Bachelor-Thesis

Bachelor of Science Wirtschaftsinformatik

# Analyse der User Experience beim Vergleich von Progressive Web Apps und nativen Apps


| | |
|---|---|
| Name: | Luca Weber |
| Anschrift: | [redacted] |
| Matr.-Nr.: | [redacted] |
| Fakultät: | III |
| Erstprüfer: | Prof. Dr. Gunnar Stevens |
| Zweitprüfer: | Prof. Dr. Volker Wulf |
| Lehrstuhl: | Wirtschaftsinformatik und Neue Medien |
| Ort, Datum: | Siegen, den 05.07.2021 |

# Abstract

*Deutsch:*

Die Entwicklung von Software verlagert sich immer weiter von klassischen Computerprogrammen zur Software von Smartphones – Apps. Diese Arbeit behandelt die Entwicklung einer Progressive Web App, eine mit anderen Technologien vergleichbar neue Möglichkeit des Cross-Platform Developments, die die Vorteile von nativen Apps und Web Apps vereint. Progressive Web Apps sind klassische Web Apps, die mit der Hilfe verschiedener Frameworks und Metadaten eine deutliche Weiterentwicklung der Web Apps darstellen, da sie lauffähig auf einem Smartphone werden und möglicherweise das Potenzial haben, zu einer Konkurrenz gegenüber nativen Apps zu werden. Um neben der Entwicklung einen Fokus auf die User Experience von Progressive Web Apps zu legen untersucht diese Arbeit die User Experience von Progressive Web Apps im Vergleich mit einer nativen App anhand einer selbst entwickelten prototypischen Progressive Web App als Abbild einer nativen App. Mit der Durchführung eines Thinking Aloud Prozesses an zwei Fokusgruppen, sowie einer Umfrage konnten die Erfahrungen der Nutzer untersucht, und daraus die empfundene User Experience abgeleitet und verglichen werden.

*English:*

The development of software is shifting more and more from classic computer programs to the software for smartphones - apps. This thesis examines the development of a Progressive Web App, a new possibility of cross-platform development, which combines the advantages of native apps and web apps. Progressive Web Apps are classic web apps that use the help of different frameworks and data, to significantly evolve the development of web apps and to possibly compete with native apps as standard software for smartphones. Additionally to the development of a Progressive Web App, this thesis examines the user experience of progressive web apps in comparison with a native app using a self-developed prototypical Progressive Web App as a copy of a native app. With the execution of a Thinking Aloud process and a survey in two focus groups, the experiences of the users were examined so that the perceived user experience could be derived and compared.

*Stichworte / Keywords:* Progressive Web Apps, PWA, Cross-Platform Development, Native Apps User Experience, Thinking Aloud

*Gender-Disclaimer:* Zur besseren Lesbarkeit wird in der vorliegenden Arbeit auf die gleichzeitige Verwendung männlicher und weiblicher Sprachformen verzichtet. Es wird das generische Maskulinum verwendet, wobei beide Geschlechter gleichermaßen gemeint sind.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Abkürzungsverzeichnis

| | | |
|---|---|---|
| SDK | - | Software Development Kit |
| HTML | - | Hypertext Markup Language |
| CSS | - | Cascading Style Sheets |
| API | - | Application Programming Interface |
| JSON | - | JavaScript Object Notation |
| HTTP | - | Hypertext Transfer Protocol |
| HTTPS | - | Hypertext Transfer Protocol Secure |
| PWA | - | Progressive Web App |
| IDE | - | Integrated Development Environment |
| URL | - | Uniform Resource Locator |
| TLS | - | Transport Layer Security |
| DOM | - | Document Object Model |
| VAPID | - | Voluntary Application Server Authentication |
| BLOB | - | Binary Large Object |
| NPM | - | Node Package Manager |
| ACME | - | Automatic Certificate Management Environment |

# 1. Einleitung

In den letzten Jahren hat die Verwendung von Smartphones und anderen mobilen Endgeräten stark zugenommen [1, 2]. Der digitale Konsum hat sich verschoben. Es werden immer weniger „standalone“ Computer benötigt, um am digitalen Leben teilzunehmen. Arbeit, Kommunikation und Unterhaltung läuft mehr über Smartphones und Tablets [3]. Die daraus resultierende Veränderung der verwendeten Hardware bringt auch eine Veränderung der verwendeten Software mit sich. Als Resultat geht die Verwendung von Computerprogrammen stetig über zur Software von Smartphones - Apps. Diese haben einen drastischen Aufschwung erhalten [4]. Die durchschnittliche, tägliche, mobile Bildschirmzeit der Deutschen mit einem Android Gerät lag in 2020 bei 2,6 Stunden pro Tag und ist damit im Vergleich zum Vorjahr um ca. 13% gestiegen [5].

Die Mehrheit des Smartphone Marktes gliedert sich in zwei Teile. Auf der einen Seite Apple mit dem Betriebssystem iOS, auf der anderen Seite ein Großteil der Hersteller mit dem Betriebssystem Android. Eine Datenerhebung des Statista Research Departments [6] hat ergeben, dass sich im Januar 2021 die mobilen Betriebssysteme in Deutschland auf ca. 64% Android und ca. 35,5% iOS verteilen [6], wodurch ein Marktoligopol entstanden ist. Die Entwicklung von nativen Apps läuft auf dem Betriebssystem des Gerätes [7], es werden native SDKs verwendet sowie eine auf das Betriebssystem angepasste Programmiersprache [4]. Die Entwicklung von Anwendungen für die beiden Plattformen unterscheidet sich in ihren Grundzügen. Die Konsequenz für Entwickler liegt darin, verschiedene Programmiersprachen zu beherrschen und doppelte Zeit investieren zu müssen, um eine Anwendung für beide Plattformen zur Verfügung zu stellen [8].

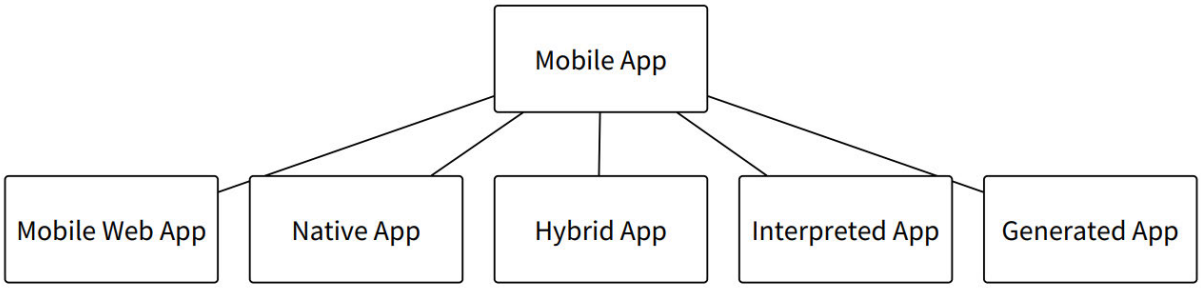


*Abbildung 1 - Möglichkeiten der mobilen App Entwicklung [10]*

Als potenzielle Lösung für dieses Problem haben sich verschiedene Cross-Plattform Development Frameworks entwickelt [9]. Diese Frameworks sollen den Entwicklern ermöglichen Apps in nur einem Schritt zu veröffentlichen. Damit können Kosten gesenkt und Produktivität gesteigert werden [2, 8, 9]. Unter den Frameworks gibt es, wie in Abbildung 1 zu sehen, verschiedene Ansätze, wie hybride Apps, interpreted Apps oder generated Apps [9, 10].

Neben den Cross-Plattform Development Frameworks hat sich bereits vor längerer Zeit durch die Verwendung von Web-Technologien eine weitere Möglichkeit zur plattformübergreifenden Entwicklung ergeben. Durch die Verwendung von HTML, CSS und JavaScript können Entwickler ihre Anwendung als eine einzelne Webseite entwerfen, die der Web Browser interpretiert und darstellt. Nach der Anpassung an die Spezifikationen jedes einzelnen Gerätes ist es unkompliziert möglich Inhalte auf allen Geräten darzustellen [8]. Dies bringt jedoch Probleme mit sich. Bei nicht ausreichender Internetverbindung ist eine zuverlässige Verwendung nicht gewährleistet [11]. Des Weiteren können Web Apps nicht auf alle plattformspezifischen APIs zugreifen und können nicht wie gewohnt installiert werden [9]. Als Resultat werden Nachteile im Vergleich zu nativen Apps im Zusammenhang mit der User Experience gesehen [2, 7, 8, 11–13].

Um den Problemen entgegenzuwirken hat Google im Jahr 2015 die Progressive Web Apps (PWA) vorgestellt [11]. Diese stellen eine neue Möglichkeit der Anwendungsentwicklung dar. Zwar basieren PWAs genau wie Web Apps auf HTML, CSS und JavaScript, aber kommen der nativen App deutlich näher. Sie können ganz wie native Apps vom Home-Bildschirm gestartet werden, haben nach dem ersten Starten die Möglichkeit ohne Internetverbindung verwendet zu werden [12] und können durch eine neue Web API auf gerätespezifische Inhalte wie die Kamera oder Push Benachrichtigungen zugreifen [11].

Es liegt nahe, dass sich die angesprochenen Probleme im Hinblick auf User Experience von Web Apps durch den Einsatz von PWAs verbessern. Die User Experience von PWAs und nativen Apps wurde im Vergleich allerdings bisher nicht ausreichend untersucht. Außerdem existieren nur wenige Arbeiten, die die Entwicklung von PWAs für einen User Experience Vergleich beschreiben.

Um die User Experience zu überprüfen wurde im Zuge dieser Arbeit eine bereits vorhandene native App als PWA möglichst detailgetreu nachprogrammiert. Mit dem Prototyp wurden verschiedene Tests durchgeführt, um die User Experience zu vergleichen. Einerseits durch eine qualitative Analyse mit Hilfe von Thinking Aloud. Es wurde eine Fokusgruppe bei der Bearbeitung von Aufgaben mit der PWA, und eine Kontrollgruppe bei der Bearbeitung derselben Aufgaben mit der originalen, nativen App, untersucht. Anschließend wurde eine qualitative Analyse in Form einer Umfrage durchgeführt, um vergleichbare, messbare Werte zur Analyse der User Experience zu erhalten.

Die Ergebnisse werden im Kontext analysiert und ausgewertet. Anschließend wird in einem Fazit erläutert, ob und inwieweit sich PWAs im Hinblick auf User Experience eignen, um eine Alternative zu nativen Apps darzustellen.

## 1.1 Aufbau der Arbeit

Zu Beginn wird ein Überblick über den aktuellen Forschungsstand von PWAs gegeben. Dabei wird erläutert welche neuen Möglichkeiten der Einsatz von PWAs mit sich bringt, und wie diese umgesetzt werden können. Außerdem wird der aktuelle Stand der Forschung im Bereich User Experience im Hinblick auf Cross-Plattform Development und insbesondere PWAs erläutert. Es wird dargestellt, was bereits erforscht wurde, an welchem Punkt noch Bedarf für eine Analyse besteht und an welchem diese Arbeit ansetzt. In der folgenden Methodologie werden die den Untersuchungsergebnissen zugrundeliegenden Forschungsmethoden dargestellt. Nachfolgend wird die originale, native App und ihre Funktionen vorgestellt. Außerdem wird der Entwicklungsprozess und die sich daraus ergebene Prototyp-PWA erläutert. Die durch die Untersuchung gesammelten Ergebnisse werden vorgestellt und im Kontext diskutiert. Aus diesen Ergebnissen werden im nachfolgenden Teil Schlüsse gezogen, um die Forschungsfrage zu beantworten.

# 2. Forschungsstand: Progressive Web Apps

Im Folgenden wird der aktuelle Forschungsstand zu Progressive Web Apps dargestellt. Dabei werden PWAs in den Kontext zwischen nativen Apps und Web Apps eingeordnet. Außerdem werden die neuartigen Funktionen, die eine PWA ausmachen, vorgestellt und deren Umsetzung erläutert. Anschließend werden aktuelle Ergebnisse der User Experience Forschung dargestellt. Außerdem werden aktuelle Forschungen im Bereich User Experience von PWAs erläutert und aus der daraus resultierenden Forschungslücke die Forschungsfragen abgeleitet.

## 2.1 Kontextuelle Einordnung

Seit dem Beginn der Entwicklung von mobilen Apps haben sich verschiedene Herangehensweisen etabliert [13]. In der ursprünglichen Entwicklung von Apps besteht keine Möglichkeit des Wiederverwendens von Programmcode auf anderen Geräten oder Plattformen [13]. Dies ist der Verwendung von nativen Apps und deren Aufbau des Codes geschuldet. Native Entwicklung auf iOS läuft über die eigene IDE Xcode. Diese ist mit der iOS SDK versehen, die die Entwicklung von Apps mit den Sprachen Objective-C und Swift unterstützt [10, 14, 15]. Die Entwicklung auf Android-Geräten läuft hingegen über die IDE Android Studio und die Android SDK mit den Sprachen Java und Kotlin [10, 14, 15]. Nach

Nunkesser (2018) [10] gelten die mit diesen SDKs entwickelten Apps als „Endemic Apps“, folglich plattformspezifische und nicht übergreifende Apps. Dies führt zum Nachteil eines beinahe linearen Wachstums des Aufwandes mit steigenden Plattformen. Daraus entwickelte sich der Bedarf von Cross-Plattform Development [16].

Neben den „Endemic Apps“ existieren nach Nunkesser [10] auch Apps auf Basis von unabhängigen Programmiersprachen. Diese „Ecdemic Apps“ können mit verschiedenen Herangehensweisen erstellt werden. Einerseits besteht die Möglichkeit, den in einer fremden Programmiersprache geschriebenen Code, mit der Hilfe einer virtuellen Maschine in

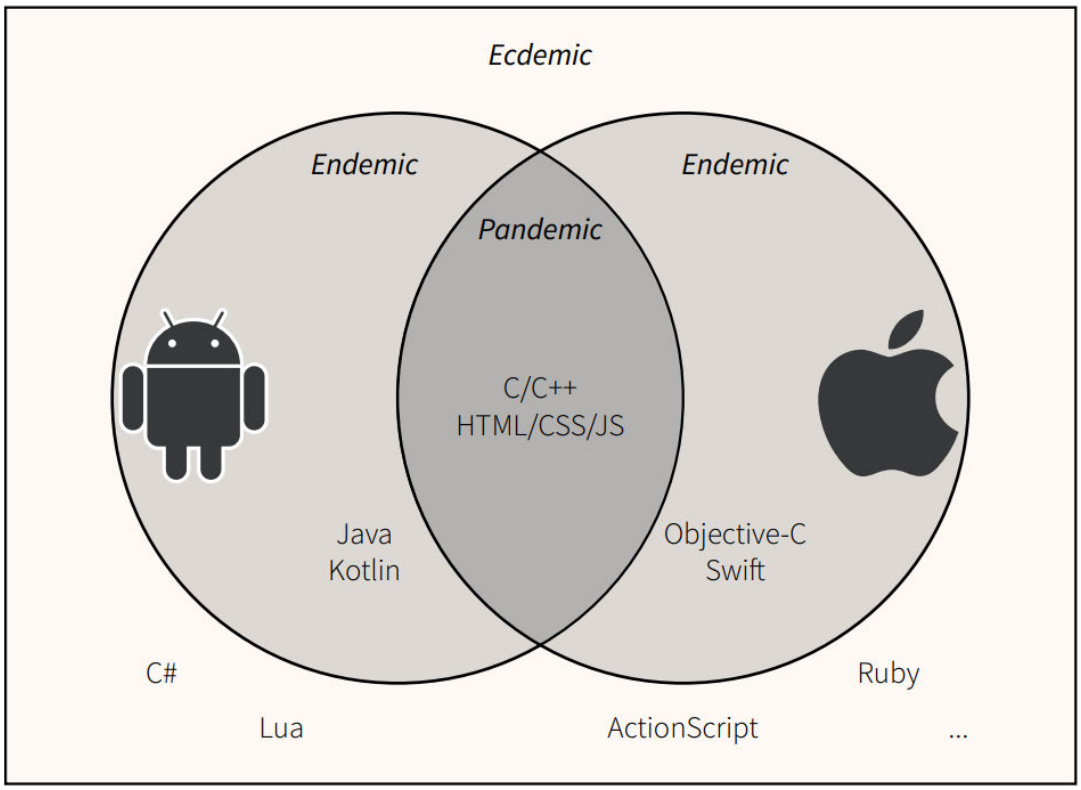


*Abbildung 2 - Kategorien von Mobile App Technologien nach Nunkesser [10]*

kompilierbaren Code umzuwandeln. Die mit Hilfe von virtuellen Maschinen verwendeten Apps werden auch als VM Apps bezeichnet [10]. Eine weitere Möglichkeit plattformfremde Programmiersprachen zu verwenden sind interpreted Apps. Diese verwenden zur Darstellung des User Interface eine native Sprache während die Logik der Anwendung in einer fremden Sprache geschrieben wird [14]. Eine dritte Möglichkeit zur Verwendung von fremdem Code sind generated Apps. Diese werden ganz in fremden Code geschrieben und dann mit der Hilfe von Software für jede Plattform einzeln in den entsprechenden nativen Code umgewandelt [10, 14]. Allerdings weisen diese Apps auch Probleme auf. Im Vergleich zu nativen Apps verbrauchen „Ecdemic Apps“ mehr Speicherplatz und besitzen trotzdem eine schlechtere Performance [17]. Eine dritte Kategorie bilden die „Pandemic Apps“. Diese sind entwickelt mit den Technologien, die von beiden Betriebssystemen unterstützt werden. Bei diesen

Technologien handelt es sich um Web-Technologien wie HTML, CSS und JavaScript sowie um C/C++ [10]. Neben den mit C/C++ entwickelten „System Language Apps“ [10] bestehen „Pandemic Apps” mindestens zu einem Teil aus Web-Technologien. Eine erste Möglichkeit, Apps auf Basis von Web-Technologien zu verwenden, sind die klassischen Web Apps. Diese basieren ausschließlich auf HTML, CSS und JavaScript. Damit sind diese Anwendungen schnell, auf allen Geräten verfügbar und platzsparend dank sparsamen Frameworks [17]. Allerdings fehlt den Web Apps der Zugang auf die native API, weshalb viele entscheidende Features nicht verfügbar sind [17]. Das Fehlen dieser Features bringt ein enormes Problem in Bezug auf User Experience mit sich [11], wodurch klassische Web Apps für viele Projekte nicht in Frage kommen. Eine Lösung für dieses Problem ist die Verwendung einer hybriden App. Hybride Apps, eine Kombination aus Web-Technologien und nativen Funktionen [8], erweitern die Web Apps um einen Zugriff auf die hardwarespezifischen Features [2]. Dies geschieht häufig über die Erstellung eines Web Views. Dieses Web View Objekt besteht aus den Web-Technologien, die die Webseite erzeugen. Es wird in eine native application shell, die Laufzeitumgebung der Anwendung, erzeugt [15]. Das bedeutet, dass die Inhalte der Web App als Objekt in eine native App eingefügt werden. Dadurch kann die App auf spezifische Features des Gerätes zugreifen, was eine normale Web App nicht kann. Außerdem kann eine hybride App zwar über einen Appstore installiert werden, problematisch ist allerdings die schlechte Performance [17]. Durch den Einsatz einer hybriden App wird zwar ein Zugriff auf die spezifischen APIs erreicht, dieser bleibt jedoch begrenzt [17].

Seit 2015 gibt es einen neuen, pandemischen Ansatz für das mobile Cross-Plattform Development. Die Technologie wurde von Alex Russel, Software Engineer bei Google, und dem Designer Frances Berriman unter dem Namen Progressive Web Apps zusammengefasst [18]. Diese basieren auch wie klassische Web Apps auf den bekannten Web-Technologien, können aber die Vorteile der neuen und moderneren Browser besser nutzen und mit der Hilfe von Service Workern und einem App Manifest in die native Umgebung der Geräte eingebunden werden [11]. Das hat zur Folge, dass Web Apps - entwickelt mit der PWA Technologie - weniger der klassischen Webseite ähneln, sondern mehr einer nativen App [9].

## 2.2 Technische Voraussetzungen von PWAs

Neben den allgemeinen Grundlagen für Web Apps (Web-Technologien wie HTML, CSS und JavaScript) gibt es technische Voraussetzungen, damit aus einer Web App eine PWA wird. Im Folgenden werden die drei grundlegenden technischen Bedingungen vorgestellt.

### 2.2.1 Web App Manifest

Erst das Web App Manifest ermöglicht die Installation einer PWA. Bei dem Web App Manifest handelt es sich um eine JSON-Datei, die Informationen über das Verhalten bei der Installation auf dem Gerät des Nutzers beinhaltet [11]. Dabei ist es bedeutsam, dass die Datei den Namen „manifest.json“ trägt und in der obersten Ebene des Webseiten Verzeichnisses liegt [19]. Die in der Datei vorgenommenen Konfigurationen sind erheblich, damit nach der Installation ein „app-like“[1] Gefühl bei der Benutzung der PWA entsteht [9]. Die Installation einer PWA verläuft anders als bei einer nativen App. Statt über den Appstore wird sie direkt über die besuchte Webseite bezogen.

```
"name": "Kochbuch",
"short_name": "Kochbuch",
"start_url": "/index.html",
"display": "standalone",
"theme_color": "#65f558",
"background_color": "#ffffff",
"orientation": "Portrait-primary",
"prefer_realted_applications": "true",

"icons": [
  {
    "src": "/img/Appimg/72.png",
    "sizes": "72x72",
    "type": "image/png"
  }
```

*Abbildung 3 - Typischer Aufbau einer manifest.json-Datei*

*Quelle: Eigene Aufnahme*

Die meisten Browser besitzen die Funktion „App zum HomeScreen hinzufügen“ [11], die aus den im Manifest gegebenen Informationen eine App auf dem Homebildschirm des Gerätes erzeugt. Mit diesem neuen Installationsweg wird eine Unabhängigkeit von den Appstores der Anbieter erzeugt [9]. Die Konfigurationsmöglichkeiten in der manifest-Datei sind umfangreich, es gibt allerdings, wie in Abbildung 3, für die Entwicklung einer PWA entscheidende

[1] vgl. [9]

Konfigurationen. Zuerst die Angabe eines Namens, bzw. einer Kurzbezeichnung durch die Eigenschaften „name“ und „short_name“. Diese geben den Namen der App bei der Installation und unter der App auf dem Homebildschirm an. Dabei wird der „short_name“ an den Stellen verwendet, an denen der Platz für den Namen beschränkt ist [19]. Die nächste bedeutsame Information stellen die Icons dar. Diese können verschiedene Funktionen einnehmen, wie zum Beispiel die Darstellung der App auf dem Homebildschirm oder als Ladebildschirm beim Laden der App. Icons werden in einem Array angegeben. Dieses Array muss die Quelle im Dateisystem, die Größe des Icons und den Typ des gegebenen Icons beinhalten. Es empfiehlt sich, eine möglichst große Varianz von Icons mit verschiedenen Größen anzugeben, da jedes Gerät eine andere Größe der Icons für die Darstellung auf dem Homebildschirm benötigt [19]. Die Angabe einer „start_url“ wird benötigt, damit der Browser erkennt, welche HTML-Seite der PWA die Startseite darstellt. Durch diese Einstellungsmöglichkeit startet die PWA von der angegeben Startseite, unabhängig davon, von welcher Unterseite die App aus dem Browser installiert wurde [19]. Mit der Angabe der Eigenschaft „display“ wird das User Interface und damit die User Experience am größten beeinflusst. Wird diese Eigenschaft auf „standalone“ oder „fullscreen“ gesetzt, so verschwindet die Ansicht des Browsers. Die App läuft nun über ein eigenes Fenster und Browser-Inhalte wie die Navigation oder die URL-Leiste verschwinden [19]. Mit der Eigenschaft „background_color“ lässt sich der Ladebildschirm der PWA anpassen. Dieser Ladebildschirm tritt bei dem ersten Öffnen der App auf, wenn die Inhalte der App im Hintergrund abgefangen und damit auf dem Gerät gespeichert werden. Der Ladebildschirm besteht aus dem gewählten Icon und der ausgewählten Farbe für den Hintergrund [19].

Um die angefertigte manifest.json Datei verwenden zu können, muss diese noch mit der Webseite verknüpft werden. Dazu wird der HTML-Tag „<link>“ verwendet[2]. Alle Seiten der PWA müssen mit Hilfe dieses Links referenziert werden [19].

Leider unterstützt iOS die Verwendung einer manifest.json Datei nicht. Um auf einem iOS Gerät über den Safari Browser trotzdem PWAs verwenden zu können, kann eine eigene Einbindung von Icons und Optik für iOS hinzugefügt werden. Im Head jeder verwendeten HTML-Datei muss dafür eine Einbindung eines passenden Icons[3], eines passenden Namens[4] und der Anzeigeeinstellungen[5] erfolgen [21].

[2] <link rel=“manifest“ href=“/manifest.json“>, Quelle: Quellcode des PWA Prototyp
[3] <link rel="apple-touch-icon" href="touch-icon-iphone.png"> [20]
[4] <meta name=“apple-mobile-web-app-title” content=”AppTitle”> [20]
[5] <meta name=”apple-mobile-web-capable” content=”yes”> [20]

### 2.2.2 Service Worker

Ein Service Worker zählt zu den relevantesten Technologien, die bei einer PWA verwendet werden. Dieser ermöglicht die meisten Konzepte (siehe 2.3 Erweiterte Eigenschaften von PWAs), die eine PWA ausmachen. Vereinfacht gesagt handelt es sich bei einem Service Worker um ein JavaScript Skript, das auf einem eigenen und unabhängigen Thread im Hintergrund des Browsers läuft [22]. Damit ist der Service Worker unabhängig von der über HTTPS laufenden eigentlichen Webseite und läuft auch weiter, wenn die Webseite geschlossen wird [23]. Bei der Registrierung des Service Workers wird er an die Webseite gebunden, indem er durch deren Protokoll, URL und verwendeten Port identifiziert wird [23]. Da es sich bei einem Service Worker um einen JavaScript Worker handelt, besitzt dieser keinen Zugang zur DOM [24, 25]. Um mit der Webseite zu kommunizieren, muss über eine Methode eine Nachricht gesendet werden. Diese gesendete Nachricht kann über einen Listener abgefangen werden, um die darin empfangenen Daten zu erhalten [26].

Der Service Worker ist ein ereignisgesteuertes Skript mit einem eigenen Lebenszyklus. Dieser Lebenszyklus besteht aus verschiedenen eintretenden Ereignissen. An erster Stelle steht dabei die Registrierung [26]. Diese wird clientseitig durch die Webseite selbst ausgelöst. Bei dem ersten Aufruf der Webseite durch einen Nutzer wird die Funktion zur Registrierung aufgerufen [23]. Diese Funktion muss sich in der Haupt-JavaScript-Datei der Webseite befinden. Sie dient dazu, dem Browser Informationen über die Herkunft des Service Workers zur Verfügung zu stellen [26].

An dieser Stelle ist es für die Sicherheit der Nutzer von Bedeutung, dass die Webseite über HTTPS läuft. Durch die Verwendung des Service Workers ergeben sich viele neue Möglichkeiten, die durch eine Sicherheitslücke ausgenutzt werden können [13], sodass eine Verwendung von HTTP nicht ausreicht. Die Verwendung von TLS mit HTTPS [23] schützt vor möglichen „man-in-the-middle“ Angriffen [23, 26]. Unter einem „man-in-the-middle“ Angriff versteht man das Abfangen von Informationen, beim Austausch zwischen zwei Parteien durch einen Angreifer. Diese abgefangenen Informationen können entweder gestohlen und möglichweise schädlich verwendet werden, oder die abgefangen Informationen werden modifiziert und falsch an den eigentlichen Empfänger weitergegeben [27].

Wurde der Service Worker erfolgreich registriert ist er nicht automatisch einsatzbereit. Nach der Registrierung lädt der Browser das Skript herunter. Wurde bisher noch kein Service Worker installiert oder haben sich Daten in der Datei des Service Workers geändert, so muss dieser installiert werden [28]. Die Installation kann dazu verwendet werden, die ersten Inhalte wie den HTML-, CSS- oder JavaScript Code der PWA zu cachen [24, 26]. Die Möglichkeit feste Inhalte zu speichern ist ein relevanter Punkt bei der von der PWA verwendeten Application Shell

Architektur (siehe 2.2.3 Application Shell Architektur) [24]. Anders als bei der Registrierung wird die Installation nicht durch die Haupt-JavaScript-Datei ausgelöst, sondern durch einen „Event-Listener“ im Service Worker Skript selbst [26].

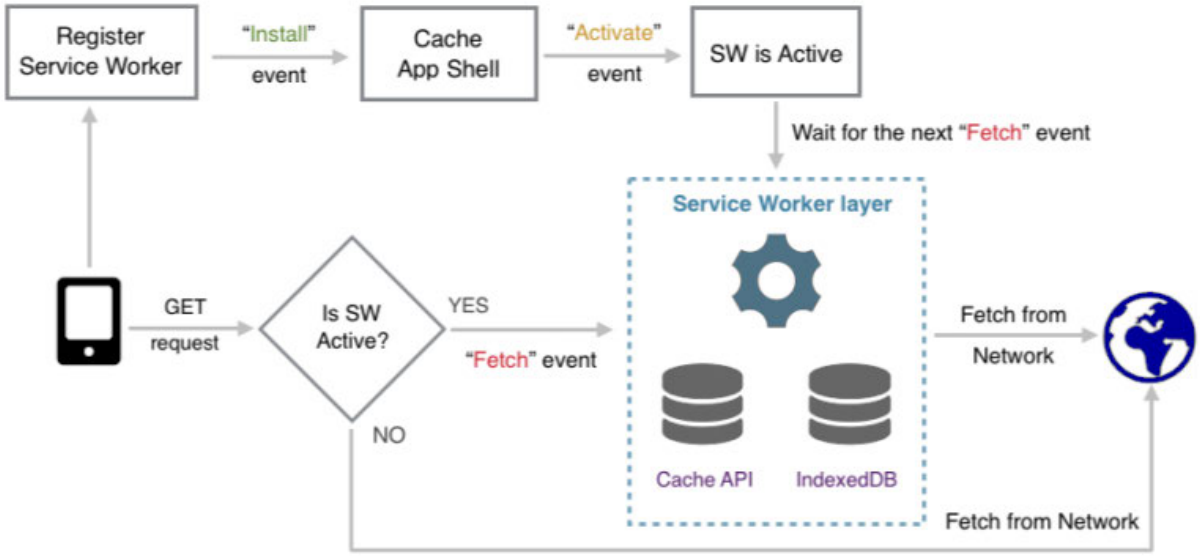


*Abbildung 4 - Lebenszyklus eines Service Workers [25]*

Nach einer erfolgreichen Installation ist der Service Worker zwar installiert, aber nicht zwingend aktiv [28]. Die Aktivierung des Service Workers erfolgt nur, wenn noch kein Service Worker aktiv ist, eine Methode zum Überspringen der Wartezeit aufgerufen wird, oder wenn der Benutzer die Webseite aktualisiert [28]. Falls keiner dieser Fälle eintritt, so geht der neue Service Worker in einen Wartezustand über, um auf die Terminierung des alten Service Workers durch das Neuladen der Webseite zu warten [26]. Tritt einer dieser Fälle ein, so wird wie bei der Installation ein Event ausgelöst. Dieses Event besteht ebenfalls aus einer vom Service Worker Skript selbst ausgelösten Methode. Diese kann unter anderem dazu verwendet werden, um bereits gespeicherte Daten im Cache zu aktualisieren oder nicht mehr gebrauchte Daten zu löschen [24].

Sobald der Service Worker erfolgreich aktiviert ist, hat dieser vollen Zugriff auf die Webseite. Er kann nun weitere Events mit der Hilfe von Listenern verarbeiten [28]. Das für eine PWA erheblichste Event, auf dessen Eintritt der Service Worker wartet, ist, wie in Abbildung 4 zu sehen, das fetch-Event. Dieses dient dazu, dynamisch Inhalte zu cachen und verfügbar zu machen. Cache bezeichnet dabei einen mit HTML5 eingeführten lokalen Speicher der Webseite, der unabhängig von der Internetverbindung zur Verfügung steht [23]. Daher können im Cache Informationen und Ressourcen lokal gespeichert und auch ohne Internetverbindung wieder abgerufen und verwendet werden. Durch den Einsatz eines Service Workers in Verbindung mit der Verwendung des Caches ergeben sich neue Möglichkeiten zur Offline-Verfügbarkeit einer Webseite oder einer PWA. Während der Installation des Service Workers werden die sicher benötigten Inhalte der PWA (die App Shell) durch den Service Worker vom

Server angefragt und im Cache gespeichert [24, 25]. Außerdem kann an dieser Stelle eine Ressource als „fallback page“ gecached werden, die vom Service Worker in dem Fall zurückgegeben wird, dass die angefragte Ressource aufgrund fehlender Internetverbindung nicht zurückgegeben werden kann [24]. Fordert der Nutzer durch seine Interaktionen auf der Webseite oder der PWA nun indirekt Informationen an, so wird, wie in Abbildung 5 zu sehen, eine Anfrage an den Server gegeben.

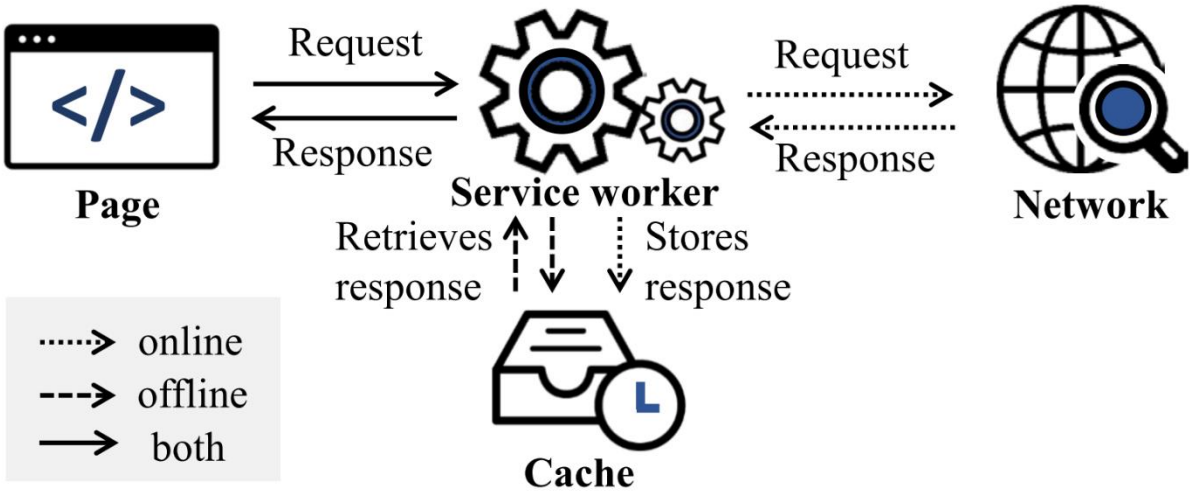


*Abbildung 5 - Arbeitsweise des Service Workers bei fetch-Event [23]*

Diese Anfrage wird vom Service Worker durch den Eintritt des fetch-Events abgefangen [29]. Besteht eine Verbindung zum Internet, wird die Anfrage an den Server über das Internet weitergegeben. Die Antwort des Servers, die die angeforderte Ressource enthält, wird nun vom Service Worker entgegengenommen. Dieser speichert die erhaltene Ressource im Cache und gibt sie gleichzeitig an die Webseite oder PWA weiter. Mit diesem Vorgang kommt ein dynamisches Caching zu Stande, das alle einmal angefragten Daten für die spätere erneute Verwendung speichert. Besteht nun keine Internetverbindung, so leitet der Service Worker die Anfrage nicht an den Server weiter, sondern versucht die passende Ressource aus dem Cache zurückzugeben. Wurde die Ressource noch nicht gecached, so kann der Service Worker die zuvor gespeicherte „fallback page“ an die Webseite oder PWA zurückgeben. Die Verwendung eines Caches wird von den meisten gängigen Browsern unterstützt [23].

Neben dem Caching durch fetch-Events hat der Service Worker noch weitere Funktionen. Mit der Hilfe von Service Workern werden bei PWAs Push-Benachrichtigungen realisiert. Diese Push-Benachrichtigungen bestehen in der Regel aus einem Titel, einem Hauptteil mit Text und ihrem Ursprung. Meistens ist aus optischen Gründen auch noch ein Icon der App in der Benachrichtigung enthalten [23]. Anders als bei nativen Apps werden die Push Benachrichtigungen nicht durch die installierte App realisiert, sondern durch den Browser [23]. Jeder Browser besitzt einen sogenannten Push Service, um Push Benachrichtigungen zu ermöglichen. Dieser dient als Instanz zwischen Server und User. Der Push Service erhält

Benachrichtigungen vom App-Server und leitet sie an den Service Worker des richtigen Nutzers weiter [23].

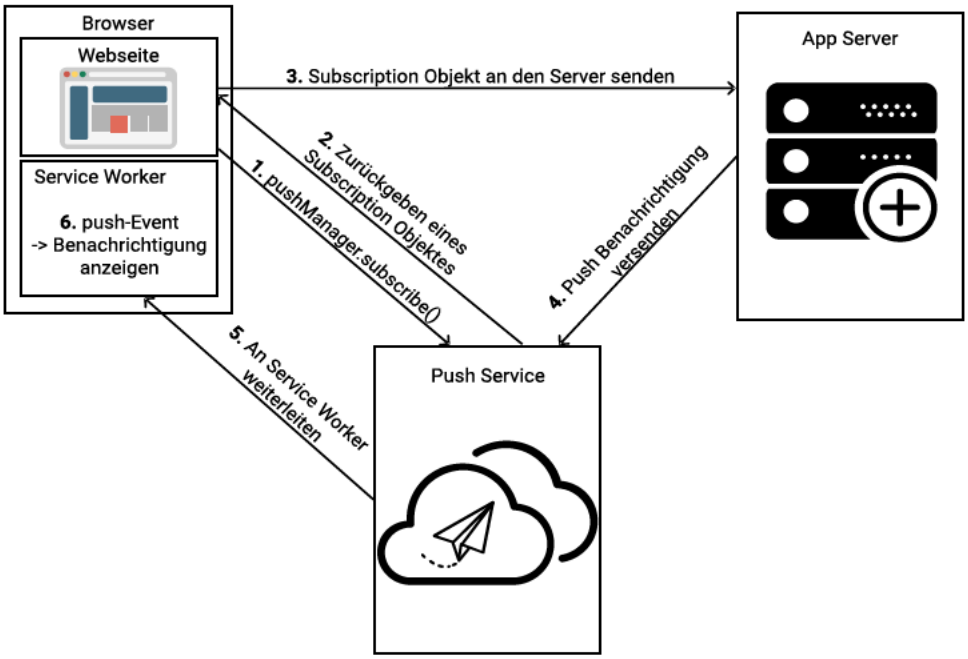


*Abbildung 6 - Ablauf von Push Benachrichtigungen bei PWAs*

*Quelle: Eigene Aufnahme*

Der Ablauf von Push Benachrichtigungen ist in Abbildung 6 zu sehen. Bevor ein Benutzer Push Benachrichtigungen erhalten kann, muss dieser dem Erhalt zustimmen. Falls der Benutzer nicht bereits dem Erhalt von Benachrichtigungen zugestimmt hat, erhält er durch den Aufruf der Methode „pushManager.subscribe()“ die Anfrage Benachrichtigungen zu erhalten [30]. Diese Methode besitzt zwei Konfigurationsmöglichkeiten. Einerseits die Eigenschaft „userVisibleOnly“. Diese stellt sicher, dass bei jeder erhaltenen Push-Benachrichtigung auch eine für den Nutzer sichtbare Benachrichtigung ausgelöst wird. Damit wird ein möglicherweise schädigender „silent push“ von Seiten des Entwicklers verhindert [30]. Die zweite Eigenschaft ist der „applicationServerKey“, einem Public-Key mit Informationen des App-Servers.

Um einen verbesserten Schutz durch Authentifikation zu gewährleisten, bietet sich hier der freiwillige Einsatz von VAPID an. Unter VAPID wird eine Erweiterung des Web Push Protokolls verstanden, die aus einem asymmetrischen Schlüsselpaar besteht [23]. Da jeder Browser seinen eigenen Push Service besitzt, können Entwickler die Authentifikation ihres App-Servers nicht durch einen Login oder ähnliches gewährleisten [31]. Durch die Verwendung des asymmetrischen Schlüsselpaares mit VAPID werden Push Benachrichtigungen, die nicht die nötige Authentifikation besitzen vom Push Service geblockt [23]. Nach dem einmaligen

Generieren des Schlüsselpaares wird der öffentliche Schlüssel in ein Uint8Array umgewandelt und als „applicationServerKey“ bei der „pushManager.subscribe()“ Methode verwendet [31].

Stimmt der Nutzer dem Erhalt von Benachrichtigungen zu, wird der Push-Service abonniert. Darauf antwortet der Push-Service mit einem „subscription object“ [23]. Dieses besteht neben kryptografischen Informationen zur Verschlüsselung von Nachrichten für den Benutzer [30] aus einer endpointURL. Diese URL besteht aus der Adresse des jeweiligen Push Service und einer einmaligen ID. Die ID identifiziert den Service Worker des Nutzers [32]. Das „subscription object“ sendet der Browser an den App-Server, der die Informationen in einer Datenbank speichert [30]. Mit der Hilfe dieser Informationen kann der Server Push-Benachrichtigungen über den Push-Service an den Nutzer senden [23].

Damit der App-Server nun Push-Benachrichtigungen mit VAPID an den Benutzer versenden kann, muss dieser erst die Web-Push-Einstellungen mit dem Schlüsselpaar konfigurieren. Der App-Server setzt seine „VapidDetails“ [31] mit der Angabe seiner E-Mail Adresse, des öffentlichen Schlüssels, der für den Nutzer auch als „applicationServerKey“ dient, und des privaten Schlüssels [31]. Nun kann der App-Server Push-Benachrichtigungen an den Benutzer senden. Dafür benötigt er lediglich das gespeicherte „subscription object“ und die zu versendende Nachricht. Versendet der Server nun eine Push-Benachrichtigung, so erhält diese der Push-Service. Dieser untersucht die endpointURL und ermittelt die ID des Benutzers. Über diese ID wird die Push-Benachrichtigung an den entsprechenden Service Worker weitergegeben [23].

Während des Wartezyklus eines Service Workers werden nicht nur fetch-Events abgefangen, sondern auch push-Events. Dieses Event wird ausgelöst, wenn der Browser eine Push-Benachrichtigung erhält [30]. Wenn die Funktion zur Darstellung einer Benachrichtigung des Service Workers durch einen Event-Listener ausgelöst wird, sind die Daten der Benachrichtigung mit Hilfe verschiedener Methoden zugänglich [30]. Um die Benachrichtigungen mit Interaktionen zu versehen, besteht die Möglichkeit dem Service Worker einen Event-Listener hinzuzufügen, der durch Interaktionen mit der Benachrichtigung ausgelöst wird. Mit Hilfe dieser Möglichkeit können dem Nutzer verschiedene Auswahlmöglichkeiten zur Reaktion auf eine Benachrichtigung gegeben werden [30].

Zu den technischen Folgen bei der Verwendung eines Service Workers gab es bereits erste Untersuchungen. Diese haben gezeigt, dass die Verwendung eines Service Workers in vielen Fällen einen geringfügig negativen Einfluss auf die Energieeffizienz von PWAs haben [12]. Gleichzeitig gibt es Ergebnisse, dass die Performance einer PWA mit Service Worker besser ist als das Pendant einer nativen Android App [24]. Außerdem verbessert die Verwendung von Caching die Performance von PWAs [24]. Daher ist abzuwägen, ob die durch die Verwendung

eines Service Workers mit Caching möglicherweise verbesserte User Experience den Mehrverbrauch an Energie rechtfertigt.

### 2.2.3 Application Shell Architektur

Unter der Application Shell (kurz App Shell) Architektur versteht man eine Möglichkeit des Aufbaus einer PWA. Eine Verwendung der App-Shell-Architektur hat zu Folge, dass Inhalte sofort laden und mit einer nativen App vergleichbare Ladezeiten entstehen [33]. Dabei ist eine gute App Shell ausschlaggebend für eine gute User Experience. Das sofortige Laden von Teilen der Inhalte verhindert das Aufkommen des Gefühls eine langsame App zu verwenden [9]. Die App Shell definiert die minimal benötigten Inhalte (HTML, CSS, JavaScript, Icons, etc.), um das User Interface darzustellen [33].

Um eine gute App-Shell-Architektur zu verwenden, ist es relevant, statische Inhalte, wie das minimale User Interface und dynamische Inhalte zu trennen [33]. Der Grund, weshalb durch die Trennung geringe Ladezeiten der Kerninhalte erzielt werden, liegt im Caching. Die App Shell und der dynamische Inhalt sollten separat gecached werden [11]. Im statischen Caching wird die App Shell bereits beim ersten Aufruf der PWA über API-Anfragen des Browsers an den Server übertragen und dann gespeichert [11]. Die dynamischen Inhalte der PWA werden mit dem dynamischen Caching erst bei einer späteren Anfrage an den Server gespeichert.

Die Verwendung dieser Architektur bringt einige Vorteile mit sich. Zuerst die schnellen und verlässlichen Ladenzeiten. Zwar sind beim ersten Aufruf nur Kerninhalte ohne Laden verfügbar, doch bei erneutem Aufruf sind alle Inhalte durch das dynamische Speichern schnell bereitgestellt [33]. Des Weiteren bietet die App-Shell-Architektur eine den nativen Apps ähnliche Benutzererfahrung und liefert die Grundlagen für die Offline-Verfügbarkeit [33]. Außerdem wird der Speicherplatzverbrauch grundsätzlich verringert. Da nur minimale Kerninhalte statisch gecached werden, ist eine PWA durch die App-Shell-Architektur für minimalen Datenverbrauch konstruiert [33]. Vorangegangene Untersuchungen von Gambhir et. al. (2018) [24] haben Ergebnisse geliefert, dass die bei der Installation gespeicherte Datenmenge einer PWA um ein vielfaches geringer ist als die Datenmenge des nativen Pendants. Dabei ist anzumerken, dass bei der Verwendung der PWA Inhalte dynamisch gecached und damit gespeichert werden, wodurch der Speicherverbrauch potenziell steigt.

## 2.3 Erweiterte Eigenschaften von PWAs

Neben den vorangegangenen technischen Voraussetzungen gibt es verschiedene Konzepte, die eine PWA beinhalten sollte. Diese ergeben sich aus den verschiedenen Techniken, die bereits beschrieben wurden (siehe 2.2 Technische Voraussetzungen von PWAs). Diese 10 verschiedenen Konzepte wurden von unterschiedlichen Quellen als Grundlagen der PWAs herausgearbeitet [9, 11, 34–37]. Im Folgenden werden die verschiedenen grundlegenden Konzepte der PWAs vorgestellt.

**Responsive:** PWAs sollten ein responsives Webdesign besitzen, damit sie auf verschiedenen Geräten verwendet werden können. Die verschiedenen Geräte, auf denen eine PWA verwendet werden kann (PC, Handy, Tablets, etc.) besitzen unterschiedliche Bildschirmgrößen und Seitenverhältnisse. Daher sollte die PWA so angepasst sein, dass sich Größen, Platzierungen und Schriftarten automatisch auf die richtigen Einstellungen anpassen [9, 11, 34–37]. Ein responsives Webdesign ist kein PWA spezifisches Konzept und sollte bei allen Webseiten und Web Apps verwendet werden [9].

**Linkable**: Ein großer Vorteil von PWAs gegenüber nativen Apps ist die URL. Mit Hilfe der URL kann die PWA problemlos geteilt werden. Dabei werden sogar Informationen, die in der URL enthalten sind, geteilt. Dieses Feature ist zwar bei allen Webseiten vorhanden, allerdings nicht bei nativen Apps [9, 34, 35, 37]. Durch den Einsatz des Android Features „Intents“ öffnet sich nach Installation einer PWA bei Klick auf die entsprechende URL nicht der Browser, sondern die installierte PWA [9].

**Discoverable:** Eine weitere Eigenschaft von Webseiten, die einer PWA zugutekommen, ist die Fähigkeit im Browser von einer Suchmaschine gefunden zu werden. Durch den Einsatz des Web App Manifests und des Service Workers können Inhalte der PWA besser analysiert werden. Folglich können PWAs daher unkompliziert anhand Ihrer Inhalte von Suchmaschinen gefunden werden [9, 11, 34–37].

**Installable:** Die Möglichkeit eine Webseite zu installieren ist eine der grundlegendsten Eigenschaften einer PWA. Die Installierbarkeit grenzt PWAs von einer normalen Webseite ab und macht sie mit nativen Apps vergleichbar. Durch den Einsatz des Web App Manifests besitzt eine installierte PWA eine native Oberfläche ohne Browserelemente. Die Installation erfolgt direkt aus dem Browser heraus über die Funktion „Zum Homebildschirm hinzufügen“, daher wird kein Appstore benötigt [9, 11, 34–37]. Nutzer, die ihre Apps installieren, verwenden sie häufiger. Daher steigt die Nutzungshäufigkeit der PWA [36].

**App-Like:** Eine PWA kann und sollte eine mögliche Alternative zu einer nativen App sein. Dafür sollte ihr Benutzergefühl möglichst nah an einer nativen App sein. Eine gut gestaltete

PWA verhält sich wie eine native App und ist nur anhand ihres Aussehens nicht von einer nativen App zu unterscheiden. Dafür sollte eine geeignete und mit einer nativen App vergleichbare App-Shell-Architektur verwendet werden [9, 11, 34, 35]. Die Vision von Google hinter PWAs sieht vor, dass sich alle installierten PWAs mit nativen Apps messen können [9].

**Network independent:** Auch die Abhängigkeit einer Internetverbindung unterscheidet eine PWA stark von einer Webseite. Nach dem ersten Öffnen der Inhalte ist eine PWA ohne Internetverbindung verfügbar oder auch wenn nur eine schlechte Internetverbindung vorliegt. Dies ermöglicht die bereits erläuterte Technologie des Caching, mit der Hilfe des Service Workers [9, 11, 34–37].

**Fresh:** Ein Service Worker läuft in einem von Webseiten unabhängigen Hintergrundprozess. Service Worker laufen auf einem eigenen Thread und können auch nach dem Schließen einer Webseite weiterarbeiten. Daher haben Service Worker die Möglichkeit neue Inhalte direkt zu laden – egal ob während der Benutzung der PWA oder im Hintergrund. Damit sind bei bestehender Internetverbindung die Inhalte in einer PWA immer aktuell. Bei fehlender Internetverbindung werden alte gespeicherte Informationen dargestellt, die bei Wiederherstellung der Internetverbindung aktualisiert werden [9, 11, 34, 35].

**Safe:** Eine Voraussetzung für PWAs ist die Verwendung von HTTPS. Durch der HTTPS zugrundeliegenden TLS-Technologie findet eine sichere Verschlüsselung statt. Durch die sichere Ende-zu-Ende-Verschlüsselung mit TLS [38] sind PWAs sicher vor Angriffen von Dritten auf möglicherweise sensible Informationen [9, 11, 34, 35, 37].

**Re-engageable:** Durch den Einsatz von den bisher nur den nativen Apps zugänglichen Push-Benachrichtigungen können PWAs ihre Benutzer kontaktieren und an ihre Anwendung erinnern. Die Option dafür liegt in der neuen Web-Push-API. Die Benachrichtigungen bringen die Möglichkeit, Benutzer bei neuen Inhalten zu informieren, um eine möglicherweise verlorene Aufmerksamkeit zurückzugewinnen [9, 11, 34, 35, 37]. Die Installation auf dem Homebildschirm, die bei einer normalen Webseite oder klassischen Web App nicht möglich ist, kann einen ähnlichen Zurückgewinnungs-Effekt auf Benutzer haben [11].

**Progressive:** Das grundlegendste und namensgebende Konzept hinter PWAs ist die Progressivität. Diese beschreibt die Möglichkeit der Unterstützung von verschiedenen Browsern. Die PWA soll auch in älteren Browsern funktionieren. In älteren Browsern werden zwar möglicherweise nicht alle Features unterstützt, aber die grundlegenden Funktionen sollten verfügbar sein [9, 11, 35–37].

Diesem Konzept liegt ein Entwicklungstrend der Prinzipien der Webentwicklung zu Grunde. Durch die sich mit der schnellen Verbesserung von Technologien verbreitenden Varianz an

Geräten mit verschiedenen Browsern und Geschwindigkeiten, begann die Entwicklung eines übergeordneten Modells [39]. Der zuerst entstandene Ansatz war die Entwicklung mit „graceful degradation“ [39]. Die Idee dahinter ist, Webseiten für die besten verfügbaren Browser und Technologien zu entwerfen. Damit ergibt sich, dass die Benutzererfahrung bei den besten Browsern optimal ist, und sich die Erfahrung bei schlechten Browsern immer weiter verschlechtert [39]. Ein aktuellerer Ansatz ist das „progressive enhancement“ [39], das eine Art Gegensatz zum „graceful degradation“ darstellt. Hier werden Webseiten auf Basis der Inhalte entwickelt, unabhängig vom optischen Ergebnis. Folglich besitzen auf diesem Ansatz basierende Webseiten eine hohe Verfügbarkeit für Browser mit verschiedenen Stärken. Dabei ist eine Unterstützung von schlechten Browsern gegeben und die Benutzererfahrung wird bei immer besseren Browsern und Technologien besser [39].

PWAs beziehen sich auf den Ansatz des „progressive enhancement“ und übernehmen die Idee. Eine PWA sollte daher auf allen Browsern verfügbar sein und sich gleichzeitig bei besseren Browsern weiter verbessern [9, 11].

## 2.4 Untersuchungen zur User Experience

Im Folgenden werden die Hintergründe zur Relevanz der Messung von User Experience von (Software-)Produkten vorgestellt. Anschließend werden bereits vorangegangene Untersuchungen der User Experience bei Cross-Plattform Development Ansätzen, inklusive PWAs, vorgestellt und dargelegt, warum eine weitere Untersuchung der User Experience bei PWAs relevant ist.

### 2.4.1 Hintergründe zur Messung von User Experience

Eine Möglichkeit um die Eignung von Systemen, Produkten oder Dienstleistungen zu beschreiben ist die User Experience (UX). In den letzten Jahren ist die User Experience ein bedeutsames Interessengebiet der Mensch-Computer Interaktion (oder Human-Computer Interaction / HCI) geworden [40]. Allerdings ist es schwierig eine einheitliche Definition für User Experience zu finden. Eine Möglichkeit, um User Experience zu verallgemeinern ist eine Definition der Internationalen Organisation für Normung (ISO). Die Norm 9241 beschreibt dabei Inhalte der Ergonomie der Mensch-System-Interaktion. Zur Definition von User Experience heißt es:

"*A person's perceptions and responses that result from the use and/or anticipated use of a product, system or service.”* (ISO 9241-110:2010, Abschnitt 2.15, [41])

Die User Experience ist ein erheblicher Aspekt wenn es um die Messbarkeit des andauernden Konsums von Produkten, Systemen und Dienstleistungen geht [42]. Allerdings scheint die ISO Definition nicht weitreichend genug zu sein. Neben den reinen Möglichkeiten und Antworten, die ein Produkt liefert, entstehen während der Benutzung Erfahrungen mit dem Produkt [40]. Diese Erfahrungen gehen über die Zeit nach der Auseinandersetzung mit einem Produkt hinaus. Erfahrungen entstehen vor und während der Benutzung, verändern sich mit der Zeit und sind subjektiv [41].

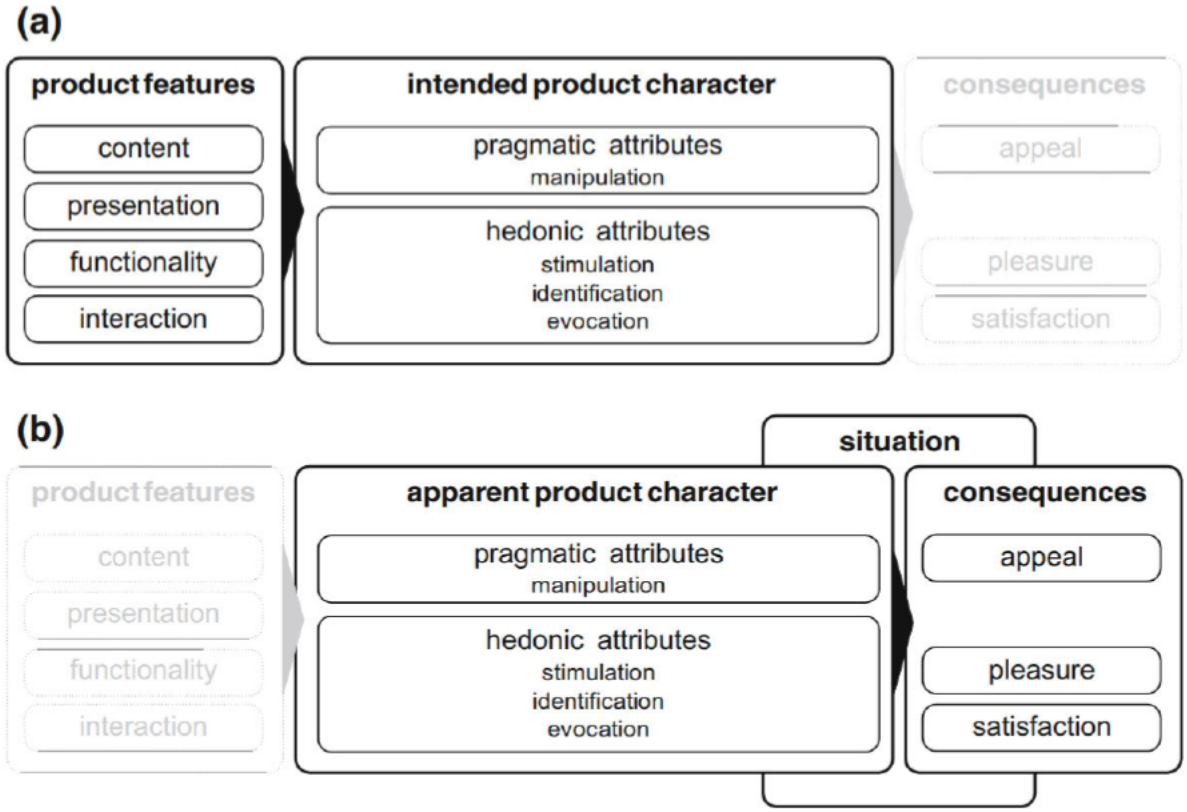


*Abbildung 7 - Elemente der User Experience aus Sicht des Designers (a) und Benutzers (b) [40]*

Die perspektivische Sicht auf die User Experience eines Produktes unterscheidet sich, wie in Abbildung 7 zu sehen, zwischen dem Designer und dem Benutzer [40]. Der Designer, in Abbildung 7 oben, eines Produktes hat die Möglichkeiten Eigenschaften anzupassen. Die Auswahl von „Inhalt, Präsentation, Stil, Funktionalität und Interaktion“[6] [40] und deren Kombination erschafft einen Produktcharakter. Allerdings handelt es sich dabei um eine subjektive Absicht und Ansicht des Produkt Designers [40]. Der Nutzer, in Abbildung 7 unten, hat keinen Einfluss auf die Eigenschaften eines Produktes, der Designer allein ist dafür verantwortlich [40]. Bei seiner Wahl der verschiedenen Eigenschaften eines Produkts besitzt der Designer eine Vorstellung und Absicht des aus der Wahl entstehenden Produktcharakters. Der Produktcharakter teilt sich in „pragmatic“ und „hedonic attributes“ auf [40, 42]. Dabei

[6] Übersetztes, direktes Zitat. Original: „content, presentational style, functionality, interactional style” [40]

bezeichnen die „pragmatic attributes“ die Charakteristiken, die zur eigentlichen Problemlösung und Zielerreichung dienen, und die Möglichkeit diese zur Problemlösung dienenden Funktionen zu verwenden [40, 42]. Gleichzeitig kennzeichnen die „hedonic attributes“ alle verbleibenden subjektiven Eigenschaften. Beim Hedonismus handelt es sich um die Sinneslust und die psychische Lust, die bei der Erfüllung der zu bewältigenden Aufgabe empfunden wird [40, 42].

Der Benutzer erhält einen subjektiv wahrgenommenen Produktcharakter aus einer Kombination der Produkteigenschaften und seinen persönlichen Erwartungen und Standards [40]. Der vom Designer angedachte Produktcharakter und der subjektiv wahrgenommene Produktcharakter des Benutzers können dabei auseinandergehen. Ebenso kann der wahrgenommene Produktcharakter zwischen verschiedenen Benutzern aufgrund von abweichenden Charakteren und Erwartungen auseinandergehen [40]. Durch die Verwendung eines Produktes mit einem eigenen Charakter erhält jeder Nutzer subjektive Gefühle als Konsequenzen. Dabei unterscheiden sich verschiedene Ebenen der Wahrnehmung. Nach Hassenzahl (2018) [40] werden „satisfaction“ (im folgenden Zufriedenheit), „pleasentness“ (im folgenden Erfreulichkeit) und „appeal“ (im folgenden Reiz) unterschieden [40].

Ein Produkt erreicht die Zufriedenheit eines Nutzers, wenn es die erwarteten Eigenschaften besitzt, demnach wenn die „pragmatic attributes“ [42] für den Nutzers zufriedenstellend sind [40]. Gleichzeitig empfinden Nutzer eine Erfreulichkeit, wenn das Produkt unerwartete positive Eigenschaften, die über die erwarteten Eigenschaften hinausgehen, besitzt. Dabei gilt, dass umso unerwarteter die Eigenschaft, desto höher ist die Freude des Benutzers [40]. Wenn das Produkt bei seiner Verwendung positive Reaktionen im Benutzer auslöst, so hat es einen Reiz. Dabei ist der Reiz eines Produktes stark subjektiv und abhängig von der Situation des Benutzers [40]. Aufgrund der verschiedenen subjektiven Reize, die ein Produkt auslöst, sollte der Reiz des Produktes bei der Bewertung der User Experience nicht zu stark bewertet werden. Der Fokus sollte eher auf die Gründe der Wahrnehmung der Benutzer und den wahrgenommenen Produktcharakter liegen [40].

Aus den Untersuchungen lässt sich ableiten, dass es sich bei User Experience um ein komplexes psychologisches Konstrukt handelt [40]. Gleichzeitig ist User Experience immer subjektiv. Es existieren verschiedene Sichten auf die User Experience des gleichen Produktes – verschiedene Sichten zwischen Benutzern und verschiedene Sichten zwischen Benutzer und Designer. Die wahrgenommene Produkterfahrung des Nutzers besteht aus einem subjektiven Produktcharakter und verschiedenen ausgelösten Gefühlen [40]. Da der erwartete Produktcharakter des Designers von denen der Benutzer auseinandergeht, und der Designer die Konsequenzen für die Benutzer nicht vorhersagen und beeinflussen kann, ist es von hoher

Relevanz die User Experience eines Produktes zu testen. Da sich der Produktcharakter und die Konsequenzen auch mit der Zeit ändern können [40], sollte die User Experience auch nach der Veröffentlichung eines Produktes getestet werden.

### 2.4.2 User Experience im Cross-Plattform Development

Das Thema Cross-Plattform Development wurde bereits in vielen verschiedenen Konferenzbeiträgen, Fachaufsätzen und wissenschaftlichen Arbeiten behandelt. Dabei werden die Themen PWA und User Experience nur in wenigen Fällen bearbeitet. Zur Vorbereitung dieser Arbeit wurden verschiedenen Quellen zum Thema Cross-Plattform Development untersucht. Daraus wurden 23 verschiedene Quellen ausgewählt. Diese werden im Folgenden zu den Themen Cross-Plattform Development, PWAs, User Experience und der Entwicklung einer eigenen Prototyp-Anwendung analysiert und eingeordnet.

| Zahl im Bild | Literaturverzeichnis |
|---|---|
| 1 | [7] |
| 2 | [18] |
| 3 | [1] |
| 4 | [15] |
| 5 | [4] |
| 6 | [9] |
| 7 | [13] |
| 8 | [8] |
| 9 | [2] |
| 10 | [16] |
| 11 | [43] |
| 12 | [23] |
| 13 | [44] |
| 14 | [45] |
| 15 | [46] |
| 16 | [22] |
| 17 | [12] |
| 18 | [47] |
| 19 | [48] |
| 20 | [49] |
| 21 | [11] |
| 22 | [14] |
| 23 | [3] |

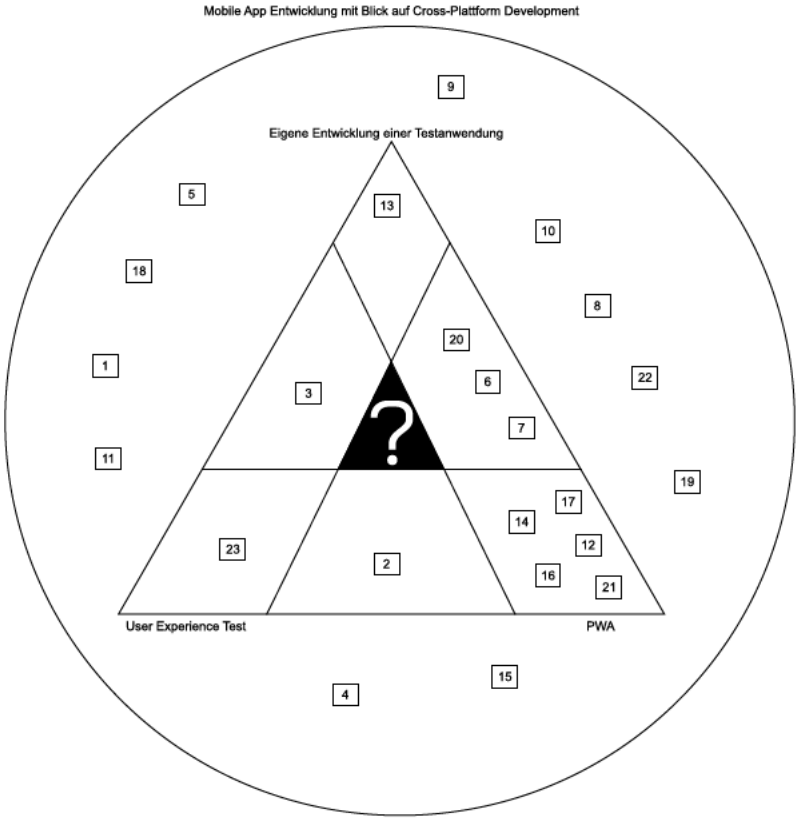


*Abbildung 8 - Forschungslücke im Bereich PWA und User Experience*

*Quelle: Eigene Anfertigung*

Die erste und größte Gruppe bilden dabei die Untersuchungen, die allgemein das Thema Cross-Plattform Development behandeln. Die Quellen sind in Abbildung 8 als die Kästen dargestellt, die sich außerhalb des Dreieckes befinden. Diese wissenschaftlichen Beiträge

haben zwar das Thema Cross-Plattform Development zum Thema, aber keinen Bezug zu PWA, User Experience oder der Entwicklung eines Prototyps.

Die weiteren Quellen, die sich in Abbildung 8 im Dreieck befinden, behandeln die in dieser Arbeit betrachteten Aspekte. Fünf verschiedene Quellen befassen sich mit der Entwicklung eines Prototyps. Majchrzak, Grønli et al. entwickelten 2017 Prototypen für drei verschiedene Cross-Plattform Development Frameworks [44]. Eine Forschung von Angulo und Ferre aus dem Jahr 2014 untersuchte die User Experience eines interpreted Prototyps und einer nativen App [1]. Andere Forschungen untersuchten Progressive Web Apps und entwickelten für ihre Forschung selbst einen Prototyp [9, 13, 49].

Auf der Asia-Pacific Software Engineering Conference im Jahr 2014 stellten Li, Jiang et al. eine nutzergeführte Technologie zum Testen von mobile Apps vor [3]. Damit beschäftigten sie sich unter anderem mit einem User-Experience-Test im Bereich des Cross-Plattform Developments.

Einige betrachtete Quellen untersuchen PWAs, ohne dabei selbst eine Entwicklung durchzuführen. Dabei werden unter anderem der Einfluss von Caching auf den Energieverbrauch [22], der Einfluss von Service Workern auf die Energieeffizienz [12] oder der Einfluss von der neuen Technologie PWA auf das Cross-Plattform Development betrachtet [11, 23, 45]. Dabei beschreibt allerdings keiner dieser Arbeiten die gesamte Entwicklung einer PWA, die sich für eine Untersuchung von User Experience eignet.

Die Literatursuche hat nur einen einzigen Konferenzbreitrag, bzw. Fachaufsatz zum Thema User-Experience-Vergleich von nativen Apps und PWAs ergeben. Im Zuge des Brazilian Symposium on Human Factors in Computing Systems im Jahr 2018 untersuchten Andrade Cardieri und Zaina die User Experience in PWAs, klassischen Web Apps und nativen Apps [18]. Für die Messung der User Experience von PWAs und nativen Apps wurden Versionen der Hotel-Metasuche Trivago verwendet. An der Untersuchung haben acht verschiedene Testpersonen teilgenommen. Ihnen wurden fünf verschiedene Aufgaben gestellt, die sie auf beiden Anwendungen zu absolvieren hatten. Die Analyse der qualitativen Ergebnisse der User Experience Untersuchung lief dabei auf zwei verschiedene Arten ab. Einerseits über die Methode Self Assessment Manikin [18]. Dabei konnten die Benutzer ihre Emotionen bei der Bewältigung einer Aufgabe in drei verschiedenen Dimensionen selbst erfassen. Andererseits untersuchten Spezialisten im Bereich Mensch-Computer Interaktion die Gesichtszüge der Testpersonen, um Emotionen zu erkennen und zu analysieren. Dabei kam die Untersuchung zu dem Ergebnis, dass es keine nennenswerte Unterschiede zwischen den unterschiedlichen Plattformen bezogen auf die User Experience gibt und, das Benutzer eine gute Erfahrung mit der Anwendung auf allen Plattformen machen können [18].

Die kaum vorhandenen Untersuchungen zum Vergleich von User Experience bei PWAs und nativen Apps zeigt, dass weitere Forschungen benötigt werden, um repräsentative Schlüsse zur Eignung von PWAs im Hinblick auf User Experience zu ziehen. Die Ergebnisse des Vergleichs in der Arbeit von Andrade Cardieri und Zaina basieren auf nur einer einzigen Anwendung und zwei Analysemethoden. Gleichzeitig liefern die aktuellen Untersuchungen keine Beschreibung zu der Entwicklung einer App, die dem Vergleich der User Experience dienen könnte.

Diese Arbeit stellt eine weitere Analyse der User Experience im Vergleich von Progressive Web Apps und nativen Apps dar. Es soll die Frage beantwortet werden, ob eine Anwendung ohne nennenswerte Einbußen der User Experience als PWA realisiert werden kann, um die Vorteile des Cross-Plattform Developments auszunutzen. Außerdem soll die Arbeit die Entwicklung einer PWA beschreiben und erproben, wie sich die Technologie PWA für den Nachbau einer nativen App eignet.

# 3. Methodologie

Die vorliegende Arbeit untersucht die User Experience von Progressive Web Apps und nativen Apps. Dafür wurde eine bereits vorhandene native Android App möglichst detailgetreu als Prototyp einer Progressive Web App nachprogrammiert. Um die Nutzererfahrung dieser beiden Anwendungen zu überprüfen, eignen sich sowohl eine qualitative Analyse als auch eine quantitative Analyse.

Bei einer quantitativen Analyse werden große Datenmengen erhoben und anschließend analysiert, um statistische Informationen mit thematischer Aussagekraft zu erhalten [50]. Eine qualitative Analyse setzt auf die Datenerhebung mit der Hilfe von Aussagen, Diskussionen und Interviews weniger verschiedener Personen. Die Ergebnisse erschließen sich hier über Beobachtungen von Verhalten oder Aussagen von Personen, anstatt über statistisch, mathematische Auswertungen [50].

Beide Methoden können für die Analyse von User Experience verwendet werden, allerdings unterscheiden sich die Ergebnisse. Quantitative Analysen legen eher allgemeine Nutzerpräferenzen dar, während qualitative Analysen unbekannte und feinere Probleme im Bereich User Experience aufdecken [50].

Gleichzeitig sollten für eine quantitative Analyse mit geringer Fehlerspanne eine große Anzahl an verschiedenen Datensätzen erhoben werden, während bei einer quantitativen Analyse mehr Zeit mit den Probanden investiert werden muss [50].

Da die vorliegende Arbeit eher auf das Aufdecken feiner Probleme im Vergleich der User Experience zwischen nativen Apps und PWAs abzielt, wurde eine qualitative Analyse durchgeführt. Außerdem ist es nicht möglich den Prototyp einer ausreichend großen Zahl an Testpersonen für die Durchführung einer validen qualitativen Analyse zur Verfügung zu stellen.

Die qualitative Analyse wurde mit Hilfe des Testansatzes Thinking Aloud durchgeführt. Um trotzdem messbare, vergleichbare quantitative Daten zu erhalten, wurde den Testpersonen nach Abschluss des Thinking Aloud Prozesses eine Umfrage zur Verfügung gestellt. Aufgrund der kleinen Teilnehmerzahl ist die Repräsentativität der Ergebnisse allerdings fraglich.

## 3.1 Thinking Aloud

Bei der Thinking Aloud Methode beschäftigen sich die Testpersonen mit der zu untersuchenden Software. Dabei haben die Testpersonen vorgegebene Aufgaben zu lösen. Während der Arbeit mit der Software sprechen die Testpersonen ihre Gedanken laut aus [51]. Dabei externalisieren sich intern gedachte Probleme, Meinungen, Erwartungen, etc. Der Designer oder Ausführende der Methode unterstützt die Testpersonen beim Absolvieren der Aufgaben und analysiert im Nachgang die aufgenommenen und externalisierten Gedanken der Testpersonen.

Thinking Aloud ist keine neue Methode zur Prüfung von User Experience, aber eine nachgewiesen effektive Methode. Jørgensen hat im Jahre 1990 bereits die Effektivität von Thinking Aloud geprüft:

„*This study has shown that the thinking-aloud method is a successful instrument in user interface in systems development as applied by motivated system designers. It is fairly straightforward to use and can even be applied with little or no human factors training. The method has the property of inherently promoting cognitive ergonomics by providing timely, genuine and applicable feedback to the system designers.*" (Jørgensen, 1990 [51])

Um die User Experience von PWAs zu untersuchen, wurde im Zuge dieser Arbeit der Thinking Aloud Prozess durchgeführt. Dafür wurde eine native Android App als Prototyp mit der PWA Technologie nachgebaut. Dabei handelt es sich um eine Kochbuch App, mit der Funktion eigene Rezepte anzulegen und zu verwalten (siehe 4. Entwicklung einer Prototyp-App). Die Testpersonen wurden in zwei Gruppen eingeteilt. Eine Gruppe führt die gestellten Aufgaben an dem programmierten Prototyp aus. Die andere Gruppe stellt die Kontrollgruppe dar. Diese absolvieren dieselben Aufgaben an der originalen, nativen App. Durch den Thinking Aloud Prozess in beiden Gruppen externalisieren sich sowohl positive Gedanken als auch Probleme bei dem Absolvieren der Aufgaben unter der Verwendung beider Apps. Anschließend kann

abgewogen und analysiert werden, welche angesprochen Probleme durch den Einsatz der PWA entstanden sind, und welche Probleme allgemein an der Anwendung selbst liegen. Gleichzeitig können so Stärken der PWA besser von allgemeinen Stärken der Anwendung, die auf beiden Technologien ähnlich gut funktioniert, differenziert werden.

Bevor die gestellten Aufgaben absolviert werden konnten, musste die App oder der Prototyp von den Teilnehmern installiert werden. Dabei unterscheidet sich die Art der Installation, da das native Original über den Appstore und der Prototyp über den Browser installiert wird. Die den Testpersonen im Thinking Aloud Prozess gestellten 12 Aufgaben (siehe Anhang 1.2) bilden einen Großteil der Funktionen der App ab. Es muss ein Rezept angelegt, gesucht, geteilt und gelöscht werden. Gleichzeitig müssen Rezeptkategorien erstellt, verwaltet und wieder gelöscht werden. Alle gestellten Aufgaben sind sowohl mit Hilfe der originalen App als auch mit dem Prototyp zu lösen. Der Thinking Aloud Prozess wurde für beide Varianten der App auf einem Huawei P10 Lite, einem Android Smartphone, durchgeführt.

## 3.2 Umfrage

Um neben den qualitativen Informationen, die sich durch den Thinking Aloud Prozess ergeben, auch vergleichbare quantitative Daten zu erhalten, wurde eine Umfrage mit den Testpersonen durchgeführt. Dabei bekamen alle Testpersonen aus beiden Gruppen dieselben Fragen. Diese Informationen können statistisch ausgewertet werden, um messbare Daten zur User Experience zu erhalten. Durch die geringe Anzahl an Teilnehmern sind die Repräsentativität und Validität der Umfrageergebnisse allerdings fraglich. Die Umfrage wurde digital mit der Hilfe einer Online-Umfragen-Plattform durchgeführt.

Die Fragen (siehe Anhang 1.1) teilen sich in vier Teilbereiche auf. Im Ersten werden Fragen zur Installation sowie zur Darstellung auf dem Homebildschirm gestellt. Auf Basis dieser Daten kann die Akzeptanz zur Installation einer PWA ohne Appstore abgeleitet werden. Der zweite Teilbereich umfasst allgemeine Fragen zum Empfinden der Nutzbarkeit. Dabei geht es um simple und die User Experience beeinflussende Inhalte, wie zum Beispiel die offensichtliche Gestaltung klickbarer Elemente oder der logischen und natürlichen Darstellung von Informationen. Im dritten Abschnitt geht es um Funktionen, die der Prototyp als PWA enthält, die nicht als klassische Web App realisierbar wären. Der Nutzer wird nach seiner Meinung zur Umsetzung auf den Zugriff auf internen Speicher und Offline-Verfügbarkeit gefragt. Dies sind kritische Punkte, die eine PWA ausmachen und ermöglichen, dass die native App als PWA Prototyp umgesetzt werden konnte. Im vierten Teilbereich bewerten die Testpersonen nicht die Anwendung selbst. Stattdessen wird ein Stimmungsbild eingeholt, auf welche

Eigenschaften die Testpersonen bei einer App besonderen Wert legen. Dabei wurden Fragen bezüglich des Energieverbrauchs, des Speicherplatzverbrauchs, der Offline-Verfügbarkeit und des internen Zugriffs gestellt.

## 3.3 Teilnehmer

Insgesamt haben 12 Personen am Thinking Aloud Prozess teilgenommen. Die gleichen 12 Personen haben auch die anschließende Umfrage ausgefüllt. Dabei hat die eine Hälfte der Teilnehmer die native originale App verwendet, die andere Hälfte der Teilnehmer den PWA Prototyp. Keiner der Teilnehmer wusste zum Zeitpunkt des Thinking Alouds davon, dass es zwei verschiedene Anwendungen gibt, an denen die Nutzererfahrungen getestet werden. Die Teilnehmer der beiden Gruppen haben für eine bessere Übersicht im Testprozess und der Auswertung eine Bezeichnung, bestehend aus einem Kürzel für die Variante der App und einer Nummer, erhalten.

| PWA | | | Nativ | | |
|---|---|---|---|---|---|
| Nummer | Geschlecht | Alter | Nummer | Geschlecht | Alter |
| PWA1 | Männlich | 51 | NAT1 | Weiblich | 49 |
| PWA2 | Weiblich | 48 | NAT2 | Männlich | 57 |
| PWA3 | Weiblich | 20 | NAT3 | Männlich | 21 |
| PWA4 | Weiblich | 21 | NAT4 | Männlich | 21 |
| PWA5 | Männlich | 22 | NAT5 | Weiblich | 46 |
| PWA6 | Weiblich | 26 | NAT6 | Männlich | 49 |

*Abbildung 9 - Übersicht über die Teilnehmer*

*Quelle: Eigene Anfertigung*

Beide Teilnehmergruppen wurden so ausgewählt, dass sie hinsichtlich verschiedener Gesichtspunkte möglichst gut übereinstimmen. Relevant waren dabei unter anderem allgemeine Punkte, wie zum Beispiel das Geschlecht und das Alter. Noch relevanter für den Nutzungstest von Apps ist jedoch die allgemeine Affinität und der Umgang mit Webseiten und Apps. Beide Gruppen enthalten Personen, die eine hohe Affinität zu einem geübten Umgang mit Apps haben, aber auch Personen, die eine eher geringere Affinität aufweisen. Außerdem enthalten beide Gruppen jeweils einen Teilnehmer, mit Kenntnissen zu Web-Entwicklung und allgemeinem Verständnis für die Entwicklung von Software. Aufgrund dieser verschiedenen Betrachtungspunkte weichen, wie in Abbildung 9 zu sehen, sowohl die Geschlechterverteilung als auch das Durchschnittsalter leicht voneinander ab. Insgesamt sollten diese geringen Abweichungen aber keinen Einfluss auf die Ergebnisse des Thinking Alouds und der Umfrage haben.

# 4. Entwicklung einer Prototyp-App

Um die User Experience auf eine repräsentative Art zu vergleichen, bedarf es zwei möglichst gleiche Anwendungen. Da diese Voraussetzung ohne eine eigene Entwicklung nur schwer zu erfüllen ist, braucht es ein prototypisches PWA-Abbild einer nativen App. Es existieren nur wenige Konferenzbeiträge und Fachaufsätze zum Thema PWA, die den Entwicklungsprozess einer mit einer nativen App vergleichbaren PWA beschreiben. Daher soll diese Arbeit, neben der User Experience, auch den Entwicklungsprozess eines PWA Prototyps, als Nachbau einer nativen App, beschreiben.

## 4.1 Darstellung der Original-App

Bei der verwendeten Original-App handelt es sich um ein Kochbuch. Die App vom Hersteller „Flose“ ist im Android Appstore, dem Google Play Store, verfügbar. Sie ist, wie in Abbildung 10 zu sehen, der Kategorie „Essen & Trinken“ zugeordnet, wurde mehr als 50.000-mal heruntergeladen und besitzt eine durchschnittliche Bewertung von 4,2 von 5 Sternen.

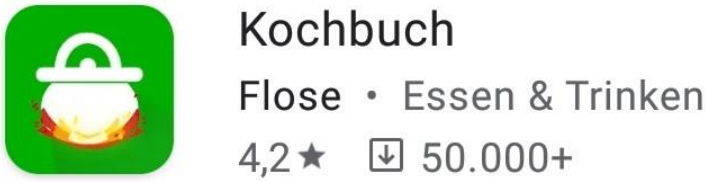


*Abbildung 10 - Original-App im Google Play Store*

*Quelle: Eigene Aufnahme*

Die Anwendung ermöglicht es Rezepte zu verwalten und zu speichern. Dabei können Nutzer ihre eigenen Rezepte erstellen. Jedem Rezept können verschiedene Eigenschaften und Informationen hinzugefügt werden. Bei den Eigenschaften handelt es sich um einen Namen, den benötigten Zutaten und einer Zubereitungsanweisung. Außerdem kann jedem Rezept eine Quelle, Anmerkungen, Personenzahl, Zubereitungszeit, Kochzeit, Bewertung oder ein Schwierigkeitsgrad hinzugefügt werden. Neben dem Speichern von Informationen besteht die Möglichkeit, über Zugriff auf den internen Dateispeicher oder die Kamera, Bilder des Rezepts hinzuzufügen. Diese Bilder werden im Rezept selbst und in der Rezeptübersicht zur besseren Übersicht angezeigt. Um die Rezepte besser zu ordnen, besteht zudem die Möglichkeit, Kategorien zu erstellen, zu bearbeiten und wieder zu löschen. Einige Kategorien sind bereits bei der Installation vordefiniert. Die Zuordnung von Kategorien eines Rezepts kann problemlos

über die Rezeptansicht bearbeitet werden. Rezepte können zu beliebig vielen Kategorien hinzugefügt werden.

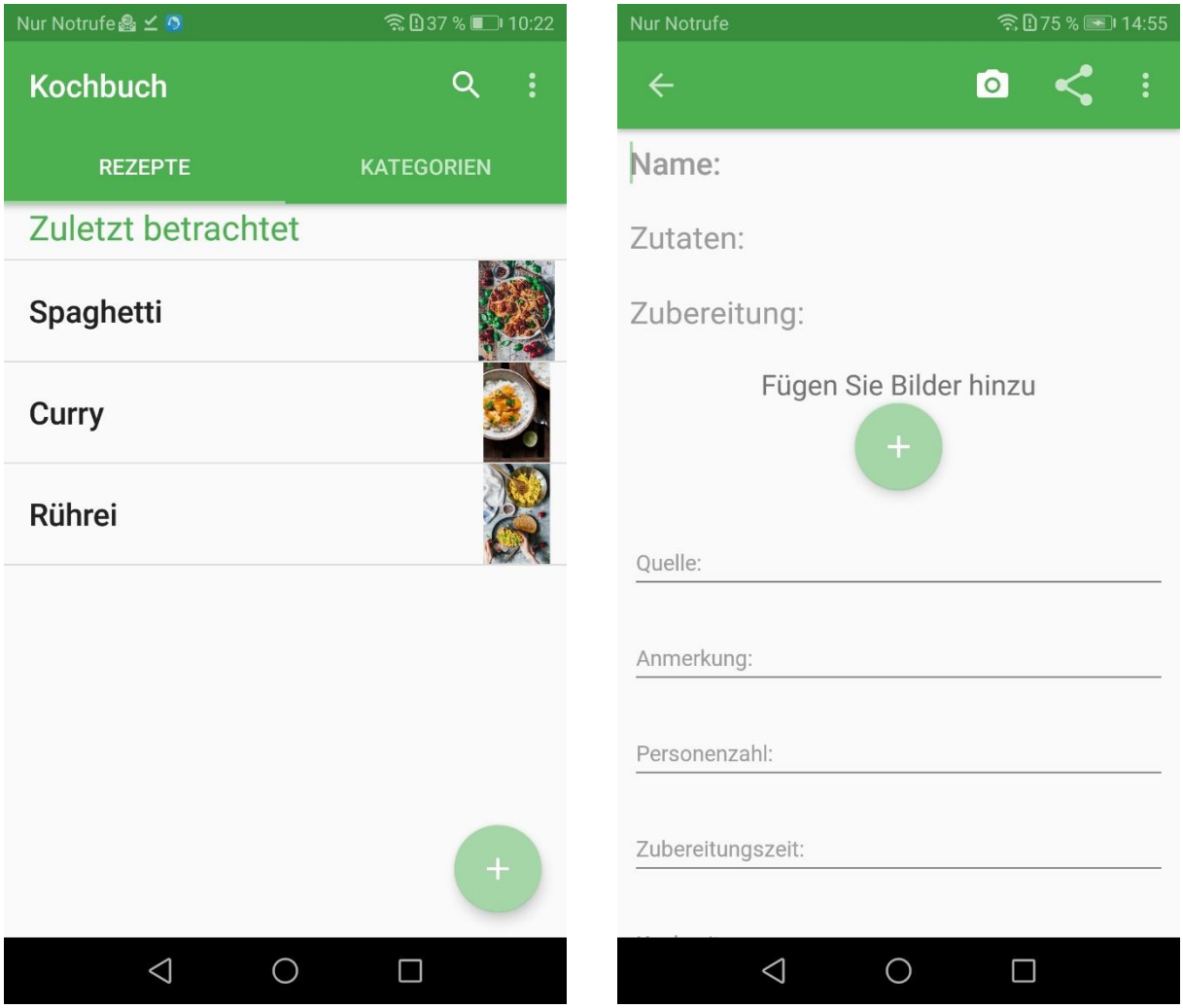


*Abbildung 11 - Übersicht der Original-App (links), sowie das Hinzufügen von Rezepten (rechts)*

*Quelle: Eigene Aufnahme*

Um bei einer großen Menge an Rezepten die Handhabung zu vereinfachen, verfügt die App über eine Suchfunktion. Diese ermöglicht die Suche nach dem Rezeptnamen und leitet den Nutzer direkt an die Rezeptansicht weiter. Außerdem können bereits angelegte Rezepte geteilt werden. Über Zugriff auf das Android File Sharing System kann der Nutzer auswählen, auf welche Art und Weise Rezepte geteilt werden sollen – per E-Mail, Nachricht, WhatsApp oder anderem Messanger.

Außerdem kann der Rezeptbestand auf eine SD-Karte exportiert oder wieder neu importiert werden. So wird dem Nutzer eine Möglichkeit für die Erstellung eines Backups gegeben. Gleichzeitig können so ganze Rezeptbestände an Freunde oder an ein neues Gerät weitergegeben werden. Aufgrund des großen entstehenden Aufwands durch die Implementierung dieser Funktion und dem fehlenden Nutzen in Bezug auf den Vergleich der User Experience von nativen Apps und PWAs wurde diese Funktion im Prototyp nicht

realisiert. Die ausgewählte native App eignet sich aufgrund verschiedener Gesichtspunkte für den Vergleich. Einerseits ist sie leicht nachzubauen, aber trotzdem komplex genug, um eine Messung der User Experience durchzuführen. Außerdem ist die App für eine breite Zielgruppe relevant und enthält mit dem Zugriff auf Kamera, internen Speicher und das File Sharing System gerätespezifische Funktionen.

## 4.2 Programmierung des Prototyps

Im Prototyp wurden, mit der Ausnahme der Import-/Export-Möglichkeit, alle Funktionen der nativen App umgesetzt. Dabei sehen sich die Benutzeroberflächen von PWA und nativer App in fast allen Bereichen ähnlich.

Der nachprogrammierte Prototyp basiert ausschließlich auf den Web-Technologien HTML, CSS und JavaScript. Das Grundgerüst bildet eine normale Webseite, somit eine klassische Web App. Elemente wie die Navigationsleiste am oberen Rand des Bildes wurden mit Hilfe des Frontend-Frameworks Materialize umgesetzt.

Die Hauptseite der PWA bildet, wie bei einer klassischen Webseite, die „index.html“. Auf dieser Seite sind im Prototyp zwei verschiedene Tabs dargestellt. Im ersten Tab findet sich eine Übersicht der gespeicherten Rezepte, im zweiten Tab eine Übersicht der Kategorien. Die Funktionalität hinter den Tabs liefert eine JavaScript-Funktion. Um ein Rezept hinzuzufügen, findet sich unter der Rezeptübersicht einen Button, der den Nutzer zu einer weiteren HTML-Seite leitet, die in Abbildung 12 rechts zu sehen ist. Hier kann, analog zur nativen App, ein Rezept hinzugefügt werden. Wurde ein Rezept hinzugefügt so kehrt der Benutzer auf die Rezeptübersicht zurück.

Per Klick auf eines der Rezepte in der Übersicht, wird der Nutzer zu einer Rezeptansicht weitergeleitet. Diese Rezeptansicht wird durch einen Link zu einer neuen HTML-Seite inklusive einer identifizierenden ID in der URL aufgerufen. Dabei wird eine JavaScript Funktion aktiviert, die die URL auf die enthaltende Rezept-ID untersucht und anschließend die entsprechenden Informationen aus dem Backend lädt. In dieser Übersicht kann das Rezept eingesehen, bearbeitet und auch wieder gelöscht werden. Die Bearbeitung kann direkt an den Rezepteinträgen vorgenommen werden. Die Einträge werden, wie in der nativen App auch, bei Klick auf den Zurück-Button automatisch gespeichert und übernommen. Das Löschen eines Rezeptes erfolgt über ein Seitenmenü, das am rechten oberen Bildrand aufgerufen wird. Außerdem bietet die Rezeptansicht die Möglichkeit, ein Rezept zu teilen. Dies kann über die verschiedenen angebotenen Wege des Smartphones erfolgen, wie zum Beispiel E-Mail, SMS oder einen Messenger-Service.

Zurück auf der Rezeptübersicht, die in Abbildung 12 links zu sehen ist, findet sich oben rechts im Menü Band die Funktion, nach einem Rezept zu suchen. Mit Hilfe dieser Suchfunktion kann ein Rezept durch Angabe des Namens gefunden werden.

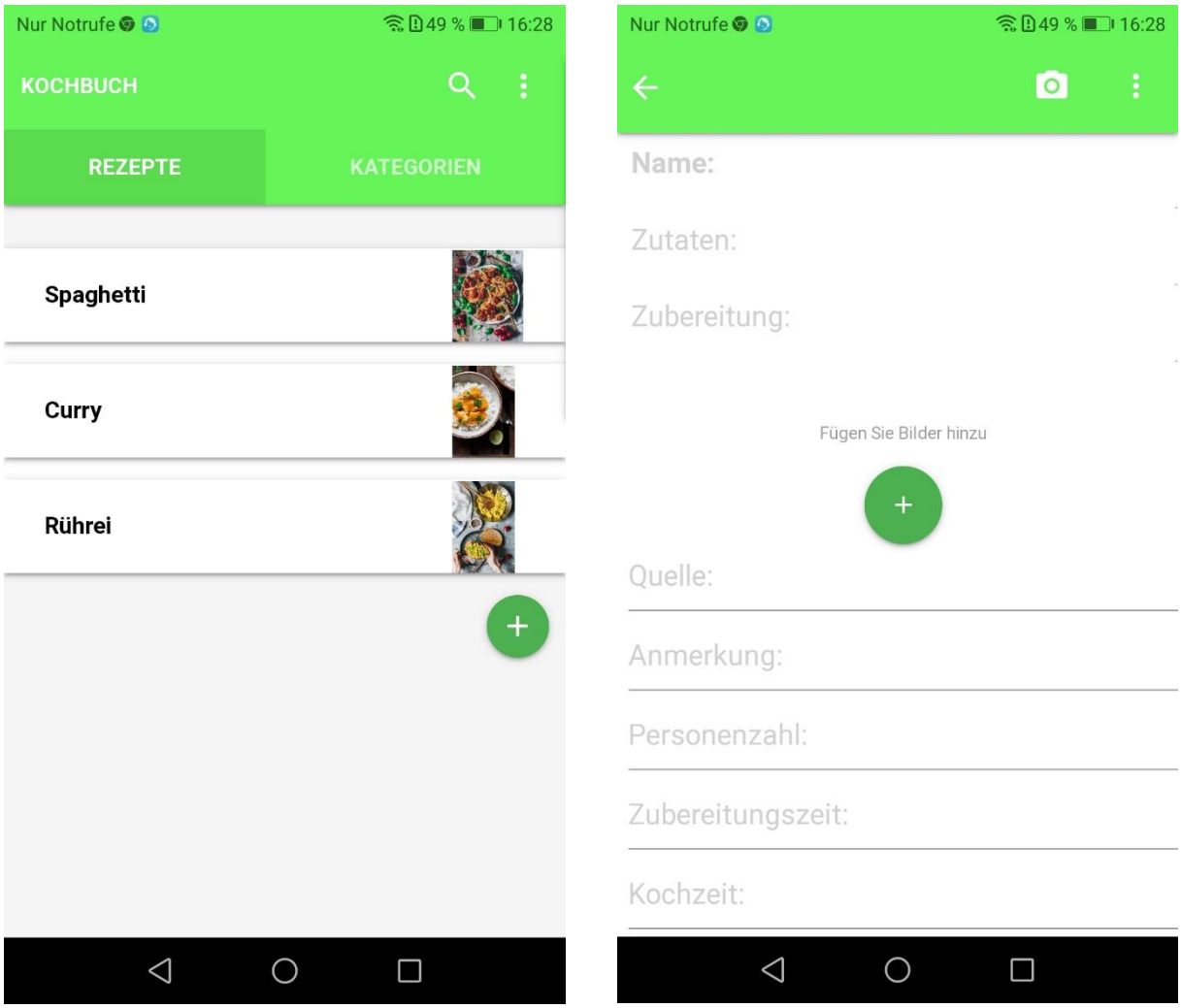


*Abbildung 12 - Übersicht der PWA (links), sowie das Hinzufügen von Rezepten (rechts)*

*Quelle: Eigene Aufnahme*

Realisiert wurde die Suche mit Hilfe einer Abfrage im Backend. Das Backend liefert die passende Rezept ID, mit der analog zur normalen Rezeptansicht die passenden Informationen angezeigt werden. Neben der Suchfunktion befindet sich ein Button, der ein Seitenmenü aufruft. Über dieses lassen sich neue Kategorien anlegen und Informationen zu der PWA einsehen. Per Klick auf den zweiten Tab mit der Beschriftung „Kategorien“ öffnet sich eine Übersicht mit allen angelegten Kategorien. Diese liefern mit der Hilfe eines Buttons alle Rezepte in einer Übersicht, die sich in einer der angelegten Kategorien befinden. Außerdem kann über einen Button eine Kategorie gelöscht werden. Es ist jedoch zu beachten, dass nur die Kategorie selbst gelöscht wird und nicht die verknüpften Rezepte.

Damit wurden alle allgemeinen Funktionen der PWA vorgestellt. Diese umfassen, bis auf das Exportieren, alle Funktionen der nativen App. Im Folgenden wird vorgestellt, wie die verschiedenen PWA Technologien, sowie das Backend und die Serverapplikation umgesetzt worden sind.

### 4.2.1 Umsetzung der PWA Technologien

Die Installation der PWA wird, wie in Kapitel 2 beschrieben, durch ein App Manifest ermöglicht. Diese JSON-Datei beinhaltet Informationen, die der Browser verwendet, um aus der Webseite eine App zu erstellen. Im Falle des hier dargestellten Prototyps wurde der Name „Kochbuch“ angegeben. Außerdem wurde die Startseite auf die „/index.html“ und der Anzeigemodus auf eine rahmenlose App gesetzt. Die Farbe der Android-Menüleiste am oberen Bildrand wurde mit der gleichen Farbe wie die Navigationsleiste der PWA gefärbt (HEX: #65f558). Um die App auf verschiedenen Geräten mit passendem Icon darzustellen, beinhaltet das App Manifest des Prototyps acht verschiedene Größen des verwendeten Bildes. Diese Bilder finden sich, zusammen mit dem favicon.ico, im Dateisystem in der Ordnerstruktur „/img/Appimg“ wieder. Das favicon.ico sorgt dafür, der App im Browser das passende Icon zu geben.

Das Manifest und das favicon.ico wurden in allen verwendeten HTML-Seiten im Head verlinkt. So wird die Installation von allen Seiten ermöglicht. Durch die Angabe der Startseite öffnet sich die PWA trotzdem, unabhängig von der Seite, von der sie installiert wurde, mit der „index.html“. Die Webseite weist alle Anwender beim ersten Aufruf der Seite auf die Möglichkeit der


*Abbildung 13 - Meldung "Add to Homescreen"*

*Quelle: Eigene Aufnahme*

Installation hin. Dafür wird, wie in Abbildung 13 zu sehen, eine eigene Meldung erzeugt und an den Nutzer gegeben. Klickt dieser auf die Meldung, wird die PWA installiert. Die Installation kann aber auch über die Einstellungen des Browsers, ohne die Verwendung der Meldung erfolgen. Um die Funktionalität mit iOS zu gewährleisten, wurde eine eigene Unterstützung in allen HTML-Seiten hinzugefügt. Diese ermöglicht die Installation und die rahmenlose App-like Verwendung auf allen iOS Geräten. Alle Funktionen, einschließlich der Offline-Verfügbarkeit, der Installation und dem Zugriff auf das File Sharing System, funktionieren sowohl auf Android als auch auf iOS. Der Prototyp wurde nur zur besseren Vergleichbarkeit mit der nativen App auf einem Android Gerät verwendet. Die Darstellung der PWA auf dem Homebildschirm sieht, wie in Abbildung 14 links zu sehen, der nativen Original App ähnlich. Im linken Teil von

Abbildung 14 findet sich auf der linken Seite das Icon der nativen App und auf der rechten Seite das der PWA.

Im rechten Teil von Abbildung 14 ist der Ladebildschirm der PWA zu sehen. Dieser tritt bei dem erstmaligen Öffnen der PWA oder nach dem kompletten Schließen des Browsers und

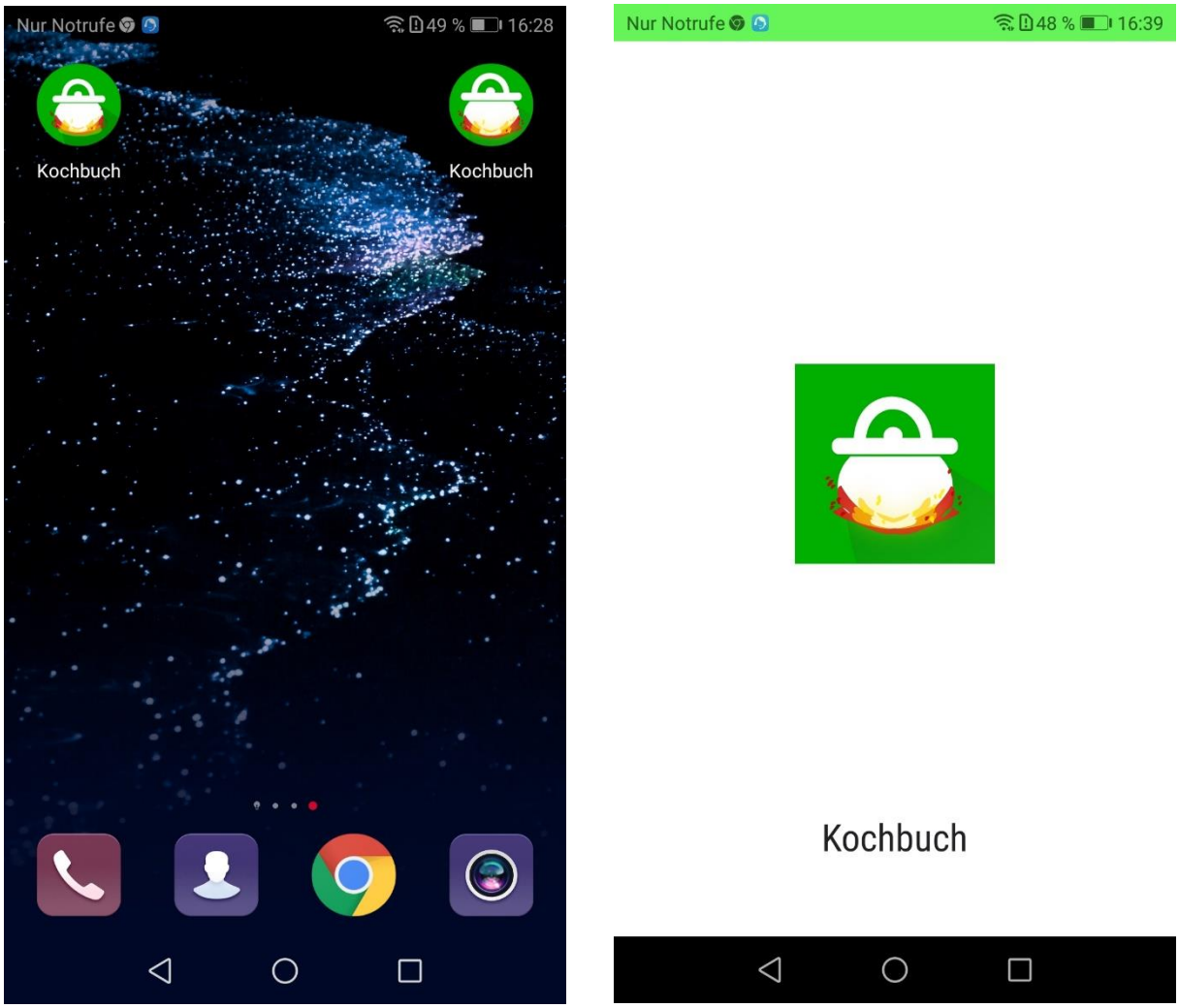


*Abbildung 14 - Icons von PWA und Original (links), sowie Ladebildschirm der PWA (rechts)*

*Quelle: Eigene Aufnahme*

Löschen des Caches auf. Die Ladeansicht besteht aus einer im App Manifest angegebenen Hintergrundfarbe (in diesem Fall weiß), einem von der Größe passenden Icon und dem Namen der PWA.

Damit lässt sich nun die App ganz wie bei einer nativen App installieren, vom Homebildschirm starten und ohne Browseransicht verwenden. Die als nächstes betrachtete Funktion ist der Zugriff auf die Kamera, den internen Speicher, sowie das File Sharing System des Smartphones. Der Zugriff auf den internen Speicher sowie die Kamera ist ohne Schwierigkeiten per HTML möglich. Mit einem input-tag und die Angabe des Dateityps „image/*“ können Bilder in den Browser hochgeladen werden. Diese Art der Bildverwendung

funktioniert sowohl im Betrieb an einem Desktop-PC als auch an einem Smartphone. Bei einem Desktop-PC wird der Nutzer in jedem Falle an das Dateisystem weitergeleitet, in dem die passende Bilddatei ausgewählt werden kann [52], es gibt keinen Kamerazugriff. Durch das Hinzufügen des Attributes „capture“ kann erreicht werden, dass an einem Smartphone eine automatische Weiterleitung an die Kamera erfolgt [52].

Im PWA Prototyp wurde der Kamera- und Bildspeicherzugriff auf genau diese Weise gelöst. In einem Pop-Up Fenster befinden sich zwei verschiede Buttons, die jeweils ein HTML input-tag repräsentieren. Das eine leitet an das interne Dateisystem weiter, das andere an die Kamera. Über ein „oninput“-Attribut wird bei erfolgreichem Upload eine JavaScript Funktion ausgelöst, die das Bild an das Backend versendet, wo es gespeichert wird. Dort ist das Bild per URL verfügbar. Gleichzeitig wird die URL des verwendeten Bildes zusammen mit den anderen Informationen des Rezeptes im Backend gespeichert, von wo aus das Bild immer wieder in der App aufgerufen werden kann.

Etwas komplexer ist die Umsetzung des Zugriffs auf das File Sharing des Smartphones. Dies wird durch die Web Share API ermöglicht. Über das Navigator-Objekt, einem JavaScript-Objekt, das Informationen über den Browser beinhaltet, kann eine Methode aufgerufen werden, die das Teilen von URLs, Texten oder Dateien ermöglicht [53]. Voraussetzung dafür ist die Verwendung von HTTPS, einem aktiven Methodenaufruf durch den Nutzer und der Angabe von mindestens einem zu teilenden Inhalt [53]. Der Aufruf dieser Methode ist allerdings nicht in allen Browsern und nicht an Desktop-PCs verfügbar.

```
if (navigator.share !== undefined) {
  navigator
  .share({
    title,
    text
  })
  .then(() => console.log("Shared!"))
  .catch(err => console.error(err));
} else {
  window.location = `mailto:?subject=${title}&body=${title + "\n" + text}`;
}
```

*Abbildung 15 - Quellcode der PWA zum Teilen von Rezepten*

*Quelle: Eigene Aufnahme*

Der Nutzer löst eine Methode zum Teilen von Informationen in der PWA gezielt über einen Button aus. Mit dieser Methode werden die Daten des Rezeptes aus dem Backend abgefragt. Sobald der Nutzer im File-Sharing-Systems seines Smartphones seine gewünschte Methode des Teilens abgefragt hat, werden die Rezeptinhalte geteilt. Außerdem wurde, wie in Abbildung 15 zu sehen, eine Fehlerbehandlung der Teilfunktion für nicht unterstütze Browser eingebaut. Ist die verwendete Methode „navigator.share()“ nicht definiert, so wird der Benutzer automatisch an ein E-Mail-Programm weitergeleitet. Dort wird eine automatisch generierte E-

Mail mit Betreff und den Inhalten des Rezeptes erstellt. Der Benutzer kann sie nach Angabe der Zieladresse versenden und so sein Rezept teilen.

Außerdem wurde im Prototyp mit Hilfe eines Service Workers eine Offline-Funktionalität ermöglicht. Die Verwendung eines Service Workers bedarf verschiedene Ereignisse zur Registrierung, Installation und Aktivierung. Dabei werden die Ereignisse nicht alle von derselben JavaScript-Datei ausgelöst. Die Aktivierung des Service Workers erfolgt über die Hauptdatei des JavaScripts, die in den HTML-Seiten der Webseite eingebunden wird. In der Registrierungs-Funktion besteht die Möglichkeit, den Service Worker für Push-Benachrichtigungen vorzubereiten. Mit der Hilfe von VAPID Keys und der Berechtigung des Nutzers kann hier ein Subscription Objekt für das Versenden von Push Benachrichtigungen erstellt werden. Da die native Original App diese Funktion nicht beinhaltet, wurde sie im Prototyp nicht umgesetzt. Die Registrierung in der PWA besteht nur aus dem Aufruf der Funktion „navigator.serviceWorker.register(‚/sw.js‘)“, inklusive einer Fehlerbehandlung. Diese Funktion wird nur ausgeführt, wenn der Browser, vertreten durch das Navigator-Objekt, über die Funktionalität eines Service Workers verfügt.

Ist die Registrierung erfolgreich, so wird der Prozess an die JavaScript-Datei des Service Workers weitergeleitet. Die Datei besteht aus drei verschiedenen Abschnitten, die jeweils mit einem Event-Listener aufgerufen werden. Zuerst folgt die Installation des Service Workers. Diese beinhaltet das Speichern von davor ausgewählten assets, somit Kerninhalten der PWA, um eine Application-Shell-Architektur zu ermöglichen. Diese assets werden in einem statisch definierten und konstanten Array festgelegt und beinhalten die Startseite des Prototyps, die die Funktionalität gewährleistenden JavaScript-Dateien, die verwendeten CSS-Dateien, die manifest.json Datei, die relevante Informationen über die Anwendung beinhaltet sowie eine „fallback page“, die der Nutzer bei dem Anfordern eines ohne Internet nicht verfügbaren Datensatzes zu sehen bekommt. Um diese Daten vorab zu speichern, wird ein neuer Bereich des Caches, der statische Cache, geöffnet. Sind alle Dateien gesichert und im Cache gespeichert, ist die Installation abgeschlossen und eine erste funktionsfähige Version ohne Internet gewährleistet. Die anschließend ausgeführte Aktivierung, ausgelöst nach der Terminierung eines alten Service Workers oder beim erstmaligen Aufruf der Webseite, löscht alle alten Versionen des Caches. Dies ermöglicht eine Versionierung des Caches. Nach einer großen Änderung im Programmcode besteht die Möglichkeit, die konstant vergebenen Namen der Cache-Verzeichnisse zu verändern. Nach der Aktivierung des neuen Service Workers wird geprüft, ob die gespeicherten Caches den neu gewählten Versionsnamen besitzen. Ist das nicht der Fall, wird das Cache-Verzeichnis mit altem Versionsnamen, inklusive aller darin enthaltener Inhalte gelöscht. Damit werden die gespeicherten Inhalte auf eine aktuelle Version gebracht.

Das dynamische Caching, zu sehen in Abbildung 16, wird durch das Auslösen eines Events bei der Anfrage von Daten vom Client an den Browser ermöglicht. Das fetch-Event kann vom Service Worker abgefangen und bearbeitet werden. Im Prototyp umfasst dieses Event die Speicherung von dynamischen Inhalten, die Überprüfung der Größe des dynamischen Caches, sowie die Bereitstellung der „fallback page“.

```
//Fetch events
self.addEventListener('fetch', evt => {
    if(evt.request.url.includes('firestore.googleapis.com') == false) {
      evt.respondWith(
        caches.match(evt.request).then(cacheRes => {
          //Either return value of the cache or open the dynamic cache to cache the ressource for the next request
          return cacheRes || fetch(evt.request).then(fetchRes => {
            return caches.open(dynamicCacheName).then(cache => {
              cache.put(evt.request.url, fetchRes.clone());
              //Check cached items size
              limitCacheSize(dynamicCacheName, 100);
              return fetchRes;
            })
          });
        }).catch(() => {
          //Implement fallback page if request can´t be found and can´t be cached
          if (evt.request.url.indexOf('.html') > -1) {
            return caches.match('/pages/fallback.html');
          }
        })
      );
    }
```

*Abbildung 16 - Quellcode vom dynamischen Cachings*

*Quelle: Eigene Aufnahme*

Nach dem Auslösen durch den Event-Listener erfolgt eine Abfrage, ob die vom Client angefragte Ressource an das Backend Google Firebase geht (siehe Kapitel 4.2.2 Backend mit Google Firebase). Ist dies der Fall, so wird die Anfrage weitergeleitet und nicht gespeichert. Inhalte aus dem Backend werden über das Browser Feature Indexed DB gespeichert. Handelt es sich um eine Anfrage, die nicht an Firebase geht, wird sie mit den bereits gecachten Inhalten abgeglichen. Liegt die angefragte Ressource im Cache vor, so wird sie aus dem Cache zurückgegeben. Liegt die angefragte Ressource noch nicht im Cache vor, so wird die Anfrage an den Server weitergeleitet und die Daten werden im dynamischen Cache-Verzeichnis gespeichert. Nachdem das erfolgt ist, wird die Anzahl der im dynamischen Cache gespeicherten Inhalte überprüft.

Um nicht zu viele Inhalte zu speichern und mögliches fehlerhaftes doppeltes Speichern zu vermeiden, kann über die selbst erstellte Hilfsfunktion „limitCacheSize(name, number)“ ein Cache Verzeichnis auf eine bestimmte Menge an gespeicherten Inhalten begrenzt werden. Wird die Obergrenze der zu speichernden Inhalte überschritten, so wird der älteste Inhalt gelöscht.

Wenn dieser Prozess erfolgt ist, wird die angefragte Ressource an den Client weitergegeben. Sollte ein Fehler auftreten, vor allem dann, wenn eine Ressource angefragt wird, die aufgrund fehlender Internetverbindung nicht angefragt werden kann, wird eine Fehlerbehandlung

durchgeführt. Diese Fehlerbehandlung besteht aus dem Bereitstellen der bereits angesprochenen „fallback page“. Diese wird bereits bei der Installation als statischer Inhalt neben den assets gespeichert und steht daher immer ohne Internetverbindung zur Verfügung.

Besonders relevant für die Arbeit mit Service Workern ist ihr Lebenszyklus. Beim einfachen Neuladen einer Webseite erneuert sich der Service Worker nicht. Um eine neue Instanz eines neuen Service Workers zu erhalten, muss entweder der Browser komplett neu gestartet werden, Änderungen im Quellcode des Service Workers müssen angewendet werden oder der Service Worker wird per Hand in den Entwicklereinstellungen des Browsers terminiert. Dieser Lebenszyklus ist vor allem bei der Arbeit an einer Webseite teilweise gewöhnungsbedürftig, da Entwickler daran denken müssen, nach Änderungen im Quellcode, die nicht den Service Worker betreffen, eine neue Service Worker Instanz zu erzeugen, um die modifizierten Inhalte korrekt zu cachen.

### 4.2.2 Backend mit Google Firebase

Als ein geeignetes Backend mit Datenbank bietet sich Google Firebase an. Der zu Google gehörende Dienst bietet verschiedene Möglichkeiten, um Entwickler bei der Erstellung von Apps und Webseiten zu unterstützen. Firebase ist ein Backend-as-a-Service und kann speziell für jede Applikation als Server, API oder Datenbank angepasst werden [54]. Da Firebase direkt von Google gehostet wird, wird Entwicklern unter Umständen viel Arbeit abgenommen. Vor allem dann, wenn die erstellte Anwendung, wie der PWA Prototyp, ein Frontend bezogenes Projekt ist, für das ein einfaches und fremd gehostetes Backend ausreicht. Das Backend und die Datenbank des PWA Prototypen bildet Google Firebase. Dafür wurden die Dienste Firestore Database und Firebase Storage in Anspruch genommen. Die Verbindung zwischen Anwendung und Firebase läuft über eine Konfiguration, die nach dem Erstellen eines Accounts für die App von der Firebase-Web-Oberfläche kopiert werden kann.

Diese beinhaltet einen „apiKey“, eine Projekt ID und Domain, eine App-ID sowie verschiedene andere Identifikatoren, anhand derer die App eine Verbindung zum richtigen Firebase-Projekt herstellen kann. Diese Konfiguration muss, wie zu sehen in Abbildung 17, am Ende aller HTML-Dateien hinzugefügt werden. Gleichzeitig wird eine globale Variable „db“ erstellt, über die die Firestore Database als Datenbank direkt angesprochen werden kann.

```
<!-- The core Firebase JS SDK is always required and must be listed first -->
<script src="https://www.gstatic.com/firebasejs/8.2.4/firebase-app.js"></script>
<script src="https://www.gstatic.com/firebasejs/8.2.4/firebase-firestore.js"></script>

<script>

  // Your web app's Firebase configuration
  // For Firebase JS SDK v7.20.0 and later, measurementId is optional
  var firebaseConfig = {

    apiKey: "AI                                    Rmk",
    authDomain: "bachelor-        firebaseapp.com",
    projectId: "bachelor-        ",
    storageBucket: "bachelor-        appspot.com",
    messagingSenderId: "6            4",
    appId: "1:6                                    :a967",
    measurementId: "G-V          5"

  };

  // Initialize Firebase
  firebase.initializeApp(firebaseConfig);
  const db = firebase.firestore();

</script>
```

*Abbildung 17 - Quellcode der Verbindung zu Google Firebase*

*Quelle: Eigene Aufnahme (IDs sind unkenntlich gemacht)*

Die Firestore Database organisiert Daten auf verschiedenen Ebenen. Die oberste Ebene bilden dabei „collections“ (im folgenden Sammlungen). Die Sammlungen sind vergleichbar mit einer Tabelle in einer SQL-Datenbank und besitzen einen Namen, der als identifizierender Schlüssel und Pfad der darin liegenden Daten gilt.

Eine Ebene tiefer finden sich Dokumente. Sie sind vergleichbar mit einer Zeile in einer Tabelle einer SQL-Datenbank. Diese besitzen eine identifizierende ID, sowie beliebige Felder, die mit Daten gefüllt werden. Ein Dokument muss mindestens ein Feld besitzen, um es zu erstellen. Einem Feld wird wieder ein Name als ID zugewiesen, ebenso wie einem Datentyp mit dem das Feld gefüllt wird. Zu den möglichen Datentypen zählen die gängigen Datentypen wie strings, integer, booleans, usw. Eine Besonderheit von Firebase ist, dass einem Dokument wieder eine Sammlung untergeordnet werden kann. Es kann somit eine tiefe Hierarchie von Datensätzen über die IDs als Dateipfade erstellt werden. Dokumente und deren Felder können über die Verbindung zu einer App hinzugefügt und mit Werten versehen, aber auch verändert und gelöscht werden.

Im PWA-Prototyp besteht die Datenhaltung aus drei verschiedenen Sammlungen. Eine dieser Sammlungen enthält Rezepte. Ein Rezept ist folglich ein Dokument. Der Name in der Datenbank eines Rezeptes stellt eine einzigartige ID dar. Ein Rezept hat verschiedene Felder, die die vom Nutzer eingegeben Daten speichern. Außerdem besitzt ein Rezept ein Feld mit dem Namen „URL“, die eine URL speichert, über die auf das vom Nutzer hinzugefügte Bild zugegriffen werden kann. Die nächste Sammlung bilden die Kategorien. Diese besitzen analog zu den Rezepten eine eindeutige ID als internen Namen und als Felder die vom Nutzer

gewählten Daten. Die dritte Sammlung beinhaltet die Verknüpfung der Rezepte zu den Kategorien. Diese Sammlung speichert, vergleichbar zu einer SQL-Verknüpfungstabelle, ein Dokument für jede Zugehörigkeit eines Rezeptes zu einer Kategorie. Dieser Datensatz besitzt wieder eine eigene ID und die IDs des Rezeptes und der Kategorie als Felder. Durch eine Suche nach einer Kategorie-ID können so alle zugehörigen Rezepte gefunden werden. Gleichzeitig können bei der Suche nach einer Rezept-ID alle zugehörigen Kategorien gefunden werden.

Die Verbindung und der Zugriff auf die Datenbank läuft im Prototyp über eine eigene JavaScript-Datei, um eine möglichst klare 3-Schichtenarchitektur zu ermöglichen. Diese Architektur besteht dabei aus der Präsentationsschicht mit den HTML-Dateien, der Logikschicht, die die Funktionalität hinter den Bedienelementen der Seite mit der JavaScript-Datei „ui.js“ ermöglicht und der Datenhaltungsschicht, die aus der Kommunikation mit dem Firestore-Database-Service in der JavaScript-Datei „db.js“ besteht.

Die Funktionen in der für die Datenbank zuständigen JavaScript-Datei sind zwei verschiedenen Kategorien zuzuordnen. Einerseits Funktionen, die Daten aus der Datenbank laden, andererseits Funktionen, die Daten in der Datenbank speichern. Die Beschaffung von Informationen aus der Datenbank läuft über die Methode „onSnapshot()“, die einen Listener für eine Sammlung der Datenbank erstellt. Diese Funktionen werden durch den Listener automatisch beim erstmaligen Aufruf der PWA aufgerufen und immer dann, wenn der Datensatz einer betroffenen Sammlung verändert wird. Innerhalb der Funktionen wird über eine Zählschleife mit der Methode „snapshot.docChanges.forEach()“ jede Veränderung in der Datenbank übernommen. Um Daten auf der Basis eines bedingten Wertes, wie einer ID, zu erhalten, wird eine von der Logik angestoßene Funktion aufgerufen. Diese asynchrone JavaScript Funktion lädt eine ganze Sammlung aus der Datenbank und führt eine Suche nach dem, anhand der Bedingung, richtigen Datum aus. Um Daten der Datenbank hinzuzufügen oder bereits vorhandene Daten zu bearbeiten, wird eine asynchrone Funktion von der Logik in der Datenhaltung angestoßen. Diese Funktion fügt über die Methode „db.collection(‚*name*‘).add(‚*datensatz*‘)“ Daten hinzu. Das Überarbeiten von bereits vorhandenen Daten kann unter Hinzunahme der ID, analog zum Hinzufügen, mit der Methode „.change()“ gelöst werden.

Der PWA-Prototyp enthält für die erstellten Rezepte Bilder. Grundsätzlich ist es möglich Bilder und Dateien in einer Datenbank als BLOB zu speichern. Dies bringt einen Vorteil bezüglich der Datensicherheit mit sich. Allerdings ist die Speicherung von Dateien in einem Dateisystem die aktuelle Herangehensweise der Wahl. Google Firebase bietet mit dem Firebase Storage ein Dateisystem zu Speicherung an. Auf dort abgespeicherte Dateien kann über eine eigene

URL zugegriffen werden. Die Bilder werden im Prototyp nach der Auswahl durch den Benutzer in einen Container geladen. Die Datenhaltung greift auf die Datei in diesem Container zu. Diese Datei, die das Bild enthält, wird nach dem aktuellen Datum benannt, bekommt einen passenden Bilddateityp und wird in den Storage geladen. Anschließend wird eine URL des Bildes erzeugt und in der Firestore Database unter dem passenden Rezept abgespeichert. Über diese URL wird das Bild in die PWA eingefügt.

### 4.2.3 Serverapplikation mit Node.js

Um den PWA-Prototyp tatsächlich testen und verwenden zu können, bedarf es zwei Voraussetzungen. Erstens muss eine Verwendung der App über HTTPS gewährleistet sein. Außerdem bedarf es eine Lösung, die auf einem externen Gerät verwendet werden kann. Das lokale Hosten einer Webseite via Localhost ermöglicht zwar die Verwendung der PWA Technologien, die eigentlich HTTPS benötigen, aber beim Aufruf der Webseite mit einem externen Gerät über die IP-Adresse des Servers funktioniert dies nicht mehr. Gleichzeitig sollte es auch möglich sein, von außerhalb des eigenen Netzwerkes die PWA zu verwenden. Um diese Probleme zu lösen, bedarf es ebenfalls zwei Elemente. Erstens die Verwendung einer Serverapplikation, wie in diesem Falle mit der Hilfe von Node.js Express. Zweitens wird zum Erreichen von HTTPS eine eigene Domain, inklusive Weiterleitung auf den richtigen Server via IP-Adresse inklusive HTTPS Zertifikat benötigt.

```
const express = require("express");
const https = require("https");
const path = require("path");
const fs = require("fs");

const app = express();

// Set static path
app.use(express.static(path.join(__dirname, "/")));

const sslServer = https.createServer(
    {
        key: fs.readFileSync(path.join(__dirname, 'cert', 'privkey.pem')),
        cert: fs.readFileSync(path.join(__dirname, 'cert', 'fullchain.pem'))
    },
    app
)

//Declaring server port and publish on port
const port = 443;

sslServer.listen(port, () => console.log('Secure Server started on Port 443'))
```

*Abbildung 18 - Quellcode der Serverapplikation*

*Quelle: Eigene Aufnahme*

Um eine Serverapplikation mit der Laufzeitumgebung Node.js verwenden zu können, müssen Node.js und der Node Package Manager, kurz NPM, installiert werden. Im root-Verzeichnis

des Servers wird eine neue JavaScript-Datei mit dem Namen server.js hinzugefügt. Diese Datei enthält, wie in Abbildung 18 zu sehen, die Serverumgebung des PWA-Prototypen.

Zuerst werden einige benötigte Bibliotheken, wie zum Beispiel Node.js Express oder die HTTPS Bibliothek importiert. Anschließend wird eine Konstante mit dem Namen „app“ erstellt, die den neuen Express Server darstellt. Dieser bekommt das passende Startverzeichnis, bevor die für HTTPS benötigten Zertifikate hinzugefügt werden. Nachdem der Server die passenden Zertifikate erhalten hat, wird dem Server ein passender Port zugewiesen. Nun kann der Server nach dem Start über einen sicheren Kanal laufen. Um den Server überhaupt starten zu können und die Zusammenhänge zwischen der eigentlichen Anwendung und dem Server zu erhalten, wird der NPM benötigt. Über NPM können benötigte Bibliotheken heruntergeladen werden. Außerdem werden mit der Hilfe von NPM die benötigten Abhängigkeiten hergestellt und die Bedienung des Servers ermöglicht. Um der Anwendung NPM hinzuzufügen, muss per Konsolenbefehl in das Verzeichnis der Anwendung navigiert und ein Initialisierungsbefehl ausgeführt werden. Dieser Befehl fügt der Anwendung eine JSON-Datei mit dem Namen package.json hinzu. Die JSON-Datei enthält relevante Informationen über die Zusammenhänge der Anwendung, wie zum Beispiel die Datei, die bei Start des Servers ausgeführt werden soll. Im PWA-Prototyp ist die beim Start ausgeführte Datei die index.html. Gleichzeitig können in der package.json Datei Skripte hinzugefügt werden, mit denen der Server per Konsole gestartet, gestoppt oder in einen Testbetrieb versetzt werden kann.

Der PWA-Prototyp wird demnach lokal gehostet. Um von einem anderen Netzwerk auf den Computer, der den Server darstellt, zugreifen zu können, muss im Router des Netzwerks der entsprechende Port freigegeben werden. Wenn im Router der entsprechende Port freigegeben wurde, werden alle Anfragen an den Router unter diesem Port an den entsprechenden angesprochenen Server weitergegeben. Der verwendete Port 443 stellt den Standard Port für HTTPS Anfragen unter dem TCP-Protokoll dar. Daher werden die im Netzwerk eingehenden Anfragen an eine HTTPS-Verbindung zu einem Webserver an den entsprechenden Computer, der als Server dient, weitergeleitet.

Um eine bessere User Experience zu gewährleisten, die öffentliche IP-Adresse nicht verwenden zu müssen und das HTTPS-Zertifikat zu erhalten, wurde dem Prototyp eine statische Domain hinzugefügt. Mit der Hilfe des Dienstes No-IP[7] können Anfragen an den Server nun über eine URL, anstatt einer öffentlichen IP-Adresse laufen. Hinter der Adresse grexbachelor.hopto.org verbirgt sich die öffentliche IP-Adresse des Servers. Alle Anfragen an

[7] https://www.noip.com/

diese URL werden an den Server weitergeleitet. So kann der Nutzer eine feste und statische URL verwenden und die öffentliche IP-Adresse wird nicht mehr sichtbar.

Diese neu vergebene Domain wird ebenfalls für den Erhalt eines Zertifikates für die HTTPS-Verbindung benötigt. Das Zertifikat für den PWA-Prototyp wurde mit Hilfe des Zertifizierungsservice Let's Encrypt[8] erstellt. Let's Encrypt ist eine freie Zertifizierungsstelle für die Öffentlichkeit. Ihr Ziel ist es, kostenlos und benutzerfreundlich digitale Zertifikate für die Aktivierung von HTTPS auszugeben. Durch die Verwendung eines ACME-Protokolls über einen ACME-Client wird ohne menschliches Handeln ein Zertifikat ausgestellt [55]. Unter einem ACME-Protokoll versteht man ein von der kalifornischen Internet Security Research Group für Let's Encrypt entwickeltes Vorgehen, bei dem mit Hilfe des bereits angesprochenen Clients ein Certificate Signing Request erstellt und verifiziert werden kann, damit eine Zertifizierungsstelle, auch ohne menschlichen Kontakt, Zertifikate ausstellen kann [56]. Nach Installation und Konfiguration des ACME-Clients übernimmt dieser die Verifizierung und Kommunikation mit der Zertifizierungsstelle [56]. Im Falle des Prototyps wurde als ACME-Client die Anwendung Certbot[9] verwendet. Certbot ist ein Open-Source-Tool, das mit der Zertifizierungsstelle Let's Encrypt interagiert, um Webseiten eine HTTPS Verwendung zu ermöglichen [57].

Nach der Installation und Bevollmächtigung von Certbot als ACME-Client für den Server gibt der Administrator die betreffende Domain an. Der ACME-Client generiert ein Schlüsselpaar und gibt bei der Zertifizierungsstelle an, dass der Server, über den er verfügt, eine Domain kontrolliert [55, 56]. Anschließend bekommt der Client eine Aufgabe, um zu beweisen, dass er die angegebene Domain auch wirklich kontrolliert. Diese Aufgabe kann das Signieren einer Nonce, eine zufällig generierte Nummer, mit dem generierten privaten Schlüssel sein. Ist die Aufgabe erfüllt, wird die Zertifizierungsstelle benachrichtigt und beginnt mit der Überprüfung [55, 56]. Bei erfolgreicher Überprüfung wird der Agent über sein nun „autorisiertes Schlüsselpaar" [55] dazu berechtigt, Zertifikate zu verwalten. Der Agent stellt nun ein mit seinem Schlüsselpaar signiertes Certificate Signing Request an die Let's Encrypt Zertifizierungsstelle. Diese überprüft die Signatur und stellt ein Zertifikat für die verwaltete Domain aus [55].

Mit der Hilfe dieses Vorgehens wurde der PWA ein HTTPS-Protokoll hinzugefügt. Durch die geöffneten Ports, in Verbindung mit der Verwendung einer Domain inklusive HTTPS-Zertifikat, kann nach Start des Servers von überall auf die PWA zugegriffen und alle Features verwendet werden.

[8] https://letsencrypt.org/
[9] https://certbot.eff.org/

# 5. Ergebnisse

Die Durchführung des Thinking Aloud mit jeweils 6 Personen pro Anwendung und die anschließend durchgeführte Umfrage haben verschiedene Ergebnisse hervorgebracht. Im Folgenden werden erst die Ergebnisse des Thinking Alouds vorgestellt, anschließend die der Umfrage. Die Analyse der dargestellten Ergebnisse im Untersuchungskontext folgt im nächsten Kapitel.

## 5.1 Ergebnisse des Thinking Aloud

Alle der Teilnehmer waren in der Lage die gestellten Aufgaben sowohl mit der nativen Original, als auch mit der Prototyp PWA App zu absolvieren. Sobald die richtige App, bzw. die richtige Webseite gefunden war, stellte die Installation weder auf die klassische Weise via Appstore noch über den Browser bei der PWA große Problem dar:

*„… Okay installieren. Lädt noch runter. Schießen sie den Browser und klicken sie auf das neu erscheinende Icon. Okay dann mache ich das mal. Ach, da ist es ja, schön! Das sieht ja gut aus. …“* (PWA1)

*„Oh, das ging aber schnell. …“* (PWA4)

*„… Oh, da kommt ja unten direkt eine Meldung und dann kann ich das direkt installieren. Anscheinend wird dem auch vertraut, das ist doch schonmal gut. […] Da ist es. Sogar mit App Icon. Ja das sieht gut aus. …“* (PWA5)

*„… Scrollen Sie herunter, bis die App Kochbuch von Flose sehen. Da ist es. Installieren. Das ist schonmal einfach. …“* (NAT1)

*„ … Suchen Sie nach dem Kochbuch von Flose in der Kategorie Essen und Trinken. So Kochbuch. Hier ist es doch.“* (NAT5)

Lediglich bei der Suche nach der richtigen Anwendung kamen bei einigen Testpersonen leichte Probleme auf:

*„Okay, ja. Ich öffne den Webbrowser und gebe in die Suchleiste die URL ein. Das mit der URL eingeben ist so eine Sache. […] Normalerweise wäre ein Direktlink der Way to Go.“* (PWA5)

*„Scrollen Sie bis Sie das Kochbuch von Flose in der Kategorie Essen und Trinken finden. […] Achso das habe ich gar nicht gesehen. Das war jetzt für mich nicht so ersichtlich. …“* (NAT2)

Anschließend bedurfte es einer ersten Navigation durch die Anwendung. Die Navigation, sowie das Zurechtfinden in der Anwendung wurde sowohl in der PWA als auch in der nativen App als grundsätzlich positiv wahrgenommen:

*„Ich finde hier alles sehr übersichtlich. Ich finde hier alles ganz klar strukturiert, also ich persönlich.“* (PWA2)

*„[Die Handhabung] ist sehr entspannt.“* (PWA4)

*„Ich finde die Handhabung einfach. Es ist eigentlich so wie man es von einem Android kennt. Alles gut erklärt. Super. Auch ich der ja kein Profi in solchen Dingen ist findet sich wunderbar zurecht. Ich würde sagen Top. Ganz einfach.“* (NAT2)

*„ … Eigentlich total einfach. …“* (NAT5)

*„Es war alles wirklich simpel erklärt hier. Du hast mir zwar ein bisschen geholfen ab und zu, aber wenn man sich ein bisschen Zeit nimmt kommt man mit Sicherheit locker alleine dadurch.“* (NAT6)

Was allerdings bei vielen Testpersonen, sowohl bei der nativen App, als auch bei der PWA zu leichten Unklarheiten und Problemen geführt hat, ist die Ableitung von der Funktion verschiedener Buttons. Einige Buttons, wie zum Beispiel der grüne runde Button um in das Menü zum Hinzufügen von Rezepten zu kommen, wurden von allen Teilnehmern schnell erkannt:

*„ … Achso plus, logisch. …“* (PWA1)

*„ … Erstellen Sie ein neues Rezept. Dann würde ich sagen ich gehe hier auf das Plus. …“* (PWA2)

*„ … Ich würde jetzt erstmal auf Plus drücken, weil das sieht danach aus als könnte ich hier ein Rezept hinzufügen. …“* (PWA5)

„ … Ich mache mal ein Hauptgericht. Wahrscheinlich hier unten auf dem Plus. …“ (NAT1)

*„Ich gehe jetzt mal davon aus, dass das hier der Button ist.“* (NAT3)

Andere Bedienelemente wurden von den Testpersonen kritisiert. Testperson PWA5 merkte an, dass das Speichern eines Rezeptes ohne Speichern Button, nur über den Pfeil zurück, aus ihrer Sicht nicht besonders logisch sei: *„Okay ich muss einfach auf den Pfeil zurück gehen und dann ist das anscheinend gespeichert. Das ist ja interessant. Ja, Usability technisch hätte ich mich über einen Speichern-Button gefreut. Weil so hat man irgendwie das Gefühl, man macht jetzt was weg oder es geht was kaputt.“* Andere Testpersonen hatten ebenfalls ein Problem mit dem Speichern eines erstellten Rezeptes. Testperson PWA4 gab nach der ersten

Aufforderung zum Zurückkehren und Speichern des Gerichtes an, es würde nicht gehen. Nach auffälliger Suche nach einem entsprechenden Bedienelement gab Testperson PWA6 auf Nachfrage an, sie suche nach einem Knopf zum Speichern, der so nicht existiere. Ein anderes kritisiertes Bedienelement war die Möglichkeit, um eine Kategorie hinzuzufügen. Dies ist in den Anwendungen über einen Button mit drei Punkten in der Navigationsleiste rechts möglich. Testperson NAT3 äußerte Kritik im Hinblick auf die User Experience für der Technologie unverbundene Benutzer: *„Ich würde sagen das ist natürlich für erfahrene User schnell zu ersehen, für andere symbolisieren die drei Punkte Einstellungen, die man eher selten gebraucht. Ich würde da eher ein Plus machen. So ein grüner Button regt da schon mehr an. Die App ist ja auch für Leute, die jetzt nicht unbedingt die krassesten Technology Natives sind."*

Ebenfalls teilweise kritisiert wurde die Darstellung von Textfeldern. Die grundsätzliche Aufmachung von Textfeldern kam bei den Testpersonen gut an. Testperson PWA1 gab an, Inhalte würden sich gut in die Textfelder eintragen lassen. Testperson PWA5 merkte an, die Aufmachung durch ein Hervorheben des Textfeldes sei ansprechend gewählt: *„ … Das funktioniert doch schonmal ganz gut soweit. Immerhin öffnet sich hier die Tastatur, wenn man in die Textfelder klickt. Und die Textfelder sind auch gehighlightet, je nach dem welches man gerade ausgewählt hat. …"* Kritik gab es von verschiedenen Testpersonen zu verschiedenen Aspekten der Textfelder. Testpersonen PWA3 und PWA4 gaben an, die Labels über einzelnen Textfeldern nach der Dateneingabe würden fehlen, konkret von Testperson PWA4: *„ … Hier sollte aber noch Zutaten und Zubereitung stehen."* Eine andere Kritik kam von Testperson PWA5, die die Anpassung der Weite der Textfelder an das verwendete Gerät kritisiert hat: *„Es scheint aber nicht so, dass das hier 100% für die Weite des Displays angepasst ist. …"* Neben den kritischen Punkten, die die Optik der Felder betreffen, gab Testperson NAT1 an, sie könne anhand des Textfeldes nicht direkt einsehen, in welcher Form die Daten einzutragen sind: *„Ich weiß jetzt nicht genau, ob ich alles hintereinander tippen soll?"*

Was bei allen Testpersonen funktioniert hat und positiv aufgefallen ist, ist die Möglichkeit Fotos von dem erstellten Rezept hinzuzufügen. Alle Testpersonen konnten ohne große Probleme entweder ein Bild aus dem internen Speicher des Gerätes auswählen, oder selbst ein Foto aufnehmen:

*„ … Das mit den Fotos hat super geklappt, die Größe ist sehr gut und passt sich gut dem Display an. …"* (PWA1)

*„Okay dann nehme ich mal ein Bild auf. Foto benutzen. Okay hat funktioniert."* (PWA4)

*„ … Wählen Sie ein Foto aus dem internen Speicher. Aus der Galerie nehme ich mal ein Bild. Hier oben das sieht gut aus. Ja ich denke mal das ist jetzt drin. …"* (NAT1)

*„Ich finde auf jeden Fall nice, […], dass das auch durch das Bild schnell wiedererkannt werden kann. Also auch in den Kategorien, du siehst ich habe das so und so benannt und man hat das mit Text und Bild in Erinnerung. …“* (NAT3)

Die einzige Kritik, die zu den Fotos kam, wurde von Testperson PWA5 geäußert. Die Testperson kritisierte dabei die Ladezeit, die es brauchte, um das aufgenommene Bild in der App anzuzeigen und eine fehlende Ladeanimation, um diese Ladezeit zu überbrücken: *„Da habe ich gerade ein Bild aufgenommen und was passiert jetzt? Aha, ja. Ich hätte jetzt fast erwartet, dass eine Art Ladeanimation kommt, sodass man einen Hinweis darauf bekommt, dass man kurz warten muss, weil da ist für ca. 1 Sekunde nichts passiert. Ich habe da gedacht habe ich was falsch gemacht, aber sonst ist das ganz gut. …“*

Alle Testpersonen konnten ihre erstellten Rezepte gut innerhalb der App wiederfinden. Grund dafür sind unter anderem die Kategorien und die Suchfunktion, deren Umsetzung bei allen Testpersonen funktioniert hat und sie unterstützt hat:

*„ … Überprüfen sie, ob ihr Gericht in der Kategorie wiederzufinden ist. Okay Kategorien, und dann gehe ich auf Kaffeetrinken, und da sind die Waffeln drin. Super. Habe ich. […] Und dann suchen. Da ist es. Okay funktioniert, habe ich. …“* (PWA2)

*„Okay also kann ich auf Kategorien gehen und meine Kategorie anklicken und dann sehe ich alle Gerichte zu dieser Kategorie. Das ist ja herrlich! Das ist ja schön. Dann sieht man ja immer welches Gericht der Kategorie zugeordnet ist, wenn ich im Kategorie Reiter auf die Kategorie klicke. […] Verwenden Sie die Suchfunktion, um ihr Gericht zu suchen. Mal gucken. Bockwurst einfach mal eingeben, auf Suchen klicken und ich finde tatsächlich mein Rezept und kann es auch anklicken und da ist es wirklich. …“* (PWA5)

*„ … Dann gehe ich jetzt in die Kategorien, Grillen. Ja ist doch drin. Funktioniert einwandfrei. Sehr gut. Einfach. […] Es wird bei der Suche sofort angezeigt. Hervorragend.“* (NAT2)

*„Ich finde auf jeden Fall sehr nutzerfreundlich, dass man die verschiedenen Kategorien zur Übersicht machen kann. …“* (NAT4)

Ebenfalls positiv aufgefallen ist die Möglichkeit, Rezepte per Nachricht, E-Mail oder Messenger zu teilen. Testperson PWA3 war begeistert von der Aufmachung des Teilens mit Zutaten und Zubereitung: *„Das ist cool. Guck die Zutaten sind da drinnen, die Zubereitung. Ja das ist gut. Das ist wirklich cool.“* Testperson PWA5 merkte positiv an, dass man die Textnachricht im Notfall auch noch einmal bearbeiten könne: *„Da oben ist das Teilen Icon, da klicke ich dann drauf, wenn ich das Rezept angeklickt habe. Und dann kann ich das per Nachricht teilen, sehr schön. Oh, ein leckeres Rezept für, hier mit Zutaten, hier steht alles drin und ich kann den Text im Notfall auch noch bearbeiten. Und hier steht dann alles wichtige drin halt in Textform. …“*

Bei den Testpersonen mit der nativen App funktionierte die Funktion, um Rezepte zu teilen ebenfalls gut. Die Testperson NAT4 gab auf Nachfragen an, die Möglichkeit ein Rezept zu teilen, sei auf jeden Fall praktisch.

Eine der bedeutsamsten Funktionen, die eine PWA ausmacht und auch der Prototyp beinhaltet, ist die Verfügbarkeit ohne Internetverbindung. Bei 11 von 12 beteiligten Testpersonen am Thinking Aloud hat die Verwendung ohne Internet ganz ohne Probleme funktioniert:

*„Ach und dann da einfach Wlan aus. Ja da ist es ja. Perfekt auch ohne Internet."* (PWA2)

*„Dann beende ich mal die App komplett, dann ist das jetzt mal zu. Dann öffne ich die App mal wieder. Okay sie lädt zumindest. Wenn ich aktualisiere, dann passiert das, was ich erwarte. […] Das Rezept ist auch ohne Internet vollständig. Wenn ich jetzt ein anderes Rezept anklicke dann funktioniert das nicht, weil ich das vorher nicht geöffnet hatte."* (PWA5)

*„Ohne Internet? Ja da ist es ja schon. Ja kann ich benutzen."* (PWA6)

*„Dann mache ich mal aus und gehe mal komplett raus und starte neu. […] „Das funktioniert denke ich mal. Ja klappt."* (NAT2)

*„Genau schalten Sie die Internetverbindung aus. […] Ich könnte ja mal ganz rausgehen. Ich könnte jetzt auf Vorspeisen gehen, weil ich ja weiß ich will eine Vorspeise haben. Und da ist es"* (NAT5)

*„Ich habe jetzt aber kein Wlan mehr. […] Ja gut das braucht es ja auch nicht. Ist alles noch da."* (NAT6)

Probleme gab es bei der Verwendung ohne Internet bei Testperson PWA4. Nach dem Ausschalten des Internets gab es eine deutlich schlechtere Performance der App und es wurden Inhalte, wie die Bilder, nicht vernünftig angezeigt. Die Testperson gab an: *„Klappt nicht so gut. Wenn man die App ohne Internet verwendet dann hängt das erst so ein bisschen und geht dann. Ist irgendwie komisch."* Möglicherweise ist dieses Problem auf einen überfüllten Cache oder einen Fehler im Vorgang der Speicherung der Inhalte in den Cache zurückzuführen. Dieses Problem zeigt jedoch, dass die Inhalte der PWA nur in 5 von 6 Fällen auch ohne Internetverbindung auf eine zuverlässige Weise zur Verfügung gestellt wurden.

Des Weiteren wurden nicht direkt zuzuordnende kleinere Probleme im PWA Prototyp festgestellt. Die Testperson PWA4 gab an, dass die Animation der Sterne für die Vergabe einer Bewertung nicht optimal funktioniere: *„Man sieht das hier, wenn man die Sterne antippt. Der Stern wird nicht sofort grün. …"* Ein anderes kleines Problem wurde von Testperson PWA5 angesprochen. Die Navigation zwischen Menü und Übersicht wird etwas erschwert, da man

immer in den Rezept-Tab der Übersicht zurückkehrt, auch wenn man vorher im Kategorien-Tab war: *„ … Wenn man die Kategorie hinzugefügt hat springt man wieder auf Rezepte zurück, anstatt bei den Kategorien zu bleiben. …“* Die Testperson PWA5 hat ebenso kleine optische Mängel an den Feldern zum Hinzufügen einer Kategorie, festgestellt: *„ … Wenn ich auf neue Kategorie klicke bekomme ich hier ein leicht durchsichtiges Feld. Okay die löschen Button der anderen Kategorien sind über dem Feld neue Kategorie erstellen und man könnte da was löschen während man eine neue Kategorie erstellt. …“*

## 5.1 Ergebnisse der Umfrage

Alle Teilnehmer haben im Anschluss an das Thinking Aloud an der Umfrage teilgenommen. Zu den Ergebnissen ist anzumerken, dass pro Gruppe nur je 6 Personen teilgenommen haben und daher die Repräsentativität der absoluten Zahl fraglich ist. Aussagekräftiger ist der Vergleich zwischen den Ergebnissen der beiden Gruppen, aber auch dabei ist die Repräsentativität fraglich. Zur vereinfachten Darstellung wurden die Fragen der Umfrage in vier Teilbereiche aufgeteilt. Dabei handelt es sich um die Installation, die Navigation, bzw. Handhabung, die für die PWA kritischen Punkte und die Schwerpunkte der Testpersonen. Die gesamten Ergebnisse der Umfrage finden sich im Anhang (Anhang 1.3: Gesamtergebnisse der Umfrage). Alle Testpersonen konnten Werte zwischen -2 und +2 vergeben. Dabei war -2 die schlechteste und +2 die beste Wertung. Aufgrund der positiven Gesamtresonanz beider Anwendungen sind die Ergebnisse in einem positiven Bereich ausgefallen, weshalb die dargestellten Werte grundsätzlich als positive Werte zu verstehen sind.

| Block | Durchschnitt Nativ | Durchschnitt PWA | Abweichung |
|---|---|---|---|
| Installation | 2 | 1,915 | 4% |
| Navigation / Handhabung | 1,23 | 1,7075 | 28% |
| PWA kritische Punkte | 1,832 | 1,7 | 8% |

*Abbildung 19 - Ergebnisse der Umfrage*

*Quelle: Eigene Anfertigung*

In Abbildung 19 finden sich die Ergebnisse der ersten drei Blöcke der Umfrage. Diese sind jeweils in die Ergebnisse der Testpersonen mit der nativen App und die Testpersonen der PWA getrennt. Dahinter finden sich die Abweichungen zwischen den beiden Anwendungen. Der dargestellte Block der Installation umfasst den Installationsvorgang und die Darstellung der App auf dem Homebildschirm. Die Bewertung der nativen App ist mit der bestmöglichen Wertung von 2 um 4% besser bewertet worden als die PWA. Die Abweichung ergibt sich, da

die Testpersonen der PWA die Installation etwas schwieriger empfunden haben als die Testpersonen der nativen App. In Block 2 haben die Testpersonen die Navigation, bzw. Handhabung der Anwendungen bewertet. Hier findet sich die größte durchschnittliche Abweichung der Blöcke. Die PWA wurde mit einer Bewertung von 1,7 um 28% besser bewertet als die native App. Außerdem wurde die PWA bei jeder Frage dieses Blockes besser bewertet als die native App. Ein besonders ausschlaggebender Punkt dafür ist die Bewertung der Ableitbarkeit der Funktionalität von Buttons und Bedienelementen aus deren Design oder Label. Hier wurde die PWA 55% besser bewertet als die native App. Da das Design der Bedienelemente allerdings bei beiden Anwendungen ähnlich ist zeigt sich hier möglicherweise die fehlende Repräsentativität der Umfrage. Trotzdem ist eine Abweichung der Bewertung der Navigation, bzw. Handhabung über acht verschiedene Fragen und jeweils sechs Personen pro Gruppe um 28% ein deutliches Indiz für eine gute Benutzererfahrung bei der Handhabung der PWA. Block 3 fasst die Fragen zusammen, die spezifisch die Funktionalität der

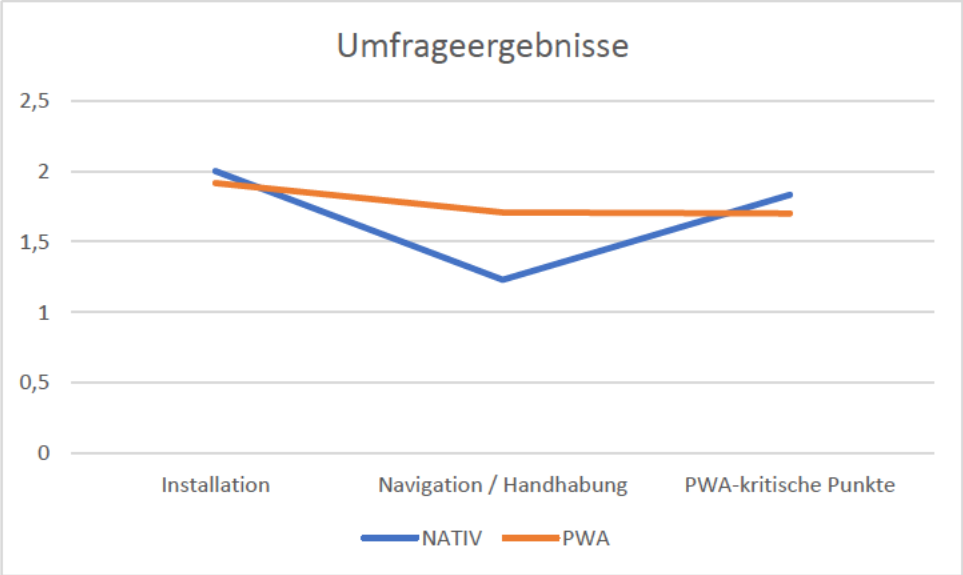


*Abbildung 20 - Grafische Darstellung der Umfrageergebnisse*

*Quelle: Eigene Anfertigung*

Technologien bewerten, die einen für eine PWA kritischen Punkt darstellen.

Die Fragen in Block 3 zielten darauf ab den Zugriff auf Kamera und Speicher, die Ladezeiten, die Funktionalität ohne Internetverbindung und den Zugriff auf File Sharing System des Gerätes zu bewerten. Über alle fünf verschiedenen Fragen hat die native App hier mit einer Wertung von 1,832 um 8% besser abgeschnitten als die PWA. Hier ist spezifisch die Bewertung der Funktion ohne Internetverbindung zu nennen. Die PWA hat hier mit einer Wertung von 1 ihre schlechteste Bewertung in der Umfrage bekommen und zu 100% schlechter als die native App abgeschnitten. Dies ist unter Umständen damit zu erklären, dass die Internetverbindung im Thinking Aloud von Testperson PWA4 nicht zuverlässig funktioniert hat. Trotz dieser großen Diskrepanz zwischen der Bewertung der Funktion ohne

Internetverbindung im Hinblick auf die Probleme bei Testperson PWA4 ist die gesamte Abweichung in Block 3 mit 8% relativ gering.

Insgesamt ergibt sich über alle 15 die Bewertung der Anwendung betreffenden Fragen eine Gesamtwertung für die native App von 23 von 30 möglichen Punkten. Die PWA wurde mit einer Wertung von 25,99 von möglichen 30 Punkten über die 15 Fragen bewertet. Damit ergibt sich, dass der PWA-Prototyp im Mittel um 13% besser bewertet wurde als die native Original App. Dies zeigt sich vor allem deutlich in Abbildung 20. Die Fläche zwischen den beiden Graphen an den Stellen, an denen der orangene Graph einen höheren Wert hat, ist deutlich größer als die Fläche zwischen den Graphen, an denen der blaue Graph einen höheren Wert hat.

Die vier letzten Fragen der Umfrage haben ein allgemeines Stimmungsbild über die Schwerpunkte der Wertschätzung bei einer App von den 12 Testpersonen eingeholt. Daraus haben sich deutliche Unterschiede bei der Wertschätzung der einzelnen Punkte gezeigt. Wie in Abbildung 21 zu sehen, legen die Testpersonen den größten Wert auf eine zuverlässige Anwendung, auch ohne Internet. Dies rechtfertigt auf der einen Seite den Vorzug der PWA durch die Offline-Verfügbarkeit gegenüber einer klassischen Web App. Auf der anderen Seite ergibt sich, dass bei einer Schwerpunktlegung nach den Ergebnissen des Stimmungsbildes auf die Umfrageergebnisse der Vorsprung der PWA deutlich sinken würde, weil das Umfrageergebnis der nativen App bezüglich der Funktion ohne Internet um 100% besser ist als das der PWA.

| Schwerpunkte der Testperson | Ergebnis |
|---|---|
| Geringer Akku-Verbrauch | 1,415 |
| Geringer Speicherplatz-Verbrauch | 1,085 |
| Zuverlässige Anwendung (ohne Internet) | 1,42 |
| Besondere Features (Benachrichtigungen, Kamera, Kalender, etc.) | 0,75 |

*Abbildung 21 - Ergebnisse des Stimmungsbildes*

*Quelle: Eigene Anfertigung*

An die zweite Stelle setzen die Testpersonen den Akku-Verbrauch, erst an dritter Stelle folgt der Speicherplatz-Verbrauch. Dies ist besonders interessant, bezogen auf die Ergebnisse von Malavolta (2017) [12] und Gambhir et. al. (2018) [24]. Die Untersuchungen von Malatova haben gezeigt, dass die Verwendung eines Service Workers negative Auswirkungen auf den Stromverbrauch des verwendeten Gerätes haben können [12]. Gleichzeitig zeigten die Untersuchungen von Gambhir et. al., dass eine PWA zumeist einen geringeren Speicherplatz-Verbrauch besitzt, als das Android oder iOS Pendant [24]. Durch diese Ergebnisse könnte die Schwerpunktlegung ebenfalls eine Verbesserung der schlechteren Bewertung der nativen

App, gegenüber der PWA darstellen. An vierter und mit relativ großem Abstand letzter Stelle setzten die Testpersonen ihren Bedarf nach besonderen Features, wie Push Benachrichtigungen, Kamerazugriff, Kalenderzugriff, etc.

# 6. Diskussion

Die Literaturrecherche zu Beginn hat gezeigt, dass es relativ problemlos möglich ist, anstatt einer klassischen Web App eine PWA zu entwickeln. Der Entwicklungsprozess hat gezeigt, dass es auch möglich ist, eine native Anwendung als prototypische PWA zu entwickeln. Die Zeitersparnis, die durch die nicht benötigte Doppeltentwicklung für mehrere Betriebssysteme erzielt wird, und der gleichzeig geringere Bedarf an Programmierkenntnissen bei der Entwicklung einer PWA anstatt einer nativen App stellen einen Vorteil dar. Die offene Frage, nach der Ersetzbarkeit einer nativen App durch eine PWA im Hinblick auf User Experience soll durch die Untersuchungen dieser Arbeit besser untersucht werden.

Durch den Thinking Aloud Prozess haben alle Testpersonen einen praktischen Einblick in die Handhabung der jeweiligen Anwendung bekommen. An dieser Stelle ist anzumerken, dass alle Testpersonen in der Lage waren, die Aufgaben mit beiden Anwendungen erfolgreich zu absolvieren. Die Installation der verschiedenen Anwendungen im Thinking Aloud wurde von beiden Gruppen gleich gut absolviert. Das zeigt, dass die bisher untypische Installation einer Webseite als App auf dem Gerät eine für den Nutzer zwar neue, aber trotzdem gute Möglichkeit darstellt. Die Ergebnisse der Umfrage bestätigen diese Erkenntnis, mit einer nahezu gleichen Bewertung von beiden Gruppen.

Die allgemeine Handhabung der Anwendungen zeigte keine großen Unterschiede beim Thinking Aloud. Grundsätzlich kamen manche Testpersonen besser mit der Handhabung klar, andere schlechter. Probleme mit der Handhabung einzelner Nutzer kamen jedoch in beiden Fokusgruppen vor. Nach genauer Beobachtung lagen die Probleme einzelner Nutzer nicht an den Unterschiedenen zwischen PWA und nativer App, sondern an der persönlichen technischen Affinität und Verständnis. Ähnlich sind die Ergebnisse des Thinking Alouds bezüglich Bedienelementen. Grundsätzlich sind alle Benutzer mit den Bedienelementen der Anwendung zurechtgekommen. Es gab jedoch Kritik an verschiedenen Stellen für die fehlende Ableitbarkeit der Funktionen, vor allem für unerfahrene Nutzer. Diese Kritikpunkte beziehen sich aber nicht auf die Unterschiede zwischen der nativen App und der PWA. Beide Anwendungen waren in dieser Hinsicht ähnlich und die angegeben Kritik besitzt keine Aussagekraft für die Vergleichbarkeit der User Experience dieser beiden Anwendungen. Die Ergebnisse der Umfrage (siehe 5.1 Ergebnisse der Umfrage) bilden hier mit 28% einen großen

Unterschied zwischen PWA und nativer App ab. Grundsätzlich wäre ein leicht abweichender Wert zu Gunsten der PWA durch geringe Abweichungen zwischen den beiden Anwendungen zwar möglich, eine Abweichung in dieser Höhe erscheint aber als nicht richtig. Die Abweichung des Umfragewertes in dieser Höhe ist vermutlich durch das persönliche Empfinden der Teilnehmer zu erklären, wodurch die Bedeutung dieser Abweichung für ein Ergebnis fraglich ist.

Die ersten PWA-kritischen Punkte, wie der Zugriff auf Bilder und deren Einbindung, wie auch der Zugriff auf File-Sharing-System wurde mit beiden Anwendungen für gut befunden. Dies zeigt, dass der Zugriff auf das Gerät mit einer PWA je nach Art der Anwendung genauso realisierbar ist wie mit einer nativen App. Die Verwendung ohne Internet funktionierte beim Großteil beider Teilnehmergruppen zwar, es gab jedoch Probleme bei einer Testperson mit PWA-Anwendung. Hier ist außerdem zu nennen, dass durch die Möglichkeit des Cachings zwar eine Verwendung ohne Internet realisiert werden kann, aber dann auch nur auf die Inhalte zugegriffen werden kann, die gespeichert wurden. Ein Gesamtzugriff auf alle Inhalte ohne Internet, wie es bei einigen nativen Apps möglich ist, ist bei einer PWA schwer zu realisieren und wenn, dann inklusive langer Ladezeiten durch das Pre-Caching aller Inhalte bei jedem Aufruf der Seite, wodurch die Umsetzung nicht besonders sinnvoll wäre. Diese etwas schlechtere Nutzererfahrung ohne Internetverbindung zeigt sich auch in den Umfrageergebnissen. Die gesamten PWA-kritischen Aspekte wurden in der Umfrage zwar bei der PWA nur mit 8% schlechter bewertet, die zuverlässige Internetverbindung allerdings um 100%.

Insgesamt gesehen lässt sich sagen, dass die durch das Thinking Aloud und die Umfrage ermittelte User Experience von der prototypisch entwickelten PWA und der ausgewählten nativen App nicht nennenswert voneinander abweicht. Beide Anwendungen haben Vorteile und Nachteile gezeigt und haben für alle Nutzer eine zufriedenstellende Nutzererfahrung gebracht. Damit ist gezeigt, dass unter Umständen eine PWA anstelle einer nativen App entwickelt werden kann, um die Mehrkosten für den Zeitaufwand der mehrfachen Entwicklung der Anwendung einzusparen. Gleichzeitig kann dabei eine ebenso zufriedenstellende User Experience erreicht werden.

An dieser Stelle sei jedoch anzumerken, dass in dieser Arbeit die User Experience einer relativ simplen Single-User-Anwendung ermittelt und verglichen wurde. Eine PWA kann zwar für bestimmte native Anwendungen eine Alternative darstellen, aber für komplexere Anwendungen sollte im Vorfeld der Entwicklung abgewogen werden, ob die Komplexität der Anwendung auch nur mit Web-Technologien bei gleicher User Experience wie bei einer

nativen App erreicht werden kann. Die PWA kann in vielen Fällen eine Alternative zu einer nativen App sein, jedoch ist zu bezweifeln, dass sich die PWA für alle Anwendungen eignet.

# 7. Fazit

Abschließend kann gesagt werden, dass Apps immer mehr an Bedeutung erhalten, da ihre Verwendung immer mehr in unseren Alltag integriert wird. Folglich rückt auch die Entwicklung von Apps immer mehr in den Fokus der Softwareentwickelung, wodurch verbesserte Möglichkeiten für die Entwicklung von Apps erarbeitet werden. Da sich im Smartphone Markt ein Oligopol, anstatt wie im Markt für Betriebssysteme für klassische PCs mit Windows ein mehr oder weniger Monopol, gebildet hat, stellt die zur Abdeckung des Marktes benötigte Mehrfachentwicklung der Software für Android und iOS Geräte einen deutlichen Mehraufwand dar. Um dieses Problem zu lösen wurden bereits verschiedenste Cross-Plattform Development Ansätze entwickelt und erforscht, es werden aber auch immer neue Ansätze vorgestellt.

Die Progressive Web Apps sind ein vielversprechender Ansatz, um die Mehrfachentwicklung für Android, iOS und Webseiten zu vermeiden. Viele ihrer Vorteile und Möglichkeiten wurden bereits erforscht und in dieser Arbeit erläutert. Offen blieb jedoch die Frage nach der User Experience, verglichen mit einer nativen App.

In dieser Arbeit wurde die Entwicklung einer PWA anhand einer bereits bestehenden nativen App durchgeführt. Dazu kann gesagt werden, dass bei der Entwicklung der App keine Probleme aufgetreten sind und die Entwicklung einer PWA als Alternative für eine simple native App möglich ist. Eine PWA zu entwickeln ist bei Kenntnis von Web-Technologien vergleichbar simpel.

Bei dem Vergleich der User Experience mit der Hilfe von Thinking Aloud waren alle teilnehmenden Testpersonen in der Lage die gestellten Aufgaben zu absolvieren. Die Testpersonen haben bei beiden Anwendungen verschiedene Punkte sowohl positiv als auch negativ hervorgehoben. Im Thinking Aloud Prozess wurden damit zwar bei beiden Anwendungen kritische Punkt ermittelt, jedoch konnte keine sichtbar schlechtere User Experience von einer der beiden Anwendungen festgestellt werden. Bei der Ermittlung eines messbaren Wertes der User Experience mit Hilfe einer Umfrage wurden zwar für unterschiedliche Bewertungspunkte geringfügig unterschiedlich gute Bewertungen erzielt, aber zusammenfassend über alle Kategorien weicht das Ergebnis beider Anwendungen nur geringfügig voneinander ab. Damit lässt sich begründet sagen, dass die Untersuchungen dieser Arbeit das Ergebnis liefern, dass die User Experience beider Apps nicht nennenswert voneinander abweicht. Diese Arbeit hat damit belegt, dass es möglich ist eine App auch als

PWA, anstatt als native App zu programmieren, ohne Einbußen in Sachen User Experience in Kauf nehmen zu müssen.

Es ist allerdings zu bemerken, dass die für diesen Vergleich verwendete Anwendung verglichen mit anderen nativen Apps simpel ist und keine besonderen Grafiken oder Rechenleistung benötigt. Es ist fraglich, ob PWAs als Alternative für komplexe native Apps umgesetzt werden können und ob bei der Umsetzung die User Experience trotzdem bei beiden Anwendungen überzeugen kann. Außerdem kann die hier vorgenommene Untersuchung alleine kein allgemeingültiges Ergebnis zur User Experience von PWAs geben. Um diese beiden Fragen weiter nachzugehen, muss in Zukunft die User Experience von PWAs weiter untersucht werden. Außerdem muss die Entwicklung von komplexen Anwendungen als PWA und die aus der Anwendung resultierende User Experience weiter erforscht werden.

# Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, insbesondere keine anderen als die angegebenen Informationen aus dem Internet. Diejenigen Paragrafen der für mich geltenden Prüfungsordnungen, die etwaige Betrugsversuche betreffen, habe ich zur Kenntnis genommen.

Der Speicherung meiner Bachelor- bzw. Masterarbeit zum Zweck der Plagiatsprüfung stimme ich zu. Ich versichere, dass die elektronische Version mit der gedruckten Version inhaltlich übereinstimmt.

_____ _____

(Datum) (Unterschrift)

# Literatur


[1] E. Angulo und X. Ferre, „A Case Study on Cross-Platform Development Frameworks for Mobile Applications and UX“ in *the XV International Conference*, Puerto de la Cruz, Tenerife, Spain, 2014, S. 1–8, doi: 10.1145/2662253.2662280.

[2] H. Heitkötter, T. A. Majchrzak und H. Kuchen, „Cross-platform model-driven development of mobile applications with md 2“ in *the 28th Annual ACM Symposium*, Coimbra, Portugal, 2013, S. 526, doi: 10.1145/2480362.2480464.

[3] X. Li, Y. Jiang, Y. Liu, C. Xu, X. Ma und J. Lu, „User Guided Automation for Testing Mobile Apps“ in *2014 21st Asia-Pacific Software Engineering Conference (APSEC)*, Jeju, South Korea, 122014, S. 27–34, doi: 10.1109/APSEC.2014.13.

[4] A. Biørn-Hansen, T.-M. Grønli, G. Ghinea und S. Alouneh, „An Empirical Study of Cross-Platform Mobile Development in Industry“, *Wireless Communications and Mobile Computing*, Jg. 2019, S. 1–12, 2019, doi: 10.1155/2019/5743892.

[5] App Annie, „State of Mobile 2021“, 2021. [Online]. Verfügbar unter: https://www.appannie.com/en/go/state-of-mobile-2021/. Zugriff am: 8. März 2021.

[6] Statista Research Department, „Marktanteile der führenden mobilen Betriebssysteme an der Internetnutzung mit Mobiltelefonen in Deutschland von Januar 2009 bis Januar 2021“, 2021. [Online]. Verfügbar unter: https://de.statista.com/statistik/daten/studie/184332/umfrage/marktanteil-der-mobilen-betriebssysteme-in-deutschland-seit-2009/#professional. Zugriff am: 8. März 2021.

[7] A. Ahmad, K. Li, C. Feng, S. M. Asim, A. Yousif und S. Ge, „An Empirical Study of Investigating Mobile Applications Development Challenges“, *IEEE Access*, Jg. 6, S. 17711–17728, 2018, doi: 10.1109/ACCESS.2018.2818724.

[8] H. Heitkötter, S. Hanschke und T. A. Majchrzak, „Evaluating Cross-Platform Development Approaches for Mobile Applications“ in *Lecture Notes in Business Information Processing, Web Information Systems and Technologies*, W. van der Aalst et al., Hg., Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, S. 120–138, doi: 10.1007/978-3-642-36608-6_8.

[9] A. Biørn-Hansen, T. A. Majchrzak und T.-M. Grønli, „Progressive Web Apps for the Unified Development of Mobile Applications“ in *Lecture Notes in Business Information Processing, Web Information Systems and Technologies*, T. A. Majchrzak, P. Traverso, K.-H. Krempels und V. Monfort, Hg., Cham: Springer International Publishing, 2018, S. 64–86, doi: 10.1007/978-3-319-93527-0_4.

[10] R. Nunkesser, „Beyond web/native/hybrid“ in *ICSE '18: 40th International Conference on Software Engineering*, Gothenburg Sweden, 05272018, S. 214–218, doi: 10.1145/3197231.3197260.

[11] S. Tandel und A. Jamadar, *Impact of Progressive Web Apps on Web App Development*, 2018.

[12] I. Malavolta, G. Procaccianti, P. Noorland und P. Vukmirovic, „Assessing the Impact of Service Workers on the Energy Efficiency of Progressive Web Apps“ in *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, Buenos Aires, Argentina, 22.05.2017 - 23.05.2017, S. 35–45, doi: 10.1109/MOBILESoft.2017.7.

[13] A. Biørn-Hansen, T. A. Majchrzak und T.-M. Grønli, „Progressive Web Apps: The Possible Web-native Unifier for Mobile Development“ in *13th International Conference on Web Information Systems and Technologies*, Porto, Portugal, 4252017, S. 344–351, doi: 10.5220/0006353703440351.

[14] S. Xanthopoulos und S. Xinogalos, „A comparative analysis of cross-platform development approaches for mobile applications“ in *the 6th Balkan Conference in Informatics*, Thessaloniki, Greece, 2013, S. 213, doi: 10.1145/2490257.2490292.

[15] A. Biørn-Hansen und G. Ghinea, „Bridging the Gap: Investigating Device-Feature Exposure in Cross-Platform Development“ in *Hawaii International Conference on System Sciences*, 2018, doi: 10.24251/HICSS.2018.716.
[16] H. Heitkötter, T. A. Majchrzak, B. Ruland und T. Weber, „Evaluating Frameworks for Creating Mobile Web Apps“ in *9th International Conference on Web Information Systems and Technologies*, Aachen, Germany, 582013, S. 209–221, doi: 10.5220/0004356702090221.
[17] S. Charkaoui, Z. Adraoui und E. H. Benlahmar, „Cross-platform mobile development approaches“ in *2014 Third IEEE International Colloquium in Information Science and Technology (CIST)*, Tetouan, Morocco, 102014, S. 188–191, doi: 10.1109/CIST.2014.7016616.
[18] G. de Andrade Cardieri und L. M. Zaina, „Analyzing User Experience in Mobile Web, Native and Progressive Web Applications“ in *IHC 2018: 17th Brazilian Symposium on Human Factors in Computing Systems*, Belém Brazil, 10222018, S. 1–11, doi: 10.1145/3274192.3274201.
[19] P. LePage, F. Beaufort und T. Steiner, *Add a web app manifest.* [Online]. Verfügbar unter: https://web.dev/add-manifest/ (Zugriff am: 29. März 2021).
[20] Apple Development Team, *Configuring Web Applications.* [Online]. Verfügbar unter: https://developer.apple.com/library/archive/documentation/AppleApplications/Reference/SafariWebContent/ConfiguringWebApplications/ConfiguringWebApplications.html (Zugriff am: 30. März 2021).
[21] E. Bacon, *Enabling iOS Splash Screens for Progressive Web Apps: Everything you should know (2019).* [Online]. Verfügbar unter: https://blog.expo.io/enabling-ios-splash-screens-for-progressive-web-apps-34f06f096e5c (Zugriff am: 30. März 2021).
[22] I. Malavolta, K. Chinnappan, L. Jasmontas, S. Gupta und K. A. K. Soltany, „Evaluating the impact of caching on the energy consumption and performance of progressive web apps“ in *MOBILESoft '20: IEEE/ACM 7th International Conference on Mobile Software Engineering and Systems*, Seoul Republic of Korea, 07132020, S. 109–119, doi: 10.1145/3387905.3388593.
[23] J. Lee, H. Kim, J. Park, I. Shin und S. Son, „Pride and Prejudice in Progressive Web Apps“ in *CCS '18: 2018 ACM SIGSAC Conference on Computer and Communications Security*, Toronto Canada, 2018, S. 1731–1746, doi: 10.1145/3243734.3243867.
[24] A. Gambhir und G. Raj, „Analysis of Cache in Service Worker and Performance Scoring of Progressive Web Application“ in *2018 International Conference on Advances in Computing and Communication Engineering (ICACCE)*, Paris, 62018, S. 294–299, doi: 10.1109/ICACCE.2018.8441715.
[25] A. Semenov, *How Progressive Web Apps make the Web great again.* [Online]. Verfügbar unter: http://webagility.com/posts/how-progressive-web-apps-make-the-web-great-again (Zugriff am: 31. März 2021).
[26] Google Development Team, *Introduction to Service Worker.* [Online]. Verfügbar unter: https://developers.google.com/web/ilt/pwa/introduction-to-service-worker (Zugriff am: 31. März 2021).
[27] D. Swinhoe, *What is a man-in-the-middle attack?: How MitM attacks work and how to prevent them.* [Online]. Verfügbar unter: https://www.csoonline.com/article/3340117/what-is-a-man-in-the-middle-attack-how-mitm-attacks-work-and-how-to-prevent-them.html (Zugriff am: 31. März 2021).
[28] C. Enyinnaya, *Demystifying The Service Worker Lifecycle.* [Online]. Verfügbar unter: https://www.digitalocean.com/community/tutorials/demystifying-the-service-worker-lifecycle (Zugriff am: 31. März 2021).

[29] Google Development Team, *Lab: Scripting the Service Worker.* [Online]. Verfügbar unter: https://developers.google.com/web/ilt/pwa/lab-scripting-the-service-worker (Zugriff am: 1. April 2021).
[30] C. Liebel und G. Homberg, *Push API: Advanced Progressive Web Apps: Push Notifications Under Control - Part 2.* [Online]. Verfügbar unter: https://www.thinktecture.com/de/pwa/push-api/ (Zugriff am: 6. April 2021).
[31] C. Liebel und G. Homberg, *HTTP Web Push: Advanced Progressive Web Apps: Push Notifications Under Control - Part 3.* [Online]. Verfügbar unter: https://www.thinktecture.com/en/pwa/http-web-push/?utm_medium=social&utm_source=facebook&utm_campaign=Article+PWA+HTTP+Web (Zugriff am: 6. April 2021).
[32] Google Development Team, *Introduction to Push Notifications.* [Online]. Verfügbar unter: https://developers.google.com/web/ilt/pwa/introduction-to-push-notifications (Zugriff am: 6. April 2021).
[33] A. Osmani, *The App Shell Model.* [Online]. Verfügbar unter: https://developers.google.com/web/fundamentals/architecture/app-shell (Zugriff am: 7. April 2021).
[34] A. Russel, *Progressive Web Apps: Escaping Tabs Without Losing Our Soul.* [Online]. Verfügbar unter: https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/ (Zugriff am: 7. April 2021).
[35] C. Liebel und G. Homberg, *Notifications API: Advanced Progressive Web Apps: Push Notifications Under Control - Part 1.* [Online]. Verfügbar unter: https://www.thinktecture.com/de/pwa/push-notifications-api/ (Zugriff am: 7. April 2021).
[36] P. LePage und S. Richard, *What makes a good Progressive Web App?* [Online]. Verfügbar unter: https://web.dev/pwa-checklist/ (Zugriff am: 7. April 2021).
[37] P. Bengtsson, S. Neethling, N. Schonning, H. Willee und F. Scholz, *Introduction to progressive web apps.* [Online]. Verfügbar unter: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Introduction (Zugriff am: 7. April 2021).
[38] S. Luber und P. Schmitz, *Was ist Ende-zu-Ende-Verschlüsselung (E2EE)?* [Online]. Verfügbar unter: https://www.security-insider.de/was-ist-ende-zu-ende-verschluesselung-e2ee-a-727147/ (Zugriff am: 9. April 2021).
[39] C. A. Hall, „Web presentation layer bootstrapping for accessibility and performance“ in *the 2009 International Cross-Disciplinary Conference*, Madrid, Spain, 2009, S. 67, doi: 10.1145/1535654.1535671.
[40] M. Hassenzahl, „The Thing and I: Understanding the Relationship Between User and Product“ in *Funology 2: From Usability to Enjoyment*, M. Blythe und A. Monk, Hg., Cham: Springer International Publishing, 2018, S. 301–313, doi: 10.1007/978-3-319-68213-6_19.
[41] A. P. O. S. Vermeeren, E. L.-C. Law, V. Roto, M. Obrist, J. Hoonhout und K. Väänänen-Vainio-Mattila, „User experience evaluation methods“ in *the 6th Nordic Conference*, Reykjavik, Iceland, 2010, S. 521, doi: 10.1145/1868914.1868973.
[42] S. Iftikhar, A.-E. Guerrero-Roldán, E. Mor und D. Bañeres, „User Experience Evaluation of an e-Assessment System“ in *Lecture Notes in Computer Science, Learning and Collaboration Technologies. Designing, Developing and Deploying Learning Experiences*, P. Zaphiris und A. Ioannou, Hg., Cham: Springer International Publishing, 2020, S. 77–91, doi: 10.1007/978-3-030-50513-4_6.
[43] M. E. Joorabchi, A. Mesbah und P. Kruchten, „Real Challenges in Mobile App Development“ in *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Baltimore, Maryland, 102013, S. 15–24, doi: 10.1109/ESEM.2013.9.

[44] T. Majchrzak und T.-M. Grønli, „Comprehensive Analysis of Innovative Cross-Platform App Development Frameworks“ in *Hawaii International Conference on System Sciences*, 2017, doi: 10.24251/HICSS.2017.745.
[45] T. A. Majchrzak, A. Biørn-Hansen und T.-M. Grønli, „Progressive Web Apps: the Definite Approach to Cross-Platform Development?“ in *Hawaii International Conference on System Sciences*, 2018, doi: 10.24251/HICSS.2018.718.
[46] T. A. Majchrzak, J. Ernsting und K. Herbert, „Achieving Business Practicability of Model-Driven Cross-Platform Apps“, *Open Journal of Information Systems (OJIS)*, S. 3–14, 2015.
[47] W. Oliveira, W. Torres, F. Castor und B. H. Ximenes, „Native or Web? A Preliminary Study on the Energy Consumption of Android Development Models“ in *2016 IEEE 23rd International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Suita, 32016, S. 589–593, doi: 10.1109/SANER.2016.93.
[48] C. Rieger und T. A. Majchrzak, „Towards the definitive evaluation framework for cross-platform app development approaches“, *Journal of Systems and Software*, Jg. 153, S. 175–199, 2019, doi: 10.1016/j.jss.2019.04.001.
[49] T. Steiner, „What is in a Web View“ in *Companion of the Web Conference 2018*, Lyon, France, 2018, S. 789–796, doi: 10.1145/3184558.3188742.
[50] A. Ternauciuc und R. Vasiu, „Testing usability in Moodle: When and How to do it“ in *2015 IEEE 13th International Symposium on Intelligent Systems and Informatics (SISY)*, Subotica, Serbia, 92015, S. 263–268, doi: 10.1109/SISY.2015.7325391.
[51] A. H. JØRGENSEN, „Thinking-aloud in user interface design: a method promoting cognitive ergonomics“, *Ergonomics*, Jg. 33, Nr. 4, S. 501–507, 1990, doi: 10.1080/00140139008927157.
[52] P. Kinlan und M. Scales, *Capturing an Image from the User.* [Online]. Verfügbar unter: https://developers.google.com/web/fundamentals/media/capturing-images (Zugriff am: 5. Mai 2021).
[53] J. Medley, *Integrate with the OS sharing UI with the Web Share API: Web apps can use the same system-provided share capabilities as platform-specific apps* (Zugriff am: 5. Mai 2021).
[54] O. Wais, *Firebase – eine kurze Einführung.* [Online]. Verfügbar unter: https://mfg.fhstp.ac.at/development/webdevelopment/firebase-eine-kurze-einfuehrung/ (Zugriff am: 10. Mai 2021).
[55] Let's Encrypt, *Wie es funktioniert.* [Online]. Verfügbar unter: https://letsencrypt.org/de/how-it-works/ (Zugriff am: 17. Mai 2021).
[56] P. Nohe, *ACME Protocol: What it is and how it works.* [Online]. Verfügbar unter: https://www.thesslstore.com/blog/acme-protocol-what-it-is-and-how-it-works/ (Zugriff am: 2. Juni 2021).
[57] Certbot, *About Certbot.* [Online]. Verfügbar unter: https://certbot.eff.org/about/ (Zugriff am: 2. Juni 2021).

# Anhang

## Anhang 1.1: Umfrage

| Frage | Stimme nicht zu | Eher Nein | Neutral | Eher Ja | Stimme zu |
|---|---|---|---|---|---|
| Die Installation der App war einfach und hat mir keine Probleme bereitet | | | | | |
| Die Darstellung auf dem Home-Bildschirm ist ansprechend und die App leicht wiederzufinden | | | | | |
| Informationen werden in einfacher, natürlicher und logischer Weise präsentiert | | | | | |
| Die App zeigt dem Nutzer intuitiv den korrekten Weg auf und animiert ihn dazu, eine der Zielerreichung förderlichen Aktion durchzuführen | | | | | |
| Die App ist auch für unerfahrene Nutzer gut bedienbar | | | | | |
| Ein erstmaliger Nutzer kann die gängigsten Aufgaben/Funktionen ohne Anleitung erledigen/nutzen | | | | | |
| Es gibt einen einfachen und nachvollziehbaren Weg, sich zwischen Seiten und Sektionen und zurück zur Übersicht zu bewegen | | | | | |
| Dateneingabefelder enthalten Platzhalter mit vordefinierten Werten und zeigen die Struktur der einzugebenden Daten | | | | | |
| Elemente, die geklickt oder gedrückt werden können, sind OFFENSICHTLICH klickbar oder drückbar | | | | | |
| Die Funktionalität von Buttons und Bedienelementen ist von deren Labels oder Design ableitbar | | | | | |
| Der Zugriff auf die Kamera und/oder den internen Speicher funktioniert gut | | | | | |
| Die Einbindung der Bilder in der App ist optisch ansprechend gelöst | | | | | |
| Die Ladezeiten der App sind angemessen (nicht störend lang) | | | | | |
| Ohne Internetverbindung funktioniert die Anwendung gut | | | | | |
| Die Funktion Rezepte zu teilen funktioniert, ist gut umgesetzt und hilft mir weiter | | | | | |
| Auf geringen Akku-Verbrauch lege ich großen Wert | | | | | |
| Auf geringen Speicherplatzverbrauch lege ich großen Wert | | | | | |
| Auf zuverlässige Anwendung auch ohne Internetverbindung lege ich großen Wert | | | | | |
| Features wie Push-Benachrichtigungen, Zugriff auf Kamera oder den Kalender sind für mich wichtig | | | | | |

## Anhang 1.2: Zu absolvierende Aufgaben Thinking Aloud

| Nummer | Aufgabe |
|---|---|
| 1 | Erstellen Sie ein neues Rezept |
| 2 | Kehren Sie zurück zur Übersicht und schauen Sie sich anschließend Ihr Rezept an |
| 3 | Kehren Sie zurück zur Übersicht und erstellen Sie eine neue Kategorie |
| 4 | Gehen Sie in Ihr erstelltes Gericht und fügen Sie es Ihrer neuen Kategorie hinzu |
| 5 | Kehren Sie zurück zur Übersicht und klicken Sie auf Ihre neue Kategorie |
| 6 | Überprüfen Sie, ob sie Ihr Gericht wiederfinden |
| 7 | Löschen Sie Ihre neu erstellte Kategorie |
| 8 | Verwenden Sie die Suchfunktion, um Ihr Gericht zu suchen |
| 9 | Teilen Sie ihr Gericht per Nachricht / WhatsApp |
| 10 | Kehren Sie zur App zurück und schalten Sie die Internetverbindung aus |
| 11 | Schauen Sie sich nun erneut Ihr Rezept an und prüfen Sie die Verwendung ohne Internet |
| 12 | Schalten Sie das Internet wieder ein und löschen Sie Ihr Rezept |

## Anhang 1.3: Gesamtergebnisse der Umfrage

| Frage | Durchschnitt Nativ | Durchschnitt PWA | Abweichung |
|---|---|---|---|
| Die Installation der App war einfach und hat mir keine Probleme bereitet | 2 | 1,83 | 9% |
| Die Darstellung auf dem Home-Bildschirm ist ansprechend und die App leicht wiederzufinden | 2 | 2 | 0% |
| Informationen werden in einfacher, natürlicher und logischer Weise präsentiert | 1,5 | 2 | 25% |
| Die App zeigt dem Nutzer intuitiv den korrekten Weg auf und animiert ihn dazu, eine der Zielerreichung förderlichen Aktion durchzuführen | 1 | 1,5 | 33% |
| Die App ist auch für unerfahrene Nutzer gut bedienbar | 1,17 | 1,67 | 30% |
| Ein erstmaliger Nutzer kann die gängigsten Aufgaben/Funktionen ohne Anleitung erledigen/nutzen | 1,17 | 1,83 | 36% |
| Es gibt einen einfachen und nachvollziehbaren Weg, sich zwischen Seiten und Sektionen und zurück zur Übersicht zu bewegen | 1,5 | 1,83 | 18% |
| Dateneingabefelder enthalten Platzhalter mit vordefinierten Werten und zeigen die Struktur der einzugebenden Daten | 1,5 | 1,33 | 13% |
| Elemente, die geklickt oder gedrückt werden können, sind OFFENSICHTLICH klickbar oder drückbar | 1,33 | 2 | 34% |
| Die Funktionalität von Buttons und Bedienelementen ist von deren Labels oder Design ableitbar | 0,67 | 1,5 | 55% |
| Der Zugriff auf die Kamera und/oder den internen Speicher funktioniert gut | 2 | 2 | 0% |
| Die Einbindung der Bilder in der App ist optisch ansprechend gelöst | 1,5 | 2 | 25% |
| Die Ladezeiten der App sind angemessen (nicht störend lang) | 1,83 | 1,83 | 0% |
| Ohne Internetverbindung funktioniert die Anwendung gut | 2 | 1 | 100% |
| Die Funktion Rezepte zu teilen funktioniert, ist gut umgesetzt und hilft mir weiter | 1,83 | 1,67 | 10% |
| Auf geringen Akku-Verbrauch lege ich großen Wert | 1,5 | 1,33 | 13% |
| Auf geringen Speicherplatzverbrauch lege ich großen Wert | 1 | 1,17 | 15% |
| Auf zuverlässige Anwendung auch ohne Internetverbindung lege ich großen Wert | 1,67 | 1,17 | 43 |
| Features wie Push-Benachrichtigungen, Zugriff auf Kamera oder den Kalender sind für mich wichtig | 1,5 | 0 | 150% |