DEPARTMENT OF COMPUTER SCIENCE

Medieninformatik

Seminararbeit

Mobile App Development Frameworks

Eingereicht am:

06. Juli 2021

Eingereicht von:

Michael Smirnov

Windfeld 37

22559 Hamburg

Tel.: 0176 40022200

E-Mail: minf103430@fh-wedel.de

Betreuer:

Prof. Dr. Andreas Häuslein

Fachhochschule Wedel

Feldstraße 143

22880 Wedel

Tel.: (041 03) 80 48-42

E-Mail: hs@fh-wedel.de

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

SDK Software Development Kit

API Application Programming Interface

HTML Hypertext Markup Language

CSS Cascading Style Sheets

SASS Syntactically Awesome Stylesheets

HTTPS Hypertext Transfer Protocol Secure

PWA Progressive Web App

UI User Interface

UX User Experience

URL Uniform Resource Locator

JIT Just in Time

AOT Ahead of Time

DOM Document Object Model

VM Virtual Machine

1 Einleitung

Weltweit wurden im Jahr 2019 1.540.655 Smartphones verkauft. Diese Anzahl an verkauften Endgeräten folgt einem seit 2016 stabilen Trend von c.a. 1.500.000 vertriebenen Exemplaren jährlich. Aufgrund der weltweiten Covid-19 Pandemie sank die Anzahl an abgesetzten Smartphones global um 10.5% auf 1.378.719 Einheiten, jedoch wird für das Jahr 2021 prognostiziert, dass sich der Markt erholt[1][2]. In relativen Zahlen bedeutet dies, dass derzeit etwa 48% der Weltbevölkerung ein Smartphone besitzen. Im Verglich dazu betrug der Anteil der Smartphonebesitzer im Jahr 2016 33,5%[3].

Daraus ergibt sich, dass Smartphones sich hoher Beliebtheit erfreuen und inzwischen fest in unser alltägliches Leben integriert sind. Dies kann damit begründet werden, dass die Möglichkeit besteht eine Vielzahl verschiedener Apps zu installieren mit deren Hilfe das Gerät um eine Vielfalt an Funktionalitäten erweitert wird, die den Bedürfnissen und Wünschen des Nutzers entsprechen. Die verschiedenen Smartphones nutzen unterschiedliche Betriebssysteme, sodass eine App für die breite Masse mit den verschiedenen Plattformen kompatibel sein sollte, um eine möglichst große Anzahl an Nutzern zu erreichen.

Android und iOS stellen 99% des Marktanteils der mobilen Betriebssysteme dar. Dabei ist Googles Android der Marktführer mit 72% weltweit und iOS, Apples proprietäres Betriebssystem, mit ca. 27% auf dem zweiten Platz [4]. Somit stellen beide Betriebssysteme eine hinreichend große Verteilung dar, um im Folgenden berücksichtigt zu werden.

Bei Betrachtung der Relevanz von Betriebssystemversionen, fällt auf, dass Android Version 8 und höher auf 82% aller Geräte zum Einsatz kommt. Im iOS Ökosystem ist die älteste relevante Version iOS 12.4 mit c.a. 7% Anteil[5][6].

Die Verteilung der Betriebssysteme und deren verwendete Versionen spielen bei der Evaluation des Entwicklungssatzes eine wichtige Rolle. Aufgrund der Tatsache, dass nicht alle Nutzer sofort das Betriebssystem aktualisieren, entsteht die Notwendigkeit eine mobile App auf den am meisten verwendeten Betriebssystemversionen explizit zu testen. Zudem müssen teilweise mehrere Versionen derselben App gepflegt werden, um ein breites Spektrum der Betriebssystemversionen zu unterstützen. Dieser Support verbraucht wertvolle Entwicklerressourcen, sodass eine Abwägung der relevanten Versionen notwendig wird[7]. Ebenfalls relevant für einige Entwicklungsansätze sind die verschiedenen Web-Browser wie Firefox, Chrome oder Safari und deren Versionen. Auf die Eigenschaften und Unterschiede der verschiedenen Web-Browser wird jedoch im Laufe dieser Arbeit nicht näher eingegangen, da diese vor allem bei webbasierten Entwicklungsansätzen relevant sind, welche nur einen Teil dieser Seminararbeit ausmachen.

Ziel dieses Seminars ist es einen konzeptionellen Überblick über die derzeit vorherrschenden Mobile App Framework Technologien zu geben und auf die sich aktuell im Trend befindlichen näher einzugehen. Es soll sowohl ein rudimentäres Verständnis über die verschiedenen Technologien als auch eine Basis geschaffen werden, auf welcher aufbauend Entscheidungen für oder gegen einen gewissen Ansatz getroffen werden können soll.

2 Theorie der „Mobile App Development Frameworks“

2.1 Definitionen

2.1.1 Mobiles Endgerät

Im weiteren Verlauf wird der Begriff mobile oder mobiles Endgerät in Anlehnung an das National Institute of Standards and Technology (NIST) genutzt. Diese definieren ein mobiles Endgerät als ein kleines Datenverarbeitungsgerät, welches leicht von einer einzelnen Person getragen werden kann. Dieses Mobile Gerät ist darauf ausgelegt ohne physische Verbindung zu funktionieren. Es besitzt einen lokalen, nicht entfernbaren Datenspeicher und wird über einen längeren Zeitraum von einer verbauten Energiequelle betrieben. Mobile Geräte können auch Sprachkommunikationsfähigkeiten, eingebaute Sensoren, die es dem Gerät ermöglichen, Informationen zu erfassen (z.B. Fotografie, Videoaufnahme und Standortbestimmung) und/oder eingebaute Funktionen zur Synchronisierung lokaler Daten mit weiteren Standorten enthalten[8].

2.1.2 App/Applikation

Mobile Apps werden im Folgenden auch als App bezeichnet. Der Begriff App ist innerhalb kürzester Zeit in den Sprachgebrauch eingegangen und wird weithin als eigenständige Anwendung auf einem mobilen Endgerät verstanden. Apps werden von eigenständig dafür ausgelegten Plattformen heruntergeladen, welche von dem Hersteller des Betriebssystems zur Verfügung gestellt werden, oder direkt aus dem Internet bezogen. Beispiele dafür sind der App Store bei iOS-Systemen und der Google Play Store bei Android-Systemen[9].

2.1.3 Software-Framework

Der Begriff Mobile App Framework ist nicht einheitlich definiert und kann je nach Kontext auf eine andere Art und Weise verwendet werden. Im Folgenden wird der Begriff Framework folgendermaßen definiert:

Bei einem Software-Framework handelt es sich um eine Abstraktion von Details, welche notwendig wären, um ein bestimmtes Ziel zu erreichen. Ein Framework stellt dem Entwickler eine gewisse Grundfunktionalität zur Verfügung, auf welcher aufgebaut werden kann, um eigene Software zu schreiben. Grundbausteine von Frameworks sind abstrakte oder konkrete Klassen, welche beerbt erweitert oder direkt verwendet werden können.

Frameworks lassen sich anhand der Kontrolle des Programmablaufes von einer Library abgrenzen. Während Frameworks die Kontrolle über den Programmablauf behalten und an bestimmten Stellen den vom Anwendungsentwickler eingefügten Code ausführen, stellen Librarys eine Ansammlung von Funktionen dar, welche vom Code des Anwendungsentwicklers aus aufgerufen werden können. Jedoch ist der Übergang von Framework zu Library fließend und hängt auch stark von dem Kontext ab in welchem der Begriff verwendet wird.

Martin Fowler definiert den Unterschied folgendermaßen:

> „Inversion of Control is a key part of what makes a framework different to a library. A library is essentially a set of functions that you can call, these days usually organized into classes. Each call does some work and returns control to the client. A framework embodies some abstract design, with more behavior built in. In order to use it you need to insert your behavior into various places in the framework either by subclassing or by plugging in your own classes. The framework's code then calls your code at these points.“ [10]

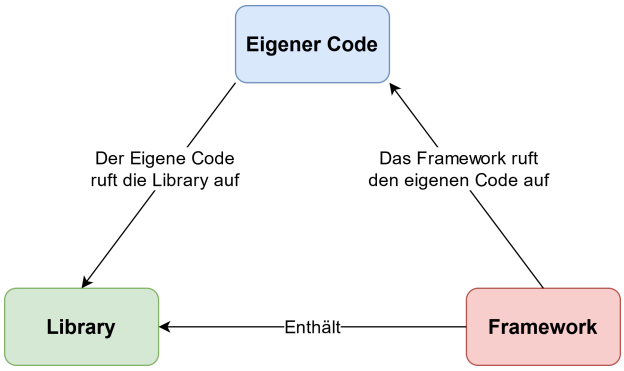

Abbildung 1: Zusammenhang zwischen einer Library und einem Framework

Zusätzlich zu der Unterscheidung zwischen Frameworks und Librarys existieren noch sogenannte „opinionated“ und „non-opinionated“ Frameworks bzw. Librarys. Bei opinionated Frameworks werden gewisse Vorgaben gemacht, z.B. wie die Ordnerstruktur des Projekts aussehen sollte oder wie gewisse Funktionen zu implementieren sind. Es wird ein Rahmen bereitgestellt, in welchem Beziehungen und Architektur der Komponenten vorgegeben werden. Abweichungen von diesem sind, je nachdem wie strikt diese durchgesetzt werden, schwer und nur mit viel Aufwand möglich. Non-opinionated Frameworks hingegen schreiben solch einen Rahmen nicht vor. Der Entwickler muss sich selber um bestimmte Aspekte wie z.B. den Anwendungszustand kümmern. Dies sorgt dafür, dass die Flexibilität auf Kosten der Komplexität erhöht wird[11].

Ein Mobile Application Development Framework ist demnach ein Framework, welches die Struktur bereitstellt, eine Anwendung zu bauen, welche auf einem Mobilgerät ausgeführt werden soll.

2.2 Einsatzgebiete und Gründe für die Nutzung eines Mobile App Development Frameworks

Die Gründe ein Framework in einem bestimmten Projekt zu nutzen sind sehr vielfältig und meist sehr individuell, da das Framework meist nur ein Hilfsmittel ist, um die Anforderungen an das Projekt zu erfüllen.

Bei Applikationen, dessen Zielgruppe Endbenutzer sind, besteht wie eingangs erwähnt die Notwendigkeit verschiedene Plattformen zu unterstützen, um eine größtmögliche Anzahl an Benutzern sicherzustellen. Dabei ist es finanziell und organisatorisch vorteilhaft nur eine einzige Codebasis pflegen zu müssen. Außerdem wird somit Feature, User Experience (UX) und User Interface (UI) Gleichheit sichergestellt.[12]

Die Herausforderung für mobile Endgeräte zu entwickeln, kommt nicht nur aus der Notwendigkeit für zwei unterschiedliche Plattformen mit verschiedenen Versionen zu entwickeln und diese über einen gewissen Zeitraum zu unterstützen, sondern auch durch die aufgesplitterte Android-Landschaft. Da Google Android als Betriebssystemplattform entwickelt, können Hersteller auf dieser aufbauend beliebige Modifikationen und Erweiterungen vornehmen[13]. Dies sorgt dafür, dass eine Vielzahl an unterschiedlichen Hard- und Softwarekombinationen existieren. Diese nutzen zwar alle Android als Plattform, das eigentliche Betriebssystem und die Hardware, auf der es läuft, unterscheiden sich trotzdem teilweise stark voneinander. 2015 gab es über 24.000 verschiedene Geräte, welche auf Android basierten von etwa 1.300 verschiedenen Herstellern[14].

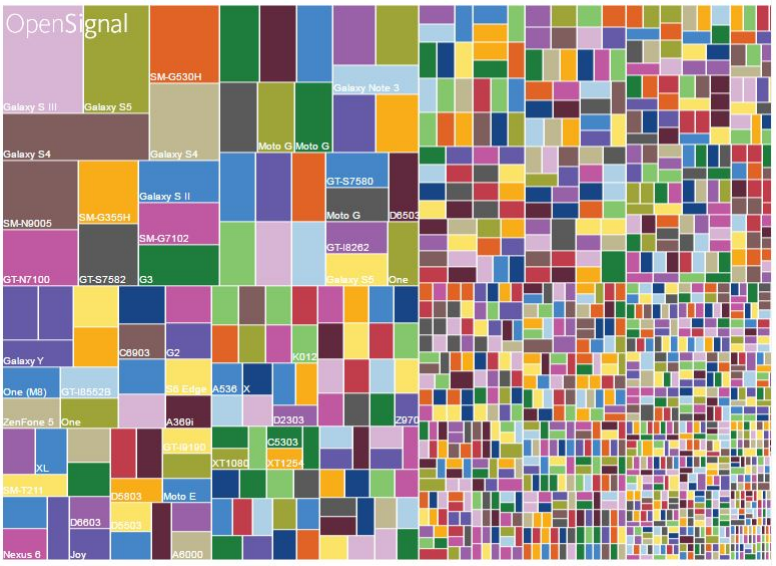

Abbildung 2: Visualisierung der Hardware-Fragmentierung von Android-Geräten [14]

Hinzu kommt noch der Trend, dass immer mehr verschiedene Gerätetypen Android als Betriebssystem nutzen und dem User ebenfalls ermöglichen Apps auf diesen zu installieren. Zu den bereits bekannten Smartphones und Tablets sind in den letzten Jahren vermehrt Smartwatches, Autos und Smart TV Fernseher dazugekommen, welche ein Android-Basiertes Betriebssystem nutzen. Dies hat zur Folge, dass zusätzlich verschiedene Konnektivitätseigenschaften der Endgeräte berücksichtigt werden müssen. Zudem verschwimmen die Grenzen zwischen den Gerätetypen immer mehr. Jedes dieser verschiedenen Gerätetypen birgt Besonderheiten und Limitierungen, was die Entwicklung einer Anwendung, welche mit mehreren Typen kompatibel sein soll, teilweise stark verkomplizieren. Mit einigen der im Folgenden erwähnten Frameworks kann solch ein breites Spektrum an Endgeräten abgedeckt werden, jedoch wird im weiteren Verlauf der Arbeit näher darauf eingegangen, da diese Endgeräte teilweise der Definition von einem mobilen Endgerät nicht entsprechen und eine Berücksichtigung dieser den Rahmen sprengen würde.

Des Weiteren können Gründe für die Nutzung bestimmter Frameworks auch organisatorischer Natur sein und müssen nicht zwingend technisch begründet sein. Beispiele dafür können die Erfahrung und Größe des Entwicklungsteams, die Größe des Projekts, Vorgaben der Organisation, Lizenzen, verfügbares Budget und noch viele Weitere sein. Generell wird eine immer kürzere und effizientere Entwicklung von Software angestrebt, was eine Reduzierung von Entwicklungszeit und somit Kosten zur Folge hat. Diese Trend hat die Popularität der cross-plattform Entwicklung stark beeinflusst.

3 Kategorisierung der verschiedenen Ansätze zur Entwicklung einer Mobile App

Es existieren viele verschiedenste Kategorisierungsansätze zur Entwicklung einer mobilen App. Im Folgenden wird die Kategorisierung nach Majchrzak verwendet[15]. Dieser unterteilt die Entwicklung in drei unterschiedliche Ansätze, wobei auch Mischformen wie React Native, NativeScript und Flutter existieren, bei denen der Übergang zwischen den folgenden Kategorien fließend ist[12]. Dabei wird ein Augenmerk auf die Nutzung nativer Elemente der Zielplattform gelegt, welche auf der horizontalen Achse der Abb. 3 gekennzeichnet wurden.

1. Native, plattformspezifische Entwicklung.
2. Anwendungen welche in einer Laufzeitumgebung ausgeführt werden. Diese werden wiederum in (Progressive) Web-Apps, Hybrid-Apps und Apps, welche in ihrer eigenen Laufzeitumgebung ausgeführt werden, unterteilt.
3. Generierende Ansätze, welche entweder auf dem Ansatz der modellgetriebenen Softwareentwicklung (Model Driven Software Development) oder dem des Cross- bzw. Transpilings basieren. Das Resultat ist dabei in jedem Fall eine native App, welche auf das Software Development Kit (SDK) des Herstellers aufbaut.

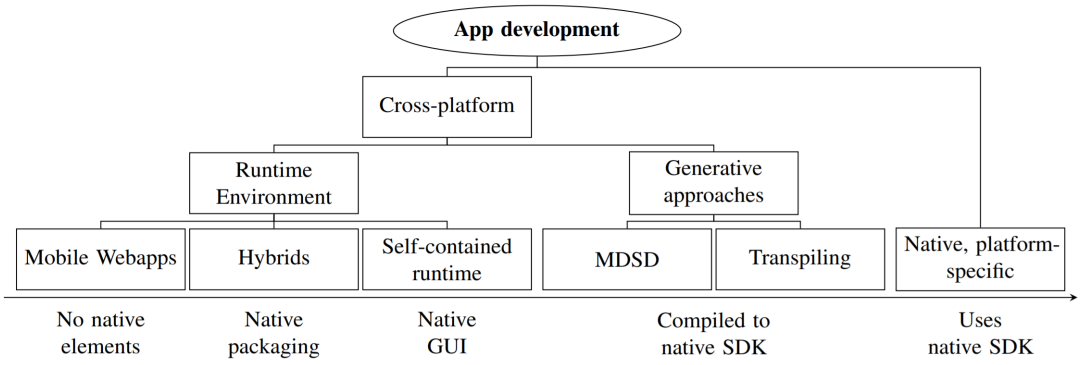

Abbildung 3: Kategorisierung von cross-plattform Entwicklungsansätzen [15]

3.1 Native App-Entwicklung

Die Native App-Entwicklung stellt zwar kein Framework dar, ist jedoch trotzdem wichtig zu erwähnen, da dies die ursprüngliche Art für ein mobiles Gerät zu entwickeln war und eine Referenz zu den folgenden Ansätzen darstellen soll.

Bei der nativen Entwicklung wird die Applikation mit den Technologien, Programmiersprachen und Tools entwickelt, welche von dem Hersteller der Zielplattform zur Verfügung gestellt wurden. Dafür wird ein SDK genutzt, um die gewünschten Funktionen der Hardware zu benutzen und die passenden UI-Elemente zu rendern und auf Eingaben des Nutzers zu reagieren. Jedoch ist der Quellcode an die jeweilige Zielplattform gebunden, da es sich um plattformspezifischen Bytecode handelt.

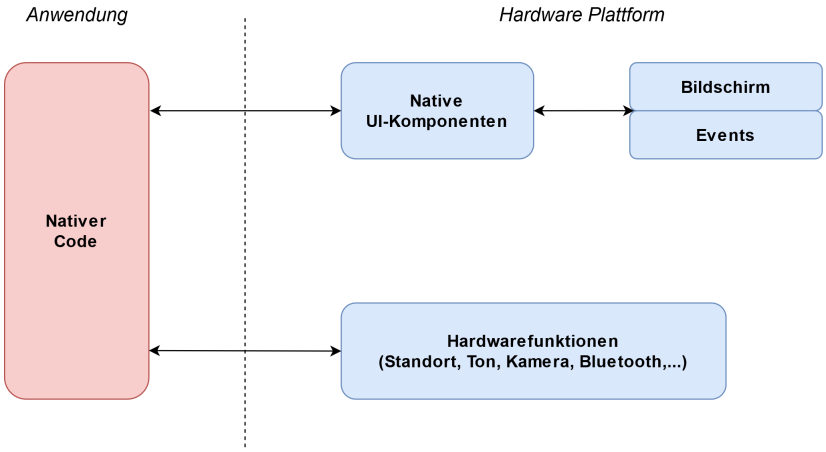

Abbildung 4: Vereinfachtes Schema einer nativen Anwendung

Somit kann eine native Android Anwendung nur in Java oder Kotlin geschrieben werden, um zusammen mit dem Android SDK die notwendige Funktionalität zu gewährleisten, während bei iOS Objective-C oder Swift benutzt werden muss, um zusammen mit Apples SDK eine native Applikation zu erstellen[12].

Die Native App wird anschließend im App-/Playstore des jeweiligen Betriebssystems veröffentlicht und kann von dort aus vom User heruntergeladen werden. Updates werden ebenfalls über diese Plattform verwaltet und dem User ausgespielt.

Vorteile	Nachteile
Zugang zu allen Features der Zielplattform über die SDK API. Bestmögliche Performance auf dem Zielgerät, verglichen mit anderen Entwicklungsansätzen. Natives „Look and Feel“ der UI-Komponenten. Sofortiger Zugriff auf alle neuen Features der jeweiligen Zielplattform, sobald diese veröffentlicht werden.	Aufgrund der plattformspezifischen Tools muss dieselbe Applikation gleich zwei Mal entwickelt werden, was zu höherem Aufwand, längerer Entwicklungszeit, Notwendigkeit von zwei Entwicklungsteams mit Kenntnissen in verschiedenen Technologien und somit zu höheren Kosten führt. Native Apps sind komplexer in der Entwicklung und benötigen einen gewissen Grad an Erfahrung[16] Einschränkungen und zusätzlichen Kosten, welche bei der Entwicklung und Veröffentlichung auf bestimmten Verteilungsplattformen entstehen. Da dies jedoch für jeden Ansatz gilt, welcher die App über den App-/Playstore verteilt wird dieser Aspekt im Folgenden nicht mehr explizit genannt. (Zugang zu Apples kostenpflichtigen Developer Program[17] und die Notwendigkeit eines Review-Prozesses von Apple, um eine App in dem App Store veröffentlichen zu können[18])

Tabelle 3.1: Vor- und Nachteile der nativen App-Entwicklung

3.2 Auf einer Laufzeitumgebung basierende Ansätze

Bei dieser Oberkategorie an mobile App Frameworks wird eine Laufzeitumgebung genutzt, um eine übergeordnete Abstraktion der darunterliegenden Plattform zu schaffen. Somit können Anwendungen, welche darauf aufbauen eine einheitliche Schnittstelle verwenden und können somit Plattformübergreifend genutzt werden. Diese Ansätze werden anhand der dafür genutzten Laufzeitumgebung weiter unterteilt.

3.2.1 (Progressive) Web-App

„Klassische“ Web-App

Eine Web-App stellt eine Art einer Anwendung dar, welche innerhalb eines Browsers auf einem mobilen Gerät ausgeführt wird und mithilfe von Web-Technologie wie Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) und JavaScript erstellt wurde. Diese sind über einer Uniform Resource Locator (URL) erreichbar und basieren, wie jede Website, auf dem klassischen Client-Server-Modell. Somit ist die Web-App komplett betriebssystemunabhängig, da lediglich ein aktueller Browser und eine Internetverbindung notwendig sind, um diese auf einem Endgerät zu nutzen. Dieser stellt somit die Laufzeitumgebung bereit, in der die Applikation ausgeführt wird. Dieser kümmert sich um die Darstellung und Verarbeitung von Inputs des Benutzers. Dabei werden keine nativen Features, welche über die Möglichkeiten des Browsers verwendet und es werden ebenfalls keine nativen UI-Komponenten benutzt.[19].

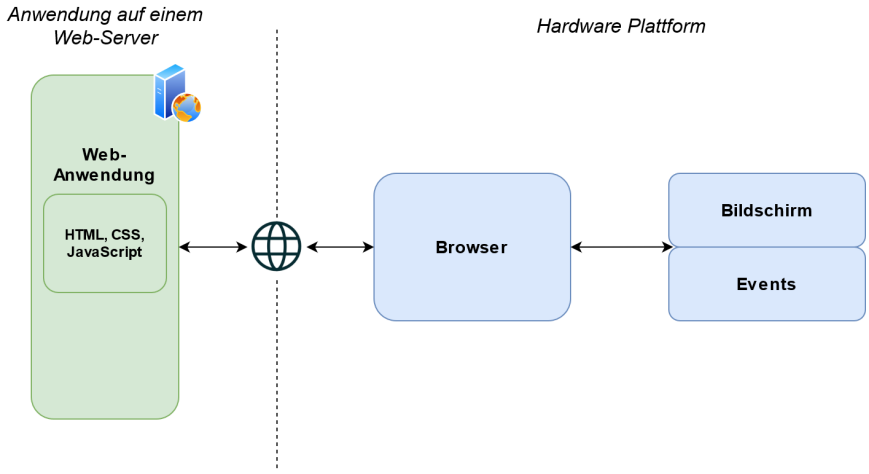

Abbildung 5: Vereinfachtes Schema einer Web-Anwendung

Vorteile	Nachteile
Die Verarbeitung geschieht auf einem Web-Server, sodass nur die UI-Elemente an das Endgerät geschickt werden.	Sind nicht im App-/Playstore verfügbar.
Die Wartung der Anwendung ist leichter, da diese nur auf dem Web-Server liegt.	Eine Internetverbindung ist zwingend erforderlich.
Dieselbe Anwendung kann direkt in Browsern verschiedener Endgeräte angezeigt werden.	Schwächere Performance, da in einem Browser ausgeführt.
	Browserkompatibilität nicht überall gewährleistet.
	Keinen Zugriff auf alle Hardwarefunktionen, sondern sind auf die des Browsers beschränkt.

Tabelle 3.2: Vor- und Nachteile einer „klassischen“ Web-App

Progressive Web-App

Der Name Progressive Web-App ist kein allgemein definierter Begriff. Dieser wurde 2015 von dem Google Entwickler Alex Russell eingeführt, um eine sich an das Endgerät anpassende Web-Applikation zu beschreiben, welche ausschließlich aus Web-Technologien wie unter anderem JavaScript, CSS und HTML besteht. Zusätzlich erweitert die PWA den Funktionsumfang einer Web-App um Verhaltensweisen und Features, welche Traditionell den nativen Apps vorbehalten waren. Dazu gehören u.A. das Hinzufügen zu dem Homescreen, eine Möglichkeit die Website auch ohne Internet zu bedienen, und Push-Benachrichtigungen zu erhalten[20]. Besonders hervorzuheben ist die Tatsache, dass der User selbst entscheidet, ob diese Features genutzt werden sollen oder nicht.
„ A PWA is a Website running in a browser that will progressively add more features based on compatibility. “ [21]
Dabei stellt die Progressive Web-App keine einzige Technologie dar, sondern eher eine Ansammlung an Eigenschaften, welche erfüllt werden sollten, um als eine PWA bezeichnet werden zu können. Die Kriterien welche Google für eine PWA nennt sind in zwei Kategorien aufgeteilt. Kernanforderungen an die Website und Zusatzanforderungen welche eine „optimale“ PWA ausmachen.

Kernanforderungen	Zusätzliche Anforderungen
Geschwindigkeit	Barrierefreiheit
Erreichbarkeit von jedem Browser und Endgerät	Sicherheit
Responsive Anpassung an die Bildschirmgröße	Sichtbarkeit (z.B. über Googles Suchfunktion)
Offline-Funktionalität	Transparenz bezüglich der erforderlichen Berechtigungen
Installierbarkeit	

Tabelle 3.3: Anforderungen an eine PWA[22]

Zu den technischen Bestandteilen, welche eine PWA von einer Web-App unterscheiden gehören:

1. Die App Shell - Stellt das minimale HTML, CSS und JavaScript Gerüst dar, in welchem der Content der PWA angezeigt werden soll. Sie ist dafür zuständig die statischen Inhalte der Anwendung anzuzeigen, welche im Nutzungsverlauf der PWA gleich bleiben. Dazu gehört üblicherweise die Navigationsleiste oder die Homepage. Diese werden lokal auf dem Endgerät abgelegt, sodass Ladezeiten minimiert werden können.

2. Der Service Worker - Ein Hintergrunddienst in JavaScript geschrieben, welcher auch bei geschlossener App weiterläuft. Dieser wird beim Abruf der PWA mitgeliefert und genutzt, falls die PWA „zum Homescreen hinzugefügt“ wird. Er hat die Aufgabe Netzwerkanfragen abzufangen und passend zu verarbeiten. Außerdem verwaltet der Service Worker den Cache, wodurch Anfragen auch ohne Internet verarbeitet und beantwortet werden können, um die offline Funktionalität zu ermöglichen. Laut Biørn-Hansen et. al., 2017[23] kann eine PWA ohne Service Worker nicht richtig funktionieren. Damit ist der Service Worker der zentrale Bestandteil einer PWA.

3. Das Web Application Manifest - Eine Datei, welche es dem Entwickler ermöglicht das Verhalten und Aussehen der „installierten“ PWA auf dem mobilen Gerät des Users anzupassen. Dazu gehören u.A. das angezeigte Logo auf dem HomeScreen oder verschiedenen Caching-Strategien.

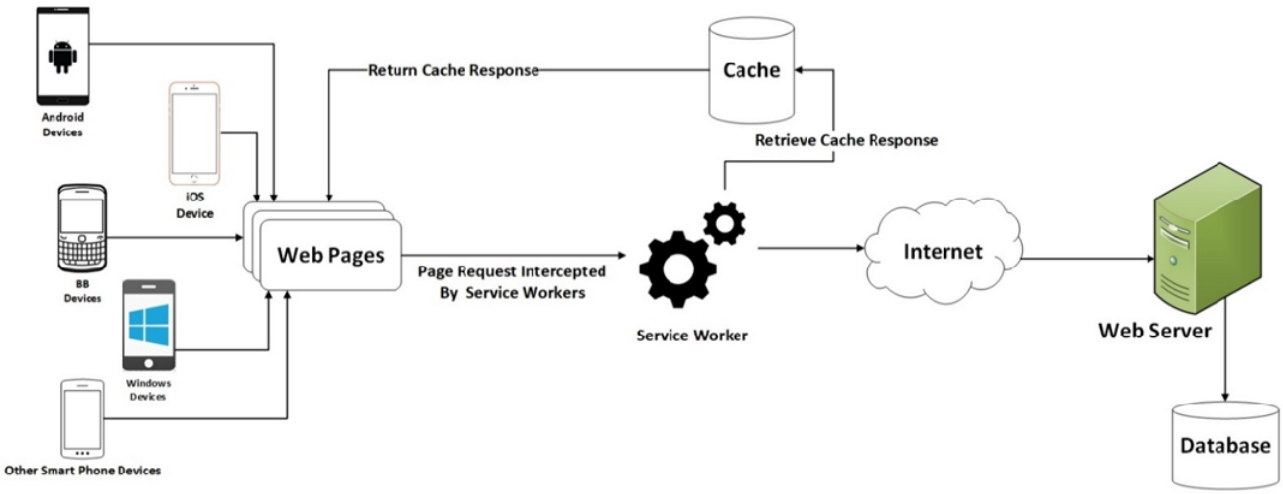

Abbildung 6: Technischer Aufbau einer PWA [24]

Vorteile	Nachteile
Eine PWA lässt sich auch ohne Internetverbindung benutzen.	PWAs haben keinen Zugriff auf alle Hardwarefunktionen, sondern sind auf die des Browsers beschränkt.
Die PWA ist leicht erreichbar und teilbar, da diese über einen Link im Browser abgerufen werden kann.	Sind nicht im App-/Playstore verfügbar.
Es ist möglich bereits bestehende Webseite zu einer PWA umzuwandeln.	Schwächere Performance, da in einem Browser ausgeführt.
Funktionen wie „zum Homescreen hinzugefügen“ oder Push-Benachrichtigungen führen dazu, dass der Nutzer die Web-App als native Anwendung wahrnimmt und anders mit dieser interagiert[25].	PWAs sind derzeit noch nicht auf allen Geräten und Browsern komplett unterstützt.
Eine PWA benötigt zwingend HTTPS, was die Datenübertragung vor „man-in-the-middle“ Attacken zusätzlich schützt.	

Tabelle 3.4: Vor- und Nachteile einer PWA[24]

3.2.2 Hybride App

Hybride Apps stellen eine Kombination von Web-Apps und nativen Apps dar. Diese verwenden, ebenfalls vorwiegend Web-Technologien. Der Quelltext wird ebenfalls innerhalb eines Browsers ausgeführt. Jedoch handelt es sich dabei nicht um den bereits installierten Browser wie bei Web-Apps, sondern um einen plattformspezifischen Container (UIWebView unter iOS und WebView unter Android)[16]. Dieser wird bei dem Download der App aus dem App-/Playstore zusammen mit dem Quellcode der App im Hintergrund bereitgestellt und stellt die Laufzeitumgebung dar. Die UI-Komponenten werden weiterhin von dem Container gerendert, obwohl manche Frameworks versuchen diese möglichst nativ aussehen zu lassen. Um die Limitierungen des Zugriffes auf bestimmte Hardware oder Informationen aus dem Browser heraus zu umgehen, wird mithilfe eines "Überbrückungs-Moduls" auf den nativen Teil der Anwendung zugegriffen. Dieser hat Zugriff auf die nativen APIs der Plattform. Dies führt, je nachdem wie aktive diese verwendet wird, zu einem Performance-Engpass, welcher in Abschnitt 4.3 näher erläutert wird. Jedoch ist dieser Teil der Hybriden App wiederum plattformspezifisch. Allerdings stellen cross-plattform Frameworks diese Funktionalität direkt bereit und nehmen dem Entwickler viel Arbeit ab, sodass es bei der Entwicklung selten notwendig ist nativen Code selbst zu schreiben[26].

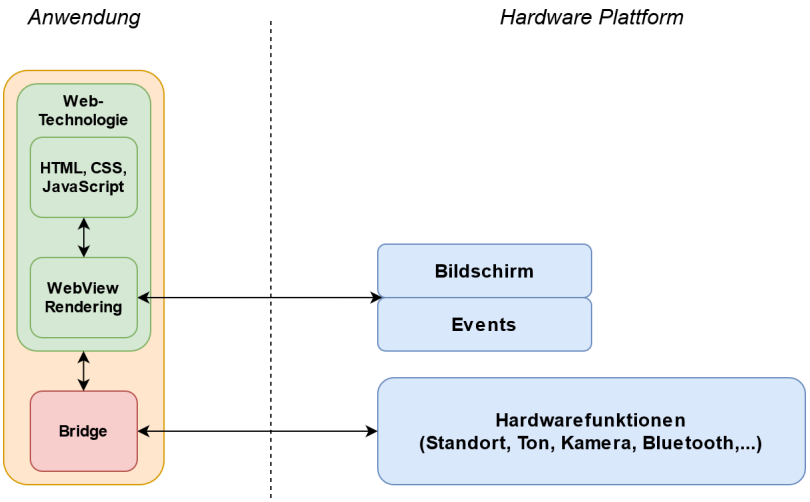

Abbildung 7: Vereinfachtes Schema einer hybriden App

Vorteile	Nachteile
Kann über den App-Store verteilt werden, sowie als Web-App bereitgestellt werden.	Schwächere Performance, da die App innerhalb einer Browser-Engine ausgeführt wird und manipulationen des DOMs viel Zeit kosten.
Für die Implementierung und Tests kann größtenteils der Desktop-Browser genutzt werden.	Häufige Nutzung nativer Funktionen kann ebenfalls die Performance beeinflussen.
Hat Zugriff auf vom Framework unterstützten Hardwarefunktionen.	Mehr Fehlerquellen durch Nutzung verschiedener Technologien, welche sich je nach Plattform unterschiedlich verhalten kann.
Die Anwendung hat auf verschiedenen Endgeräten ein einheitliches UI/UX, da der Inhalt mit Web-Technologie definiert und in einem browserähnlichem Container dargestellt wird.	Kein Zugriff auf native UI-Elemente.
Eine bereits bestehende Webseite kann in kurzer Zeit zu einer hybriden App umgewandelt werden.	

Tabelle 3.5: Vor- und Nachteile einer hybriden App

3.2.3 Eigenständige Laufzeitumgebung

Im Gegensatz zu der (progressiven) Web-App und dem hybriden Ansatz greift diese Art der Anwendung nicht mehr auf die üblichen Web-Technologien zurück und rendert den Inhalt nicht innerhalb eines Browsers oder browserähnlichen Containers, sondern stellt eine vollständig eigenständige Laufzeitumgebung zur Verfügung. Somit wird es den Entwicklern der Applikation ermöglicht Programmiersprachen wie JavaScript (z.B. React Native) oder C# (z.B. Xamarin) zu verwenden. Um das Übersetzen der Aufrufe aus dem plattformunabhängigen Quellcode in das passende native Äquivalent kümmert sich ein "Überbrückungs-Modul" oder API des Frameworks. Dies führt jedoch, je nach Verwendungshäufigkeit, ebenfalls zu einem Performance-Engpass, welcher in Abschnitt 4.3 näher erläutert wird. Die Laufzeitumgebung deckt somit einen größeren Aufgabenbereich als der hybride Ansatz ab und ist somit flexibler, da hierbei das Rendern der UI mit nativen Elementen realisiert werden kann.[27]

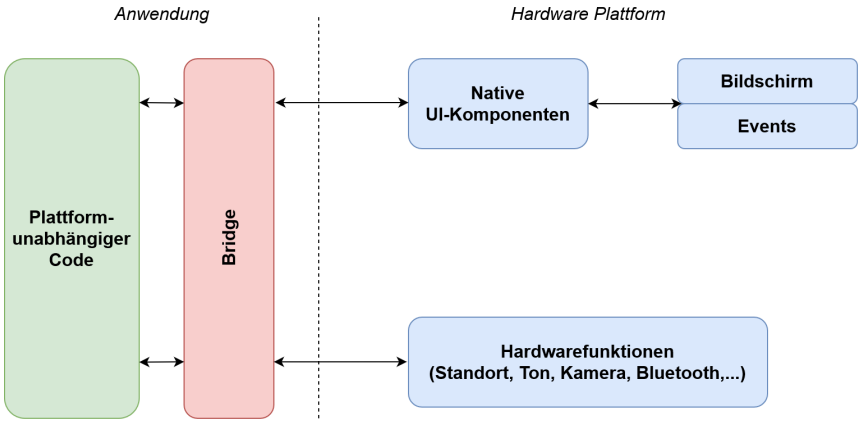

Abbildung 8: Vereinfachtes Schema einer App mit eigener Laufzeitumgebung

Vorteile	Nachteile
Es besteht kein Zwang Web-Technologien verwenden zu müssen.	Komplexe Frameworks, da diese mehrere verschiedene Aufgaben übernehmen.
Die Anwendung kann über den App-/Playstore verteilt werden.	Je nach Kenntnisstand höherer Einarbeitungsaufwand.
Es können native UI-Elemente verwendet werden.	Performance leidet unter der Notwendigkeit mit nativen Komponenten über die Bridge kommunizieren zu müssen.
Es sind vom Framework unterstützten Hardwarefunktionen verfügbar.	Initiale Startzeit der App ist hoch, da aufgrund der erhöhten Komplexität eine größere Codebasis interpretiert und JIT kompiliert werden muss.

Tabelle 3.6: Vor- und Nachteile einer App mit eigenständiger Laufzeitumgebung

3.3 Generierende Ansätze

Bei diesen Ansätzen wird es dem Entwickler ermöglicht eine Applikation einmalig zu entwickeln und daraus nativen Code für die jeweilige Zielplattform zu generieren. Dabei wird zwischen modellgetriebenen und cross-kompilierenden Ansätzen weiter unterschieden.

3.3.1 Modellgetrieben

Bei diesem Ansatz wird die Applikation in einer Modellierungssprache (Textuell oder Grafisch) plattformunabhängig definiert. Hierbei kann wiederum in zwei Gruppen unterteilt werden: Mehrstufig und einstufig generierende Ansätze.

Bei der Einstufigen Code Generation wird aus der Modellierungssprache mit Hilfe von mehreren Codegeneratoren nativer Quellcode für die jeweilige Zielplattform erzeugt. Beispiele dafür wären MD2, MobML oder Mobl.

Bei der Mehrstufigen Code Generation wird Quellcode für ein weiteres Cross-Plattform Framework erzeugt. Daraus wird in einem weiteren Schritt wiederum Quellcode für die jeweilige Zielplattform erzeugt. Ein Beispiel für solch ein Framework wäre AXIOM.

Diese Ansätze sind jedoch als Spezialfälle zu betrachten und werden vorwiegend akademisch Diskutiert. Kommerzielle Ansätze existieren haben jedoch alle eigene Modellierungssprachen, wodurch sich diese Ansätze nicht durchsetzen konnten[29][27].

Vorteile	Nachteile
Definition der Applikation auf einem sehr hohen und abstrakten Level möglich.	Zersplitterte Framework-Landschaft.
Keine Notwendigkeit Code zu schreiben.	Fehleranfällig, da viele verschiedene Technologien aufeinander aufbauen.
Sehr hohe Entwicklungsgeschwindigkeit bei Entwicklung von Prototypen möglich.	Können häufig nur die Features anbieten, welche auf allen Plattformen verfügbar sind.
	Kleine Community und wenig Hilfsmaterialien, daher vorwiegend akademisch Diskutiert.

Tabelle 3.7: Vor- und Nachteile einer modellgetriebenen App

3.3.2 Cross-Kompilierend

Bei diesem Ansatz wird versucht den geschrieben Quellcode der Anwendung in eine, für die Zielplattform passende Repräsentation umzuwandeln. Dies kann auf verschiedenen Abstraktionsebenen des Quellcodes geschehen. Daher wird in der Literatur der Begriff kompilierend oder transpilierend benutzt. Bei der Kompilation wird meist Quellcode einer niedrigeren Abstraktionsstufe erzeugt wird, während bei der Transpilation Quellcode häufig auf ähnlicher Abstraktionsstufe erzeugt wird. Diese Trennung ist jedoch nicht einheitlich definiert und ist daher häufig kontextabhängig. Allerdings fokussieren sich solche Ansätze die Regel nur auf einen bestimmten Aspekt der Anwendung, sodass noch zusätzliche Schritte notwendig sind, um eine vollständige App zu erzeugen. Ein Beispiel dafür ist Googles J2ObjC [27]. Besonders hervorzuheben ist dabei Googles Flutter Framework, bei welchem die Applikation in Dart geschrieben wird. Dieses ist der jüngste Vertreter dieser Kategorie. Eine Besonderheit bei Flutter besteht darin, dass keine Nativen Komponenten gerendert werden. Das Framework überlässt diese Aufgabe komplett der mitgelieferten Skia Graphics Engine. Daraus folgt eine erhöhte Flexibilität und Performance[28].

Vorteile	Nachteile
Ähnliche Performance zu nativen Apps möglich.	Es werden, je nach Framework, nur eine Begrenze Anzahl an Plattformen unterstützt.
Es können native UI-Elemente verwendet werden.	Es werden nur Features unterstützt, welche auf allen Plattformen verfügbar sind.
	Legen den Fokus teilweise nur auf einen bestimmten Aspekt der Anwendung, sodass noch weitere Schritte notwendig sind, um eine vollwertige App zu erzeugen.

Tabelle 3.8: Vor- und Nachteile einer cross-kompilierenden App

4 Framework Beispiele

4.1 Auswahlkriterien

Im Folgenden wird auf einige ausgewählte Frameworks konkret eingegangen, welche auf verschiedenen Ansätzen beruhen. Dabei wurde der native Ansatz nicht mit in die nähere Betrachtung aufgenommen, da dieser nicht plattformunabhängig ist. Ebenso wurden PWAs nicht weiter berücksichtigt, da deren Möglichkeiten Browserabhängig sind. Zunächst wurde eine Literaturrecherche durchgeführt bei der folgende Frameworks häufig genannt wurden:

- Titanium
- Sencha Touch
- Corona SDK
- PhoneGap
- Native Script
- Xamarin
- Ionic
- React Native
- Flutter

Um die Relevanz und Aktivität der Community jener Frameworks zu bewerten, wurde das StackOverflow-Trends Tool [1] genutzt. Dieses zeigt die relative Verteilung der gestellten Fragen zu bestimmten Technologien im Zeitverlauf.

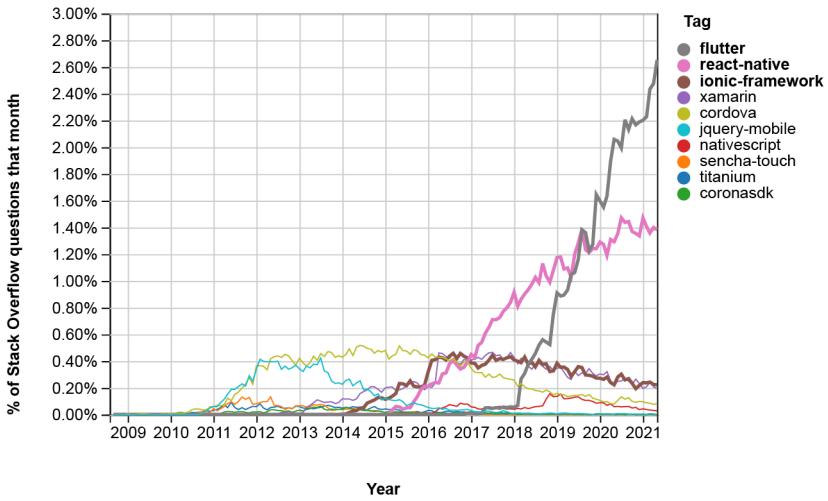

Abbildung 9: Verteilung der gestellten Fragen auf StackOverflow ausgewählter Frameworks

Auf dieser Grundlage wurden die drei populärsten Frameworks ausgewählt auf die im Folgenden näher eingegangen wird.

[1] https://insights.stackoverflow.com/trends

4.2 Das Hybride Framework Ionic

Ionic ist ein Open Source cross-plattform Framework welches 2013 von dem Startup Drifty veröffentlicht wurde. Eine Ionic App nutzt Web-Technologien wie HTML, CSS, TypeScript und baut auf AngularJS und Apache Cordova auf[30].

Es kann flexibel für hybride Apps und PWAs verwendet werden. Der Fokus von Ionic liegt auf dem Frontend. Zunächst aufbauend auf dem Angular Framework stellt Ionic UI- Komponenten für z.B. Listen, Buttons, Layouts bereit, welche den nativen Gegenstücken nachempfunden sind. Technisch bildet dabei Apache Cordova die Schnittstelle für plattformspezifische Funktionalitäten als auch die Grundlage, um plattformunabhängige Applikationen erstellen zu können. Seit Version 4 ist Ionic nicht mehr fest an AngularJS gekoppelt und es können beliebige Front-End-Frameworks wie React oder Vue benutzt werden, sowie Ionics eigene Komponenten, ohne spezifisches Front-End-Framework. Außerdem kann statt Apache Cordova Ionics eigene Laufzeitumgebung Capacitor genutzt werden, um die Anwendung auf einem Endgerät auszuführen[31][32].

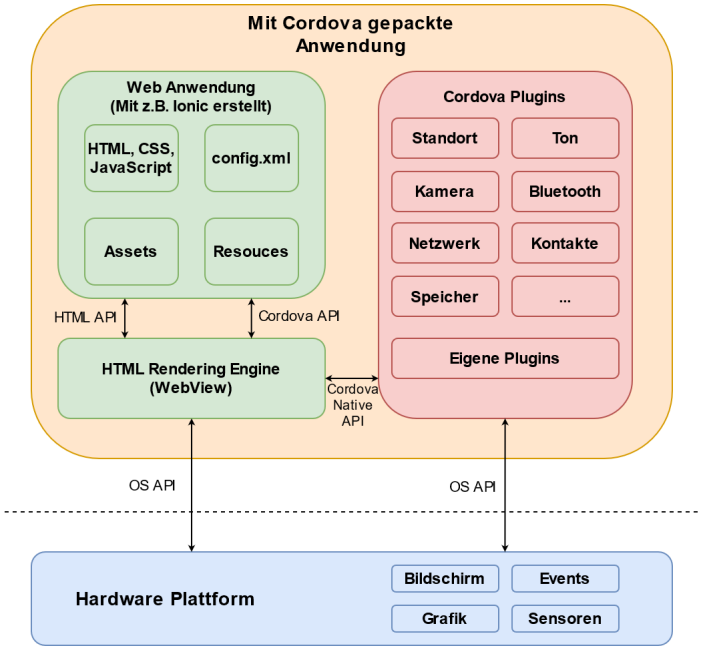

Abbildung 10: Übersicht einer mit Cordova gepackten App (Anlehnung an [33])

Die Komponenten der Applikation werden mit HTML definiert, mit Syntactically Awesome Stylesheets (SASS), einem Superset von CSS mit Zusatzfunktionen, gestyled und deren verhalten mit JavaScript oder optional mit TypeScript definiert. Die Sprache TypeScript wurde von dem Angular Framework übernommen und stellt ein Superset von JavaScript dar, welche einige zusätzliche Sprachfeatures als auch ein Typsystem bietet. Die mit TypeScript definierten Komponenten werden zu JavaScript tanspiliert, sodass Typfehler während des Kompilierens identifiziert werden können. Die mit Ionic gebauten Anwendungen können über die Plattform spezifischen App-Stores ausgeliefert, oder als Web-App verfügbar gemacht werden.

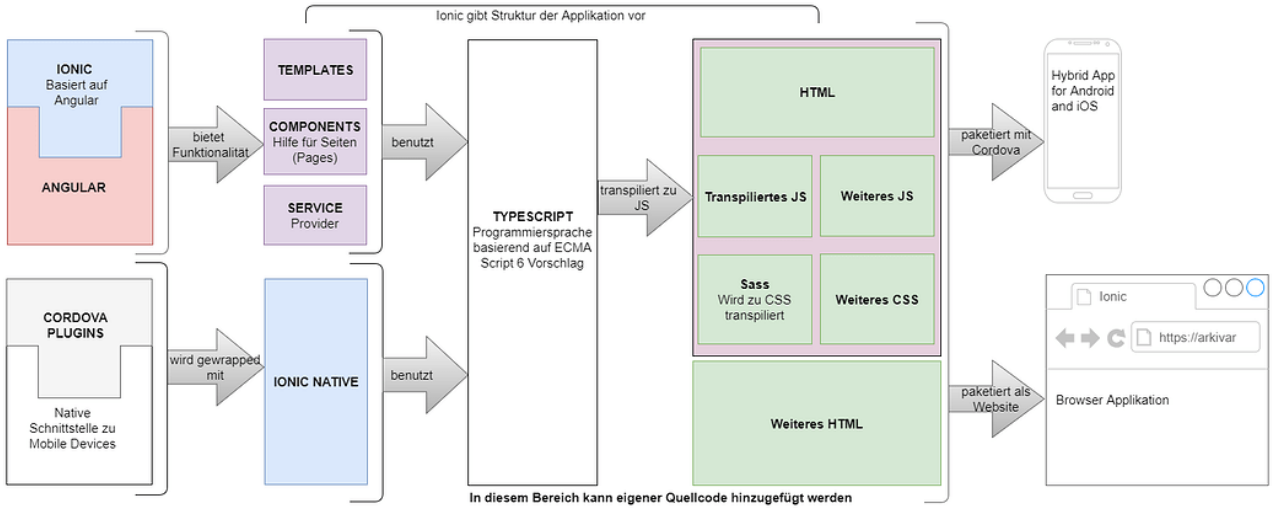

Abbildung 11: Übersicht der verwendeten Technologien einer Ionic App[34]

Werden bei der Entwicklung die angebotenen Komponenten genutzt, entsteht ein einheitliches Design, zudem ein umfassendes Theming viele Aspekte der Gestaltung bereits abnimmt. Jedoch kann dies auch zu einem Nachteil werden, vor allem dann, wenn Abweichungen von den Standardkomponenten gefordert sind. Ebenfalls als Nachteil ist hierbei die Performance von Ionic zu nennen, da im Gegensatz zu anderen Frameworks welche native UI-Komponenten verwenden, diese hierbei im WebView gerendert werden[35].

4.3 React Native - Framework mit eigenständiger Laufzeitumgebung

React Native wurde 2013 während eines Facebook internen Hackathon entwickelt und im Mai 2015 veröffentlicht, wobei zunächst nur iOS unterstützt wurde. Die Android-Unterstützung folgte wenige Monate später. React Native baut auf der API des populären und ebenfalls von Facebook entwickelten JavaScript Framework React auf. Dieses wurde 2013 für Web-Anwendungen entwickelt und legt den Fokus auf die Darstellung von Komponenten abhängig von den zur Verfügung gestellten Daten des Anwenders oder der Anwendung. React Native übernimmt die Prinzipien von ReactJS, sodass jemand der mit ReactJS vertraut ist ohne Probleme auf React Native umsteigen könnte. Die gesamte Anwendung, zusammen mit den UI-Komponenten wird mit JavaScript oder optional TypeScript, zusammen mit der eigenen Templating Syntax JSX geschrieben. React Native stellt in erster Linie abstrakte UI-Komponenten bereit, welche anschließend auf das jeweilige Gegenstück der Zielplattform abgebildet werden. Somit kann eine Button-Komponente, wie jede andere React-Komponente, direkt in dem JavaScript Code importiert und verwendet werden. Diese hat auf der entsprechenden Plattform ein natives Gegenstück, welches gerendert wird[12][27][36].

React Native selbst besteht aus nativen Komponenten für die jeweilige Plattform, einer JavaScript Laufzeitumgebung, und der React Native Bridge[37].

- Die nativen Komponenten sind in der Sprache der jeweiligen Plattform geschrieben. Diese stellen die Funktionen und UI-Komponenten für den in JavaScript geschriebenen Anwendungscode bereit.
- Die JavaScript Laufzeitumgebung führt die eigentliche Anwendungslogik auf dem Endgerät aus. Bei iOS Geräten nutzt React Native die JavaScriptCore Laufzeitumgebung, welche vom Betriebssystem bereitgestellt wird. Diese wird ebenfalls von Safari verwendet und ist somit bereits auf dem Gerät vorhanden. Bei der Android Zielplattform hingegen, wird JavaScriptCore zusammen mit der Applikation ausgeliefert, da diese nicht vom Betriebssystem bereitgestellt werden kann. Die führt zu einer erhöhten Anwendungsgröße von React Native Anwendungen unter Android.
- Um die Kommunikation zwischen dem von Anwendungsentwickler geschriebenen JavaScript Code und den nativen Komponenten zu ermöglichen, wird die sogenannte React Native Bridge verwendet. Diese reagiert auf Eingaben des Benutzers, welche in dem nativen Teil der Applikation registriert werden, übersetzt diese in deren JavaScript äquivalent, damit der Anwendungscode diese verarbeiten und darauf reagieren kann. Diese Reaktion wird nun wiederum von einer JavaScript Anweisung in das native äquivalent abgebildet und dem User angezeigt.

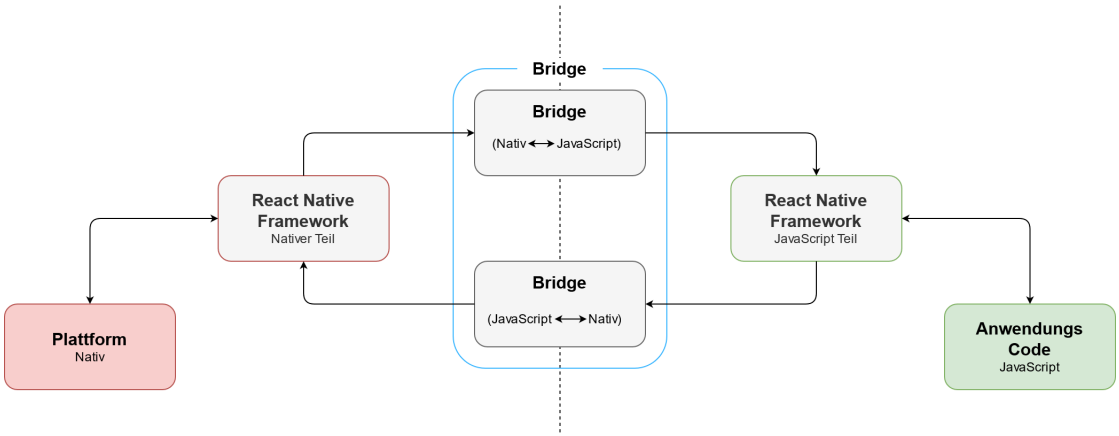

Abbildung 12: Übersicht der Kommunikation zwischen dem Nativen- und JavaScript-Teil des React Frameworks (Anlehnung an [36])

Da die gesamte Kommunikation des nativen und des JavaScript Teils über die React Native Bridge laufen, stellt diese einen Engpass dar, welcher zu Performanceeinbußen führt. Vor allem bei Animationen, welche z.B. von der Position des Fingers des Users abhängig sind, muss bis zu 60-mal pro Sekunde über die Bridge kommuniziert werden.
„ Here lies one of the main keys to understanding React Native performance. Each realm by itself is blazingly fast. The performance bottleneck often occurs when we move from one realm to the other. In order to architect performant React Native apps, we must keep passes over the bridge to a minimum. “ [38]
Um dieses Problem zu beheben, arbeitet das React Native Team derzeit an einer grundlegend neuen Architektur. Diese trägt den Projektnamen Fabric[39].

4.4 Das Cross-Kompilierende Framework Flutter

Flutter ist ein von Google entwickeltes Open Source cross-plattform SDK, um nativ kompilierte Anwendungen für Web, Desktop und Mobilgeräte ausgehend von einer einzigen Codebasis zu entwickeln. Es wurde 2018 offiziell veröffentlicht und benutzt die ebenfalls von Google entwickelte Programmiersprache Dart[40].

Dart ist eine ECMA-Standardisierte Vielzweck-Programmiersprache und wurde ursprünglich als alternative zu JavaScript entwickelt[41]. Dart ist optional typisiert, somit kennt Dart zwei Laufzeit-Modi: Produktion und Checked. Im Produktionsmodus wählt der Compiler selbstständig einen Typen und ignoriert Typisierungsanweisungen sowie Typfehler, um eine möglichst hohe Effizienz zu gewährleisten. Im Checked-Modus werden Typen strikt beachtet und bei Fehlern Exceptions geworfen. Die dafür notwendige Codeanalyse macht diesen Modus jedoch langsamer. Dart kann je nach Verwendungszweck auf verschiedene Arten kompiliert werden. Dazu gehören für die native Entwicklung JIT, Ahead of Time (AOT) sowie der dart2js und dartdevc Compiler, welcher aus Dart Code JavaScript für das Web erzeugt[42].

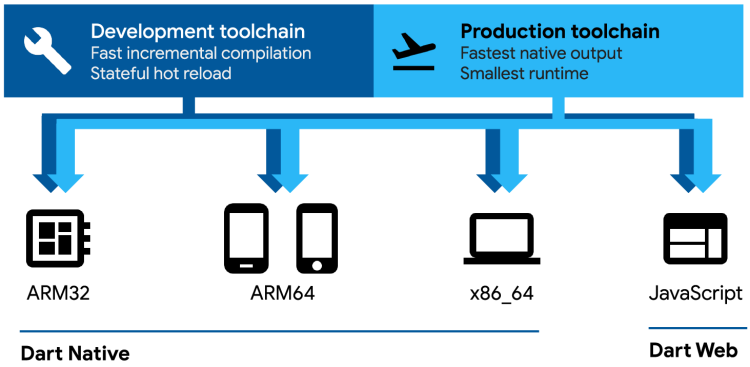

Abbildung 13: Übersicht der Zielplattformen der Programmiersprache Dart [42]

Diese Flexibilität macht Dart einzigartig, in der Hinsicht, dass es die einzige breit verwendete Programmiersprache ist, welche diese Möglichkeiten bietet. Bei der Entwicklung wird meist der JIT Ansatz zusammen mit einer Dart Virtual Machine (VM) verwendet, da damit der Entwicklungsfluss gefördert wird, da Anpassungen im Quellcode direkt beobachtet werden können. Zum Ausliefern der Anwendung wird diese schließlich AOT kompiliert. Der Vorteil von AOT kompiliertem Code ist, dass dadurch die Startzeit der Anwendung stark verringert werden kann. Anwendungen, welche den Code zuerst kompilieren müssen, haben aufgrund der notwendigen Codeanalyse und der Kompilation selbst eine längere Startzeit. Es wurde nachgewiesen, dass bei längeren Verzögerungen, User von der Benutzung einer App oder Website absehen, falls die Ladezeit drei Sekunden übersteigt[43]. Ein weiterer Vorteil des AOT Ansatzes ist, dass damit dem Problem der „JavaScript Bridge“ entgegengewirkt

wird. Dieses entsteht durch die Notwendigkeit der Kommunikation zwischen einer dynamischen und somit JIT kompilierten Sprache wie JavaScript sowie dem nativen Code der Plattform.

Dieses Problem wird durch die AOT Kompilation von Dart zwar nicht komplett beseitigt, da dieser ebenfalls über ein Interface mit dem nativen Code kommunizieren muss, jedoch muss kein kompletter Kontextwechsel vollzogen werden. Da Flutter die UI-Komponenten selbst mithilfe der mitgelieferten Skia Engine rendert ist die Notwendigkeit häufiger Kommunikation ebenfalls geringer und dadurch ressourcenschonender. Die Tatsache, dass die UI-Komponenten (bei Flutter Widgets genannt) von der Applikation und nicht von der Plattform gerendert werden birgt einige Vorteile. Einerseits sind die UI-Komponenten erweiterbar und anpassbar, andererseits entfällt die Notwendigkeit eines virtuell DOMs. Dieser ist bei Frameworks wie React die Grundlage, um die interne Repräsentation des Inhaltes darzustellen und wird bei Veränderungen mit dem tatsächlichen DOM verglichen, sodass Unterschiede gebündelt angepasst werden können. Da Flutter die UI-Komponenten selber rendert, entfällt die Notwendigkeit und der damit verbundene Aufwand, da der virtuelle DOM gleichzeitig den reellen DOM darstellt. Andererseits steigt durch die Tatsache, dass Flutter die UI-Komponenten und den passenden Renderer bereitstellt die Kompatibilität der verschiedenen Betriebssystemversionen. Die Notwendigkeit auf all diesen Versionen testen zu müssen entfällt, da sich die Flutter-Entwickler darum kümmern. Der Dart Code einer Flutter Anwendung kommuniziert mittels der sogenannten Platform Channels mit dem nativen API der Plattform. Dabei werden keine Abbildungen von Dart Code zu nativen API-Aufrufen verwendet, sondern bidirektionale Nachrichten, welche asynchron verarbeitet werden und mithilfe von Callbacks dem Aufrufer ihr Ergebnis mitteilen. Dadurch entfällt der Engpass der zentralen Bridge.

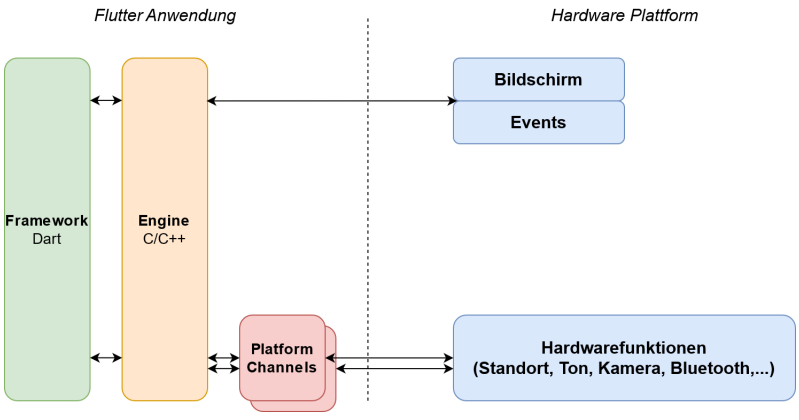

Abbildung 14: Vereinfachtes Schema einer Flutter Anwendung

Im Allgemeinen ähnelt das Flutter Framework einer Game Engine. Da Flutter Anwendungen aus dem Framework selbst (Dart), einer performanten Rendering Engine (C/C++) und einem plattformspezifischen Teil bestehen, welcher den Einstiegspunkt der Anwendung bildet und sich um die Kommunikation mit dem Betriebssystem kümmert.

5 Fazit

Die Entwicklung einer Anwendung mithilfe eines cross-plattform Frameworks verringert den Entwicklungsaufwand und die dafür notwendige Zeit. Dies führt zu schnelleren Ergebnissen, da die Anwendung nur auf einer Codebasis aufbaut. Jedoch existiert einer Vielzahl an verschiedenen Entwicklungsansätzen, welche alle bestimmte Vor- und Nachteile bieten. Dazu gehören (Progressive) Web-Apps, hybride Apps, Apps mit eigener Laufzeitumgebung, cross-kompilierte sowie generierte Apps.

Allgemein lässt sich feststellen, dass native Apps die meisten Features bieten, dies jedoch auf Kosten der Notwendigkeit zwei komplett eigenständige Anwendungen entwickeln zu müssen. Auf der anderen Seite des Spektrums befinden sich Web-Applikationen, welche einfacher zu entwickeln und pflegen sind, meist jedoch nicht die benötigen Features bereitstellen. Diese Lücke versuchen PWAs zu füllen, indem diese einige native Funktionalitäten bereitstellen und somit die größten Nachteile der Web-Applikationen aufheben. Hybride App Frameworks versuchen die Vorteile der Entwicklung mit Web-Technologie sowie den Zugang zu Hardware-Features für die Erstellung von Apps, welche über den App-Store verteilt werden können, zugänglich zu machen. Diese haben jedoch den Nachteil, dass damit der native „Look and Feel“ einer nativen App nicht trivial repliziert werden kann. Auf einer eigenen Laufzeitumgebung basierende Apps versuchen diesen Nachteil zu umgehen, in dem native UI-Komponenten verwendet werden, welche mit JavaScript definiert werden können. Einen neuen Weg versucht das Flutter Framwork zu gehen. Dieses cross-kompilierende Framework verwaltet die Darstellung und Rendering der UI-Komponenten selbst und versucht somit die Performance-Probleme der Framworks mit eigenständiger Laufzeitumgebung zu beheben. Der aktuelle Trend spricht derzeit für diesen Game-Engine-ähnlichen Ansatz, jedoch bleibt abzuwarten, ob sich dieser Trend fortsetzt, da die Entwicklung sich noch am Anfang befindet.

Welches Framework sollte denn nun für das nächste Projekt verwendet werden? Diese Frage ist stark einzelfallabhängig und es gibt, wie so häufig, keine eindeutige Antwort. Jedoch möchte ich trotzdem einige grundsätzliche Fragen formulieren, welche bei der Evaluation solcher Frameworks beantwortet werden können, um die Auswahl zu vereinfachen. Zunächst wäre es sinnvoll folgende Fragen bezüglich der Anwendung und der Zielgruppe zu beantworten.

- Welches Problem wird mit der Anwendung für wen gelöst?
- Welche Technik ist die Zielgruppe vertraut?
- Mit welchen Entwicklungsansätzen, Sprachen und Frameworks sind die Entwickler vertraut?
- Wie soll die Anwendung verteilt werden? (App-/Playstore oder Browser)
- Wie performant soll die Anwendung sein?
- Wie viele und welche plattformspezifische Features sollen verwendet werden?
- Ist eine Internetverbindung am Einsatzort verfügbar?
- Müssen spezifische Zertifizierungen wie z.B. im Gesundheits- und Finanzsektor erfüllt werden?

Erst nachdem die vorangestellten Fragen beantwortet wurden, sollten die verschiedenen Vor- und Nachteile der verschiedenen Entwicklungsansätzen gegeneinander aufgewogen und verglichen werden.

Aufgrund der in der Arbeit genannten Punkte ist es für ein Team, welches keine ausgeprägten Kenntnisse mit Web-Technologien besitzt, meistens besser Flutter zu verwenden. Falls im Entwicklerteam bereits Web-Kenntnisse mit React vorhanden sind, ist React Native eine sinnvolle Wahl. Ansonsten bietet Ionic eine flexible Alternative, da dort verschiedene Front-End-Frameworks verwendet werden können.

Literaturverzeichnis

[1] S. O'Dea. *Number of smartphones sold to end users worldwide from 2007 to 2021*. 11. Juni 2021. URL: https://www.statista.com/statistics/263437/global-smartphone-sales-to-end-users-since-2007/ (besucht am 31. 03. 2021).

[2] Inc. Gartner. *Gartner Says Global Smartphone Sales Fell Slightly in the Fourth Quarter of 2019*. 3. März 2020. URL: https://www.gartner.com/en/newsroom/press-releases/2020-03-03-gartner-says-global-smartphone-sales-fell-slightly-in (besucht am 31. 03. 2021).

[3] Ash Turner. *how many smartphones are in the world?* 2021. URL: https://www.bankmycell.com/blog/how-many-phones-are-in-the-world (besucht am 03. 07. 2021).

[4] Statcounter. *Mobile Operating System Market Share Worldwide*. 2021. URL: https://gs.statcounter.com/os-market-share/mobile/worldwide (besucht am 19. 06. 2021).

[5] Statcounter. *Mobile & Tablet Android Version Market Share Worldwide*. 2021. URL: https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide (besucht am 29. 06. 2021).

[6] Statcounter. *Mobile & Tablet iOS Version Market Share Worldwide*. 2021. URL: https://gs.statcounter.com/os-version-market-share/ios/mobile-tablet/worldwide (besucht am 29. 06. 2021).

[7] Ray Rischpater. *Providing operating system compatibility on a large scale*. 2021. URL: https://medium.com/flutter/providing-operating-system-compatibility-on-a-large-scale-374cc2fb0dad (besucht am 03. 07. 2021).

[8] National Institute of Standards und Technology. *Security and Privacy Controls for Information Systems and Organizations*. 12. Okt. 2020, S. 435. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r5.pdf (besucht am 29. 06. 2021).

[9] Henning Heitkötter u. a. *Business Apps: Grundlagen und Status quo*. Jan. 2012. URL: https://www.uni-muenster.de/imperia/md/content/angewandteinformatik/aktivitaeten/publikationen/business-apps.pdf (besucht am 29. 06. 2021).

[10] Martin Fowler. *InversionOfControl*. 26. Juni 2005. URL: https://martinfowler.com/bliki/InversionOfControl.html (besucht am 29. 06. 2021).

[11] mostlyjason. *Opinionated or Not: Choosing the Right Framework for the Job*. 28. Okt. 2019. URL: https://hackernoon.com/opinionated-or-not-choosing-the-right-framework-for-the-job-6x1u2ga0 (besucht am 03. 07. 2021).

[12] Christoph Rieger und Tim A. Majchrzak. „Towards the definitive evaluation framework for cross-platform app development approaches“. In: *Journal of Systems and Software* 153 (2019), S. 175–199. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2019.04.001. URL: https://www.sciencedirect.com/science/article/pii/S0164121219300743.

[13] Google LLC. *What is Andriod*. 2021. URL: https://www.android.com/what-is-android/ (besucht am 29. 06. 2021).

[14] Opensignal Limited. *Android Fragmentation 2015*. 2015. URL: https://www.opensignal.com/sites/opensignal-com/files/data/reports/global/data-2015-08/2015_08_fragmentation_report.pdf (besucht am 29. 06. 2021).

[15] Tim A. Majchrzak, Jan Ernsting und Herbert Kuchen. „Achieving Business Practicability of Model-Driven Cross-Platform Apps“. In: *Open Journal of Information Systems (OJIS)* 2 (Jan. 2015), S. 3–14.

[16] Spyros Xanthopoulos und Stelios Xinogalos. „A Comparative Analysis of Cross-Platform Development Approaches for Mobile Applications“. In: *Proceedings of the 6th Balkan Conference in Informatics*. BCI '13. Thessaloniki, Greece: Association for Computing Machinery, 2013, S. 213–220. ISBN: 9781450318518. DOI: 10.1145/2490257.2490292. URL: https://doi.org/10.1145/2490257.2490292.

[17] Apple Inc. *Apple Developer Program*. 2021. URL: https://developer.apple.com/de/support/compare-memberships/ (besucht am 29. 06. 2021).

[18] Apple Inc. *Submit your iOS and iPadOS apps to the App Store*. 2021. URL: https://developer.apple.com/ios/submit/ (besucht am 29. 06. 2021).

[19] Wafaa S. El-Kassas u. a. „Taxonomy of Cross-Platform Mobile Applications Development Approaches“. In: *Ain Shams Engineering Journal* 8.2 (2017), S. 163–190. ISSN: 2090-4479. DOI: https://doi.org/10.1016/j.asej.2015.08.004. URL: https://www.sciencedirect.com/science/article/pii/S2090447915001276.

[20] Mozilla Community. *Introduction to progressive web apps*. 2. Juni 2021. URL: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Introduction (besucht am 30. 06. 2021).

[21] M. Firtman. *High Performance Mobile Web: Best Practices for Optimizing Mobile Web Apps*. O'Reilly Media, 2016. ISBN: 9781491913666. URL: https://books.google.de/books?id=FOYPDQAAQBAJ.

[22] Pete LePage Sam Richard. *What makes a good Progressive Web App?* 6. Jan. 2020. URL: https://web.dev/pwa-checklist/ (besucht am 30. 06. 2021).

[23] Andreas Biørn-Hansen, Tim A. Majchrzak und Tor-Morten Grønli. „Progressive Web Apps: The Possible Web-native Unifier for Mobile Development“. In: Jan. 2017, S. 344–351. DOI: 10.5220/0006353703440351.

[24] Oluwatofunmi Adetunji, Chigozirim Ajaegbu und Otuneme Nzechukwu. „Dawning of Progressive Web Applications (PWA): Edging Out the Pitfalls of Traditional Mobile Development“. In: *American Scientific Research Journal for Engineering, Technology, and Sciences* 68 (Mai 2020), S. 85–99.

[25] IQUII. *Progressive Web App (PWA): what they are, pros and cons and the main examples on the market*. 2021. URL: https://medium.com/iquii/progressive-web-app-pwa-what-they-are-pros-and-cons-and-the-main-examples-on-the-market-318f4538c670 (besucht am 04. 03. 2019).

[26] Johannes Feiner (Hsg) und Petra Kletzenbauer (Hsg). *Internet-Technologien und -AnwendungenFH JOANNEUM*. 1. März 2015. URL: https://cdn.fh-joanneum.at/media/2016/02/KMUgoesmobile-Sammelband-1.pdf (besucht am 30. 06. 2021).

[27] Andreas Biørn-Hansen u. a. „An empirical investigation of performance overhead in cross-platform mobile development frameworks“. In: *Empirical Software Engineering* (Juli 2020). DOI: 10.1007/s10664-020-09827-6.

[28] inVerita. *Flutter vs Native vs React-Native: Examining performance*. 10. März 2020. URL: https://medium.com/swlh/flutter-vs-native-vs-react-native-examining-performance-31338f081980 (besucht am 29. 06. 2021).

[29] Eric Umuhoza und Marco Brambilla. „Model Driven Development Approaches for Mobile Applications: A Survey“. In: Aug. 2016. ISBN: 978-3-319-44214-3. DOI: 10.1007/978-3-319-44215-0_8.

[30] Wikipedia Community. *Ionic (mobile app framework)*. 23. Mai 2021. URL: https://en.wikipedia.org/wiki/Ionic_(mobile_app_framework) (besucht am 01. 07. 2021).

[31] Ionic Community. *Ionic Framework*. 2021. URL: https://ionicframework.com/docs (besucht am 01. 07. 2021).

[32] Pankaj Kumar Choudhary. *Introduction To Ionic*. 10. Juni 2020. URL: https://www.c-sharpcorner.com/article/introduction-to-ionic/ (besucht am 01. 07. 2021).

[33] The Apache Software Foundation. *Overview*. 2015. URL: https://cordova.apache.org/docs/en/10.x/guide/overview/ (besucht am 01. 07. 2021).

[34] Arkivar. *Is there a graphic diagram available that shows how Ionic 2 and all related technologies connect?* 17. Aug. 2018. URL: https://forum.ionicframework.com/t/is-there-a-graphic-diagram-available-that-shows-how-ionic-2-and-all-related-technologies-connect/88375/20 (besucht am 01. 07. 2021).

[35] AltexSoft LLC. *Xamarin vs React Native vs Ionic vs NativeScript: Cross-platform Mobile Frameworks Comparison*. 4. Okt. 2021. URL: https://www.altexsoft.com/blog/engineering/xamarin-vs-react-native-vs-ionic-vs-nativescript-cross-platform-mobile-frameworks-comparison/ (besucht am 01. 07. 2021).

[36] Bartosz Skuza. *What is React Native and When to Use it? Introduction for App Owners*. 29. Okt. 2020. URL: https://www.thedroidsonroids.com/blog/what-is-react-native-introduction (besucht am 01. 07. 2021).

[37] Atul R Rahul Gaba. *React Made Native Easy*. 15. Sep. 2020. URL: https://github.com/react-made-native-easy/book (besucht am 01. 07. 2021).

[38] Tal Kol. *Performance Limitations of React Native and How to Overcome Them*. 19. Juni 2016. URL: https://talkol.medium.com/performance-limitations-of-react-native-and-how-to-overcome-them-947630d7f440 (besucht am 01. 07. 2021).

[39] Parashuram N. *React Native's New Architecture*. 28. Okt. 2018. URL: https://www.youtube.com/watch?v=UcqRXTriUVI (besucht am 01. 07. 2021).

[40] Google LLC. *Flutter*. 2021. URL: https://flutter.dev/ (besucht am 01. 07. 2021).

[41] Frederic Lardinois. *Google's Dart Programming Language Is Coming To The Server*. 29. Juni 2014. URL: https://techcrunch.com/2014/06/29/googles-dart-programming-language-is-coming-to-the-server/ (besucht am 01. 07. 2021).

[42] Google LLC. *Dart overview*. 2021. URL: https://dart.dev/overview (besucht am 01. 07. 2021).

[43] Tammy Everts. *Mobile Load Time and User Abandonment*. 9. Okt. 2016. URL: https://developer.akamai.com/blog/2016/09/14/mobile-load-time-user-abandonment (besucht am 01. 07. 2021).