Institut für Hochfrequenztechnik
RWTH AACHEN UNIVERSITY

Seminararbeit

Evaluation von Cross-Platform-App-Frameworks für ein Amateurfunk-Paging-Netzwerk

von Valentin Geyer
Matr.-Nr. 3273710

Betreut durch:
Prof. Dr. rer. nat. Philipp Rohde
Florian Reher, M.Sc.

Seminararbeit
Aachen, den
15. Dezember 2022

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die Seminararbeit mit dem Thema

Evaluation von Cross-Platform-App-Frameworks für ein Amateurfunk-Paging-Netzwerk

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war. Ich verpflichte mich, ein Exemplar der Seminararbeit fünf Jahre aufzubewahren und auf Verlangen dem Prüfungsamt des Fachbereiches Medizintechnik und Technomathematik auszuhändigen.

Name: Valentin Geyer

Aachen, den 15. Dezember 2022

Unterschrift des Studenten

Kurzfassung

Durch die Heterogenität der Betriebssysteme mobiler Endgeräte ist die Entwicklung von Apps aufwendig, weil betriebssystemabhängig verschiedene Technologien benutzt werden. Da diese Technologien nicht miteinander kompatibel sind, wird für jedes Betriebssystem - in diesem Kontext **Plattform** genannt - eine eigene, voneinander unabhängig App entwickelt. Kosten für Entwicklung und Wartung steigen jedoch, wenn nicht auf geteilten Quellcode zurückgegriffen werden kann.

Ein **Cross-Platform-App-Framework** erlaubt das Erstellen einer App für mehrere Plattformen aus einer gemeinsamen Codebase heraus. Vor dem Hintergrund einer App zum Nutzen und Verwalten von DapNET, einem Pager-Funknetzwerk des Amateurfunkes, sollen verschiedene Cross-Platform-App-Frameworks für Android und iOS verglichen werden.

Diese Arbeit beschreibt die Funktionsweise der Frameworks in der Theorie. Es werden Kriterien zur Bewertung aus der Fachliteratur abgeleitet, kontextualisiert und gewichtet. Mithilfe von Ausschlusskriterien wird eine Vorauswahl auf die vier Frameworks *Flutter*, *React Native*, *Ionic Framework* und *Xamarin* getroffen. Diese werden anhand von Bewertungskriterien evaluiert und einer Nutzwertanalyse unterzogen.

Im Ergebnis ist *React Native* das Framework mit der größten Eignung für den spezifischen Anwendungsfall, eng gefolgt von *Flutter*. *Ionic Framework* und *Xamarin* sind weniger geeignet.

Inhaltsverzeichnis

1 Motivation

Durch die stetig wachsende Anzahl an mobilen Endgeräten in Deutschland wächst auch der Markt für Software, genannt **Apps**, die auf ihnen ausgeführt wird. Dabei ist die Gruppe der mobilen Endgeräte heterogen, umfasst also eine Vielzahl an Hardwarekonfigurationen und Betriebssystemen, die sich durch Anpassungen des Herstellers oder unterschiedlichen Versionen untereinander weiter unterscheiden können. Während früher eine Vielzahl verschiedener Betriebssysteme den Markt unter sich aufteilten, dominieren heute das quelloffene und von Google entwickelte Android und Apple's iOS, jeweils eingebunden in ihren eigenen Ökosystemen. [VuM21; Sta22f; Sta22b; Sca+19]

Trotzdem erschwert diese Heterogenität weiterhin die Appentwicklung, denn Apps sind stark an das Betriebssystem, für welches sie geschrieben sind, gebunden. Das **Software Development Kit**, kurz **SDK**, welches zum Entwickeln, Bauen, Veröffentlichen, Emulieren und Debuggen einer App benötigt wird, ist im Allgemeinen betriebssystemspezifisch. So findet die typische Entwicklung von Apps für verschiedene Betriebssysteme, in diesem Kontext **Plattformen** genannt, in unterschiedlichen Programmiersprachen, SDKs und Entwicklungsumgebungen (eng. **Integrated Development Environment**, kurz **IDE**) statt, fordert also unterschiedliche Fähigkeiten und Erfahrungen von den Entwicklern.
Meistens werden Apps deshalb für alle großen Plattformen separat entwickelt, um insgesamt einen Großteil der Geräte und somit der Nutzer erreichen zu können. Das erhöht die Kosten, denn diese separaten Apps können häufig untereinander nicht auf geteilten Code zurückgreifen und benötigen spezialisierte Entwickler. [Grø+14; PSC12]

Ein **Cross-Platform-App-Framework** hilft, diesen Aufwand zu reduzieren. Ein Framework stellt eine **Abstraktionsschicht** bereit, die zwischen der Plattform und der entwickelten App vermittelt. So kann die App aus einer Codebase für alle Plattformen entsteht. Das spart Entwicklungszeit, Kosten und verringert den langfristigen Wartungsaufwand. [PSC12; BH+20; Naw+21]
Trotzdem werden oft native Apps geschrieben, da Cross-Platform-Apps allgemein mit schlechter Performance und Nutzererfahrung assoziiert werden. [BHGG18]

Basierend auf dem IP-vernetzten HAMNET haben sich Funkamateure ein Funkrufnetzwerk, das DapNET, geschaffen, welches Nachrichten von Nutzern annimmt und an einzelne Beteiligte oder auch Nutzergruppen aussendet. Der Empfang erfolgt mittels kleiner tragbarer Geräte mit einzeiligem alphanumerischem Display, besser bekannt als Pager oder Skyper. Neben individuellen Mitteilungen können auch Sammelmeldungen zu Unwetterlagen, zivilen Katastrophenfällen und Vergleichbares an die Pager abgesetzt werden. Das DapNET darf nicht kommerziell genutzt werden. [RA; Dapa]
Zur Eingabe von Nachrichten, dem Administrieren und Konfigurieren der Pager (Empfangsgerät), Funkrufsender (Sendestation) und Überwachung der Infrastruktur wird, im

Rahmen einer Reimplementierung von DapNET, eine App benötigt, deren Implementierungsgrundlage in der vorliegenden Arbeit untersucht wird.

In dieser Arbeit werden verschiedene Cross-Platform-App-Frameworks theoretisch beschrieben. Danach werden die Vergleichskriterien aus der Fachliteratur abgeleitet, kontextualisiert und gewichtet. Anschließend wird mit einigen dieser Kriterien eine Vorauswahl auf vier Frameworks getroffen. Diese wird abschließend im Rahmen einer Nutzwertanalyse evaluiert.

2 Funktionsweise von Cross-Platform-App-Frameworks

Die Funktionsweise eines Cross-Platform-App-Frameworks wird grundsätzlich in zwei Ansätze geteilt: Den **interpreterbasierten Ansatz** und den **generatorbasierten Ansatz**.

Bei einem interpreterbasierten Framework wird der Programmcode der App innerhalb einer Laufzeitumgebung ausgeführt. Zugriff auf Funktionen der Plattform oder Hardware werden meist über Plugins realisiert, die eine „Brücke“ zwischen einer nativen Implementierung dieses Zugriffes und der Laufzeitumgebung bauen.
Die Laufzeitumgebung kann von der App selbst mitgeliefert werden oder von der Plattform bereitgestellt werden. Die Laufzeitumgebung zusammen mit den Plugins und Brücken bildet die Abstraktionsschicht. [BH+20; Naw+21]

Ein Beispiel einer von der Plattform bereitgestellten Laufzeitumgebung ist der **WebView**, eine Komponente zum Darstellen von Webinhalten innerhalb einer nativen App.
Der aus dem interpreterbasierten Ansatz abgeleitet **hybride Ansatz** nutzt ausschließlich einen WebView zum Darstellen und Ausführen der App. Die App wird vollständig mittels Webtechnologien, also `HTML`, `JavaScript` und `CSS`, implementiert. Der WebView wird in eine native App, genannt „Wrapper“, eingebettet. Diese kann dann in einem Appstore veröffentlicht, installiert und ohne Internetverbindung ausgeführt werden. Außerdem können durch den „Wrapper“ Funktionen der Plattform und des Geräts angesprochen werden, die aus einem Webkontext unerreichbar wären. [BH+20; Naw+21]

Webseiten und **Progressive Web Apps**, kurz **PWA**, zählen ebenfalls zu dem interpreterbasierten Ansatz, nutzen aber den Webbrowser der Plattform zum Darstellen und Ausführen der App. Innerhalb des Webbrowser ist der Zugriff auf die Plattform und das Gerät eingeschränkt.
Eine PWA ist eine normale Webseite, die zusätzlich ein **Appmanifest** und einen **Service Worker** bereitstellt, wodurch sie, analog zu einer hybriden App, installiert werden können. Dies speichert alle statischen Dateien lokal auf dem Gerät und erstellt einen Eintrag in der App-Liste. Bei erneutem Öffnen der Applikation kann diese sofort geladen werden, da die nötigen Daten aus dem Speicher geladen werden. Der Service Worker ist für diesen

Prozess zentral: Als Netzwerkproxy der App kann er nach Bedarf Daten zwischenspeichern und ausliefern. Das ermöglicht das Nutzen der App auch ohne Internetverbindung. Da der Service Worker in einem eigenen Thread ausgeführt wird, kann er im Hintergrund arbeiten und Push-Benachrichtigungen senden, auch wenn die App selbst geschlossen ist. [BHMG17; MBHG18]

Der generatorbasierte Ansatz konvertiert das Projekt in das native Format der Plattform, ohne das zur Laufzeit „übersetzt“ werden muss. Daraus leiten sich die folgenden zwei Ansätze ab:
Der **compilierbasierte Ansatz** nutzt einen Compiler, der den Programmcode der App in plattformspezifischen Maschinencode konvertiert. Dieser wird dann analog zu dem hybriden Ansatz in einen Wrapper eingebettet.
Der **modellorientierte Ansatz** generiert aus einem abstrakten Modell der App plattformnativen Quellcode. Dieser kann mit dem SDK der Plattform in eine native Applikation überführt werden. [BH+20; Naw+21]

Diese Einordnung, siehe Abb. 2.1, ist aufgrund der Komplexität des Themas nicht exakt. So gibt es Frameworks, die die Grenzen verschiedener Ansätze überschneiden. Ein Beispiel hierfür ist *Flutter*, ein compilerbasiertes Framework, welches im Debug-Modus eine Laufzeitumgebung benutzt.
Dennoch erlaubt diese Einordnung eine nützliche Unterscheidung verschiedener Vorgehensweisen.

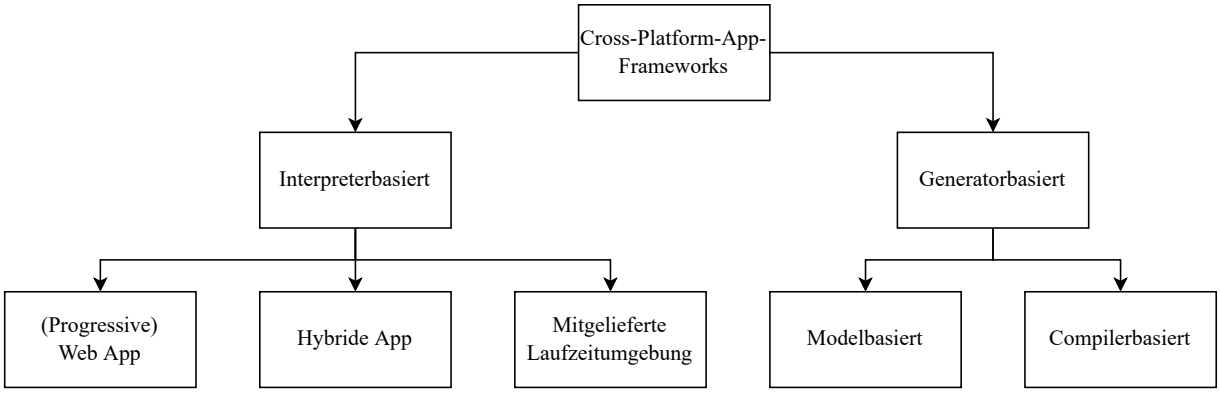

Abb. 2.1: Einordnung von Cross-Platform-App-Frameworks

3 Kriterien

Eine 2019 von C. Rieger und T. A. Majchrzak veröffentlichte Arbeit [RM19] bietet einen umfassenden Kriterienkatalog zur Evaluation von Cross-Platform-App-Frameworks an, der viele andere bis 2019 veröffentlichte Arbeiten zu dieser Problemstellung aufgreift. Dabei ist das Bewertungsschema auf „appfähige“ Geräte im Allgemeinen, also auch Smart TVs,

Smart Watches oder IoT (Internet of Things), ausgelegt. Daraus resultiert ein Kriterienkatalog, der 33 Kriterien umfasst. Nicht alle davon sind im Kontext der DapNET-Applikation relevant, so werden Monetarisierungsmöglichkeiten wie In-App-Käufe nicht benötigt und entsprechend nicht betrachtet. Andere Kriterien werden möglichst auf den Anwendungsfall fokussiert zusammengefasst.
Die hieraus abgeleiteten Kriterien werden in Ausschlusskriterien und Bewertungskriterien unterteilt. Ausschlusskriterien dienen dem Selektieren einer Vorauswahl, werden binär gewertet und nicht gewichtet, da sie nicht direkt in die Evaluation eingehen. Bewertungskriterien werden in der Nutzwertanalyse in Kapitel 5 genutzt, um die Evaluation durchzuführen.

Bei einer Nutzwertanalyse werden alle gegebenen Optionen anhand gewichteter Kriterien numerisch bewertet. Dabei sollte die Bewertung der Kriterien objektiv stattfinden. Die Gewichtung eines Kriteriums spiegelt seine Relevanz vor dem Hintergrund des Projektes wider und kann entsprechend subjektiv sein. Jede Option wird auf alle Kriterien hin untersucht, bepunktet und diese rohe Punktzahl mit dem Gewicht des Kriteriums multipliziert. Die Gesamtwertung einer Option ist die Summe aller gewichteten Kriterien. [FfE20]
So können komplexen Entscheidungsproblemen auf transparente und erweiterbare Art und Weise gelöst werden, denn der Prozess ist auch für Außenstehende einfach nachzuvollziehen und leicht um neue Kriterien oder Optionen erweiterbar.
Aus diesen Gründen wird auch in der unseren Kriterien zugrundeliegenden Arbeit von C. Rieger und T. A. Majchrzak eine Nutzerwertanalyse verwendet, um Webseiten, PWAs, *Adobe PhoneGap*, *React Native* und native Apps im Kontext verschiedener Mobilgerätetypen zu bewerten. [RM19]

Analog zu dieser Arbeit wird die Summe aller Gewichte auf 100 beschränkt. Die Punkteskala für die Kriterien wird auf Ganzzahlen von `0` (nicht erfüllt) bis `10` (voll erfüllt) angesetzt, um zusammengelegte Kriterien zu kompensieren.
Dabei bekommt jedes Kriterium eine eindeutige Nummer, mit der es später referenziert wird. Außerdem wird angegeben, aus welchen Kriterien der Primärliteratur [RM19] dieses abgeleitet wurde.
Im Weiteren werden die Ausschlusskriterien und die Bewertungskriterien gelistet und anschließend gewichtet.

3.1 Ausschlusskriterien

AK1: Lizenzierung

siehe Kriterium I1 in [RM19]

Die Lizenzierung eines Frameworks bestimmt, unter welchen Bedingungen und für welche Anwendungszwecke damit erzeugte Applikationen genutzt und vertrieben werden dürfen. Da der Amateurfunk ausschließlich durch seine Mitglieder finanziert wird, sind die Kosten des Frameworks ein wichtiger Aspekt. Es sollte kostenfrei nutzbar sein. Außerdem gilt im Rahmen des Amateurfunkgedankens der offenen Kommunikation und Völkerverständigung [Ama], dass ein quelloffenes und frei nutzbares Framework zu benutzen ist.

AK2: Zielplattformen

siehe Kriterium I2 in [RM19]

Das Framework muss mindestens Android und iOS als Zielplattformen unterstützen.

AK3: Mehrsprachigkeit

siehe Kriterium I6 in [RM19]

Da Apps nicht geografisch an die unmittelbare Region geknüpft sind, sondern potenziell global nutzbar sind ist es sinnvoll, Applikationen mehrsprachig zu designen [RM19]. Im Rahmen des Amateurfunkes, der sich der Völkerverständigung verpflichtet [Ama], gilt dies umso mehr. Entsprechend ist die Unterstützung des Frameworks für sprachabhängige Formatierung und Übersetzung der Inhalte erforderlich.

AK4: Zukunftssicherheit

siehe Kriterien I7, D9 in [RM19]

Die Festlegung auf ein Framework ist ein Risiko, da der Wechsel auf ein anderes Framework häufig sehr aufwendig und somit zeit- und kostenintensiv ist. Entsprechend ist die Wahl eines Frameworks, welches möglichst lange gewartet wird, anzustreben. Gute Indizien für solch eine Langlebigkeit sind eine große Nutzerbasis, ein aktives Entwicklerteam, welches regelmäßig Updates veröffentlicht, und Sponsoren, die das Framework und seine Entwickler unterstützen. Sind diese Indizien erfüllt, so sollte selbst im Falle der Aufgabe des Frameworks immer noch eine große Community existieren, die entweder bei der Migration auf ein anderes Framework hilft oder ein Nachfolgeprojekt anstößt.
Das Framework muss alle diese Indizien erfüllen.

3.2 Bewertungskriterien

K1: Entwicklerwerkzeuge

siehe Kriterum D1 in [RM19]

Ein guter Werkzeugsatz kann das Erstellen und Warten von Apps massiv beschleunigen. Dazu tragen besonders eine integrierte Dokumentation des Frameworks, Autovervollständigung, Fehlererkennung durch statische Codeanalyse und die Möglichkeit, mittels eines Emulators die App in einer kontrollierten Umgebung direkt auf dem Entwicklungscomputer auszuführen, bei.
Ein Debugger ermöglicht es, die App im laufenden Zustand zu untersuchen, um Fehler zu diagnostizieren.
Mit einem Profiler wird die Performance der App evaluiert. Die daraus erhaltenen Daten sind essenziell, um die App zu optimieren.

Die Fähigkeit, Änderungen im Code direkt in die laufende App zu laden, genannt „Hot-Reloading“, sorgt für zeitnahes Feedback und erlaubt so schnelle Iteration.

K2: Entwicklungsplattformen

siehe Kriterium I3 in [RM19]

Die Entwicklungswerkzeuge sollten auf allen gängigen Betriebssystemen, also Windows, Linux und MacOS [Sta22c] vollständig unterstützt werden.

K3: Einarbeitungszeit

siehe Kriterien D2, D12 in [RM19]

Das Framework sollte für einen möglichst große Gruppe an Entwicklern zugänglich und leicht zu Erlernen sein.
Die Programmiersprachen, mit denen die App entwickelt wird, sollten auch außerhalb des Frameworks möglichst verbreitet sein.
Außerdem sind gute Dokumentation und Tutorials für das Frameworks erforderlich. Sie sollten klar strukturiert, aktuell und vollständig sein.

K4: Veröffentlichung

siehe Kriterien I4, D7 in [RM19]

Um die Applikation dem Nutzer zugänglich zu machen, muss diese veröffentlicht werden. Für die Mehrheit der Nutzer geschieht dies über einen Appstore, welcher es erlaubt, eine große Menge von Apps zu durchsuchen, zu installieren und aktuell zu halten. [JB13]
Das Veröffentlichen für einen Appstore bringt aber neue Schwierigkeiten mit sich, da die Prozesse dahinter aufwendig sind. Das Cross-Platform-App-Framework sollte möglichst tief in diese Prozesse integriert sein.
PWA können eine Alternative zur Veröffentlichung der App über einen Appstore sein. So können manche Frameworks wie etwa *Ionic Framework* eine App ohne große Änderungen als PWA bereitstellen [Comh]. Obwohl unter anderem der Zugriff auf die Hardware eingeschränkt ist, gibt es gute Gründe dafür, eine App ebenfalls als Webseite/PWA zu veröffentlichen. Zum einen lässt sich eine Webseite direkt, ohne Installation, ausprobieren. Zum anderen sind PWA bedeutend kleiner als normale Apps [MBHG18] und müssen nicht über einen Appstore veröffentlicht werden. Im Kontext des Amateurfunks, der als nicht-kommerzielle Entität selbstfinanziert ist, gilt es Kosten zu vermeiden. Das Veröffentlichen in Google's Play Store kostet einmalig $25 Dollar. [Goo]
Apple's Appstore setzt für das Veröffentlichen von Software ein „Entwicklerkonto“ voraus, welches $100 USD *pro Jahr* kostet. [App]
Entsprechend kostet die aktuelle DapNET-iOS-App $2.79 Dollar [Dapb], da diese Kosten an die Nutzer weitergegeben werden müssen.
Deshalb wäre die Möglichkeit, die App auch als Webseite/PWA bereitzustellen, wünschenswert.

Durch externe, cloudbasierte Dienstleister für *Continuous Integration/Continuous Delivery*, kurz *CI/CD*, lässt sich der Aufwand des Bauens und Veröffentlichens weiter reduzieren. Vor dem Hintergrund des nicht-kommerziellen Amateurfunks ist hier ein kostenloser Dienstleister vorzuziehen.

K5: Testbarkeit

siehe Kriterium D6 in [RM19]

Das Framework sollte verschiedene Arten von Tests unterstützen, um Fehler früh zu erkennen und mit *CI/CD*-Dienstleistern integrieren.
Ein *Unit Test* überprüft einen kleinen, isolierten Teil der App, meist eine Funktion, auf Korrektheit. Diese sind einfach zu Schreiben, Durchzuführen und zu Warten.
Ein Komponententest wird bei komponentenbasierten Frameworks angewandt, um einzelne Bausteine der App zu testen.
Ein Integrationstest prüft, ob verschiedene Teil der Applikation richtig zusammenarbeiten.
Ein *End-to-End Test* überprüft die Applikation als Ganzes auf Korrektheit. Diese sind aufwendig in der Erstellung, Durchführung und Wartung. [Pit; Devp]
Das Framework sollte alle diese Tests unterstützen.

K6: Plattformzugriff

siehe Kriterien A1, A2, A3, A6, U4 in [RM19]

Mobile Geräte haben eine Vielzahl an Sensoren, um Daten über die Umgebung und den Nutzer zu sammeln. Beispiele sind etwa Beschleunigungssensor, Kamera, Mikrofon und der Standort. Über das Betriebssystem ist unter anderem Zugriff auf Kontakte, Batteriezustand, Netzwerkzustand, den lokalen Speicher, Benachrichtigungen, Hintergrunddienste oder andere verbunden Geräte (z. B. eine Smartwatch) möglich. Da diese Funktionen nur über plattformnative Schnittstellen bedienbar sind, müssen Cross-Platform-App-Frameworks eine Abstraktionsschicht errichten. Der Umfang dieser wird in diesem Kriterium bewertet.

K7: Erweiterbarkeit

siehe Kriterien D3, D10, D11 in [RM19]

Im Idealfall lässt sich die gesamte Logik und Oberfläche der App durch das Cross-Platform-Framework aus einer gemeinsamen Codebasis heraus implementiert. Falls Drittparteiensoftware inkludiert werden müssen, es keine plattformübergreifende Lösung gibt oder die Erstellung einer solchen unwirtschaftlich ist, sollte man auf native Implementierungen zurückzufallen können.

K8: Aussehen und Verhalten

siehe Kriterien U1, D5 in [RM19]

Die Cross-Platform-App sollte sich aus Sicht des Benutzers von einer nativen Applikation weder im Aussehen noch Verhalten unterscheiden. Der einfachste Weg, dies zu erreichen, ist das Nutzen von nativen Nutzeroberflächenelementen (eng. *native UI elements*).

K9: Performance

siehe Kriterium U2 in [RM19]

Die App sollte nicht durch lange Ladezeiten, hohen Speicherverbrauch, hohe Prozessorauslastung, lange Wartezeiten oder große Applikationsgröße auffallen. Grundsätzlich sollte die Bedienerfahrung analog zu der einer nativen Applikation sein.

3.3 Gewichtung

Der Fokus der App liegt auf Wartbarkeit und Aussehen. Hardwarezugriff ist keine Priorität. Zur Herleitung der Gewichte wird die *Direct Ranking*-Methode benutzt. Bei dieser werden die Gewichte direkt zugeordnet, siehe Abb. 3.1.

Kriterium	Gewicht	Kriterium	Gewicht
K1: Entwicklerwerkzeuge	15	*K6: Plattformzugriff*	06
K2: Entwicklungsplattformen	05	*K7: Erweiterbarkeit*	06
K3: Einarbeitungszeit	20	*K8: Aussehen und Verhalten*	20
K4: Veröffentlichung	08	*K9: Performance*	10
K5: Testbarkeit	10		
Summe: 100			

Abb. 3.1: Gewichtung der Kriterien

4 Vorauswahl der Frameworks

Aufgrund der Vielzahl an Frameworks, die dazu geeignet sind, plattformübergreifend Apps zu entwickeln, ist eine Vorauswahl nötig. Die in Kapitel 3.1 herausgearbeiteten Ausschlusskriterien *AK1: Lizenzierung*, *AK2: Zielplattformen*, *AK3: Mehrsprachigkeit* und *AK4: Zukunftssicherheit* dienen dieser als Grundlage.
Zuerst wird über *AK4: Zukunftssicherheit* eine Eingrenzung auf die Frameworks mit den meisten aktiven Entwicklern durchgeführt. Die dadurch erhaltene Vorauswahl wird

anschließend auf aktive Weiterentwicklung *(AK4)*, Sponsoren *(AK4)*, freie Lizenzierung *(AK1)*, Kosten *(AK1)*, Internationalisierung *(AK3)* und Plattformunterstützung für Android und iOS *(AK2)* untersucht.

Nach Umfragen von JetBrains [Jet21] und Stack Overflow [Ove22] sind die unter Entwicklern meistgenutzten Cross-Platform-App-Frameworks *Flutter*, *React Native*, *Ionic Framework*, *Apache Cordova*, *Unity*, *Xamarin* und *NativeScript*.
Im Weiteren werden diese Frameworks erklärt und auf die Vorauswahlkriterien hin untersucht.

4.1 Flutter

Flutter ist ein kostenfreies, quelloffenes, von Google entwickeltes Framework. Die erste stabile Version wurde 2018 veröffentlicht und seitdem regelmäßig aktualisiert. Es ist *frei lizenziert*, hat eine eingebaute i18n-Übersetzung [Devn] und unterstützt neben Android und iOS auch Desktop und Web. [Devj]
Flutter benutzt keine vom Betriebssystem bereitgestellten, nativen UI-Komponenten, sondern stellt mittels der *Skia Graphics Engine* eigene Komponenten, genannt *Widgets*, dar. Durch Verschachtelung von *Widgets* können komplexe Benutzeroberflächen geschaffen werden. Diese Komponenten können beliebig angepasst werden und es existieren Vorlagen für Nutzeroberflächen, die nativen Android und iOS-Oberflächen stark gleichen. Außerdem benutzt Flutter einen Compiler, um den in *Dart* geschriebenen Quellcode, die Komponenten und Flutter selbst in plattformnativen Maschinencode zu übersetzen. [Devh]
Architektonisch besteht Flutter aus drei aufeinander aufbauenden Teilen (siehe Abb. 4.1): Der *platform-specific Embedder*, der *Engine* und dem nutzerzugewanden *Framework*. Der *Embedder* stellt die direkte Verbindung zu der Plattform dar. Er verpackt Flutter in eine native App, welche die *Engine* startet. Die *Engine* ist in `C++` geschrieben und das zentrale Element von Flutter. Es stellt die gesamte Funktionalität Flutter's bereit, darunter Rendering mit Skia, Layout, Input-Output, Systemeventverarbeitung und die Dart-Laufzeitumgebung. Diese kümmert sich um *Garbage-Collection*, Multithreading und mehr. Im Debug-Modus ermöglicht sie durch *Just-in-Time-Compilation* das direkte Anwenden von Codeänderungen in der laufenden Applikation, genannt „Hot-Reloading“. Im Release-Modus wird der Programmcode in nativen Maschinencode übersetzt. *Flutter* zählt somit zu den compilerbasierten Frameworks.

Das *Framework* ist in Dart geschrieben und stellt unter anderem die *Widgets* und Animationen bereit.
Dabei ist jeder dieser drei Komponenten in Schichten aufgeteilt, die untereinander isoliert sind, also nicht auf Implementationsdetails untereinander zurückgreifen, und jederzeit ersetzbar sind, siehe Abb. 4.1. [Ego; Devi]

4.2 React Native

React Native wurde 2015 von Facebook veröffentlicht [Occ]. Seitdem wird es in vielen kom-

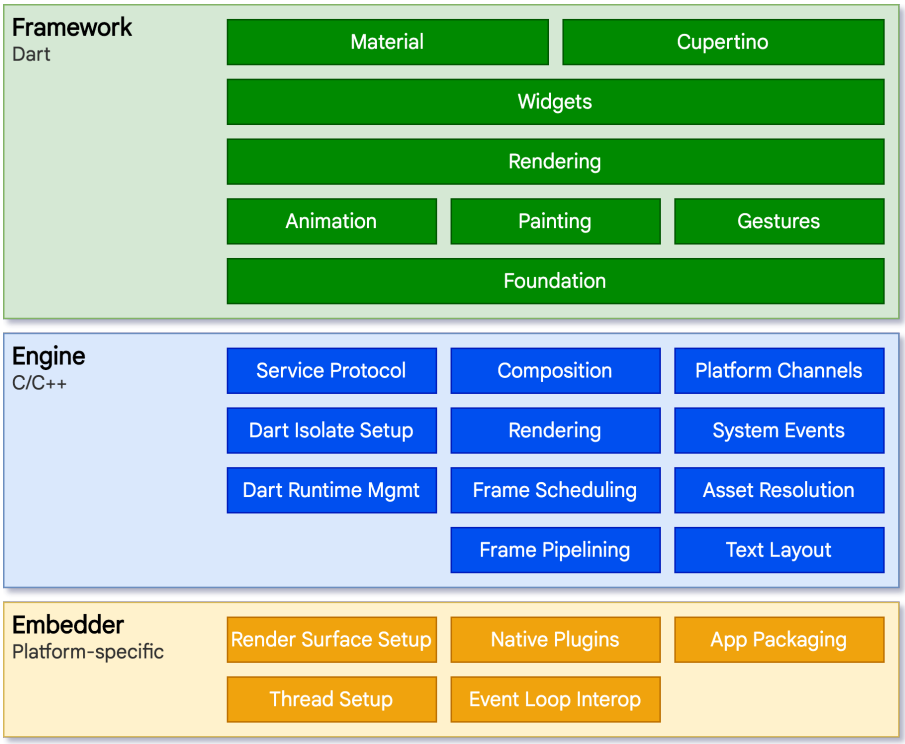

Abb. 4.1: Architektur von Flutter, siehe [Devi], *lizenziert unter CC BY 4.0*

merziellen Projekten genutzt [RNCn]. Es hat sich eine aktive Community auch außerhalb von Facebook gebildet. So hat das GitHub-Repository über 100.000 Sterne und Millionen an Nutzern [RNCk]. Das Framework wird regelmäßig aktualisiert und ist unter der freien MIT-Lizenz lizenziert [RNCk]. Unterstützung für Internationalisierung wird mittels Community-Plugins bereitgestellt. [Reaa]

React Native benutzt das JavaScript-Framework *React.js* und somit primär `JavaScript`, `HTML` und `CSS` zum Erstellen von Apps. Der Quellcode der App wird durch *Metro* analysiert und transformiert, um innerhalb einer modifizierten `JS`-Laufzeitumgebung namens *Hermes* ausgeführt zu werden. Die Nutzeroberfläche wird mit nativen Elementen dargestellt.

React Native wird im Allgemeinen zu den *interpreterbasierten Frameworks* gezählt, obwohl es durch seine Verwendung von Webtechnologie und der Fähigkeit, mit *Hermes* den Programmcode in Bytecode zu compilieren, auch den *hybriden* oder den *compilierbasierten* Ansatz schneidet. [RNCb]

Expo ist ein privates Unternehmen, das Werkzeuge zur Apperstellung mit React Native anbietet. Mit der *Expo Go* App lässt sich die entwickelte App schnell auf physischen Geräten testen, indem der Quellcode in der von der *Expo Go* App bereitgestellten *React Native* Umgebung ausgeführt wird. So kann ohne das Einrichten eines nativen SDK an der App gearbeitet werden.
Expo Go wird auf der offiziellen Webseite empfohlen. [RNCl; Expb]

4.3 Ionic Framework, Apache Cordova und Capacitor

Ionic Framework, welches 2013 vorgestellt wurde [Lyna] und 2015 die erste stabile Version veröffentlichte [Spe], arbeitet ebenfalls mit `JavaScript`, `HTML` und `CSS`. Anders als *React Native* erlaubt dieses neben *React.JS* auch das Arbeiten mit den JavaScript-Frameworks *Angular* und *Vue*. Als *hybrides* Framework benutzt es keine nativen UI-Elemente, sondern rendert die gesamte App in einer WebView-Komponente. Der Zugriff auf Hard- und Software der Plattform geschieht durch *Capacitor* oder *Apache Cordova*, welche über JavaScript angesteuert werden. *Ionic Framework* ist quelloffen, kostenfrei nutzbar, wird regelmäßig aktualisiert [Iond], unterstützt Internationalisierung [Comd] und wird von dem namensgebenden Unternehmen *Ionic* finanziert.

Apache Cordova ist als quelloffene Version aus dem 2009 bekanntgegebenen und 2020 eingestellten [Bui; Liea] Framework *Adobe PhoneGap* entstanden. Es ist eine Plattformabstraktion, die es innerhalb eines WebView ausgeführten Programmcode ermöglicht, über sogenannte Plugins mit nativem Quellcode der Plattform zu kommunizieren. Diese Plugins können per NPM, dem *Node Package Manager*, in das Projekt integriert werden. [Fou]

Die 2019 entwickelte Plattformabstraktionsschicht *Capacitor* dient für *Ionic Framework* als Nachfolger von *Apache Cordova* [Lync]. *Capacitor* ist dabei rückwärtskompatibel und die Migration ist einfach durchzuführen [Lynb]. Es ist ebenso eine webviewbasierte Plattformabstraktion, die mit Plugins nativen Code ansteuern kann. Capacitor kann unabhängig von *Ionic Framework* genutzt werden. Capacitor unterstützt das Veröffentlichen der App als PWA. [Coma]

Im Allgemeinen wird neuen Entwicklerteams empfohlen, *Capacitor* anstelle von *Apache Cordova* zu nutzen [Lieb; MRT]. Die Entwicklerumfrage von Stack Overflow [Ove22], die „State of JavaScript 2021"-Umfrage [al.21] und aktuelle *Downloadzahlen* beider Frameworks sprechen ebenfalls für eine stetig wachsende Dominanz von Capacitor gegenüber Cordova. Außerdem bekommt Capacitor häufiger Updates als Cordova. Stand 29.11.2022 liegt Cordovas *letztes Update* fast ein Jahr zurück, Capacitors nur *13 Tage*.

Deshalb findet *Apache Cordova* im Weiteren keine Berücksichtigung, da *Ionic Framework* den hybriden Entwicklungsansatz zukunftssicherer abbildet. *Ionic Framework* wird entsprechend nur mit *Capacitor* als Plattformabstraktion evaluiert.

4.4 Unity

Unity ist ein kostenpflichtiges Framework zur plattformübergreifenden Videospielentwicklung mit proprietärem Quellcode. Da dies den Vorauswahlkriterien nicht genügt, wird es in dieser Arbeit nicht weiter betrachtet, obwohl es theoretisch zur Entwicklung von Apps fähig ist. [Sin]

4.5 Xamarin

Xamarin ist eines der ältesten aktiv gepflegten Cross-Platform-Frameworks. Es wurde 2011 gegründet und 2016 von Microsoft gekauft. Seitdem ist es kostenlos, MIT-lizenziert und der Quellcode ist öffentlich verfügbar. Neben Android und iOS werden auch Windows und Mac als Zielplattform unterstützt. [Naw+21; Coms; Mic]
Xamarin benutzt die `.NET`-Laufzeitumgebung, um `C#`-Quellcode auf allen unterstützten Plattformen auszuführen. Dabei besteht das Framework aus 4 Teilen: *Xamarin.Android*, *Xamarin.iOS*, *Xamarin.Essentials* und *Xamarin.Forms*.
Xamarin.Android und *Xamarin.iOS* ermöglichen vollständigen Zugriff auf die Android- und respektive iOS-Plattform. Das Schreiben von Benutzeroberflächen ist durch diese Schnittstelle nur plattformspezifisch möglich. [Coms]

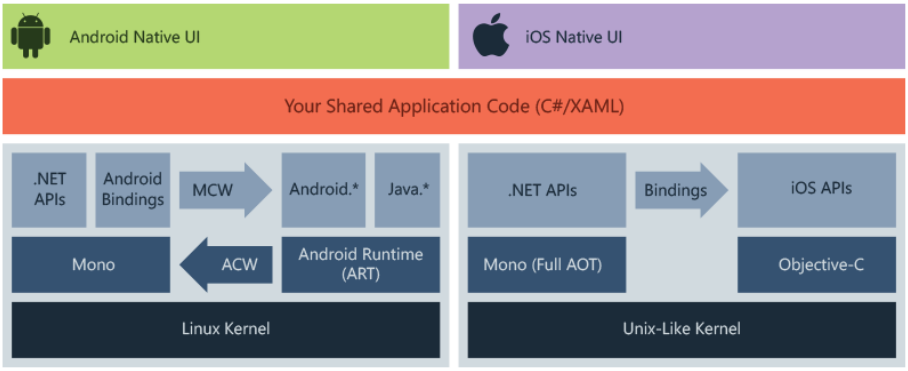

Abb. 4.2: Architektur von Xamarin.Android und Xamarin.iOS, siehe [Coms], *lizenziert unter CC BY 4.0*

Xamarin.Essentials stellt eine einheitliche Schnittstelle für Gerätefunktionalität zur Verfügung, so etwa für Dateizugriff und Geräteinformation. [Coms]
Xamarin.Forms stellt eine einheitliche XAML-basierte Syntax zur Verfügung, um plattformübergreifende Benutzeroberflächen zu schreiben. Diese können mit Oberflächen aus *Xamarin.Android* und *Xamarin.iOS* kombiniert werden. [Coms; Comw; Dav]

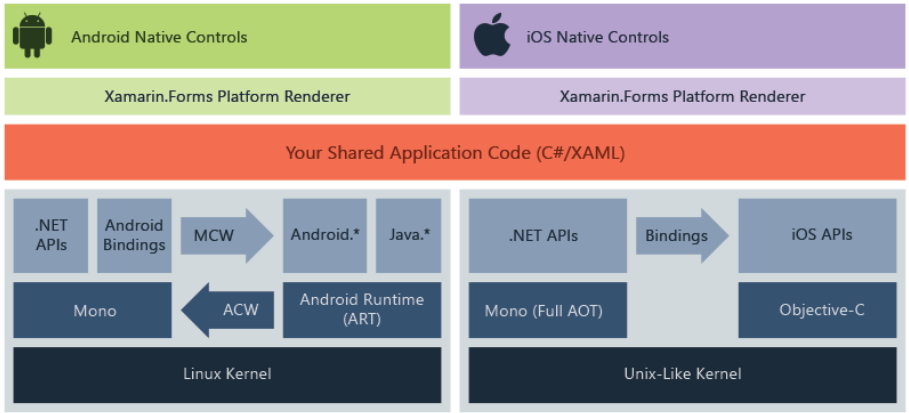

Abb. 4.3: Architektur von Xamarin.Forms, siehe [Comt], *lizenziert unter CC BY 4.0*

4.6 NativeScript

NativeScript basiert wie *React Native* auf einer `JavaScript`-Laufzeitumgebung, die native Nutzeroberflächen erzeugen und plattformspezifische API's ohne eingebetteten WebView benutzen kann. Es wurde 2014 veröffentlicht [Sto] und seitdem kontinuierlich weiterentwickelt [Nat].
NativeScript erlaubt die Nutzung vieler `JavaScript`-Frameworks wie *Vue* und *Angular*, was einem größeren Teil der JavaScript-Gemeinschaft einen einfachen Einstieg ermöglicht. Trotzdem wird React Native, welches nur das JS-Framework *React.js* unterstützt, bedeutend öfter genutzt. So wird das GitHub-Repository von React Native im Vergleich zu dem von NativeScript mit mehr Sternen (100k vs. 20k) bewertet. Außerdem hat es signifikant mehr Code-Kontributoren (2.4k vs. 0.2k) und abhängige externe Repositories (1.1M vs. 5.2k) [RNCk; Nat].
Die *Downloadzahlen* der NPM-Pakete beider Projekte und die zwei neusten Entwicklerumfragen [Ove22; Jet21] sprechen ebenfalls für eine starke Dominanz von React Native gegenüber NativeScript. Da die beiden Frameworks sehr ähnliche Charakteristiken haben, React Native aber bedeutend populärer ist, wird NativeScript im Rahmen dieser Arbeit nicht evaluiert. [Asi]

4.7 Übersicht

Damit sind *Flutter*, *React Native*, *Ionic Framework*, und *Xamarin* die im Weiteren zu evaluierenden Frameworks, siehe Abb. 4.4.

	Flutter	***React Native***	***Ionic***	***Xamarin***
Lizenz *(AK1)*	*BSD 3-Clause „New" or „Revised" License*	*MIT License*	*MIT License*	*MIT License*
unterstützte Plattformen *(AK2)*	Android, iOS, Linux, macOS, Web, Windows	Android, iOS, Web	Android, iOS, Linux, macOS, Web, Windows	Android, iOS, Linux, macOS, Windows
Internationalisierung *(AK3)*	integriert [Devn]	extern [Reaa]	extern [Comd]	integriert [Comx]
Verbreitung *(AK4)*	häufig	häufig	öfters	gelegentlich
Letztes Update, Stand 12.12.22 *(AK4)*	23.11.22 [Devj]	15.11.22 [RNCk]	7.12.22 [Iond]	*Android: 9.11.22 iOS: 8.11.22 Forms: 5.12.22*
Sponsor *(AK4)*	Google [Devh]	*Meta*	Ionic	Microsoft

Abb. 4.4: Übersicht der zu evaluierenden Cross-Platform-App-Frameworks

5 Evaluation

Im Weiteren werden die Frameworks bezüglich der in Kapitel 3.2 beschriebenen Bewertungskriterien evaluiert.

K1: Entwicklerwerkzeuge

Der Vergleich von Entwicklerwerkzeugen hat eine große subjektive Komponente, da sich Vorerfahrung oder mangelnde Erfahrung stark auf die Nützlichkeit dieser auswirkt. [Naw+21]
Deshalb wird versucht, durch das Bewerten des Funktionsumfanges des Frameworks bezüglich des Debuggers, des Profilers und der Hot-Reload-Funktion eine objektive Bewertung zu erreichen. Hier wird eine einheitliche, vom Framework bereitgestellt Lösung bevorzugt.
Ergänzend wird eine zusammengefasste Bewertung der Literatur in die Wertung einbezogen.

Flutter hat sehr gute Debug- und Profilingwerkzeuge. So kann mit *DevTools* der Speicher, die UI, der Programmfluss und mehr visualisiert und analysiert werden. Es ist außerdem möglich, einen Debugger direkt an den Quellcode anzuschließen. Mit *Flutter inspector* lässt sich die Benutzeroberfläche in ihre einzelnen Widgets zerlegen und diese detailliert betrachten. Nativer Code lässt sich mit dem im Plattform-SDK enthaltenen OEM-Debugger analysieren. [Devg; Devq]
Wird die App im Debug-Modus betrieben, so wird diese nicht in Maschinencode compiliert, sondern von einem Interpreter direkt ausgeführt. Dadurch können Änderungen am Code direkt in der laufenden App angewendet werden. [Devh; Devm]

React Native hat ebenfalls gute Werkzeuge zum Debuggen und Profilen. In der App selbst lassen sich über ein „Debugging-Menü“ zahlreiche Werkzeuge auf dem Gerät selbst und von verbundenen Computern aus ausführen. Zum Profilen der Nutzeroberfläche muss aber auf den Systemprofiler zurückgegriffen werden, da diese nativ ist. [RNCh; RNCc]
Mit „Fast-Refresh“ lassen sich Änderungen im Projekt schnell anwenden. Werden fehlerhafte Änderungen auf diese Weise verarbeitet, so stürzt die App nicht ab, sondern stoppt die Komponente und zeigt den aufgetretenen Fehler an. In den meisten Fällen bleibt der Zustand der Applikation erhalten. [RNCd]

Ionic Framework stellt selbst keine Debugging- und Profilingwerkzeuge zur Verfügung. Stattdessen werden Werkzeuge für Webanwendungen genutzt, die sich mit dem WebView in der App verbinden. Zum Debuggen und Profilen von nativen Plugins müssen die Systemwerkzeuge verwendet werden. [Comc; Gri]
Live-Reload wird unterstützt. [Comf]

Xamarin benutzt den in Visual Studio integrierten Debugger. [Comj; Comk]
Zum Profilen müssen die nativen Profiler der Plattform oder der kostenpflichtige *Xamarin Profiler* genutzt werden. [Comm; Comn; Comu]

Xamarin.Forms unterstüzt das Neuladen von XAML-Nutzeroberflächendefinitionen. Native, mit Xamarin.iOS oder Xamarin.Android geschriebene Oberflächen und `C#`-Programmcode verfügen nicht über eine Hot-Reload-Funktion.

In der Literatur wird die Entwicklungserfahrung von *React Native*, *Ionic Framework* und *Xamarin* unter der von *Flutter* eingeordnet. Die Entwicklungserfahrung von *Flutter* wird besser als von nativen Applikationen beschrieben. *React Native* hat am häufigsten zu Problemen beim Entwickeln geführt. Xamarins fehlende Hot-Reload-Funktion führt zu einer langen „Feedback Loop“. [MG17; Naw+21; Lym]
Die Stack Overflow Entwicklerumfrage aus dem Jahr 2022 sieht Flutter mit knapp 70 Prozent Zufriedenheit vor React Native (∼56%), Ionic (∼43%) und Xamarin (∼39%). [Ove22]

		Flutter		***React Native***		***Ionic***		***Xamarin***	
Debugger	2	Ja	2	Ja	2	Nur JS	1	Ja	2
Hot-Reloading	3	Ja	3	Ja	3	Ja	3	Nur UI	1
Profiling	2	Ja	2	Nur JS	1	Nur JS	1	Nativ	1
Literatur	3	sehr gut	3	mäßig	2	mäßig	2	schlecht	1
Summe			**10**		**8**		**7**		**5**

Abb. 5.1: Punktevergabe für *K1: Entwicklerwerkzeuge*

K2: Entwicklungsplattformen

Das Benutzen einer gut integrierten Entwicklungsumgebung erhöht die Produktivität, wenn der Entwickler mit ihr vertraut ist. Microsofts *Visual Studio Code* (kurz VSCode), *Visual Studio* und JetBrains *IntelliJ IDEA* sind die drei am meisten benutzten IDEs. [Ove22]
Dabei sind, aufgrund der Verbreitung dieser IDEs, auch viele communitygebaute Integrationen in Form von Plugins verfügbar. In diesem Kriterium werden nur die offiziell empfohlenen IDEs betrachtet. Bewertet wird die Unterstützung der IDEs für die Desktop-Betriebssysteme Windows, Linux und macOS, gewichtet nach deren Popularität. [Sta22a]

Flutter bietet Integrationen des eigenen *Command Line Interface*, kurz *CLI*, für Android Studio, VSCode und IntelliJ an [Devh]. *React Native* empfiehlt keine IDEs. Es existieren aber Plugins, so etwa ein von Microsoft veröffentlichtes für VSCode mit mehr als 3 Millionen Installationen [Reac]. *Ionic Framework* empfiehlt VSCode [Iona]. *Xamarin* benutzt ausschließlich Visual Studio, welches im Gegensatz zu den anderen IDEs nicht für Linux-basierte Betriebssysteme verfügbar ist [Coml].

		Flutter		***React Native***		***Ionic***		***Xamarin***	
präferierte IDE		VSCode, IntelliJ		beliebig		VSCode		Visual Studio	
Windows	5	Ja	5	Ja	5	Ja	5	Ja	5
Linux	2	Ja	2	Ja	2	Ja	2	Nein	0
macOS	3	Ja	3	Ja	3	Ja	3	Ja	3
Summe			**10**		**10**		**10**		**8**

Abb. 5.2: Punktevergabe für *K2: Entwicklungsplattformen*

K3: Einarbeitungszeit

Das Bewerten der Qualität von Dokumentation oder der Schwierigkeit des Erlernens einer Programmiersprache ist nur begrenzt objektiv möglich. Entsprechend ist das Bewerten der Einarbeitungszeit in ein Framework schwer.
Deshalb wird die Verbreitung der zum Erstellen einer App benötigten Technologien betrachtet. Bei einer hohen Verbreitung der Technologie existiert eine große Gemeinschaft an Entwicklern, die Hilfestellung geben kann oder bereits eingearbeitet ist. Diese Metrik dient zusammen mit der Verbreitung des Frameworks aus *AK4: Zukunftssicherheit* als Bewertung dieses Kriteriums.

Mit *Flutter* entwickelte Apps werden mit *Dart* entwickelt, einer modernen Programmiersprache für Client-Applikationen.
React Native benutzt das JavaScript-Framework *React.js*, also die Webtechnologien `JavaScript`, `CSS` und `HTML`, zum Entwickeln einer App.
Apps, die mit *Ionic Framework* entwickelt werden, benutzen ebenfalls `JavaScript`, `CSS` und `HTML`. Es werden die JavaScript-Frameworks *React.js*, *Vue.js* und *Angular* unterstützt.
Xamarin-Apps werden mit `C#` und `XAML` entwickelt.

In der Stack Overflow Entwicklerumfrage aus dem Jahr 2022 zur Nutzung von Programmiersprachen liegen `JS`, `CSS` und `HTML` weit vor `C#` und `Dart`.
React.js zählt zu den verbreitetsten JavaScript-Frameworks, gefolgt von *Vue.js* und *Angular*. [Ove22]
Flutter und *React Native* sind die unter Entwicklern populärsten Frameworks, gefolgt von *Ionic* und *Xamarin*. [Ove22; Jet21, *(AK4)*]

		Flutter		***React Native***		***Ionic***		***Xamarin***	
Technologien	4	Dart	1	Webtech.	3	Webtech.	4	C#	2
Verbreitung	6	Hoch	6	Hoch	6	Mittel	4	Mittel	3
Summe			**7**		**9**		**8**		**5**

Abb. 5.3: Punktevergabe für *K3: Einarbeitungszeit*

K4: Veröffentlichung

Durch eine gute Integration des Veröffentlichungsprozesses seitens des Frameworks kann der manuelle Aufwand der Veröffentlichung für den Entwickler gesenkt werden. Dieser kann durch *CI/CD* nahezu eliminiert werden.
Mittels Live-Updating kann eine App direkt ohne erneuten Veröffentlichungsprozess aktualisiert werden. Dies ist nützlich, um Fehler direkt beheben zu können, ohne das der Benutzer manuell ein Update aus dem Appstore herunterladen muss.
Trotzdem ist es wichtig, die Applikation auch lokal Bauen und Veröffentlichen zu können.
Deshalb wird im Folgenden bewertet, wie tief das Framework in den Veröffentlichungsprozess integriert ist, ob PWA und Live-Updating unterstützt werden und ob es geeignete externe Dienstleister für *CI/CD* gibt.

Flutter benutzt das eigene CLI zum Compilieren der App. Dazu ist eine plattformnative Umgebung, die vom Entwickler selber bereitgestellt und eingerichtet werden muss, nötig. Das Veröffentlichen muss manuell durchgeführt werden. [Devd; Deve]
PWA-Unterstützung ist in Arbeit. [Devc]
Live-Reloading wird nicht unterstützt.

React Native geht analog zu Flutter bezüglich des Veröffentlichungsprozesses vor. [RNCi; RNCj]
PWA werden nicht unterstützt.
Expo bietet eine vollwertige *CI/CD*-Lösung für React Native an. Diese umfasst in der kostenlosen Stufe [Expe] Bauen und Veröffentlichen der App in der Cloud. Mit Expo *EAS Update* lassen sich die webbasierten Bestandteile der App ohne eine Neuveröffentlichung im Appstore aktualisieren. [Expd]

Ionic Framework geht analog zu Flutter bezüglich des Veröffentlichungsprozesses vor. [Comb; Come]
PWA werden unterstützt. [Comh]
Live-Updates werden durch *Applow*, Ionics kostenpflichtigen *CI/CD*-Service angeboten. [Ionb; Ionc]

Xamarin hat den Veröffentlichunsgprozess vollständig in Visual Studio integriert. [Como; Comp]
PWA werden nicht unterstützt, da das Web als Plattform an sich nicht unterstützt wird [Coms]
Live-Updates werden nicht unterstützt.

Es gibt viele *CI/CD*-Dienste, die alle hier besprochenen Frameworks unterstützen, darunter *CodeMagic* und *Bitrise*. Außerdem können mittels GitHub Actions ebenfalls Apps getestet und gebaut werden. Alle drei Dienste sind, in kleinem Rahmen, kostenlos nutzbar. [San; Cod; Bit; Git]

		Flutter		***React Native***		***Ionic***		***Xamarin***	
Integration Android	3	nur Bauen	2	nur Bauen	2	nur Bauen	2	Integriert in VS	3
Integration iOS	3	nur Bauen	2	nur Bauen	2	nur Bauen	2	Integriert in VS	3
PWA	2	In Arbeit	1	Nein	0	Ja	2	Nein	0
externe *CI/CD*	1	Ja	1	Ja	1	Ja	1	Ja	1
Live-Updating	1	Nein	0	Ja	1	Bezahlt	0	Nein	0
Summe			**6**		**6**		**7**		**7**

Abb. 5.4: Punktevergabe für *K4: Veröffentlichung*

K5: Testbarkeit

Alle betrachteten Frameworks unterstützen Unit-Tests, Komponententests, Integrationstests und End-to-End-Tests und sind gut in *CI/CD* integriert, siehe *K4: Veröffentlichung*. [Devp; RNCm; Ionf; Mer; Xam]
Deshalb werden alle Frameworks mit `10` Punkten gewertet.

K6: Plattformzugriff

Eine gute Anbindung der Hard- und Software der Plattform an das Framework ermöglicht das Entwickeln komplexer und funktionsreicher Apps.

Flutter, *React Native*, und *Ionic Framework* bieten selbst keine vollständige Schnittstelle zum Zugriff auf das Plattform-SDK an. Diese lassen sich aber durch eine große Menge an Plugins nachrüsten. Während Ionic und Flutter einen Teil dieser Plugins selber warten, überlässt React Native dies ganz der Community. Das sorgt für teils schlecht dokumentierte oder dysfunktionale Plugins. [Naw+21; Reab; Dar; Comg; Devh]

Xamarin bietet eine vollständige Abstraktion über das iOS- und Android-SDK an. [Comt; Comv]

Grundsätzlich können, im Gegensatz zu Aussagen älterer Arbeiten, alle modernen Cross-Platform-App-Frameworks einen vollständigen Zugriff auf die darunterliegende Plattform realisieren. [BHG18; BHGG18]
Entsprechend erhält *Xamarin* `10` Punkte und *React Native* `8` Punkte. *Ionic Framework* und *Flutter* erhalten `9` Punkte.

K7: Erweiterbarkeit

Dieses Kriterium bewertet, ob nativer Code, native Bibliotheken und native Benutzeroberflächen in das Framework inkludiert werden können.

Flutter kann UI-Elemente der Plattform benutzen und ansprechen. [Devk; Devl]
Nativer Code, mitsamt Bibliotheken in C oder C++, kann eingebunden werden. [Devr; Devb; Deva]

React Native hat ebenfalls volle Unterstützung für native UI-Elemente, nativen Code oder Bibliotheken. [RNCg; RNCf; RNCa]

Aufgrund der Nutzung eines WebView kann *Ionic Framework* native UI-Elemente nur schwer in die eigene Oberfläche integrieren. Der WebView benutzt sein eigenes internes Layout, welches nicht mit dem Layout der nativen UI synchronisiert ist. Dadurch ist es aufwendig, native Komponenten relativ zu denen im WebView zu positionieren. Native Elemente, die absolut positioniert sind oder den ganzen Bildschirm bedecken lassen sich entsprechend benutzen. [Mor]
Mittels *Capacitor* lässt sich nativer Code und Bibliotheken einbinden [Cap].

Xamarin unterstützt ebenfalls nativen Code, native Bibliotheken und native UI, indem ein „Binding“ erzeugt wird. [Comi; Comr; Comq; Comw].

		Flutter		***React Native***		***Ionic***		***Xamarin***	
Native UI	4	Ja	4	Ja	4	nur absolut	2	Ja	4
Nativer Code	6	Ja	6	Ja	6	Ja	6	Ja	6
Summe			**10**		**10**		**8**		**10**

Abb. 5.5: Punktevergabe für *K7: Erweiterbarkeit*

K8: Aussehen und Verhalten

Flutter nutzt keine nativen UI-Elemente, sondern zeichnet die gesamte Benutzeroberfläche mittels Widgets selbst. Diese Widgets können nach *Material Design* Richtlinien für Android oder mit *Cupertino* iOS-ähnlich designt sein. Dadurch ist eine native Nutzererfahrung zu erwarten [Devo; Devf].
Ionic Framework benutzt Webtechnologie zum Darstellen der Benutzeroberfläche. Diese kann an die Plattform angepasst, um eine native Benutzererfahrung zu erzielen. [Ione]
React Native und *Xamarin* benutzen native UI-Elemente.

Arbeiten zu diesem Thema kommen zu dem Schluss, dass moderne Cross-Platform-App-Frameworks native Nutzererfahrung bieten können. [Fre18; MG17]
Da *React Native* und *Xamarin* native Komponenten benutzen, erhalten sie `10` Punkte. *Flutter* und *Ionic Framework* erhalten `8` Punkte, da sie durch plattformspezifisches Styling ebenfalls nativ wirken können.

K9: Performance

Drei Arbeiten, welche die Performance verschiedener Cross-Platform-App-Frameworks betrachten, werden beschrieben, nummeriert und anschließend zusammengefasst.

Eine im Jahr 2020 veröffentlichte Arbeit (1) betrachtet eine native Android-App sowie mit *Ionic Framework*, *Flutter*, *React Native* und zwei weiteren Frameworks erstellte Apps. [BH+20]
Diese werden bezüglich Prozessorauslastung, Arbeitsspeicherverbrauch und Ausführgeschwindigkeit von API-Zugriffen auf den Beschleunigungssensor, die Kontakte, das Dateisystem und den Standort analysiert. Beschränkt auf die drei Frameworks, die im Rahmen dieser Arbeit betrachtet werden, wurde *Flutter* am schnellsten mit den API-Zugriffen fertig, gefolgt von *React Native* und *Ionic Framework*. Die niedrigste Prozessorauslastung hatten *Flutter*, *Ionic Framework* und *React Native*. *React Native* hatte den niedrigsten Speicherverbrauch, gefolgt von *Flutter* und *Ionic Framework*. Die native App war in allen Metriken vergleichbar oder besser. [BH+20]

Eine weitere Arbeit (2) aus dem Jahr 2020 vergleicht die Prozessorauslastung, den Speicherverbrauch, den Akkuverbrauch und die Ausführgeschwindigkeit einer Bilderkennungssoftware. Untersucht wurden *Xamarin*, *Flutter* und eine native Android-Applikation. [BA20]
Flutter und *Xamarin* hatten eine ähnliche Prozessorauslastung. *Flutter* hat etwas mehr Arbeitsspeicher verbraucht als *Xamarin*. *Xamarin* hatte den höchsten Akkuverbrauch, *Flutter* den niedrigsten. *Xamarin* hat bedeutend länger zum Ausführen der Bilderkennungssoftware gebraucht, als die native App und *Flutter*. Die native App war in allen Metriken, außer Akkuverbrauch, vergleichbar oder besser. [BA20]

Eine Arbeit (3) aus dem Jahr 2021 vergleicht auf Android und iOS *Flutter*, *Xamarin* und *React Native* mit jeweils einer nativen App. [Naw+21]
Es wurden die Größe der App, Start bis zum Laden der App, der Speicherverbrauch und die Prozessorauslastung betrachtet. Um Prozessor- und Arbeitsspeicherbelastung zu erzeugen hat die App im Hintergrund Sinuswerte berechnet sowie HTTP-Anfragen an einen Webserver geschickt.
Xamarin hatte die längste Ladezeit, die größte App und mit *React Native* zusammen die höchste Prozessorauslastung. *Flutter* hatte den höchsten Arbeitsspeicherverbrauch und die niedrigste Prozessorauslastung. *React Native* und *Flutter* hatten vergleichbar lange Ladezeiten und Appgrößen. In allen Metriken besser waren hier die nativen iOS- und Androidapps. [Naw+21]

Diese Daten werden pro Arbeit auf die Zahlen `1` (10% der nativen Leistung) bis `10` (100% der native Leistung) prozentual zur nativen App abgebildet und tabellarisch aufgetragen (siehe Tab. 5.6). War die Leistung besser als die einer nativen App, so sind Werte über `10` möglich. Durch Mittelwertbildung aller Datenpunkte wird die Punktzahl des Kriteriums bestimmt.

	Flutter			***React Native***			***Ionic***			***Xamarin***		
	(1)	(2)	(3)	(1)	(2)	(3)	(1)	(2)	(3)	(1)	(2)	(3)
RAM	5	6	4	9		6	5				6	6
CPU	9	15	6	7		5	3				15	5
Appgröße			6			6						4
Ladezeit			6			6						4
A. gesch.	8	11		4			3				2	
Mittelwert	76/10 ≈ **8**			43/7 ≈ **6**			11/3 ≈ **4**			42/7 ≈ **6**		

Abb. 5.6: Punktevergabe für *K9: Performance*

5.1 Nutzwertanalyse

In Tabelle 5.7 wird die Nutzwertanalyse durchgeführt. *React Native* hat mit 866 die höchste Wertung, gefolgt von *Flutter* mit 842. *Xamarin* und *Ionic Framework* erzielen mit 751 und 733 Punkten eine niedrigere Wertung.

Kriterien und Gewichtung		***Flutter***		***React Native***		***Ionic***		***Xamarin***	
		roh	*gewi.*	*roh*	*gewi.*	*roh*	*gewi.*	*roh*	*gewi.*
K1: Entwicklerwerkzeuge	15	10	**150**	8	**120**	7	**105**	5	**75**
K2: Entwicklungsplattformen	05	10	**50**	10	**50**	10	**50**	8	**40**
K3: Einarbeitungszeit	20	7	**140**	9	**180**	8	**160**	5	**100**
K4: Veröffentlichung	08	6	**48**	6	**48**	7	**56**	7	**56**
K5: Testbarkeit	10	10	**100**	10	**100**	10	**100**	10	**100**
K6: Plattformzugriff	06	9	**54**	8	**48**	9	**54**	10	**60**
K7: Erweiterbarkeit	06	10	**60**	10	**60**	8	**48**	10	**60**
K8: Aussehen und Verhalten	20	8	**160**	10	**200**	8	**160**	10	**200**
K9: Performance	10	8	**80**	6	**60**	4	**40**	6	**60**
Wertung			**842**		**866**		**773**		**751**

Abb. 5.7: Nutzwertanalyse

6 Zusammenfassung und Ausblick

Die vorliegende Arbeit hat aus der Literatur Kriterien zur Vorauswahl und Evaluation von Cross-Platform-App-Frameworks abgeleitet. Es wurden vier Frameworks selektiert. Diese wurden dann mithilfe der Literatur evaluiert. Die Ergebnisse wurden in einer Nutzwertanalyse zusammengetragen.

Diese kommt zu dem Schluss, dass *React Native* das Framework mit der größten Eignung für den spezifischen Anwendungsfall ist, eng gefolgt von *Flutter*. *Ionic Framework* und *Xamarin* sind weniger geeignet.

Ein Aspekt, der in dieser Arbeit nicht beleuchtet wurde, ist Sicherheit. So gelten vor allem hybride Frameworks als unsicher, da durch *Cross-Site-Scripting*, kurz *XSS*, Schadcode innerhalb des WebView ausgeführt werden kann. Da dieser WebView durch die Schnittstellen des Frameworks direkten Zugang zu dem Gerät hat, kann dieser potenziell fatalen Schaden anrichten. [BHGG18]
XSS-Schwachstellen sind auf Platz drei der „OWASP Top Ten“, einer Liste der kritischsten Sicherheitsrisiken. [Owa]

Das Gebiet der Cross-Platform-App-Frameworks ist hochinnovativ. Folglich sind ältere Studien und Vergleiche nur bedingt aussagekräftig. Außerdem beschäftigen sich ältere Arbeiten mit Frameworks, deren Entwicklung heute eingestellt ist, wie etwa *Adobe PhoneGap*, oder die stark an Relevanz verloren haben. [BHGG18]
Um dem Anspruch an Innovation und Entwicklungstätigkeit zu genügen, wurden vornehmlich jüngere Quellen herangezogen.

Die Bewertung von *K8: Aussehen und Verhalten* und *K9: Performance* hat sich als besonders schwierig herausgestellt. Die Literatur zu nativem Look-And-Feel einer App ist, noch immer, für eine fundierte Aussage unzureichend. [BHGG18]
Die Leistung eines Frameworks ist sehr schwer zu evaluieren, da es viele Metriken gibt, die erfasst werden können, und sehr viele Faktoren, die es zu berücksichtigen gilt. Da es keine standardisierte Form für die Leistungsbewertung gibt, ist es schwierig, verschiedene Arbeiten zu vergleichen oder zusammenzufassen. Durch die Fragmentierung der Hardware und Software von mobilen Endgeräten wird das Zusammenfassen verschiedener Arbeiten nahezu unmöglich. [BH+20; BA20]

Ein weiteres hochaktuelles Thema sind PWA, die im Rahmen dieser Arbeit nur kurz betrachtet und in *K4: Veröffentlichung* in die Evaluation einflossen. Vor allem für Apps, die nicht an Hardware gebunden sind, bieten PWA eine ausgezeichnete Alternative. [BHMG17; MBHG18]

Einige Frameworks, die noch zu jung für eine tiefe Betrachtung als Cross-Platform-App-Framework sind, sind *Avalonia UI*, *Kotlin Multiplatform Mobile* und *Tauri*. Alle diese Frameworks sind bezüglich mobilen Plattformen noch in der Beta, könnten aber eine Rolle für die plattformübergreifende Appentwicklung spielen.

Abbildungsverzeichnis

Literatur

[al.21] Sacha Greif Et al. *The State of JS 2021*. 2021. URL: https://2021.stateofjs.com/en-US/libraries/mobile-desktop (besucht am 07. 11. 2022).

[Ama] *Gesetz über den Amateurfunk*. 1997. URL: https://www.gesetze-im-internet.de/afug_1997/AFuG_1997.pdf (besucht am 30. 11. 2022).

[App] URL: https://developer.apple.com/forums/thread/76587 (besucht am 15. 12. 2022).

[Asi] Ejiro Asiuwhu. *NativeScript vs. React Native*. URL: https://blog.logrocket.com/nativescript-react-native/ (besucht am 12. 12. 2022).

[BA20] Max Bekkhus und Lucas Arvidsson. *Resource utilization and performance: A comparative study on mobile crossplatform tools*. 2020. URL: https://www.diva-portal.org/smash/get/diva2:1462226/FULLTEXT01.pdf.

[BH+20] Andreas Biørn-Hansen u. a. „An empirical investigation of performance overhead in cross-platform mobile development frameworks“. In: *Empirical Software Engineering* 25.4 (2020), S. 2997–3040. URL: https://link.springer.com/article/10.1007/s10664-020-09827-6.

[BHG18] Andreas Biørn-Hansen und Gheorghita Ghinea. „Bridging the gap: investigating device-feature exposure in cross-platform development“. In: (2018). URL: https://aisel.aisnet.org/cgi/viewcontent.cgi?article=1713&context=hicss-51.

[BHGG18] Andreas Biørn-Hansen, Tor-Morten Grønli und Gheorghita Ghinea. „A survey and taxonomy of core concepts and research challenges in cross-platform mobile development“. In: *ACM Computing Surveys (CSUR)* 51.5 (2018), S. 1–34. URL: https://dl.acm.org/doi/pdf/10.1145/3241739.

[BHGG19] Andreas Biørn-Hansen, Tor-Morten Grønli und Gheorghita Ghinea. „Animations in cross-platform mobile applications: An evaluation of tools, metrics and performance“. In: *Sensors* 19.9 (2019), S. 2081.

[BHMG17] Andreas Biørn-Hansen, Tim A Majchrzak und Tor-Morten Grønli. „Progressive web apps for the unified development of mobile applications“. In: *International Conference on Web Information Systems and Technologies*. Springer. 2017, S. 64–86. URL: https://link.springer.com/chapter/10.1007/978-3-319-93527-0_4.

[Bit] Bitrise. *Powering the world's best mobile engineering teams*. URL: https://www.bitrise.io/pricing (besucht am 13. 12. 2022).

[Bui] Adobe PhoneGap Build. *Ende des Dienstes für Adobe PhoneGap*. URL: https://helpx.adobe.com/de/experience-manager/kb/adobe-phonegap-end-of-service.html (besucht am 12. 12. 2022).

[Cap] *Plugin Development Workflow*. URL: https://capacitorjs.com/docs/plugins/workflow (besucht am 14. 12. 2022).

[Cod] Codemagic. *Predictable pricing. Pay as you go or unlimited usage for fixed price.* URL: https://codemagic.io/pricing/ (besucht am 13. 12. 2022).

[Coma] Capacitor Community. *FAQs.* URL: https://capacitorjs.com/docs/getting-started/faqs (besucht am 12. 12. 2022).

[Comb] Ionic Framework Community. *Android Play Store Deployment.* URL: https://ionicframework.com/docs/deployment/play-store (besucht am 13. 12. 2022).

[Comc] Ionic Framework Community. *Debugging.* URL: https://ionicframework.com/docs/troubleshooting/debugging (besucht am 13. 12. 2022).

[Comd] Ionic Framework Community. *Globalization.* URL: https://ionicframework.com/docs/native/globalization (besucht am 12. 12. 2022).

[Come] Ionic Framework Community. *iOS App Store Deployment.* URL: https://ionicframework.com/docs/deployment/app-store (besucht am 13. 12. 2022).

[Comf] Ionic Framework Community. *Live Reload.* URL: https://ionicframework.com/docs/cli/livereload (besucht am 13. 12. 2022).

[Comg] Ionic Framework Community. *Native APIs.* URL: https://ionicframework.com/docs/native (besucht am 14. 12. 2022).

[Comh] Ionic Framework Community. *Progressive Web Apps in Angular.* URL: https://ionicframework.com/docs/angular/pwa (besucht am 13. 12. 2022).

[Comi] Xamarin Community. *Binding a Java Library.* URL: https://learn.microsoft.com/en-us/xamarin/android/platform/binding-java-library/ (besucht am 14. 12. 2022).

[Comj] Xamarin Community. *Debug on an Android device.* URL: https://learn.microsoft.com/en-us/xamarin/android/deploy-test/debugging/debug-on-device?tabs=windows (besucht am 12. 12. 2022).

[Comk] Xamarin Community. *Debugging Xamarin.iOS Apps.* URL: https://learn.microsoft.com/en-us/xamarin/ios/deploy-test/debugging-in-xamarin-ios?tabs=macos (besucht am 12. 12. 2022).

[Coml] Xamarin Community. *Installing Xamarin.* URL: https://learn.microsoft.com/en-us/xamarin/get-started/installation (besucht am 13. 12. 2022).

[Comm] Xamarin Community. *Profiling Android Apps.* URL: https://learn.microsoft.com/en-us/xamarin/android/deploy-test/profiling (besucht am 12. 12. 2022).

[Comn] Xamarin Community. *Profiling Xamarin.iOS Applications with Instruments.* URL: https://learn.microsoft.com/en-us/xamarin/ios/deploy-test/using-instruments-to-detect-native-leaks-using-markheap (besucht am 12. 12. 2022).

[Como] Xamarin Community. *Publishing to Google Play.* URL: https://learn.microsoft.com/en-us/xamarin/android/deploy-test/publishing/publishing-to-google-play (besucht am 13. 12. 2022).

[Comp] Xamarin Community. *Publishing Xamarin.iOS apps to the App Store.* URL: https://learn.microsoft.com/en-us/xamarin/ios/deploy-test/app-distribution/app-store-distribution/publishing-to-the-app-store (besucht am 13. 12. 2022).

[Comq] Xamarin Community. *Referencing Native Libraries in Xamarin.iOS*. URL: https://learn.microsoft.com/en-us/xamarin/ios/platform/native-interop (besucht am 14. 12. 2022).

[Comr] Xamarin Community. *Using Native Libraries*. URL: https://learn.microsoft.com/en-us/xamarin/android/platform/native-libraries (besucht am 14. 12. 2022).

[Coms] Xamarin Community. *What is Xamarin?* URL: https://learn.microsoft.com/en-us/xamarin/get-started/what-is-xamarin (besucht am 12. 12. 2022).

[Comt] Xamarin Community. *What is Xamarin.Forms?* URL: https://learn.microsoft.com/en-us/xamarin/get-started/what-is-xamarin-forms (besucht am 12. 12. 2022).

[Comu] Xamarin Community. *Xamarin Profiler*. URL: https://learn.microsoft.com/en-us/xamarin/tools/profiler (besucht am 12. 12. 2022).

[Comv] Xamarin Community. *Xamarin.Essentials*. URL: https://learn.microsoft.com/en-us/xamarin/essentials/ (besucht am 14. 12. 2022).

[Comw] Xamarin Community. *Xamarin.Forms - Native Forms*. URL: https://learn.microsoft.com/en-us/samples/xamarin/xamarin-forms-samples/native2forms/ (besucht am 12. 12. 2022).

[Comx] Xamarin Community. *Xamarin.Forms Localization*. URL: https://learn.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/localization/ (besucht am 12. 12. 2022).

[Dapa] .

[Dapb] URL: https://apps.apple.com/ca/app/dapnet/id1638627303?platform=iphone (besucht am 15. 12. 2022).

[Dar] URL: https://pub.dev (besucht am 14. 12. 2022).

[Dav] Ryan Davis. *From Xamarin Native to Xamarin.Forms: Reaping the Rewards without the Risk*. URL: https://www.codemag.com/Article/1911092/From-Xamarin-Native-to-Xamarin.Forms-Reaping-the-Rewards-without-the-Risk (besucht am 12. 12. 2022).

[Deva] Flutter Developers. *Binding to native Android code using dart:ffi*. URL: https://docs.flutter.dev/development/platform-integration/android/c-interop (besucht am 14. 12. 2022).

[Devb] Flutter Developers. *Binding to native iOS code using dart:ffi*. URL: https://docs.flutter.dev/development/platform-integration/ios/c-interop (besucht am 14. 12. 2022).

[Devc] Flutter Developers. *Build and release a web app*. URL: https://docs.flutter.dev/deployment/web (besucht am 13. 12. 2022).

[Devd] Flutter Developers. *Build and release an Android app*. URL: https://docs.flutter.dev/deployment/android (besucht am 13. 12. 2022).

[Deve] Flutter Developers. *Build and release an iOS app*. URL: https://docs.flutter.dev/deployment/ios (besucht am 13. 12. 2022).

[Devf] Flutter Developers. *Cupertino (iOS-style) widgets*. URL: https://docs.flutter.dev/development/ui/widgets/cupertino (besucht am 14. 12. 2022).

[Devg] Flutter Developers. *Debugging Flutter apps*. URL: https://docs.flutter.dev/testing/debugging (besucht am 12. 12. 2022).

[Devh] Flutter Developers. *FAQ*. URL: https://docs.flutter.dev/resources/faq (besucht am 30. 11. 2022).

[Devi] Flutter Developers. *Flutter Architectural Overview*. URL: https://docs.flutter.dev/resources/architectural-overview (besucht am 12. 12. 2022).

[Devj] Flutter Developers. *Flutter SDK Releases*. URL: https://docs.flutter.dev/development/tools/sdk/releases (besucht am 12. 12. 2022).

[Devk] Flutter Developers. *Hosting native Android views in your Flutter app with Platform Views*. URL: https://docs.flutter.dev/development/platform-integration/android/platform-views (besucht am 14. 12. 2022).

[Devl] Flutter Developers. *Hosting native iOS views in your Flutter app with Platform Views*. URL: https://docs.flutter.dev/development/platform-integration/ios/platform-views?tab=ios-platform-views-swift-tab (besucht am 14. 12. 2022).

[Devm] Flutter Developers. *Hot reload*. URL: https://docs.flutter.dev/development/tools/hot-reload (besucht am 12. 12. 2022).

[Devn] Flutter Developers. *Internationalizing Flutter apps*. URL: https://docs.flutter.dev/development/accessibility-and-localization/internationalization (besucht am 12. 12. 2022).

[Devo] Flutter Developers. *Material Components widgets*. URL: https://docs.flutter.dev/development/ui/widgets/material (besucht am 14. 12. 2022).

[Devp] Flutter Developers. *Testing Flutter apps*. URL: https://docs.flutter.dev/testing (besucht am 14. 12. 2022).

[Devq] Flutter Developers. *Using an OEM debugger*. URL: https://docs.flutter.dev/testing/oem-debuggers (besucht am 12. 12. 2022).

[Devr] Flutter Developers. *Writing custom platform-specific code*. URL: https://docs.flutter.dev/development/platform-integration/platform-channels (besucht am 14. 12. 2022).

[DPJ21] Blaž Denko, Špela Pečnik und Iztok Fister Jr. „A Comprehensive Comparison of Hybrid Mobile Application Development Frameworks“. In: *International Journal of Security and Privacy in Pervasive Computing (IJSPPC)* 13.5 (1 2021), S. 13. DOI: 10.4018/IJSPPC.2021010105. URL: https://www.researchgate.net/profile/Iztok-Fister-Jr/publication/348132143_A_Comprehensive_Comparison_of_Hybrid_Mobile_Application_Development_Frameworks/links/5ff3795a299bf140886ffc56/A-Comprehensive-Comparison-of-Hybrid-Mobile-Application-Development-Frameworks.pdf.

[Ego] Vyacheslav Egorov. *Introduction to Dart VM*. URL: https://mrale.ph/dartvm (besucht am 12. 12. 2022).

[EK+17] 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.

[Expa] Expo. *Add custom native code.* URL: https://docs.expo.dev/workflow/customizing/ (besucht am 13. 12. 2022).

[Expb] Expo. *Common questions.* URL: https://docs.expo.dev/introduction/faq/ (besucht am 13. 12. 2022).

[Expc] Expo. *EAS Submit.* URL: https://docs.expo.dev/submit/introduction/ (besucht am 13. 12. 2022).

[Expd] Expo. *EAS Update.* URL: https://docs.expo.dev/eas-update/introduction/ (besucht am 13. 12. 2022).

[Expe] Expo. *Incredible apps. Pricing to match.* URL: https://expo.dev/pricing (besucht am 13. 12. 2022).

[FfE20] FfE. *Was ist eine Nutzwert-Analyse?* Juli 2020. URL: https://www.ffe.de/veroeffentlichungen/was-ist-eine-nutzwert-analyse/ (besucht am 08. 12. 2022).

[Fou] The Apache Software Foundation. *Cordova Plugins.* URL: https://cordova.apache.org/plugins/ (besucht am 12. 12. 2022).

[Fre18] Rasmus Fredrikson. *Emulating a Native Mobile Experience with Cross-platform Applications.* 2018. URL: https://www.diva-portal.org/smash/get/diva2:1246007/FULLTEXT01.pdf.

[Git] GitHub. *About billing for GitHub Actions.* URL: https://docs.github.com/en/billing/managing-billing-for-github-actions/about-billing-for-github-actions (besucht am 13. 12. 2022).

[Goo] URL: https://support.google.com/googleplay/android-developer/answer/6112435\#zippy=,step-pay-registration-fee (besucht am 15. 12. 2022).

[Gri] Simon Grimm. *Debugging.* URL: https://ionic.io/blog/debugging-tips-for-your-ionic-app (besucht am 13. 12. 2022).

[Grø+14] Tor-Morten Grønli u. a. „Mobile Application Platform Heterogeneity: Android vs Windows Phone vs iOS vs Firefox OS“. In: *2014 IEEE 28th International Conference on Advanced Information Networking and Applications.* 2014, S. 635–641. DOI: 10.1109/AINA.2014.78. URL: https://ieeexplore.ieee.org/abstract/document/6838724.

[Iona] Ionic. *Environment Setup.* URL: https://ionicframework.com/docs/intro/environment (besucht am 13. 12. 2022).

[Ionb] Ionic. *Live Update.* URL: https://ionic.io/docs/appflow/deploy/intro (besucht am 13. 12. 2022).

[Ionc] Ionic. *Mobile DevOps plans suited for any team.* URL: https://ionic.io/appflow/pricing (besucht am 13. 12. 2022).

[Iond] Ionic. *Releases.* URL: https://ionicframework.com/docs/reference/release-notes (besucht am 12. 12. 2022).

[Ione] *Platform Styles.* URL: https://ionicframework.com/docs/theming/platform-styles (besucht am 14. 12. 2022).

[Ionf] *Testing.* URL: https://ionicframework.com/docs/angular/testing (besucht am 14. 12. 2022).

[JB13] Slinger Jansen und Ewoud Bloemendal. „Defining App Stores: The Role of Curated Marketplaces in Software Ecosystems“. In: *Software Business. From Physical Products to Software Services and Solutions*. Hrsg. von Georg Herzwurm und Tiziana Margaria. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, S. 195–206. ISBN: 978-3-642-39336-5. URL: https://link.springer.com/chapter/10.1007/978-3-642-39336-5_19.

[Jet21] JetBrains. *Cross-platform Mobile Frameworks Used by Software Developers Worldwide from 2019 to 2021*. 2021. URL: https://www.jetbrains.com/lp/devecosystem-2021/miscellaneous/ (besucht am 27. 11. 2022).

[Liea] Christian Liebel. *Adobe stellt PhoneGap ein: Drama oder Chance?* URL: https://www.heise.de/blog/Adobe-stellt-PhoneGap-ein-Drama-oder-Chance-4868133.html (besucht am 12. 12. 2022).

[Lieb] Christian Liebel. *Capacitor oder Cordova: Welche Plattform sollten Entwickler webbasierter Mobilapps wählen?* URL: https://www.heise.de/blog/Capacitor-oder-Cordova-Welche-Plattform-sollten-Entwickler-webbasierter-Mobilapps-waehlen-4690975.html (besucht am 12. 12. 2022).

[Lym] saac Lyman. *Flutter vs. React Native: Which is the right cross-platform framework for you?* URL: https://stackoverflow.blog/2022/10/31/comparing-frameworks-for-cross-platform-apps-flutter-vs-react-native/ (besucht am 14. 12. 2022).

[Lyna] Max Lynch. *Announcing The Ionic Framework*. URL: https://ionic.io/blog/announcing-ionic (besucht am 12. 12. 2022).

[Lynb] Max Lynch. *Capacitor vs Cordova: What are the Differences Between Them*. URL: https://ionic.io/resources/articles/capacitor-vs-cordova-modern-hybrid-app-development\#h-migrating-to-capacitor-is-easy:-capacitor-is-backwards-compatible-with-cordova (besucht am 12. 12. 2022).

[Lync] Max Lynch. *Ionic isn't Cordova Anymore*. URL: https://ionic.io/blog/ionic-isnt-cordova-anymore (besucht am 12. 12. 2022).

[MBHG18] Tim A Majchrzak, Andreas Biørn-Hansen und Tor-Morten Grønli. „Progressive web apps: the definite approach to cross-platform development?“ In: (2018). URL: https://scholarspace.manoa.hawaii.edu/server/api/core/bitstreams/9564e191-55e2-4042-a27e-88bda679d4fe/content.

[Mer] Codrina Merigo. *Testing Your Xamarin Apps*. URL: https://www.youtube.com/watch?v=HOfmteIoSV4 (besucht am 14. 12. 2022).

[MG17] Tim Majchrzak und Tor-Morten Grønli. „Comprehensive analysis of innovative cross-platform app development frameworks“. In: *Proceedings of the 50th Hawaii International Conference on System Sciences*. 2017. URL: https://scholarspace.manoa.hawaii.edu/server/api/core/bitstreams/7a6c6ce1-e508-4528-9a35-a71033d1a615/content.

[Mic] Microsoft. *Visual Studio Tools für Xamarin*. URL: https://visualstudio.microsoft.com/de/xamarin/ (besucht am 12. 12. 2022).

[Mor] Josh Morony. *How to Launch Native Views in an Ionic Application*. URL: https://eliteionic.com/tutorials/how-to-launch-native-views-in-an-ionic-application (besucht am 14. 12. 2022).

[MRT] MRT. *Capacitor or Cordova: Which platform should developers of web-based mobile apps choose?* URL: https://marketresearchtelecast.com/capacitor-or-cordova-which-platform-should-developers-of-web-based-mobile-apps-choose/210132/ (besucht am 12. 12. 2022).

[Nat] NativeScript. *Releases.* URL: https://github.com/NativeScript/NativeScript/releases (besucht am 12. 12. 2022).

[Naw+21] Piotr Nawrocki u. a. „A Comparison of Native and Cross-Platform Frameworks for Mobile Applications". In: *Computer* 54.3 (2021), S. 18–27. DOI: 10.1109/MC.2020.2983893. URL: https://ieeexplore.ieee.org/abstract/document/9378923.

[Occ] Tom Occhino. *React Native: Bringing modern web techniques to mobile.* URL: https://engineering.fb.com/2015/03/26/android/react-native-bringing-modern-web-techniques-to-mobile/ (besucht am 12. 12. 2022).

[Ove22] Stack Overflow. *2022 Developer Survey.* 2022. URL: https://survey.stackoverflow.co/2022 (besucht am 28. 11. 2022).

[Owa] *OWASP Top Ten.* URL: https://owasp.org/www-project-top-ten/ (besucht am 15. 12. 2022).

[PDL13] Joachim Perchat, Mikael Desertot und Sylvain Lecomte. „Component based Framework to Create Mobile Cross-platform Applications". In: *Procedia Computer Science* 19 (2013). The 4th International Conference on Ambient Systems, Networks and Technologies (ANT 2013), the 3rd International Conference on Sustainable Energy Information Technology (SEIT-2013), S. 1004–1011. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2013.06.140. URL: https://www.sciencedirect.com/science/article/pii/S1877050913007485.

[Pit] Sten Pittet. *Unterschiedliche Arten von Softwaretests.* URL: https://www.atlassian.com/de/continuous-delivery/software-testing/types-of-software-testing (besucht am 14. 12. 2022).

[PSC12] Manuel Palmieri, Inderjeet Singh und Antonio Cicchetti. „Comparison of cross-platform mobile development tools". In: *2012 16th International Conference on Intelligence in Next Generation Networks.* 2012, S. 179–186. DOI: 10.1109/ICIN.2012.6376023. URL: https://ieeexplore.ieee.org/abstract/document/6376023.

[RA] Amateurfunkgruppe der RWTH Aachen. *Home | DAPNet Web.* URL: https://hampager.de (besucht am 08. 12. 2022).

[Reaa] URL: https://react.i18next.com (besucht am 12. 12. 2022).

[Reab] URL: https://reactnative.directory (besucht am 14. 12. 2022).

[Reac] *React Native Tools.* URL: https://marketplace.visualstudio.com/items?itemName=msjsdiag.vscode-react-native (besucht am 13. 12. 2022).

[RM18] Christoph Rieger und Tim A. Majchrzak. „A Taxonomy for App-Enabled Devices: Mastering the Mobile Device Jungle". In: *Web Information Systems and Technologies.* Hrsg. von Tim A. Majchrzak u. a. Cham: Springer International Publishing, 2018, S. 202–220. ISBN: 978-3-319-93527-0. URL: https://link.springer.com/chapter/10.1007/978-3-319-93527-0_10.

[RM19] 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.

[RNCa] Meta Platforms Inc. React Native Community. *Android Native UI Components*. URL: https://reactnative.dev/docs/native-components-android (besucht am 14. 12. 2022).

[RNCb] Meta Platforms Inc. React Native Community. *Architecture Overview*. URL: https://reactnative.dev/architecture/overview (besucht am 12. 12. 2022).

[RNCc] Meta Platforms Inc. React Native Community. *Debugging*. URL: https://reactnative.dev/docs/debugging (besucht am 12. 12. 2022).

[RNCd] Meta Platforms Inc. React Native Community. *Fast Refresh*. URL: https://reactnative.dev/docs/fast-refresh (besucht am 12. 12. 2022).

[RNCe] Meta Platforms Inc. React Native Community. *Introduction*. URL: https://reactnative.dev/docs/the-new-architecture/landing-page (besucht am 12. 12. 2022).

[RNCf] Meta Platforms Inc. React Native Community. *iOS Native UI Components*. URL: https://reactnative.dev/docs/native-components-ios (besucht am 14. 12. 2022).

[RNCg] Meta Platforms Inc. React Native Community. *Native Modules Intro*. URL: https://reactnative.dev/docs/native-modules-intro (besucht am 14. 12. 2022).

[RNCh] Meta Platforms Inc. React Native Community. *Profiling*. URL: https://reactnative.dev/docs/profiling (besucht am 12. 12. 2022).

[RNCi] Meta Platforms Inc. React Native Community. *Publishing to Apple App Store*. URL: https://reactnative.dev/docs/publishing-to-app-store (besucht am 13. 12. 2022).

[RNCj] Meta Platforms Inc. React Native Community. *Publishing to Google Play Store*. URL: https://reactnative.dev/docs/signed-apk-android (besucht am 13. 12. 2022).

[RNCk] Meta Platforms Inc. React Native Community. *Releases*. URL: https://github.com/facebook/react-native/releases (besucht am 12. 12. 2022).

[RNCl] Meta Platforms Inc. React Native Community. *Setting up the development environment*. URL: https://reactnative.dev/docs/environment-setup (besucht am 13. 12. 2022).

[RNCm] Meta Platforms Inc. React Native Community. *Testing*. URL: https://reactnative.dev/docs/testing-overview (besucht am 12. 12. 2022).

[RNCn] Meta Platforms Inc. React Native Community. *Who is using React Native?* URL: https://reactnative.dev/showcase (besucht am 12. 12. 2022).

[RRT12] C.P Rahul Raj und Seshu Babu Tolety. „A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach“. In: *2012 Annual IEEE India Conference (INDICON)*. 2012, S. 625–629. DOI: 10.1109/INDCON.2012.6420693. URL: https://ieeexplore.ieee.org/abstract/document/6420693.

[San] Tiago Santo. *Test and deploy an iOS App with GitHub Actions*. URL: https://engineering.talkdesk.com/test-and-deploy-an-ios-app-with-github-actions-44de9a7dcef6?gi=debbb5ece115 (besucht am 13. 12. 2022).

[Sca+19] Simone Scalabrino u. a. „Data-Driven Solutions to Detect API Compatibility Issues in Android: An Empirical Study“. In: *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. 2019, S. 288–298. DOI: 10.1109/MSR.2019.00055. URL: https://ieeexplore.ieee.org/document/8816731.

[Sin] Adam Sinicki. *How to create non-game apps in Unity*. URL: https://www.androidauthority.com/make-unity-apps-1073017/ (besucht am 13. 12. 2022).

[Spe] Ben Sperry. *Announcing Ionic 1.0!* URL: https://ionic.io/blog/announcing-ionic-1-0 (besucht am 12. 12. 2022).

[Sta22a] Statcounter. *Desktop Operating System Market Share German*. Sep. 2022. URL: https://gs.statcounter.com/os-market-share/desktop/germany (besucht am 25. 10. 2022).

[Sta22b] Statcounter. *Mobile Operating System Market Share Germany*. Sep. 2022. URL: https://gs.statcounter.com/os-market-share/mobile/germany (besucht am 25. 10. 2022).

[Sta22c] Statcounter. *Operating System Market Share Germany*. Sep. 2022. URL: https://gs.statcounter.com/os-market-share/all/germany (besucht am 25. 10. 2022).

[Sta22d] Statcounter. *Tablet Operating System Market Share German*. Sep. 2022. URL: https://gs.statcounter.com/os-market-share/tablet/germany (besucht am 25. 10. 2022).

[Sta22e] Statista. *Consumer electronics ownership in Germany in 2022*. 2022. URL: https://www.statista.com/forecasts/998736/consumer-electronics-ownership-in-germany (besucht am 25. 10. 2022).

[Sta22f] Statista. *Outlook: App - Germany*. 2022. URL: https://www.statista.com/outlook/dmo/app/germany?currency=EUR#revenue (besucht am 03. 12. 2022).

[Sto] Valio Stoychev. *Announcing NativeScript - cross-platform framework for building native mobile applications*. URL: https://www.telerik.com/blogs/announcing-nativescript---cross-platform-framework-for-building-native-mobile-applications (besucht am 12. 12. 2022).

[Tan+11] Wei Tang u. a. „Multi-Platform Mobile Thin Client Architecture in Cloud Environment“. In: *Procedia Environmental Sciences* 11 (2011). 2011 2nd International Conference on Challenges in Environmental Science and Computer Engineering (CESCE 2011), S. 499–504. ISSN: 1878-0296. DOI: https://doi.org/10.1016/j.proenv.2011.12.079. URL: https://www.sciencedirect.com/science/article/pii/S1878029611009029.

[TP22] Deutsche TV-Plattform. *Share of internet-enabled television sets (smart TVs) in total TV sales in Germany from 2010 to 2021*. Jan. 2022. URL: https://www.statista.com/statistics/485443/smart-tv-share-of-total-tv-sales-germany/ (besucht am 25. 10. 2022).

[VuM21] VuMA. *Number of smartphone users in Germany from January 2009 to 2021*. Nov. 2021. URL: https://www.statista.com/statistics/461801/number-of-smartphone-users-in-germany/ (besucht am 24. 10. 2022).

[Xam] *Xamarin.UITest*. URL: `https://learn.microsoft.com/en-us/appcenter/test-cloud/frameworks/uitest/` (besucht am 14. 12. 2022).