Hochschule Düsseldorf
University of Applied Sciences

Fachbereich Medien
Faculty of Media

Vergleich von Progressive Web Apps und nativen Android Apps

Norbert Libront

614902

Bachelorarbeit im Studiengang

BSc. Medieninformatik

November 2018

Betreuer:

Prof. Dr. Manfred Wojciechowski

Prof. Dr.-Ing. Thomas Rakow

Abstract

Die vorliegende Bachelorarbeit befasst sich mit der Fragestellung, inwiefern Progressive Web Apps zum jetzigen Zeitpunkt in der Lage sind, native Apps zu ersetzen. Dazu wurde ein Vergleich auf Basis ausgewählter Vergleichskategorien durchgeführt. Zur Eingrenzung des Umfangs beschränkte sich der Vergleich auf native Android Apps. Die beiden App-Typen wurden im Rahmen einer Literaturrecherche und auf Basis ausgewählter Vergleichskategorien untersucht und verglichen. Zudem wurde ein Prototyp einer Progressive Web App implementiert, um die Plattformunterstützung und Funktionalität zu überprüfen. Außerdem wurde auch ein Prototyp einer nativen Android App implementiert. Dabei wurde nur die Benutzeroberfläche umgesetzt, um das Look and Feel vergleichbar zu machen.

Die Untersuchung und der Vergleich haben gezeigt, dass Progressive Web Apps, anders als native Apps, mit Webtechnologien entwickelt und keine plattformspezifischen Programmiersprachen benötigt werden. Sie werden im Browser ausgeführt und sind dadurch Plattformunabhängig. Zudem wurde gezeigt, dass Progressive Web Apps über Funktionen und Eigenschaften verfügen, die bisher nur nativen Apps vorbehalten waren. Dennoch können sie noch nicht auf alle Funktionen und Komponenten eines mobilen Gerätes zugreifen. Darüber hinaus verfügen sie aber auch über Funktionen und Eigenschaften, die native Apps nicht bieten. Des Weiteren wurde mithilfe eines Prototyps die Plattformunterstützung von Progressive Web Apps überprüft. Die Überprüfung zeigte, dass die Unterstützung durch das Betriebssystem Android weit fortgeschritten und bei iOS hingegen nur sehr eingeschränkt ist. Dadurch variiert der Funktionsumfang je nach Browser und Betriebssystem. Die Untersuchung und der Vergleich der Prototypen zeigten, dass Progressive Web Apps aufgrund ihrer Eigenschaften, Funktionen und die app-ähnliche Benutzeroberfläche, ein ähnliches Look and Feel bieten können, wie native Apps. Außerdem wurde gezeigt, dass Progressive Web Apps, anders als native Apps, i. d. R. nicht über App Stores angeboten werden. Sie durchlaufen dadurch keinen Überprüfungs- und Freigabeprozess und sind nicht an die Richtlinien der App Stores gebunden. Stattdessen können sie über eine URL und über Suchmaschinen aufgerufen und aufgefunden werden. Die durchgeführte Untersuchung und der Vergleich haben gezeigt, dass Progressive Web Apps aufgrund des eingeschränkten Zugriffs auf Geräte- und Software-Funktionen sowie fehlender Browser- und Betriebssystemunterstützung, zum jetzigen Zeitpunkt native Apps noch nicht vollumfänglich ersetzen können.

Inhaltsverzeichnis

Abkürzungsverzeichnis

AOSP	Android Open Source Project
API	Application Programming Interface
APK	Android Package
App	Application
ART	Android Runtime
BLE	Bluetooth Low Energy
CSS	Cascading Style Sheets
DOM	Document Object Model
GATT	Generic Attribute
GPS	Global Positioning System
HAL	Hardware Abstraction Layer
HTML	Hypertext Markup Language
HTTPS	HyperText Transfer Protocol Secure
IDE	Integrierte Entwicklungsumgebung
JDK	Java Development Kit
JRE	Java Runtime Environment
JSON	JavaScript Object Notation
NDEF	NFC Data Exchange Format
NFC	Nahfeldkommunikation
OHA	Open Handset Alliance
SAF	Storage Access Framework
SDK	Software Development Kit
SQL	Structured Query Language
SSL	Secure Sockets Layer
UI	User Interface
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
Web App	Webapplikation

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Ausgangslage und Motivation

In den vergangenen Jahren verlagerte sich die Nutzung des Internets zunehmend auf mobile Geräte, wie Smartphones und Tablet-PCs. Durch die stetig wachsende Zahl an Benutzern mobiler Geräte, spielt die Entwicklung mobiler Webapplikationen eine immer wichtigere Rolle (vgl. Initiative D21, 2014). Jedoch nutzen Webapplikationen nicht alle Funktionen und Möglichkeiten, die von den mobilen Geräten zur Verfügung gestellt werden. Dies führt dazu, dass neben einer Webapplikation zusätzlich native Apps für die Plattformen Android und iOS entwickelt werden müssen. Webapplikationen unterscheiden sich bspw. auch in der Handhabung und Darstellung von nativen Apps. Zudem können diese nicht ohne Internetverbindung funktionieren, auf mobilen Geräten installiert werden, auf native Funktionen zugreifen sowie keine Push-Benachrichtigungen empfangen.

Mit Progressive Web Apps soll sich dies zukünftig ändern. Dabei werden die wichtigsten Merkmale und Charakteristiken von Webapplikationen und nativen Apps miteinander kombiniert. Progressive Web Apps sollen über Funktionen und Möglichkeiten verfügen, die bisher nur von nativen Apps angeboten wurden. Dazu zählen bspw. die Verfügbarkeit und Nutzbarkeit bei eingeschränkter oder keiner Internetverbindung, die Möglichkeit zur Installation von Progressive Web Apps auf mobilen Geräten sowie die Nutzung und der Empfang von Push-Benachrichtigungen (Push Notifications).

Progressive Web Apps wird derzeit bei den o. g. Aspekten ein hohes Potenzial beigemessen und sollen eine Alternative zu nativen Apps bieten (vgl. Petereit, 2016). Hieraus ergibt sich auch die Relevanz für meine Bachelorarbeit, diese beiden Ansätze zu untersuchen und miteinander zu vergleichen.

1.2 Zielsetzung

Im Rahmen dieser Arbeit wird aufgezeigt, inwiefern Progressive Web Apps zum jetzigen Zeitpunkt in der Lage sind, native Apps zu ersetzen. Dazu soll ein Vergleich auf Basis ausgewählter Vergleichskategorien durchgeführt werden. Zur Eingrenzung des Umfangs beschränkt sich der Vergleich auf native Android Apps.

Zunächst werden die drei verschiedenen App-Typen, Progressive Web Apps, Native Apps und Hybrid Apps sowie die verfügbaren mobilen Plattformen kurz vorgestellt. Anschließend werden auf Basis von ausgewählten Vergleichskategorien (siehe Aufzählung unten) die beiden App-Typen (native Android Apps und Progressive Web Apps) im Rahmen einer Literaturrecherche untersucht. Die Untersuchung und der darauffolgende Vergleich wird anhand der nachfolgenden Vergleichskategorien durchgeführt:

- **Entwicklung:** Welche Programmiersprachen sowie Entwicklungsumgebungen/Tools werden benötigt oder stehen zur Verfügung?
- **Technische und funktionale Möglichkeiten:** Welche technischen und funktionalen Möglichkeiten bieten die jeweiligen App-Typen (Offline-Nutzbarkeit, Benachrichtigungen, Geräte-Zugriff etc.)?
- **Plattformunterstützung:** Welche Plattformen/Browser unterstützen die beiden App-Typen? Gibt es Einschränkungen oder lassen sich beide App-Typen auf allen Plattformen/Browsern ausführen?
- **Look and Feel:** Welche Möglichkeiten gibt es, ein natives Look and Feel zu realisieren? Werden Style-Guides oder Empfehlungen zur Gestaltung der Benutzeroberfläche zur Verfügung gestellt? Hierbei wird auch iOS betrachtet, da sich die Gestaltung und das Layout der Benutzeroberfläche von Android unterscheidet.
- **Veröffentlichung und Verbreitung:** Welche Möglichkeiten zur Veröffentlichung und Verbreitung der beiden App-Typen gibt es und wie läuft diese ab?

Um die Plattformunterstützung und Funktionalität zu überprüfen, wird ein Prototyp, eine einfache Progressive Web App (News-App), mit den wesentlichen Merkmalen und Funktionen einer Progressive Web App, implementiert. Auch wird ein Prototyp einer nativen Android App implementiert. Bei diesem Prototyp wird nur die Benutzeroberfläche umgesetzt, um das Look and Feel vergleichbar zu machen. Funktionen, wie z. B. Benachrichtigungen, werden nicht umgesetzt, da zur Eingrenzung des Umfangs angenommen wird, dass diese von nativen Android Apps unterstützt und angeboten werden. Die Funktionen nativer Android Apps und dessen Umsetzbarkeit, werden jedoch im Rahmen der Untersuchung näher betrachtet.

Zur Beantwortung der Fragestellung erfolgt anschließend der Vergleich von Progressive Web Apps und nativen Android Apps. Der Vergleich basiert auf Grundlage der Ergebnisse,

die sich aus der Literaturrecherche auf Basis der ausgewählten Vergleichskategorien und den Erkenntnissen der prototypischen Implementierung, ergeben haben.

1.3 Aufbau der Arbeit

Im zweiten Kapitel werden zunächst die verschiedenen App-Typen (Progressive Web Apps, Native Apps, Hybrid Apps) sowie die verfügbaren mobilen Plattformen kurz vorgestellt. Dabei werden auch die wesentlichen Merkmale und Eigenschaften der App-Typen erwähnt. Zudem werden in diesem Kapitel die für die Untersuchung und den Vergleich ausgewählten Vergleichskategorien vorgestellt und dessen Auswahl und Relevanz erläutert. Im dritten und vierten Kapitel werden die beiden App-Typen, native Android Apps und Progressive Web Apps jeweils auf Basis der ausgewählten Vergleichskategorien im Rahmen einer Literaturrecherche untersucht und erläutert. Zu den Vergleichskategorien zählen u. a. Entwicklung, technische und funktionale Möglichkeiten, Plattformunterstützung, Look and Feel sowie die Veröffentlichung und Verbreitung. Im fünften Kapitel werden die zwei zu implementierenden Prototypen (Progressive Web App und native Android App) sowie deren Funktionen und Eigenschaften, die umgesetzt wurden, kurz beschrieben und vorgestellt. Im sechsten Kapitel erfolgt auf Grundlage der im dritten und vierten Kapitel erarbeiteten Ergebnisse, die sich aus der Literaturrecherche auf Basis der ausgewählten Vergleichskategorien und den Erkenntnissen der Implementierung ergeben haben, ein Vergleich von Progressive Web Apps und nativen Android Apps. Abschließend folgt im siebten Kapitel eine Zusammenfassung der verfassten Kapitel und eine Antwort auf die Fragestellung, inwiefern Progressive Web Apps zum jetzigen Zeitpunkt in der Lage sind, native Apps zu ersetzen.

2 Theoretische Grundlagen mobiler Applikationen

In diesem Kapitel werden die Begriffe Progressive Web App, Native App und Hybrid App definiert sowie die Merkmale und Charakteristiken dieser verschiedenen App-Typen erläutert. Zudem werden die mobilen Plattformen Android und iOS vorgestellt. Anschließend werden die für die Untersuchung und den Vergleich verwendeten Vergleichskategorien vorgestellt sowie dessen Auswahl und Relevanz erläutert.

2.1 Progressive Web Apps

Progressive Web Apps sind moderne Webapplikationen, die auf Basis von modernen Webtechnologien entwickelt werden (vgl. Osmani, 2015). Der Begriff Progressive Web Apps wurde erstmalig von dem Google Mitarbeiter Alex Russell erwähnt. In seinem Blog-Artikel werden Progressive Web Apps als eine neue Art von Webapplikation beschrieben (vgl. Russell, 2015). Bei Progressive Web Apps handelt es sich um eine Sammlung von Webtechnologien, Strategien und APIs die es ermöglichen, eine moderne und app-ähnliche Webapplikation zu entwickeln (vgl. Sheppard, 2017 S. 6). Entwickler sind dadurch in der Lage, Progressive Web Apps zu entwickeln, die über eine ähnliche User Experience verfügen, wie native Apps (vgl. Springer, 2016 S. 120 ff.).

Progressive Web Apps verfügen über Merkmale und Charakteristiken, die bisher nur nativen Apps vorbehalten waren. Dabei werden die positiven Eigenschaften und Charakteristiken von Webapplikationen und nativen Apps miteinander kombiniert (vgl. Braun, 2017 S. 104). Sie verfügen über ein responsives Design und passen sich so an unterschiedliche Geräte, wie PCs, Smartphones und Tablet-PCs an. Zudem basieren Progressive Web Apps auf dem Konzept des Progressive Enhancement und können dadurch auf allen aktuellen Browsern ausgeführt und genutzt werden (vgl. Osmani, 2015). Um jedoch alle Vorteile und Funktionen einer Progressive Web App nutzen zu können, wird vorausgesetzt, dass die Technologien und Konzepte von Progressive Web Apps von dem verwendeten Browser unterstützt werden (vgl. Steyer, 2017a S. 5 f.). Des Weiteren lassen sie sich auf dem Homescreen eines mobilen Gerätes installieren (vgl. Osmani, 2015). Durch das sogenannte Web App Manifest kann das Verhalten und die Darstellung, der auf dem Homescreen installierten App, festgelegt werden (vgl. Springer, 2016 S. 123). Ferner können Progressive Web Apps unabhängig von der Internetverbindung verwendet werden und Push Notifications empfangen (vgl. Osmani, 2015). Die Application-Shell-Architektur von Progressive Web Apps sorgt für eine schnelle und zuverlässige Darstellung der Benutzeroberfläche (vgl. Google Developer Training, o.J.). Durch das Ablegen von statischen Komponenten der Benutzeroberfläche in den Cache, wird ein schnelles, von der Internetverbindung unabhängiges Laden und Darstellen der Benutzeroberfläche, ermöglicht (vgl. Steyer, 2017a S. 6). Der Funktionsumfang von Progressive Web Apps ist begrenzt, da sie nicht uneingeschränkt auf die gerätespezifische Hardware und Funktionen zugreifen können (vgl. Vaadin, o.J.). Es kann lediglich nur auf Hardware und Funktionen zugegriffen werden, die von HTML5- und JavaScript-APIs unterstützt werden

(vgl. Reshetilo et al., 2017). Progressive Web Apps werden, anders als native- und hybride Apps, direkt über den Browser ausgeführt und sind dadurch plattformunabhängig (vgl. Osmani, 2015).

Auf Basis der Definition, der Merkmale und Charakteristiken, lassen sich Progressive Web Apps von klassischen Webapplikationen, hybriden- und nativen Apps abgrenzen.

2.2 Native Apps

Native Apps sind Anwendungen, die speziell für mobile Plattformen, wie bspw. Android oder iOS entwickelt werden. Native Apps werden direkt auf dem mobilen Gerät des Benutzers installiert und ausgeführt (vgl. Tosic, 2015 S. 9). Die Entwicklung einer nativen App für eines der mobilen Plattformen (siehe Kapitel 2.4), ist nur mit einer der plattformspezifischen Programmiersprachen möglich. Das bedeutet, dass Apps für iOS nur mit den Programmiersprachen Swift oder Objective-C und für Android nur mit der Programmiersprache Java entwickelt werden können (vgl. ebd.). Aus diesem Grund müssen native Apps für jedes Betriebssystem speziell neu entwickelt werden (vgl. Vollmer, 2017 S. 16). Dadurch sind native Apps für die jeweilige spezifische Plattform optimiert (vgl. Steyer, 2017b S. 30). Für die Entwicklung einer nativen App, für eines dieser Plattformen, benötigt der Entwickler, neben der dafür vorgesehenen Programmiersprache, auch eine integrierte Entwicklungsumgebung (IDE) sowie das zugehörige Software Development Kit (SDK) der jeweiligen Plattform (vgl. Vollmer, 2017 S. 16).

Durch die plattformspezifische Entwicklung können native Apps auf eine Vielzahl an Funktionen und Möglichkeiten, die von den mobilen Geräten und dem Betriebssystem zur Verfügung gestellt werden, zurückgreifen. So bietet bspw. das SDK, die Möglichkeit, direkt auf spezielle Geräteeigenschaften und Funktionen zuzugreifen (vgl. Steyer, 2017b S. 30). Dazu gehören u. a. der Zugriff auf verschiedene Hardwarekomponenten sowie auf spezifische Sensoren und Aktoren, die von den jeweiligen mobilen Geräten zur Verfügung gestellt werden. Zudem sind native Apps aufgrund der Optimierung für die jeweiligen Plattformen ressourcenschonend, performant und passen sich vor allem auch an die Geräteeigenschaften an. Dadurch wird dem Benutzer eine gute User Experience und Usability ermöglicht (vgl. Vollmer, 2017 S. 16 f.).

Native Apps werden nach Fertigstellung in den für sie vorgesehenen plattformspezifischen App Stores zum Herunterladen und Installieren zur Verfügung gestellt. Auf den mobilen

Geräten werden native Apps dann durch die Laufzeitumgebung des jeweiligen mobilen Betriebssystems ausgeführt (vgl. Vollmer, 2017 S. 16 f.).

2.3 Hybrid Apps

Hybrid Apps sind eine Kombination aus nativer App und Webapplikation (vgl. Schickler et al., 2015 S. 19). Bei hybriden Apps handelt es sich ebenfalls um Apps für mobile Geräte, die jedoch anders als native Apps, auf Webtechnologien basieren (vgl. Vollmer, 2017 S. 19). Die Entwicklung der Benutzeroberfläche und der Anwendungslogik erfolgt demnach mithilfe von aktuellen Webtechnologien, wie bspw. HTML5, CSS3 und JavaScript. Die daraus resultierende Webapplikation, wird dann in einem sogenannten nativen Container ausgeführt. Dieser Container stellt anschließend auf allen Plattformen eine Web-View dar, die HTML-Inhalte darstellen kann (vgl. Schickler et al., 2015 S. 19). Dadurch sind hybride Apps plattformunabhängig und somit auf verschiedenen Betriebssystemen ausführbar. Mittels Plug-ins kann auf die gerätespezifischen Funktionen und Komponenten, wie bspw. Kamera, GPS oder die Kontakte, eines mobilen Gerätes zugegriffen werden (vgl. Vollmer, 2017 S. 19). Dazu werden von verschiedenen Cross-Plattform-Frameworks diverse native Bibliotheken angeboten, die in die Anwendung eingebunden werden können. Anders als bei nativen Apps, ist der Hardware-Zugriff nur eingeschränkt möglich und hängt von dem verwendeten Cross-Plattform-Framework ab (vgl. Schickler et al., 2015 S. 19). Hybride Apps können, wie native Apps, über einen plattformspezifischen App Store zum Herunterladen und Installieren, angeboten werden (vgl. Vollmer, 2017 S. 19).

Die nachfolgende Abbildung 1 stellt die Unterschiede der zuvor vorgestellten App-Typen grafisch dar.

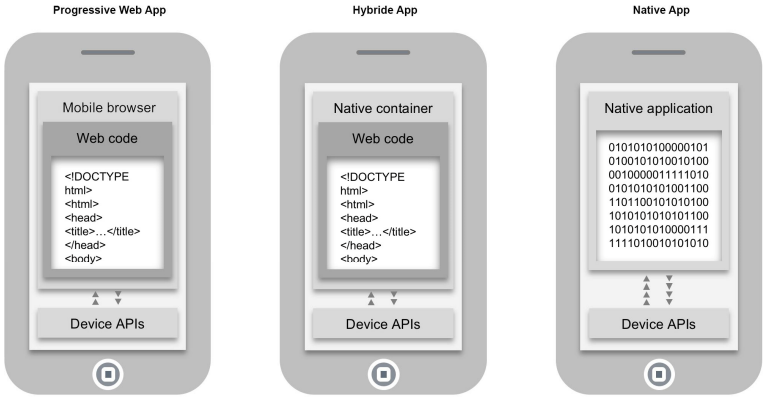

Abbildung 1 - Unterschiedliche Ansätze zur Entwicklung mobiler Applikationen - Quelle: In Anlehnung an Kumar, 2018

2.4 Mobile Plattformen

2.4.1 Android

Android ist ein Betriebssystem für mobile Geräte, wie Smartphones und Tablet-PCs und wurde von Google eingeführt. Android wird von vielen Geräteherstellern als Betriebssystem für ihre Geräte verwendet. Mittlerweile gibt es auch andere Einsatzbereiche des mobilen Betriebssystems, wie Android Wear, Android Auto und Android TV (vgl. Künneth, 2018 S. 19 ff.). Android ist ein auf dem Linux-Kernel basierendes Betriebssystem (vgl. Google Inc., o.J.j). Die Entwicklung von Android erfolgt durch das von Google gegründete Open Handset Alliance (OHA) Konsortium (vgl. Künneth, 2018 S. 20), dass sich mittlerweile aus 84 Unternehmen, darunter Softwareunternehmen und Geräteherstellern sowie Netzbetreiben zusammensetzt (vgl. Open Handset Alliance, o.J.). Das Konsortium verfolgt das Ziel, mit Android eine offene Plattform für mobile Geräte zu schaffen (vgl. Tosic, 2015 S. 5). Android wurde im Rahmen des Android Open Source Project (AOSP) von Grund auf als freies und quelloffenes Betriebssystem entwickelt (vgl. Google Inc., o.J.g). Dadurch wird die Möglichkeit geboten, die Software zu erweitern, zu modifizieren und auf verschiedenen Geräten anzubieten (vgl. Google Inc., o.J.c). Android ist eine Open-Source-Software, die größtenteils unter der Apache Software License 2.0 steht (vgl. Google Inc., o.J.e). Die aktuelle Versionsnummer von Android ist 9 (API-Level 28) und trägt den Codenamen „Pie“ (vgl. Google Inc., o.J.d). Um den Entwicklern die Möglichkeit zu bieten, Apps für Android zu entwickeln, wurde das Android SDK zur Verfügung gestellt (vgl. Künneth, 2018 S. 19).

Die Architektur des Android Betriebssystems setzt sich aus mehreren Schichten zusammen (siehe Abbildung 2). Dazu gehört (von unten nach oben betrachtet) der Linux-Kernel, die Hardwareabstraktionsschicht (Hardware Abstraction Layer, HAL), die Android-Laufzeitumgebung (Android Runtime, ART), die nativen C/C++ Bibliotheken (Libraries), das Java API Framework sowie die System-Apps. Android basiert auf dem Linux-Kernel, dieser ist für die Speicher- und Prozessverwaltung zuständig. Zudem werden Treiber und das Power-Management durch den Linux-Kernel verwaltet (vgl. Google Inc., o.J.j).

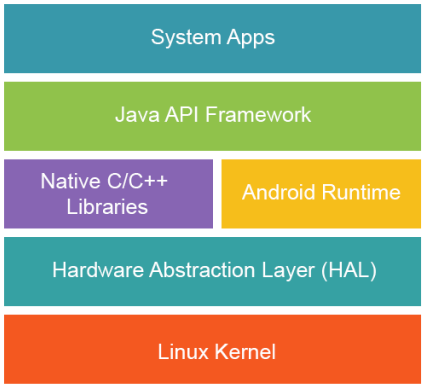

Abbildung 2 - Android Architektur - Quelle: In Anlehnung an Google Inc., o.J.j

Um den Zugriff auf Gerätekomponenten zu ermöglichen, stellt die Hardwareabstraktionsschicht dem Java API Framework, Standard-Interfaces zur Verfügung. Die Hardwareabstraktionsschicht besteht aus mehreren Modulen, die ein Interface für die Hardware-Komponenten, wie bspw. die Kamera, implementieren. Beim Zugriff auf die Hardware-Komponenten über das Java API Framework, wird das entsprechende Modul der Hardwareabstraktionsschicht von der Android-Laufzeitumgebung geladen. Alle nativen Android Apps werden in einem eigenen Prozess und einer eigenen Instanz der Android-Laufzeitumgebung ausgeführt. Einige Komponenten, wie bspw. die Android-Laufzeitumgebung und die Hardwareabstraktionsschicht, benötigen native Bibliotheken, die in C und C++ geschrieben sind. Über das Java API Framework werden von der Android-Plattform, APIs für den Zugriff auf einige native Bibliotheken zur Verfügung gestellt, die in den Apps verwendet werden können (vgl. Google Inc., o.J.j).

Das Java API Framework stellt den Entwicklern den gesamten Funktionsumfang des Android Betriebssystems über Java APIs zur Verfügung, die zur Entwicklung einer nativen Android App benötigt werden. Es beinhaltet u. a. ein View System, zur Erstellung von Benutzeroberflächen, einen Content Provider, für den Zugriff auf die Daten anderer Apps sowie einen Notification Manager, der den Zugriff auf die Statusleiste zur Darstellung von Benachrichtigungen, ermöglicht (vgl. ebd.).

Android stellt viele verschiedene vorinstallierte System-Apps, wie bspw. E-Mail, Kalender, Kontakte sowie einen Internet Browser zur Verfügung (vgl. ebd.).

2.4.2 iOS

iOS ist ein von Apple entwickeltes mobiles Betriebssystem und wird für die eigenen mobilen Geräte, wie bspw. iPhone, iPad und iPod, entwickelt (vgl. Steyer, 2018 S. 26). Im Jahr 2007 hatte Apple das erste iPhone und somit auch das erste Smartphone vorgestellt (vgl. Apple Inc., 2007). Zeitgleich mit dem iPhone wurde die erste Version von iOS veröffentlicht (vgl. Sillmann, 2017 S. 1). Das mobile Betriebssystem iOS stammt von Apples Desktop-Betriebssystem macOS ab und basiert auf einem Unix-Kernel (vgl. Steyer, 2018 S. 26). Anders als andere mobile Betriebssysteme, kann iOS nur auf Apple-Geräten installiert und verwendet werden. Um dem Benutzer eine gute Performance und User Experience bieten zu können, ist iOS speziell auf die Hardware dieser Geräte abgestimmt (vgl. Apple Inc., o.J.f). Aktuell ist iOS in der Version 12 erhältlich (vgl. Apple Inc., o.J.e). Für iOS lassen sich Apps mit den

Programmiersprachen Objective-C und Swift entwickeln (vgl. Apple Inc., o.J.i). Zur Entwicklung von Apps stellt Apple die integrierte Entwicklungsumgebung Xcode sowie das iOS Software Development Kit zur Verfügung (vgl. Apple Inc., o.J.m).

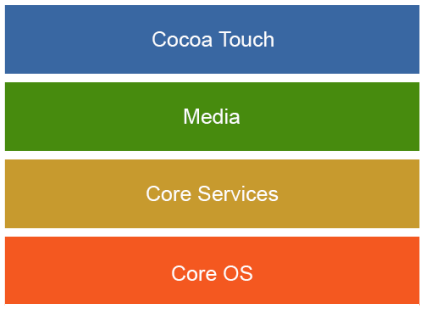

Abbildung 3 - iOS Architektur - Quelle: In Anlehnung an Apple Inc., o.J.b

Die Architektur von iOS setzt sich aus vier Schichten (Layer) zusammen (siehe Abbildung 3). Dazu gehört (von unten nach oben betrachtet) der Core OS-Layer, Core Services-Layer, Core Media-Layer sowie der Cocoa Touch-Layer. Der Core OS-Layer stellt die grundlegenden Funktionen und Frameworks zur Verfügung. Hierbei werden Funktionen für die Speicherverwaltung, den Zugriff auf das Dateisystem, die Sicherheit und für die Netzwerkkommunikation zur Verfügung gestellt (vgl. Sillmann, 2017 S. 8). Der Core Services-Layer baut auf dem Core OS-Layer auf und stellt bspw. Dienste für den Aufbau einer Bluetooth-Verbindungen zwischen zwei iOS-Geräten zur Verfügung. Diese Schicht bietet u. a. auch viele verschiedene Frameworks zur Nutzung von iCloud oder Time Machine an. Der Media-Layer bietet verschiedene Audio-, Video-, Animationsfunktionen, Grafikfunktionen und Frameworks an, die verwendet werden können (vgl. Apple Inc., o.J.b). Die oberste Schicht, der Cocoa Touch-Layer, bietet alle erforderlichen Funktionen und Frameworks, die zur Entwicklung von Apps benötigt werden. Diese Schicht enthält u. a. das UIKit-Framework, welches die Entwicklung von Benutzeroberflächen einer App, ermöglicht (vgl. Sillmann, 2017 S. 8).

2.5 Auswahl der Vergleichskategorien

Um native Android Apps und Progressive Web Apps untersuchen und einheitlich vergleichbar machen zu können, werden Vergleichskategorien benötigt. Die Vergleichskategorien sollen einerseits auf die Antwort der Fragestellung dieser Arbeit hinführen und andererseits den begrenzten Umfang berücksichtigen. Aufgrund dessen wird in diesem Kapitel die Auswahl und die Relevanz der Vergleichskategorien, die für die Untersuchung und den Vergleich verwendet werden, begründet. In Anbetracht der genannten Bedingungen, wurden die folgenden fünf Vergleichskategorien ausgewählt:

- **Entwicklung:** Zur Entwicklung von Apps für die verschiedenen mobilen Plattformen, werden i. d. R. unterschiedliche Programmiersprachen, Entwicklungsumgebungen und Tools verwendet. Um die Unterschiede und Möglichkeiten im Hinblick auf die Entwicklung von nativen Android Apps und Progressive Web Apps darstellen zu können, ist es bedeutsam, die beiden App-Typen anhand dieser Vergleichskategorie zu untersuchen und zu vergleichen.
- **Technische und funktionale Möglichkeiten:** Diese Vergleichskategorie wurde ausgewählt und ist relevant, um die verschiedenen technischen und funktionalen Möglichkeiten (Offline-Nutzbarkeit, Benachrichtigungen, Geräte-Zugriff etc.) von nativen Android Apps und Progressive Web Apps zu untersuchen und zu vergleichen. Mithilfe dieser Vergleichskategorie ist es möglich, Unterschiede und Einschränkungen der beiden App-Typen aufzuzeigen.
- **Plattformunterstützung:** Die Plattformunterstützung der App-Typen kann je nach Browser und Betriebssystem unterschiedlich und eingeschränkt sein. Zur Verdeutlichung der Unterschiede und Einschränkungen, ist die Vergleichskategorie für die Untersuchung und den Vergleich der Plattformunterstützung von nativen Android Apps und Progressive Web Apps von Bedeutung.
- **Look and Feel:** Das Look and Feel von Apps kann je nach Plattform unterschiedlich sein. Zur Realisierung eines nativen Look and Feels, werden für die verschiedenen Plattformen unterschiedliche Möglichkeiten und Style-Guidelines zur Gestaltung von Benutzeroberflächen zur Verfügung gestellt. Aufgrund dessen ist es relevant, native Android Apps und Progressive Web Apps im Hinblick auf die Unterschiede, Möglichkeiten und Style-Guidelines zur Gestaltung von Benutzeroberflächen zu untersuchen und zu vergleichen.
- **Veröffentlichung und Verbreitung:** Zur Veröffentlichung und Verbreitung von Apps werden verschiedene Möglichkeiten geboten. Auch können sich hierbei die Abläufe voneinander unterscheiden. Daher ist diese Vergleichskategorie geeignet, um native Android Apps und Progressive Web Apps im Hinblick auf die verschiedenen Möglichkeiten und Abläufe zu untersuchen und zu vergleichen.

Nachdem die Auswahl und die Relevanz der Vergleichskategorien erläutert wurde, werden in den nachfolgenden Kapiteln, native Android Apps und Progressive Web Apps anhand dieser ausgewählten Vergleichskategorien näher untersucht und verglichen.

3 Untersuchung von nativen Android Apps

In diesem Kapitel wird der App-Typ Native Android App auf Basis der in Kapitel 2.5 vorgestellten Vergleichskategorien im Rahmen einer Literaturrecherche untersucht, vorgestellt und kurz erläutert. Zudem wird in Kapitel 3.4.2 das Look and Feel von iOS betrachtet, da sich die Gestaltung und das Layout der Benutzeroberfläche von Android unterscheidet.

3.1 Entwicklung

3.1.1 Java

Java ist eine ursprünglich von Sun Microsystems entwickelte objektorientierte Programmiersprache und bietet dem Entwickler die Möglichkeit, wiederverwendbaren Code- und Softwarekomponenten zu programmieren (vgl. Ullenboom, 2018 S. 49). Die aktuelle Version ist Java 10 (vgl. Oracle Corp., o.J.). Bei Java handelt es sich um eine plattform- und betriebssystemunabhängige Programmiersprache, die nicht an bestimmte Prozessoren oder Architekturen gebunden ist (vgl. Ullenboom, 2018 S. 51). Dementsprechend können in Java programmierte Programme auf unterschiedlichen Betriebssystemen, wie bspw. Windows, Linux und macOS ausgeführt werden. Java ist kostenlos und zeichnet sich vor allem durch eine einfache Syntax aus. Zudem steht Java als Open-Source-Code zu Verfügung und kann somit für neue Plattformen genutzt und portiert werden, wie es bei dem Betriebssystem Android der Fall ist. Mit Java lassen sich verschiedene Arten von Programmen entwickeln. So ist es möglich, Konsolenprogramme, Windows-Programme mit Benutzeroberfläche, Webanwendungen und auch native Android Apps zu entwickeln (vgl. Kofler, 2018 S. 24).

Java ist in verschiedene Editionen aufgeteilt, die für unterschiedliche Einsatzzwecke vorgesehen sind. Aus diesem Grund gibt es bspw. für Desktopanwendungen, die Java Standard Edition (Java SE) und zur Entwicklung komplexer mehrschichtiger Programme und Webanwendungen, die Java Enterprise Edition (Java EE). Zur Entwicklung von Programmen wird das kostenlose Java Development Kit (JDK) zur Verfügung gestellt. Das JDK enthält u. a. Klassenbibliotheken sowie Programme und Tools, die zur Entwicklung und zum Ausführen

von Programmen benötiget werden. Ebenfalls ist im JDK die Java-Laufzeitumgebung (Java Runtime Environment, JRE) enthalten, die zur Ausführung der Programme erforderlich ist (vgl. Abts, 2018 S. 2).

Die Entwicklung von nativen Android Apps unterscheidet sich von herkömmlichen Java-Programmen. Zur Entwicklung von nativen Android Apps wird das Android Software Development Kit (SDK) zur Verfügung gestellt und verwendet. Dieses bietet, anders als das JDK, spezielle Bibliotheken und Tools zur Entwicklung von Apps für das Android-Betriebssystem (vgl. Künneth, 2018 S. 19). Darüber hinaus werden, wie in Kapitel 2.4.1 erläutert, native Android Apps in der zur Verfügung stehenden Android Runtime des Betriebssystems (vgl. ebd. S. 26 f.) und nicht in der originalen Java-Laufzeitumgebung ausgeführt (vgl. Kofler, 2018 S. 26).

3.1.2 Android Studio

Android Studio ist die offizielle von Google entwickelte und freie IDE zur Entwicklung von nativen Android Apps. Android Studio basiert auf dem Code-Editor und den Entwicklertools von IntelliJ IDEA. Die IDE bietet dem Entwickler eine große Vielfalt an Features, die die Entwicklung von nativen Android Apps ermöglichen und vereinfachen. Des Weiteren ist auch das zur Entwicklung von nativen Android Apps erforderliche Android SDK in Android Studio enthalten (vgl. Google Inc., o.J.i). Android Studio ist aktuell in der Version 3.1.3 erhältlich (vgl. Google Inc., o.J.f). Die Benutzeroberfläche von Android Studio bietet dem Entwickler einen guten Überblick über das Android-Projekt sowie dessen Struktur. Dabei stellt Android Studio dem Entwickler einen umfangreichen Code-Editor mit Funktionen, wie bspw. Syntax-Highlighting, automatischer Code-Vervollständigung und Formatierung des Quellcodes zur Verfügung (vgl. Google Inc., o.J.i).

Zum Testen von Apps wird dem Entwickler ein Emulator zur Verfügung gestellt. Dieser ermöglicht es, Apps auf einem emulierten Gerät zu testen. Dabei können viele verschiedene vordefinierte Testgeräte mit unterschiedlichen Eigenschaften und Android-Versionen ausgewählt und emuliert werden. Ferner bietet Android Studio dem Entwickler Tools zum Debuggen sowie zur Performanceverbesserung des Quellcodes an, um die App ggf. zu optimieren (vgl. Google Inc., o.J.n).

Zur Erzeugung einer fertigen und installierbaren App, ist in Android Studio das Build-Tool Gradle integriert. Das Build-Tool Gradle erstellt automatisch aus dem Quellcode eine ausführbare App. Hierbei wird der Quellcode kompiliert, verknüpft und in eine ausführbare Datei verpackt. Native Android Apps werden in APK-Dateien (Android Package) verpackt und sind danach auf dem Betriebssystem Android ausführbar. Das Build-Tool Gradle bietet u. a. die Möglichkeit, mehrere APKs mit verschiedenen Funktionen einer nativen Android App zu erstellen. So ist es bspw. möglich, APKs für verschiedene Geräte, mit unterschiedlichen Bildschirmauflösungen zu erstellen (vgl. Google Inc., o.J.i).

3.2 Technische und funktionale Möglichkeiten

3.2.1 Offline-Nutzbarkeit und Datenhaltung

Native Android Apps können unabhängig von der Internetverbindung ausgeführt und verwendet werden. Bei der Installation einer nativen Android App werden alle für die Ausführung und Nutzung relevanten Dateien und Daten auf dem mobilen Gerät gespeichert. Dazu zählen bspw. das Layout, Grafiken und der kompilierte Quellcode, die das Grundgerüst einer nativen Android App bilden (vgl. Staudemeyer, 2018 S. 56). Inhalte und Daten, die i. d. R. von einem Server abgefragt werden, werden ebenfalls auf dem mobilen Gerät des Benutzers gespeichert, um eine native Android App unabhängig von der Internetverbindung nutzbar zu machen. Dies lässt sich mit verschiedenen Technologien, wie den SharedPreferences, SQLite und dem Android Sync Adapter Framework umsetzen. Zur lokalen Speicherung von Daten auf dem mobilen Gerät des Benutzers, stellt Android die SharedPreferences API und eine SQLite API zur Verfügung (vgl. Android Developers, o.J.j). Die SharedPreferences API kann zur Speicherung von kleinen Mengen an primitiven Daten in Form von Schlüssel-Wert-Paaren (key-value pairs) verwendet werden. Zu den primitiven Datentypen gehören boolean, int, long, float und string (vgl. Post, 2018 S. 218). Ein *SharedPreferences*-Objekt referenziert auf eine Datei mit Schlüssel-Wert-Paaren und bietet zudem Methoden zum Lesen und Schreiben an. Die SharedPreferences werden in einer XML-Datei auf dem mobilen Gerät gespeichert und durch das Android Betriebssystem verwaltet. Durch die Speicherung der Daten in einer XML-Datei, bleiben diese auch nach Ablauf der Session oder auch wenn die App geschlossen wurde oder abgestürzt ist, bestehen (vgl. Android Developers, o.J.m).

Eine weitere Möglichkeit zur lokalen Speicherung von Daten auf dem mobilen Gerät bietet das Open Source Datenbanksystem SQLite (vgl. sqlite.org, o.J.). SQLite ist ein relationales

Datenbanksystem, dass in das Android Betriebssystem integriert ist. Bei SQLite handelt es sich um eine sogenannte embedded Datenbank, die der jeweiligen App zugeordnet ist. Dadurch, dass nur eine App auf die Datenbank zugreifen kann, ist keine Benutzer- und Rechteverwaltung vorhanden (vgl. Richter, 2018 S. 113). Zudem bietet SQLite die Funktionen eines relationalen Datenbanksystems und speichert die Daten in einer Datei ab. Zu den gebotenen Funktionen zählen bspw. Transaktionen, Subselects und Views (vgl. sqlite.org, o.J.). Um eine Datenbank zu erstellen oder zu verwenden, stellt Android die SQLite API zur Verfügung (vgl. Android Developers, o.J.m).

Damit eine App dem Benutzer aktuelle Daten und Informationen anzeigen kann, müssen oft Daten von einem Server abgefragt werden. Native Android Apps können auch neue Daten erzeugen oder erfassen, wie bspw. Formular-Daten oder verfasste Artikel, die zur weiteren Verarbeitung oder Veröffentlichung an einen Server gesendet werden sollen. Um dem Benutzer eine von der Internetverbindung unabhängige Übertragung der Daten zu ermöglichen, stellt Android das Sync Adapter Framework zur Verfügung. Das Android Sync Adapter Framework ermöglicht die Synchronisation von Daten zwischen einer nativen Android App und einem Server (siehe Abbildung 4). Zudem bietet das Framework die Möglichkeit, die Übertragung von Daten zu verwalten und zu automatisieren. Für die automatische Übertragung und Synchronisation mit dem Server können bestimmte Kriterien festlegt werden. So kann u. a. festgelegt werden, dass nur Daten, die verändert wurden, synchronisiert werden sollen. Außerdem bietet das Framework die Möglichkeit, den Netzwerkstatus zu überprüfen um somit die Daten nur bei vorhandener Netzwerk- bzw. Internetverbindung an einen Server zu übertragen (vgl. Android Developers, o.J.ae). Das Framework ermöglicht so die Datenübertragung und Offline-Nutzbarkeit einer nativen Android App und stellt sicher, sobald wieder eine

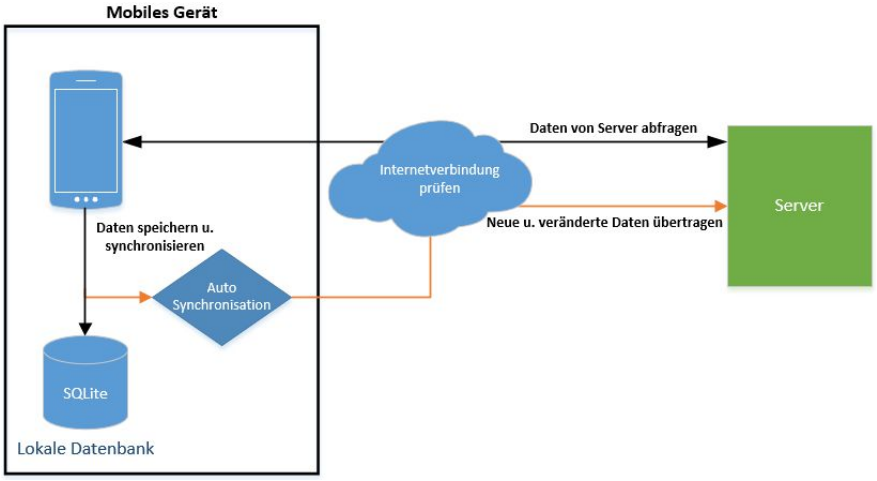

Abbildung 4 - Offline-Nutzbarkeit u. Synchronisation von Daten einer nativen Android App - Quelle: Eigene Darstellung

stabile Internetverbindung vorhanden ist, dass die Daten sowohl auf dem Server, als auch in der App immer auf dem aktuellsten Stand sind (siehe Abbildung 4).

3.2.2 Installierbarkeit

Native Android Apps müssen zuerst auf dem mobilen Gerät installiert werden, damit sie verwendet werden können. Die Installation erfolgt i. d. R. über einen App Store (z. B. Google Play Store). Alternativ kann eine native Android App auch direkt durch das Ausführen einer APK-Datei auf dem mobilen Gerät installiert werden. Bei der APK-Datei handelt es sich um ein Archiv, dass alle für die Installation und Nutzung der nativen Android App erforderlichen Dateien enthält. Dazu gehört der kompilierte Quellcode, Ressourcen (wie z. B. Grafiken, Icons oder Sounds) sowie die für die Nutzung und Installation erforderlichen Metadaten der App (vgl. Staudemeyer, 2018 S. 56).

Die Metadaten einer nativen Android App werden in der *AndroidManifest.xml*-Datei definiert, die ebenfalls in der APK-Datei enthalten ist. Über die XML-Datei werden verschiedene notwendige Informationen über die native Android App hinterlegt. Dazu gehört u. a. der Name der App sowie die zu verwendenden Icons, die auch auf dem Homescreen des Gerätes angezeigt werden. Darüber hinaus werden in der *AndroidManifest*-Datei, alle verwendeten Komponenten, wie Activities, Services, Broadcast Receivers und Content Provider, definiert (vgl. Android Developers, o.J.c). Des Weiteren kann der Entwickler die erforderlichen Zugriffsberechtigungen für die Nutzung bestimmter Funktionen und Systemkomponenten definieren, die zur Laufzeit der App benötigt werden (siehe Kapitel 3.2.5). Ferner wird in der *AndroidManifest*-Datei festgelegt, welche Android Version mindestens erforderlich ist, um die native Android App installieren und ausführen zu können (vgl. Jackson, 2013 S. 36). Damit soll sichergestellt werden, dass alle verwendeten APIs auch tatsächlich von den Android Versionen unterstützt und angeboten werden (vgl. Android Developers, o.J.c). Die APK-Datei kann mithilfe von Android Studio und dem integrierten Build-Tool Gradle erstellt und anschließend zur Installation verwendet werden (siehe Kapitel 3.1.2) (vgl. Google Inc., o.J.i). Nach erfolgreicher Installation erscheint die native Android App auf dem Homescreen (siehe Abbildung 5) und kann über die Einstellungen wieder deinstalliert werden.

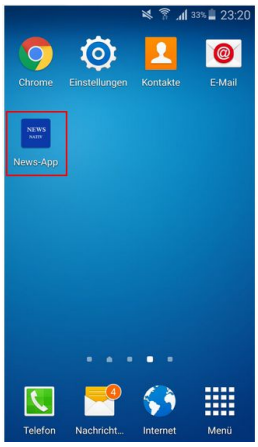

Abbildung 5 - Installierte native Android App - Quelle: Eigene Darstellung

3.2.3 Benachrichtigungen

Bei Benachrichtigungen handelt es sich um Push Notifications, die an das mobile Gerät eines Benutzers gesendet werden können. Push Notifications werden dazu verwendet, den Benutzer über wichtige Ereignisse oder Neuigkeiten zu informieren. Android unterstützt den Empfang und die Darstellung von Push Notifications auf mobilen Geräten (vgl. Android Developers, o.J.af). Dazu muss die native Android App nicht im Vordergrund stehen oder geöffnet sein. Push Notifications lenken die Aufmerksamkeit auf die App, unterbrechen den Benutzer aber nicht bei der Durchführung seiner Arbeit. Damit der Benutzer über den Eingang einer neuen Push Notification informiert wird, werden Push Notifications kurz auf dem Display des Benutzers eingeblendet und anschließend in Form eines Icons in der Status-Bar des mobilen Gerätes angezeigt. Der Benutzer kann sich die eingegangenen Push Notifications jederzeit durch herunterziehen der Status-Bar anzeigen lassen und bei Betätigen einer Push Notification, die App direkt öffnen. Ebenfalls ist es möglich, den Benutzer durch ein akustisches Signal, z. B. einen Klingelton oder durch einen Vibrationsalarm, auf die eingegangene Push Notification aufmerksam zu machen. Einige mobile Geräte verfügen über eine integrierte LED, die dazu verwendet werden kann, den Benutzer bei neuen Push Notifications zu informieren, wenn sich das Gerät im Standby-Modus befindet. Die Push Notifications werden dem Benutzer dabei auch auf dem Sperrbildschirm angezeigt (vgl. Becker et al., 2015 S. 241 f.). Damit Push Notifications in nativen Android Apps implementiert und dargestellt werden können, stellt Android die *Notification*-Klasse und einen *NotificationManager* zur Verfügung. Die *Notification*-Klasse bietet u. a. verschiedene Methoden zum Festlegen von Titel, Betreff, Text sowie Icons einer Push Notification an, die zur Darstellung auf dem mobilen Gerät verwendet werden können. Zudem kann auch ein Vibrationsalarm, ein akustisches Signal, das Blinken der LED sowie verschiedene andere Parameter festgelegt werden (vgl. Android Developers, o.J.x).

Der *NotificationManager* informiert den Benutzer über eine neue Benachrichtigung und stellt die Push Notification auf dem mobilen Gerät in der Status-Bar, im Notification Drawer und auf dem Sperrbildschirm dar (vgl. Becker et al., 2015 S. 241 f.).

Für den Empfang von Push Notifications registriert sich die native Android App mit der Sender ID, einem API Key und der App ID bei dem zugehörigen Push-Service (z. B. Firebase Cloud Messaging). Der Push-Service vergibt einen eindeutigen Registration-Token und sendet diesen an die native Android App zurück. Dieser wird dann an den Server übermittelt

und in einer Datenbank für den Versand von Push Notifications an den Benutzer gespeichert. Zum Senden einer Push Notfication an das mobile Gerät eines Benutzers, wird die Nachricht und der Empfänger an den Push-Service übermittelt (siehe Abbildung 6). Der Push-Service ordnet den Empfänger anhand des Registration-Tokens zu. Anschließend wird die Push Notification vom Push-Service an den Empfänger gesendet, sofern dieser erreichbar ist. Bei Nichterreichbarkeit des Empfängers, bzw. des mobilen Geräts, verweilt die Push Notification in einer Warteschlange beim Push-Service solange, bis der Empfänger wieder erreichbar ist (siehe Abbildung 6) (vgl. McLemore et al., 2018). Bei erfolgreicher Übertragung an das mobile Gerät des Benutzers, wird die *onMessageReceived*-Methode des *FirebaseMessagingService* ausgeführt, ein *Notification*-Objekt mit allen Informationen und Parametern erzeugt und über den *NotificationManager* dem Benutzer auf dem mobilen Gerät dargestellt.

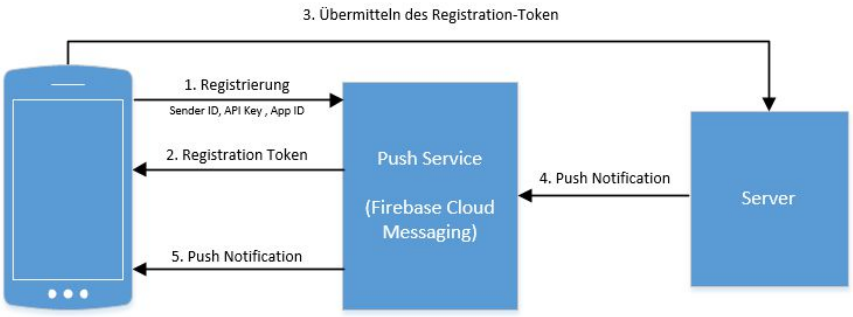

Abbildung 6 - Funktionsweise von Push Notifications unter Android - Quelle: Eigene Darstellung

3.2.4 Inter-App Kommunikation

Native Android Apps können Inhalte und Daten an andere Apps, die auf dem mobilen Gerät installiert sind, senden. Native Android Apps bestehen aus verschiedenen *Activities*, die über eine Benutzeroberfläche verfügen. Über die *Activities* und die dazugehörige Benutzeroberfläche kann der Benutzer bestimmte Aufgaben erledigen. Dazu gehört z. B. das Ausfüllen von Formularen, Anzeigen von Artikeln oder auch das Aufnehmen von Bildern. Der Benutzer und auch die Daten einer *Activity*, können an andere *Activities* und auch Apps weitergeleitet und gesendet werden. Dazu kommen sogenannte *Intents* zum Einsatz (vgl. Android Developers, o.J.o). *Intents* sind asynchrone Nachrichten, die den Austausch von Inhalten und Daten zwischen Android Komponenten, wie z. B. *Activities* ermöglichen. *Intents* können für den Austausch zwischen *Activities* innerhalb einer App sowie für den Austausch zwischen anderen auf dem Gerät installierten Apps verwendet werden (vgl. ebd.). Es wird hierbei zwi-

schen einem *expliziten* und *impliziten Intent* unterschieden. *Explizite Intents* rufen eine bestimmte *Activitiy* innerhalb derselben App auf und *implizite Intents* hingegen rufen ein *Activity* einer anderen App auf (vgl. Künneth, 2018 S. 131). Durch die Nutzung von *Intents* kann so die Inter-App Kommunikation realisiert werden und Daten an andere Apps gesendet oder von anderen Apps empfangen werden (vgl. Android Developers, o.J.o). Bei einem *impliziten Intent* wird dem Benutzer der native Auswahl-Dialog dargestellt (siehe Abbildung 7). Der Benutzer kann darüber auswählen, an welche App die Daten gesendet werden sollen. So kann mithilfe von *Intents* und dem nativen Auswahl-Dialog die Möglichkeit geboten werden, Inhalte und Daten an andere Apps zu senden und bspw. eine „Share/Teilen"-Schaltfläche und Funktion umgesetzt werden (vgl. Android Developers, o.J.aa).

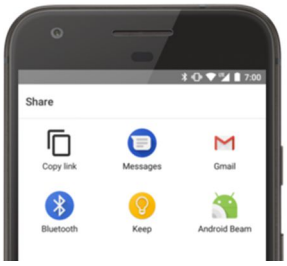

Abbildung 7 - Auswahl-Dialog bei Inter-App Kommunikation - Quelle: (Android Developers, o.J.aa)

3.2.5 Zugriffskontrolle

Native Android Apps benötigen für die Nutzung bestimmter Funktionen und Systemkomponenten, die Zugriffsberechtigung des Benutzers. In einer sogenannten Sandbox werden native Android Apps ausgeführt und können standardmäßig nur mit einer Berechtigung des Benutzers auf bestimmte Funktionen und Systemkomponenten zugreifen (vgl. Künneth, 2018 S. 158). Die Zugriffe werden durch das Android Betriebssystem verwaltet. Durch die Aufforderung zur Erteilung von Berechtigungen soll verhindert werden, dass ein unerlaubter Zugriff auf Funktionen und sensiblen Daten durch unerwünschte Apps ermöglicht wird. Dementsprechend sind verschiedene APIs und Funktionen vor unerlaubten Zugriffen geschützt. Zu den APIs und Funktionen, die eine Zugriffsberechtigung benötigen, gehören bspw. Kamera-, GPS-, Bluetooth-, Telefon- sowie SMS/MMS-Funktionen (vgl. Google Inc., o.J.a). Damit diese Funktionen und APIs von einer nativen Android App verwendet werden können, muss der Entwickler die benötigte Berechtigung für die Funktionen und APIs, in der *AndroidManifest.xml*-Datei, über das <*uses-permission*>-Element definieren (vgl. Künneth, 2018 S. 158 f.).

Bei der Installation der nativen Android App wird der Benutzer dann über die benötigten Zugriffsberechtigungen informiert. Dazu wird ein Dialog-Fenster (Pop-up) mit allen erforderlichen Berechtigungen eingeblendet, die die App für die Nutzung und die Installation benötigt (siehe Abbildung 8). Zu beachten ist, dass der Benutzer nicht individuell entscheiden kann, für welche Funktionen und Komponenten er die Zugriffsberechtigung erteilen möchte und für welche nicht. Damit eine App installiert und verwendet werden kann, muss die Erlaubnis für alle angefragten Zugriffsberechtigungen erteilt werden. Das nachträgliche wiederrufen einzelner Zugriffsberechtigungen ist somit nicht möglich. Alle erteilten Zugriffsberechtigungen bleiben solange gültig, bis die App wieder von dem Gerät deinstalliert wurde (vgl. Google Inc., o.J.a).

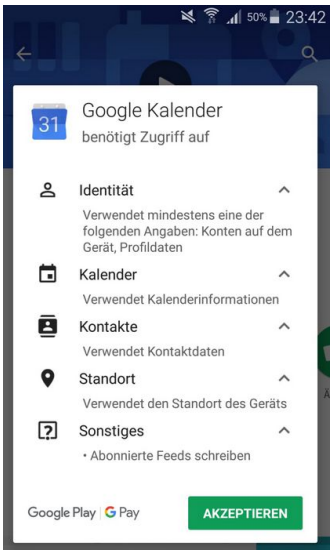

Abbildung 8 - Android Zugriffsberechtigung - Quelle: Eigene Darstellung

3.2.6 Multimedia

Native Android Apps können alle gängigen Audio- und Video-Medien abspielen. Möglich ist die Wiedergabe über das Android Multimedia Framework. Dieses Framework bietet u. a. die Möglichkeit, Audio- und Video-Medien direkt aus dem Ressourcen-Ordner der App (RAW-Verzeichnis) oder die im Dateisystem gespeicherten Medien des mobilen Gerätes wiederzugeben. Auch die Wiedergabe von Medien, die über einen Daten-Stream über das Netzwerk oder Internet gesendet werden, ist ebenfalls möglich (vgl. Android Developers, o.J.u). Die Wiedergabe und Steuerung von Medien kann mithilfe der *MediaPlayer*-Klasse ermöglicht werden. Hierzu bietet die *MediaPlayer*-Klasse verschiedene Methoden zur Steuerung von Medien an (vgl. Post, 2018 S. 187). Dazu gehören bspw. Methoden zum Starten, Stoppen, Pausieren, Neustarten der Medienwiedergabe, anpassen der Lautstärke sowie die Ausgabe von Informationen zum aktuell wiedergegebenen Medium (vgl. Android Developers, o.J.t).

Android unterstützt viele verschiedene Multimedia-Formate, die wiedergegeben werden können. Dazu gehören u. a. folgende Audio- und Video-Formate (vgl. Android Developers, o.J.ad):

- **Audio:** WAV, MP3, HE-AAC, AAC-LC, AMR, MIDI, FLAC, OPUS, Ogg Vorbis
- **Video:** MPEG-4, H263, H264, WebM

Über die Android Media Session API können die wiedergegebenen Audio- und Video-Medien auf dem Sperrbildschirm und in der Notification-Bar des Android-Gerätes dargestellt und gesteuert werden. In der Notification-Bar und auf dem Sperrbildschirm lassen sich mit der Media Session API und der *MediaMetadataCompat*-Klasse verschiedene Metadaten zum aktuell wiedergebenden Medium anzeigen. Zu den Metadaten, die angezeigt werden können, gehört bspw. Titel, Künstler, Album, Lied sowie die Dauer des abgespielten Liedes. Auch die Darstellung eines Cover-Bildes ist möglich. Ferner können Schaltflächen zur Steuerung der Wiedergabe (Pause, Play, Weiter, Zurück) angezeigt und von dem Benutzer betätigt werden (vgl. Android Developers, o.J.ag).

3.2.7 Geräte-Zugriff

Native Android Apps können auf verschiedene Geräte- und Software-Funktionen eines mobilen Gerätes zugreifen. Dazu stellt das Android Framework dem Entwickler verschiedene APIs und Klassen zur Verfügung. Diese APIs und Klassen bieten u. a. den Zugriff auf die im Gerät verbauten Hardware-Komponenten, wie bspw. Kamera, Mikrofon, GPS, Bluetooth, NFC sowie den Zugriff auf die verschiedenen Sensoren. Zudem werden auch APIs für den Zugriff auf das Dateisystem, die Kontakte, die Zwischenablage, Akku- und Netzwerkinformationen sowie auf den Kalender zur Verfügung gestellt.

Der Zugriff auf die Kameras wird mithilfe der Camera API ermöglicht. Die Camera API bietet dabei unterschiedliche Kamera-Funktionen an, die die Aufnahme von Bildern und Videos innerhalb einer nativen Android App ermöglichen. Zum Aufnehmen von Bildern bietet die Camera API verschiedene Funktionen, die über die zur Verfügung stehenden Methoden und Parameter definiert und verwendet werden können (vgl. Meier, 2018 S. 699 f.). So ist es möglich, für Bilder bestimmte Parameter, wie u. a. das Format, die Größe (height und width) sowie auch Einstellungen für das Blitzlicht oder zur Fokussierung der Kamera bei Bilderaufnahmen festzulegen (vgl. Android Developers, o.J.h). Die Aufzeichnung von Audio- und Video-Medien ist in Kombination mit der Camera API und der MediaRecorder API möglich. Die MediaRecorder API unterstützt die Aufzeichnung von Audio- und Video-Medien, eine Vielzahl an verschiedenen Audio- und Video-Formaten (siehe Kapitel 3.2.6) sowie den Zugriff auf das Mikrofon des mobilen Gerätes (vgl. Meier, 2018 S. 695 ff.). Ferner bietet die MediaRecorder API verschiedene Methoden, die das Festlegen des Formats, Dateinamen oder der Audio-/Video-Quelle (Kamera, Mikrofon) ermöglichen. Ebenso können

speziell für Video-Aufnahmen bestimmte Parameter, wie die Größe (height und width) und die Auflösung festgelegt werden (vgl. Android Developers, o.J.v).

Native Android Apps können auf das Dateisystem zugreifen und somit Dateien öffnen, lesen und auch auf dem Gerät speichern. Zur Speicherung von Dateien stehen verschiedene Speicherorte zur Verfügung. Android bietet die Möglichkeit, Dateien auf einem externen- oder internen Speicher abzulegen. Bei einem externen Speicher handelt es sich bspw. um eine SD-Karte. Der externe Speicher wird i. d. R. zur Speicherung von Benutzerdateien, wie z. B. für Bilder verwendet. Bei dem internen Speicher hingegen, handelt es sich um den internen Gerätspeicher, der zur Speicherung von App-Dateien verwendet wird. Die auf dem internen Gerätespeicher ablegten Dateien können nur von den zugehörigen Apps geöffnet und gelesen werden (vgl. Android Developers, o.J.m). Zur Speicherung von Dateien, auf dem zur Verfügung stehenden Speicher, bietet Android die File API an. Diese API bietet verschiedene Methoden zum Lesen und Schreiben von Dateien sowie zum Öffnen von Ordnern (vgl. Android Developers, o.J.z). Darüber hinaus bietet Android das Storage Access Framework (SAF), dass für den Zugriff auf das Dateisystem verwendet werden kann. Mit diesem Framework kann dem Benutzer ermöglicht werden, das Dateisystem nach Bildern, Dateien oder Dokumenten zu durchsuchen und diese zu öffnen (vgl. Becker et al., 2015 S. 501). Dazu stellt dieses Framework dem Benutzer einen Picker zur Verfügung, der eine einfache Benutzeroberfläche für die Auswahl der Dateien bietet (vgl. Android Developers, o.J.y).

Moderne mobile Geräte, wie Smartphones und Tablet-PCs, verfügen über ein integriertes GPS-Modul, dass zur Ermittlung des Standorts verwendet werden kann (vgl. Lee, 2012 S. 375). Es gibt unterschiedliche Methoden, die zur Ermittlung des Standortes verwendet werden können. Eine Methode zur Standortermittlung ist die Nutzung von GPS. Mittels GPS lässt sich der Standort des Benutzers sehr genau bestimmen, jedoch funktioniert dies nur außerhalb von Gebäuden. Eine weitere Methode ist die Ermittlung des Standortes anhand von Mobilfunksignalen und WLAN-Netzen (vgl. Bach, 2012 S. 411). Die Position eines mobilen Gerätes lässt sich anhand des Standortes eines Mobilfunkmastes und der Vermittlungsknoten eines Providers ermitteln. Diese Methode funktioniert, anders als GPS, auch innerhalb von Gebäuden (vgl. Lee, 2012 S. 375). Zur Ermittlung von Standortinformationen eines mobilen Gerätes, stellt Android die Location API und die *LocationManager*-Klasse zur Verfügung (vgl. Android Developers, o.J.q). Diese API und Klasse bietet die Möglich-

keit, die Standortinformationen anhand dieser genannten Methoden zu ermitteln (vgl. Künneth, 2018 S. 343 f.). Unter Android werden diese beiden Methoden zur Standortermittlung als *LocationProvider* (*GPS_Provider* und *Network_Provider*) bezeichnet (vgl. Bach, 2012 S. 410). Durch festlegen von Kriterien kann bestimmt werden, über welchen *LocationProvider* (GPS, Mobilfunk und WLAN) die Standortinformationen ermittelt werden sollen. Der *LocationProvider* stellt anschließend die Standortinformationen des mobilen Gerätes zur Verfügung (vgl. Künneth, 2018 S. 343 f.). Diese werden über die *getLastKnownLocation*-Methode einem *Location*-Objekt hinzugefügt (vgl. Android Developers, o.J.q). Das *Location*-Objekt enthält Standortinformationen, wie den Breiten- und Längengrad (*lati-* und *longitude*), die Geschwindigkeit (*speed*), Angaben zur Genauigkeit (*accuracy*) und der Höhe in Metern über Normalnull (*altitude*) (vgl. Android Developers, o.J.p). Anhand dieser Informationen, wird anschließend der genaue Standort des Benutzers ermittelt (vgl. Künneth, 2018 S. 343 f.).

Neben einem GPS-Modul sind in den meisten mobilen Geräten verschiedene Sensoren, wie bspw. ein Beschleunigungssensor, Gyroskop, Gravitationsmesser, Magnetometer oder Annäherungssensor integriert. Anhand von Bewegungen entlang und um die X-, Y- und Z-Achsen, ermöglichen die Sensoren u. a. das Erfassen von Beschleunigungen, Bewegungen, Drehungen und Neigungen eines mobilen Gerätes. Sie können z. B. dazu verwendet werden, die Ausrichtung des Gerätes zu ermitteln und dementsprechend die Darstellung der App auf dem Bildschirm anpassen. Der Annäherungssensor wird dazu genutzt, den Bildschirm des Gerätes auszuschalten, sobald der Benutzer das Gerät an den Kopf hält (vgl. ebd. S. 323). Auch können Helligkeits-, Temperatur- und Luftdruck-Sensoren verbaut sein. Für den Zugriff auf die Sensoren, wird das Android Sensor Framework zur Verfügung gestellt. Über die *SensorManager*-Klasse kann eine Instanz des Sensor Service erzeugt werden und auf die im Gerät verbauten Sensoren zugegriffen und dessen Werte und Daten ausgelesen werden (vgl. Android Developers, o.J.ac). Der *SensorManager* bietet zum Auslesen der Sensordaten verschiedene Methoden an. So kann z. B. die Gerätetemperatur, Umgebungshelligkeit sowie die Werte der X-,Y- und Z-Achsen der Bewegungssensoren ausgelesen und somit u. a. die Ausrichtung und Bewegung des Gerätes ermittelt werden (vgl. Android Developers, o.J.ab).

Des Weiteren bietet Android die Möglichkeit, Informationen über den Akku und die Netzwerkverbindung auszulesen und auf die im Gerät integrierte Vibration zuzugreifen und diese

zu verwenden. Informationen über den Akku eins mobilen Gerätes können mit der *BatteryManager*-Klasse ausgelesen werden. Diese Klasse enthält Methoden und Konstanten, die u. a. zum Auslesen des Ladestatus, der verbleibenden Akkukapazität und Restladezeit genutzt werden können (vgl. Android Developers, o.J.d). Für das Auslesen von Informationen über die Netzwerkverbindung, stellt Android die *ConnectivityManager*- und *NetworkInfo*-Klasse zur Verfügung. Mit diesen Klassen können bspw. Änderungen der Netzwerkverbindung festgestellt und Netzwerkinformationen ausgelesen werden. So kann ermittelt werden, ob das Gerät mit dem WLAN oder dem Mobilfunk-Netz verbunden ist, um damit bspw. den Verbrauch des mobilen Datenvolumens zu reduzieren (vgl. Android Developers, o.J.r). Der Zugriff und die Steuerung, der im Gerät verbauten Vibration, ist über die *Vibrator*-Klasse möglich. Diese Klasse stellt Methoden zur Steuerung der Vibrations-Komponente zur Verfügung und bietet die Möglichkeit, verschiedene Vibrationsmuster und Vibrationsintervalle zu definieren (vgl. Android Developers, o.J.ah).

Native Android Apps stellen auch APIs und Klassen für den Zugriff und die Nutzung von Bluetooth und NFC zur Verfügung. Die Zugriffmöglichkeiten auf diese beiden Komponenten werden im nachfolgenden Kapitel 3.2.8 näher vorgestellt.

Neben den bereits erwähnten Zugriffsmöglichkeiten auf die verschiedenen Hardware-Komponenten, bietet Android auch die Möglichkeit, auf die Zwischenablage, den Kalender und die Kontakte des mobilen Gerätes zuzugreifen. Für den Zugriff auf die Zwischenablage, stellt Android das Clipboard Framework und die *ClipboardManager*-Klasse zur Verfügung. Diese können zur Zwischenspeicherung von Texten, Daten und Bildern mittels „Copy and Paste“ verwendet werden. So ist es bspw. möglich, kopierte Inhalte innerhalb derselben oder in andere Apps einzufügen (vgl. Android Developers, o.J.l). Für den Zugriff auf den Kalender und die Kontakte eines mobilen Gerätes, stellt Android den Calendar- und Contacts Provider zur Verfügung. Der Calendar Provider verfügt über APIs, die das Abfragen, Erstellen, Ändern und Löschen von Kalendereinträgen, Ereignissen und Erinnerungen ermöglichen (vgl. Android Developers, o.J.g). Über die API des Contacts Provider, können bspw. die Kontaktdaten, der auf dem Gerät gespeicherten Kontakte, abgerufen und verwendet werden. Die APIs bieten neben dem Abfragen von Kontaktdaten auch die Möglichkeit, bestehende zu Ändern oder neue Kontakte hinzuzufügen (vgl. Android Developers, o.J.k).

3.2.8 Datenaustausch und Kommunikation

Android stellt unterschiedliche APIs für den Datenaustausch und die Kommunikation mit anderen Geräten zur Verfügung. Diese APIs bieten mobilen Geräten die Möglichkeit, mithilfe von verschiedenen Technologien, wie Bluetooth oder NFC, eine Verbindung mit anderen Geräten aufzubauen und mit diesen zu interagieren (vgl. Android Developers, o.J.j).

Die Bluetooth-Technologie wird von dem Android Betriebssystem unterstützt und ermöglicht Daten drahtlos mit anderen Bluetooth-Geräten auszutauschen (vgl. Künneth, 2018 S. 357). Das Android Application Framework bietet für den Zugriff auf die Bluetooth-Funktion des mobilen Gerätes, die Bluetooth API an (vgl. Android Developers, o.J.f). Neben der Übertragung von Daten, ist mit der Bluetooth API das Suchen nach anderen Bluetooth-Geräten in der Umgebung, das Abfragen des Bluetooth-Adapters nach gekoppelten Geräten sowie das Verwalten von mehreren Verbindungen möglich. Eine stromsparende Alternative ist die Bluetooth Low Energy-Technologie (BLE), die ebenfalls von Android unterstützt wird (vgl. Künneth, 2018 S. 373). Bluetooth Low Energy ist für eine stromsparende Übertragung von kleinen Datenmengen auf kurzer Distanz konzipiert. Die Bluetooth Low Energy-Technologie ermöglicht es, mit anderen auf dieser Technologie basierenden Geräten, wie Herzfrequenzmesser oder Fitnessgeräten zu kommunizieren. Die Übertragung und Kommunikation erfolgt über das General Attribute Profil (GATT), dass eine stromsparende Übertragung von Daten zwischen zwei Bluetooth Low Engery-Geräten ermöglicht (vgl. Android Developers, o.J.e). Zur Verwendung dieser Technologien, stellt die Bluetooth API verschiedene Methoden zur Verfügung. So kann bspw. mit der *startDicovery()*- oder *getBondedDevices()*-Methode die Suche nach Bluetooth-Geräten in der Umgebung gestartet oder alle gekoppelten Geräte abgefragt werden. Besteht eine Bluetooth-Verbindung, dann kann über das verbundene *BluetoothSocket* und den *InputStream* und *OutputStream* kommuniziert werden (vgl. Android Developers, o.J.f).

NFC ist eine weitere von Android unterstützte Technologie, die den Datenaustausch und die Kommunikation zwischen zwei mobilen Geräten oder einem NFC-Tag ermöglicht. Die NFC-Technologie bietet die Möglichkeit, Daten auf sehr kurzer Distanz drahtlos zu übertragen. Für einen Verbindungsaufbau werden nur wenige Zentimeter benötigt (vgl. Merkert, 2016 S. 170). Mittels NFC können kleine Datenmengen an NFC-Tags oder mobile NFC-fähige Geräte übertragen werden. Die NFC-Technologie wird bspw. für die Übertragung von Kontaktdaten, Bilder oder anderen Daten zwischen zwei Geräten verwendet

(vgl. Bach, 2012 S. 50). Die Daten werden im speziellen NDEF-Format (NFC Data Exchange Format) übertragen und können mithilfe der Android NFC API über das *Ndef-Message*-Interface ausgelesen und weiterverarbeitet werden. Dazu stellt die Android NFC API dem Entwickler verschiedene Interfaces und Methoden zur Verfügung, die den Zugriff und die Nutzung dieser Technologie ermöglichen (vgl. Android Developers, o.J.b).

3.2.9 Auffindbarkeit durch Suchmaschinen

Native Android Apps können i. d. R. nur über App Stores gefunden und installiert werden. Sie werden auf dem mobilen Gerät des Benutzers installiert und können nicht, wie Webapplikationen über eine URL aufgerufen und verwendet werden. Dadurch, dass sie auf dem mobilen Gerät des Benutzers installiert werden müssen, sind native Android Apps und dessen Inhalte, nicht direkt über das Internet zugänglich. Aus diesem Grund können die Inhalte der nativen Android App auch nicht von den Suchmaschinen gefunden und indexiert werden.

3.2.10 Zugangsdatenverwaltung

In manchen Fällen ist eine Anmeldung zur Nutzung einer App erforderlich. Dazu wird der Benutzer aufgefordert, seine Zugangsdaten einzugeben oder sich zu registrieren. Die Zugangsdaten des Benutzers werden dabei innerhalb der nativen Android App gespeichert. Durch die Speicherung der Zugangsdaten soll vermieden werden, dass bei jedem Aufruf der App erneut die Zugangsdaten eingegeben werden müssen. Damit wird dem Benutzer der immer wiederkehrende Anmeldeprozess erspart. Zur Speicherung der Zugangsdaten werden die in Kapitel 3.2.1 vorgestellten SharedPreferences verwendet. Die Zugangsdaten werden dabei in einer XML-Datei gespeichert und können nur durch die App selbst verwendet und abgerufen werden. Die Zugangsdaten bleiben auch nach Ablauf der Session, wenn die App geschlossen wurde oder abgestürzt ist, bestehen (vgl. Android Developers, o.J.m). Aufgrund dessen kann die App beim erneuten Aufruf, nachdem die Zugangsdaten gespeichert wurden, sofort verwendet werden.

3.3 Plattformunterstützung

Das Android Betriebssystem kommt auf unterschiedlichen Geräte-Typen, wie Smartphones oder Tablet-PCs zum Einsatz. Android ist ein Open Source Projekt, sodass jeder Gerätehersteller, ein auf Android basierendes Gerät entwickeln kann. Native Android Apps können

nur auf Android-Geräten installiert und verwendet werden. Damit sie auf unterschiedlichen Android-Geräten installiert und verwendet werden können, sollten sich native Android Apps an die unterschiedlichen Geräte-Eigenschaften anpassen können. Dazu gehört u. a. ein flexibles Layout und Design sowie die Anpassung an unterschiedliche Bildschirmgrößen. Zudem sollte auch beachtet werden, dass nicht immer alle Funktionen (bspw. Kompass) von den Geräten unterstützt und angeboten werden. Um eine native Android App für unterschiedliche Geräte-Typen entwickeln zu können, bietet Android ein dynamisches App Framework an, dass die Möglichkeit bietet, verschiedene Anpassungen für die unterschiedlichen Geräte-Konfigurationen zur Verfügung zu stellen. So ist es möglich, mehrere Layouts für unterschiedliche Bildschirmgrößen zu erstellen. Android wählt dann das zu den Geräte-Eigenschaften passende Layout und die Ressourcen aus (vgl. Android Developers, o.J.n).

Android bietet eine große Vielzahl an Funktionen und APIs, die je nach Gerät und Android-Version unterstützt werden. Auf vielen mobilen Geräten werden unterschiedliche Android-Versionen verwendet. Neuen Android-Versionen werden häufig neue APIs und Funktionen hinzugefügt (vgl. ebd.). Jede Android-Version stellt ein neues API-Level dar (vgl. Diedrich, 2017). Android 1.0 entspricht bspw. dem API-Level 1 und Android 9 dem API-Level 28 (vgl. Android Developers, o.J.a). Der Entwickler kann mithilfe dieser API-Level die Mindest-Android-Version festlegen, mit der eine native Android App noch kompatibel ist (vgl. Diedrich, 2017). Diese Mindest-Version kann in der *AndroidManifest.xml*-Datei festgelegt werden. Da nicht jedes Gerät alle Funktionen bietet und unterstützt, können bestimmte Apps nur für Benutzer zur Verfügung gestellt werden, dessen Geräte die Voraussetzungen für die Nutzung erfüllen. Diese Apps werden dem Benutzer dann auch nicht im Google Play Store vorgeschlagen und angezeigt. Dazu bietet Google dem Entwickler die Möglichkeit, bestimmte Voraussetzungen, wie Geräte-Funktionen, Android-Versionen oder Bildschirmgrößen festzulegen, die das Gerät erfüllen muss, damit die App im Google Play Store angezeigt und installiert werden kann (vgl. Android Developers, o.J.n). Der Google Play Store bietet jedoch die Möglichkeit, mehrere APKs (Multiple APK support) mit unterschiedlichen Konfigurationen und API-Level für verschiedene Geräte und Android-Versionen anzubieten. So kann eine App an die unterschiedlichen Geräte und Android-Version angepasst und dem Benutzer angeboten werden. Dem Benutzer wird im Google Play Store automatisch, die zu seinem Gerät und der Android-Version passende APK-Datei zum Herunterladen und Installieren zur Verfügung gestellt (vgl. Android Developers, o.J.w).

3.4 Look and Feel

Hersteller, wie Google und Apple, haben für ihre Plattformen (Android und iOS) eigene Komponenten, Layouts und Style-Guidelines entwickelt, die bei der Gestaltung und Entwicklung von Benutzeroberflächen einer App berücksichtigt werden sollen. Die Hersteller legen in ihren Style-Guidelines u. a. Richtlinien für Layouts, Navigation, Farbe, Icons, Typografie und verschiedene Komponenten fest. Durch die Einhaltung, der in den Style-Guidelines definierten Richtlinien, soll die App einfach zu verwenden, zu erlernen und vertraut sein. Damit soll dem Benutzer bei der Bedienung und Nutzung eine gute User Experience geboten werden. In den folgenden Kapiteln 3.4.1 und 3.4.2 werden die plattformspezifischen Style-Guidelines für Android und iOS sowie die zur Gestaltung angebotenen UI-Komponenten kurz vorgestellt.

3.4.1 Android

Zur grafischen Gestaltung von Benutzeroberflächen, stellt Android das Material Design zur Verfügung. Material Design ist eine Designsprache, die verschiedene Komponenten und Layouts zur grafischen Gestaltung der Benutzeroberflächen bietet. Um dem Benutzer eine einfache, schnell erlenbare und vertraute Benutzeroberfläche anbieten zu können, die eine gute User Experience bietet, stellt Google die Material Design-Guidelines zur Verfügung. Dabei handelt es sich um Richtlinien zur Gestaltung von Benutzeroberflächen (vgl. Android Developers, o.J.s).

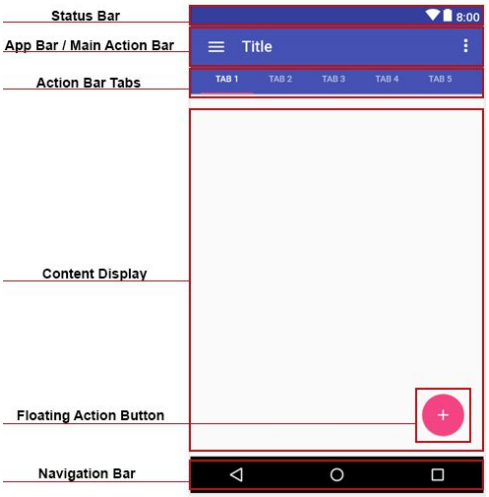

Abbildung 9 - Android Layout - Quelle: Eigene Darstellung

Das Layout einer nativen Android App besteht aus verschiedenen Komponenten, die dazu dienen, dem Benutzer eine übersichtliche und strukturierte Darstellung der Inhalte zu ermöglichen (siehe Abbildung 9). Dazu gehört die **Status Bar,** die u. a. Icons für Benachrichtigungen, Netzwerkverbindung, Akkuinformationen und die Uhrzeit enthält. Die **App Bar/Main Action Bar** enthält Schaltflächen zum Öffnen von Menüs sowie den Titel der aktuellen Seite. Die **Action Bar Tabs** ermöglichen den Wechsel zwischen verschiedenen Seiten und dienen zur Kategorisierung von Inhalten. Texte und andere Inhalte der App, werden im **Content Display**-Bereich dargestellt. Der **Floating**

Action Button dient dazu, eine bestimmte Funktion, wie bspw. das Hinzufügen von Artikeln oder Inhalten in den Mittelpunkt zu stellen und darauf aufmerksam zu machen. Die **Navigation Bar** bietet Schaltflächen zur Navigation innerhalb der App und auf dem Gerät.

Material Design bietet eine große Auswahl an vorgegeben Komponenten, die zur grafischen Gestaltung der Benutzeroberfläche, Inhalte und Funktionen einer nativen Android App verwendet werden können. Dazu zählen bspw. Komponenten, wie Buttons, Checkboxen, Radio Buttons, Switches, Slider, Spinner und auch Toolbars. In den Guidelines werden bestimmte Richtlinien zur Positionierung und Färbung dieser Komponenten definiert. Diese können farblich individuell an das eigene Design angepasst werden (vgl. Material.io, o.J.a). Abbildung 10 stellt einige der für die Gestaltung zur Verfügung stehenden Material Design Komponenten vor.

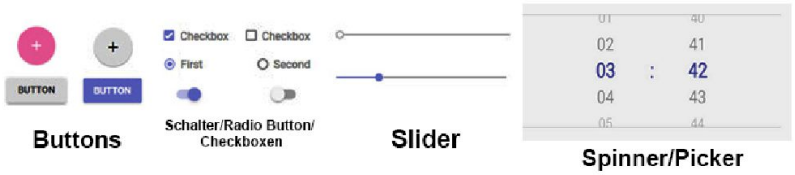

Abbildung 10 - Material Design UI-Komponenten - Quelle: Eigene Darstellung

Zur farblichen Gestaltung der Benutzeroberfläche bietet das Material Design Farbsystem 2014 Farbpaletten an. Zudem wird dem Entwickler ein Material Theme Editor angeboten, der die farbliche Gestaltung der App erleichtert. Das Farbsystem und der Editor ermöglichen es somit, die App so zu gestalten, dass sie zu der eigenen Marke oder dem gewünschten Design passt. Die Farbpaletten bestehen aus einer Primär- und Sekundärfarbe, die durch helle und dunkle Farbtöne dieser Farben erweitert wird. Diese Farben sind aufeinander abgestimmt und können bei der Gestaltung der Benutzeroberfläche dazu verwendet werden, Elemente hervorzuheben und eine gute Lesbarkeit zu ermöglichen (vgl. Material.io, o.J.c).

Des Weiteren bietet das Material Design eine Vielzahl an vordefinierten Icons (siehe Abbildung 11). Diese werden zur Gestaltung von Schaltflächen und dem Design angeboten, um damit die Bedienung und die Benutzeroberfläche der App leichter und verständlicher zu gestalten (vgl. Material.io, o.J.b).

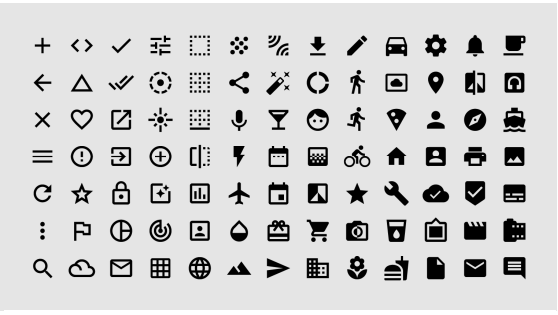

Abbildung 11 - Material Design Icons - Quelle: (Material.io, o.J.b)

Um die Inhalte und das Design einer App gut lesbar und sauber zu gestalten, spielt die Schriftart eine wichtige Rolle. Dementsprechend verwendet das Android Betriebssystem die

serifenlose Schriftart Roboto. Material Design bietet verschiedene Schriftschnitte (light, regular, medium) dieser Schriftart an, die zur Gestaltung des Designs und der Inhalte benutzt werden können. Das Material Design kann auf verschiedenen Plattformen verwendet werden. In den Material Design Guidelines sind u. a. die verschiedenen Einheiten (sp, pt, rem) für die Schriftgröße definiert, die für die verschiedenen Plattformen, wie Android, iOS und das Web gelten. Zudem sind Richtlinien zur Verwendung der verschiedenen Schriftschnitte und Schriftgrößen bspw. für Buttons, Überschriften, Bildbeschriftungen und Inhalte dargelegt, damit die Inhalte der Benutzeroberfläche gut lesbar und einheitlich dargestellt werden (vgl. Material.io, o.J.d).

3.4.2 iOS

Apple stellt zur grafischen Gestaltung von Benutzeroberflächen für iOS, das UIKit zur Verfügung. Das UIKit ist ein Framework, dass die einheitliche Gestaltung von iOS Apps ermöglicht und verschiedene Komponenten und Layouts zur grafischen Gestaltung der Benutzeroberfläche zur Verfügung stellt (vgl. Apple Inc., o.J.c). Um dem Benutzer eine gute User Experience bieten zu können, legt Apple, ähnlich wie Android, mit den Human Interface Guidelines, Richtlinien zur Gestaltung der Benutzeroberfläche mit dem UIKit fest (vgl. Apple Inc., o.J.d).

Die Benutzeroberfläche einer iOS App besteht im Wesentlichen aus drei Arten von Komponenten. Dazu gehören die sogenannten Bars, Views und Controls. Die Bars bieten dem Benutzer die Möglichkeit zur Navigation sowie Schaltflächen zur Ausführung von Funktionen. Die Views enthalten u. a. Inhalte, wie Texte, Grafiken und interaktive Elemente. Die Controls bieten verschiedene Komponenten, wie Buttons oder Switches, die die Ausführung von Funktionen ermöglichen (vgl. Apple Inc., o.J.c). In Abbildung 12 wird das Layout einer iOS App dargestellt. Wie auch bei Android, verfügt iOS über eine **Status Bar** mit einer Uhrzeit sowie Akku- und Netzwerkinformationen (vgl. Apple Inc., o.J.h). Die **Navigation Bar** ist, anders als bei Android, im oberen Bereich angeordnet und enthält Schaltflächen zum Navigieren

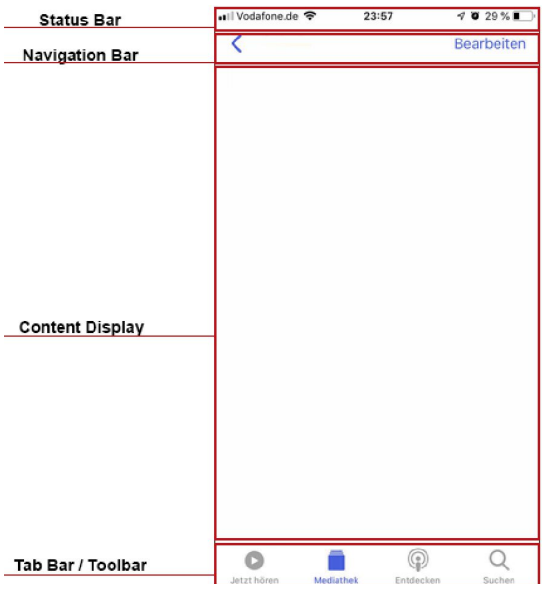

Abbildung 12 - iOS Layout - Quelle: Eigene Darstellung

und Ausführen von verschiedenen Funktionen (vgl. Apple Inc., o.J.g). Mittig im **Content Display**-Bereich werden, wie bei Android, die Inhalte dargestellt. Das iOS Layout bietet im unteren Bereich eine **Tab Bar/Toolbar**, zur direkten Navigation zu bestimmten Seiten und Funktionen innerhalb der App an (vgl. Apple Inc., o.J.k). Im wesentlichen unterscheidet sich das iOS- und Android Layout in der Anordnung der einzelnen Komponenten.

Zur grafischen Gestaltung der Benutzeroberfläche, Inhalten und Funtkionen, bietet Apple ebenfalls eine große Auswahl an Komponenten an, die sich jedoch im Design von den Android Material Design Komponenten unterscheiden (siehe Abbildung 13).

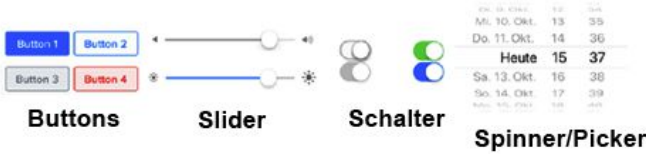

Abbildung 13 - iOS UI-Komponenten - Quelle: Eigene Darstellung

Die farbliche Gestaltung einer iOS App wird ebenfalls in den Human Interface Guidelines festgelegt. Ferner wird eine aus acht Farben bestehende Farbpalette angeboten. Apple erlaubt in seinen Richtlinien, Farben zu wählen, die zur eigenen Marke und dem Design passen. Jedoch soll darauf geachtet werden, dass die gewählten Farben miteinander kombinierbar sind und zueinander passen (vgl. Apple Inc., o.J.a).

Apple stellt ebenfalls eine große Auswahl an verschiedenen vordefinierten Icons zur Gestaltung von einfachen und verständlichen Benutzeroberflächen und Schaltflächen zur Verfügung (siehe Abbildung 14) (vgl. Apple Inc., o.J.j).

Abbildung 14 - Apple Icons - Quelle: In Anlehnung an Apple Inc., o.J.j

Damit die Inhalte und das Design einer iOS App deutlich und klar lesbar sind, verwendet Apple für iOS die Schriftart San Francisco (SF). Dabei handelt es sich ebenfalls um eine serifenlose Schriftart. Apple stellt, wie auch Android, verschiedene Schriftschnitte (regular und semi-bold) und Schriftgrößen zur Verfügung, dessen Verwendung in den Human Interface Guidelines festgelegt wird. Die Richtlinien legen u. a. fest, welche Schriftschnitte und Schriftgrößen bspw. für Überschriften, Unterüberschriften und Bildbeschriftungen verwendet werden sollen. Des Weiteren legt Apple in den Richtlinien fest, dass in iOS Apps nur eine Schriftart verwendet werden

soll und wichtige Informationen durch die Verwendung von anderen Schriftgrößen und einem anderen Schriftgewicht gekennzeichnet und hervorgehoben werden sollen (vgl. Apple Inc., o.J.l).

3.5 Veröffentlichung und Verbreitung

Fertiggestellte native Android Apps lassen sich über unterschiedliche Wege veröffentlichen und verbreiten. Damit eine native Android App veröffentlicht und verbreitet werden kann, muss zunächst eine signierte APK-Datei erstellt werden, die sich auf den mobilen Android-Geräten installieren lässt (siehe Kapitel 3.2.2). Die fertige APK-Datei enthält alle für die Ausführung und Installation notwendigen Dateien, wie bspw. den kompilierten Quellcode, Ressourcen sowie die *AndroidManifest.xml*-Datei. Die APK-Datei lässt sich mit Android Studio und dem integrierten Build-Tool Gradle erstellen (siehe Kapitel 3.1.2). Die native Android App kann anschließend über einen App Store veröffentlicht werden (vgl. Google Inc., o.J.k). Zur Veröffentlichung und Verbreitung von nativen Android Apps bietet Google den Google Play Store an. Der Google Play Store ist auf den meisten mobilen Android-Geräten vorinstalliert (vgl. Künneth, 2018 S. 106). Native Android Apps können auch über andere App Stores, wie z. B. den Amazon App Store oder Samsung Galaxy Store veröffentlicht und verbreitet werden. Über einen App Store kann der Benutzer nach nativen Android Apps suchen, diese herunterladen und auf seinem mobilen Gerät installieren (vgl. Richter, 2018 S. 343). Native Android Apps können auch auf einer eigenen Webseite zum Herunterladen angeboten werden (vgl. Künneth, 2018 S. 107). Ferner lassen sie sich auch per E-Mail als Anhang versenden und verbreiten. Um die native Android App auf dem mobilen Gerät installieren zu können, genügt das Herunterladen und Ausführen der APK-Datei (vgl. Richter, 2018 S. 343). Dazu muss zunächst in den Einstellungen die Installation aus unbekannten Quellen zugelassen werden (vgl. Künneth, 2018 S. 106).

Um eine native Android App bspw. über den Google Play Store veröffentlichen und verbreiten zu können, wird ein kostenpflichtiges Entwicklerkonto benötigt. Für die meisten anderen App Stores fallen jedoch keine Kosten an (vgl. Richter, 2018 S. 343). Damit eine native Android App in App Stores veröffentlicht werden kann, müssen die für den jeweiligen App Store geltenden Richtlinien erfüllt und eingehalten werden (vgl. Google Inc., o.J.m). Zusätzlich müssen beim Einreichen verschiedene Angaben zur App gemacht werden. Dazu gehören u. a. der App-Name (in unterschiedlichen Sprachen), eine kurze und lange Beschreibung

der App (vgl. Richter, 2018 S. 343) sowie ein App-Icon. Ebenfalls können Screenshots oder optional ein Video-Trailer, die eine Vorschau ermöglichen, hinterlegt werden. Des Weiteren können Angaben zur Gerätekompatibilität, bzw. den unterstützten Android-Versionen gemacht und ggf. der Preis für die native Android App festgelegt werden (vgl. Android Developers, o.J.i). Abschließend durchläuft die zur Veröffentlichung eingesendete native Android App den Überprüfungs- und Freigabeprozess des App Stores und kann nach erfolgreicher Überprüfung im App Store veröffentlicht werden (vgl. Google Inc., o.J.l).

4 Untersuchung von Progressive Web Apps

In diesem Kapitel wird der App-Typ Progressive Web App auf Basis der in Kapitel 2.5 vorgestellten Vergleichskategorien im Rahmen einer Literaturrecherche untersucht, vorgestellt und kurz erläutert.

4.1 Entwicklung

4.1.1 Webtechnologien

Progressive Web Apps sind moderne Webapplikationen, die auf verschiedenen Webtechnologien basieren. Sie werden wie herkömmliche Webapplikationen mit HTML, CSS und JavaScript entwickelt (vgl. Osmani, 2015).

HTML (HyperText Markup Language) ist eine textbasierte Auszeichnungssprache, die zur strukturierten Darstellung von Texten und verschiedenen Medien, wie bspw. Grafiken und Videos verwendet wird. HTML-Dateien sind die Grundlage für Webapplikationen und das World Wide Web (vgl. Wolf, 2016 S. 39). Sie werden vom Webbrowser in Webapplikationen umgewandelt und anschließend dargestellt (vgl. Robbins, 2014 S. 1).

HTML5 stellt den neuen Standard für die Auszeichnungssprache im Web dar und basiert auf der HTML 4.01 Strict Spezifikation (vgl. Wolf, 2016 S. 40). HTML5 wird von den meisten aktuellen Browsern unterstützt (vgl. Robbins, 2014 S. 7). Zudem bietet HTML5 zahlreiche neuen Funktionen und Features, darunter neue Elemente, Attribute, Event-Handler und APIs (vgl. ebd. S. 2 f.). Über die Media API ist es z. B. möglich, Audio- und Video-Medien innerhalb von Webapplikationen einzubinden und abspielen zu können. Außerdem bietet HTML5 die Möglichkeit, über die Web Storage API, verschiedene Webressourcen lokal im Cache des Browsers zu speichern. Neben den hier erwähnten APIs, stehen noch zahlreiche

andere zur Verfügung, die zur Entwicklung von Webapplikationen verwendet werden können (vgl. Robbins, 2014 S. 5 f.).

Zur optischen Gestaltung von Webapplikationen kommt die Formatierungssprache CSS (Cascading Style Sheets) zum Einsatz (vgl. Wenz, 2014 S. 315), denn HTML wird i. d. R. ausschließlich zur semantischen Strukturierung von Inhalten verwendet. HTML und CSS ermöglichen es, die Inhalte von der Darstellung einer Webapplikation zu trennen (vgl. Wolf, 2016 S. 41). CSS bietet dem Entwickler zahlreiche Möglichkeiten zur Gestaltung von Webapplikationen (vgl. Bühler et al. S. 44). Mit CSS lassen sich die Eigenschaften von HTML-Elementen verändern. So ist es z. B. möglich, Hintergrundfarben, Textfarben, Höhe und Breite mithilfe von speziellen CSS-Regeln festzulegen (vgl. selfhtml.org, o.J.a) und auch Animationen zu erstellen. Auch lassen sich Webapplikationen dynamisch an unterschiedliche Bildschirmgrößen und Geräte anpassen (vgl. Bühler et al. S. 44).

Mit JavaScript können dynamische und interaktive Funktionalitäten zu einer Webapplikation hinzugefügt werden (vgl. Mozilla Developer Network, 2016a). JavaScript ist eine objektorientierte Programmiersprache (vgl. Theis, 2016 S. 15), die clientseitig im Browser und nicht auf einem Server ausgeführt wird (vgl. Bühler et al., 2018 S. 1). JavaScript wird durch ECMAScript (ECMA 262) standardisiert (vgl. Mozilla Developer Network, 2016a). Dieser Standard wird von den meisten Webbrowsern umgesetzt und unterstützt (vgl. Theis, 2016 S. 17).

Mit JavaScript können Inhalte einer Webapplikation verändert und auf bestimmte Aktionen, Interaktionen und Ereignisse reagiert werden (vgl. ebd. S. 15). Zudem ist es bspw. möglich, interaktive und dynamische Inhalte, wie Bildergalerien oder auch bestimmte Reaktionen auf Button-Klicks oder Elemente, zu realisieren. Durch den Zugriff auf das *Document Object Model (DOM)* eines HTML-Dokumentes, ist es mit Hilfe von JavaScript möglich, dieses zu manipulieren und so jedes Element der HTML-Struktur anzupassen. Dem Entwickler werden zahlreiche APIs, Frameworks und Bibliotheken angeboten, die zur Entwicklung verwendet werden können (vgl. Mozilla Developer Network, 2016a).

4.1.2 Tools

Progressive Web Apps sind moderne Webapplikationen und benötigen demnach keine gerätespezifische IDE. Zur Entwicklung von Progressive Web Apps eignen sich einfache

Code-Editoren, die zur Entwicklung von herkömmlichen Webapplikationen verwendet werden. Entwickler können somit auf viele Editoren und Programme zurückgreifen, die sich zur Entwicklung von Progressive Web Apps eignen. Zum Analysieren und Debuggen von Progressive Web Apps, stellt Google die Tools Lighthouse und Chrome DevTools zur Verfügung.

Lighthouse ist ein von Google entwickeltes Open Source Tool und bietet die Möglichkeit, eine Webapplikation automatisch auf die Eigenschaften und Funktionen einer Progressive Web App zu überprüfen (vgl. Google Inc., o.J.h). Das Tool wird als Browser-Erweiterung und Command-line Tool zur Verfügung gestellt (vgl. Ater, 2017 S. 61). Dabei wird die Webapplikation unter verschiedenen Bedingungen ausgeführt und geladen. Beispielsweise wird die Offline-Nutzbarkeit oder der Aufruf der Webapplikation bei schlechter Internetverbindung überprüft. Es wird ebenfalls überprüft, ob ein Service Worker registriert ist oder ein Ladebildschirm konfiguriert wurde. Neben den hier genannten Kriterien gibt es noch weitere, die Bestandteil der Überprüfung sind (vgl. Domes, 2017 S. 248 f.).

Nach Abschluss der Überprüfung, wird dem Entwickler eine Checkliste mit allen erfüllten und nicht erfüllten Kriterien, inklusive Verbesserungsvorschlägen angezeigt. Die Checkliste gibt dem Entwickler einen Überblick, ob und inwiefern die Webapplikation alle Eigenschaften und Funktionen einer Progressive Web App erfüllt (vgl. ebd. S. 249).

Zur Entwicklung von Progressive Web Apps stehen neben Lighthouse, die Google Chrome DevTools zur Auswahl. Die Chrome DevTools sind eine Reihe von Entwicklertools zum Analysieren und Debuggen von Webapplikationen. Diese sind standardmäßig im Google Chrome Browser integriert. Die Chrome DevTools bieten dem Entwickler eine Vielzahl an Tools, die u. a. die Möglichkeit zur Simulation von gerätespezifischen Darstellungen sowie Tools zur Durchführung von Änderungen am *Document Object Model* und den Cascading Style Sheets (CSS) bieten. Ebenfalls kann das JavaScript debuggt werden (vgl. Google Inc., o.J.b). Zudem können mit Hilfe der Chrome DevTools die wichtigsten Kernkomponenten einer Progressive Web App analysiert und debuggt werden. Auf diese Weise ist es möglich, das Web App Manifest, den Service Worker sowie die Service Worker Caches zu analysieren und zu debuggen (vgl. Basques, o.J.). Ebenso können die in der IndexedDB gespeicherten Daten eingesehen und bei Bedarf auch gelöscht werden (vgl. Sheppard, 2017 S. 18). Dasselbe gilt auch für die Service Worker Caches (vgl. Basques, o.J.). Zusätzlich bieten die Chrome DevTools Funktionen zum Testen der Offline-Nutzbarkeit, Background Sync, Add

to Homescreen-Funktion sowie zum Senden einer Test-Benachrichtigung (Push Notification) (vgl. Sheppard, 2017 S. 18).

4.2 Technische und funktionale Möglichkeiten

4.2.1 Offline-Nutzbarkeit und Datenhaltung

Die Offline-Nutzbarkeit und Datenhaltung lässt sich mithilfe verschiedener neuer Technologien, wie dem Service Worker, Background Sync und der IndexedDB realisieren. Die Offline-Nutzbarkeit von Progressive Web Apps wird mit Service Workern umgesetzt. Dementsprechend lassen sich Progressive Web Apps unabhängig von der Netzwerkverbindung und bei keiner oder schlechter Internetverbindung verwenden (vgl. Springer, 2016 S. 121). Ein Service Worker ist ein JavaScript, welches unabhängig von der Webapplikation und im Hintergrund des Browsers ausgeführt wird. Service Worker können nicht auf das *Document Object Model* zugreifen, weil sie in einem anderen Thread, als das JavaScript der Progressive Web App, ausgeführt werden (vgl. Gaunt, o.J.b). Durch diese Trennung soll sichergestellt werden, dass der Service Worker asynchron funktioniert und nicht blockiert wird (vgl. Mozilla Developer Network, 2018b). Der Service Worker funktioniert wie ein Proxy zwischen dem Browser und dem Server (siehe Abbildung 15) (vgl. Braun, 2017 S. 105). Netzwerk-Anfragen, die von der Progressive Web App getätigt werden, können abgefangen und entsprechend behandelt werden (vgl. Gaunt, o.J.b). Demnach kann mithilfe eines Service Workers, unabhängig von der Netzwerkverbindung, festgelegt werden, ob die Daten einer Progressive Web App zuerst aus dem Cache oder direkt vom Server geladen werden sollen (siehe Abbildung 15) (vgl. Braun, 2017 S. 108). So kann bspw. über den Service Worker die Application-Shell, die das Grundgerüst und die Oberfläche einer Progressive Web App bildet und meist aus statischen Elementen besteht (vgl. East, 2015), im Cache abgelegt werden. Dadurch wird vermieden, dass diese bei jedem Aufruf der Webapplikation erneut heruntergeladen werden müssen. So lässt sich u. a. auch die Offline-Nutzbarkeit realisieren (vgl. Osmani, o.J.). Dazu können, je nach Anwendungsfall, verschiedenen Caching-Strategien eingesetzt werden (vgl. Braun, 2017 S. 108).

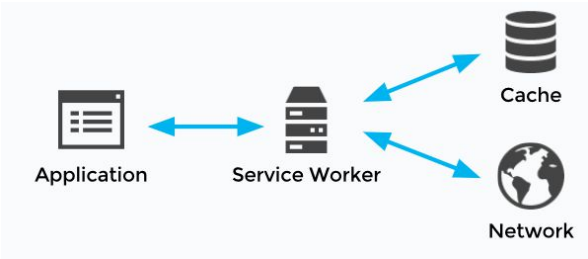

Abbildung 15 - Service Worker als Proxy zwischen Browser und Server - Quelle: (Belmont, 2016)

Häufig bieten Webapplikationen die Möglichkeit, Daten an den Server zu senden. Dazu gehören, je nach Anwendungsfall, Formulare, Nachrichten, Beiträge oder Einstellungen, die versendet oder bearbeitet werden können (vgl. Archibald, o.J.). Um dem Benutzer eine von der Internetverbindung unabhängige Übertragung der Daten zu ermöglichen, wird die Background Sync API zur Verfügung gestellt. Durch die Background Sync API werden die zu versendenden Daten, bei schlechter oder keiner Internetverbindung in einer Warteschlange abgelegt. Die Daten eines POST-Requests können auch über die IndexedDB zwischengespeichert werden. Sobald eine stabile Internetverbindung wieder vorhanden ist und das Sync-Event ausgelöst wurde, werden die zu übertragenden Daten durch den Service Worker an den Server gesendet und synchronisiert (vgl. Hume et al., 2018 S. 120 ff.). Die Übertragung und Synchronisation der Daten erfolgt auch, wenn die Progressive Web App oder der Browser bereits geschlossen oder eine andere Webapplikation aufgerufen wurde (vgl. Archibald, o.J.). Dem Benutzer kann so eine unterbrechungsfreie Offline-Nutzbarkeit der Progressive Web App ermöglicht werden (vgl. Hume et al., 2018 S. 120).

Die lokale Speicherung und Datenhaltung ist mit der IndexedDB API möglich. Die IndexedDB ist eine API zur clientseitigen Speicherung und Datenhaltung von Daten und Dateien innerhalb des Browsers (vgl. Mozilla Developer Network, 2013b). Es handelt sich bei der IndexedDB um eine JavaScript basierte und objektorientierte Datenbank. Diese kann zur Speicherung von JavaScript Objekten, Numbers, Blobs, Strings, Arrays und anderen Datentypen, die von JavaScript unterstützt werden, verwendet werden (vgl. Ater, 2017 S. 94 f.). Die IndexedDB speichert Daten als Schlüssel-Wert-Paare, um den Zugriff auf die gespeicherten Daten zu ermöglichen. Indizes bieten die Möglichkeit einer schnellen und effizienten Suche nach Einträgen (vgl. Mozilla Developer Network, 2013b). Die IndexedDB ist eine browserbasierte Datenbank, die unabhängig von der Internetverbindung verwendet werden kann. Durch die Same-Origin-Policy können andere Webapplikationen mit einer anderen Domain nicht auf die in der IndexedDB hinterlegten Daten zugreifen. Um die Datenintegrität zu gewährleisten, kommen Transaktionen zum Einsatz. Dabei wird sichergestellt, dass Operationen erfolgreich ausgeführt werden oder im Fehlerfall, alles wieder rückgängig gemacht werden kann. Transaktionen stellen sicher, dass alle Daten vollständig und korrekt in der Datenbank abgespeichert werden (vgl. Ater, 2017 S. 94 f.). Die meisten Operationen werden asynchron von der IndexedDB ausgeführt. Um Werte aus der Datenbank abzufragen, können Callback-Funktionen definiert werden, die beim Aufruf den abgefragten Wert beinhalten.

Die IndexedDB verwendet keine Structured Query Language (SQL). Damit über die Ergebnismenge einer Abfrage iteriert werden kann, können Cursor verwendet werden (vgl. Mozilla Developer Network, 2013a).

4.2.2 Installierbarkeit

Progressive Web Apps lassen sich mithilfe von modernen Webetechnologien, wie dem Web App Manifest und der App Install Banner-Funktion, direkt auf dem Homescreen eines mobilen Geräts installieren. Ausgeführt wird die Progressive Web App durch Betätigen des Icons auf dem Homscreen. Dadurch muss weder der Browser geöffnet, noch die URL eingegeben werden (vgl. Mozilla Developer Network, 2018a).

Abbildung 16 - App Install Banner - Quelle: Eigene Darstellung

Für die Installation ist eine sogenannte Web App Manifest-Datei erforderlich. Dabei handelt es sich um eine JSON-Datei, die die Eigenschaften und Informationen einer Progressive Web App enthält (vgl. Hume et al., 2018 S. 67). Mithilfe der Web App Manifest-Datei können u. a. Eigenschaften, wie Name, Titel, Icons, Hintergrundfarben, URL für die bevorzugte Startseite, die beim Ausführen geöffnet werden soll, festgelegt werden. Zudem kann die Ausrichtung (Hoch- oder Querformat) und der Anzeigemodus (bspw. Fullscreen) festgelegt werden. Ebenso ist es möglich, einen Begrüßungsbildschirm (Splash-Screen) sowie verschiedene andere Eigenschaften festzulegen. Mithilfe der definierten Eigenschaften, ist es möglich, eine vergleichbare User Experience zu schaffen, wie bei nativen Apps (vgl. W3C, 2018a). Die Web App Manifest-Datei kann über ein <*link*>-Element in die Webapplikation eingebunden werden (vgl. Mozilla Developer Network, 2018a). Progressive Web Apps können somit auf einem mobilen Gerät installiert werden, ohne einen App Store verwenden zu müssen (vgl. LePage, 2018b). Zur Installation auf dem Homescreen, wird dem Benutzer im Browser ein sogenanntes App Install Banner angezeigt (siehe Abbildung 16) (vgl. Springer, 2016 S. 123). Damit das App Install Banner im Browser dargestellt wird, müssen verschiedene Voraussetzungen erfüllt werden. Es muss bspw. ein Service Worker registriert sein, die Progressive Web App muss über eine verschlüsselte Verbindung (HTTPS) zur Verfügung gestellt werden und eine Web App Manifest-Datei mit Eigenschaften und Parametern, wie z.B. short_name, name und Icon, enthalten sein. Zudem muss die Progressive Web App mindestens zweimal im Abstand von fünf Minuten besucht

worden sein und darf nicht bereits auf dem Homescreen installiert sein (vgl. Mozilla Developer Network, 2018a). Der Chrome-Browser auf Android-Geräten, erstellt während der Installation eine sogenannte WebAPK-Datei, auf Basis der Informationen und Eigenschaften aus der Web App Manifest-Datei. Durch die Installation über die WebAPK-Datei, kann die Progressive Web App im App-Launcher sowie in den Android Einstellungen angezeigt werden (vgl. LePage, 2018a).

App Install Banner werden nicht von Safari unterstützt und dadurch nicht auf iOS-Geräten angezeigt. iOS-Benutzer können jedoch Progressive Web Apps manuell über die Add to Homescreen-Funktion zum Homescreen hinzufügen (vgl. Firtman, 2018).

4.2.3 Benachrichtigungen

Push Notifications werden dazu verwendet, um den Benutzer über wichtige Informationen oder Neuigkeiten zu informieren. Sie haben den Vorteil, dass sie die Aufmerksamkeit auf die Webapplikation lenken und den Benutzer an diese binden. Der Benutzer hat die Möglichkeit, empfangene Push Notifications zu schließen oder die Progressive Web App durch Betätigen der Benachrichtigung direkt aufzurufen (vgl. Hume et al., 2018 S. 81). Push Notifications werden mittlerweile von vielen Browsern unterstützt und können dadurch auch für Webapplikationen genutzt werden (vgl. Steyer, 2017a S. 40). Progressive Web Apps bieten die Möglichkeit, Push Notifications direkt über den Browser zu empfangen, auch wenn diese nicht im Vordergrund steht oder der Browser geschlossen ist (vgl. Hume et al., 2018 S. 82).

Die Grundlage für Push Notifications bilden verschiedene Technologien und Schnittstellen. Dazu gehören u. a. der Service Worker, das Web App Manifest (vgl. Springer, 2016 S. 124) sowie die Push- und Notifications API (vgl. Medley, o.J.). Die Push API ist für den Empfang von Push Notifications, die von einem Server an den Service Worker des Empfängers gesendet werden, zuständig (vgl. Mozilla Developer Network, o.J.h). Die Notification API hingegen ermöglicht die Darstellung von Push Notifications auf dem mobilen Gerät des Benutzers, auch wenn die Progressive Web App nicht im Vordergrund steht oder geöffnet ist (vgl. Mozilla Developer Network, o.J.g).

Für den Empfang von Push Notification muss der Webapplikation eine Erlaubnis erteilt werden. Dazu wird der Benutzer beim Aufruf der Webapplikation aufgefordert (vgl. Braun, 2017 S. 108). Bei Einverständnis werden Push Notifications durch den Benutzer abonniert.

Anschließend wird vom Browser ein sogenanntes *PushSubscription*-Objekt zur Verfügung gestellt und eine Anfrage an den zugehörigen Push-Service (z. B. Google Cloud Messaging) gesendet (vgl. Gaunt, o.J.a). Dabei wird der Browser des Benutzers anhand einer eindeutigen ID beim Push-Service registriert (vgl. Steyer, 2017a S. 41). Die ID wird dem *PushSubscription*-Objekt hinzugefügt und die darin enthaltenen Informationen mittels JavaScript und der Push API an den Server übermittelt (siehe Abbildung 17). Diese Informationen werden anschließend in einer Datenbank gespeichert und zum Versenden von Push Notifications an den Benutzer verwendet (vgl. Gaunt, o.J.a).

Zum Senden von Push Notifications an den Benutzer, wird die API des Push-Service aufgerufen und die zu versendenden Daten und der Empfänger angegeben. Zusätzlich können verschiedene Parameter übergeben werden. Unter anderem kann die Dringlichkeit oder die Lebensdauer bzw. Verweildauer in der Wartschlange, bei Nichterreichbarkeit des Empfängers, definiert und übergeben werden. Nach Ausführen des API-Aufrufs wird die Anfrage durch den Push-Service überprüft und die Push Notification an den entsprechenden Empfänger (Browser) gesendet (siehe Abbildung 17). Bei Nichterreichbarkeit des Browsers, wird die Benachrichtigung in einer Warteschlange abgelegt (vgl. ebd.).

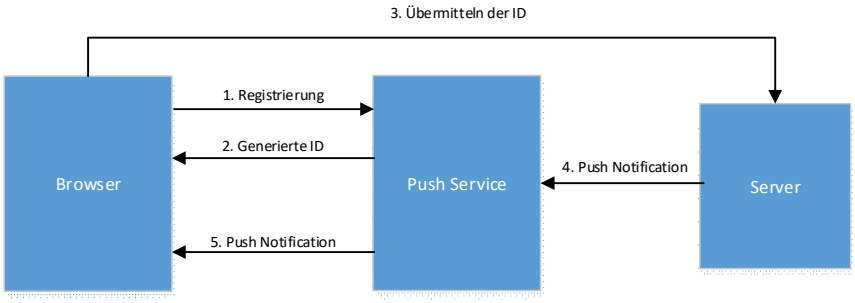

Abbildung 17 - Funktionsweise von Push Notifications - Quelle: Eigene Darstellung

Bei erfolgreicher Übertragung an den Empfänger (Browser) wird die vom Push-Service gesendete Benachrichtigung vom Browser entschlüsselt und der zugehörige Service Worker gestartet und das *push*-Event wird ausgelöst (vgl. ebd.). Über die *showNotification*-Methode kann die Push Notification dem Benutzer angezeigt werden (vgl. Springer, 2016 S. 124).

4.2.4 Inter-App Kommunikation

Progressive Web Apps können Inhalte mit anderen Apps auf einem mobilen Gerät teilen. Ermöglicht wird diese Funktion durch zwei neue Schnittstellen, die Web Share API und die

Web Share Target API. Die Web Share API ermöglicht es, Inhalte einer Progressive Web App mit anderen mobilen Apps zu teilen. Dabei können, Titel, Text und eine URL übertragen werden. Zukünftig soll es möglich werden, auch Bilddaten (Blobs) zu übertragen (vgl. Ackermann, 2016b). Die Web Share-Funktion zum Teilen von Inhalten wird durch Betätigen einer Teilen/Share-Schaltfläche aufgerufen und ausgeführt. Dazu muss die Web-Share-Funktion in der Progressive Web App implementiert sein. Anschließend wird der native Teilen-Dialog des mobilen Gerätes aufgerufen. Der Benutzer hat die Möglichkeit, die gewünschte App auszuwählen, an die die Inhalte gesendet werden sollen (siehe Abbildung 18). Damit die Web Share API verwendet werden kann, muss die Progressive Web App mittels HTTPS verschlüsselt sein und von dem verwendeten Browser unterstützt werden (vgl. Kinlan et al., o.J.).

Umgekehrt soll es auch möglich werden, Inhalte, die von anderen Apps geteilt oder gesendet werden, zu empfangen. Dazu soll die Web Share Target API zur Auswahl stehen. Wird die Web Share Target API eingesetzt, dann steht die Progressive Web App im nativen Teilen-Dialog als Ziel zur Auswahl (siehe Abbildung 18) (vgl. Ackermann, 2016b).

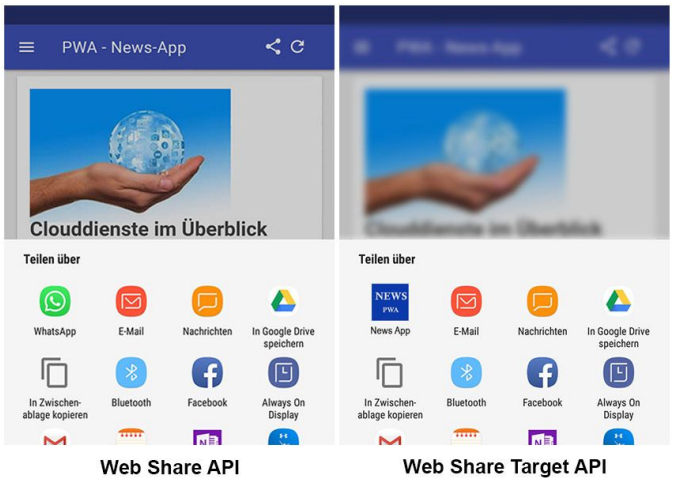

Abbildung 18 - Web Share- und Web Share Target API - Quelle: Eigene Darstellung

4.2.5 Zugriffskontrolle

Progressive Web Apps können über bestimmte Funktionen und Features verfügen, die eine Zugriffsberechtigung des Benutzers erfordern. Dazu gehört u. a. der Zugriff auf die Kamera oder das Mikrofon. Damit diese oder andere Funktionen genutzt werden können, wird der Benutzer i. d. R. über ein Pop-up-Fenster im Browser aufgefordert, eine Entscheidung über

die Zugriffsberechtigung zu treffen (siehe Abbildung 19). Dadurch wird verhindert, dass Webapplikationen nicht ohne ausdrückliche Erlaubnis auf bestimmte Funktionen oder Daten zugreifen können. Für jede Funktion können die Zugriffsberechtigung separat erteilt werden.

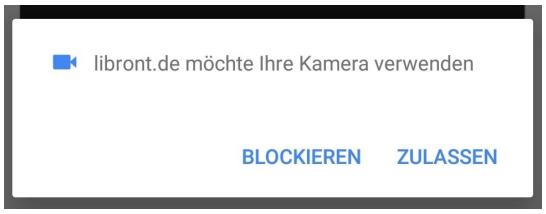

Abbildung 19 - Zugriffsberechtigung - Quelle: Eigene Darstellung

Die Permissions API ermöglicht es, die Zugriffsberechtigungen bestimmter Funktionen einer Webapplikation anzuzeigen und erteilte Berechtigungen zu wiederrufen. Das Wiederrufen wird jedoch noch nicht von allen Browsern unterstützt (vgl. Mozilla Developer Network, 2013c). Es lassen sich Berechtigungen für verschiedene Funktionen, wie z. B. Benachrichtigungen, Geolocation, Background-Sync, Zugriff auf Hardware und Sensoren (wie Kamera, Mikrofon, Accelerometer, Gyroskop, Magnetometer) und verschiedene anderen Funktionen anzeigen (vgl. W3C, 2017b). Mit der Permissions API ist es ebenfalls möglich, erst bei eintreten bestimmter Ereignisse (bspw. Button-Klick), die Berechtigung des Benutzers für den Zugriff auf die notwendige Funktion einzuholen (vgl. Matt Gaunt, o.J.).

4.2.6 Multimedia

Die HTML5 <*audio*>- und <*video*>-Elemente ermöglichen Progressive Web Apps, ohne die Verwendung von speziellen Plugins, verschiedene Audio- und Video-Medien direkt in der Webapplikation darzustellen und wiederzugeben (vgl. Mozilla Developer Network, 2009). Audio- und Video-Dateien lassen sich über das *source*-Element einbinden. Jedoch gibt es hierbei verschiedene Einschränkungen. Wie im Folgenden aufgeführt, werden nicht alle Formate von den Browsern unterstützt. Um möglichst viele Plattformen unterstützen zu können, ist es möglich, mehrere Source-Dateien mit unterschiedlichen Formaten als Fallback anzugeben. Der Browser geht alle Dateien durch und wählt das unterstützte Format aus (vgl. LePage, 2014). Die nachfolgende Aufzählung zeigt die verschiedenen Formate, die zur Wiedergabe von Medien in Progressive Web Apps verwendet werden können. Diese Formate werden von den meisten Browsern unterstützt (vgl. Mozilla Developer Network, 2009):

- **Audio:** WAV, MP3, AAC, FLAC, WebM, Ogg Vorbis
- **Video:** MPEG-4, H264, H263, FLAC, WebM, Ogg Theora und Vorbis

Die <*audio*> und <*video*>-Elemente bieten zahlreiche Anpassungs- und Konfigurationsmöglichkeiten an. So ist es bspw. möglich, die Darstellung und Steuerung der Multimedia-Elemente anhand verschiedener Optionen und Attribute anzupassen. Zu den Optionen und Attributen gehören bspw. die Höhe und Breite, das automatische Starten (autoplay), das erneute Abspielen (loop), das Stummschalten (muted) sowie die Darstellung eines Posters (poster) vor der Wiedergabe des Audio/Video-Mediums (vgl. Mozilla Developer Network, 2016b). Mittels JavaScript und CSS können verschiedene Events des Video-Elements gesteuert und die Darstellung angepasst werden (vgl. LePage, 2014).

Durch die Media Session API ist es möglich, in der Progressive Web App wiedergegebene Audio- und Video-Medien über die Notification-Bar und den Sperrbildschirm des Gerätes zu steuern. Dabei wird ein Hinweis in der Notification-Bar angezeigt, dass der Browser Audio- oder Video-Medien abspielt. Unter anderem wird der Titel und Name der Webapplikation sowie die über die Media Session API festgelegten Metadaten, der wiedergegebenen Audio- oder Video-Medien, dargestellt. Dazu zählen bspw. Titel, Album, Coverbild sowie der Künstlername. Zudem werden verschiedene Interaktionsmöglichkeiten geboten, um die Wiedergabe der Medien über die Schaltflächen Pause, Play, Weiter und Zurück, zu steuern (siehe Abbildung 20) (vgl. Ater, 2017 S. 246 ff.).

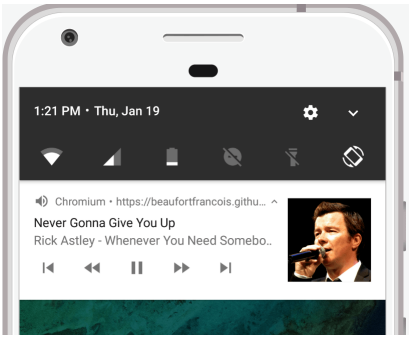

Abbildung 20 - Audio/Video Steuerung mittels Media Session API - Quelle: (Beaufort, o.J.)

4.2.7 Geräte-Zugriff

Der Zugriff einer Progressive Web App auf Geräte- und Software-Funktionen kann durch die Nutzung verschiedener moderner Browser APIs umgesetzt werden. Diese APIs bieten u. a. Zugriffe auf die im Gerät verbauten Hardware-Komponenten, wie bspw. Kamera, Mikrofon, GPS, Bluetooth sowie den Zugriff auf die verschiedenen Sensoren des Gerätes. Neben dem Hardware-Zugriff, stehen z. B. auch APIs für den Zugriff auf die Zwischenablage und Dateien zur Verfügung (siehe Abbildung 21). Jedoch handelt es

Zugriffsmöglichkeiten und Funktionen	APIs
Kamera und Mikrofon	getUserMedia API
Dateizugriff	File API
Geolocation/GPS	Geolocation API
Geräteausrichtung und Bewegung	Device Ortientation API, Device Motion Events
Beschleunigungssensor	Accelerometer API
Gyroskop	Gyroscope API
Magnetometer	Magnetometer API
Gravity-Sensor	Gravity Sensor API
Akkuinformationen	Battery API
Vibrations-Komponente	Vibration API
Netzwerkinformationen	Network Information API
Zwischenablage	Clipboard API
Bluetooth	Web Bluetooth API
NFC	Web NFC API

Abbildung 21 - APIs für den Geräte- und Software-Zugriff - Quelle: Eigene Darstellung

sich bei einigen APIs noch um Entwürfe, die noch nicht endgültig standardisiert wurden (vgl. Hume et al., 2018 S. 165). Die verschiedenen APIs und Zugriffsmöglichkeiten werden in diesem Kapitel vorgestellt. Welche APIs und Funktionen konkret von den einzelnen Browsern und Plattformen unterstützt werden, wird in Kapitel 4.3 aufgezeigt.

Der Zugriff auf die Kamera und das Mikrofon ist mit der getUserMedia API möglich. Diese API bietet den Zugriff auf die Multimedia-Streams des mobilen Gerätes und ermöglicht es, Bilder, Videos oder Audio aufzunehmen (vgl. W3C, 2017a). Der Benutzer muss für den Zugriff auf die Kamera und das Mikrofon die Berechtigung erteilen. Dazu wird der Benutzer beim Aufruf der Funktion aufgefordert. Über Parameter kann festgelegt werden, ob Video- und Audioinhalte übertragen und dargestellt werden sollen. Außerdem lassen sich durch Definieren verschiedener Parameter, wie bspw. *width* und *height*, optional die Auflösung des Videos einstellen sowie die zu verwendende Kamera (Front- oder Backkamera) festlegen (vgl. Mozilla Developer Network, 2015). Video und Audio können dann in das definierte <*video*>-HTML-Element eingefügt, dargestellt und darin wiedergegeben werden (vgl. W3C, 2017a).

Die File API bietet die Möglichkeit, lokale Dateien innerhalb einer Webapplikation auszuwählen und die Inhalte der Dateien auszulesen. Die File API wurde mit HTML5 zum DOM hinzugefügt. Die Auswahl der Dateien ist über das <*input*>-Element oder mittels Drag and Drop möglich (vgl. Mozilla Developer Network, o.J.i). Die File API ermöglicht den Dateizugriff mittels JavaScript. Ein beliebiger und direkter Zugriff auf alle Dateien durch die Webapplikation ist mit dieser API aus Sicherheitsgründen nicht möglich. Mithilfe der File API kann nur auf die Dateien zugegriffen werden, die von dem Benutzer ausgewählt und zur Verfügung gestellt wurden. Die API bietet u. a. die Möglichkeit, auf die Inhalte und Informationen der Dateien, wie Name, Größe, MIME-Type, zuzugreifen (vgl. Wenz, 2014 S. 460).

Die Nutzung der Geolocation API ermöglicht es, die aktuelle Position und den Standort des Benutzers zu ermitteln und diesen der Webapplikation mitzuteilen (vgl. ebd. S. 413). Auch hier muss der Benutzer zunächst die Berechtigung erteilen, damit die Standortinformationen ermittelt werden dürfen (vgl. Mozilla Developer Network, o.J.d). Bei der Geolocation API handelt es sich um eine JavaScript API (vgl. Wenz, 2014 S. 413). Der Standort des Benutzers kann anhand verschiedener Kriterien ermittelt werden. Dazu gehört die IP-Adresse sowie der entsprechende Vermittlungsknoten des Providers, WLAN-Netze und Mobilfunksignale.

Falls vorhanden kann der Standort auch direkt über das GPS-Modul des mobilen Gerätes ermittelt werden, dass über eine sehr gute Genauigkeit verfügt (vgl. selfhtml.org, o.J.b). Die Standortinformationen des Benutzers werden in einem Positionsobjekt zur Verfügung gestellt, welches verschiedene wichtige Informationen zum Standort des Benutzers enthält. Dazu zählt bspw. die Genauigkeit der Messung (*accuracy*), Höhe über Normalnull in Metern (*altitude*), Richtungsgrad (*heading*), Breiten- und Längengrad (*lati- und longitude*) sowie die Geschwindigkeit in Metern pro Sekunde (*speed*) (vgl. Wenz, 2014 S. 414). Mit diesen Informationen kann so der Standort des Benutzers ermittelt und dargestellt werden.

Für den Zugriff auf die im Gerät verbauten Sensoren, wie Beschleunigungssensor, Gyroskop und Kompass, kann die Device Orientation API und die Device Motion Events verwendet werden (vgl. LePage, o.J.). Bei den Device Motion Events handelt es sich um JavaScript Events, die Orientierungsinformationen verarbeiten können. Diese Events werden erst dann ausgelöst, wenn der Beschleunigungssensor eine Änderung der Ausrichtung oder Bewegung erfasst. Die APIs und Events können Daten in Form von Koordinaten und der Ausrichtung anhand der X-, Y- und Z-Achse zurückgeben und dementsprechend Aufschluss über die Neigung und Ausrichtung des Gerätes geben (vgl. Mozilla Developer Network, o.J.c). Alternativ gibt es auch APIs, die den direkten Zugriff und das Auslesen der Daten der einzelnen Sensoren ermöglichen. Dabei handelt es sich um die Accelerometer- , Gyroscope- , Magnetometer- und Gravity Sensor API, die den Entwicklern zur Auswahl stehen (vgl. Adam Bar, o.J.). Diese Sensoren lassen sich bspw. dazu verwenden, um dem Benutzer bei der Orientierung und der Verwendung von GPS zu helfen. Ebenso können diese Events und APIs dazu genutzt werden, um festzustellen, ob das Gerät gedreht wurde, um dann die Darstellung entsprechend anzupassen (vgl. LePage, o.J.).

Des Weiteren gibt es APIs, die die Möglichkeit bieten, Informationen über den Akku auszulesen oder auf die im Gerät integrierte Vibration zuzugreifen. Zum Auslesen von Informationen des Akkus, steht die Battery API zur Verfügung (vgl. Mozilla Developer Network, o.J.a). Mithilfe dieser API können Informationen, wie die Restladezeit, verbliebene Akkulaufzeit sowie der Ladestatus ausgelesen werden (vgl. W3C, 2016). Der Zugriff auf die für die Vibration zuständige Hardware wird durch die Vibration API ermöglicht, sofern die entsprechende Hardware im mobilen Gerät verbaut ist. Mit dieser API können verschiedene Vibrationsmuster und Vibrationsintervalle festgelegt werden (vgl. Mozilla Developer Network, o.J.e).

Neben den beiden genannten APIs, können mit der Network Information API, Informationen über die Netzwerkverbindung abgefragt werden. Über die abgefragten Informationen kann bspw. festgestellt werden, ob das mobile Gerät mit dem WLAN oder dem Mobilfunk-Netz verbunden ist (vgl. Mozilla Developer Network, o.J.f).

Die Clipboard API bietet der Webapplikationen den Zugriff auf die Zwischenablage des mobilen Gerätes. Durch diese API kann von der Systemzwischenablage des mobilen Gerätes asynchron gelesen und auf diese geschrieben werden. Die API kann auf Befehle, wie Ausschneiden, Kopieren und Einfügen reagieren und die gewünschten Texte und Daten in der Zwischenablage speichern (vgl. Mozilla Developer Network, o.J.b).

Ebenfalls gibt es APIs, die die Nutzung von Bluetooth und NFC ermöglichen. Diese beiden APIs werden im nachfolgenden Kapitel 4.2.8 näher vorgestellt.

4.2.8 Datenaustausch und Kommunikation

Der Datenaustausch und die Kommunikation werden i. d. R. mithilfe verschiedener Technologien, wie Bluetooth oder NFC ermöglicht. Zwei neue JavaScript APIs sollen Progressive Web Apps den Austausch von Daten und die Kommunikation mit anderen mobilen Geräten ermöglichen. Um solche Funktionen in Webapplikationen umsetzten zu können, können Entwickler zukünftig auf verschiedenen APIs, wie z. B. auf die Web Bluetooth API (vgl. Web Bluetooth Community Group, 2018) und Web NFC API (vgl. W3C, 2018b) zurückgreifen. Jedoch gilt zu beachten, dass diese APIs sich noch in der Entwicklungsphase befinden und dadurch noch nicht von allen Browsern unterstützt werden.

Bei der Web Bluetooth API handelt es sich um eine Schnittstelle, die es ermöglicht, mittels Bluetooth nach Geräten zu suchen und mit diesen zu kommunizieren. Bluetooth ist ein Standard für die drahtlose Kommunikation zwischen Geräten über kurze Entfernungen (vgl. Web Bluetooth Community Group, 2018). Zukünftig soll es möglich sein, dass sich Progressive Web Apps per Bluetooth mit anderen Geräten verbinden, um diese zu steuern oder auch Daten auszutauschen (vgl. Ackermann, 2016a). Die Kommunikation erfolgt durch den Bluetooth 4 Standard und über das Generic Attribute Profil (GATT). Diese ermöglichen zwei Bluetooth Low Energy-Geräten, Daten in beide Richtungen zu übertragen (vgl. Hume et al., 2018 S. 158). Durch diese neue Bluetooth API ist es bspw. möglich, Smarthome-Geräte, Herzfrequenzmesser, Temperatursensoren etc., direkt über die Webapplikation und ohne Installation auf dem mobilen Gerät zu steuern und auszulesen (vgl. Shaked, 2016).

Eine weitere Möglichkeit zur Übertragung von Daten zwischen mobilen Geräten, bietet NFC. Diese Technologie ermöglicht die drahtlose Kommunikation und Datenübertragung zwischen zwei mobilen Geräten, die sich in unmittelbarer Nähe befinden (siehe Kapitel 3.2.8). Zukünftig soll die Kommunikation und Übertragung von Daten über die NFC-Technologie auch für Webapplikationen und somit auch für Progressive Web Apps mittels der Web NFC API zur Verfügung stehen. Zum jetzigen Zeitpunkt befindet sich die API jedoch noch in der Entwicklungsphase und ist noch nicht durch das W3C standardisiert (vgl. W3C, 2018b).

4.2.9 Auffindbarkeit durch Suchmaschinen

Progressive Web Apps sind moderne Webapplikationen, die größtenteils auf JavaScript basieren. Sie lassen sich wie herkömmliche Webseiten von Suchmaschinen vollständig indexieren und auffinden. Suchmaschinen, wie u. a. Google, haben ihre Webcrawler verbessert, sodass sie in der Lage sind, Skriptsprachen, wie JavaScript, auszulesen und dessen Inhalte zu verfolgen und zu referenzieren (vgl. Ryte, o.J.). Die Seiten einer Progressive Web App verfügen über eigene URLs und sind dadurch eindeutig adressierbar (vgl. Peters, 2017 S. 93). Sie erscheinen in den Suchergebnisseiten der Suchmaschinen und können über diese direkt aufgerufen werden (vgl. Kling, 2017).

Damit Progressive Web Apps von den Suchmaschinen indexiert und von den Webcrawlern gelesen und erfasst werden, müssen sie zahlreiche Anforderungen erfüllen. Zu den Anforderungen für die Aufnahme einer Progressive Web App durch die Suchmaschinen, gehört u. a. die Angabe eines eindeutigen Seitentitels auf jeder Seite, mithilfe des <*title*>-HTML-Tags. Eine weitere Angabe, die eine wichtige Rolle spielt, ist der Meta-Tag *description*. Mittels dieses Meta-Tags sollen die wesentlichen Inhalte und Themen der jeweiligen Seite einer Progressive Web App für die Suchmaschine zusammengefasst werden. Der Meta-Tag <*meta name="description" content="Beschreibung der Inhalte"*> wird innerhalb des <*head*>-Elements der Webapplikation eingefügt (vgl. Google Inc., o.J.o).

Des Weiteren können Inhalte und Daten anhand von Daten-Markups (Rich-Snippets) für die Suchmaschinen strukturiert werden, sodass bspw. bestimmte Informationen in den Suchergebnissen besser dargestellt werden können. Darunter zählen u. a. die Angabe von Standort, Öffnungszeiten, Auflistungen von Produkten sowie deren Preise. Auch die Angabe von Alternativtexten bei der Verwendung von Grafiken und Bildern innerhalb der Webapplikation

ist für das Ranking in den Suchmaschinen ausschlaggebend. Alternativtexte werden angezeigt, wenn Ressourcen oder Bilder nicht mehr verfügbar sind oder dargestellt werden können. Dann wird alternativ, über das „*alt*“-Attribut der definitere Alternativtext, anstelle des Bildes, angezeigt (vgl. Google Inc., o.J.o).

Neben diesen hier genannten Punkten, gibt es noch zahlreiche andere Kriterien, die für das Suchmaschinen-Ranking einer Progressive Web App eine Rolle spielen. Dazu gehören u. a. auch eine klare Navigation, deutliche und verständliche Links sowie optimierte Ressourcen und Bilder, die die Ladezeiten sowie die Performance der Webapplikation nicht beeinträchtigen und ausbremsen (vgl. ebd.). Aufgrund von modernen Technologien, wie Service Worker und Caching-Strategien (siehe Kapitel 4.2.1), haben Progressive Web Apps eine sehr gute Ladegeschwindigkeit, die sich ebenfalls positiv auf das Suchmaschinen-Ranking auswirkt (vgl. Peters, 2017 S. 93).

4.2.10 Zugangsdatenverwaltung

Progressive Web Apps können mithilfe der Credential Management API, Benutzerdaten, wie bspw. Benutzername und Passwort speichern. Die im Registrierungs- oder Anmeldeformular verwendeten Zugangsdaten, werden lokal im Browser und nicht in der Progressive Web App selbst gespeichert. Dazu wird dem Benutzer beim Registrieren oder der erstmaligen Anmeldung angeboten, die Zugangsdaten im Browser zu speichern (siehe Abbildung 22). Durch diese neue API wird dem Benutzer die zukünftige Anmeldung ohne die Eingabe von Benutzername und Passwort ermöglicht (vgl. Kitamura, o.J.). Dadurch fällt der Anmeldevorgang für den Benutzer weg. Mithilfe der Credential Management API wird der Benutzer beim Besuch der Webapplikation mit nur einem Schritt angemeldet (vgl. Ater, 2017 S. 241). Ebenfalls kann der Benutzer entscheiden, ob er direkt beim Aufruf der Progressive Web App automatisch angemeldet werden möchte. Beim Aufrufen und Betätigen der „anmelden“-Schaltfläche einer Progressive Web App bekommt der Benutzer, den bei der Registrierung verwendeten Account, über die native Account-Auswahl angezeigt. Sofern die Credential Management API imple-

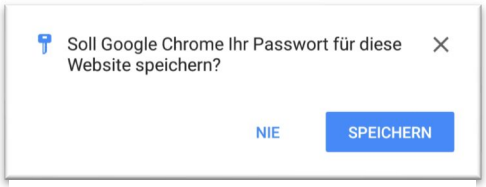

Abbildung 22 - Speicherung von Zugangsdaten - Quelle: Eigene Darstellung

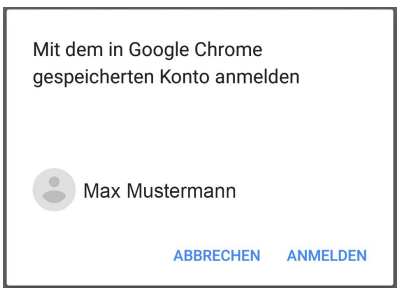

Abbildung 23 - Native Account-Auswahl - Quelle: Eigene Darstellung

mentiert wurde (siehe Abbildung 23). Dieser Account enthält den hinterlegten Benutzernamen und das Passwort. Zum Anmelden kann der Benutzer das entsprechende Konto auswählen und sich ohne die Eingabe von Benutzername und Passwort direkt anmelden (vgl. Kitamura, o.J.). Nach Ablauf der Session wird der Benutzer wieder automatisch angemeldet. So kann mithilfe dieser neuen API, die User Experience verbessert werden. Diese API hat u. a. auch den Vorteil, dass der Benutzer sich, das bei der Registrierung festgelegte Passwort, nicht merken muss (vgl. Ater, 2017 S. 241).

4.3 Plattformunterstützung

Progressive Web Apps verfügen über ein responsives Design und können dementsprechend auf unterschiedlichen Geräten, wie Smartphones und Tablet-PCs dargestellt werden (vgl. Osmani, 2015). Progressive Web Apps werden über die URL im Browser aufgerufen und ausgeführt. Dadurch können sie auf verschiedenen Betriebssystemen verwendet werden (vgl. 1und1, 2017). Zudem basieren Progressive Web Apps auf modernen HTML5 APIs, die zur Umsetzung verschiedener Funktionen, Eigenschaften und Charakteristiken verwendet werden. Damit der Benutzer diese auch nutzen kann, müssen die HTML5 APIs von den verwendeten Browsern und Betriebssystemen unterstützt werden. Um eine genaue Aussage über die Browserunterstützung der verschiedenen Funktionen und HTML5 APIs treffen zu können, wurde zur Überprüfung ein Prototyp implementiert. Der Prototyp überprüft die Browserunterstützung der in Kapitel 4.2 vorgestellten Funktionen und HTML5 APIs.

Zur Überprüfung der Browserunterstützung wurde ein Samsung Galaxy S7 mit Android 8 und ein Apple iPhone 8 mit iOS 12 verwendet. Der Prototyp wurde auf diesen Geräten in unterschiedlichen Browsern (Google Chrome, Firefox, Microsoft Edge und Apple Safari) ausgeführt und getestet. In den nachfolgenden Tabellen werden die Ergebnisse der Überprüfung aufgelistet und kurz erläutert.

Um die Plattformunterstützung einheitlich und übersichtlichen darzustellen, werden in den Tabellen folgende Symbole verwendet:

\+ - wird unterstützt.

\- - wird nicht unterstützt.

	Android			iOS			
	v. 69	v. 62	v. 42	v. 69	v. 13.2	v. 42.6	v. 12
Offline-Nutzbarkeit							
Service Worker	+	+	+	-	-	-	+
Background Synchronisation	+	-	+	-	-	-	-
Datenhaltung	+	+	+	+	+	+	+
Installierbarkeit	+	-	+	-	-	-	-
Benachrichtigungen	+	+	+	-	-	-	-
Inter-App Kommunikation	+	-	+	-	-	-	-
Zugriffskontrolle	+	+	+	-	-	-	-
Zugangsdatenverwaltung	+	+	+	-	-	-	-

Tabelle 1 - Browserunterstützung verschiedener Funktionen und APIs - Quelle: Eigene Darstellung

Die Tabelle 1 zeigt auf, dass die aufgelisteten Funktionen, wie bspw. die Offline-Nutzbarkeit, Installierbarkeit, Benachrichtigungen, bis auf wenige Ausnahmen, alle von dem Android Betriebssystem und den meisten darauf verwendeten Browsern unterstützt werden. Einzig der Firefox Browser, auf dem Android Betriebssystem, unterstützt keine Background Synchronisation, Installierbarkeit und Inter-App Kommunikation. Auf iOS-Geräten hingegen ist die Browserunterstützung der aufgelisteten Funktionen sehr gering. Ausschließlich der Safari Browser unterstützt den Service Worker und ermöglicht somit die Offline-Nutzbarkeit von Progressive Web Apps. Allerdings unterstützt dieser keine Background Synchronisation bei fehlender Internetverbindung. Nur die Datenhaltung mittels IndexedDB wird auf iOS-Geräten von allen Browsern unterstützt.

	Android			iOS			
	v. 69	v. 62	v. 42	v. 69	v. 13.2	v. 42.6	v. 12
Geräte-Zugriff							
Kamera und Mikrofon	+	-	+	-	-	-	+
Dateizugriff	+	+	+	+	+	+	+
GPS/Geolokalisierung	+	+	+	+	+	+	+
Geräteausrichtung und Bewegung	+	+	+	+	+	+	+
Akkuinformationen	+	-	+	-	-	-	-
Netzwerkinformationen	+	+	+	-	-	-	-
Vibration	+	+	+	-	-	-	-
Zwischenablage	+	+	+	+	+	+	+

Tabelle 2 - Browserunterstützung - Geräte-Zugriff - Quelle: Eigene Darstellung

Die Ergebnisse der Browserunterstützung für den Zugriff auf die verschiedenen Gerätefunktionen und -informationen werden in Tabelle 2 aufgelistet. Wie die Ergebnisse in der Tabelle 2 zeigen, wird der Zugriff auf die verschiedenen Gerätefunktionen von dem Android Betriebssystem und den meisten Browsern unterstützt. Nur der Firefox Browser unterstützt die getUserMedia- und Battery API nicht und bietet somit keinen Zugriff auf die Kamera, das Mikrofon und die Akkuinformationen des mobilen Gerätes. Auf iOS-Geräten wird der Zugriff auf die Kamera und das Mikrofon nur von dem Safari Browser unterstützt. Zudem unterstützt keiner der verwendeten Browser den Zugriff auf die Vibrations-Komponente, Akku- und Netzwerkinformationen des mobilen Gerätes. Alle anderen aufgelisteten Funktionen werden, wie in der Tabelle 2 abgebildet, von dem iOS Betriebssystem und den verwendeten Browsern unterstützt.

	Android			**iOS**			
	v. 69	v. 62	v. 42	v. 69	v. 13.2	v. 42.6	v. 12
Sensoren							
Beschleunigungssensor	+	-	+	-	-	-	-
Gyroskop	+	-	+	-	-	-	-
Magnetometer	-	-	-	-	-	-	-
Gravitationsmesser	-	-	-	-	-	-	-

Tabelle 3 - Browserunterstützung - Sensor-Zugriff - Quelle: Eigene Darstellung

Tabelle 3 stellt die Ergebnisse der Browserunterstützung für den Zugriff auf die verschiedenen, im Gerät verbauten Sensoren, dar. Die Ergebnisse zeigen deutlich, dass der Zugriff auf die verschiedenen im Gerät verbauten Sensoren kaum von den Browsern und Betriebssystemen unterstützt wird. Lediglich der Chrome und der Edge Browser auf dem Android Betriebssystem bieten den Zugriff auf den Beschleunigungssensor und das Gyroskop. Das iOS Betriebssystem und die darauf verwendeten Browser unterstützten keinen Zugriff auf die Gerätesensoren.

	Android			**iOS**			
	v. 69	v. 62	v. 42	v. 69	v. 13.2	v. 42.6	v. 12
Datenaustausch und Kommunikation							
Bluetooth	+	-	+	-	-	-	-
NFC	-	-	-	-	-	-	-

Tabelle 4 - Browserunterstützung - Datenaustausch und Kommunikation - Quelle: Eigene Darstellung

Abschließend werden in der Tabelle 4 die Ergebnisse zur Browserunterstützung der Bluetooth- und NFC-Technologie, die den drahtlosen Datenaustausch und die Kommunikation zwischen anderen Geräten ermöglichen, aufgeführt. Die Nutzung der Bluetooth-Technologie, über die Web Bluetooth API, wird nur von dem Android Betriebssystem sowie von dem Chrome- und Edge Browser unterstützt. Die NFC-Technologie und die dazugehörige Web NFC API wird zurzeit von keinem Betriebssystem und Browser unterstützt.

Insgesamt machen die Ergebnisse der Überprüfung deutlich, dass die Browserunterstützung von Progressive Web Apps auf dem Android Betriebssystem und den darauf verwendeten Browsern, wie Chrome, Firefox und Edge, sehr weit fortgeschritten ist und die meisten Funktionen und APIs unterstützt werden. Auf iOS-Geräten hingegen können nicht alle Funktionen und Eigenschaften einer Progressive Web App verwendet werden, da die Browserunterstützung der verschiedenen Funktionen und APIs fehlt.

4.4 Look and Feel

Progressive Web Apps können über Eigenschaften und ein ähnliches Look and Feel verfügen, wie native Apps. Ermöglicht werden kann dies mit verschiedenen modernen Webtechnologien. Zu den app-ähnlichen Eigenschaften einer Progressive Web App, die für ein natives Look and Feel sorgen, gehört die Installierbarkeit auf dem Homescreen, die Offline-Nutzbarkeit, der Empfang von Push Notifications sowie die Verwendung eines responsiven Designs und der Application-Shell-Architektur. Um ein natives Look and Feel zu realisieren, werden neben diesen Eigenschaften auch Empfehlungen und die Material Design Lite Bibliothek zur Gestaltung einer Progressive Web App angeboten.

Wie in Kapitel 4.2.2 erläutert, lassen sich Progressive Web Apps auf dem Homescreen eines mobilen Gerätes installieren (vgl. Mozilla Developer Network, 2018a). Dabei wird, wie bei einer nativen App, ein Icon der Progressive Web App auf dem Homescreen des mobilen Gerätes hinterlegt. Durch das Definieren bestimmter Parameter in der Web App Manifest-Datei können Eigenschaften festgelegt werden, die für ein natives Look and Feel einer Progressive Web App sorgen. So ist es über Parameter u. a. möglich, einen Begrüßungsbildschirm (Splash-Screen) zu erstellen oder die Progressive Web App im Vollbildmodus, d. h. ohne Darstellung des Browsers und dessen Bedienelemente, ausführen zu lassen. Dadurch wird die Progressive Web App, wie eine native App, dargestellt. Des Weiteren kann in der Web App Manifest-Datei die Bildschirmausrichtung (Hoch- oder Querformat) sowie die

Farben der Benutzeroberfläche festgelegt werden (vgl. W3C, 2018a). Die Progressive Web App wird, wie eine native App, im App-Launcher und in den Einstellungen des Betriebssystems angezeigt (vgl. LePage, 2018a).

Eine weitere Eigenschaft und Funktion, die für ein natives Look and Feel einer Progressive Web App sorgt, ist die Offline-Nutzbarkeit (siehe Kapitel 4.2.1). Der Benutzer kann die Progressive Web App unabhängig von der Internetverbindung verwenden und wird bei schlechter oder keiner Internetverbindung nicht unterbrochen (vgl. Springer, 2016 S. 121). Moderne Technologien, wie Service Worker , Background Sync und verschiedene Caching-Strategien (vgl. Braun, 2017 S. 108) ermöglichen so, dass der Benutzer bei der Verwendung einer Progressive Web App nicht bemerkt, dass er offline oder die Internetverbindung schlecht ist (vgl. Hume et al., 2018 S. 120). Eine weitere app-ähnliche Funktion ist der Empfang von Push Notifications (siehe Kapitel 4.2.3). Push Notifications waren bisher nur nativen Apps vorbehalten. Mittlerweile können Progressive Web Apps auch Push Notifications empfangen und darstellen (vgl. Steyer, 2017a S. 40). Die Benachrichtigungen werden, wie bei nativen Apps, auf dem Sperrbildschirm oder in der Notification-Bar des mobilen Gerätes angezeigt. Dadurch kommen Progressive Web Apps dem nativen Look and Feel immer näher.

Progressive Web Apps können durch die Verwendung eines responsiven Designs an unterschiedliche Geräte, wie Smartphones und Tablet-PCs angepasst werden (vgl. Pete LePage, o.J.). Durch die Verwendung der Application-Shell-Architektur, werden Progressive Web Apps sofort und zuverlässig auf dem Bildschirm des mobilen Gerätes dargestellt. Die Application-Shell-Architektur trennt die Benutzeroberfläche (App Shell) von den Inhalten der Progressive Web App. Die Benutzeroberfläche, die das Grundgerüst für die Progressive Web App und die Inhalte bildet (vgl. East, 2015), kann mithilfe von Service Workern in den Cache abgelegt werden. Dadurch kann die Benutzeroberfläche sofort und unabhängig von der Internetverbindung und ohne nachladen, dem Benutzer angezeigt werden (vgl. Osmani, o.J.). Die sofortige Darstellung der Benutzeroberfläche und Anpassung an das mobile Gerät, bietet dem Benutzer ein app-ähnliches Look and Feel.

4.4.1 Style-Guidelines und Empfehlungen

Damit Progressive Web Apps über ein app-ähnliches Look and Feel verfügen können, sollten bestimmte Eigenschaften und Empfehlungen zur Gestaltung einer Progressive Web App

eingehalten werden. Zudem sollten bei der Gestaltung der Benutzeroberfläche, die plattformspezifischen Style-Guidelines von nativen Apps berücksichtigt werden (vgl. Campbell-Moore, 2016). Die plattformspezifischen Style-Guidelines für Android und iOS, werden in Kapitel 3.4 kurz vorgestellt. Ein Style-Guide speziell für Progressive Web Apps existiert nicht.

Der Google Mitarbeiter Owen Campbell-Moore hat jedoch einen Artikel mit Empfehlungen zur Gestaltung von Benutzeroberflächen einer Progressive Web App veröffentlicht, die ein natives Look and Feel ermöglichen. Dabei sollte die Benutzeroberfläche einer Progressive Web App beim Betätigen von Schaltflächen oder Links umgehend geladen und dargestellt werden. Links sollten kaum verwendet werden, stattdessen sollten Schaltflächen zum Einsatz kommen. Des Weiteren sollte bei der Gestaltung der Benutzeroberfläche darauf geachtet werden, dass der Benutzer bei Betätigen von Schaltflächen eine Bestätigung durch bestimmte Effekte bekommt. Dies kann durch visuelle Effekte, wie Animationen, Übergänge oder Umfärbung der betätigten Schaltflächen ermöglicht werden. Zudem sollte beim Scrollen durch Menüs verhindert werden, dass versehentlich Schaltflächen betätigt werden können. Inhalte und Schalflächen einer Progressive Web App sollten die plattformspezifischen Schriftarten verwenden, die dem Benutzer bereits bekannt sind. Die Benutzeroberfläche und Inhalte einer Progressive Web App sollten während des Ladevorgangs nicht springen. Dazu können für Bilder Platzhalter (graue Felder) oder die Dimensionen des Bildes im <*img*>-Tag definiert werden, sodass die Webapplikation, bevor die Bilder geladen werden, sofort richtig dargestellt wird. Elemente der Benutzeroberfläche, die nicht zu den Inhalten gehören, sollten nicht auswählbar oder markierbar sein. Dazu können Schalflächen oder Elemente der Benutzeroberfläche mittels CSS (*user-select: none;*) von der Auswahl ausgeschlossen werden. Ferner sollte beachtet werden, dass betätigte Eingabefelder nicht durch die Tastatur verdeckt werden. Um dies zu verhindern, können die Eingabeelemente mittels Element.scrollIntoView() ins Sichtfeld gescrollt werden (vgl. ebd.).

4.4.2 Material Design Lite

Material Design Lite ist eine Bibliothek, die die Entwicklung der Benutzeroberfläche einer Progressive Web App mit Material Design-Komponenten ermöglicht (vgl. getmdl.io, o.J.). Die Material Design Lite Bibliothek bietet die Möglichkeit, die Benutzeroberfläche einer Progressive Web App so zu gestalten, sodass sie sich kaum von nativen Android Apps un-

terscheiden lässt. Die Bibliothek kann jedoch auch zur Entwicklung und Gestaltung von Benutzeroberflächen einer Progressive Web App, speziell für das iOS-Betriebssystem, verwendet werden. Grund dafür ist, dass die Material Design-Komponenten für eine plattformunabhängige Verwendung vorgesehen sind. Die Bibliothek besteht aus CSS- und JavaScript-Dateien, die sich über das <*link*>-Element in die Webapplikation einbinden lassen. Über vordefinierte CSS-Klassen können den HTML-Elementen der Progressive Web App die verschiedenen Material Design Lite-Komponenten zugewiesen werden. Beim Aufrufen der Webapplikation werden dann die gewünschten Komponenten im Material Design dargestellt (vgl. getmdl.io, o.J.). Die Material Design Lite Bibliothek bietet dem Entwickler zahlreiche Komponenten, die zur Gestaltung einer app-ähnlichen Benutzeroberfläche einer Progressive Web App verwendet werden können. Dazu zählen u. a. verschiedene Buttons, Tabellen, Listen, Menüs, Slider, Formularelemente und Ladeelemente sowie Layouts (vgl. ebd.). Die nachfolgende Abbildung 24 stellt einige der verfügbaren Material Design Lite-Komponenten grafisch dar.

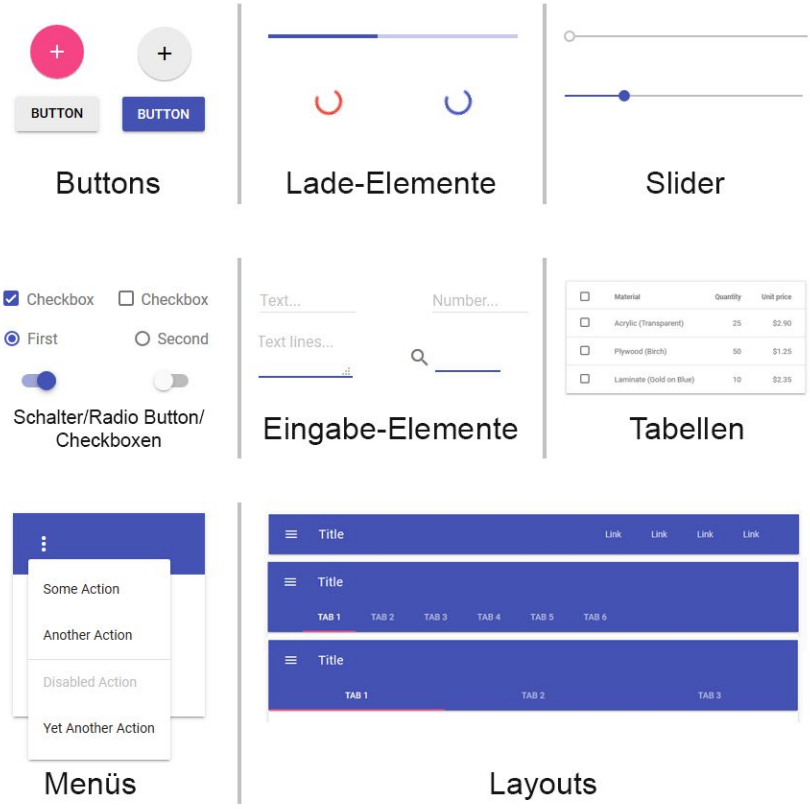

Abbildung 24 - Material Design Lite Komponenten - Quelle: Eigene Darstellung

Wie zu Beginn kurz erläutert, kann die Material Design Lite Bibliothek auch zur grafischen Gestaltung von Benutzeroberflächen für iOS-Geräte verwendet werden. Wie in Kapitel 3.4.2 dargelegt wird, unterscheidet sich die Benutzeroberfläche einer iOS- und Android App in der Anordnung und in der grafischen Darstellung der verschiedenen Komponenten, wie bspw. der Navigations- und Tab Bar. Diese Komponenten und deren Anordnung lassen sich mittels CSS an das iOS-Layout anpassen. Auch die Darstellung und Positionierung des „zurück"-Button in der Navigations-Bar (oben), lässt sich zur Bedienung der Progressive Web App auf iOS-Geräten umsetzen. Um plattformspezifische Layouts und Anpassungen darzustellen, können Media Queries verwendet werden. Denn die Material Design Lite Bibliothek bietet für iOS keine speziell angepassten und vorgefertigten Komponenten an.

4.5 Veröffentlichung und Verbreitung

Progressive Web Apps werden im Browser ausgeführt und sind dementsprechend plattformübergreifend nutzbar (vgl. Liebel, 2017). Zur Veröffentlichung wird ein Webspace oder ein entsprechender Webserver benötigt, die von verschiedenen Hosting-Anbietern angeboten werden. Damit Progressive Web Apps funktionieren und ausgeführt werden können, wird ein SSL-Zertifikat zur Verschlüsselung der Verbindung mittels HTTPS benötigt. Zur Veröffentlichung einer Progressive Web App können Entwickler auch die App- und Web-Plattform Firebase von Google verwenden. Firebase bietet die Möglichkeit, die Progressive Web App zu hosten. Diese ist anschließend über eine vorgegebene URL erreichbar und kann darüber verwendet werden (vgl. Peters, 2017 S. 89 f.).

Für Progressive Web Apps gibt es keinen App Store, über den alle Progressive Web Apps an zentraler Stelle zur Verfügung gestellt werden. Dadurch ist es schwieriger neue Benutzer auf die Progressive Web App aufmerksam zu machen (vgl. Fern, 2016). Progressive Web Apps lassen sich auch ohne die Verwendung eines plattformspezifischen App Stores auf dem mobilen Gerät des Benutzers installieren (vgl. Firtman, 2018). Damit bleibt dem Benutzer der umständliche Installationsprozess über den App Store erspart (vgl. Fern, 2016). Microsoft macht jedoch eine Ausnahme. Microsoft bietet die Möglichkeit, Progressive Web Apps über den Microsoft Store anzubieten und zu installieren. Die Progressive Web App muss dazu einige Kriterien erfüllen. Demnach muss eine Web App Manifest-Datei, ein Service Worker sowie eine mittels HTTPS verschlüsselte Verbindung existieren. Zudem darf die angebotene Progressive Web App nicht gegen die geltenden Richtlinien verstoßen. Die

Progressive Web App kann manuell über das Visual Studio und Windows Dev Center oder auch automatisch durch den Bing-Webcrawler in den Microsoft Store hinzugefügt werden (vgl. Navara et al., 2018). Die anderen plattformspezifischen App Stores bieten keine Möglichkeiten zur Veröffentlichung von Progressive Web Apps an (vgl. Fern, 2016).

Da Progressive Web Apps i. d. R. nicht über App Stores veröffentlicht und verbreitet werden, müssen diese vor der Veröffentlichung auch nicht den umfangreichen Freigabeprozess durch den App Store durchlaufen. Die großen App Store Anbieter, wie Apple und Google haben somit keinen Einfluss auf die Webapplikationen. Da Progressive Web Apps keinen Freigabe- und Überprüfungsprozess der App Store-Anbieter durchlaufen (vgl. Dickehut, 2018), ist es für den Benutzer schwieriger festzustellen, ob eine Progressive Web App vertrauenswürdig ist oder nicht (vgl. Fern, 2016). Wie in Kapitel 4.2.9 beschrieben, lassen sich Progressive Web Apps jedoch, wie herkömmliche Webapplikationen und Webseiten, über die Suchmaschine auffinden. Zudem können sie per Verlinkung über das Internet verbreitet werden (vgl. Pete LePage, o.J.).

5 Implementierung

In diesem Kapitel werden die für die Untersuchung und den Vergleich implementierten Prototypen vorgestellt. Implementiert wurde ein Prototyp einer Progressive Web App sowie ein Prototyp einer nativen Android App.

5.1 Progressive Web App-Prototyp

Der Prototyp der Progressive Web App wurde mit modernen Webtechnologien, wie HTML5, CSS3 und JavaScript entwickelt und enthält die wesentlichen Funktionen und Eigenschaften einer Progressive Web App. Dieser Prototyp dient zur Überprüfung der Plattformunterstützung und Funktionalität von Progressive Web Apps. Dazu wurde prototypisch eine einfache News App als Progressive Web App umgesetzt. Für die Überprüfung der Plattformunterstützung und den Vergleich, sind Funktionen und Eigenschaften, wie die Offline-Nutzbarkeit, Installierbarkeit, Funktionen für den Empfang von Push Notifications sowie ein app-ähnliches und responsives Design implementiert worden. Am Beispiel einer News App werden diese Funktionen und Eigenschaften demonstriert. Der Prototyp verfügt über eine app-ähnliche Benutzeroberfläche, die auf der Material Design Lite Bibliothek basiert und

die Material Design-Komponenten verwendet (siehe Abbildung 25). Die Benutzeroberfläche des Prototyps bietet über die Navigation die Möglichkeit, innerhalb der App zu navigieren (siehe Abbildung 26). Über die Navigation können andere Seiten des Prototyps aufgerufen werden, um z. B. Beispielartikel aus einer anderen Kategorie abzurufen und anzuzeigen. Die Beispielartikel der News App werden von einem Server abgerufen und in einer Liste dargestellt (siehe Abbildung 25).

Abbildung 25 - Progressive Web App-Prototyp - Artikelübersicht - Quelle: Eigene Darstellung

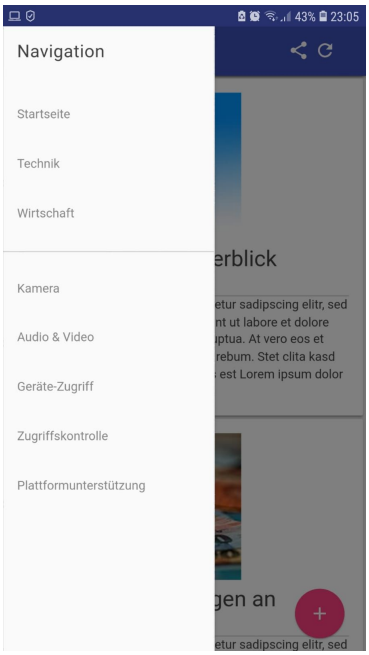

Abbildung 26 - Progressive Web App-Prototyp - Navigation - Quelle: Eigene Darstellung

Um die Offline-Nutzbarkeit des Prototyps umzusetzen und überprüfen zu können, wurde ein Service Worker und die Background Sync API implementiert. Zur Überprüfung der von der Interverbindung unabhängigen Übertragung von Daten über die Background Sync API, wurde eine prototypische Funktion zum Einsenden von Artikeln implementiert (siehe Abbildung 27). Diese Funktion kann über den Floating Action Button aufgerufen werden. Des Weiteren wurde die Installierbarkeit des Prototyps mithilfe der Web App Manifest-Datei und den für die Installation notwendigen Parametern (siehe Kapitel 4.2.2) umgesetzt. Außerdem wurden prototypisch die Funktionen für den Empfang und die Darstellung von Push Notifications mit der Push- und Notification API sowie die Inter-App Kommunikation implementiert. Zusätzlich wurden prototypisch der Dateizugriff und die Zugriffskontrolle mit der File- und Permissions API implementiert. Neben den genannten Funktionen und Eigenschaften, wurde ebenfalls prototypisch der Geräte-Zugriff auf Komponenten, wie Kamera,

Mikrofon, GPS und die Sensoren umgesetzt. Diese Funktionen wurden in den Prototyp der Progressive Web App eingebunden, sind jedoch nicht miteinander verknüpft. Diese Funktionen sind als eigenständig anzusehen und wurden implementiert, um die Plattformunterstützung und Funktionalität überprüfen zu können.

Um eine genaue Aussage über die Browser- und Betriebssystemunterstützung von Progressive Web Apps geben zu können, wurde eine Funktion implementiert, die den verwendeten Browser des mobilen Gerätes auf die Unterstützung der verschiedenen APIs überprüft und die Ergebnisse darstellt. Diese Funktion wurde zu dem Prototyp der Progressive Web App hinzugefügt und liefert beim Aufruf über die Navigation, eine Übersicht über die Browserunterstützung der verschiedenen APIs und Funktionen. Die Ergebnisse der Überprüfung beziehen sich immer auf den ausführenden und verwendeten Browser des mobilen Gerätes (siehe Abbildung 28). Mithilfe dieses Prototyps konnten die in Kapitel 4.3 aufgeführten Ergebnisse über die Plattformunterstützung von Progressive Web Apps ermittelt werden. Zudem konnte mit diesem Prototyp das Look and Feel einer Progressive Web App, mit dem im folgenden Kapitel vorgestellten Prototyp einer nativen Android App vergleichbar gemacht werden.

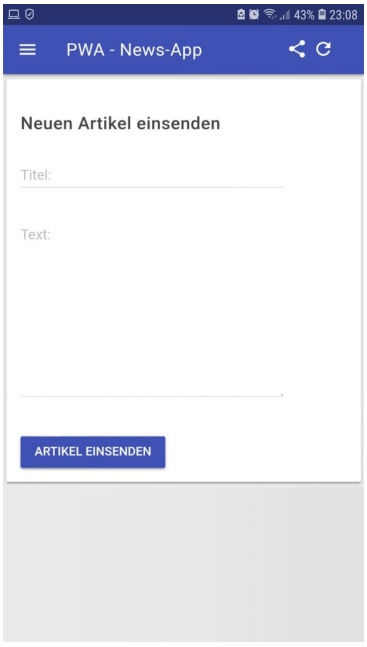

Abbildung 27 - Progressive Web App-Prototyp - Artikel einsenden Formular - Quelle: Eigene Darstellung

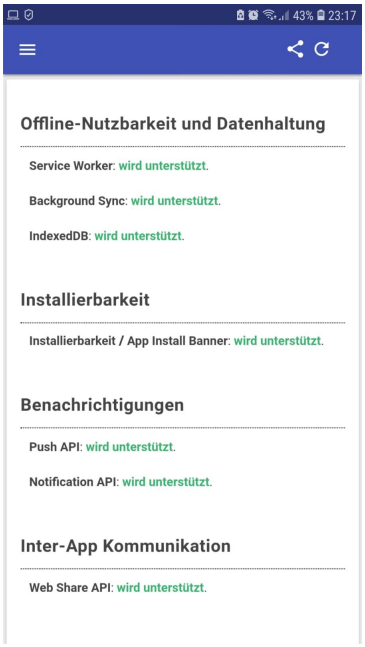

Abbildung 28 - Funktion zum Überprüfen der Plattformunterstützung - Quelle: Eigene Darstellung

5.2 Native Android App-Prototyp

Der Prototyp der nativen Android App wurde entwickelt, um das Look and Feel mit dem Prototyp der Progressive Web App vergleichbar machen zu können. Dazu wurde nur die Benutzeroberfläche umgesetzt. Funktionen, wie bspw. Benachrichtigungen oder Geräte-Zugriffe wurden nicht implementiert, da zur Eingrenzung des Umfangs angenommen wird, dass diese Funktionen und Eigenschaften von nativen Android Apps unterstützt und angeboten werden. Entwickelt wurde der Prototyp mit der Programmiersprache Java. Zudem wurde zur Umsetzung der Benutzeroberfläche der nativen Android App das Material Design verwendet (siehe Abbildung 29). Um das Look and Feel dieser beiden Prototypen vergleichbar machen zu können, wurden die in Kapitel 3.4.1 vorgestellten Material Design Komponenten zur Gestaltung der Benutzeroberfläche verwendet. Zudem wurden die Benutzeroberflächen der beiden Prototypen ähnlich gestaltet, um einen Vergleich ermöglichen zu können. Die Benutzeroberfläche des Prototyps stellt in der Mitte Beispielartikel dar und bietet zudem über den Floating Action Button die Möglichkeit, das Formular zum Einsenden von neuen Beiträgen aufzurufen (siehe Abbildung 31). Außerdem wird über die Navigation die Möglichkeit geboten, innerhalb der App zu navigieren und Beispielartikel anderer Kategorien aufzurufen (siehe Abbildung 30). Der Prototyp wird durch das Ausführen der erstellten APK-Datei auf dem Gerät installiert und kann über das bei der Installation auf dem Homescreen abgelegte Icon ausgeführt werden.

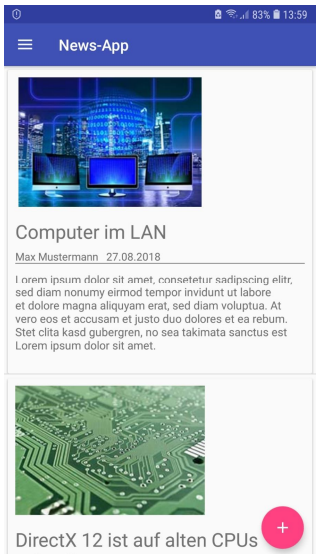

Abbildung 29 - Native Android App-Prototyp - Artikelübersicht - Quelle: Eigene Darstellung

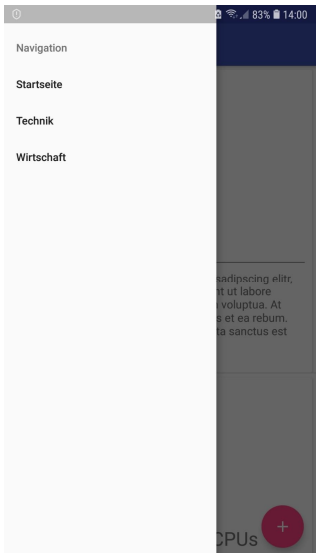

Abbildung 30 - Native Android App-Prototyp - Navigation - Quelle: Eigene Darstellung

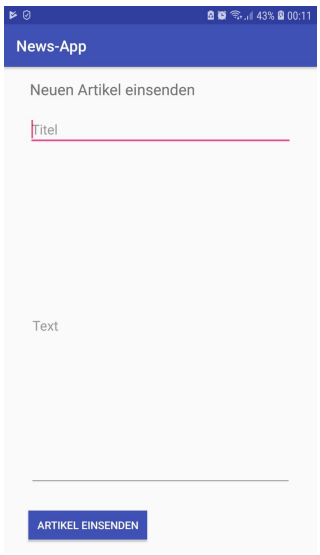

Abbildung 31 - Native Android App-Prototyp - Artikel einsenden Formular - Quelle: Eigene Darstellung

6 Vergleich von Progressive Web Apps und nativen Android Apps

In diesem Kapitel werden die beiden App-Typen Progressive Web Apps und native Android Apps gegenübergestellt und miteinander verglichen. Der Vergleich basiert auf Grundlage der Ergebnisse aus der durchgeführten Literaturrecherche in Kapitel 3 und 4 sowie den Erkenntnissen aus der Implementierung der Prototypen. Verglichen wird die Entwicklung, die technischen und funktionalen Möglichkeiten, die Plattformunterstützung, das Look and Feel sowie die Veröffentlichung und Verbreitung der beiden App-Typen.

Um einen einheitlichen und übersichtlichen Vergleich zu ermöglichen, werden in den Tabellen folgende Symbole verwendet:

✓ - wird unterstützt.

○ - wird teilweise unterstützt.

✗ - wird nicht unterstützt.

Entwicklung:

	Native Android Apps	Progressive Web Apps
Programmiersprachen	Java	HTML5, CSS und JavaScript
Software Development Kit (SDK)	Android SDK	-
Integrierte Entwicklungsumgebung (IDE)	Android Studio	Einfache Code-Editoren

Tabelle 5 - Vergleich der Entwicklung von Progressive Web Apps u. nativen Android Apps - Quelle: Eigene Darstellung

Native Android Apps werden mit der objektorientierten Programmiersprache Java entwickelt und benötigen dazu das Android SDK, welches alle notwendigen Bibliotheken und Tools zur Entwicklung bereitstellt (vgl. Künneth, 2018 S. 19). Zudem ist die integrierte Entwicklungsumgebung Android Studio erforderlich (vgl. Google Inc., o.J.i). Die in Java entwickelten nativen Android Apps lassen sich nur auf dem Android Betriebssystem ausführen und nutzen. Eine plattformübergreifende Nutzung von nativen Android Apps ist daher nicht möglich (vgl. Android Developers, o.J.n). Progressive Web Apps hingegen, werden mit modernen Webtechnologien, wie HTML, CSS und JavaScript entwickelt und im Browser ausgeführt (vgl. Osmani, 2015). Aufgrund dessen müssen die Entwickler keine komplexen plattformspezifischen Programmiersprachen erlernen. Auch sind Progressive Web Apps auf verschiedenen Plattformen, wie Android oder iOS, nutzbar und somit plattformunabhängig.

Anders als bei der Entwicklung von nativen Android Apps, wird auch keine spezielle integrierte Entwicklungsumgebung oder ein SDK benötigt. Zur Entwicklung von Progressive Web Apps eignen sich einfache Code-Editoren. Zum Debuggen werden Tools, wie die Chrome DevTools und Lighthouse zur Verfügung gestellt (siehe Kapitel 4.1.2).

Technische und funktionale Möglichkeiten:

	Native Android Apps	Progressive Web Apps
Offline-Nutzbarkeit	✓	✓
Datenhaltung	✓	✓
Installierbarkeit	✓	✓
Benachrichtigungen	✓	✓
Inter-App Kommunikation	✓	○
Zugriffskontrolle	✓	✓
Multimedia	✓	✓
Auffindbarkeit über Suchmaschinen	✗	✓
Zugangsdatenverwaltung	✓	✓

Tabelle 6 - Vergleich der technischen und funktionalen Möglichkeiten - Quelle: Eigene Darstellung

Native Android Apps und auch Progressive Web Apps können unabhängig von der Internetverbindung verwendet werden. Bei nativen Android Apps werden alle für die Nutzung und Darstellung relevanten Daten und Dateien bereits während der Installation auf dem Gerät gespeichert (vgl. Staudemeyer, 2018 S. 56). Bei Progressive Web Apps hingegen ermöglicht ein Service Worker die Offline-Nutzbarkeit. Dieser speichert die Daten und Dateien beim Aufruf der Seite im Cache ab und stellt diese bei eingeschränkter Internetverbindung zur Verfügung (vgl. Springer, 2016 S. 121). Über das Android Sync Adapter Framework (vgl. Android Developers, o.J.ae) und die Background Sync API (vgl. Hume et al., 2018 S. 120 ff.), können native Android Apps und Progressive Web Apps unabhängig von der Internetverbindung Daten an den Server übertragen. Bei wiederkehrender und stabiler Internetverbindung ermöglichen diese Technologien die Datenübertragung und Synchronisation mit dem Server. Ferner können native Android Apps, als auch Progressive Web Apps, Daten lokal speichern. Die Speicherung von Daten ist unter nativen Android Apps über SharedPreferences oder SQLite möglich (vgl. Android Developers, o.J.j). Progressive Web Apps können Daten über die IndexedDB API im Browser speichern (vgl. Mozilla Developer Network, 2013b). Die Technologien zur Datenhaltung können sowohl bei nativen Android Apps, als auch bei Progressive Web Apps, zur Zwischenspeicherung von Anfragen und Daten bei fehlender Internetverbindung verwendet werden.

Native Android Apps als auch Progressive Web Apps können auf mobilen Geräten installiert werden. Die Installation unterscheidet sich allerdings voneinander. Während native Android Apps über einen App Store oder direkt durch das Ausführen einer APK-Datei installiert werden können (vgl. Staudemeyer, 2018 S. 56), werden Progressive Web Apps beim Aufruf und durch Betätigen des im Browser angezeigten App Install Banners auf dem Homescreen installiert (vgl. Springer, 2016 S. 123). Für die Installation von nativen Android Apps wird eine APK-Datei benötigt, die alle für die Ausführung und Darstellung benötigten Dateien und Informationen enthält. Progressive Web Apps benötigen zur Installation allerdings eine Web App Manifest-Datei, die alle Informationen und Parameter über die App und zur Darstellung beinhaltet (vgl. Staudemeyer, 2018 S. 56). Bei beiden App-Typen werden Icons auf dem Homescreen abgelegt und sind dementsprechend schneller aufrufbar und erscheinen ebenfalls im App-Drawer und in den Einstellungen.

Der Empfang von Push Notifications war bisher nur nativen Apps vorbehalten (vgl. Steyer, 2017a S. 40). Dazu gehört auch das Empfangen und Darstellen von Push Notifications, wenn die native App im Hintergrund ausgeführt wird oder gar geschlossen ist (vgl. Becker et al., 2015 S. 241 f.). Progressive Web Apps bieten nun durch die Verwendung neuer und moderner APIs (Push- und Notification API), ebenfalls die Möglichkeit, Push Notifications zu empfangen und auf dem mobilen Gerät darzustellen (vgl. Medley, o.J.). Ebenso können Push Notifications empfangen und dargestellt werden, wenn die Progressive Web App im Hintergrund ausgeführt wird oder ganz geschlossen ist (vgl. Mozilla Developer Network, o.J.g). Die Anzeige und Darstellung neuer Push Notifications erfolgt sowohl bei nativen Android Apps, als auch bei Progressive Web Apps, auf dem Sperrbildschirm und in der Status-Bar.

Die Inter-App Kommunikation wird bei nativen Android Apps durch die Nutzung von *Intents* ermöglicht. Dadurch können Daten an andere auf dem Gerät installierte Apps gesendet und auch Daten von anderen Apps empfangen werden (vgl. Android Developers, o.J.o). Progressive Web Apps hingegen können Inhalte über die Web Share API an andere auf dem Gerät installierte Apps senden (vgl. Ater, 2017 S. 245). Sowohl bei nativen Android Apps, als auch bei Progressive Web Apps, bekommt der Benutzer den nativen Teilen-Dialog angezeigt und kann entscheiden, an welche App die ausgewählten Daten gesendet werden sollen. Der Empfang von gesendet Daten und Inhalten anderer Apps über die Web Share Target API ist zurzeit noch nicht möglich, da diese API noch nicht endgültig spezifiziert wurde und von keinem Browser zur Verfügung gestellt wird (vgl. Ackermann, 2016b). Aus diesem

Grund wird die Inter-App Kommunikation von Progressive Web Apps zum jetzigen Zeitpunkt nur teilweise unterstützt.

Native Android Apps und Progressive Web Apps bieten die Möglichkeit, sensible Daten vor unerlaubte Zugriffe zu schützen. Dementsprechend wird für die Nutzung und den Zugriff auf bestimmte Funktionen und Systemkomponenten die Berechtigung des Benutzers benötigt. Bei nativen Android Apps bekommt der Benutzer zu Beginn der Installation, die für die Nutzung erforderlichen Berechtigungen dargestellt und kann entscheiden, ob er mit diesen einverstanden ist. Jedoch kann er nicht einzelne Zugriffsberechtigungen verbieten und andere wiederrum erlauben (vgl. Google Inc., o.J.a). Bei Progressive Web Apps wird die Zugriffskontrolle über die Permissions API ermöglicht (vgl. Mozilla Developer Network, 2013c). Diese API bietet die Möglichkeit, die Zugriffsberechtigung für verschiedene Funktionen einzuholen (vgl. Matt Gaunt, o.J.). Anders als bei nativen Android Apps, wird die Zugriffsberechtigung nicht schon bei der Installation abgefragt, sondern erst wenn eine Funktion (wie bspw. die Kamera) aufgerufen wird. Der Benutzer bekommt dabei ein Pop-up-Hinweis im Browser dargestellt. Ferner kann er frei entscheiden, für welche Funktionen er die Erlaubnis erteilen möchte und für welche nicht, ohne ganz auf die Progressive Web App verzichten zu müssen.

Eine weitere Möglichkeit, die Progressive Web Apps und native Android Apps bieten, ist das Einbinden und Wiedergeben von Audio- und Video-Medien. Beide App-Typen unterstützten die am meisten genutzten Multimedia-Formate, die zur Einbindung und Wiedergabe von Audio- und Video-Medien verwendet werden können (siehe Kapitel 3.2.6 und 4.2.6). Zudem bietet die gleichnamige Media Session API sowohl für Progressive Web Apps, als auch für native Android Apps die Möglichkeit, dem Benutzer Informationen über das aktuell wiedergegebene Audio- oder Video-Medium auf dem Sperrbildschirm und in der Notifications-Bar anzuzeigen und darüber zu steuern.

Progressive Web Apps werden, anders als native Android Apps, wie herkömmliche Webseiten von Suchmaschinen indexiert und können auch über diese gefunden werden. Dies liegt daran, dass alle Inhalte einer Progressive Web App über eine URL erreichbar sind (vgl. Peters, 2017 S. 93). Diese Eigenschaft verfügen native Android Apps nicht. Denn native Android Apps können nur über den App Store gefunden und installiert werden. Zudem haben Suchmaschinen keinen Zugriff auf die Inhalte der nativen Android App und können diese nicht indexieren, da sie auf dem mobilen Gerät installiert werden und nicht über eine

URL aufgerufen werden können. In diesem Punkt haben Progressive Web Apps einen Vorteil gegenüber nativen Android Apps.

Eine weitere Funktion, die Progressive Web Apps und native Android Apps bieten, ist die Speicherung von Registrierungs- und Anmeldedaten eines Benutzers. Progressive Web Apps können die Zugangsdaten mithilfe der Credential Management API speichern. Dadurch kann der Anmeldeprozess bei der Nutzung von Progressive Web Apps deutlich vereinfacht werden und der Benutzer muss keine Zugangsdaten zum Anmelden eingeben. Die Zugangsdaten werden dabei nicht in der Progressive Web App selbst, sondern im Browser gespeichert. Der Benutzer wählt lediglich das gewünschte Konto für die Anmeldung aus (vgl. Kitamura, o.J.). Außerdem kann er, wenn gewünscht, dauerhaft angemeldet bleiben (vgl. Ater, 2017 S. 241). Native Android Apps bieten ebenfalls die Möglichkeit, die Zugangsdaten des Benutzers durch die Nutzung von SharedPreferences zu speichern. Anders als Progressive Web Apps, werden die Zugangsdaten innerhalb der App gespeichert. Durch die Verwendung der SharedPreferences, bleibt der Benutzer auch dann angemeldet, wenn die Session abgelaufen ist oder die App geschlossen wurde. Daher ist es auch mit nativen Android Apps möglich, den Anmeldeprozess für den Benutzer zu vereinfachen.

Geräte-Zugriff:

	Native Android Apps	Progressive Web Apps
Kamera und Mikrofon	✓	○
Dateizugriff	✓	○
GPS/Geolokalisierung	✓	✓
Akkuinformationen	✓	✓
Netzwerkinformationen	✓	✓
Vibration	✓	✓
Zwischenablage	✓	✓
Kontakte	✓	✗
Kalender	✓	✗

Tabelle 7 - Vergleich der Geräte-Zugriffe - Quelle: Eigene Darstellung

Des Weiteren bieten Progressive Web Apps und native Android Apps die Möglichkeit, auf verschiedene Gerätefunktionen und -informationen zuzugreifen. Jedoch gibt es auch hier einige Unterschiede. Sowohl Progressive Web Apps, als auch native Android Apps, können zur Aufnahme von Audio, Video oder Fotos, auf die im Gerät verbauten Kameras und das Mikrofon zugreifen (siehe Kapitel 3.2.7 und 4.2.7). Native Android Apps können zusätzlich

verschiedene Einstellungen für das Blitzlicht oder zur Fokussierung der Kamera bei Aufnahmen festlegen (vgl. Android Developers, o.J.h). Progressive Web Apps hingegen werden solche Einstellungsmöglichkeiten nicht geboten. Native Android Apps können außerdem auf das Dateisystem des mobilen Gerätes zugreifen und dementsprechend Dateien lesen, öffnen und auch speichern. Ebenfalls ist auch der Zugriff auf externe Speichermedien, wie bspw. eine SD-Karte, möglich (vgl. Android Developers, o.J.m). Progressive Web Apps bieten jedoch nur das Lesen von Dateien über die File API an, die der Benutzer ausgewählt hat (vgl. Wenz, 2014 S. 460). Auch wird kein direkter Zugriff auf das Dateisystem des Gerätes und auf externe Speichermedien geboten. Aus diesem Grund wird der Dateizugriff nur teilweise von Progressive Web Apps unterstützt.

Die Standortermittlung mittels GPS ist mit nativen Android Apps und Progressive Web Apps möglich. Auch werden von beiden App-Typen die verschiedenen, in Kapitel 3.2.7 und 4.2.7, vorgestellten Methoden zur Standortermittlung des mobilen Gerätes unterstützt. Progressive Web Apps und native Android Apps können beide auf Akku- und Netzwerkinformationen sowie auf die Vibrations-Komponente des mobilen Gerätes zugreifen. Dazu werden alle notwendigen APIs und Klassen zur Verfügung gestellt. Zudem können mit beiden App-Typen verschiedene Vibrationsmuster und -intervalle definiert und zur Steuerung der Vibration verwendet werden (siehe Kapitel 3.2.7 und 4.2.7).

Native Android Apps können auf die Zwischenablage zugreifen und diese zur Zwischenspeicherung von kopierten Daten, Bildern und Inhalten verwenden (vgl. Android Developers, o.J.l). Ebenso können native Android Apps auf die Kontakte und den Kalender zugreifen. So können bspw. Kontaktdaten ausgelesen, neue Kontakte hinzugefügt, geändert oder sogar gelöscht werden (vgl. Android Developers, o.J.k). Das gleiche gilt auch für den Kalender (vgl. Android Developers, o.J.g). Hier können auch mithilfe verschiedener Methoden Einträge hinzugefügt, geändert oder gelöscht werden. Progressive Web Apps hingegen können nur auf die Zwischenablage zugreifen und diese ebenfalls zur Zwischenspeicherung verwenden (vgl. Mozilla Developer Network, o.J.b). Der Zugriff auf die Kontakte und den Kalender ist nicht möglich, da keine APIs zur Verfügung gestellt werden. An dieser Stelle haben Progressive Web Apps einen Nachteil gegenüber nativen Android Apps.

Sensor-Zugriff:

	Native Android Apps	Progressive Web Apps
Beschleunigungssensor	✓	✓
Gyroskop	✓	✓
Magnetometer	✓	✓
Gravitationsmesser	✓	✓
Annäherungssensor	✓	✗
Helligkeitssensor	✓	✗
Temperatursensor	✓	✗
Luftdrucksensor	✓	✗

Tabelle 8 - Vergleich der Sensor-Zugriffe - Quelle: Eigene Darstellung

Native Android Apps können uneingeschränkt und direkt auf die verschiedenen, in Tabelle 8 aufgelisteten Sensoren zugreifen, die in mobilen Geräten verbaut sind. Wie in Kapitel 3.2.7 erläutert, erfolgt der Zugriff auf die verschiedenen Sensoren bei nativen Android Apps über die *SensorManager*-Klasse (vgl. Android Developers, o.J.ac). Progressive Web Apps können jedoch nur auf den Beschleunigungssensor, das Gyroskop, das Magnetometer und den Gravitationsmesser zugreifen. Für diese Sensoren werden, wie in Kapitel 4.2.7 erläutert, APIs zur Verfügung gestellt, die den Zugriff auf diese Sensoren bieten (vgl. Adam Bar, o.J.). Der Zugriff auf die anderen Sensoren, wie den Annäherungs-, Helligkeits-, Temperatur- und Luftdrucksensor wird zum jetzigen Zeitpunkt nicht unterstützt, da dafür keine APIs angeboten werden. Dies wirkt sich im Vergleich zu nativen Android Apps nachteilig aus.

Datenaustausch und Kommunikation:

	Native Android Apps	Progressive Web Apps
Bluetooth	✓	✓
NFC	✓	✗

Tabelle 9 - Vergleich von Datenaustausch und Kommunikation - Quelle: Eigene Darstellung

Eine weitere technische Funktion ist der Datenaustausch und die Kommunikation mittels Bluetooth und NFC. Native Android Apps unterstützten den Zugriff und die Verwendung dieser beiden Technologien (vgl. Android Developers, o.J.j). Wie in Kapitel 3.2.8 erklärt, können diese Technologien zur drahtlosen Übertragung von Daten und zur Kommunikation mit anderen Geräten, die ebenfalls diese Technologien unterstützten, verwendet werden (vgl. Künneth, 2018 S. 357). Dazu wird für native Android Apps die Bluetooth- und NFC API zur Verfügung gestellt (vgl. Android Developers, o.J.j). Progressive Web Apps unterstützen

zum jetzigen Zeitpunkt nur die Bluetooth-Technologie. Hierfür wird für Progressive Web Apps die Web Bluetooth API angeboten (vgl. Web Bluetooth Community Group, 2018). Außerdem wird von beiden App-Typen auch die stromsparende Bluetooth Low Energy-Technologie unterstützt (siehe Kapitel 3.2.8 und 4.2.8). Die NFC-Technologie hingegen, wird im Moment noch nicht von den Browsern und dadurch auch nicht von Progressive Web Apps unterstützt. Die Web NFC API befindet sich noch in der Entwicklungsphase und wurde noch nicht standardisiert (vgl. W3C, 2018b). Aufgrund dessen haben Progressive Web Apps einen Nachteil gegenüber nativen Android Apps.

Plattformunterstützung:

	Native Android Apps	Progressive Web Apps
Plattformunabhängigkeit	✗	✓
Darstellung und Anpassung an unterschiedliche Geräte-Typen	✓	✓

Tabelle 10 - Vergleich der Plattformunterstützung - Quelle: Eigene Darstellung

Native Android Apps können ausschließlich auf Android basierenden mobilen Geräten, wie Smartphones und Tablet-PCs, installiert und verwendet werden. Die Installation und Verwendung auf anderen Betriebssystemen, wie iOS, ist nicht möglich. Je nach Gerät und Bildschirmgröße passen sich native Android Apps an die unterschiedlichen Geräte-Eigenschaften und -Konfigurationen an. Native Android Apps, die neue APIs und Funktionen neuerer Android-Version verwenden, können u. U. nicht auf älteren, bzw. vorherigen Android Versionen installiert und verwendet werden. Denn, wie in Kapitel 3.3 beschrieben, könnten bestimmte APIs ggf. in älteren Versionen nicht mehr verfügbar sein. Dazu wurden API-Level eingeführt, die zur Bestimmung von Mindestvoraussetzungen einer Android-Version verwendet werden können (vgl. Android Developers, o.J.n). Progressive Web Apps hingegen werden mit verschiedenen Webtechnologien, wie HTML, CSS und JavaScript entwickelt und im Browser ausgeführt. Aufgrund dessen sind Progressive Web Apps plattformunabhängig und lassen sich auch durch ihr responsives Design auf verschiedenen Geräte-Typen, wie Smartphone und Tablet-PCs, ausführen und nutzen (vgl. Osmani, 2015). Durch die Plattformunabhängigkeit können Progressive Web Apps sowohl auf Android-Geräten, als auch auf iOS-Geräten, verwendet werden. Die Nutzbarkeit und der Funktionsumfang einer Progressive Web App ist von der Browser- und Betriebssystemunterstützung verschiedener HTML5 APIs abhängig (vgl. Reshetilo et al., 2017). In Kapitel 4.3 wird die Browser- und Betriebssystemunterstützung der verschiedenen HTML5 APIs aufgelistet. Auf dem Android

Betriebssystem ist die Unterstützung der HTML5 APIs sehr weit fortgeschritten. Auf dem iOS-Betriebssystem hingegen, werden viele Funktionen und APIs nicht unterstützt. Dennoch sind Progressive Web Apps auf allen Plattformen nutzbar, jedoch variiert (je nach verwendetem Browser und Betriebssystem) ihr Funktionsumfang. Da sie dennoch auf verschiedenen Plattformen verwendet werden können, sind Progressive Web Apps im Vorteil gegenüber nativen Android Apps.

Look and Feel:

Native Android Apps verfügen über Eigenschaften, Merkmale und Funktionen, die dem Benutzer eine gute User Experience bieten. Dazu zählen bspw. die Nutzbarkeit der App bei schlechter oder keiner Internetverbindung, die Installierbarkeit auf dem Gerät oder auch die Möglichkeit, bei Neuigkeiten mittels Push Notifications benachrichtigt zu werden (siehe Kapitel 3.2). Ebenso das Layout und das Design einer nativen Android App spielt eine Rolle. Native Android Apps sind optimal an das Gerät angepasst und verfügen i. d. R. über ein einheitliches Design und Layout, das den plattformspezifischen Style-Guidelines entspricht (siehe Kapitel 3.4.1). Ein einheitliches und an die Vorgaben der Plattform angepasstes Design und Layout sorgen dafür, dass die App für den Benutzer einfach zu erlernen, verwenden und vertraut erscheint (vgl. Android Developers, o.J.s). Zur Umsetzung der Designs und Layouts, werden zur Entwicklung plattformspezifische Standard-Komponenten und das Material Design zur Verfügung gestellt (vgl. Material.io, o.J.a). Dadurch, dass native Android Apps an das Betriebssystem und das Gerät angepasst sind, bieten diese dem Benutzer eine gute User Experience und somit auch ein gutes Look and Feel.

Progressive Web Apps können durch die Verwendung moderner Webtechnologien und Bibliotheken über ein ähnliches Look and Feel verfügen, wie native Android Apps (vgl. Sheppard, 2017 S. 6). Sie können ebenfalls, wie es bei nativen Android Apps der Fall ist, auf dem Gerät installiert, über den Homescreen gestartet und offline verwendet werden. Genauso können Progressive Web Apps Push Notifications empfangen und aufgrund ihres responsiven Designs auch auf unterschiedlichen Geräten dargestellt werden (siehe Kapitel 4.2). Progressive Web Apps können zudem beim Ausführen im Vollbildmodus angezeigt werden (vgl. W3C, 2018a) und unterscheiden sich dadurch kaum von einer nativen Android App.

Die Material Design Lite Bibliothek bietet dieselben Komponenten zur Gestaltung von Benutzeroberflächen für Progressive Web Apps, die auch zur Entwicklung von nativen Android Apps verwendet werden. So lassen sich durch die Nutzung dieser Bibliothek, Benutzeroberflächen für Progressive Web Apps entwickeln, die sich kaum von nativen Android Apps unterscheiden lassen (vgl. getmdl.io, o.J.). Durch diese Bibliothek können Progressive Web Apps dem Benutzer ein ähnliches Look and Feel bieten, wie native Android Apps.

Um das Look and Feel von Progressive Web Apps mit nativen Android Apps vergleichbar zu machen, wurden zwei Prototypen entwickelt. Der direkte Vergleich der Prototypen, im Hinblick auf das Look and Feel zeigt, dass die Apps beim Ausführen über den Homescreen, im Layout, im Design und in der Bedienung der Benutzeroberfläche kaum zu unterscheiden sind. Auch die Offline-Nutzbarkeit ist sowohl mit der Progressive Web App, als auch mit der nativen Android App möglich. Daraus lässt sich schließen, dass Progressive Web Apps eine ähnliche User Experience bieten können, wie native Android Apps. Abbildung 32 stellt die Benutzeroberflächen der beiden Prototypen gegenüber.

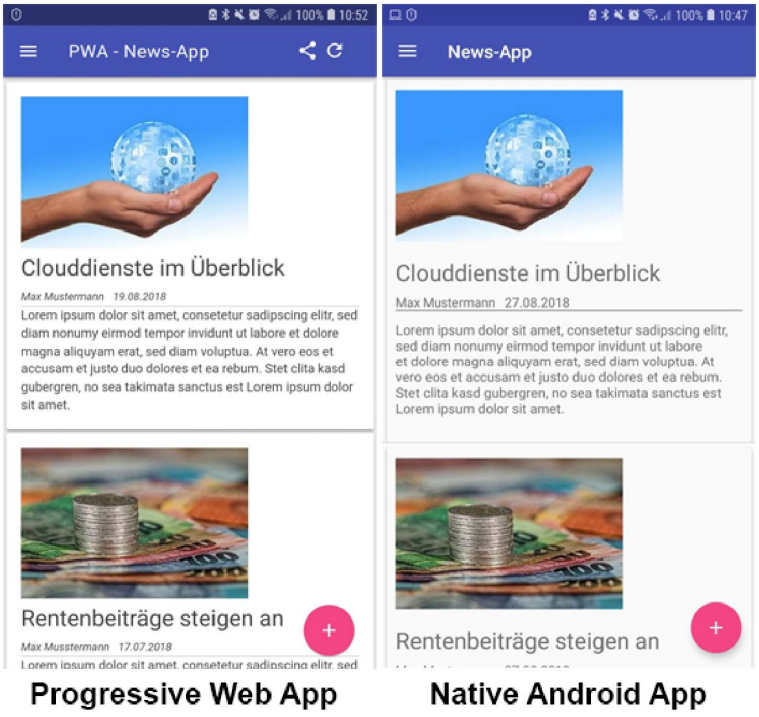

Abbildung 32 - Benutzeroberflächen der beiden Prototypen - Quelle: Eigene Darstellung

iOS Apps unterscheiden sich im Layout, Design und in den Komponenten von nativen Android Apps. Zudem gelten auch für iOS andere Style-Guidelines (siehe Kapitel 3.4.2). Progressive Web Apps können auch an iOS-Geräte angepasst werden. Dazu müssen die

Style-Guidelines berücksichtig werden. Speziell für iOS vorgefertigte Komponenten zur Gestaltung von Benutzeroberflächen werden nicht zur Verfügung gestellt. Jedoch kann die Material Design Lite Bibliothek ebenfalls zur Gestaltung verwendet werden. Der Grund dafür ist, dass diese Bibliothek zur plattformunabhängigen Verwendung entwickelt wurde. Plattformspezifische Anpassungen der Benutzeroberfläche können mit Media Queries vorgenommen und dargestellt werden.

Veröffentlichung und Verbreitung:

	Native Android Apps	Progressive Web Apps
Notwendigkeit eines App Stores	Teilweise notwendig	Nicht notwendig
Überprüfungs- und Freigabeprozess zur Veröffentlichung	Über App Stores notwendig	-
Verlinkbarkeit	✗	✓

Tabelle 11 - Vergleich der Veröffentlichung und Verbreitung - Quelle: Eigene Darstellung

Die Veröffentlichung und Verbreitung von nativen Android Apps und Progressive Web Apps unterscheidet sich voneinander. Native Android Apps benötigen in erster Linie einen App Store, um verbreitet und von Benutzern mobiler Android-Geräte gefunden und auf diesen installiert zu werden. Damit eine native Android App in einem App Store veröffentlicht werden kann, muss diese zunächst einen länger andauernden Überprüfungs- und Freigabeprozess der App Stores durchlaufen (vgl. Google Inc., o.J.l). Dadurch ist die native Android App nicht sofort im App Store verfügbar. Jedoch wird durch diesen Prozess gewährleistet, dass die App vertrauenswürdig ist. Native Android Apps müssen aber nicht zwangsläufig über einen App Store verbreitet werden. Sie können auch auf eigenen Webseiten zum Herunterladen angeboten oder per E-Mail versendet und verbreitet werden. Durch Ausführen der APK-Datei können diese anschließend installiert werden (vgl. Richter, 2018 S. 343). Aus diesem Grund ist ein App Store nur bedingt notwendig.

Für Progressive Web Apps gibt es keinen speziellen App Store (vgl. Fern, 2016), sie benötigen diesen auch nicht um verbreitet, gefunden und installiert zu werden (vgl. Firtman, 2018). Eine Ausnahme macht Microsoft. Sie bieten seit kurzem auch Progressive Web Apps in ihrem Microsoft Store an, welcher jedoch nicht für die Nutzung zwingend erforderlich ist (vgl. Navara et al., 2018). Progressive Web Apps werden stattdessen über einen Webserver zur Verfügung gestellt und sind über eine URL erreichbar (vgl. Peters, 2017 S. 89 f.). Diese URL kann zur Verbreitung der Progressive Web App über verschiedene Kanäle im Internet verwendet werden. Außerdem können einzelne Seiten direkt verlinkt werden

(vgl. Pete LePage, o.J.), was bei nativen Android Apps nicht möglich ist. Zudem sind Progressive Web Apps durch Suchmaschinen auffindbar (vgl. ebd.). Dadurch, dass Progressive Web Apps nicht über einen App Store angeboten werden, entfällt der Überprüfungs- und Freigabeprozess durch die App Stores (vgl. Dickehut, 2018). Das hat den Vorteil, dass Progressive Web Apps sofort erreichbar und nicht an die geltenden App Store-Richtlinien gebunden sind. Der Nachteil jedoch ist, dass der Benutzer die Vertrauenswürdigkeit kaum feststellen kann. Progressive Web Apps sind zudem immer aktuell und auf dem neuesten Stand und müssen nicht, wie native Android Apps, bei neuen Versionen über den App Store geupdatet werden.

7 Zusammenfassung

In diesem abschließenden Kapitel werden die Ergebnisse dieser Arbeit zusammengefasst dargestellt und eine Antwort auf die Fragestellung gegeben. Die Zielsetzung und Fragestellung der vorliegenden Arbeit war es aufzuzeigen, inwiefern Progressive Web Apps zum jetzigen Zeitpunkt in der Lage sind, native Apps zu ersetzen. Zur Eingrenzung des Umfangs beschränkte sich der Vergleich auf native Android Apps. Dazu wurden die beiden App-Typen (native Android Apps und Progressive Web Apps) im Rahmen einer Literaturrecherche und auf Basis ausgewählter Vergleichskategorien untersucht und verglichen. Zu den Vergleichskategorien gehörten die Entwicklung, die technischen und funktionalen Möglichkeiten, die Plattformunterstützung, das Look and Feel sowie die Veröffentlichung und Verbreitung. Es wurde zudem ein Prototyp einer Progressive Web App implementiert, um die Plattformunterstützung und Funktionalität zu überprüfen. Außerdem wurde auch ein Prototyp einer nativen Android App implementiert. Dabei wurde nur die Benutzeroberfläche umgesetzt, um das Look and Feel vergleichbar zu machen.

Die Untersuchung und der darauffolgende Vergleich haben gezeigt, dass sich die Entwicklung von Progressive Web Apps und nativen Apps unterscheidet. Progressive Web Apps werden mit Webtechnologien entwickelt, im Browser ausgeführt und können dadurch plattformunabhängig verwendet werden. Zudem werden zur Entwicklung keine plattformspezifischen Software Development Kits benötigt. Native Apps hingegen werden mit komplexen plattformspezifischen Programmiersprachen entwickelt und sind an die jeweilige Plattform gebunden. Außerdem wurden neben der Entwicklung auch die technischen und funktionalen Möglichkeiten untersucht und verglichen. Dabei wurde festgestellt, dass Progressive Web

Apps durch die Verwendung moderner HTML5- und JavaScript APIs, über Funktionen und Eigenschaften verfügen können, die bisher nur nativen Apps vorbehalten waren. Sie können u. a. unabhängig von der Internetverbindung verwendet und auf dem mobilen Gerät installiert werden. Auch der Empfang von Push Notifications und der Zugriff auf Geräte- und Software-Funktionen ist ebenfalls mit Progressive Web Apps möglich. Jedoch wurde gezeigt, dass Progressive Web Apps noch nicht auf alle Funktionen und Komponenten eines mobilen Gerätes zugreifen können, wie native Apps. So können Progressive Web Apps zum jetzigen Zeitpunkt noch nicht auf alle Kamera-Funktionen, das Dateisystem, die NFC-Komponente, den Kalender, die Kontakte und nicht auf alle Geräte-Sensoren zugegriffen werden. Auch die Inter-App Kommunikation ist nur eingeschränkt möglich. In diesen Punkten sind native Apps immer noch im Vorteil gegenüber Progressive Web Apps. Dennoch verfügen Progressive Web Apps über Funktionen und Eigenschaften, die native Apps nicht bieten. So können bspw. Progressive Web Apps, anders als native Apps, direkt über die Suchmaschine aufgefunden werden.

Um eine genaue Aussage über die Plattformunterstützung von Progressive Web Apps machen zu können, wurde ein Prototyp einer Progressive Web App implementiert und die Unterstützung der APIs auf unterschiedlichen Geräten und Plattformen getestet. Die Ergebnisse der durchgeführten Tests haben gezeigt, dass noch nicht alle APIs von den Browsern und den Betriebssystemen unterstützt werden. Die Unterstützung von Progressive Web Apps durch das Android-Betriebssystem ist weit fortgeschritten. iOS unterstützt Progressive Web Apps nur sehr eingeschränkt. Dadurch variiert, je nach Browser und Betriebssystem, der Funktionsumfang von Progressive Web Apps.

Des Weiteren hat die Untersuchung und der Vergleich der Prototypen gezeigt, dass die Benutzeroberfläche von Progressive Web Apps durch die Verwendung der Material Design Lite Bibliothek, dem Benutzer ein ähnliches Look and Feel bieten können, wie native Apps. Auch die Anpassung der Benutzeroberfläche an iOS ist möglich. Zudem tragen auch Funktionen und Eigenschaften, wie die Installierbarkeit, die Offline-Nutzbarkeit sowie der Empfang von Push Notifications dazu bei, dass Progressive Web Apps ein app-ähnliches Look and Feel bieten können.

Abschließend wurden die Möglichkeiten zur Veröffentlichung und Verbreitung untersucht und verglichen. Dabei konnte aufgezeigt werden, dass Progressive Web Apps, anders als native Apps, nicht über die plattformspezifischen App Stores heruntergeladen und installiert

werden. Zudem stellte sich heraus, dass es für Progressive Web Apps, bis auf den Microsoft Store, keine speziellen App Stores gibt. Stattdessen können Progressive Web Apps direkt über eine URL oder über Suchmaschinen aufgerufen, verbreitet und aufgefunden werden. Progressive Web Apps haben gegenüber nativen Apps den Vorteil, dass sie nicht den Überprüfungs- und Freigabeprozess der App Stores durchlaufen müssen und an dessen Richtlinien gebunden sind. Dies ist zugleich aber auch ein Nachteil für Progressive Web Apps, denn es findet keine Überprüfung statt und der Benutzer kann nur schlecht feststellen, ob eine Progressive Web App vertrauenswürdig ist. Anders als native Apps sind Progressive Web Apps immer aktuell und müssen nicht über einen App Store aktualisiert werden.

Insgesamt konnte durch die Untersuchung und den Vergleich festgestellt werden, dass Progressive Web Apps zum jetzigen Zeitpunkt noch nicht vollumfänglich native Apps ersetzen können. Der Grund dafür ist zum einen, der eingeschränkte Zugriff auf Geräte- und Software-Funktionen und zum anderen die eingeschränkte und fehlende Browser- und Betriebssystemunterstützung. Dadurch ist, je nach Funktionsumfang und Plattform, dennoch die Entwicklung von nativen Apps erforderlich. Die Zukunft wird zeigen, ob die Browser- und Betriebssystemhersteller die Unterstützung von Progressive Web Apps weiter ausweiten und verbessern werden. Insbesondere bei Apples iOS und Safari gibt es verbesserungsbedarf.

8 Literatur

1und1, 2017. *Progressive Web-Apps | Welche Vorteile bieten sie?* [online] [Zugriff am: 17. Oktober 2018]. Verfügbar unter: https://hosting.1und1.de/digitalguide/websites/webentwicklung/progressive-web-apps-welche-vorteile-bieten-sie/.

Abts, D., 2018. *Grundkurs JAVA. Von den Grundlagen bis zu Datenbank- und Netzanwendungen.* 10. Aufl. 2018. Wiesbaden: Springer Fachmedien Wiesbaden. ISBN 9783658219062.

Ackermann, P., 2016a. *Features von übermorgen: die Web Bluetooth API* [online] [Zugriff am: 3. November 2018]. Verfügbar unter: https://www.heise.de/developer/artikel/Features-von-uebermorgen-die-Web-Bluetooth-API-3167796.html.

—, 2016b. *Features von übermorgen: Die Web Share API und die Web Share Target API* [online] [Zugriff am: 30. November 2018]. Verfügbar unter: https://www.heise.de/developer/artikel/Features-von-uebermorgen-Die-Web-Share-API-und-die-Web-Share-Target-API-3506197.html.

Adam Bar, o.J. *What Web Can Do Today* [online] [Zugriff am: 11. September 2018]. Verfügbar unter: https://whatwebcando.today/device-motion.html.

Android Developers, o.J.a. *<uses-sdk> | Android Developers* [online] [Zugriff am: 4. Oktober 2018]. Verfügbar unter: https://developer.android.com/guide/topics/manifest/uses-sdk-element.

—, o.J.b. *android.nfc | Android Developers* [online] [Zugriff am: 27. September 2018]. Verfügbar unter: https://developer.android.com/reference/android/nfc/package-summary.

—, o.J.c. *App Manifest Overview | Android Developers* [online] [Zugriff am: 4. September 2018]. Verfügbar unter: https://developer.android.com/guide/topics/manifest/manifest-intro.

—, o.J.d. *BatteryManager | Android Developers* [online] [Zugriff am: 9. Oktober 2018]. Verfügbar unter: https://developer.android.com/reference/android/os/BatteryManager.

—, o.J.e. *Bluetooth low energy overview | Android Developers* [online] [Zugriff am: 27. September 2018]. Verfügbar unter: https://developer.android.com/guide/topics/connectivity/bluetooth-le.

—, o.J.f. *Bluetooth overview | Android Developers* [online] [Zugriff am: 27. September 2018]. Verfügbar unter: https://developer.android.com/guide/topics/connectivity/bluetooth.

—, o.J.g. *Calendar provider overview | Android Developers* [online] [Zugriff am: 9. Oktober 2018]. Verfügbar unter: https://developer.android.com/guide/topics/providers/calendar-provider.

—, o.J.h. *Camera API* [online] [Zugriff am: 1. Oktober 2018]. Verfügbar unter: https://developer.android.com/guide/topics/media/camera.

—, o.J.i. *Checkliste für die Einführung | Android Developers* [online] [Zugriff am: 9. September 2018]. Verfügbar unter: https://developer.android.com/distribute/best-practices/launch/launch-checklist?hl=de.

—, o.J.j. *Connectivity | Android Developers* [online] [Zugriff am: 27. September 2018]. Verfügbar unter: https://developer.android.com/guide/topics/connectivity/.

—, o.J.k. *Contacts Provider | Android Developers* [online] [Zugriff am: 9. Oktober 2018]. Verfügbar unter: https://developer.android.com/guide/topics/providers/contacts-provider.

—, o.J.l. *Copy and Paste | Android Developers* [online] [Zugriff am: 9. Oktober 2018]. Verfügbar unter: https://developer.android.com/guide/topics/text/copy-paste.

—, o.J.m. *Data and file storage overview | Android Developers* [online] [Zugriff am: 3. Oktober 2018]. Verfügbar unter: https://developer.android.com/guide/topics/data/data-storage.

—, o.J.n. *Device compatibility overview | Android Developers* [online] [Zugriff am: 3. Oktober 2018]. Verfügbar unter: https://developer.android.com/guide/practices/compatibility.

—, o.J.o. *Interacting with Other Apps* [online] [Zugriff am: 23. September 2018]. Verfügbar unter: https://developer.android.com/training/basics/intents/.

—, o.J.p. *Location | Android Developers* [online] [Zugriff am: 7. Oktober 2018]. Verfügbar unter: https://developer.android.com/reference/android/location/Location.

—, o.J.q. *LocationManager | Android Developers* [online] [Zugriff am: 7. Oktober 2018]. Verfügbar unter: https://developer.android.com/reference/android/location/LocationManager.

—, o.J.r. *Manage network usage | Android Developers* [online] [Zugriff am: 9. Oktober 2018]. Verfügbar unter: https://developer.android.com/training/basics/network-ops/managing.

—, o.J.s. *Material Design for Android | Android Developers* [online] [Zugriff am: 13. Oktober 2018]. Verfügbar unter: https://developer.android.com/guide/topics/ui/look-and-feel/.

—, o.J.t. *MediaPlayer | Android Developers* [online] [Zugriff am: 22. September 2018]. Verfügbar unter: https://developer.android.com/reference/android/media/MediaPlayer.

—, o.J.u. *MediaPlayer overview | Android Developers* [online] [Zugriff am: 22. September 2018]. Verfügbar unter: https://developer.android.com/guide/topics/media/mediaplayer.

—, o.J.v. *MediaRecorder overview | Android Developers* [online] [Zugriff am: 9. Oktober 2018]. Verfügbar unter: https://developer.android.com/guide/topics/media/mediarecorder.

—, o.J.w. *Multiple APK support | Android Developers* [online] [Zugriff am: 3. Oktober 2018]. Verfügbar unter: https://developer.android.com/google/play/publishing/multiple-apks.

—, o.J.x. *NotificationManager | Android Developers* [online] [Zugriff am: 28. September 2018]. Verfügbar unter: https://developer.android.com/reference/android/app/NotificationManager.

—, o.J.y. *Open files using storage access framework | Android Developers* [online] [Zugriff am: 9. Oktober 2018]. Verfügbar unter: https://developer.android.com/guide/topics/providers/document-provider.

—, o.J.z. *Save files on device storage | Android Developers* [online] [Zugriff am: 9. Oktober 2018]. Verfügbar unter: https://developer.android.com/training/data-storage/files.

—, o.J.aa. *Sending the User to Another App | Android Developers* [online] [Zugriff am: 27. Oktober 2018]. Verfügbar unter: https://developer.android.com/training/basics/intents/sending.

—, o.J.ab. *SensorManager | Android Developers* [online] [Zugriff am: 8. Oktober 2018]. Verfügbar unter: https://developer.android.com/reference/android/hardware/SensorManager.

—, o.J.ac. *Sensors Overview | Android Developers* [online] [Zugriff am: 7. Oktober 2018]. Verfügbar unter: https://developer.android.com/guide/topics/sensors/sensors_overview.

—, o.J.ad. *Supported media formats | Android Developers* [online] [Zugriff am: 22. September 2018]. Verfügbar unter: https://developer.android.com/guide/topics/media/media-formats.

—, o.J.ae. *Transfer data using sync adapters | Android Developers* [online] [Zugriff am: 3. Oktober 2018]. Verfügbar unter: https://developer.android.com/training/sync-adapters/.

—, o.J.af. *Über umfassende Benachrichtigungen interagieren | Android Developers* [online] [Zugriff am: 28. September 2018]. Verfügbar unter: https://developer.android.com/distribute/best-practices/engage/rich-notifications.

—, o.J.ag. *Using a media session | Android Developers* [online] [Zugriff am: 22. September 2018]. Verfügbar unter: https://developer.android.com/guide/topics/media-apps/working-with-a-media-session.

—, o.J.ah. *Vibrator | Android Developers* [online] [Zugriff am: 9. Oktober 2018]. Verfügbar unter: https://developer.android.com/reference/android/os/Vibrator.

Apple Inc., 2007. *Apple Reinvents the Phone with iPhone* [online] [Zugriff am: 21. September 2018]. Verfügbar unter: https://www.apple.com/newsroom/2007/01/09Apple-Reinvents-the-Phone-with-iPhone/.

—, o.J.a. *Color - Visual Design - iOS - Human Interface Guidelines - Apple Developer* [online] [Zugriff am: 13. Oktober 2018]. Verfügbar unter: https://developer.apple.com/design/human-interface-guidelines/ios/visual-design/color/.

—, o.J.b. *Core Services Layer* [online] [Zugriff am: 21. Oktober 2018]. Verfügbar unter: https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/OSX_Technology_Overview/CoreServicesLayer/CoreServicesLayer.html.

—, o.J.c. *Interface Essentials - iOS - iOS - Human Interface Guidelines - Apple Developer* [online] [Zugriff am: 13. Oktober 2018]. Verfügbar unter: https://developer.apple.com/design/human-interface-guidelines/ios/overview/interface-essentials/.

—, o.J.d. *iOS - Human Interface Guidelines - Apple Developer* [online] [Zugriff am: 13. Oktober 2018]. Verfügbar unter: https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/.

—, o.J.e. *iOS 11* [online] [Zugriff am: 21. Oktober 2018]. Verfügbar unter: https://www.apple.com/ios/ios-11/.

—, o.J.f. *iPhone 8 - iOS* [online] [Zugriff am: 21. Oktober 2018]. Verfügbar unter: https://www.apple.com/de/iphone-8/ios/.

—, o.J.g. *Navigation Bars - Bars - iOS - Human Interface Guidelines - Apple Developer* [online] [Zugriff am: 13. Oktober 2018]. Verfügbar unter: https://developer.apple.com/design/human-interface-guidelines/ios/bars/navigation-bars/.

—, o.J.h. *Status Bars - Bars - iOS - Human Interface Guidelines - Apple Developer* [online] [Zugriff am: 13. Oktober 2018]. Verfügbar unter: https://developer.apple.com/design/human-interface-guidelines/ios/bars/status-bars/.

—, o.J.i. *Swift - Apple Developer* [online] [Zugriff am: 21. September 2018]. Verfügbar unter: https://developer.apple.com/swift/.

—, o.J.j. *System Icons - Icons and Images - iOS - Human Interface Guidelines - Apple Developer* [online] [Zugriff am: 13. Oktober 2018]. Verfügbar unter: https://developer.apple.com/design/human-interface-guidelines/ios/icons-and-images/system-icons/.

—, o.J.k. *Tab Bars - Bars - iOS - Human Interface Guidelines - Apple Developer* [online] [Zugriff am: 13. Oktober 2018]. Verfügbar unter: https://developer.apple.com/design/human-interface-guidelines/ios/bars/tab-bars/.

—, o.J.l. *Typography - Visual Design - iOS - Human Interface Guidelines - Apple Developer* [online] [Zugriff am: 13. Oktober 2018]. Verfügbar unter: https://developer.apple.com/design/human-interface-guidelines/ios/visual-design/typography/.

—, o.J.m. *Xcode - IDE - Apple Developer* [online] [Zugriff am: 21. September 2018]. Verfügbar unter: https://developer.apple.com/xcode/ide/.

Archibald, J., o.J. *Introducing Background Sync | Web | Google Developers* [online] [Zugriff am: 29. September 2018]. Verfügbar unter: https://developers.google.com/web/updates/2015/12/background-sync.

Ater, T., 2017. *Building progressive web apps. Bringing the power of native to the browser.* First edition. Beijing: O'Reilly. ISBN 9781491961650.

Bach, M., 2012. *Mobile Anwendungen mit Android. Entwicklung und praktischer Einsatz.* München: Addison-Wesley. Always learning. ISBN 9783827330475.

Basques, K., o.J. *Debug Progressive Web Apps | Tools for Web Developers | Google Developers* [online] [Zugriff am: 4. September 2018]. Verfügbar unter: https://developers.google.com/web/tools/chrome-devtools/progressive-web-apps.

Beaufort, F., o.J. *Customize Media Notifications and Handle Playlists | Web | Google Developers* [online] [Zugriff am: 22. Oktober 2018]. Verfügbar unter: https://developers.google.com/web/updates/2017/02/media-session.

Becker, A. und **M. Pant,** 2015. *Android 5. Programmieren für Smartphones und Tablets.* 4., aktuelle und erw. Aufl. Heidelberg: dpunkt.Verl. ISBN 9783864902604.

Belmont, J., 2016. *Building Progressive Web Apps: Webinar and Tutorial | Blog | Vaadin* [online] [Zugriff am: 29. August 2018]. Verfügbar unter: https://vaadin.com/blog/building-progressive-web-apps-webinar-and-tutorial.

Braun, H., 2017. Progressive Web Apps. *c't Programmieren*.

Bühler, P., P. Schlaich und **D. Sinner,** 2018. *Webtechnologien. JavaScript - PHP - Datenbank.* Berlin: Springer Vieweg. Bibliothek der Mediengestaltung. ISBN 9783662547298.

—. *HTML5 und CSS3. Semantik - Design - Responsive Layouts:* Springer Vieweg. ISBN 978-3-662-53915-6.

Campbell-Moore, O., 2016. *Designing Great UIs for Progressive Web Apps – Owen Campbell-Moore – Medium* [online] [Zugriff am: 17. September 2018]. Verfügbar unter: https://medium.com/@owencm/designing-great-uis-for-progressive-web-apps-dd38c1d20f7.

Dickehut, A., 2018. *Progressive-Web-Apps: Das App-Modell der Zukunft* [online] [Zugriff am: 18. September 2018]. Verfügbar unter: https://t3n.de/news/progressive-web-apps-app-modell-1078161/.

Diedrich, O., 2017. *Android-Entwicklung: Google zieht die Daumenschrauben an* [online] [Zugriff am: 4. Oktober 2018]. Verfügbar unter: https://www.heise.de/ix/meldung/Android-Entwicklung-Google-zieht-die-Daumenschrauben-an-3924579.html.

Domes, S., 2017. *Progressive web apps with React. Create lightning fast web apps with native power using React and Firebase.* Birmingham, UK: Packt Publishing. ISBN 9781788297554.

East, D., 2015. *Chrome Developer Summit Recap – David East – Medium* [online] [Zugriff am: 4. September 2018]. Verfügbar unter: https://medium.com/@davideast/chrome-developer-summit-recap-1137b022b2dc.

Fern, D., 2016. *Progressive Web Applications, Part 2: Pros, Cons, and Looking Ahead* [online] [Zugriff am: 18. September 2018]. Verfügbar unter: https://digital.gov/2016/10/13/progressive-web-applications-part-2-pros-cons-and-looking-ahead/.

Firtman, M., 2018. *Progressive Web Apps on iOS are here – Maximiliano Firtman – Medium* [online] [Zugriff am: 13. August 2018]. Verfügbar unter: https://medium.com/@firt/progressive-web-apps-on-ios-are-here-d00430dee3a7.

Gaunt, M., o.J.a. *How Push Works | Web Fundamentals | Google Developers* [online] [Zugriff am: 30. August 2018]. Verfügbar unter: https://developers.google.com/web/fundamentals/push-notifications/how-push-works.

—, o.J.b. *Service Workers: an Introduction | Web Fundamentals | Google Developers* [online] [Zugriff am: 29. August 2018]. Verfügbar unter: https://developers.google.com/web/fundamentals/primers/service-workers/.

getmdl.io, o.J. *Material Design Lite* [online] [Zugriff am: 17. September 2018]. Verfügbar unter: https://getmdl.io/index.html.

Google Developer Training, o.J. *Introduction to PWA Architectures · Progressive Web Apps ILT - Concepts* [online] [Zugriff am: 18. September 2018]. Verfügbar unter: https://google-developer-training.gitbooks.io/progressive-web-apps-ilt-concepts/content/docs/introduction-to-progressive-web-app-architectures.html.

Google Inc., o.J.a. *Application security | Android Open Source Project* [online] [Zugriff am: 25. September 2018]. Verfügbar unter: https://source.android.com/security/overview/app-security.

—, o.J.b. *Chrome DevTools | Tools for Web Developers | Google Developers* [online] [Zugriff am: 4. September 2018]. Verfügbar unter: https://developers.google.com/web/tools/chrome-devtools/.

—, o.J.c. *Codelines, Branches, and Releases | Android Open Source Project* [online] [Zugriff am: 20. September 2018]. Verfügbar unter: https://source.android.com/setup/start/codelines.

—, o.J.d. *Codenames, Tags, and Build Numbers | Android Open Source Project* [online] [Zugriff am: 20. September 2018]. Verfügbar unter: https://source.android.com/setup/start/build-numbers.

—, o.J.e. *Content License | Android Open Source Project* [online] [Zugriff am: 20. September 2018]. Verfügbar unter: https://source.android.com/setup/start/licenses.

—, o.J.f. *Download Android Studio and SDK tools | Android Developers* [online] [Zugriff am: 24. September 2018]. Verfügbar unter: https://developer.android.com/studio/.

—, o.J.g. *Frequently Asked Questions | Android Open Source Project* [online] [Zugriff am: 20. September 2018]. Verfügbar unter: https://source.android.com/setup/start/faqs.

—, o.J.h. *Lighthouse PWA Analysis Tool | Web | Google Developers* [online] [Zugriff am: 3. September 2018]. Verfügbar unter: https://developers.google.com/web/ilt/pwa/lighthouse-pwa-analysis-tool.

—, o.J.i. *Meet Android Studio | Android Developers* [online] [Zugriff am: 2. September 2018]. Verfügbar unter: https://developer.android.com/studio/intro/.

—, o.J.j. *Platform Architecture | Android Developers* [online] [Zugriff am: 20. September 2018]. Verfügbar unter: https://developer.android.com/guide/platform/.

—, o.J.k. *Prepare for release | Android Developers* [online] [Zugriff am: 30. September 2018]. Verfügbar unter: https://developer.android.com/studio/publish/preparing.

—, o.J.l. *Publish an app - Play Console Help* [online] [Zugriff am: 30. September 2018]. Verfügbar unter: https://support.google.com/googleplay/android-developer/answer/6334282?hl=en.

—, o.J.m. *Richtlinienübersicht für Entwickler* [online] [Zugriff am: 30. September 2018]. Verfügbar unter: https://play.google.com/about/developer-content-policy/.

—, o.J.n. *Run apps on the Android Emulator | Android Developers* [online] [Zugriff am: 24. September 2018]. Verfügbar unter: https://developer.android.com/studio/run/emulator.

—, o.J.o. *Startleitfaden zur Suchmaschinenoptimierung (SEO) - Hilfe für Search Console* [online] [Zugriff am: 24. August 2018]. Verfügbar unter: https://support.google.com/webmasters/answer/7451184?hl=de.

Hume, D.A. und **A. Osmani,** 2018. *Progressive web apps.* Shelter Island, NY. ISBN 9781617294587.

Initiative D21, 2014. *Mobile Internetnutzer - Anteil in Deutschland 2014 | Statistik* [online] [Zugriff am: 31. Oktober 2018]. Verfügbar unter: https://de.statista.com/statistik/daten/studie/197383/umfrage/mobile-internetnutzung-ueber-handy-in-deutschland/.

Jackson, W., 2013. *Learn Android App Development:* Apress. ISBN 9781430257462.

Kinlan, P. und **S. Thorogood,** o.J. *Introducing the Web Share API | Web | Google Developers* [online] [Zugriff am: 30. September 2018]. Verfügbar unter: https://developers.google.com/web/updates/2016/09/navigator-share.

Kitamura, E., o.J. *Streamlining the Sign-in Flow Using Credential Management API | Web | Google Developers* [online] [Zugriff am: 3. September 2018]. Verfügbar unter: https://developers.google.com/web/updates/2016/04/credential-management-api.

Kling, B., 2017. *Welche Vorteile bieten Progressive Web Apps?* [online] [Zugriff am: 29. September 2018]. Verfügbar unter: https://www.heise-regioconcept.de/mobile-marketing/progressive-web-apps-vorteile.

Kofler, M., 2018. *Java. Der Grundkurs.* 2., aktualisierte Auflage. Bonn: Rheinwerk Verlag. Rheinwerk Computing. ISBN 9783836245814.

Kumar, S., 2018. *Speeding up development of your enterprise mobile apps* [online] [Zugriff am: 22. Oktober 2018]. Verfügbar unter: https://developer.ibm.com/code/2018/02/08/speeding-development-enterprise-mobile-apps-using-open-source-technologies-cloud-services/.

Künneth, T., 2018. *Android 8. Das Praxisbuch für Java-Entwickler.* 5., aktualisierte Auflage. Bonn: Rheinwerk Verlag. Rheinwerk Computing. ISBN 9783836260589.

Lee, W.-M., 2012. *Beginning Android 4 application development.* Indianapolis, IN: Wiley. ISBN 1280672927.

LePage, 2018a. *WebAPKs on Android | Web Fundamentals | Google Developers* [online] [Zugriff am: 30. September 2018]. Verfügbar unter: https://developers.google.com/web/fundamentals/integration/webapks.

—, 2018b. *Your First Progressive Web App | Web Fundamentals | Google Developers* [online] [Zugriff am: 30. September 2018]. Verfügbar unter: https://developers.google.com/web/fundamentals/codelabs/your-first-pwapp/.

LePage, P., 2014. *HTML5-Video* [online] [Zugriff am: 30. Oktober 2018]. Verfügbar unter: https://www.html5rocks.com/de/tutorials/video/basics/.

—, o.J. *Device Orientation & Motion | Web Fundamentals | Google Developers* [online] [Zugriff am: 3. Oktober 2018]. Verfügbar unter: https://developers.google.com/web/fundamentals/native-hardware/device-orientation/.

Liebel, C., 2017. *Progressive Web Apps, Teil 1: Das Web wird nativ(er)* [online] [Zugriff am: 18. September 2018]. Verfügbar unter: https://www.heise.de/developer/artikel/Progressive-Web-Apps-Teil-1-Das-Web-wird-nativ-er-3733624.html.

Material.io, o.J.a. *Components - Material Design* [online] [Zugriff am: 13. Oktober 2018]. Verfügbar unter: https://material.io/design/components/.

—, o.J.b. *System icons - Material Design* [online] [Zugriff am: 13. Oktober 2018]. Verfügbar unter: https://material.io/design/iconography/system-icons.html.

—, o.J.c. *The color system - Material Design* [online] [Zugriff am: 13. Oktober 2018]. Verfügbar unter: https://material.io/design/color/the-color-system.html.

—, o.J.d. *The type system - Material Design* [online] [Zugriff am: 13. Oktober 2018]. Verfügbar unter: https://material.io/design/typography/the-type-system.html.

Matt Gaunt, o.J. *Permissions API for the Web | Web | Google Developers* [online] [Zugriff am: 30. September 2018]. Verfügbar unter: https://developers.google.com/web/updates/2015/04/permissions-api-for-the-web.

McLemore, M., T. Opgenorth, **C. Dunn** und **B. Umbaugh,** 2018. *Firebase Cloud Messaging* [online] [Zugriff am: 28. September 2018]. Verfügbar unter: https://docs.microsoft.com/en-us/xamarin/android/data-cloud/google-messaging/firebase-cloud-messaging.

Medley, J., o.J. *Web Push Notifications: Timely, Relevant, and Precise | Web Fundamentals | Google Developers* [online] [Zugriff am: 26. August 2018]. Verfügbar unter: https://developers.google.com/web/fundamentals/push-notifications/.

Meier, R., 2018. *Professional Android.* 4th ed. Somerset: John Wiley & Sons Incorporated. ISBN 9781118949528.

Merkert, J., 2016. NFC-Programmierung mit Android [online], (4) [Zugriff am: 27. September 2018]. Verfügbar unter: https://www.heise.de/ct/ausgabe/2016-4-NFC-Programmierung-mit-Android-3087215.html.

Mozilla Developer Network, 2009. *Media formats for HTML audio and video* [online] [Zugriff am: 30. September 2018]. Verfügbar unter: https://developer.mozilla.org/en-US/docs/Web/HTML/Supported_media_formats.

—, 2013a. *Grundkonzepte* [online] [Zugriff am: 3. September 2018]. Verfügbar unter: https://developer.mozilla.org/de/docs/IndexedDB/Grundkonzepte_hinter_IndexedDB.

—, 2013b. *IndexedDB* [online] [Zugriff am: 29. Oktober 2018]. Verfügbar unter: https://developer.mozilla.org/de/docs/IndexedDB.

—, 2013c. *Permissions API* [online] [Zugriff am: 30. Oktober 2018]. Verfügbar unter: https://developer.mozilla.org/en-US/docs/Web/API/Permissions_API.

—, 2015. *MediaDevices.getUserMedia* [online] [Zugriff am: 12. September 2018]. Verfügbar unter: https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia.

—, 2016a. *JavaScript-Grundlagen* [online] [Zugriff am: 1. Oktober 2018]. Verfügbar unter: https://developer.mozilla.org/de/Learn/Getting_started_with_the_web/JavaScript_basis.

—, 2016b. *Video and audio content* [online] [Zugriff am: 30. Oktober 2018]. Verfügbar unter: https://developer.mozilla.org/en-US/docs/Learn/HTML/Multimedia_and_embedding/Video_and_audio_content.

—, 2018a. *How to make PWAs installable* [online] [Zugriff am: 30. Oktober 2018]. Verfügbar unter: https://developer.mozilla.org/en-US/Apps/Progressive/Installable_PWAs.

—, 2018b. *Service Worker API* [online] [Zugriff am: 29. Oktober 2018]. Verfügbar unter: https://developer.mozilla.org/de/docs/Web/API/Service_Worker_API.

—, o.J.a. *Battery Status API* [online] [Zugriff am: 12. September 2018]. Verfügbar unter: https://developer.mozilla.org/en-US/docs/Web/API/Battery_Status_API.

—, o.J.b. *Clipboard API* [online] [Zugriff am: 12. September 2018]. Verfügbar unter: https://developer.mozilla.org/en-US/docs/Web/API/Clipboard_API.

—, o.J.c. *Detecting device orientation* [online] [Zugriff am: 12. September 2018]. Verfügbar unter: https://developer.mozilla.org/en-US/docs/Web/API/Detecting_device_orientation.

—, o.J.d. *Geolocation API* [online] [Zugriff am: 12. September 2018]. Verfügbar unter: https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API.

—, o.J.e. *Navigator.vibrate* [online] [Zugriff am: 12. September 2018]. Verfügbar unter: https://developer.mozilla.org/de/docs/Web/API/Navigator/vibrate.

—, o.J.f. *Network Information API* [online] [Zugriff am: 3. Oktober 2018]. Verfügbar unter: https://developer.mozilla.org/en-US/docs/Web/API/Network_Information_API.

—, o.J.g. *Notifications API* [online] [Zugriff am: 30. September 2018]. Verfügbar unter: https://developer.mozilla.org/en-US/docs/Web/API/Notifications_API.

—, o.J.h. *Push API* [online] [Zugriff am: 30. September 2018]. Verfügbar unter: https://developer.mozilla.org/en-US/docs/Web/API/Push_API.

—, o.J.i. *Zugriff auf Dateien von Webapplikationen* [online] [Zugriff am: 1. September 2018]. Verfügbar unter: https://developer.mozilla.org/de/docs/Web/API/File/Zugriff_auf_Dateien_von_Webapplikationen.

Navara, E.D., L. McCormick, **Y. Huang** und **W. Matt,** 2018. *Progressive Web Apps in the Microsoft Store - Microsoft Edge Development* [online] [Zugriff am: 18. September 2018]. Verfügbar unter: https://docs.microsoft.com/en-us/microsoft-edge/progressive-web-apps/microsoft-store.

Open Handset Alliance, o.J. *Android Overview | Open Handset Alliance* [online] [Zugriff am: 20. Oktober 2018]. Verfügbar unter: http://www.openhandsetalliance.com/android_overview.html.

Oracle Corp., o.J. *Java SE - Downloads | Oracle Technology Network | Oracle* [online] [Zugriff am: 24. Oktober 2018]. Verfügbar unter: http://www.oracle.com/technetwork/java/javase/downloads/index.html.

Osmani, A., 2015. *AddyOsmani.com - Getting started with Progressive Web Apps* [online] [Zugriff am: 18. Oktober 2018]. Verfügbar unter: https://addyosmani.com/blog/getting-started-with-progressive-web-apps/.

—, o.J. *The App Shell Model | Web Fundamentals | Google Developers* [online] [Zugriff am: 4. Oktober 2018]. Verfügbar unter: https://developers.google.com/web/fundamentals/architecture/app-shell.

Pete LePage, o.J. *Your First Progressive Web App | Web Fundamentals | Google Developers* [online] [Zugriff am: 18. Oktober 2018]. Verfügbar unter: https://developers.google.com/web/fundamentals/codelabs/your-first-pwapp/.

Petereit, D., 2016. *Was sind Progressive Web Apps?* [online] [Zugriff am: 31. Oktober 2018]. Verfügbar unter: https://t3n.de/news/progressive-web-apps-739224/.

Peters, M., 2017. *Progressive web apps. Der Sprung ins nächste App-Zeitalter.* Frankfurt am Main: [Selbstverlag]. ISBN 9781521881644.

Post, U., 2018. *Android-Apps entwickeln für Einsteiger.* 7., aktualisierte Auflage. Bonn: Rheinwerk Verlag. Rheinwerk Computing. ISBN 9783836261395.

Reshetilo, K. und **S. Opanasenko,** 2017. *Progressive Web Apps vs Native: Which Is Better for Your Business?* [online] [Zugriff am: 19. September 2018]. Verfügbar unter: https://www.technology.org/2017/07/28/progressive-web-apps-vs-native-which-is-better-for-your-business/.

Richter, E., 2018. *Android-Apps programmieren. Praxiseinstieg mit Android Studio.* Frechen: MITP. mitp Professional. ISBN 9783958452589.

Robbins, 2014. *HTML5 - kurz & gut.* Beijing: O'Reilly. Kurz & gut. ISBN 9783955616564.

Russell, 2015. *Progressive Web Apps: Escaping Tabs Without Losing Our Soul | Infrequently Noted* [online] [Zugriff am: 18. September 2018]. Verfügbar unter: https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/.

Ryte, o.J. *Progressive Web Apps – Ryte Wiki - Digitales Marketing Wiki* [online] [Zugriff am: 30. August 2018]. Verfügbar unter: https://de.ryte.com/wiki/Progressive_Web_Apps.

Schickler, M., M. Reichert, **R. Pryss**, **J. Schobel**, **W. Schlee** und **B. Langguth,** 2015. *Entwicklung mobiler Apps.* Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-33056-8.

selfhtml.org, o.J.a. *CSS – SELFHTML-Wiki* [online] [Zugriff am: 30. September 2018]. Verfügbar unter: https://wiki.selfhtml.org/wiki/CSS.

—, o.J.b. *JavaScript/Geolocation – SELFHTML-Wiki* [online] [Zugriff am: 3. September 2018]. Verfügbar unter: https://wiki.selfhtml.org/wiki/JavaScript/Geolocation.

Shaked, U., 2016. *Start Building with Web Bluetooth and Progressive Web Apps* [online] [Zugriff am: 3. September 2018]. Verfügbar unter: https://medium.com/@urish/start-building-with-web-bluetooth-and-progressive-web-apps-6534835959a6.

Sheppard, D., 2017. *Beginning Progressive Web App Development. Creating a Native App Experience on the Web.* Berkeley, CA: Apress. ISBN 9781484230893.

Sillmann, T., 2017. *Apps für iOS 10 professionell entwickeln. Sauberen Code schreiben mit Swift 3 und Objective-C : stabile Apps für iPhone und iPad programmieren : Techniken & Methoden von Grund auf verstehen.* München: Hanser Verlag. ISBN 9783446452060.

Springer, S., 2016. Dienst am Kunden. *iX - Magazin für professionelle Informationstechnik*, (6).

sqlite.org, o.J. *About SQLite* [online] [Zugriff am: 3. Oktober 2018]. Verfügbar unter: https://www.sqlite.org/about.html.

Staudemeyer, J., 2018. *Android mit Kotlin – kurz & gut. Inklusive Android 8 und Android Studio 3.0.* Heidelberg: O'Reilly. Kurz & gut. ISBN 9783960090380.

Steyer, M., 2017a. *Progressive Web-Apps. Offlinefähige Web-Anwendungen mit nativen Qualitäten.* Frankfurt am Main: entwickler.press. shortcut. v.209. ISBN 978-3-86802-753-2.

Steyer, R., 2017b. *Cordova. Entwicklung plattformneutraler Apps.* Wiesbaden: Springer Vieweg. ISBN 9783658167240.

—, 2018. *Jetzt lerne ich : App-Entwicklung für iOS mit Xcode und Swift.* Burgthann: Markt+Technik. Markt+Technik. 2043. ISBN 9783959820431.

Theis, T., 2016. *Einstieg in JavaScript.* 2., aktualisierte und erweiterte Auflage. Bonn: Rheinwerk Verlag GmbH. Rheinwerk Computing. ISBN 9783836240741.

Tosic, M., 2015. *Apps für KMU. Praktisches Hintergrundwissen für Unternehmer.* Wiesbaden: Springer Gabler. essentials. ISBN 9783658105365.

Ullenboom, C., 2018. *Java ist auch eine Insel. Einführung, Ausbildung, Praxis.* 13., aktualisierte und überarbeitete Auflage. Bonn: Rheinwerk. ISBN 9783836258692.

Vaadin, o.J. *PWA pros and cons* [online] [Zugriff am: 19. September 2018]. Verfügbar unter: https://vaadin.com/pwa/learn/pros-and-cons.

Vollmer, G., 2017. *Mobile App Engineering. Eine systematische Einführung - von den Requirements zum Go Live.* Heidelberg: dpunkt.verlag. ISBN 9783864904219.

W3C, 2016. *Battery Status API* [online] [Zugriff am: 3. September 2018]. Verfügbar unter: https://www.w3.org/TR/battery-status/.

—, 2017a. *Media Capture and Streams* [online] [Zugriff am: 12. September 2018]. Verfügbar unter: https://www.w3.org/TR/mediacapture-streams/.

—, 2017b. *Permissions* [online] [Zugriff am: 30. September 2018]. Verfügbar unter: https://w3c.github.io/permissions/.

—, 2018a. *Web App Manifest* [online] [Zugriff am: 30. September 2018]. Verfügbar unter: https://www.w3.org/TR/appmanifest/.

—, 2018b. *Web NFC API* [online] [Zugriff am: 3. September 2018]. Verfügbar unter: https://w3c.github.io/web-nfc/.

Web Bluetooth Community Group, 2018. *Web Bluetooth* [online] [Zugriff am: 3. September 2018]. Verfügbar unter: https://webbluetoothcg.github.io/web-bluetooth/.

Wenz, C., 2014. *JavaScript. Das umfassende Handbuch ; [Grundlagen, Programmierung, Praxis ; für Einsteiger, Fortgeschrittene und Profis ; browserübergreifende Lösungen ; DOM, CSS, Ajax, XML, WebSockets ; inkl. HTML5 und jQuery].* 11. Aufl. Bonn: Galileo Press. Galileo Computing. ISBN 9783836219792.

Wolf, J., 2016. *HTML5 und CSS3. Das umfassende Handbuch.* 2., aktualisierte und erweiterte Auflage; 1., korrigierter Nachdruck. Bonn: Rheinwerk Verlag GmbH. Rheinwerk Computing. ISBN 9783836241588.