Versioning Konzept, wie beim Versionsnamen bei Android Anwendungen (Semantic Versioning 2021). Das System verwendet diesen Schlüssel, um die Version des aktuellen Bundles zu identifizieren. Darüber hinaus wird die Versionsnummer im App Store angezeigt. Des Weiteren gibt es den Build-String, welcher als maschinen-lesbare Zeichenkette konzipiert wurde. Dieser folgt wieder dem Semanting-Versioning Format, mit dem Unterschied, dass fehlende Angaben mit einer 0 aufgefüllt werden, um eine immer gleiche maschinen-lesbare Versionsnummer zu generieren. Dieser wird nur intern im App Store verwendet, um die veröffentlichten und nicht-veröffentlichten Build im System zu erkennen und zu unterscheiden (XCode Help 2021e).

Nachdem die Versionsinformationen angegeben worden sind, kann die Anwendung wie bei der erstmaligen Veröffentlichung bereitgestellt werden. Dabei gibt es die Möglichkeit, diese neue Version mithilfe von Apple App Store angebotene Services direkt zu testen. Zum einen gibt es den Apple „TestFlight“ Service, welcher es möglich macht, kostenlose beta-Tests mit internen

bzw. externen Testerinnen und Tester durchzuführen und Feedback zu erhalten. Somit können Fehler vor der tatsächlichen Veröffentlichung minimiert und Feedback eingearbeitet werden. Es kann dabei zwischen Modellen der internen und externen Tests gewählt werden. Wenn man sich für interne Tests entscheidet, ist es möglich bis zu 100 Personen einzuladen, um die aktuelle Version zu testen. Als interne Testerinnen und Tester kommen dabei App Store Connect Benutzerinnen bzw. Benutzer zur Auswahl, welche die Rollen Admin, App Managerin bzw. Manager, Entwicklerin bzw. Entwickler oder Marketing-Verantwortliche bzw. Verantwortlicher besitzen. Sind diese dazu bereit, das aktive Build zu testen. Dann ist es möglich, mithilfe von „TestFlight“ ein Feedback zu geben. Auf der anderen Seite gibt es externe Test. Hier ist es möglich, externe Personen per öffentlichen Link zum Betatest der Anwendung einzuladen. Die potenziellen Testpersonen sind alles Benutzerinnen bzw. Benutzer mit einem mobilen Apple Endgerät. Dadurch können es auch alle App Store Connect-Benutzerinnen bzw. Benutzer sein, jedoch mit der Einschränkung, dass diese nicht sowohl interne als auch externe Testerinnen bzw. Tester sein können. Bei externen Tests ist es möglich, bis zu 10.000 Personen einzuladen, wie auch das Gruppieren der Testpersonen in Kategorien (App Store Connect Help 2021a, 2021d; Apple Developer 2021f).

Darüber hinaus bietet Apple mit seinem App Store verschiedene Verfahren zur Veröffentlichung einer neuen Version. Grundsätzlich gibt es den Weg des klassischen, manuellen Release und das Update in Phasen. Beim manuellen Release kann die neue Version händisch freigegeben werden, nachdem die aktualisierte Anwendung die App Store Überprüfung bestanden hat. Wenn die Anwendung alle Anforderungen des App Stores erfüllt hat und genehmigt wurde, bekommt diese den Release Status „Pending Developer Release“. Das bedeutet, dass die Anwendung ab diesem Zeitpunkt freigegeben werden kann. Die Veröffentlichung der Anwendung nach der manuellen Freigabe kann bis zu 24 Stunden dauern, bis sie tatsächlich im App Store verfügbar ist. Als Alternative gibt es die Möglichkeit der Updates in Phasen. Anhand dieser Methode kann die Anwendung schrittweise immer mehr Endbenutzerinnen bzw. -benutzern zur Verfügung gestellt werden. Dieser Ansatz folgt dem Prinzip, dass das Versions-Update innerhalb eines Zeitraums von sieben Tagen jeden Tag den Prozentsatz erhöht bis 100% das Update erhalten haben. Diese 100% beziehen sich auf Endbenutzerinnen und -benutzer, welche bei ihren Endgeräten automatische Updates erlaubt haben. Der Update-Prozess läuft nach dem Zufallsprinzip, welches die Benutzerinnen und Benutzer anhand einer aktiven Apple-ID auswählt. Nichtsdestotrotz wird das Update auch für alle anderen Benutzerinnen und Benutzer angezeigt. Das bietet die Möglichkeit, dass jede Endanwenderin bzw. jeder Endanwender die Softwarelösung auch manuell herunterladen kann. Darüber hinaus kann das Update unabhängig von der Anzahl der Pausen über 30 Tage hinweg pausiert werden. Anzumerken ist, dass das Update nicht wieder rückgängig gemacht werden kann. Wenn das Update pausiert wurde, ist es jedoch trotzdem möglich, das Update manuell zu installieren. Dasselbe gilt, wenn das Update pausiert wurde, um währenddessen ein neues Update zu veröffentlichen. In diesem Fall bleibt die momentane Version im App Store verfügbar, bis die aktualisierte Version wieder bereit zur Veröffentlichung ist. Der Prozentsatz der Benutzerinnen bzw. Benutzer mit aktivierten, automatischen Updates wird in der Tabelle 5 gegenüber den Phasen (Tage) visualisiert. (App Store Connect Help 2021e, 2021f)

Tag	Prozent der Benutzerinnen bzw. Benutzer
1	1%
2	2%
3	5%
4	10%
5	20%
6	50%
7	100%

Tabelle 5: Apple App Store Update Phasen (App Store Connect Help 2021e)

Da wie bereits erwähnt automatische Updates nur möglich sind, wenn die Benutzerin bzw. der Benutzer diese erlaubt (E2), kann dieses Qualitätsteilmerkmal nur mit dieser Einschränkung erreicht werden. Des Weiteren gibt es auch keine bereitgestellte Methode, um sicherheitsrelevante Updates am Endgerät zu erzwingen. Es besteht die Möglichkeit, die Benutzerin bzw. den Benutzer programmatisch über ein neues Update zu informieren, jedoch kann hier kein Zwang zur Aktualisierung ausgeübt werden (Stackoverflow 2021).

Feedback: Um in weitere Folge Feedback der Benutzerinnen bzw. Benutzer und Laufzeit-Metriken der Anwendung zu berücksichtigen, ist es notwendig, diese Daten den Entwicklerinnen und Entwicklern zu veranschaulichen. Die darauffolgende Evaluierung wird den Fokus auf die zwei Qualitätsteilmerkmale setzen, welche zuvor in der Kriterien-Definition Phase näher definiert wurden. Diese sind:

- Bewertungen und Rezension: Wie sind zufrieden sind die Benutzerinnen und Benutzer mit der Anwendung?
- App-Performance Metriken: Wie gut performt die Anwendung?

Sollten bei den Technologieansätzen keine speziellen Analysetools von der jeweiligen Plattform vorhanden sein, werden die notwendigen Schritte für die Implementierung eines Drittanbieters betrachtet.

Apple: Im Bereich von nativen IOS Anwendungen bietet der Apple App Store Connect Service eine „Bewertungen und Rezensionen“ Übersicht. Hier ist es möglich, einzelne Bewertungen und Rezensionen, wie auch gruppiert und gefiltert, einzusehen. Ab der IOS Version 11 wird eine zusammengefasste Bewertung aller Benutzerinnen und Benutzer generiert und auf der Produktseite im App Store gezeigt. Grundsätzlich kann eine zusammengefasste Bewertung von der Entwicklerin bzw. dem Entwickler zurückgesetzt werden, jedoch bleiben frühere Kommentare weiterhin bestehen. Möchte man als Entwicklerin bzw. als Entwickler der Anwendung einzelne Bewertungen einsehen, kann man einen Filter anwenden. Dabei ist die Einschränkung nach einer Anwendungsversion, einer speziellen Anzahl an Sternen (1 - 5 Sterne) oder bereits beantworteten oder bearbeiteten Bewertungen möglich. Darüber hinaus können unpassende

Anwendungs-Reviews bei der bzw. dem App Store Betreiberin bzw. Betreiber gemeldet werden, um ein Ansuchen auf eine Löschung zu beantragen. Des Weiteren besteht die Möglichkeit Bewertungen zu beantworten, welche innerhalb von 24 Stunden veröffentlicht werden. Jedoch kann nur eine Antwort per Review im App Store gegeben werden.

Da auch anwendungsspezifische Analysen für eine erfolgreiche Software unumgänglich sind, wird die Vielfalt solcher Auswertungen mit dem Apple App Store geklärt. Für diesen Zwecks bietet App Store Connect mehrere Services. Der „App Analytics“ Service hilft, Daten zu Nutzerinteraktionen und zu Werbekampagnen zusammenzufassen. Laut der Apple Dokumentation sind die Metriken in drei Kategorien eingeteilt (App Store Connect Help 2021g, 2021b):

- **App Store**: Hier werden Metriken zusammengefasst, welche zeigen, wie oft eine Anwendung im App Store aufgerufen bzw. wie oft die Produktseite besucht wurde. Damit lässt sich zum Beispiel die Conversion-Rate berechnen, also wie viele Personen die Produktseite angesehen und wie viele sie dann tatsächlich heruntergeladen haben.
- **Sales**: Des Weiteren fasst dieser Service weitere Verkaufs-Metriken zusammen. Hier werden Messungen von In-App-Käufen, Gesamtzahl der erstmaligen Downloads, Anzahl der eindeutigen Zahlerinnen bzw. Zahler und weitere Informationen im „App Analytics“ Dashboard zusammengefasst.
- **Nutzung**: In dieser Kategorie werden Interaktionsmetriken, wie Installationen, Sitzungen, Abstürze und aktive Geräte, zusammengefasst. Die Messwerte werden jedoch nur an die Entwicklerinnen und Entwickler weitergegeben, wenn die Endbenutzerinnen bzw. Benutzer der Diagnose- und Nutzungsinformationen-Weitergabe zugestimmt haben (E3).

Bei der Analyse der Messwerte ist es möglich, Filterungen, wie nach Regionen, Betriebssystem-Versionen, Geräte etc., anzuwenden, um diese Metriken einzuschränken bzw. gegenüberzustellen (App Store Connect Help 2021g).

Android: Auch seitens des Betriebssystems Android können mithilfe des Google Play Stores Analysen der nativen Anwendung erstellt werden. Laut der offiziellen Play Store Dokumentation bietet die Google Play Console dabei die Möglichkeit Bewertungen (1 – 5 Sterne) und Rezensionen einzusehen. Dabei gibt es mehrere Metriken, wie die Bewertungen angezeigt bzw. zusammengefasst werden.

- **Google Play Bewertung**: Hier sieht man die Bewertung, welche auch die Endnutzerinnen und Endnutzer bei der Google Play Store Produktübersicht angezeigt bekommen. Sie wird auf der Grundlage der letzten Bewertungen berechnet.
- **Durchschnittliche Bewertung über die Lebensdauer**: Hier wird die durchschnittliche Bewertung seit der initialen Veröffentlichung angezeigt.
- **Nutzerin bzw. Nutzer**: Die Anzahl der Nutzerinnen und Nutzer wird hier als eine Metrik aufgelistet.

- **Bewertungen im Vergleich zu ähnlichen Anwendungen**: Dabei wird die Anwendung gegenüber einer definierten Gruppe an ähnlichen Softwarelösungen verglichen. Hier erhält man Metriken, wie die eigene Anwendung zu anderen abschneidet.

Des Weiteren können diese Metriken auf unterschiedliche Schlüssel heruntergebrochen werden. Man kann die Anwendung gegenüber den Dimensionen, der Region, der Sprache, der Versionen, der Gerätemodelle etc. filtern, um einen detaillierteren Einblick zu bekommen. Darüber hinaus können in der Google Play Console Rezensionen der Anwendung verwaltet werden. Hier stellt die App Store Betreiberin bzw. der Betreiber mehrere Features zur Analyse bereit. Zum einen gibt es Highlights an Rezensionen. Hier werden häufig benutzte Begriffe und Schlagwörter mit einer künstlichen Intelligenz zusammengefasst, um die aktuellen Erfahrungen der Anwendung zu veranschaulichen. Des Weiteren ist es möglich, Rezensionen nach speziellen Themen auszuwerten. Hier werden Angaben gegenüber dem Design, der Geschwindigkeit, der Nutzerfreundlichkeit und vieles mehr in einer eigenen Metrik zusammengefasst. Wurden die Rezensionen analysiert, kann die Entwicklerin bzw. der Entwickler darauf antworten. Den Entwicklerinnen und Entwicklern wird dabei nur ein Kommentar pro Rezension gewährt, welches jedoch immer wieder bearbeitet werden kann. Rezensionen, welche als unangemessen angesehen werden, kann man beim Google Play Store auf Löschung beantragen (Google Support 2021e).

Neben der Nutzerzufriedenheit könnten die Performance und Trends der Anwendung interessante Fakten sein. Deshalb bietet der Google Play Store diverse Statistiken, welche sich mit der Anwendung selbst beschäftigen. Hier gibt eine Vielzahl an Key Performance Indikatoren (=KPIs). Dabei ist anzumerken, dass nur die Daten der Benutzerinnen bzw. Benutzer die Grundlage bilden, welche der Nutzung auch zugestimmt haben (E4). Im Wesentlichen bietet Google Play Store hierfür drei Gruppen von Metriken:

- **Statistiken zu Installationen:** In dieser Gruppe werden die Metriken, welche die Installation betreffen, zusammengefasst. Aufgrund der Vielzahl der unterschiedlichen Messwerte werden hier nur einige beispielhaft genannt: gewonnene, verlorene, wiederkehrende bzw. aktive Nutzerinnen und Nutzer, unterschiedliche Geräte-Spezifikationen und vieles mehr.
- **Umsatz**: Wenn man kostenpflichtige Inhalte in der Anwendung bzw. Abo-Modelle anbietet, werden Umsätze generiert, welche in der Google Play Console mit unterschiedlichsten Metriken hinterlegt sind, wie zum Beispiel zeitraumbezogene und nutzerbezogene Umsätze.
- **Abstürze und ANR-Fehler (Application not Responding)**: Hier werden Absturzberichte seitens Google erstellt. Des Weiteren wird protokolliert, wenn die Anwendung zwar nicht abstürzt, aber nicht mehr reagiert und somit geschlossen werden muss. Hier spricht man von einem ANR-Fehler.

Bei der detaillierteren Analyse ist es wieder möglich, die erzeugten Metriken zu filtern und nach dem eigenen Interesse anzupassen, um bestmöglich Schwachstellen zu erkennen und darauf einzugehen (Google Support 2021c).

Progressive Web-Anwendung: Progressive Web-Anwendungen sind dagegen mit einigen Einschränkungen behaftet. Da diese Art des Technologieansatzes nur mit Ausnahme von Trusted Web Activities im Google Play Store veröffentlicht werden kann, ist es notwendig, eine Alternative zu den inkludierten Analyse-Services der IOS und Android App Stores zu finden.

Da das Analyse Tool von Google „Google Analytics“ in ca. 86% aller Webseiten, die solche Tools benutzen, verwendet wird bzw. es von 56,7% aller browser-basierten Anwendungen in Verwendung ist, werden im ersten Schritt die Möglichkeiten mit Progressive Web-Anwendungen betrachtet. Google Analytics ist ein Tracking-Tool für eine Menge von Metriken. Dieser Service bietet dabei zum Beispiel geräte-spezifische, browser-spezifische, länder-spezifische oder kampagnen-spezifische Informationen. Auf die einzelnen Messwerte wird im Detail im Zuge dieser Arbeit nicht eingegangen, da das Augenmerk auf die bereits verfügbaren Funktionalitäten und Services gesetzt wird (W3 Techs 2021).

Da eine Progressive Web-Anwendung mithilfe des Service-Workers Offline verfügbar ist, jedoch Google Analytics eine aktive Netzwerkverbindung benötigt, wurde mit der „Offline Google Analytics“ Service Abhilfe geschaffen. Damit Google Analytics-Anfragen im Offline-Modus nicht verloren gehen, werden diese in der Zwischenzeit in der IndexedDB des Browsers gespeichert. Wenn die Anwendung wieder eine aktive Internetverbindung aufweist, werden die zwischengespeicherten Anfragen an den Google Analytics Service geschickt. Somit ist es auch möglich, Progressive Web-Anwendungen zu messen, sofern die Netzwerkverbindung jemals wieder aktiv ist (Google Developers 2021c).

Da, wie bereits erwähnt, sich das Augenmerk dieser Arbeit auf vorhandene Services und Funktionalitäten richtet, werden diese im Anschluss betrachtet. Da nur Progressive Web-Anwendungen mit Trusted Web Activities auf dem Google Play Store installierbar sind, beruhen solche Anwendungen der eigenen Implementierung, wenn Bewertungen und Rezension erhalten werden sollen. Hier muss es in irgendeiner Weise sichergestellt werden, dass die Benutzerin bzw. der Benutzer aufgefordert wird, die Anwendung zu bewerten. Wird jedoch zusätzlich eine native Android Anwendung anhand von Trusted Web Activities emuliert, dann kann auf die Metriken und Funktionalitäten des Google Play Stores zugegriffen werden. Wie bereits erwähnt, ist diese Art von Veröffentlichung nicht beim Apple App Store gestattet. Aus diesem Grund kann man nur Bewertungen und Rezensionen erhalten, wenn man selbst eine Maßnahme zur Analyse schafft oder im Bereich von Android Geräten die Anwendung im Google Play Store veröffentlicht (E5).

Zur Messung von anwendungsspezifischen Metriken stellt Awwwards und Google in ihrem gemeinsam veröffentlichten E-Book mögliche Ansätze bereit. Dabei werden benutzer-relevante Metriken, wie zum Beispiel Installationen, Conversion Rates bzw. die Performance-Metriken anhand von Progressive Web-Anwendungen vom Autor Martin Schierle im Kapitel 8 diskutiert. Da die Installationsfähigkeit eines der primären Merkmale von Progressive Web-Anwendungen ist, können durch bereitgestellte Events des Service-Workers gewünschte Metriken programmatisch ausgewertet werden.

- Anzahl berechtige Nutzerinnen bzw. Nutzer zur Installation: Diese Messung ist mithilfe des „beforeinstallprompt“ Events möglich. Dieses Event wird ausgelöst, wenn ein potenzielles Endgerät die Anforderungen zur Installation erfüllt.

- Anzahl Klicks auf Installations-Prompt: Da das Installationsprompt programmatisch implementiert werden muss, kann auf den Klick gewartet werden und somit die Anzahl ausgewertet werden.
- Anzahl der Nutzerinnen und Nutzer Installation akzeptiert/abgelehnt: Nachdem das eigens implementierte Prompt die Nutzerin bzw. den Nutzer akzeptiert, erscheint standartmäßig das Chrome-Installations-Prompt. Die ist mit der „userChoice“ Eigenschaft versehen, welche wiederrum für Auswertungen genutzt werden kann, ob die Installation bestätigt oder abgelehnt wurde.
- Anzahl der Nutzerinnen bzw. Nutzer installiert: Wurde die Anwendung in weiterer Folge erfolgreich installiert, wird das „appinstalled“ Event der Anwendung ausgelöst. Somit ist es möglich, die tatsächlichen Installationen auszuwerten.

Um das Verhalten der Nutzerin bzw. des Nutzers besser zu verstehen, kann es eine weitere Anforderung sein, die Anzahl der Nutzerinnen bzw. Nutzer zu messen, welche die Awendung vom Homescreen öffnen. Wie bereits im Kapitel 2.5.4 erwähnt, hat die Manifest-Datei die Eigenschaft „start-url“, welche den Einsprungspunkt der Anwendung darstellt. Aus diesem Grund kann man den Traffic auf dieser URL messen. Nichtsdestotrotz ist es möglich, die Anwendung ohne Installation im Browser aufzurufen. Deshalb ist es notwendig, dynamisch im Quellcode zu erkennen, ob es eine installierte Progressive Web-Anwendung ist. Dafür gibt es die „display“ Eigenschaft der Manifest-Datei, welche den Modus (Vollbild, Standalone, Minimal-UI, Browser) in der sich die Anwendung befindet, beschreibt. Wenn darüber hinaus das Offline und Caching Verhalten der Anwendung von Relevanz sind, kann man die bereitgestellten Browser-Events nutzen, die als Ergebnis den Status der aktuellen Verbindung liefern. Auf diese Weise ist es möglich, zum einen die Anzahl der Offline-Nutzerinnen bzw. Nutzer, die eine Netzwerkverbindung wiederherstellen, zu messen. Zum anderen kann die Zeit im Offline-Modus gemessen werden. Und zwar anhand der Differenz der Zeit zwischen den Offline und Online-Ereignis (Martin Schierle 2021).

Sofern eine gute Caching-Strategie gewählt wurde, kann durch den Service-Worker nach wiederholtem Laden die Anwendung schneller gemacht werden. Um diese Performance zu messen, gibt es das Chrome Browser Tool „Lighthouse“, welche die Performance mit bzw. ohne dem Service-Worker testet und dann einen Score von 0% – 100% ausgibt. Dies ist jedoch nicht auf allen Browserplattformen verfügbar. Um einen tieferen Einblick in die Performance-Metriken zu erhalten, muss ein großer Mehraufwand betrieben werden, wie die Studien im Kapitel des Qualitätsteilmerkmal der Performance zeigen (Abhi Gambhir 2018).

Aufgrund der gewonnenen Erkenntnisse kann man zusammenfassen, dass die Messung von App-Performance Metriken einer expliziten Implementierung bedürfen (E6) und somit nur mit dieser Einschränkung möglich sind. Tabelle 6 fasst die Ergebnisse zusammen.

Qualitätsteilmerkmal	PWA			Native Anwendung	
	Safari	Chrome	Samsung	IOS	Android
Automatische Updates				E2	E1
Sicherheitsupdates				E2	E1
Update von Endbenutzer/in steuerbar					
Updates in Phasen veröffentlichen					
Bewertungen und Rezensionen	E5	E5	E5		
App-Performance Metriken	E6	E6	E6	E3	E4

Tabelle 6: Gegenüberstellung der Wartbarkeit von PWAs und nativen Anwendungen

4.1.5 Sicherheit

Im Anschluss wird das Qualitätsmerkmal (factor) der Sicherheit betrachtet. Ziel ist es, die Nutzerinnen bzw. Nutzer und die Anwendung von Schadsoftware und böswilligen Handlungen zu schützen. Um dies zu gewährleisten, werden im Anschluss Sicherheitsprüfungen der Installationsdatei und die Möglichkeit der Multi-Faktor Authentifizierung betrachtet (ISO/IEC 25010:2011).

Sicherheitsprüfung der Installationsdatei: Im ersten Schritt wird die Sicherheitsprüfung der Technologieansätze gegenüber Schadsoftware belichtet.

Android: Wie bereits beim Installations-Prozess in Kapitel 4.1.3 beschrieben, wird von Google eine Überprüfung gegenüber den Anwendungen gemacht, um Schadsoftware nicht zu veröffentlichen. Dafür hat Google einen Dienst mit dem Namen „Bouncer“ entwickelt. Dieser ermöglicht es, die Anwendungen automatisch zu überprüfen, ohne dass die Entwicklerin bzw. der Entwickler einen Genehmigungsprozess durchläuft. Dieser Dienst führt eine Reihe an Analysen durch. Es werden Anwendungen geprüft, welche neu oder bereits vorhanden sind, zum Beispiel nach einem Update. Sobald eine Anwendung hochgeladen ist, beginnt der Dienst sofort mit der Analyse auf bekannte Malware, Spyware und Trojaner. Er sucht auch nach Verhaltensweisen, die darauf hindeuten, dass sich eine Anwendung möglicherweise falsch verhält und vergleicht sie mit zuvor analysierten Anwendungen, um mögliche Warnsignale zu erkennen (Ge Gao 2016).

Dass jedoch nicht alle Schadsoftwares zu 100% herausgefiltert werden können, hat eine Studie von Telemetriedaten von Platon et al. (2020) ergeben. Es wurden rund 12 Millionen Android-Geräte zwischen Juni und September 2019 betrachtet. Basierend auf diesen Daten kam es zu 34 Millionen Installationen, wobei es sich um rund 7,9 Millionen unterschiedliche Anwendungen handelte. Im Gesamten wurden dabei zwischen 10 bis 24% Softwarelösungen als bösartige bzw. unerwünscht eingeschätzt. Davon wurden 67% vom Google Play Store heruntergeladen. Ergänzend verglich diese Studie auch die ungewollten, installierten Anwendungen mit allen installierten Anwendungen der Plattformen. Dabei kam beim Google Play Store nur eine 0,6% Wahrscheinlichkeit heraus (Platon et al. 2020).

IOS: Beim Unternehmen Apple sind wesentliche Unterschiede erkennbar. Wie bereits erwähnt, lässt Apple kein Sideloading in seinen Anwendungen zu, was bedeutet, dass dieses System isoliert ist und nur Anwendungen vom offiziellen Apple App Store downloadbar sind. Apple hat im Oktober 2021 ein Paper „Building a Trusted Ecosystem for Millions of Apps“ publiziert, das seine Anhaltspunkte für ein isoliertes System beschreibt. Im Zuge dessen wurde eine Studie (Threat Intelligence Report) erwähnt, welche von Nokia initiiert wurde. Dabei konnte Nokia die Ergebnisse präsentieren, dass Android Geräte 15 bis 47-mal häufiger Schadsoftware vorweisen gegenüber Apple Geräte. Aufgrund dem sehr eingeschränkten Ecosystem hat das Unternehmen die Möglichkeit alle Anwendungen, welche bereitgestellt werden, zu kontrollieren. Apple macht dabei bei jeder Veröffentlichung und jedem Update einen automatischen Malwarescan, wie auch manuelle Kontrollen. Beim sogenannten „App-Vetting“ wird die Anwendung von Apple Mitarbeiterinnen und Mitarbeitern überprüft, ob alle Zugriffe und Features rechtmäßig sind. Aber auch der Apple Play Store schafft es nicht, 100% aller Schadsoftware zu identifizieren. In der Literatur lassen sich jedoch keine genauen Zahlen des Unternehmens finden. Nichtsdestotrotz gibt es bekannte Fälle, wo IOS Anwendungen mit Schadsoftware downloadbar waren. Das Unternehmen Wandera Thread Labs konnte 17 IOS-App entdecken, welche mit Schadsoftware versehen waren. In all diesen Fällen handelte es sich um Klick-Trojaner, welche im Hintergrund Webseiten öffneten und Klicks auf Onlineanzeigen simulierten, ohne eine persönliche Interaktion der Endbenutzerin bzw. des Endbenutzers. Somit konnten große Summen an Umsätzen generiert werden (Apple 2021; wandera 2021)

Progressive Web-Anwendungen: Bei Progressive Web-Anwendungen gibt es gänzlich keine Sicherheitsprüfungen einer zweiten Instanz, bei einer Installation aus dem Browser. Wird dabei jedoch eine Progressive Web-Anwendung, wie in Kapitel 4.1.3, im Google Play Store anhand von „Trusted Web Activities“ hochgeladen, dann gelten dieselben Sicherheitsprüfungen wie bei nativen Android Anwendungen. Da der Service-Worker als Netzwerk-Proxy zwischen dem Frontend und Backend agiert, ist es eine Schwachstelle der Anwendung und bietet die Möglichkeit für „man-in-the-middle“ Angriffe. Dabei kann sich die Angreiferin bzw. der Angreifer in die aktive Verbindung zwischen Backend und Frontend hängen und die Antworten auf böswillige Weise modifizieren. Um diese Gefährdung maximal einzuschränken, ist es nur möglich, Progressive Web-Anwendungen per HTTPS bereit zu stellen. Somit kann eingeschränkt werden, dass empfangene Service-Worker manipuliert werden. Dennoch gibt es Wege, Angriffsvektoren in die Anwendung einzuschleusen, welche im Anschluss erläutert werden.

Grundsätzlich gelten alle Angriffsarten, welche ein Browser mit sich bringt, als relevant. Jedoch werden im Anschluss die in der Literatur am häufigsten mit Progressive Web-Anwendungen verbundenen Angriffsquellen genannt. Zu erwähnen ist, dass die potenziellen Angriffsquellen nur in ihrem Dasein beschrieben werden, um Bewusstsein dafür zu schaffen und nicht in der Bekämpfung anhand eines Sicherheitskonzeptes. Zum einen bieten Progressive Web-Anwendungen Raum für Cross-Site-Scripting (XSS) Angriffe als eine der Hauptangriffsarten. Dabei werden Sicherheitslücken ausgenutzt, um bösartige Inhalte in den Kontext der Anwendung zu laden, welche als vertrauenswürdiger Kontext eingestuft sind. Gibt es zum Beispiel die Möglichkeit, Daten hochzuladen, kann dies ausgenutzt werden und es kann eine manipulierte Javascript-Datei in das System eingeschleust werden. Diese Datei verschafft sich damit den

vollen Zugriff auf Objekte und Daten der Anwendung, die aus der Benutzersession wie auch aus dem lokalen Speicher ausgelesen werden können. Diese Daten können dann an einen dafür konzipierten Server geschickt werden. Eine weitere bekannte Angriffsquelle ist der sogenannte „Dormant-Service-Worker“ Angriff. Dabei wird unwissentlich ein Service-Worker auf localhost mit einem bestimmten Port installiert. Dies kann gelingen, wenn man zum Beispiel durch Social-Engineering Zugriff auf den Rechner des Opfers bekommt. Dabei wird ein bösartiges Skript auf dessen Computer installiert. Im Zuge dessen läuft der Service-Worker auf localhost im Leerlauf. Führt die Benutzerin bzw. der Benutzer anschließend eine andere Anwendung auf localhost mit demselben Port aus, dann wird dieser Dienst aktiv und ermöglicht der Angreiferin bzw. dem Angreifer einen „man-in-middle-attack“ (NVISO Lab 2021).

Somit kann bei Progressive Web-Anwendungen zusammengefasst werden, dass es keine weitere Instanz zur Überprüfung der Sicherheit gibt. Hier trägt die Entwicklerin bzw. der Entwickler die volle Verantwortung, die Sicherheit zu gewährleisten.

Multi-Faktor Authentifizierung: Mit dem Sicherheitskonzept der Multi-Faktor Authentifizierung wird überprüft, dass mindestens zwei von drei Beweisstücke mit dem mobilen Endgerät vorhanden sind. Wie bereits in der Kriterien-Definitionsphase beschrieben, gibt es drei Kategorien (Wissen, Eigenschaft und Besitz) zur Identifikation, wobei der Fokus auf den biometrischen Daten gelegt wird. Es ist notwendig, Zugriffe auf den Fingerabdruck- bzw. Gesichtserkennungssensor zu erhalten. Wie in der Evaluierung des Qualitätsmerkmals der Funktionalität beschrieben, wird der Zugriff dieser Sensoren zum Zeitpunkt der Verfassung der Arbeit im Bereich der Browsertechnologien nur von Chrome und Safari Browser unterstützt. Der Samsung Internet Browser gewährt diesen Zugriff nicht. Im Gegensatz dazu kann von beiden nativen Plattformen auf die Sensoren zugegriffen werden und für eine Multi-Faktor Authentifizierung genutzt werden (MDN Web Docs 2021u; Apple Developer 2021d; Android Developers 2021z).

Tabelle 7 stellt die oben beschriebenen Informationen gegenüber.

Qualitätsteilmerkmal	PWA			Native App	
	Safari	Chrome	Samsung	IOS	Android
Sicherheitsprüfung der Installationsdatei					
Multi-Faktor Authentifizierung					

Tabelle 7: Gegenüberstellung der Sicherheit von PWAs und nativen Anwendungen

4.1.6 Kompatibilität

Im nächsten Schritt wird die Software-Kompatibilität als Qualitätsmerkmal der ISO 25010 näher betrachtet. Im Zuge der Evaluierung sollen die Plattformunabhängigkeit, die Auf- und Abwärtskompatibilitäten der verschiedenen Entwicklungsansätze und die Möglichkeiten zur Veröffentlichung verschiedener Versionen geklärt werden. Ziel ist es, einen Überblick zu geben, wie und mit welchen Einschränkungen eine Software in Bezug auf diese Kriterien entwickelt und möglichst effizient für unterschiedliche Geräte und Plattformen angeboten werden kann.

Plattformunabhängigkeit: Die Evaluierung startet mit dem Qualitätsteilmerkmal der Plattformunabhängigkeit. Im Zuge dessen wird evaluiert, inwiefern es möglich ist, einen Quellcode bzw. die resultierende Installationsdatei über mehrere Plattformen hinweg zu nutzen.

Android & iOS: Bei IOS und Android Anwendungen kann die Plattformunabhängigkeit nicht erreicht werden. Der wesentlichste Grund dafür ist, dass die beiden Plattformen unterschiedliche Installationsdatei-Formate verwenden, welche einer plattformspezifischen Behandlung bedürfen. Android verwendet hier das Android-Package (.apk) und Apple das IOS App Store Package (.ipa) Dateiformat. Aus diesem Grund sind diese Installationsdateien nur auf der jeweiligen Plattform verwendbar (fileformat 2021; Droid Wiki 2021).

Progressive Web-Anwendungen: Da Progressive Web-Anwendungen auf Basis von Webtechnologien aufgebaut und im Browser lauffähig sind, ist es gegenüber nativen Varianten möglich, mit einer Code-Basis beide Plattformen IOS und Android zu bedienen (MDN Web Docs 2021m). Wie man dem Kapitel 4.1.1 entnehmen kann, kommt es jedoch bei gewissen Hardwarezugriffen zu immensen Einschränkungen, vor allem im Bereich von IOS Geräten (E1). Somit ist es zwar prinzipiell aufgrund des durchgängigen Service-Worker-Browsersupports möglich, Progressive Web-Anwendungen mit jeglichen Browsern zu nutzen (Can I use 2021c), jedoch muss mit Einschränkungen gerechnet werden. Es ist daher nicht in allen Fällen möglich, eine gleichwertige, plattformübergreifende Softwarelösung anzubieten.

Sicherstellung der Auf- und Abwärtskompatibilität: Des Weiteren werden die Möglichkeiten und Einschränkungen zur Sicherstellung der Auf- und Abwärtskompatibilität betrachtet. Wie bereits in der Kriterien-Definitionsphase erläutert, ist es ein wesentlicher Faktor, die installierten Versionen der Nutzerinnen-/NutzerEndgeräte zu berücksichtigen, um stets die Kompatibilität zu gewährleisten.

Android: Wie bereits beim Qualitätsmerkmal der Wartbarkeit erwähnt, ist es anhand von verschiedenen Parametern bei nativen Android-Anwendungen möglich, die Auf- und Abwärtskompatibilität in verschiedenen Versionen sicherzustellen. Der Google Play Store verwendet dabei das in der Manifest-Datei deklarierte <uses-sdk> Attribut, um Endgeräte zu filtern, welche die Plattform-Version nicht erfüllen können. Zur Erreichung der Kompatibilität wird die API-Version des Endgerätes und der Anwendung verglichen. Dafür gibt es drei Parameter der Manifest-Datei, welche in der Abbildung 29 dargestellt und im Anschluss erläutert werden (Android Developers 2021b).

```
<uses-sdk android:minSdkVersion="integer"
          android:targetSdkVersion="integer"
          android:maxSdkVersion="integer" />
```

Abbildung 29: <uses-sdk> Android Manifest-Datei (Android Developers 2021b)

Die „minSdkVersion“ ist die minimale API-Version, welche für die Nutzung der nativen Anwendung erforderlich ist. Damit verhindert der Google Play Store, dass die Nutzerin bzw. der Nutzer eine Anwendung installiert, wenn diese SDK Version niedriger ist als der angegebene Parameter. Wird dieser Parameter nicht konfiguriert, wird von System immer der Standardwert 1

angenommen, somit ist die Anwendung mit allen Android Geräten kompatibel. Sollte die Anwendung trotzdem nicht mit allen API-Versionen kompatibel sein, dann würde es zu einem Fehler kommen. Aus diesem Grund wird in der Android Dokumentation empfohlen, diese Version immer bereit zu stellen, um die Abwärtskompatibilität zu gewährleisten (Android Developers 2021b).

Die „targetSdkVersion“ ist die API-Version, auf welche die Anwendung abzielt. Wenn dieser Parameter nicht angegeben wird, wird die „minSdkVersion“ herangezogen. Dieses Attribut informiert das Android System dabei darüber, dass die Anwendung mit dieser Version getestet wurde und somit die Kompatibilität zu höheren API-Versionen des Gerätes nicht sichergestellt werden kann. Da die Android Version kontinuierlich erhöht wird und neue Features implementiert bzw. alte verbessert werden, stellt das Android Betriebssystem nur Features bereit, welche vor der „targetSdkVersion“ veröffentlicht wurden, um so Fehler zu vermeiden. Um die Nutzerinnen bzw. Nutzer stets mit allen aktuellen Features auszustatten, verfolgt man das Ziel, diesen Parameter immer auf der höchsten API-Version konfiguriert bzw. getestet zu haben. Somit können die Mittel zur Sicherstellung der Vorwärts-Kompatibilität bereitgestellt werden (Android Developers 2021b).

Der „maxSdkVersion“ Parameter gibt im Gegenzug die maximale API-Version an, auf welcher die Anwendung lauffähig sein kann. Somit wird eine Anwendung nicht installiert, wenn ein Endgerät bereits eine höhere API-Version als das „maxSdkVersion“ Attribut verfügbar hat. Der wesentliche Unterschied zur „targetSdkVersion“ ist, dass die Möglichkeit der Installation gänzlich nicht gestattet ist und nicht nur Features unterdrückt werden. Die Validierung auf die maximale API-Version wird dabei bei der Neuinstallation und bei Updates durchgeführt. Sollte eine Anwendung bereits installiert sein und die maximale API-Version ist kleiner als die Geräte-Version, dann wird die Anwendung entfernt. Nichtsdestotrotz wird die Deklaration dieses Attributs von Google nicht empfohlen, da erstens die neuen Versionen blockiert werden und zweitens bereits installierte Versionen der Benutzerinnen und Benutzer entfernt werden, wenn die maximale API-Version der Anwendung niedriger ist als die vom Endgerät (Android Developers 2021b).

iOS: Der Apple App Store bietet hingegen die Möglichkeit, die zwei Parameter „Deployment-Target“ and „Base SDK“ anzugeben. Das „Deployment-Target“ ist die niedrigste Version, welche von der Anwendung unterstützt wird. Der App Store ist dabei so konzipiert, dass Geräte mit älteren Versionen die Anwendung nicht herunterladen bzw. bei bereits installierten Anwendungen keine neuen Updates installiert werden können. Stellen die Entwicklerinnen und Entwickler die Unterstützung einer älteren Version ein, bedeutet das nicht, dass bereits installierte Anwendungen vom Endgerät gelöscht werden, sondern nur keine Updates ausgeliefert werden. Die Entwicklungs-IDE Xcode bietet hierfür in den Deployment-Einstellungen die Möglichkeit, die minimale Version anzugeben, welche das „Deployment-Target“ widerspiegelt. Wenn die SDK Version, welche zur Entwicklung der Anwendung genutzt wird, höher ist als das „Deployment-Target“, wird dabei eine Warnung von Xcode ausgegeben. Dadurch wird verhindert, dass Funktionen in einer neuen Version genutzt werden, welche vom mobilen Gerät nicht unterstützt werden. Seitens Apple gibt es keine Möglichkeit, eine Maximum-Version festzulegen, welche neue Versionen einschränkt. Um diese Lücke jedoch zu schließen, gibt es den „Base SDK“ Parameter. Im Anschluss wird dieses Attribut in Verbindung mit dem „Deployment-Target“

betrachtet. Das Zusammenspiel dieser Versionseigenschaften wird in der Abbildung 30 dargestellt (XCode Help 2021c, 2021d).

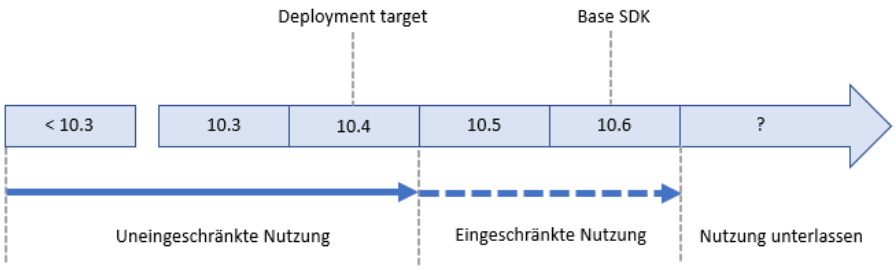

Abbildung 30: SDK Developement Timeline (Eigene Darstellung angelehnt an (Apple Developer 2021u))

Wie bereits erwähnt, stellt die „Deployment-Target“ Version die minimale SDK Version bereit, welche es erlauben kann, uneingeschränkt Funktionen einschließlich dieser Version zu nutzen. Des Weiteren ist es möglich die „Base SDK“ Version zu wählen. Diese setzt die Obergrenze der Nutzung von Software-Features. Soll dabei eine Feature, welches zwischen dem „Deployment-Target“ und der „Base SDK„ Version veröffentlich wurde, genutzt werden, dann muss eine Prüfung des Vorhandenseins programmatisch durchgeführt werden. Die „Base SDK“ Version wird auf die letzte verfügbare Version gesetzt, wenn kein Parameter angegeben wurde. Was wiederum bedeutet, wenn dieses Attribut nicht angegeben wird, dass die Nutzung aller Features über dem „Deployment-Target“ überprüft werden müssen. Wird trotzdem ein Feature genutzt, welches nicht verfügbar ist, kommt es zu einer „Compile-Time Error“ Exception (Apple Developer 2021u).

Aufgrund der genannten Tatsachen kann die Auf- und Abwärtskompatibilität bei nativen iOS Anwendungen sichergestellt werden.

Progressive Web-Anwendungen: Im Anschluss wird die Kompatibilität von Progressive Web-Anwendungen betrachtet. Wie bereits erwähnt wird eine Anwendung dieses Technologieansatzes nicht von einem offiziellen App Store bereitgestellt und unterliegt somit keiner Kompatibilitätsprüfung dieser Hosting-Anbieterinnen/Anbieter. Des Weiteren wurde das Thema „Trusted Web Activities“ betrachtet, welches es möglich macht, eine Progressive Web-Anwendung auf den Google Play Store zu veröffentlichen. Wie in der Evaluierungsphase der Auf- und Abwärtskompatibilitäten der nativen Android-Anwendungen erläutert, kann die SDK Version der Anwendungen mit den Versionen des Endgeräts kontrolliert werden. Da jedoch eine Progressive Web-Anwendung nur einen nativen Wrapper mithilfe der „Trusted Web Activities“ Technologie bekommt, im Kern eine Webseite bleibt und einem Chrome Browser läuft, bringt es nichts, die Kompatibilitätsprüfungen seitens der installierten Betriebssystem-Version zu machen. Ausschlaggebend ist die verwendete Chrome Version. Da auch hier keine Kompatibilitätsprüfung im Google Play Store konfiguriert werden kann, muss die Entwicklerin bzw. der Entwickler selbst einen Weg finden, dies zu überprüfen. Aufgrund der Tatsache, dass Progressive Web-Anwendungen nicht auf eine Plattform beschränkt sind und mehrere Laufzeitumgebungen vorweisen, gestaltet sich die Sicherstellung der Kompatibilität etwas schwieriger. Wie bei Evaluierung des Qualitätsmerkmales der Funktionalität veranschaulicht, bietet jeder Browser

unterschiedliche Zugriffsfreiheiten und somit auch Kompatibilitäten. Darüber hinaus sind die Zugriffsfreiheiten zusätzlich in ihrem Bestehen von der Browserversion abhängig. Somit ist zum Beispiel die Bluetooth Web-API beim Chrome Browser seit der Version 56 verfügbar, beim Samsung Internet Browser seit der Version 6.0 und beim Safari Browser gänzlich nicht gestattet. Aus diesem Grund müssen die Funktionen des jeweiligen Browsers mit der bestehenden Version geprüft und einzeln verwaltet werden, um die Anwendung ohne Fehlerquellen zu veröffentlichen. Dabei müssen programmatisch die Geräte-Spezifikation ausgelesen werden. Wie man diese Spezifikationen unterscheidet und daraus mehrere Versionen für abweichende Freiheiten generiert, wird im nächsten Schritt beim Qualitätsteilmerkmal „Verschiedene Versionen veröffentlichen“ im Detail betrachtet. Als Fazit kann gesagt werden, dass die Auf- und Abwärtskompatibilität nur mit der Einschränkung (E2) der eigenen Implementierung anwendbar ist (Chrome Developer 2021; MDN Web Docs 2021v).

Verschiedene Versionen veröffentlichen: Im Anschluss werden die Möglichkeiten und Schritte betrachtet, um eine Softwarelösung anhand verschiedener Versionen bereitzustellen.

Android: Im ersten Schritt werden dabei die nativen Android-Anwendungen evaluiert. Wie man der Android Developer Dokumentation entnehmen kann, gibt es einen „Multiple APK“ Support. Zur Veröffentlichung von unterschiedlichen Versionen bietet der Google Play Store mehrere Ansätze, um auf variierende Geräte-Spezifikationen abzuzielen. Dieser Service verfolgt das Konzept, dass nur eine Anwendung im App Store aufscheint, aber die verschiedenen Geräte unterschiedliche Installationsdateien herunterladen. Somit müssen nur einmal die Produkt-Details gewartet werden, die Nutzerinnen bzw. Nutzer sind nicht verwirrt wegen mehreren Versionen und alle Bewertungen und Rezensionen laufen auf demselben Anwendungs-Listing. Des Weiteren wird ein Update seitens Google Play veranlasst, wenn sich die Nutzerin bzw. der Nutzer durch ein System-Update für eine neue Version qualifiziert. Um zu konfigurieren, welches Gerät die jeweiligen APK Versionen erhält, wird durch die bereitgestellten Google Play-Filter bestimmt. Die Veröffentlichung mehrerer Versionen ist nur möglich, wenn mindestens ein Filter angewandt wurde (E3). Die APK-Filter, welche im Anschluss einzeln betrachtet werden, sind Teil der Manifest-Datei (Android Developers 2021q).

- **Bildschirmgröße:** Dabei ist es möglich, anhand von zwei Manifest-Parametern die Bildschirmgröße auf die verschiedenen Versionen anzupassen. Es gibt das <supports-screens> oder das <compatible-screens> Element. Laut Google wird dies aber nicht empfohlen, sondern es soll eine Anwendung geschaffen, welche auf die verschiedenen Bildschirmgrößen skalierbar ist (Android Developers 2021q).
- **OpenGL-Texturkomprimierungsformate:** Bei der Implementierung von Spiele-Anwendungen kann es von Relevanz sein, verschiedene OpenGL Feature Versionen zu verwenden. Um diese Programmierschnittstelle für 2D und 3D Grafikanwendungen zu konfigurieren, wird dies mit dem <supports-gl-texture> realisiert (Android Developers 2021q).
- **Geräte-Features:** Dieser Filter basiert auf dem <uses-feature> Element der Manifest-Datei. Dieser Parameter gibt die notwendigen Hardware-Zugriffe der Versionen an, um eine Trennung für Geräte zu bilden, welche gewisse Zugriffe aufgrund der fehlerhaften

Komponenten nicht ausführen können und somit eine eigene Version bedürfen. Dabei können die gewünschten Funktionen wie Bluetooth, NFC, biometrische Sensoren und viele weitere angeführt werden (Android Developers 2021q).

- **CPU architecture:** Unterschiedliche CPU-Architekturen erfordern in vielen Fällen bestimmte Pakete von nativen Bibliotheken. Anstatt alle Bibliotheken gesammelt in eine Installationsdatei zu packen, kann jeweils eine separate Version erstellt werden (Android Developers 2021q).

Des Weiteren gibt es zusätzliche Vorschriften seitens des Google Play Store, welche befolgt werden müssen. Es muss zum einen gewährleistet werden, dass alle Versionen den bereits existierenden Package-Namen vorweisen können und mit denselben Zertifizierungsschlüssel signiert werden. Des Weiteren muss der Version-Code der Manifest-Datei unterschiedlich sein, bzw. eine Version mit einer höheren API-Version muss den höheren Version-Code aufweisen. Wenn man sich in weiterer Folge dazu entschieden hat, mehrere Versionen zu veröffentlichen, dann muss ein eigenes APK-Projekt für jede Version erstellt werden. Dabei ist es jedoch möglich, das bestehende Projekt zu duplizieren und nur mit den neuen Version-Informationen zu bereichern. Sollte im Anschluss beim Download vom Google Play Store keine richtige Version identifizierbar sein, kann es vom Vorteil sein, eine universell-kompatible Version zu erstellen. Wird der Anwendung dabei die niedrigst-getestete Version-Nummer zugewiesen, kann die Chance auf Inkompatibilitäten mit dem Zielgerät minimiert werden (Android Developers 2021q).

IOS: Da für native IOS Anwendungen nur der unternehmenseigene Apple App Store zur Verfügung steht, wird dieser im Anschluss in Bezug auf die Veröffentlichung von mehreren Versionen betrachtet. Dieser App Store ist jedoch etwas stäker eingeschränkt. Das Kapitel 4.3 Spam der App Store Review Guideline besagt Folgendes (Apple Developer 2021b):

> *„Don't create multiple Bundle IDs of the same app. If your app has different versions for specific locations, sports teams, universities, etc., consider submitting a single app and provide the variations using in-app purchase."* - (Apple Developer 2021b)

Somit wird eine Anwendung, welche sich nicht von anderen Versionen abhebt, nicht von der bzw. dem App Store Betreiberin bzw. Betreiber zur Veröffentlichung freigegeben.

Da Apple verschiedenste Modelle von mobilen Endgeräten mit unterschiedlichen CPU-Architekturen, Bildschirmauflösungen etc. auf den Markt gebracht hat, hat dieses Unternehmen das Konzept des „App Thinnig" entwickelt. Dieses Feature bietet die Möglichkeit, eine optimierte Anwendungs-Binary zu veröffentlichen, welches das Ziel hat, die gewünschten Architektur-Varianten bestmöglich zu unterstützt. Anstatt dabei nicht jedes Mal die universelle Installationsdatei für alle Geräte bereit zu stellen, liefert der „App Thinning" Dienst nur die notwendigen Features für das Gerät aus. Hierbei gibt es drei Begriffe, welche einer Erklärung bedürfen (XCode Help 2021f):

Slicing: Slicing wird dabei der Prozess genannt, welcher unterschiedliche Varianten von App-Bundles für die variierenden Endgeräte erstellt. Wie man der Abbildung 31 entnehmen kann, werden dabei im App Store mehrere Versionen bereitgestellt, welche nur die ausführbare

Architektur und Ressourcen, die vom Zielgerät benötigt werden, beinhaltet. Nichtsdestotrotz muss von der Entwicklerin bzw. vom Entwickler eine Vollversion der Anwendung im App Store Connect hochgeladen werden. Möchte die Anwendung in einem weiteren Schritt von App Store heruntergeladen werden, wird von dieser nur eine spezifische Installationsdatei-Variante ausgeliefert, welche nur die essenziellen Features für die vorhandenen Geräte-Spezifikationen enthält (XCode Help 2021f).

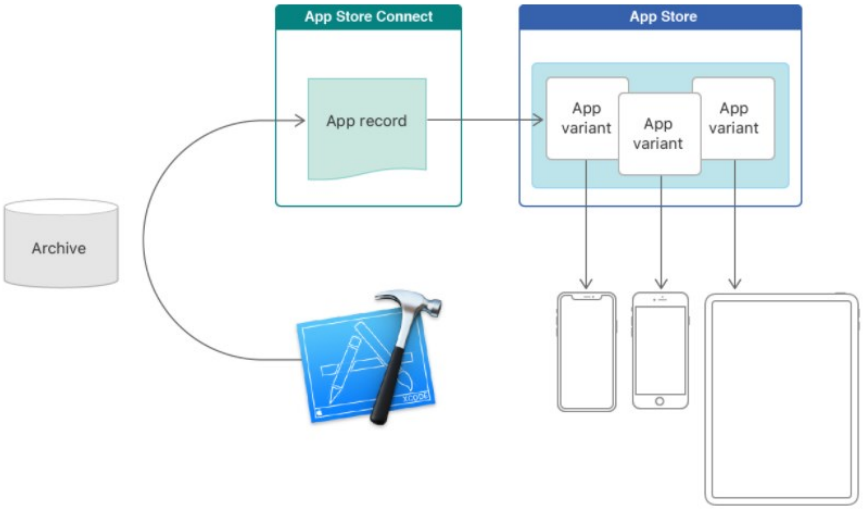

Abbildung 31: App Thinning Slicing Prozess (XCode Help 2021f)

Bitcode: Der Bitcode ist eine Zwischen-Darstellung des kompilierten Programms. Um die Funktionalität von „App Thinning“ zu nutzen, wird dieser Bitcode auf dem App Store Connect hochgeladen, welcher die notwendigen Informationen, um eine mobile Anwendung zu kompilieren, enthält. Erst wenn diese Binärdatei in den App Store hochgeladen wird, wird diese in Maschinencode kompiliert. Somit kann der App Store in weiterer Folge unterschiedliche Versionen anhand der unterschiedlichen Geräte-Spezifikationen kompilieren (XCode Help 2021f).

Asset-Katalog: Um die unterschiedlichen Ressourcen, Bildauflösungen etc. für die Geräte-Spezifikationen bereit zu stellen, gibt es den Asset-Katalog. Dieser bildet die Basis, um die unterschiedlichen Versionen zu konfigurieren und bereitzustellen (XCode Help 2021a, 2021b).

Zusammenfassend lässt sich sagen, dass es nicht möglich ist, ohne Vorschriften mehrere Versionen einer Anwendung auf den Apple App Store zu veröffentlichen. Hierfür muss das Apple „App Thinning“ Feature genutzt werden (E4), welches die Anwendung anhand definierter Geräte-Spezifikationen ausliefern kann.

Progressive Web-Anwendungen: Im nächsten Schritt wird das Veröffentlichen verschiedener Versionen anhand von Progressive Web-Anwendungen untersucht. Wie bereits im Kapitel des Qualitätsmerkmal der Portierbarkeit erwähnt, ist eine Progressive Web-Anwendung nicht an einen App Store oder eine Plattform zur Veröffentlichung gebunden, da diese in der Basis eine Webseite darstellen. Des Weiteren konnte in diesem Teil der Arbeit festgestellt werden, dass Progressive Web-Anwendungen wie eine klassische Webseite auf einen Web-Server oder mit dem „Trusted Web Acitivities“ Ansatz auf den Google Play Store veröffentlicht werden können.

Wird die letztgenannte Variante realisiert, kann auf die Funktionen vom Google Play Store zurückgegriffen werden und mehrere Versionen, wie erläutert, veröffentlicht werden.

Kümmert man sich jedoch selbst um die Veröffentlichung auf einen Web-Server und möchte mehrere Versionen für unterschiedliche Nutzerinnen bzw. Nutzer oder Geräte bereitstellen, dann ist es notwendig, dies selbst programmatisch zu realisieren. In Abbildung 32 wird der Workflow schematisch dargestellt. Dabei werden je nach Geräte-Spezifikationen über den Browser unterschiedliche Versionen vom Web-Server ausgeliefert. Um jedoch verschiedene Versionen für die jeweiligen Endgeräte anzubieten, ist es notwendig, die Endgeräte mit deren Eigenschaften zu kennen. Jedoch ist zu erwähnen, dass es nicht Teil der Arbeit sein wird, eine passende Softwarearchitektur und Veröffentlichungsstrategie zu finden.

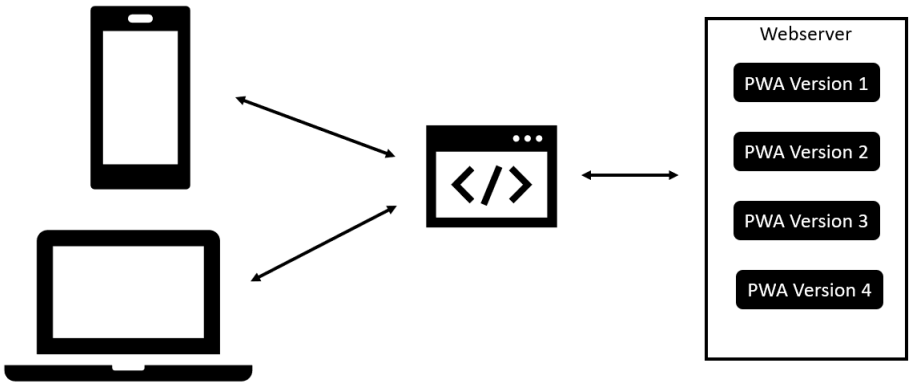

Abbildung 32: Schematischer Workflow den Abruf unterschiedlicher PWA-Versionen (Eigene Darstellung)

Nichtsdestotrotz betrachtet die vorliegende Arbeit die Möglichkeiten, wie Nutzerinnen bzw. Nutzer mit unterschiedlichen Geräte-Spezifikationen unterscheidbar gemacht werden können und dadurch eine eigene Version erfordern. Um dieser Problemstellung gerecht zu werden, ist es notwendig anhand verschiedenster Web-Bibliotheken Geräte-Spezifikationen abzufragen. Im Anschluss werden bereits vorhandene Schnittstellen des Browsers betrachtet, welche diese Eigenschaften für die dafür konzipierte Version liefern. Prinzipiell kommen hier speziell zwei JavaScript-Browserobjekte zum Einsatz, welche im Anschluss erläutert und in Verbindung mit unterschiedlichen Geräte-Spezifikationen gebracht werden:

Window Objekt: Das Window Objekt wird von allen Browsern unterstützt, welches die gesamten JavaScript Objekte, Funktionen und Variablen beinhaltet. Somit können hiermit Eigenschaften des Browser-Fensters abgerufen werden (MDN Web Docs 2021w).

Navigator Objekt: Darüber hinaus gibt es das Navigator Objekt, welches Informationen des physischen Endgerätes mit der vorhandenen Software und Hardware liefert (MDN Web Docs 2021p).

Um diese Browser Objekte auf Metriken herunterzubrechen, werden im Anschluss die Möglichkeiten zur Identifikation der Geräteinformationen betrachtet.

Bildschirmgröße: Soll es möglich sein, verschiedene Versionen der Progressive Web-Anwendung anhand ihrer Bildschirmgröße bereit zu stellen, kann auf das Window Objekt zurückgegriffen werden. Speziell werden hier verschiedenste Metriken des Browser- und

Gerätefensters geliefert. „window.screen.width“ würde dabei die Breite des Browserfenster und „window.innerWidth“ die Breite des gesamten Gerätefensters liefern. Um dabei jedoch nicht bei jeder abweichenden Gerätespezifikationen eine eigene Version zu erstellen, kann die Anwendungen dem Responsive-Design Ansatz von Progressive Web-Anwendungen folgen. Hierfür könnten zum Beispiel CSS-Media-Queries Abhilfe schaffen. Damit ist es möglich, unterschiedliche CSS-Styles für unterschiedliche Bildschirmgrößen und Ausrichtungen bereitzustellen. Die Abbildung 33 zeigt eine beispielhafte CSS-Media-Regel. Mit der dargestellten Regel würden somit Geräte mit einer Browser-Breite zwischen 768px und 1024px, welche im Landspace-Modus (Querformat) sind, angesprochen werden. Dies kann für verschiedene Geräte-Spezifikationen bereitgestellt werden, ohne eine eigene Version bereit zu stellen (MDN Web Docs 2021w, 2021a).

```
@media (min-width: 768px) and (max-width: 1024px) and (orientation: landscape)
```

Abbildung 33: CSS Media Queries (Eigene Darstellung)

Sprache: Will man die Version der Progressive Web-Anwendungen von der Sprache abhängig machen, dann gibt es die schreibgeschützte „Navigator.language“ Eigenschaft des Navigator Objekts. Diese spiegelt die Browser-Sprache der Nutzerinnen und Nutzer wider. Nichtsdestotrotz gibt es für diese Problemstellung verschiedenste Übersetzungsbibliotheken, welche es möglich machen, Progressive Web-Anwendungen dynamisch zu übersetzen. Ein Beispiel hierfür ist i18n Browser-API, welche die Anwendung mit der aktuellen Browser-Sprache anhand von selbst-definierten Übersetzungsdateien in die richtige Sprache transformiert. Somit wäre es nicht notwendig, für jede Sprache eine eigene Version bereitzustellen (MDN Web Docs 2021i, 2021a).

Browserinformationen: Ist es darüber hinaus eine Anforderung, verschiedene Versionen für unterschiedliche Browserversionen und Anbieterinnen bzw. Anbieter zu erstellen, dann bietet das Navigator Objekt die notwendigen Browserinformationen. Wie man in der Evaluierung der Hardware- und Softwarezugriffe im Kapitel 4.1.1 entnehmen kann, gibt es unterschiedliche Zugriffsfreiheiten unterschiedlicher Browserversion bzw. Anbieterinne/Anbieter. Aus diesem Grund können die Informationen des Navigator Objekt des Browsers genutzt werden, um unterschiedliche Versionen für variierende Zugriffsfreiheiten bereit zu stellen. Will man zum Beispiel die Progressive Web-Anwendung-Version von der genutzten Browser-Version abhängig machen, kann die „navigator.appVersion“ Eigenschaft genutzt werden (MDN Web Docs 2021p).

Geräte-Features: Wie erwähnt, bieten unterschiedliche Browser unterschiedliche Zugriffsmöglichkeiten, welche sich im Bereich der Webtechnologien sehr dynamisch ändern. Um nicht durchgängig seitens der Anwendungen diese Zugrifffreiheiten zu pflegen, können sie programmatisch abgefragt werden. Sollte es zum Beispiel von Relevanz sein, die HTML5 Geolocation API zu nutzen, obwohl diese nicht verfügbar ist, kann es zu einer schlechten User-Experience kommen, wenn es im Zuge dessen zu einem Fehler kommt. Deshalb soll es möglich sein, unterschiedliche Versionen bereitzustellen, wenn gewisse Schnittstellen nicht vorhanden sind. Das Navigator Objekt bietet die Möglichkeit, die vorhandenen Schnittstellen abzufragen und darauf zu reagieren. Die Abbildung 34 zeigt, wie eine Abfrage aussehen könnte, um die Geolocation API abzufragen (MDN Web Docs 2021k).

```
if ("geolocation" in navigator)
```

Abbildung 34: Geräte-Features Detection (MDN Web Docs 2021k)

Des Weiteren wurde der @support CSS-Feature-Detection Mechanismus in allen untersuchten mobilen Browsern inkludiert. Dieser funktioniert ähnlich wie die Media-Queries Regeln, nur dass diese nicht von einer Bildschirm-Metrik abhängig sind, sondern die CSS-Variablen anhand des Vorhandenseins einer CSS-Funktion hinzugefügt werden müssen. Ist es zum Beispiel, wie in der Abbildung 35 gezeigt, gewünscht, einen Flex-Container zu erstellen, kann die Existenz der CSS-Eigenschaft abgefragt werden, um diese zu verwenden oder eine Alternative zu finden (MDN Web Docs 2021b).

```
@supports (display: grid) {
  div {
    display: grid;
  }
}
```

Abbildung 35: @supports CSS Detection (MDN Web Docs 2021b)

Zusammenfassend lässt sich sagen, dass es mithilfe der vorhandenen Webtechnologie möglich ist, die Nutzerinnen bzw. Nutzer und Endgeräte für unterschiedliche Versionen zu unterscheiden. Jedoch gibt es die Einschränkung (E5), dass diese Funktionen eigens implementiert werden müssen, da Progressive Web-Anwendungen im klassischen Sinne keinem App Store unterliegen, außer man verwendet „Trusted Web Activities“ und veröffentlicht die Anwendung am Google Play Store und nutzt diese bereitgestellten Funktionalitäten.

In Tabelle 8 können die Ergebnisse in einer Gegenüberstellung verglichen werden.

Qualitätsteilmerkmal	PWA			Native App	
	Safari	Chrome	Samsung	IOS	Android
Plattformunabhängigkeit	E1	E1	E1		
Sicherstellung der Auf- und Abwärtskompatibilität	E2	E2	E2		
Verschiedene Versionen veröffentlichen	E5	E5	E5	E4	E3

Tabelle 8: Gegenüberstellung der Kompatibilität von PWAs und nativen Anwendungen

4.1.7 Verlässlichkeit

Im Zuge dieses Kapitel wird das ISO 25010 Qualitätsmerkmal der Verlässlichkeit evaluiert. Hier wird primär die Betriebsbereitschaft und die Zugänglichkeit einer Software untersucht, auch wenn diese nicht geplanten Ereignissen und Einflüssen ausgesetzt ist (ISO/IEC 25010:2011).

Offline-Datenhaltung: Die Offline-Datenhaltung ist in Bezug auf die Verlässlichkeit relevant, da es auch in der gegenwärtigen Welt nicht überall selbstverständlich ist, eine gute Internetverbindung zu haben (S. O'Dea 2021b). Die Gründe dafür sind weitreichend, wie bereits im Kapitel der Kriterien-Definition erläutert. Deshalb ist es notwendig, ein Konzept zu schaffen, um die Anwendung auch offline bereitstellen zu können. Inwiefern das möglich ist, wird im Anschluss anhand der Technologieansätze einzeln evaluiert.

Android: Im ersten Schritt wird die native Android Anwendung betrachtet, um eine durchgängige Offline-Nutzung zu bieten. Auch wenn es bei diesem Technologieansatz möglich ist, ohne Netzwerkverbindung durch die Anwendung zu navigieren und statischen Inhalte anzuzeigen, können keine netzwerk-relevanten Daten geladen werden. Aus diesem Grund bietet diese Plattform unterschiedliche Speicherarten an, um Daten zwischenzuspeichern und zu einem späteren Zeitpunkt wiederzuverwenden. Hierfür gibt es vier Arten, welche im Anschluss betrachtet werden (Android Developers 2021l):

App-spezifischer Speicher: Hier werden Daten gespeichert, welche nur für die Anwendung anhand eines dedizierten Speichers bestimmt sind. Somit werden bei einem app-spezifischen Speicher vertrauliche Informationen gespeichert, welche nicht von anderen Anwendungen erreichbar sein soll (Android Developers 2021c).

Geteilter Speicher: Dieser Ansatz wird genutzt, um Daten, wie Dokumente, Medien und weitere Dateien, mit anderen Anwendungen zu teilen (Android Developers 2021s).

Geteilte Präferenzen: Diese werden verwendet, um Schlüssel-Wert Paare zu speichern, welche private und primitive Daten in einem File speichern und somit eine simple Methode für Lese- und Schreibmechanismen bieten (Android Developers 2021x).

Datenbanken: Werden nicht-triviale Mengen an strukturierten Daten verwaltet, gibt es die Möglichkeit, diese in eine lokale Datenbank zu schreiben, welche auf dem relationalen Datenbanksystem SQLite basiert (Android Developers 2021w).

Aufgrund der Vielfalt der unterschiedlichen Speicherarten kann die Offline-Fähigkeit sichergestellt werden. Des Weiteren bieten diese Offline-Speicher besonders für das Speichern von Netzwerk-Requests Abhilfe.

iOS: Seitens der Plattform iOS sieht es sehr ähnlich aus. Um Daten offline bzw. lokal zu speichern, werden verschiedenste Ansätze bereitgestellt. Die Auswahl hängt vor allem von der Struktur der Arten bzw. der Menge und Größe der Daten ab. Nichtsdestotrotz schließt die Verwendung eines Speichers nicht die Umsetzung mehrerer bereitgestellter Technologien aus. Im Anschluss werden die Haupt-Speicherarten zur Offline-Datenhaltung betrachtet.

Property List: Diese Art von Speicher bietet die Möglichkeit, einfache Strukturen und kleine Datenmengen zu speichern. Die Speicherung erfolgt im XML-Format, welche mit Schlüssel-Wert Paaren aufgebaut ist. Eine Property List hat dabei die Eigenschaft, dass nicht alle Datentypen akzeptiert werden. Beispiele für akzeptierte Typen sind unter anderem Arrays, Integer, String und Boolean (Apple Developer 2021s).

Core Data: Der als „Core Data“ bezeichnete Service wird von Apple für eine lokale Ablage von Anwendungsdaten empfohlen. Dieser Ansatz verwendet hierfür im Hintergrund das relationale Datenbanksystem SQLite als Haupt-Datenbank. „Core Data“ macht dabei SQLite Abfragen, um die Daten lokal und offline abzulegen. Somit muss keine weitere Datenbank eingerichtet werden. Es wird eine Schnittstelle angeboten, welche es möglich macht, Funktionalitäten wie Speichern, Löschen, Auslesen etc. von Daten zu gewährleisten. Darüber hinaus werden die gespeicherten Inhalte, mit der Voraussetzung einer aktiven Netzwerkverbindung, über alle Geräte synchronisiert, welche mit denselben Apple Account eingeloggt sind (Apple Developer 2021j, 2021q).

NSUserDefaults: Dieser Ansatz bietet eine Schnittstelle, um die Standard-Einstellungen der Benutzerin bzw. des Benutzers anhand Schlüssel-Wert Paare abzulegen. Die NSUserDefaults-Kalsse bietet eine Schnittstelle, um ein System für konfigurierte Benutzereinstellungen bereitzustellen. Somit kann das System an die Verhaltensweisen und Vorlieben der Benutzerinnen bzw. Benutzer angepasst werden. Zum Beispiel ist es möglich, Benachrichtigungseinstellungen oder den User-Token abzulegen. Eine Anwendung speichert dabei diese Einstellungen in der Standarddatenbank und kann diese bei einem Neustart abrufen, damit die gewünschte Funktionsweise beibehalten bleibt. Um nicht jedes Mal die Daten von der Datenbank abfragen zu müssen, werden diese in einem Cache geladen und von dort geladen. Erst beim erneuten Setzen eines Default-Werts wird die Datenbank aktualisiert (Apple Developer 2021r).

Key Chain: Sollen sensible Daten gespeichert werden, bietet Apple den „Key Chain“ Service an. Dieser verfolgt den Ansatz, sensible Daten wie Passwörter, Zertifikate und weitere Informationen verschlüsselt in eine dafür konzipierte Datenbank zu speichern (Apple Developer 2021o).

Progressive Web-Anwendungen: Da eines der Schlüssel-Prinzipien von Progressive Web-Anwendungen die Netzwerk-Unabhängigkeit darstellt, werden im Anschluss die bereitgestellten Mittel und Einschränkungen betrachtet, um eine positive Offline-Erfahrung zu schaffen (MDN Web Docs 2021m).

Um diese Art von Anwendung Offline betreiben zu können, ist es notwendig, statische Daten wie zum Beispiel HTML, CSS und JavaScript bereitzustellen, wie auch dynamische Daten der Anwendung zu speichern. Grundsätzlich verfolgen Progressive Web-Anwendungen das Ziel, die generierten Daten mithilfe des Service-Workers zwischenzuspeichern und falls eine Verbindung mit einem Server hergestellt werden kann, dass diese Daten übermittelt werden. Dasselbe gilt bei der Nutzung der Daten. Ist eine aufrechte Verbindung mit dem Server vorhanden, werden die Daten von dort geladen. Diese werden nicht nur zur Laufzeit verwendet, sondern auch auf lokalen Speichern zwischengespeichert, sodass diese Daten zu einem späteren Zeitpunkt wiederverwendet werden können, falls keine Netzwerkverbindung vorhanden ist (Google Developers 2021b).

Zur lokalen Ablage der Daten gibt es seitens der Browser-Technologie mehrere Speicherarten, um diese für einen späteren Zeitpunkt bereitzustellen. Im Anschluss werden diese unterschieden:

- Cache: Zur Realisierung eines Cache-Speichers gibt es im Bereich der Web-Schnittstellen die Cache Storage Browser API. Diese setzt den Fokus auf die Ablage von

statischen Ressourcen, wie HTML, CSS und JavaScript Dateien. Die Cache Storage API bietet hiermit einen persistenten Speicher für Request/Response Objekte, welche für einen späteren Anwendungsfall wiederverwendet werden können. Da diese im Einklang mit Service-Worker zugänglich sind, wird dieser Ansatz im Anschluss näher betrachtet (MDN Web Docs 2021g, 2021f).

- IndexedDB: Eine IndexedDB ist eine NoSQL-Datenbank des Browsers. Die Einträge werden dabei in Paare, welche aus einem Schlüssel und aus einem Wert bestehen, angelegt. Dieser Ansatz bietet im Gegenzug zu den anderen Speicherarten eine höhere Speicherkapazität. Somit können hier Dateien gespeichert werden, welche zu einem späteren Zeitpunkt bei zum Beispiel schlechter Netzwerkverbindung direkt aus diesem Speicher geladen werden, ohne auf den Online-Status zu warten. Aufgrund der Tatsache, dass dieses Feature mit dem Service-Worker umsetzbar ist, wird dies im Anschluss detailliert betrachtet (MDN Web Docs 2021l).
- SessionStorage: Dies ist ein tab-spezifischer Speicher, welcher nur auf die Lebensdauer der Registerkarte oder des Browsers eingeschränkt ist. Dieser ist nützlich, um sitzungsspezifische Daten zu speichern. Dieser Speicher arbeitet synchron und blockiert bei der Verwendung den Hauptthread der Anwendung. Des Weiteren ist dieser je nach Browser auf nur ca. 5 MB eingeschränkt. Dieser Speicher ist jedoch nicht vom Service-Worker benutzbar und somit für die Sicherstellung der Offline-Fähigkeit nicht relevant (selfhtml 2021).
- LocalStorage: Der LocalStorage der Browser ist ähnlich wie der Session Speicher, mit der Abweichung, dass dieser nicht mit einer Ablaufdauer hinterlegt ist. Dieser wird somit erst dann gelöscht, wenn die Benutzerin bzw. der Benutzer oder die Anwendung selbst diesen Speicher leert. Nichtsdestotrotz kann diese Art von Speicher auch nur synchron arbeiten, ist auf ca. 5 MB Kapazität beschränkt und nicht mit dem Service-Worker zugänglich. Aus diesen Gründen ist dieser Speicher auch nicht brauchbar für die Offline-Fähigkeit (selfhtml 2021).

Da in Bezug auf Progressive Web-Anwendungen der Cache Speicher und die IndexedDB die wesentlichsten Bestandteile zur Sicherstellung der Offline-Fähigkeit sind, werden diese, wie bereits oben beschreiben, im Anschluss näher betrachtet:

Der **Cache Speicher** verfolgt das Ziel, statische Ressourcen mit der Cache Storage API asynchron bereitzustellen. Diese Schnittstelle wird von der essenziellen Schnittstelle von Progressive Web-Anwendungen der Service Worker API inkludiert. Obwohl diese initial für den Service-Worker implementiert wurde, kann diese Schnittstelle von überall im Quellcode verwendet werden. Wie man bereits der Abbildung 7 entnehmen konnte, ist es anhand der „caches.open()“ und „cache.addAll()“ Methode möglich, die statischen Ressourcen der Anwendung im Cache zwischenzuspeichern. Um diese Daten im Anschluss aus diesem Speicher zu laden, gibt es verschiedenste Ansätze. Hierfür hat Jake Archibald und Google gemeinsam den „The Offline Cookbook“ Beitrag veröffentlich, welcher sich mit den unterschiedlichen Varianten befasst. Das Unternehmen Google referenziert in der offiziellen Dokumentation auf diesen Artikel, betrachtet hier jedoch nur eine eingeschränkte Anzahl an entscheidenden Ansätzen, welche auch

auf diese Herangehensweisen in dieser Arbeit beschränkt werden (Google Developers 2021g; Jake Archibald 2021).

Cache only: Dieses Konzept eignet sich für statische Ressourcen, welche Teil der Haupt-Version der Anwendung sind. Anfragen an den Service-Worker werden dabei direkt aus dem Cache ausgeliefert, wie Abbildung 35 zeigt. Da diese Daten nicht aktualisiert werden, ist es notwendig, die Daten bereits bei Installationsprozess in den Cache abzulegen, um die Verlässlichkeit der Daten zu gewährleisten (Jake Archibald 2021).

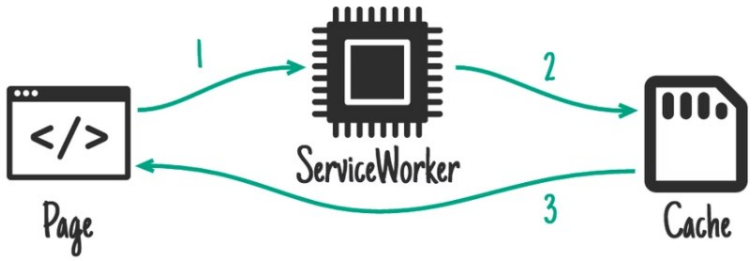

Abbildung 36: Service Worker Caching - Cache only (Jake Archibald 2021)

Cache falling back to network: Dieser Ansatz verfolgt einen ähnlichen Ansatz, wie das „Cache only“ Konzept. Lediglich mit der Abweichung, dass Service-Worker Anfragen, welche nicht vom Cache ausgeliefert werden können, in letzter Instanz mithilfe des Netzwerks neu geladen werden. Der Prozess hierfür wird in Abbildung 36 veranschaulicht (Jake Archibald 2021).

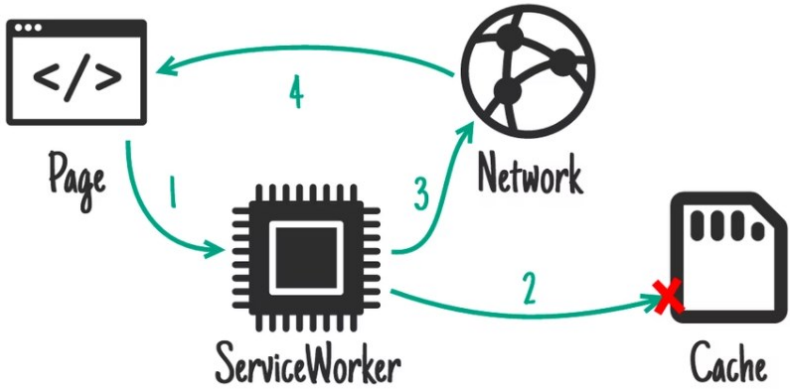

Abbildung 37: Service Worker Caching - Cache falling back to network (Jake Archibald 2021)

Network falling back to cache: Die Abbildung 38 zeigt einen sehr ähnlichen Ansatz, wie der davor genannte. Dieser Ansatz ist hilfreich, wenn Ressourcen häufig aktualisiert werden müssen, auch bei schlechter oder nicht vorhandener Netzwerkverbindung, oder zumindest veraltete Daten geladen werden sollen. Ein Beispiel sind diverse Social Media Kanäle, welche bereits geladene Daten anzeigen, jedoch nicht aktualisieren. Somit erhalten Online-Benutzerinnen und -Benutzer immer die aktuellen Inhalte und Offline-Benutzerinnen bzw. -Benutzer können trotzdem veraltete Daten laden, was die Anwendung in einer gewissen Weise noch benutzbar machen kann. Nichtsdestotrotz kann dieser Ansatz große Performance-Einschränkungen mit sich bringen. Hat man eine langsame Verbindung, dann muss gewartet werden bis die Service-Worker Anfrage abbricht, um es erst dann aus dem Cache zu laden (Jake Archibald 2021).

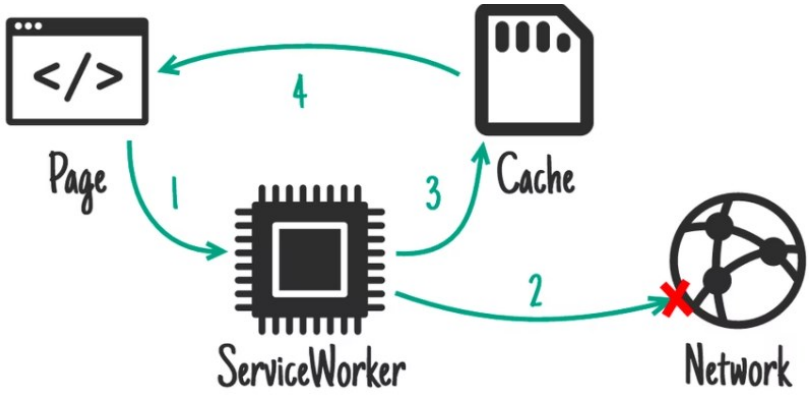

Abbildung 38: Service Worker Caching - Network falling back to cache (Jake Archibald 2021)

Cache then network: Um den genannten Performance-Einschränkungen entgegenzuwirken, gibt es den „Cache then network“ Ansatz. Dieser Ansatz dient für Inhalte, welche häufig aktualisiert werden müssen, jedoch in erster Instanz bereits geladene Daten verwenden. Die Abbildung 39 zeigt, dass dieses Konzept zwei Anfragen stellt. Eine wird an den Cache direkt gestellt und eine weitere wird über den Service-Worker an das Netzwerk realisiert. Die grundlegende Idee ist dabei, die zwischengespeicherten Daten zuerst anzuzeigen und erst dann die Aktualisierung der Anwendung durchführen, wenn die Daten vollständig vom Netzwerk geladen wurden (Jake Archibald 2021).

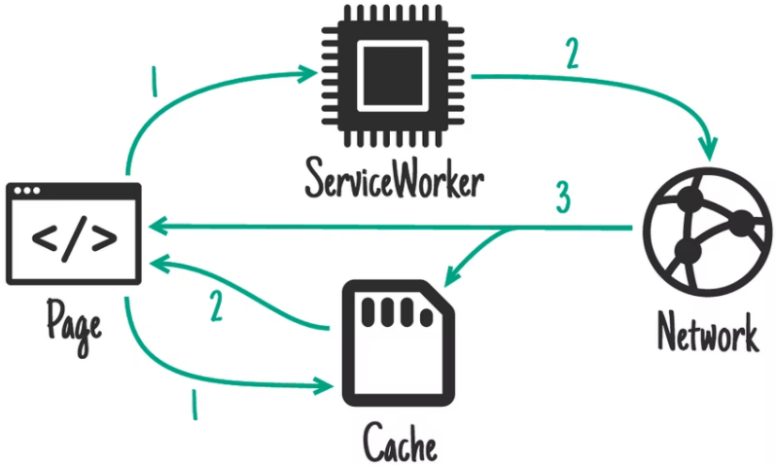

Abbildung 39: Service Worker Caching - Cache then network (Jake Archibald 2021)

Des Weiteren gibt es den bereits genannten Ansatz von IndexedDB, welcher eine NoSQL-Datenbank des Browsers darstellt. Die Datenbank-Objekte bestehen aus Schlüssel-Wert Paaren, welche in der Abbildung 40 gezeigt werden. Diese Paare werden in der Browser-Datenbank für spätere Anwendungszwecke gespeichert und bei Bedarf ohne einer aktive Netzwerkverbindung daraus geladen. Die Operationen mit der IndexedDB werden asynchron ausgeführt, was wiederum bedeutet, dass die Anwendung zu diesen Zeitpunkten nicht blockiert wird und trotzdem bedienbar bleibt (MDN Web Docs 2021l).

#	Key (keypath 'id')	Value
0	1234	{id: 123, name: 'coke', price: 10.99, quantity: 200}
1	9876	{id: 321, name: 'pepsi', price: 8.99, quantity: 100}
2	4567	{id: 222, name: 'water', price: 11.99, quantity: 300}

Abbildung 40: IndexedDB Schlüssel-Wert Paare (Google Developers 2021b)

Zusammenfassend lässt sich sagen, dass aufgrund des Service-Worker Cache und der IndexedDB die Möglichkeit geboten wird, dass eine Progressive Web-Anwendung Offline verfügbar gemacht wird und somit die Verlässlichkeit gewährleistet werden kann. Diese zwei Schnittstellen sind dabei für alle drei Browser, dem mobilen Chrome, Safari und Samsung Internet Browser verfügbar (Can I use 2021b).

Hintergrund-Synchronisation: Des Weiteren wird die Sicherstellung der Daten-Synchronisation mit einem Server bei nicht-vorhandener Netzwerkverbindung betrachtet. Wie bereits in der Kriteriendefinitions-Phase erläutert, verfolgt man das Ziel Server-Anfragen zwischenzuspeichern und bei aktiver Netzwerkverbindung abzuarbeiten. Um diese Anforderung zu gewährleisten, werden die einzelnen Technologieansätze nacheinander evaluiert.

Android: Wenn die Anwendung sich im Offline-Modus einer nativen Android-Anwendung befindet und man trotzdem Anfragen an einen Server nicht verlieren will, kann man diese zum Beispiel als Schlüssel-Wert Paare in einem „Geteilten Präferenzen“ Speicher ablegen und bei aktiver Netzwerk-Verbindung von dort nacheinander abarbeiten. Um die Anwendung nicht zu blockieren, bietet Android den Sync-Adapter. Diese Komponente kapselt die Anwendung für den Datenaustausch zwischen Client und Server während der Übertragung aus. Somit kann die Synchronisation der Daten im Hintergrund sichergestellt werden (Android Developers 2021i).

iOS: Im Bereich von nativen iOS Anwendungen wird seit der Betriebssystem-Version 13 das „BackgroundTasks“ Framework angeboten. Dieses bietet die aktuellen Features zur Umsetzung von Aufgaben im Hintergrund. Zur Realisierung gibt es die „BGTaskScheduler“ API, welche es möglich macht, Aufgaben für die Ausführung im Hintergrund zu planen. Somit lassen sich Interaktionen, welche im Offline-Betrieb getätigt wurden, nach der Bewerkstelligung einer aktiven Netzwerkverbindung im Hintergrund ausführen. Folglich kann die Verlässlichkeit gewährleistet werden, ohne die Anwendung zu stoppen (Apple Developer 2021h, 2021e).

Progressive Web-Anwendungen: Auch bei Progressive Web-Anwendungen soll es möglich sein, Daten im Hintergrund zu synchronisieren. Um dieses Problem zu adressieren, wurde die Background Synchronization Web-API veröffentlicht. Diese kann mithilfe eines installierten Service-Workers genutzt werden. Diese Schnittstelle hilft, die Interaktionen mithilfe des Service-Workers zusammenzufassen und bei aktiver Netzwerkverbindung diese Aufgaben asynchron abzuarbeiten. Dabei registriert die Anwendung eine Synchronisations-Aufgabe beim Service-Worker. Bei aktiver Netzwerkverbindung wird dieser mit einem Event aufmerksam gemacht und arbeitet somit die Server-Anfragen im Hintergrund ab. Aus diesem Grund kann das Qualitätsziel der Verlässlichkeit gewährleistet werden, ohne die Anwendung zu unterbrechen, da die Aufgaben

asynchron ausgeführt werden. Nichtsdestotrotz ist die Verwendung dieser Schnittstelle nicht bei allen Browser-Anbieterinnen/Anbieter gewährleistet. Der mobile Chrome und Samsung Internet Browser bieten den vollen Funktionsumfang, der mobile Safari Browser jedoch keine Unterstützung. Um dieses Problem zu adressieren, wäre es ein möglicher Ansatz, die durchgeführten Anfragen im Offline-Modus in einen der genannten Speicherarten anzulegen und dann synchron abzuarbeiten. Trotzdem kann man dabei nicht gewährleisten, dass die Anwendung in der Zwischenzeit bedienbar bleibt (MDN Web Docs 2021d).

In Tabelle 9 werden die Ergebnisse zusammengefasst.

Qualitätsteilmerkmal	PWA			Native App	
	Safari	Chrome	Samsung	IOS	Android
Offline-Datenhaltung					
Hintergrund-Synchronisation					

Tabelle 9: Gegenüberstellung der Verlässlichkeit von PWAs und nativen Anwendungen

4.1.8 Usabilty

Im letzten Schritt der Evaluierungsphase wird das ISO 25010 Qualitätsmerkmal Usability betrachtet. Wie im Kapitel der Kriterien-Definition bereits erläutert, wird der Fokus auf Features und Funktionen gelegt, welche den Nutzer/innen eine einfache Bedienung und Steuerung ermöglichen. Des Weiteren wird betrachtet, wie fern sichergestellt werden kann, dass die Ästhetik der Benutzeroberfläche den gewohnten Vorstellungen der Nutzer/innen entspricht. Aus diesem Grund werden im Anschluss die bereits definierten Kriterien einzeln betrachtet und in ihren Qualifikationen evaluiert.

Einheitliches Look & Feel: Im ersten Schritt wird das Qualitätsteilmerkmal „Einheitliches Look & Feel“ betrachtet. Ziel dabei ist es, die Interaktionen mit der Software so intuitiv wie möglich zu gestalten, um für die Benutzerinnen bzw. Benutzer die gewohnte Benutzeroberfläche zu bieten.

Android: Bei der Evaluierung dieses Qualitätsziels wurde mit der native Android Anwendung gestartet. Um eine einheitliche Style-Guideline für die Anwenderin oder den Antwender zu schaffen, wurde seitens des Unternehmens Google im Jahre 2014 die UI-Bibliothek Material Design ins Leben gerufen. Diese Bibliothek soll eine Android-Software durchgängig über alle Anwendungen als solche Plattform erkennbar machen bzw. UI-Elemente und Interaktionen über alle Softwarelösungen vereinheitlichen. Material Design bietet für diesen Zweck zwei wesentliche Bereiche an. Zum einen sind das Style-Guidelines, um ein ideales natives Benutzererlebnis zu schaffen. Zum anderen werden bereits definierte UI-Komponenten bereitgestellt, welche nur in die Anwendung eingebunden werden müssen. Die Design-Richtlinien, welche als Best-Practice für die Entwicklerin bzw. den Entwickler aufbereitet sind, sind sehr weitreichend. Diese beinhalten Empfehlungen für Layout-Adaptierungen an verschiedene Bildschirmauflösungen, eine

Navigation innerhalb der Anwendung, Farben, Schriftarten, Töne, Icons, Formen und vieles mehr. Des Weiteren gibt es, wie bereits erwähnt, vordefinierte UI-Komponenten, welche in mehreren Varianten für verschiedene Anwendungszwecke bereitgestellt werden. Hier reicht die Auswahl von Buttons, Cards, Dialoge und weitere verwendbare Benutzeroberfläche-Elemente (Material Design 2021a; Android Developers 2021o; Material Design 2021b).

iOS: Im Anschluss wird das beobachtete Qualitätsteilmerkmal in Bezug auf native iOS Anwendungen betrachtet. Um seitens Apple eine einheitliche Benutzererfahrung zu schaffen, gibt es das „Human Interface Guidelines“ Handbuch, welches die Design-Prinzipien für native iOS Anwendungen zusammenfasst. Darüber hinaus kann aus dieser Anleitung entnommen werden, dass seitens der Benutzung von Komponenten auf die UI-Bibliothek „UIKit“ zurückgegriffen werden soll. Diese ist ähnlich der „Material Design“ Bibliothek aufgebaut. Sie besteht grundsätzlich auch aus den zwei Bestandteilen der definierten Komponenten und der allgemeinen Style-Guidelines. Nichtsdestotrotz schreibt die App Store Review Guideline zu einem gewissen Grad vor, dass man diese Design Anleitung befolgen muss. Somit kann es im Gegensatz zu Android Anwendungen passieren, dass Anwendungen bei Nichteinhaltung von der App Store Betreiberin bzw. dem Betreiber nicht veröffentlicht werden (Apple Developer 2021b, 2021w).

Progressive Web-Anwendungen: Der letzte Teil dieses Qualitätsteilmerkmales bildet die Progressive Web-Anwendung. Aufgrund der Tatsache, dass diese Art an Softwarelösung plattformunabhängig ist und in der Basis eine Webseite mit den Basistechnologien HTML, CSS und Javascript darstellt, gibt es keine spezifischen Design-Guidelines. Die Freiheiten sind im vollen Umfang den Entwicklerinnen und Entwicklern überlassen. Jedoch stehen diese vor der Herausforderung, ein natives Benutzererlebnis zu schaffen, ohne auf die plattformspezifischen UI-Komponenten zurückgreifen zu können. Wie man der Abbildung 41 entnehmen kann, sind die verschiedenen Bestandteile über die Plattformen hinweg unterschiedlich ausgeprägt. Grundsätzlich könnte man über das Javascript Window-Navigator Objekt den Browser und die Plattform herausfinden und mithilfe dieser Informationen auf die davor genannten Style-Guidelines zurückgreifen. Jedoch wäre dies ein immenser Mehraufwand und der Vorteil der Plattformunabhängigkeit von Progressive Web-Anwendungen würde verloren gehen (designflyover 2021; MDN Web Docs 2021p; Huber et al. 2021).

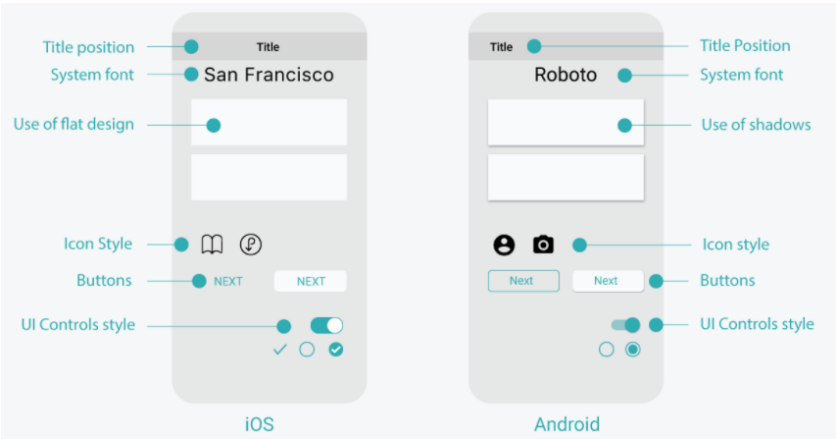

Abbildung 41: Unterschiede UI-Element iOS und Android (designflyover 2021)

Zusammenfassend lässt sich feststellen, dass die Plattformen Android und iOS eigene Richtlinien zur Umsetzung der Benutzeroberfläche bzw. Interaktionen bereitstellen und somit ein intuitives und durchgängiges Design bieten können. Bei Progressive Web-Anwendungen liegt kein Standard vor, welcher zum Einsatz kommt, da diese in der Basis eine klassische Webseite sind.

Inter-app Sharing: Des Weiteren wird im Anschluss das Qualitätsteilmerkmal des Inter-app Sharings betrachtet. Hier wird das Ziel verfolgt, Daten von einer Anwendung in eine andere, mit den bereitgestellten Features der Technologieansätze, zu transferieren. Inwiefern diese funktionale Anforderung verfügbar ist, wird im Anschluss einzeln betrachtet.

Android: Bei der nativen Android-Anwendungsentwicklung werden zwei unterschiedliche Ansätze bereitgestellt, um Inhalte direkt aus der Anwendung mit anderen zu teilen. Dabei gibt es zum einen das Android Sharesheet und zum anderen den Android-Intent-Resolver. Beide nutzen Android Intents zur Umsetzung dieser Funktionalität. Intents werden jedoch nicht explizit im Detail beschrieben. In Kürze zusammengefasst sind diese seitens der Android-Plattform asynchrone Nachrichten. Diese kommen immer dann zum Einsatz, wenn Aktionen von weiteren Anwendungskomponenten bezogen werden sollen oder Information von Ereignissen benötigt werden. Um den Unterschied zu verdeutlichen, werden die zwei Möglichkeiten im Anschluss nacheinander betrachtet (Android Developers 2021n, 2021y).

Als erstes wird der Android Sharesheet Ansatz untersucht. Dieser ist primär dafür gedacht, Daten wie zum Beispiel eine URL direkt mit anderen Benutzerinnen bzw. Benutzern auszutauschen. Um diesen Ansatz nutzen zu können, gibt es die Intent „ACTION_SEND“ Funktion, welche die gewünschte Aktion zum Teilen von Inhalten beschreibt. Die Abbildung 42 zeigt dabei die Einfachheit der programmatischen Umsetzung zum Senden von Text. Es ist prinzipiell nur notwendig, einen Intent mit einer Action, Extras und den Type als Parameter zu initialisieren. Um in weiterer Folge das Android Sharesheet anzuzeigen, muss die “createChooser()” Funktion mit dem Intent Objekt aufgerufen werden. Im Anschluss kann eine neue Instanz einer Activity mit dem Intent gestartet werden, welche einen einzelnen Screen in der Anwendung repräsentiert. Darüber hinaus ist es möglich, weitere Typen an Daten zu teilen. Diese Unterscheidung der Datentypen ist mit der Angabe des standardisierten Internet Media Type (MIME) als Parameter des Intents möglich (Android Developers 2021y).

```
val sendIntent: Intent = Intent().apply {
    action = Intent.ACTION_SEND
    putExtra(Intent.EXTRA_TEXT, "This is my text to send.")
    type = "text/plain"
}

val shareIntent = Intent.createChooser(sendIntent, null)
startActivity(shareIntent)
```

Abbildung 42: Android Intent zum Teilen von Daten (Android Developers 2021y)

Der Android Intent Resolver wird im Gegensatz am besten dann verwendet, wenn Daten im Rahmen eines klar definierten Taskflows an eine andere App gesendet werden sollen. Wenn

mehrere Anwendungen mit Filtern installiert sind, die mit ACTION_SEND und dem MIME-Typ übereinstimmen, zeigt das System einen Dialog namens Intent Resolver an, der es der Benutzerin bzw. dem Benutzer ermöglicht, ein Ziel für die Freigabe auszuwählen (Android Developers 2021y).

Um dabei auf der anderen Seite auch Daten von anderen Anwendungen empfangen zu können und somit als Empfängerin oder Empfänger zu agieren, gibt es die Möglichkeit, die Manifest-Datei der Anwendung mit dem <intent-filter> Parameter anzupassen. Die Intent-Filter informieren dabei das System, welche Daten von der Anwendung akzeptiert werden. Wenn eine Anwendung Daten wie Bilder und Text-Inhalte behandeln kann, dann würde die Manifest Konfiguration wie in Abbildung 43 dargestellt aussehen. (Android Developers 2021v)

```
<activity android:name=".ui.MyActivity" >
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="image/*" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="text/plain" />
    </intent-filter>
</activity>
```

Abbildung 43: Android Manifest <intent-filter> (Android Developers 2021v)

Sollte in weiterer Folge eine andere Anwendung einen Intent erzeugen und die dazugehörige „startActivity()“ aufrufen (Abbildung 42), dann wird die vorliegende Anwendung als Share-Target angezeigt. Wenn die Nutzerin bzw. der Nutzer die Anwendung wählt, wird die dazugehörige Aktivität gestartet, welche in der vorliegenden Darstellung „MyActivity“ wäre. Daraufhin liegt es in der Hand der Entwicklerinnen bzw. Entwickler, sich um die Daten zu kümmern (Android Developers 2021v).

iOS: In einem weiteren Schritt wird bei der Plattform iOS betrachtet, inwiefern es möglich ist, Daten innerhalb von Anwendungen zu teilen. Im Bereich der iOS Entwicklung gibt es zahlreiche Erweiterungen (Extensions), um Daten verfügbar zu machen, während Anwendungen miteinander interagieren. Um diese Problemstellung speziell zu adressieren, wird die Share-Extension von Apple angeboten (Apple Developer 2021v).

Primär wird die Erweiterung genutzt, um Daten an weitere Anwendungen zu senden. Grundsätzlich wird hier bei einem Klick auf den Teilen-Button die “didSelectPost” Methode der Share-Extension aufgerufen. Bei dieser Methode kann eine “background-mode” Session (NSURLSession) aufgerufen werden, welche den Inhalt zum Teilen enthält und die Teilen-Ansicht mit den verfügbaren Anwendungen öffnet. Damit die Benutzerin bzw. der Benutzer eine Vorschau von den ausgewählten Inhalten erhält, bietet der bereitgestellte Controller (SLComposeServiceViewController) des Systems die Möglichkeit, Standarddatentypen, wie Videos, Fotos etc., anzuzeigen. (Apple Developer 2021v)

Um im Gegensatz dazu Daten zu erhalten, kann man mithilfe der IDE XCode eine Share-Schema anlegen. Hier ist es notwendig, eine “NSExtensionActivationRule” anzugeben, um das System zu

informieren, wann die vorliegende Anwendung als Ziel für Daten in Frage kommt. Sonst würde die Anwendung immer als Ziel vorgeschlagen werden, auch wenn die Anwendung nicht in der Lage ist, diese Daten zu verwalten. Mithilfe des UIViewController ist es möglich, die Daten zu erhalten und in weiterer Folge an die gewünschten Anforderungen anzupassen (Fabio Pelizzola 2020).

Progressive Web-Anwendungen: Im Anschluss werden die Möglichkeiten zur Realisierung von Inter-app Sharing bei Progressive Web-Anwendungen betrachtet. Im Bereich der Webtechnologien gibt es dabei die Web Share API und die Web Share Target API, welche es der Anwendung erlauben, Daten mit anderen Softwarelösungen zu teilen und zu erhalten. Mit der Web Share API kann auf dieselben Freigabefunktionen wie bei plattformspezifischen Anwendungen zugegriffen werden. Aus diesem Grund ist es damit möglich, Text, Links und Dateien für bereits installierte Anwendungen bereitzustellen. Die Untersuchung der Browserimplementierung hat ergeben, dass bereits alle drei untersuchten Alternativen diese Schnittstelle bereitstellen (MDN Web Docs 2021q; Joe Medley 2021). Die Abbildung 44 zeigt das Teilen von Links und Texten. Um dies zu realisieren, wird die Navigator Funktion „share()“ mit der Möglichkeit der Eigenschaften „title“, „text“, „url“ oder „files“ aufgerufen und teilt somit dem System mit, dass Daten mit anderen Anwendungen ausgetauscht werden möchten.

```
if (navigator.share) {
  navigator.share({
    title: 'web.dev',
    text: 'Check out web.dev.',
    url: 'https://web.dev/',
  })
    .then(() => console.log('Successful share'))
    .catch((error) => console.log('Error sharing', error));
}
```

Abbildung 44: Web Share Target API share() Methode (Joe Medley 2021)

Darüber hinaus legt die Web Share Target API den Fokus auf das Empfangen von Daten anderer Anwendungen. In der Vergangenheit war die Möglichkeit, Inhalte von anderen Anwendungen zu erhalten, nur für native Alternativen gestattet. Nichtsdestotrotz ist es mit der Web Share Target API auch bei Progressive Web-Anwendungen in allen drei untersuchten Browsern zulässig, als Share-Ziel zu agieren. Voraussetzung für die Nutzung sind zum einen die Installation der Anwendung am Home-Screen und zum anderen die Konfiguration des „share_target“ Attribut des Manifest-Files der Progressive Web-Anwendung. Wird dieser Eintrag hinzugefügt, dann wird das Betriebssystem darauf aufmerksam gemacht, dass es sich hier um ein Share-Ziel handelt. Es gibt dabei folgende drei Wege, wie die Manifest-Datei konfiguriert werden kann (Joe Medley und Pete LePage 2019):

- Akzeptanz grundlegender Informationen: Bei der Akzeptanz grundlegender Informationen agiert man als Empfängerin bzw. Empfänger von Daten, Links und Texten. Hierfür müssen folgende Konfigurationen aus der Abbildung 45 erstellt werden.

```
"share_target": {
  "action": "/share-target/",
  "method": "GET",
  "params": {
    "title": "title",
    "text": "text",
    "url": "url"
  }
}
```

Abbildung 45: Share Target API - Akzeptanz grundlegender Informationen (Joe Medley und Pete LePage 2019)

- Akzeptanz von Anwendungsänderungen: Die Ziel-Anwendung kann mit der Web Share Target API anhand freigegebener Daten verändert werden. Hier muss der Methodenwert auf „POST“ gesetzt und der enctype angegeben werden.

```
"share_target": {
  "action": "/bookmark",
  "method": "POST",
  "enctype": "multipart/form-data",
  "params": {
    "url": "link"
  }
}
```

Abbildung 46: Share Target API - Akzeptanz von Anwendungsänderungen (Joe Medley und Pete LePage 2019)

- Akzeptanz von Dateien: Darüber hinaus bietet diese Schnittstelle die Möglichkeit, Dateien auszutauschen. Wie bei der Akzeptanz der Anwendungsänderungen ist es hier notwendig, die „POST“ Methode den enctype anzugeben. Jedoch muss noch ein Datei-Array bei den Parametern hinzugefügt werden, welche den Namen und den gewünschten Datentyp enthält.

```
"files": [
  {
    "name": "records",
    "accept": ["text/csv", ".csv"]
  },
  {
    "name": "graphs",
    "accept": "image/svg+xml"
  }
]
```

Abbildung 47: Share Target API - Akzeptanz von Dateien (Joe Medley und Pete LePage 2019)

Schnellzugriffe per Kontextmenü: Den letzten Punkt der Evaluierung bildet die Verwendung eines Quick-Action Kontextmenüs. Dabei soll es möglich sein zum Beispiel häufig verwendete Funktionen direkt über das Homescreen-Icon über einem Kontextmenü zu erhalten. Um dieser Anforderung gerecht zu werden, werden im Anschluss die unterschiedlichen Plattformen erneut gegenübergestellt.

Android: Das Android Ecosystem bietet eine Reihe an Diensten an, mit denen es machbar ist, ein Kontextmenü für Schnellzugriffe zu schaffen. Zur Umsetzung bietet diese Plattform dabei drei Typen an Shortcuts, welche im Anschluss erläutert werden (Android Developers 2021k, 2021e):

- Statische Schnellzugriffe: Diese Art von Zugriff bietet dabei der Benutzerin bzw. dem Benutzer die Möglichkeit, eine Reihe an Verknüpfungen zu definieren, welche in der Anwendung fest verankert sind und sich mit der Zeit nicht verändern. Aufgrund der Tatsache, dass diese einen undynamischer Quellcode darstellen, können diese auch aufgerufen werden, wenn die Anwendung nicht im Hintergrund läuft (Android Developers 2021k).
- Dynamische Schnellzugriffe: Diese Schnellzugriffe werden dazu verwendet, um kontext-abhängige Aktionen zu realisieren. Hierfür werden die Menüeinträge dynamisch basierend auf ein Benutzer-Verhalten erstellt. Somit können die Nutzerinnen bzw. Nutzer auf ihren Anwendungszweck bestimmte Schnellzugriffe verwenden (Android Developers 2021k).
- Angeheftete Schnellzugriffe: Diese Verknüpfung wird verwendet, um eine spezielle Benutzer-Aktion auszuführen. Hierfür wird ein eigenes Homescreen-Icon erstellt, welches eine vordefinierte Handlung durchführt. Man kann direkt mit Klick auf das Anwendungsicon in die Anwendung springen und eine Funktion ausführen. Ein Beispiel hierfür ist die Möglichkeit eine Webseiten-URL direkt am Homebildschirm mithilfe eines Icons zu hinterlegen, um diese direkt von dort zu öffnen (Android Developers 2021k).

Nichtsdestotrotz gibt es hier Limitierungen, welche im Anschluss betrachtet werden. Zum einen kann die Entwicklerin bzw. der Entwickler nur vier Aktionen beim statischen und dynamischen Kontextmenü hinzufügen. Zum anderen gibt es bei angehefteten Schnellzugriffen kein Maximum beim Hinzufügen zum Homescreen. Darüber hinaus kann die Entwicklerin bzw. der Entwickler die obsoleten Shortcuts nicht von den Nutzerinnen oder Nutzern entfernen, diese können lediglich deaktiviert werden (Android Developers 2021e).

iOS: Um bei nativen iOS Anwendungen den Schnellzugriff per Kontextmenü zu gewährleisten, gibt es seit der Betriebssystem-Version 13 die Möglichkeit, “Quick Actions” für diesen Zweck zu nutzen. Dabei gibt es ähnlich wie beim Android-Ansatz statische und dynamische Menü-Einträge.

Statische Schnellzugriffe: Die statischen Menüeinträge sind sichtbar, sobald die Nutzerin bzw. der Nutzer die Anwendung installiert hat oder ein Update durchführt. Mithilfe der UIApplicationShortcutItems Klasse wird dabei der Title, das Icon, die Info und der Eintrag-Type jedes Menüeintrages angegeben. Die Kontextmenü-Konfiguration einer Suche könnte wie in Abbildung 47 aufgebaut sein (Karthikeyan T 2020).

```
<key>UIApplicationShortcutItems</key>
<array>
    <dict>
        <key>UIApplicationShortcutItemTitle</key>
        <string>Search</string>
        <key>UIApplicationShortcutItemSubtitle</key>
        <string>by product name</string>
        <key>UIApplicationShortcutItemIconType</key>
        <string>UIApplicationShortcutIconTypeSearch</string>
        <key>UIApplicationShortcutItemType</key>
        <string>QuickAction.Search</string>
    </dict>
</array>
```

Abbildung 48: iOS Quick Actions – UIApplicationShortcutItems Konfiguration (Karthikeyan T 2020).

Dynamische Schnellzugriffe: Im Gegenzug werden die dynamischen Kontextmenü-Einträge erst nach dem erstmaligen Start verfügbar gemacht, welche sie an das Verhalten oder Eigenschaften der Nutzerinnen bzw. Nutzer angepasst sind. Um dabei die Kontextmenü-Einträge dynamisch zu generieren, gibt es die “UIApplication.shared.shortcutItems” Eigenschaft. Diese kann zur Laufzeit programmatisch je nach Anforderungen und Benutzer-Daten angepasst werden. (Yong Cui 2020)

```
shortcutItems += [
    UIApplicationShortcutItem(type: "Test Type 0", localizedTitle: "Test Title 0"),
    UIApplicationShortcutItem(type: "Test Type 1", localizedTitle: "Test Title 1")
]

UIApplication.shared.shortcutItems = shortcutItems
```

Abbildung 49: iOS Quick actions - UIApplication.shared.shortcutItems (Yong Cui 2020)

Progressive Web-Anwendung: Um diese Problemstellung bei Progressive Web-Anwendungen zu adressieren, wird die Möglichkeit der Anpassung der Manifest-Datei geboten, um Anwendungs-Shortcuts zu realisieren. Jede Kontextmenü Interaktion ist dabei im Prinzip eine URL, welche innerhalb der Anwendung geöffnet wird. Wie bereits erwähnt ist es dabei notwendig, die Manifest-Datei der Softwarelösung anzupassen. Hierfür gibt es den optionalen Parameter ‚shortcuts“, welcher als ein Array mit Objekten für jeden Menüeintrag aufgebaut ist. Im Anschluss in der Abbildung 50 ist eine beispielhafte Konfiguration dargestellt (François Beaufort und Jungkee Song 2021).

```
"start_url": "https://player.fm?utm_source=homescreen",
…
"shortcuts": [
  {
    "name": "Open Play Later",
    "short_name": "Play Later",
    "description": "View the list of podcasts you saved for later",
    "url": "/play-later?utm_source=homescreen",
    "icons": [{ "src": "/icons/play-later.png", "sizes": "192x192" }]
  },
```

Abbildung 50: Manifest-Datei "shortcuts" (François Beaufort und Jungkee Song 2021)

Wie der Darstellung zu entnehmen ist, setzt sich ein Shortcut-Objekt aus mehreren Parametern zusammen, welche im Anschluss kurz betrachtet werden. Jedes Menü-Eintrag muss mindestens einen Namen und eine URL haben, um die Mindestanforderungen zu gewährleisten (François Beaufort und Jungkee Song 2021).

- name: Dies ist der dargestellte Text des einzelnen Menü-Eintrages.
- short_name: Sollte es aus Auflösungsgründen nicht möglich sein, den gesamten Namen des Menü-Eintrages darzustellen, dann wird dieser Parameter herangezogen.
- description: Dieser Parameter stellt die Beschreibung der Anwendung dar. Zum Zeitpunkt der Verfassung dieser Arbeit ist dieser nicht nutzbar, wurde aber für zukünftige Features bereits inkludiert.
- url: Das ist die Adresse, welche aufgerufen wird, wenn die Benutzerin bzw. der Benutzer den Menü-Eintrag auswählt. Wichtig dabei ist, dass sich die Url im Scope der Anwendung

befindet. Der Scope ist der Bereich, welcher unterschiedliche URL's beinhaltet, die der Browser als innerhalb der Anwendung ansieht. Somit kann der Browser erkennen, ob eine Adresse von außen oder innerhalb der Anwendung kommt. Wird dabei kein Scope angegeben, wird das Verzeichnis, in welchem die Manifest-Datei liegt, herangezogen. Des Weiteren kann man der Abbildung 50 entnehmen, dass es sich beim URL Parameter um einen relativen Pfad handelt und er somit in Kombination mit der „start_url" genutzt wird (Pete LePage et al. 2021).

- icons: Des Weiteren können optional mehrere Icons für verschiedene Auflösungen angegeben werden, welche mit einem Pfad und einer Auflösung definiert sind.

Zusätzlich muss die Reihenfolge der definierten Menü-Einträge beachtet werden. Der Grund hierfür ist, dass verschiedene Browser bzw. Plattformen eine variierende Anzahl an Einträgen darstellen und bei der Darstellung mit der erst-definierten Konfiguration gestartet wird. Dieses Feature kann nur beim mobilen Chrome und Samsung Internet Browser verwendet werden. Der mobile Safari kann in Bezug auf diese Funktion nicht genutzt werden (MDN Web Docs 2021s).

Tabelle 10 bietet einen Überblick über die oberhalb gewonnenen Informationen.

Qualitätsteilmerkmal	PWA			Native App	
	Safari	Chrome	Samsung	IOS	Android
Einheitliches Look & Feel					
Inter-app Sharing					
Schnellzugriffe per Kontextmenü					

Tabelle 10: Gegenüberstellung der Usability von PWAs und nativen Anwendungen

4.2 Erstellung der Entscheidungsmatrix

Im Anschluss werden die Ergebnisse der Evaluierungsphase mithilfe einer Entscheidungsmatrix zusammengefasst. Mithilfe dieser Entscheidungshilfe soll für Software-Entwicklungsprojekte die Möglichkeit geboten werden, auf Basis dieser eine Nutzwertanalyse durchzuführen. Die Nutzwertanalyse ist ein bekanntes Tool der qualitativen Analysemethoden der Entscheidungstheorie. Die Methodik dahinter soll die Beurteilung von Alternativen erleichtern und somit die Entscheidung auf wissenschaftlicher Basis mit bestmöglichem Outcome ermöglichen. Wie man der Abbildung 51 entnehmen kann, besteht der Prozess der Nutzwertanalyse im Wesentlichen aus acht Schritten. Da das Ziel der vorliegenden Arbeit die Bereitstellung einer Entscheidungshilfe bei mobiler Anwendungsentwicklung ist, wurden die ersten fünf Prozessschritte bereits für die zukünftigen Anwender und Anwenderinnen durchgeführt. Bereits zu Beginn der Arbeit wurden die Ziele definiert, welche die Effektivität und Effizienz der mobilen Anwendungsentwicklung im Fokus haben. Um diese in einem weiteren Schritt mit messbaren Kriterien zu versehen, wurden sie mithilfe der ISO 25010 mit dem FCM Modell auf messbare Sachverhalte heruntergebrochen. Darüber hinaus wurden die Alternativen der native Plattformen

iOS und Android wie auch der Ansatz rund um Progressive Web-Anwendungen definiert, wobei bei der letztgenannten Technologie der mobile Chrome, Safari und Samsung Internet Browser als Evaluierungsgrundlage definiert wurden. Nach der Definition des Kontextes wurde die Alternative im Anschluss anhand der Möglichkeiten und Einschränkungen evaluiert.

Ein weiterer Schritt der Nutzwertanalyse bildet das Transformieren der Werte der Evaluierung. Da die Kriterien bereits in der Definitionsphase auf dieselbe Skala (Rot, Gelb, Grün) reduziert wurden, mussten diese nicht mehr transformiert werden, um einen vergleichbaren Wertebereich zu schaffen. Nichtsdestotrotz sind die weiteren Schritte besonders abhängig vom Kontext des Entwicklungsprojektes. Aus diesem Grund wurden hier die notwendigen Mittel zur Vervollständigung der Nutzwertanalyse bereitgestellt, um eine global anwendbare Entscheidungshilfe zu schaffen (Carl Rauch und Andreas Rauber 2021).

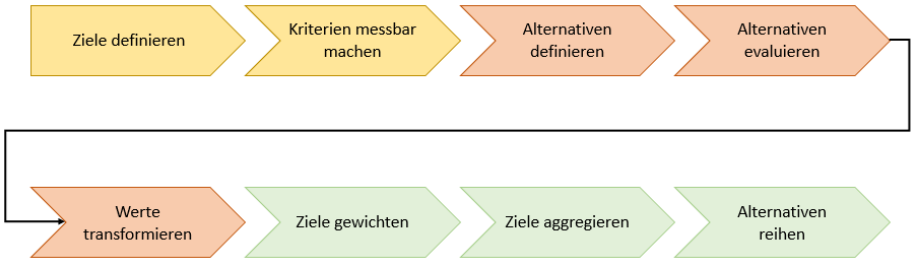

Abbildung 51: Schritte Nutzwertanalyse (Carl Rauch und Andreas Rauber 2021)

Wie bereits erwähnt wurden die ersten fünf Schritte bereits durchgeführt und in einer Entscheidungsmatrix zusammengefasst, welche in der Tabelle 12 ersichtlich ist. Die erste Spalte zeigt dabei die Qualitätsteilmerkmale, welche in der Kriterien-Definitionsphase definiert und daraufhin evaluiert wurden. Des Weiteren wurden die Auswertungen der Metriken von den Gegenüberstellungstabellen der einzelnen Qualitätsmerkmale übernommen und in den unterschiedlichen Technologieansätzen bzw. Browsern aufgesplittet. Diese wurden wiederrum mit den Farben Rot, Gelb und Grün gekennzeichnet. Kann ein Merkmal nur mit einer Einschränkung umgesetzt werden, ist es möglich, im jeweiligen Kapitel die Einschränkung mit der dazugehörigen Bezeichnung zu finden. Die zweite und dritte Spalte bilden dabei die KO-Kriterien und die Gewichtung der einzelnen Qualitätsteilmerkmale. Handelt es sich bei einer Anforderung um ein unabdingbares Merkmal der gewünschten Anwendung, stellt diese ein KO-Kriterium dar. Bei Nichterfüllung des Ansatzes kann die Alternative sofort aus der Entscheidungsfindung entfernt werden. Der nächste Schritt fokussiert sich auf die Bewertung der Kriterien. Hier soll die Relevanz der Merkmale einfließen, da sie nicht in jedem Projekt dieselbe Wertigkeit aufweisen können. Um eine Gewichtung durchzuführen, gibt es die unterschiedlichsten Konzepte in der Literatur. Diese Ansätze reichen von der Swing Methode über die Trade-off Methode bis hin zum Paarweisen Vergleich und mehr. Nichtsdestotrotz wird es nicht Teil der Arbeit sein, die passende Methode für die Bewertung zu finden. Diese kann individuell bei der Anwendung der Entscheidungsmatrix festgelegt werden. Um in Anschluss metrische Werte als Vergleichsbasis zu erhalten, werden die Zielwerte aggregiert. Die Tabelle 11 zeigt dabei eine demonstrative Berechnung der Teilnutzwerte (TNW) des Qualitätsteilmerkmals NFC. Hier wurde eine Gewichtung frei von einer Projektanforderung gewählt, da es sich nur um eine Verdeutlichung der Berechnung handelt. Darüber hinaus ist zu erwähnen, dass eine erfüllte

Anforderung mit 1 Punkt, eine Erfüllung mit Einschränkung mit 0,5 Punkte und eine nicht-erfüllte Anforderung mit 0 Punkte bewertet wurde.

Qualitätsteilmerkmal	KO	Gewichtung	PWA						Native App			
			Safari	TNW	Chrome	TNW	Samsung	TNW	IOS	TNW	Android	TNW
NFC	Nein	4		0	E3	2		0	E4 + E5	2		4

0 Punkte	0,5 Punkte	1 Punkt

Tabelle 11: Entscheidungsmatrix - Zielwert Aggregation

Multipliziert man die Gewichtung mit dem Erfüllungsgrad des Qualitätsteilmerkmales, dann erhält man die Teilnutzwerte. Wenn diese anhand aller Qualitätsteilmerkmale eines Technologieansatzes aufsummiert werden, erhält man den Gesamtnutzen der Alternativen. Die Formel wird im Anschluss dargestellt.

$$Gesamt\_Alternative\_A = \sum_{k=1}^{n} Gewichtung_n * Qualitätsteilmerkmal_n$$

Die einzelnen Nutzwert-Summen bilden die Entscheidungsgrundlage, um den letzten Schritt der Alternativen-Reihung gerecht zu werden und somit den optimalen Ansatz zu finden.

Qualitätsteilmerkmal	KO	Gewichtung	PWA						Native App			
			Safari	TNW	Chrome	TNW	Samsung	TNW	IOS	TNW	Android	TNW
Push Benachrichtigung												
Bluetooth					E1+E2		E1+E2					
NFC					E3				E4+E5			
Vibration												
Kamera und Mikrofon			E6									
Dateien Zugriff												
Geolocation												
Geofencing												
Device Motion							E7					
Fingerabdruck/Gesichtserkennung												
Gleichbleibende Energieverbrauch durch Service-Worker												
Energieverbrauch bei Interaktionen mit UI-Elementen unter 4 Joule												
Reaktionszeit Kamerazugriff unter 1 Sekunde			E1									
Reaktionszeit Standortzugriff unter 1 Sekunde												
Verfügbarkeit auf Webseiten											E2+E3	
Verfügbarkeit auf Google Play Store			E4		E4		E4					
Verfügbarkeit auf Apple App Store												
Aufruf über URL												
Automatische Updates									E2		E1	
Sicherheitsupdates									E2		E1	
Update vom Endbenutzer/in steuerbar												
Updates in Phasen veröffentlichen												
Bewertungen und Rezensionen			E5		E5		E5					
App-Performance Metriken			E6		E6		E6		E3		E4	
Sicherheitsprüfung der Installationsdatei												
Multi-Faktor Authentifizierung												
Plattformunabhängigkeit			E1		E1		E1					
Sicherstellung der Auf- und Abwärtskompatibilität			E2		E2		E2					
Verschiedene Versionen veröffentlichen			E5		E5		E5		E4		E3	
Offline-Datenhaltung												
Hintergrund-Synchronisation												
Einheitliches Look & Feel												
Inter-app sharing												
Quick Action Kontexmenü												
		Gesamt										

Funktionalität
Performance
Portierbarkeit
Wartbarkeit
Sicherheit
Kompatibilität
Verlässlichkeit
Usability

Tabelle 12: Entscheidungsmatrix

5 FAZIT UND AUSBLICK

Im Anschluss wird ein Fazit dieser Arbeit gezogen und es werden die wesentlichsten Erkenntnisse dieser Studie zusammengefasst. Des Weiteren wird ein Ausblick für weitere mögliche Forschungskontexte und Zukunftsszenarien diskutiert.

Wie die Einführung dieser Arbeit bereits dargestellt hat, heben sich die verbrachten Stunden in einer mobilen, installierten Anwendung von diesen in mobilen Browsern ab. Jedoch bedarf eine native Anwendung im herkömmlichen Sinne für jede Plattform, wie iOS und Android, eine eigene Quellcode-Basis, wobei als Produkt die Entwicklungs- und Wartungskosten für jedes Software-Projekt mehrmals anfallen. Progressive Web-Anwendungen sollen hier Abhilfe schaffen, weil diese die Eigenschaft der Plattformunabhängigkeit aufweisen und somit nur eine Quellcode-Basis für alle Plattformen erfordern. Dieser Technologieansatz, welcher in der Basis eine klassische Webseite darstellt und als Laufzeitumgebung einen Browser benutzt, verspricht ein natives Benutzererlebnis mit der Möglichkeit der Installation am Homescreen und der Offline-Fähigkeit.

In dieser Masterarbeit wurde analysiert, inwiefern es möglich ist, die nativen Entwicklungsansätze iOS und Android durch Progressive Web-Anwendungen zu ersetzen, ohne wesentliche Einschränkungen hinnehmen zu müssen. Um eine systematische und wissenschaftliche Grundlage dieser Evaluierung zu bilden, wurden als Basis die Qualitätsteilmerkmale der ISO 25010 Norm verwendet, welche für System und Software-Engineering – Qualitätskriterien und Bewertung von System und Softwareprodukten (SQuaRE) steht. Um einen Entscheidungsbehelf für alle mobilen Software-Projekte zu schaffen, wurde daraufhin eine Entscheidungsmatrix generiert, welche den Grundstein für eine Nutzwertanalyse legt. Beim Einsatz dieser Matrix muss lediglich die Relevanz der einzelnen Qualitätsmerkmale für das bestehende Softwareprojekt bewertet und im Anschluss die Teilnutzwerte der Technologieansätze aufsummiert werden.

Bei der Evaluierung der Qualitätsmerkmale der ISO 25010 wurden seitens Progressive Web-Anwendungen die mobilen Chrome, Samsung Internet und Safari Browser in Betracht gezogen, da diese im Jahre 2021 bereits knapp 97% aller verwendeten Browser einnehmen. Die Ergebnisse zeigen, dass vor allem der mobile Safari Browser Einschränkungen bei den Zugriffsfreiheiten der Hard- und Softwarekomponenten aufweist. Soll dabei auf spezifische Features, wie zum Beispiel Push Benachrichtigungen, Bluetooth, NFC und weitere zugegriffen werden, dann wird dies gänzlich von der Plattform untersagt. Bei den weiteren untersuchten Browsern ist ein viel zustimmender Trend in Richtung Progressive Web-Anwendungen erkennbar, da diese die Zugriffsfreiheiten auf vorhandene Gerätefeatures weitreichend erlauben. Vor allem der Chrome Browser bietet ein großes Spektrum an Berechtigungen. Dies kann auf den Fakt zurückgeführt werden, dass der Initiator von Progressive-Web Anwendungen Google ist. Darüber hinaus konnten bei der Evaluierung der Performance nur sehr unwesentliche Abweichungen vom nativen Ansatz erkannt werden. Nichtsdestotrotz konnte festgestellt werden, das Progressive Web-Anwendungen im Gegensatz zu den nativen Anwendungen nicht nur in den plattformspezifischen App-Stores bereitstellbar sind, sondern auch über den Browser, einer URL und im Google Play Store. Damit kann die Conversion Rate (Downloads pro Impressionen) im Vergleich zur nativen Anwendungsentwicklung erhöht werden. Seitens des Apple App Stores

kommt es hier erneut zu Einschränkungen. Darüber hinaus konnte bei der Plattformunabhängigkeit und der Wartbarkeit vor allem der browser-basierte Ansatz sehr positiv punkten, da nur eine Quellcode-Basis notwendig ist, wobei hier eine große Abhängigkeit des Anwendungsfalls besteht. Soll auf nicht-gestattete Features auf einem Apple Endgerät zugegriffen werden, kann die Plattformunabhängigkeit nicht in allen Softwareprojekten gewährleistet werden, da die Anwendung dann nicht für alle Systeme kompatibel ist. Bei der anschließenden Betrachtung eines intuitiven und gewohnten Benutzer-Erlebnisses bieten die nativen Ansätze Style-Guidelines und bereitgestellte Komponenten, wobei Progressive Web-Anwendung keiner spezifischen Plattform unterliegen und somit bei einem einheitlichen „Look & Feel“ etwas in den Schatten der Alternativen rücken. Grundsätzlich lässt sich jedoch zusammenfassen, dass Progressive Web-Anwendungen sich bereits je nach Entwicklungsprojekt auf Augenhöhe mit den nativen Varianten begegnen können. Um die substanziellen Projektanforderungen im Entscheidungsprozess einfließen zu lassen, bietet die erstellte Entscheidungsmatrix die Grundlage, um den bestmöglichen Entwicklungsansatz zu finden.

Vor allem bei der Betrachtung des Fortschritts von Progressive Web-Anwendungen in den letzten Jahren lässt sich sagen, dass dieser Bereich der Anwendungsentwicklung ein sehr dynamischer ist und sich bei Aktualisierungen von Browser-Anwendungen kontinuierlich neue Wege öffnen können. Besonders interessant wäre es, bei zukünftigen Forschungsprojekt die Möglichkeiten von Progressive Web-Anwendungen im Bereich der Desktop-Anwendungen zu identifizieren. Aufgrund der Tatsache, dass Anwendungen dieses Konzeptes bereits im Microsoft App Store veröffentlichbar sind, kann hieraus ein hohes Potential an neuen Chancen resultieren.

ABKÜRZUNGSVERZEICHNIS

PWA..... Progressive Web App

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

LITERATURVERZEICHNIS

Abhi Gambhir, Gaurav Raj (Hg.) (2018): Analysis of Cache in Service Worker and Performance Scoring of Progressive Web Application(ICACCE).

Amazon AWS (2021): Amazon Simple Notification Service. Hg. v. Amazon AWS. Online verfügbar unter https://aws.amazon.com/de/sns/?whats-new-cards.sort-by=item.additionalFields.postDateTime&whats-new-cards.sort-order=desc.

Android Developers (2021a): <manifest>. Online verfügbar unter https://developer.android.com/guide/topics/manifest/manifest-element.

Android Developers (2021b): <uses-sdk>. Online verfügbar unter https://developer.android.com/guide/topics/manifest/uses-sdk-element#ApiLevels.

Android Developers (2021c): Access app-specific files. Online verfügbar unter https://developer.android.com/training/data-storage/app-specific.

Android Developers (2021d): App data and files. Online verfügbar unter https://developer.android.com/guide/topics/data.

Android Developers (2021e): App shortcuts overview. Online verfügbar unter https://developer.android.com/guide/topics/ui/shortcuts.

Android Developers (2021f): Audio & Video. Online verfügbar unter https://developer.android.com/guide/topics/media.

Android Developers (2021g): Bluetooth overview. Online verfügbar unter https://developer.android.com/guide/topics/connectivity/bluetooth.

Android Developers (2021h): Camera. Online verfügbar unter https://developer.android.com/training/camera.

Android Developers (2021i): Create a sync adapter. Online verfügbar unter https://developer.android.com/training/sync-adapters/creating-sync-adapter.

Android Developers (2021j): Create and monitor geofences. Online verfügbar unter https://developer.android.com/training/location/geofencing.

Android Developers (2021k): Create shortcuts. Online verfügbar unter https://developer.android.com/guide/topics/ui/shortcuts/creating-shortcuts.

Android Developers (2021l): Data and file storage overview. Online verfügbar unter https://developer.android.com/training/data-storage.

Android Developers (2021m): In-app updates. Online verfügbar unter https://developer.android.com/guide/playcore/in-app-updates.

Android Developers (2021n): Intent. Online verfügbar unter https://developer.android.com/reference/android/content/Intent.

Android Developers (2021o): Material Design for Android. Online verfügbar unter https://developer.android.com/guide/topics/ui/look-and-feel.

Android Developers (2021p): Motion sensors. Online verfügbar unter https://developer.android.com/guide/topics/sensors/sensors_motion.

Android Developers (2021q): Multiple APK support. Online verfügbar unter https://developer.android.com/google/play/publishing/multiple-apks.

Android Developers (2021r): Near field communication overview. Online verfügbar unter https://developer.android.com/guide/topics/connectivity/nfc.

Android Developers (2021s): Overview of shared storage. Online verfügbar unter https://developer.android.com/training/data-storage/shared.

Android Developers (2021t): Publish your app. Online verfügbar unter https://developer.android.com/studio/publish.

Android Developers (2021u): Publish your app. Online verfügbar unter https://developer.android.com/studio/publish.

Android Developers (2021v): Receiving simple data from other apps. Online verfügbar unter https://developer.android.com/training/sharing/receive.

Android Developers (2021w): Save data in a local database using Room. Online verfügbar unter https://developer.android.com/training/data-storage/room.

Android Developers (2021x): Save key-value data. Online verfügbar unter https://developer.android.com/training/data-storage/shared-preferences.

Android Developers (2021y): Sending simple data to other apps. Online verfügbar unter https://developer.android.com/training/sharing/send.

Android Developers (2021z): Show a biometric authentication dialog. Online verfügbar unter https://developer.android.com/training/sign-in/biometric-auth.

Android Developers (2021aa): Version your App. Online verfügbar unter https://developer.android.com/studio/publish/versioning.

Android Developers (2021ab): Vibrator. Online verfügbar unter https://developer.android.com/reference/android/os/Vibrator.

App Store Connect Help (2021a): Add internal testers. Online verfügbar unter https://help.apple.com/app-store-connect/#/dev839fb66e9?sub=devfefc41353.

App Store Connect Help (2021b): App metrics. Online verfügbar unter https://help.apple.com/app-store-connect/#/itc21781223f.

App Store Connect Help (2021c): Create a new version. Online verfügbar unter https://help.apple.com/app-store-connect/#/dev480217e79.

App Store Connect Help (2021d): Invite external testers. Online verfügbar unter https://help.apple.com/app-store-connect/#/dev859139543?sub=dev6fe14bffe.

App Store Connect Help (2021e): Manually release a version. Online verfügbar unter https://help.apple.com/app-store-connect/#/devabbb787a6.

App Store Connect Help (2021f): Release a version update in phases. Online verfügbar unter https://help.apple.com/app-store-connect/#/dev3d65fcee1.

App Store Connect Help (2021g): View app metrics. Online verfügbar unter https://help.apple.com/app-store-connect/#/itc14b94d665.

Apple (2021): Building a Trusted Building a Trusted Ecosystem for Millions of Apps. Online verfügbar unter https://www.apple.com/privacy/docs/Building_a_Trusted_Ecosystem_for_Millions_of_Apps_A_Threat_Analysis_of_Sideloading.pdf.

Apple Developer (2021a): APNs Overview. Hg. v. Apple. Online verfügbar unter https://developer.apple.com/library/archive/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/APNSOverview.html#//apple_ref/doc/uid/TP40008194-CH8-SW1.

Apple Developer (2021b): App Store Review Guidelines. Online verfügbar unter https://developer.apple.com/app-store/review/guidelines/.

Apple Developer (2021c): App Store Review Guidelines. Online verfügbar unter https://developer.apple.com/app-store/review/guidelines/.

Apple Developer (2021d): Authentication. Online verfügbar unter https://developer.apple.com/design/human-interface-guidelines/ios/user-interaction/authentication/.

Apple Developer (2021e): Background Tasks. Online verfügbar unter https://developer.apple.com/documentation/backgroundtasks.

Apple Developer (2021f): Beta Testing Made Simple with TestFlight. Online verfügbar unter https://developer.apple.com/testflight/.

Apple Developer (2021g): Cameras and Media Capture. Online verfügbar unter https://developer.apple.com/documentation/avfoundation/cameras_and_media_capture.

Apple Developer (2021h): Choosing Background Strategies for Your App. Online verfügbar unter https://developer.apple.com/documentation/backgroundtasks/choosing_background_strategies_for_your_app.

Apple Developer (2021i): Core Bluetooth. Online verfügbar unter https://developer.apple.com/documentation/corebluetooth.

Apple Developer (2021j): Core Data. Online verfügbar unter https://developer.apple.com/documentation/coredata.

Apple Developer (2021k): Core Haptics. Online verfügbar unter https://developer.apple.com/documentation/corehaptics.

Apple Developer (2021l): Core Location. Online verfügbar unter https://developer.apple.com/documentation/corelocation.

Apple Developer (2021m): Core Motion. Online verfügbar unter https://developer.apple.com/documentation/coremotion.

Apple Developer (2021n): Core NFC. Online verfügbar unter https://developer.apple.com/documentation/corenfc.

Apple Developer (2021o): Keychain Services. Online verfügbar unter https://developer.apple.com/documentation/security/keychain_services.

Apple Developer (2021p): Monitoring the User's Proximity to Geographic Regions. Online verfügbar unter https://developer.apple.com/documentation/corelocation/monitoring_the_user_s_proximity_to_geographic_regions.

Apple Developer (2021q): NSPersistentStore. Online verfügbar unter https://developer.apple.com/documentation/coredata/nspersistentstore/.

Apple Developer (2021r): NSUserDefaults. Online verfügbar unter https://developer.apple.com/documentation/foundation/nsuserdefaults.

Apple Developer (2021s): Property list. Online verfügbar unter https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/PropertyList.html.

Apple Developer (2021t): Providing Access to Directories. Online verfügbar unter https://developer.apple.com/documentation/uikit/view_controllers/providing_access_to_directories.

Apple Developer (2021u): SDK Compatibility Guide. Online verfügbar unter https://developer.apple.com/library/archive/documentation/DeveloperTools/Conceptual/cross_development/Configuring/configuring.html.

Apple Developer (2021v): Share. Online verfügbar unter https://developer.apple.com/library/archive/documentation/General/Conceptual/ExtensibilityPG/Share.html#//apple_ref/doc/uid/TP40014214-CH12-SW1.

Apple Developer (2021w): UIKit. Online verfügbar unter https://developer.apple.com/documentation/uikit.

Apple Developer (2021x): When should I use a wildcard App ID? Online verfügbar unter https://developer.apple.com/library/archive/qa/qa1713/_index.html.

B. W. Boehm, J. R. Brown, M. Lipow (1978): Characteristics of Software Quality. Hg. v. North Holland Publishing.

Biørn-Hansen, Andreas; Grønli, Tor-Morten; Ghinea, Gheorghita; Alouneh, Sahel; Canfora, Gerardo (2019): An Empirical Study of Cross-Platform Mobile Development in Industry. In: *Wireless Communications and Mobile Computing* 2019, S. 5743892. DOI: 10.1155/2019/5743892.

Bisdikian, C. (2001): An overview of the Bluetooth wireless technology. In: *IEEE Communications Magazine* 39 (12), S. 86–94. DOI: 10.1109/35.968817.

Can I use (2021a): Geolocation API. Online verfügbar unter https://caniuse.com/?search=geolocation%20api.

Can I use (2021b): IndexedDB. Online verfügbar unter https://caniuse.com/?search=indexeddb.

Can I use (2021c): Service Workers. Online verfügbar unter https://caniuse.com/serviceworkers.

Can I use (2021d): Web NFC. Online verfügbar unter https://caniuse.com/webnfc.

Carl Rauch; Andreas Rauber (2021): Anwendung der Nutzwertanalyse zur Bewertung Anwendung der Nutzwertanalyse zur Bewertung von Strategien zur langfristigen Erhaltung digitaler Objekte.

Chen, Chen; Tock, Yoav; Girdzijauskas, Sarunas (2018): BeaConvey: Co-Design of Overlay and Routing for Topic-Based Publish/Subscribe on Small-World Networks. In: Proceedings of the 12th ACM International Conference on Distributed and Event-Based Systems. New York, NY, USA: Association for Computing Machinery (DEBS '18), S. 64–75.

Chrome Developer (2021): Trusted Web Activities. Online verfügbar unter https://developer.chrome.com/docs/android/trusted-web-activity/.

Daniel Nations (2021): What Is AirDrop? How Does It Work? Online verfügbar unter https://www.lifewire.com/what-is-airdrop-how-does-it-work-1994512.

Debnath Bhattacharyya; Rahul Ranjan; Farkhod Alisherov A.; Minkyu Choi (2009): Biometric Authentication: A Review. Korea.

designflyover (2021): Understanding the iOS and Android UI Guidelines. Online verfügbar unter https://designflyover.com/understanding-the-ios-and-android-ui-guidelines/.

Droid Wiki (2021): Android package (APK) Datei. Online verfügbar unter https://www.droidwiki.org/wiki/Apk.

Fabio Pelizzola (2020): iOS Share extension — Swift 5.1. Online verfügbar unter https://medium.com/macoclock/ios-share-extension-swift-5-1-1606263746b.

fileformat (2021): What is a IPA file? Online verfügbar unter https://docs.fileformat.com/executable/ipa/.

François Beaufort; Jungkee Song (2021): Get things done quickly with app shortcuts. Online verfügbar unter https://web.dev/app-shortcuts/.

Ge Gao (2016): Android Applications, Can You Trust Google Play on These.

Golmie, N.; Rebala, O.; Chevrollier, N. (2003): Bluetooth adaptive frequency hopping and scheduling. In: IEEE Military Communications Conference, 2003. MILCOM 2003, Bd. 2, 1138-1142 Vol.2.

Gomez, Carles; Oller, Joaquim; Paradells, Josep (2012): Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology. In: *Sensors* 12 (9), S. 11734–11753. DOI: 10.3390/s120911734.

Google (2021): Verarbeitung von Informationen in der Google-Suche. Online verfügbar unter https://www.google.com/intl/de/search/howsearchworks/crawling-indexing/.

Google Developers (2021a): Introduction to service worker. Online verfügbar unter https://developers.google.com/web/ilt/pwa/introduction-to-service-worker.

Google Developers (2021b): Live Data in the Service Worker. Online verfügbar unter https://developers.google.com/web/ilt/pwa/live-data-in-the-service-worker.

Google Developers (2021c): Offline Google Analytics Made Easy. Online verfügbar unter https://developers.google.com/web/updates/2016/07/offline-google-analytics#whats_going_on_under_the_hood.

Google Developers (2021d): Standortdaten. Online verfügbar unter https://developers.google.com/maps/documentation/android-sdk/location?hl=de.

Google Developers (2021e): Statement List Generator and Tester. Online verfügbar unter https://developers.google.com/digital-asset-links/tools/generator.

Google Developers (2021f): The App Shell Model. Online verfügbar unter https://developers.google.com/web/fundamentals/architecture/app-shell.

Google Developers (2021g): Using the Cache API in the service worker. Online verfügbar unter https://developers.google.com/web/ilt/pwa/caching-files-with-service-worker.

Google Firebase (2021): Firebase Cloud-Messaging. Hg. v. Google. Online verfügbar unter https://firebase.google.com/docs/cloud-messaging.

Google Github (2021): Web NFC Sample. Online verfügbar unter https://googlechrome.github.io/samples/web-nfc/.

Google Support (2021a): Android-Apps aktualisieren. Online verfügbar unter https://support.google.com/googleplay/answer/113412?hl=de.

Google Support (2021b): App aktualisieren oder Veröffentlichung aufheben. Online verfügbar unter https://support.google.com/googleplay/android-developer/answer/9859350?hl=de&ref_topic=7072031#zippy=%2Czertifizierung-eines-app-bundles-pr%C3%BCfen.

Google Support (2021c): App-Statistiken ansehen. Online verfügbar unter https://support.google.com/googleplay/android-developer/answer/139628?hl=de&co=GENIE.Platform%3DDesktop&oco=1.

Google Support (2021d): App-Updates mit gestaffelten Roll-outs veröffentlichen. Online verfügbar unter https://support.google.com/googleplay/android-developer/answer/6346149#staged_country&zippy=%2Cgestaffelten-roll-out-in-bestimmten-l%C3%A4ndern-vornehmen.

Google Support (2021e): Bewertungen und Rezensionen Ihrer App ansehen und analysieren. Online verfügbar unter https://support.google.com/googleplay/android-developer/answer/138230?hl=de#zippy=%2Creview-highlights-see-popular-themes-in-your-apps-reviews.

Google Support (2021f): Progressive Web-Apps verwenden. Online verfügbar unter https://support.google.com/chrome/answer/9658361?hl=de&co=GENIE.Platform%3DiOS.

Google Web (2021): The Service Worker Lifecycle. Online verfügbar unter https://developers.google.com/web/fundamentals/primers/service-workers/lifecycle#avoid-url-change.

Google web.dev (2021): Progressive Web Apps. Online verfügbar unter https://web.dev/progressive-web-apps/.

Huber, Stefan; Demetz, Lukas; Felderer, Michael (2021): PWA vs the Others: A Comparative Study on the UI Energy-Efficiency of Progressive Web Apps. In: Marco Brambilla, Richard Chbeir, Flavius Frasincar und Ioana Manolescu (Hg.): Web Engineering. Cham, 2021. Cham: Springer International Publishing, S. 464–479.

Hubert, Richard (2002): Plattformunabhängige Softwareentwicklung. In: Gerhard Versteegen (Hg.): Software Management: Beherrschung des Lifecycles. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 83–97.

Ionos (2021): Eine iOS-App veröffentlichen. Online verfügbar unter https://www.ionos.at/digitalguide/websites/web-entwicklung/die-eigene-app-entwickeln-eine-ios-app-veroeffentlichen/.

ISO/IEC 25010:2011, 2011: ISO/IEC 25010:2011.

Ivano Malavolta; Giuseppe Procaccianti; Paul Noorland; Petar Vukmirovic (2017): Assessing the Impact of Service Workers on the Energy Efficiency of Progressive Web Apps, Vrije Universiteit Amsterdam.

Jake Archibald (2021): The Offline Cookbook. Online verfügbar unter https://web.dev/offline-cookbook/.

Jan vom Brocke; Alexander Simons; Björn Niehaves; Kai Riemer; Ralf Plattfaut; Anne Cleven (2009): Reconstructing the giant: On the importance of rigour in documenting the literature search process. Liechtenstein: ECIS.

Joe Medley (2021): Integrate with the OS sharing UI with the Web Share API. Hg. v. web.dev. Online verfügbar unter https://web.dev/web-share/.

Joe Medley; Pete LePage (2019): Receiving shared data with the Web Share Target API. Hg. v. web.dev. Online verfügbar unter https://web.dev/web-share-target/.

Karthikeyan T (2020): Quick Actions in iOS. Online verfügbar unter https://medium.com/ivymobility-developers/quick-actions-in-ios-8b8f182124e4.

Khandeparkar Anmol, Gupta Rashmi, B. Sindhya (2015): An Introduction to Hybrid Platform Mobile Application Development.

L. Ceci (2021): Number of apps available in leading app stores 2021. Hg. v. statista. Online verfügbar unter https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/.

Lawrence, Tavakol (2007): Website Hosting and Website Management. In: Dave Lawrence und Soheyla Tavakol (Hg.): Balanced Website Design. London: Springer London.

Lewis Shaun; Dunn Mike (2020): Native mobile development. A cross-reference for iOS and Android. First edition. Sebastopol, CA: O'Reilly Media, Inc.

Martin Schierle (2021): The Superpowers of The Web and Native Apps Combined. Online verfügbar unter https://pwa-book.awwwards.com.

Material Design (2021a): Components. Online verfügbar unter https://material.io/components?platform=android.

Material Design (2021b): Introduction. Online verfügbar unter https://material.io/design/introduction.

MDN Web Docs (2021a): @media. Online verfügbar unter https://developer.mozilla.org/de/docs/Web/CSS/@media.

MDN Web Docs (2021b): @supports. Online verfügbar unter https://developer.mozilla.org/en-US/docs/Web/CSS/@supports.

MDN Web Docs (2021c): Accelerometer. Online verfügbar unter https://developer.mozilla.org/en-US/docs/Web/API/Accelerometer.

MDN Web Docs (2021d): Background Synchronization API. Online verfügbar unter https://developer.mozilla.org/en-US/docs/Web/API/Background_Synchronization_API.

MDN Web Docs (2021e): Bluetooth.getDevices. Hg. v. Mozilla. Online verfügbar unter https://developer.mozilla.org/en-US/docs/Web/API/Bluetooth/getDevices.

MDN Web Docs (2021f): Cache. Online verfügbar unter https://developer.mozilla.org/en-US/docs/Web/API/Cache.

MDN Web Docs (2021g): CacheStorage. Online verfügbar unter https://developer.mozilla.org/en-US/docs/Web/API/CacheStorage.

MDN Web Docs (2021h): Gyroscope. Online verfügbar unter https://developer.mozilla.org/en-US/docs/Web/API/Gyroscope.

MDN Web Docs (2021i): i18n. Online verfügbar unter https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/i18n.

MDN Web Docs (2021j): ImageCapture. Online verfügbar unter https://developer.mozilla.org/en-US/docs/Web/API/ImageCapture.

MDN Web Docs (2021k): Implementing feature detection. Online verfügbar unter https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Cross_browser_testing/Feature_detection.

MDN Web Docs (2021l): IndexedDB. Online verfügbar unter https://developer.mozilla.org/de/docs/Web/API/IndexedDB_API.

MDN Web Docs (2021m): Introduction to progressive web apps. Hg. v. MDN Web Docs. Online verfügbar unter https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Introduction.

MDN Web Docs (2021n): Magnetometer. Online verfügbar unter https://developer.mozilla.org/en-US/docs/Web/API/Magnetometer.

MDN Web Docs (2021o): Media Recorder. Online verfügbar unter https://developer.mozilla.org/en-US/docs/Web/API/MediaRecorder.

MDN Web Docs (2021p): Navigator. Online verfügbar unter https://developer.mozilla.org/de/docs/Web/API/Navigator.

MDN Web Docs (2021q): Navigator.share. Online verfügbar unter https://developer.mozilla.org/en-US/docs/Web/API/Navigator/share.

MDN Web Docs (2021r): Push API. Hg. v. Mozilla. Online verfügbar unter https://developer.mozilla.org/de/docs/Web/API/Push_API.

MDN Web Docs (2021s): shortcuts. Online verfügbar unter https://developer.mozilla.org/en-US/docs/Web/Manifest/shortcuts.

MDN Web Docs (2021t): Vibration API. Online verfügbar unter https://developer.mozilla.org/en-US/docs/Web/API/Vibration_API.

MDN Web Docs (2021u): Web Authentication API. Online verfügbar unter https://developer.mozilla.org/en-US/docs/Web/API/Web_Authentication_API.

MDN Web Docs (2021v): Web Bluetooth API. Hg. v. Mozilla. Online verfügbar unter https://developer.mozilla.org/en-US/docs/Web/API/Web_Bluetooth_API.

MDN Web Docs (2021w): Window. Online verfügbar unter https://developer.mozilla.org/de/docs/Web/API/Window.

Michael Kende (1994): A note on backward compatibility. In: *Economics Letters* 45 (3), S. 385–389. DOI: 10.1016/0165-1765(94)90042-6.

Mozilla (2021a): Service Worker API. Online verfügbar unter https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API.

Mozilla (2021b): Web App Manifest. Online verfügbar unter https://developer.mozilla.org/de/docs/Web/Manifest.

Nishihara, Saki; Aoki, Takuya; Ebihara, Tadashi; Mizutani, Koichi; Wakatsuki, Naoto (2018): Software Modem for Secure Near-field Communication for Smartphones Using Sound and Vibration. In: 2018 Eleventh International Conference on Mobile Computing and Ubiquitous Network (ICMU), S. 1–2.

NVISO Lab (2021): Deep dive into the security of Progressive Web Apps. Online verfügbar unter https://blog.nviso.eu/2020/01/16/deep-dive-into-the-security-of-progressive-web-apps/.

Olli Särkkä; Tuukka Nieminen; Saku Suuriniemi; Lauri Kettunen (2021): A multi-position calibration method for consumer-grade accelerometers, gyroscopes, and magnetometers to field conditions.

Pete LePage; François Beaufort; Thomas Steiner (2021): Add a web app manifest. Online verfügbar unter https://web.dev/add-manifest/#scope.

Platon, Kotzias; Juan, Caballero; Leyla, Bilge (2020): How Did That Get In My Phone? Unwanted App Distribution on Android Devices.

PWA.BAR (2021): PWA BAR Home. Online verfügbar unter https://pwa.bar/.

Rebecca Fransson, Alexandre Driaguine (2017): Comparing Progressive Web Applications with Native Android Applications. an evaluation of performance when it comes to response time.

Reclus, Fabrice; Drouard, Kristen (2009): Geofencing for fleet amp; freight management. In: 2009 9th International Conference on Intelligent Transport Systems Telecommunications, (ITST), S. 353–356.

Rossi, Gustavo (2008): Web engineering. Modelling and implementing web applications. London: Springer (Human-computer interaction series, 12).

S. O'Dea (2021a): Market share of mobile operating systems worldwide 2012-2021. Online verfügbar unter https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/.

S. O'Dea (2021b): Mobile broadband internet subscription rate 2020, by region. statista. Online verfügbar unter https://www.statista.com/statistics/370694/mobile-broadband-internet-penetration-region/.

Sangani, K. (2011): Computer says no [software compatibility]. In: *Engineering & Technology* 6 (9), S. 76–77. DOI: 10.1049/et.2011.0913.

selfhtml (2021): JavaScript/Web Storage. Online verfügbar unter https://wiki.selfhtml.org/wiki/JavaScript/Web_Storage.

Semantic Versioning (2021): Semantic Versioning 2.0.0. Online verfügbar unter https://semver.org/lang/de/.

Stackoverflow (2021): Can I force an iPhone user to upgrade an application? Online verfügbar unter https://stackoverflow.com/questions/2221436/can-i-force-an-iphone-user-to-upgrade-an-application.

Statcounter (2021): Mobile Browser Market Share Worldwide - June 2021. Online verfügbar unter https://gs.statcounter.com/browser-market-share/mobile/worldwide.

statista (2020a): Google Play: number of available apps 2009-2020 Published by Statista Research Department, Feb 4, 2021 How many apps are there in the Play Store? The number of available apps in the Google Play Store was most recently placed at 3.04 million apps, after surpassing 1 million apps in July 2013. Google Play Store Google Play was originally launched in October 2008 under the name Android Market. As Google’s official app store, it offers its customers a wide range of applications and digital media including music, magazines, books, film, and TV. As of June 2020, the ranking of the top-grossing Android apps worldwide is dominated by mobile games with Coin Master, Garena Free Fire: Rampage, and PUBG Mobile leading the ranking. This hardly comes as a surprise as it is estimated that gaming apps account for over 80 percent of Google Play app revenues. As most apps in the Google Play Store are available for free, the company needs to utilize effective business models to secure healthy revenue. How do apps make money? Despite the robust gaming app revenues, most gaming apps are free to download and rely on monetization via in-app advertising or in-game purchases of items such as boosters and accessories. An October 2019 survey of mobile app publishers found that video ads were

the most popular mobile app monetization model, followed by in-app purchases and display ads. According to app developers, the biggest challenges when monetizing apps was ensuring that the ads are not too intrusive to the content and that app monetization does not affects the app experience. Read more Number of available applications in the Google Play Store from December 2009 to December 2020. Online verfügbar unter (https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/.

statista (2020b): Number of active apps from the Apple App Store 2008-2020 Published by Statista Research Department, Feb 4, 2021 In 2020, the App Store offered 957,390 gaming apps. In contrast, as of that period of time the store had 3.42 million non-gaming apps available. As of June 2017, 180 billion apps had been downloaded from Apple App Store. Apple App Store The Apple App Store opened on July 10, 2008 via an update to iTunes. This coincided with Apple’s launch of the second-generation iPhone 3G which supported mobile apps. Applications in the App Store are only available for iOS devices - the tech industry frequently refers to Apple's ecosystem of devices and operating systems as a "walled garden". As of 2020, the most popular Apple App Store category was gaming with over 22 percent of apps being games. The second most popular app category was business, followed by education, and lifestyle apps. Mobile gaming apps usually generate the majority of their revenue through advertising and in-app purchases with only little more than a third of games being paid downloads. Popular gaming apps in the App Store include Clash Royale, Candy Crush Saga, casino games and Clash of Clans, the currently top grossing iOS game. Read more Number of available apps in the Apple App Store from 2008 to 2020. Online verfügbar unter https://www.statista.com/statistics/268251/number-of-apps-in-the-itunes-app-store-since-2008/.

Stephan Spitz (2021): Mobile Multifaktor – Authentisierung.

W3 Techs (2021): Usage statistics and market share of Google Analytics for websites. Online verfügbar unter https://w3techs.com/technologies/details/ta-googleanalytics.

wandera (2021): Trojan malware infecting 17 apps on the App Store. Online verfügbar unter https://www.wandera.com/ios-trojan-malware/.

Warren, Ian; Meads, Andrew; Srirama, Satish; Weerasinghe, Thiranjith; Paniagua, Carlos (2014): Push Notification Mechanisms for Pervasive Smartphone Applications. In: *IEEE Pervasive Computing* 13 (2), S. 61–71. DOI: 10.1109/MPRV.2014.34.

WHAT WEB CAN DO TODAY? (2021a): Audio & Video Capture. Online verfügbar unter https://whatwebcando.today/camera-microphone.html.

WHAT WEB CAN DO TODAY? (2021b): Device Motion. Online verfügbar unter https://whatwebcando.today/device-motion.html.

WHAT WEB CAN DO TODAY? (2021c): File Access. Online verfügbar unter https://whatwebcando.today/files.html.

WHAT WEB CAN DO TODAY? (2021d): Geofencing. Online verfügbar unter https://whatwebcando.today/geofencing.html.

WHAT WEB CAN DO TODAY? (2021e): Handling Service Worker updates – how to keep the app updated and stay sane. Online verfügbar unter https://whatwebcando.today/articles/handling-service-worker-updates/.

WHAT WEB CAN DO TODAY? (2021f): WHAT WEB CAN DO TODAY? Online verfügbar unter https://whatwebcando.today.

Wolfgang W. Osterhage (Hg.) (2018): Near Field Communication. Sicherheitskonzepte in der mobilen Kommunikation : Drahtlose Kommunikation - Protokolle und Gefahren. Berlin, Heidelberg: Springer Berlin Heidelberg.

XCode Help (2021a): asset catalog. Online verfügbar unter https://help.apple.com/xcode/mac/current/#/dev8f229268b.

XCode Help (2021b): Create asset catalogs and sets. Online verfügbar unter https://help.apple.com/xcode/mac/current/index.html?localePath=en.lproj#/dev10510b1f7.

XCode Help (2021c): deployment target. Online verfügbar unter https://help.apple.com/xcode/mac/current/#/dev110756139.

XCode Help (2021d): Edit deployment info settings. Online verfügbar unter https://help.apple.com/xcode/mac/current/#/deve69552ee5.

XCode Help (2021e): Set the version number and build string. Online verfügbar unter https://help.apple.com/xcode/mac/current/#/devba7f53ad4.

XCode Help (2021f): What is app thinning? (iOS, tvOS, watchOS). Online verfügbar unter https://help.apple.com/xcode/mac/current/#/devbbdc5ce4f.

Yong Cui (2020): Create Dynamic Custom Home Screen Quick Actions for Your iOS Apps Using SwiftUI. Online verfügbar unter https://medium.com/swlh/create-dynamic-custom-home-screen-quick-actions-for-your-ios-apps-using-swiftui-9e6f9b19c8db.

Yoram Wurmser (2020): The Majority of Americans’ Mobile Time Spent Takes Place in Apps. Online verfügbar unter https://www.emarketer.com/content/the-majority-of-americans-mobile-time-spent-takes-place-in-apps.