BROCKHAUS AG

APP DEVELOPMENT

Die Tücken und Möglichkeiten der App-Entwicklung

WHITEPAPER

Philip Hahn
IT-Consulting

KURZGEFASST

App – diese Abkürzung hat seit dem Siegeszug der Smartphones ihren Einzug in das allgemeine Vokabular gefunden. Apps erledigen häufig kleinere oder einzelne Aufgaben, um eine einfache Bedienbarkeit zu ermöglichen.

Die Mobilisierung der Computerlandschaft durch tragbare Geräte verändert das Nutzungsverhalten der Anwender. Dadurch braucht es mehr als nur eine Desktop-Anwendung, um eine breite Masse an Nutzern zu erreichen.

Bei dem Wunsch nach einer App sollten mehrere Fragen vorab beantwortet werden, wie die Frage nach den unterstützten Bildschirmgrößen, welche Plattformen bedient werden und wie hoch die Kosten für die Entwicklung sein dürfen. Auch die gewünschte Nutzererfahrung und die benötigten Funktionen müssen berücksichtig werden. Entsprechend der jeweiligen Antworten auf diese Fragen gibt es unterschiedliche Entwicklungsphilosophien, die in Betracht gezogen werden können: Nativ, Cross-Plattform oder Cross-Compilation.

Die Vor- und Nachteile der einzelnen Methoden werden, soweit möglich, in diesem Dokument u.a. anhand von Fallbeispielen aufgezeigt. Ziel des Whitepapers ist jedoch nicht, eine einzige mögliche Art der App-Entwicklung zu skizzieren, sondern aufzuzeigen, wann welche Möglichkeit sinnvoll ist.

1 EINLEITUNG

App - dieses Schlagwort hat spätestens seit dem Siegeszug der Smartphones im Jahr 2007, mit dem ersten iPhone, im Wortschatz der Menschen Einzug gehalten. App ist hierbei die Kurzform für Application (Anwendung) und wurde zu Beginn vor allem benutzt, um ein Programm auf mobilen Geräten zu bezeichnen. Inzwischen hat der Begriff jedoch auch seinen Weg in die Desktop-Betriebssysteme von Microsoft, Google und Apple gefunden. Es wird ebenfalls von Apps gesprochen, wenn aus dem WebStore des Browsers Chrome-Anwendungen heruntergeladen werden. In Googles Betriebssystem Chrome OS, das auf den sogenannten Chromebooks läuft, wird ausschließlich von Apps gesprochen.

Letzten Endes sind Apps nichts anderes als Programme, die in einer bestimmten Programmiersprache geschrieben, kompiliert und als ein Artefakt ausgeliefert werden. Das Artefakt ist die Installationsdatei, wie sie unter Windows, macOS oder Linux zu finden ist. Die Programmiersprache, die Kompilierung und der Typ des Artefakts unterscheiden sich von Betriebssystem zu Betriebssystem.

Ein Spezialfall sind die PWAs (Progressive Web Application). Hierbei handelt es sich um Webseiten, die darauf abzielen, durch hohe Responsivität auf möglichst vielen Bildschirmgrößen funktional zu sein - unabhängig von dem darunter liegenden Betriebssystem. Zusätzlich bieten PWAs eine Offline-Funktionalität, um das Gefühl einer echten App für den Nutzer zu simulieren. Diese Apps bedürfen keinerlei Stores, da sie als Webseiten direkt über eine URL angesteuert und aufgerufen werden.

Apps bieten, anders als die klassischen Computerprogramme, häufig einen kleinen Ausschnitt eines Aufgabengebietes. Sie konzentrieren sich auf das Erledigen einer einzelnen Aufgabe, was sie insgesamt leichter in der Handhabung macht. Sie starten schnell, sind aufgrund der Umgebung (Smartphone/Tablet) jederzeit und überall einsetzbar und vermitteln ein modernes Flair. Hinzu kommt, dass viele Apps kostenlos oder zu minimalen Kosten über den jeweiligen Store angeboten werden. Dies ist bei Computerprogrammen als Shareware bekannt, findet aber lange nicht so eine weite Verbreitung wie bei Apps. Inzwischen sind die Nutzer daran gewöhnt, Werbung in kostenlosen Apps zu konsumieren oder dazu bereit, einen gewissen Betrag zu zahlen, um die App werbefrei zu nutzen.

Alle Apps haben gemeinsam, dass sie über eine zentrale Stelle zum Download bereitstehen. Gemeint sind Stores wie AppStore (Apple), Play Store (Google) oder Enterprise-Lösungen mit einem „eigenem“ HTML-basierten, nativen Android oder iOS Store. Letztere stehen ausschließlich ausgewählten Benutzern zur Verfügung (vgl. digital.ai 2021).

Wird eine App neu oder als ein Update in einem Store angeboten, durchläuft sie automatisch eine Überprüfung. Dabei werden nicht nur Sicherheitsaspekte getestet, wie das mögliche Vorhandensein von Schadcode, sondern auch, ob die App gewissen vorgegebenen Qualitätsstandards und -richtlinien genügt.

Angefangen hat alles mit iOS, dem mobilen Betriebssystem von Apple im Jahr 2007. Google startete 2008 mit Android.

Ab 2013 gab es kein Vorbeikommen mehr an den beiden Plattformen, auch wenn Nokia und Microsoft es mit neuen Systemen probierten.

Alle anderen mobilen Betriebssysteme kommen zusammen auf weniger als 1% Marktanteil. Mit 0.44% sind sie somit faktisch irrelevant.

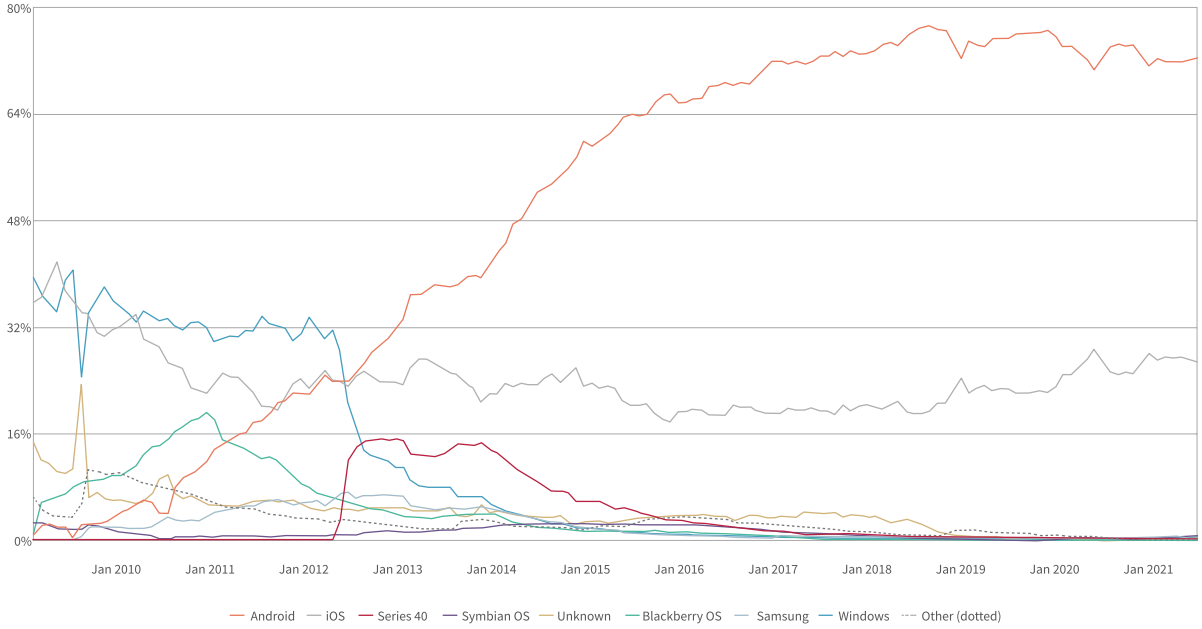


Abbildung 1: Entwicklung mobiler Betriebssysteme von Januar 2009 bis Mai 2021; vgl. https://gs.statcounter.com

Was mobile Betriebssysteme angeht, sind somit nur noch Android und iOS von Bedeutung.

An dieser Stelle lohnt es sich, die Nutzung des Internets zu beleuchten und die Browser als eigene „Plattform“ zu betrachten: Mit der Verbreitung des Smartphones hat das Internet in einem viel größerem Maßstab Einzug in das tägliche Leben der Menschen gehalten und ist zu einem allgegenwärtigen Gut geworden. Aufgrund dieser Tatsache ist der Browser als Plattform auf Standrechnern, Laptops, Tablets und auf Smartphones ein wichtiger Bestandteil.

Im Jahr 2018 nutzen die Deutschen im Durchschnitt 196 Minuten täglich das Internet. Das war zum damaligen Zeitpunkt eine Steigerung von 47 Minuten zum Vorjahr (vgl. DZW, 2018). Wenig überrascht die Altersgruppe der unter 30-Jährigen, die im Durchschnitt doppelt so viel Zeit im Internet verbringt. Die Werte der im Jahr 2020 veröffentlichten Studie liegen generell noch höher (vgl. ARD-ZDF-Onlinestudie, 2020).

Das Aufkommen von Smartphones veränderte auch die Art der Nutzung von Computern. Weg vom klassischen stationären PC über Laptops hin zum Smartphone. Insgesamt spielt der Mobilitätsfaktor des Computers eine immer größere Rolle, um überall und jederzeit auf ihn zurückzugreifen.

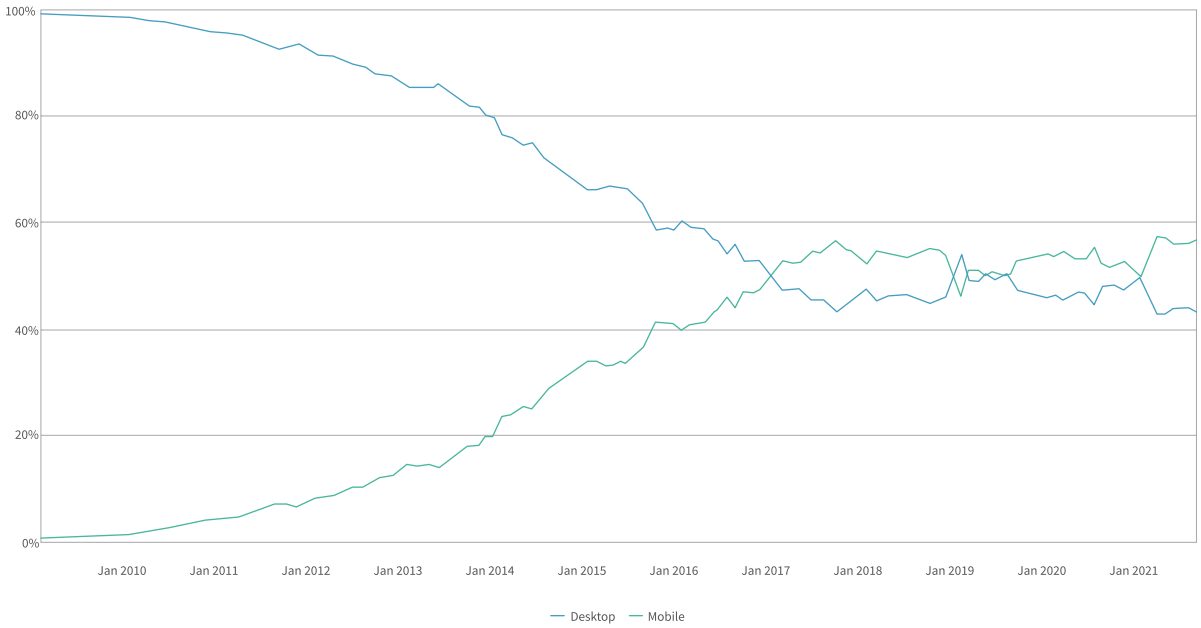


Abbildung 2: Entwicklung Marktanteile Desktop PC im Vergleich zu mobilen Endgeräten; vgl. https://gs.statcounter.com

Diese Entwicklungen zeigen sehr deutlich, dass mit einer reinen Applikation für Standrechner zwar immer noch eine breite Masse erreicht, aber über die Hälfte potenzieller Nutzer außer Acht gelassen wird. In der heutigen Zeit sollte eine Anwendung demnach möglichst auf unterschiedlichen Geräten zur Verfügung stehen. Das sind mobile Geräte wie Smartphones oder Tablets, Laptops, Standrechner oder der Browser als Plattform.

2 ARTEN DER APP-ENTWICKLUNG

Grob unterschieden, existieren drei Strategien der App-Entwicklung: Die Native, Cross-Plattform sowie Cross-Compilation-Entwicklung. Was diese Begriffe bedeuten und ob es innerhalb dieser Abgrenzung noch weitere Unterscheidungen gibt, wird im Folgenden näher beleuchtet.

2.1 NATIVE ENTWICKLUNG

Die Native Entwicklung fing 2008 mit dem iPhone von Apple an. Nach einer anfänglich auf dem Web basierenden Lösung, war dies der Ansatz, den Apple den Entwicklern als favorisierten Weg zur Erstellung einer App vorgab. Der webbasierte Ansatz blieb. Apple selbst richtete jedoch das Augenmerk auf die eigene Sprache. Google folgte ein Jahr später mit seinen Android Smartphones und einer eignen Plattform. Beide Plattformen haben gemein, dass es einer speziellen Programmiersprache bedarf, um ein Programm zu schreiben. Für Applikationen im Apple-Kontext ist zusätzlich ein Gerät mit mac-OS z.B. ein Mac-Book erforderlich, um die App zu bauen.

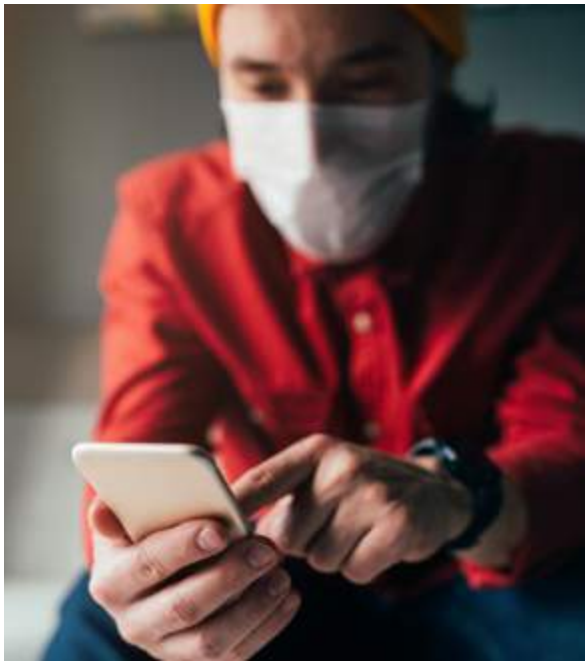
Programmiersprache

Die benötigten Programmiersprachen für iOS sind Objective-C und seit 2014 die Apple-Eigenentwicklung Swift. Eine Koexistenz beider Sprachen in einem Projekt ist möglich, um die maximale Kompatibilität zu älteren Versionen einer App zu gewährleisten.

Für die Android Plattform kommen die Sprachen Java und – seit der Entwicklerkonferenz Google I/O 2017 – Kotlin zum Einsatz. Seit 2019 gilt Kotlin als die bevorzugte Sprache zur Entwicklung von Android Apps. Für beide Plattformen existiert somit eine präferierte Sprache. Apples Swift ist überwiegend für iOS/macOS Apps einsatzfähig, während sich mit Kotlin ebenfalls Native iOS Apps erstellen lassen (vgl. Kotlin 2021).

Kosten

Für professionelle Projekte werden in der Regel zwei getrennte Teams beschäftigt. Ein Team für die Entwicklung der Android App, ein anderes für iOS. Beide Plattformen weisen eine so hohe Komplexität auf, dass das benötigte Know-how einen Grad erreicht hat, der es Entwicklern unmöglich macht, ein Spezialist für beide Plattformen zu sein. Der Markt an verfügbaren Experten ist klein und geeignete Entwickler zu finden ist schwierig. Durch die Trennung der Teams in Android und iOS steigen zudem tendenziell die Entwicklungskosten.



Geräteressourcen

Die Ressourcen des Gerätes werden durch eine Native Entwicklung bestens genutzt, was sich in der CPU- und GPU-Last sowie der Arbeitsspeicherauslastung zeigt (vgl. InVerita 2020). Das stellt im Grunde keine Überraschung dar, da das Betriebssystem auf die jeweilige Programmiersprache abgestimmt ist und allein dadurch an Effizienz gewinnt.

Native API

Der größte Pluspunkt für die Native Entwicklung ist der direkte Zugriff auf alle Funktionen des Gerätes, auf dem die Anwendung läuft. Dies hat besondere Relevanz, wenn die App hardwarenahe Funktionen aufweist, wie die Kommunikation mit anderen Geräten via Bluetooth und dabei Kommandos an Chipsätze bitweise übermittelt.

Aktualität

Ein weiterer Vorteil der nativen Entwicklung von Apps stellt die Aktualität der zur Verfügung stehenden SDKs dar. SDK ist die Abkürzung für Software Development Kit und stellt Tools und Hilfsmittel zur Programmierung zur Verfügung. Google wie auch Apple stellen vor dem jeweiligen Release Beta-Versionen der SDKs für ihre Betriebssysteme bereit, mit denen die Entwickler sich vorab einen Überblick über die neusten Funktionen verschaffen und ihre Apps auf Kompatibilität prüfen. Nach dem Release des SDK muss dieses nur noch heruntergeladen werden, um mit der Entwicklung zu beginnen.

UI und UX

UI ist das User Interface, die Benutzeroberfläche, über die der Anwender mit dem Programm interagiert. Der Begriff UX steht für User Experience und bezeichnet das Benutzer-Erlebnis während der Nutzung des Programms.
Bei der nativen Programmierung wird sichergestellt, dass sich der Look & Feel der App zu 100% nach der entsprechenden Plattform richtet. Somit finden sich Benutzer schnell mit allen Interaktionsmöglichkeiten aufgrund des Wiedererkennungswertes zurecht.

2.2 CROSS-PLATTFORM-ENTWICKLUNG

Der Grundstein für die Cross-Plattform-Entwicklung wurde 2008 in San Francisco in den Büroräumen des Unternehmens Apache von einer kanadischen Firma namens Nitobi gelegt (vgl. Ionic 2021). Hier untersuchten Web-Entwickler die Möglichkeit, Native WebView als Container für Web-Anwendungen auf Smartphones zu nutzen. Der Versuch gelang und in der Folgezeit bauten die Entwickler von Nitobi ein Framework, das das native Grundgerüst für Web-Anwendungen auf Smartphones bereitstellt. Was sich aus dieser Idee Ende des Jahres 2008 bis heute entwickelt hat, wird nachfolgend näher beschrieben.

2.2.1 HYBRID

Zunächst zur Begriffsklärung: Hybride Apps sind eine Mischung aus nativen Bestandteilen und einer Web-Anwendung.

Wie in den einleitenden Worten dieses Abschnitts beschrieben, nutzen diese Apps einen nativen Container in Form einer WebView. Hinter dem Begriff WebView verbirgt sich ein Browserfenster, das standardmäßig ohne die üblichen Steuerungselemente angezeigt wird. Es handelt sich also um eine native Komponente, eingebunden in eine App, die Web-Inhalte darstellt.

Seit dem Jahr 2008 haben sich, neben dem Framework PhoneGap von Nitobi (später von Adobe gekauft und in Cordova umbenannt), noch weitere Alternativen gebildet. So schöpfen Web-Entwickler heute aus einem reichen Portfolio an Frameworks.

Neben Cordova ist React Native zu erwähnen. React Native ist aus Facebooks React entstanden, das zum Erstellen hochdynamischer Webseiten entwickelt wurde. Die Idee hinter React Native ist identisch zu Cordova. Die eigentliche Umsetzung unterscheidet sich in Teilen. Identisch ist jedoch, dass die App mit Web-Technologien geschrieben und dann mit weiteren Hilfsmitteln auf mobile Geräte gebracht wird.

Hybride Apps können ebenfalls für Desktop Betriebssysteme erstellt werden, allerdings ist dafür ein anderes Framework nötig, das den entsprechenden Rahmen zur Verfügung stellt. So fällt die Arbeit doppelt an: Für die mobile App und die Desktop-Variante.

Programmiersprache

Die Entwicklung im Web-Kontext erfordert Kenntnisse in der Programmiersprache JavaScript bzw. inzwischen TypeScript. Zusätzlich kommen HTML und CSS/SCSS zum Einsatz. Der große Vorteil dieses Ansatzes: Viele Entwickler sind mit den Sprachkonzepten vertraut und müssen sich nur noch die Besonderheiten des jeweils eingesetzten Frameworks aneignen.

Kosten

Die Kosten sind tendenziell niedriger als bei der Nativen Entwicklung, da nur ein Team mit Experten beschäftigt wird. Bei der Rekrutierung von Fachkräften steht zudem ein sehr großer Pool von Experten aus dem Bereich der Web-Entwicklung zur Verfügung.

Geräteressourcen

Was der Nutzer der App auf dem Bildschirm sieht, muss vorerst aus JavaScript übersetzt werden. Der Verbrauch von Ressourcen ist bei der hybriden App daher höher (vgl. InVerita 2020). Das bedeutet, dass im Schnitt mehr Arbeitsspeicher sowie eine höhere CPU-Leistung benötigt und mehr Akku-Kapazität verbraucht wird. Dies kann sich beispielsweise in einer geringeren Laufzeit des Gerätes, bis zum nächsten Ladevorgang, bemerkbar machen.

Native API

Um die Funktionalitäten des Smartphones nutzen zu können, muss aus dem Container WebView „ausgebrochen“ werden. Hierfür werden Plugins benutzt, welche die einzige Verbindung zwischen Web-Inhalt und nativen Funktionen des Gerätes darstellen. Um diese Brücke zu schlagen, ist allerdings nativer Code erforderlich. Dies kann ein negativer Punkt sein, wenn etwas sehr Spezielles für die Applikation benötigt wird, es bisher jedoch noch kein Plugin gibt. Für alle gängigen Standardfunktionen, wie Kamerazugriff oder Push Nachrichten, sind jedoch fertige Plugins erhältlich, die eingebunden werden können.

Aktualität

Es bedarf etwas mehr Zeit, bis mit dem Framework der Wahl (Cordova etc.), auf die neuste Version von Android bzw. iOS gewechselt werden kann. Die Kompatibilität der nötigen Bestandteile des Frameworks mit der neuen Version des jeweiligen Betriebssystems muss mit gegebenenfalls vorgenommenen Anpassungen gewährleistet werden. Bereits durch den Testaufwand verzögert sich der Zugang. Sollten neue Funktionalitäten durch das Betriebssystem angeboten werden, sind meist neue Plugins erforderlich. Generell lässt sich also sagen, dass hybride Apps eine gewisse Übergangszeit brauchen, ehe sie auf den aktuellen Stand gehoben werden können.
Damit die Nutzer eine neue Version der App erhalten, wird eine aktualisierte Version in den Stores hochgeladen und ist dort für den Nutzer zum Download verfügbar.

UI und UX

React Native bedient sich an nativen Elementen, wie einem Button oder eines Eingabefeldes und stellt diese dar. Dadurch entsteht ein plattformspezifisches Aussehen. Cordova versucht gar nicht das Aussehen der Plattform, auf der die App läuft, zu imitieren. Dies muss auf Wunsch über Styling erfolgen. Ein zeitlicher Nachteil kann entstehen, wenn versucht wird die App an

das Aussehen der Plattform anzupassen. Wird ein eigenes Designkonzept verfolgt, das vom Design der Plattform abweicht, ist dieser Punkt zu vernachlässigen.

2.2.2 PWA

Wie eingangs erwähnt, handelt es sich bei einer PWA (Progressive Web Application) um eine Webseite, die durch Offline-Funktionalität das Verhalten von Apps simulieren kann, da sie nicht permanent eine Internetverbindung braucht, um Inhalte darzustellen.

Es handelt sich hier also um reine Webinhalte ohne nativen Container, die direkt über den System-Browser aufgerufen werden. Die PWA kann auf Wunsch auch in den nativen Stores veröffentlicht werden. Dafür muss sie allerdings in eine Native App umgewandelt werden. Dies kann zum Beispiel mit Hilfe der Trusted Web Activity Technologie geschehen. Hier kommt dann ein ähnliches Prinzip wie bei den Hybride Apps zum Einsatz. Kurz gesagt bestätigt die Trusted Web Activity der Plattform, dass die zu ladenden Webinhalte vom gleichen Entwickler stammen, der auch dieses native Grundgerüst erstellt hat.

Ein negativer Punkt der PWA ist, dass man mit ihr zwar viel von einer App simulieren kann, aber je nach Plattform durch diese auch reglementiert wird. Zum Beispiel können unter iOS noch keine Web Push Notifications verschickt werden. Allerdings wird daran gearbeitet diese und weitere Funktionen, wie zum Beispiel Bluetooth-Konnektivität (vgl. Web-Bluetooth 2021), ermöglichen zu können.

Programmiersprache

Wenig überraschend ist der Technologie-Stack, mit dem hier gearbeitet wird, identisch zu dem der Hybride Apps. Somit sind HTML, CSS/SCSS, JavaScript/TypeScript das Mittel der Wahl. Meist wird zusätzlich ein Framework wie zum Beispiel Angular oder React verwendet, um die PWA zu erstellen.

Kosten

Das sich die Entwicklung auf den Browser beschränkt, sind die Kosten gleich mit denen der Hybride Apps, eventuell sogar noch etwas niedriger, da hier nicht auf spezielles Aussehen für die einzelnen Plattformen geachtet werden muss.

Geräteressourcen

Es werden die ganz normalen Ressourcen zum Betrieb des Browsers auf der Plattform benötigt.

Native API

In einer PWA kann nicht uneingeschränkt auf alle nativen Funktionen des Gerätes zurückgegriffen werden. Das kann Fingerabdruckerkennung, NFC oder Bluetooth sein, um ein paar Beispiele zu nennen. Gerade der Fingerabdruck ist ein heute weit verbreitetes Feature, das den Zugang zu einer App mit Login für den Nutzer erheblich vereinfacht. Dieses Fehlen bestimmter nativer Funktionen ist das größte Manko der PWA.

Aktualität

Da die PWA auf keine Plattform angewiesen ist, abgesehen vom Browser, auf dem sie läuft, ist die App von Natur aus immer auf dem neusten Stand. Gleiches gilt auch für die Nutzer der App. Es muss kein Update heruntergeladen werden, noch müssen die Entwickler eine neue Version

in einen Store laden und den Review Prozess vor der Veröffentlichung abwarten. Somit kommen Endnutzer mit dieser Variante am schnellsten und komfortabelsten an die aktuelle Version der Anwendung.

UI und UX

Die grafische Oberfläche der Anwendung ist nicht an die Plattform gebunden und braucht somit nicht den Plattformvorgaben zu folgen. Oft kommt das sogenannte Material Design von Google zum Einsatz, was zumindest für Android Nutzer einen Wiedererkennungswert bietet.

2.3 CROSS-COMPILATION

Als letztes soll die Erstellung von Apps für mehrere Plattformen mittels Cross-Compilation (vgl. Gupta, Yadav, Chakraborty 11/2016) angeschaut werden.

Dieses Vorgehen unterscheidet sich vor allem durch den mittels Kompilierung entstehenden Code. Während bei den hybriden Apps HTML, CSS und JavaScript ausgeliefert wird, entsteht bei der Cross-Compilation direkt von der Plattform lesbarer Code. Das Prinzip dahinter ist so einfach wie auch kompliziert. Man schreibt seine App in einer Sprache wie im Fall von Microsoft Xamarin in C# und dieser Code wird später in den benötigten Plattform-Code übersetzt. Der Vorteil liegt hier vor allem darin, dass es sich am Ende des Prozesses wieder um eine Native App handelt. Xamarin (vgl. Microsoft 2021) ist ein, in dieser Kategorie, schon lange bestehendes Tool, das 2011 veröffentlich wurde und seit 2016 offiziell zu Microsoft gehört.

Flutter (vgl. Google Flutter 2021) ist ein vergleichsweise junges Tool in diesem Bereich, das im Dezember 2018 in Version 1.0 veröffentlicht wurde und von Google entwickelt wird. Flutter ist ein UI Framework für die Programmiersprache Dart (vgl. Google Dart 2021), die ebenfalls aus dem Hause Google stammt. Dart ist für die Cross-Compilation prädestiniert, da diese Sprache in verschiedene andere Sprachen transpiliert werden kann, zum Beispiel in JavaScript, was einen Einsatz im Web problemlos möglich macht. Anders als bei anderem maschinell erzeugten Code ist die Performance von JavaScript Code aus Dart heraus sogar besser als handoptimierter Code in der V8 Engine. Google selbst spricht bei dem erzeugten Code für die verschiedenen Plattformen von nativem Maschinencode, der am Beispiel von Android C oder C++ Code ist.

Eine Besonderheit der Cross-Compilation ist, dass nicht nur mobile Apps erstellt werden können, sondern dass man eine solche Applikation auch für die Desktops von Windows, macOS und Linux bereitstellen kann. Allerdings kann bisher keines dieser Tools alles. Das Erstellen einer PWA bietet Xamarin gar nicht erst

an, während Flutter inzwischen eine stabile Version für die Erstellung einer PWA zur Verfügung stellt. Dafür ist Flutter noch nicht so weit bei der Erstellung von Desktop-Anwendungen, wie es beispielsweise Xamarin schon kann. Aber auch hier ist das Core-Team von Flutter bemüht, um dieses Feature bis zur produktionsreife zu bringen.

Programmiersprache

Hier kann keine pauschale Aussage gemacht werden, da die verwendete Sprache vom jeweiligen Tool abhängt. Xamarin hat mit C# eine schon sehr weit verbreitete Sprache, anders als Flutter mit Dart. Auch wenn Dart eine gute Lernbarkeit attestiert wird, bleibt es eine Sprache, die erst noch in die breite Masse dringen muss.

Kosten

Die Tools Xamarin und Flutter sind kostenlos. Somit entstehen hier keine zusätzlichen Kosten. Auch hier kommt der Vorteil zum Tragen, dass man generell nur ein Team zur Entwicklung einer App braucht. Entwickler, die mit C# Expertise besitzen, sind am Markt ausreichend vorhanden, während sich eine entsprechende Entwickler-Community mit Dart-Kenntnissen im Aufbau befindet.

Geräteressourcen

Auch wenn der Code des eingesetzten Tools in plattformspezifischen Code übersetzt wird und eine Native App erstellt, erreichen diese Applikationen nicht den Effizienzgrad einer Native App. Dies bezieht sich vor allem auf die Arbeitsspeicher- oder CPU-Auslastung. Plattformen mittels Cross-Compilation sind in den meisten Fällen jedoch ressourcenschonender als eine hybride App (vgl. InVerita 2020).

Native API

Generell kann jede native Funktion des Gerätes angesprochen werden. Kommt allerdings eine neue Funktion der Plattform hinzu, ist es erforderlich, im Cross-Compilation-Tool die Übersetzungsfunktion zu schreiben. Dies ist gleichwertig zu dem hybriden Entwicklungsansatz.

Aktualität

Wie bei den hybriden Apps, ist es erforderlich, dass die Betreiber des Cross Compliation Tool auf neue Versionen der Plattformen reagieren und diese zur Nutzung bereitstellen. Zwischen dem Release einer neuen Plattform-Version und der Bereitstellung zur Cross Compliation vergeht jedoch Zeit.

Damit Nutzer die neue Version der App erhalten, muss die aktualisierte Version in dem Store zum Download bereitgestellt werden.

UI

Eine pauschale Aussage ist hier nicht zu treffen. Xamarin erzeugt automatisch ein Layout, das die jeweilige Plattform repräsentiert, wohingegen Flutter standardmäßig auf Material Design ausgelegt ist. Es muss explizit angegeben werden, wenn das Komponenten-Design von iOS gewünscht ist und es sind gegebenenfalls andere Komponenten zu nutzen.

3 FALLBEISPIELE FÜR EINE APP

Nach der Betrachtung der unterschiedlichen Herangehensweisen zum Erstellen einer App, widmet sich der folgende Abschnitt Fallbeispielen. Zum jeweiligen Beispiel wird der Funktionsumfang grob beschrieben und geschaut, welche der Entwicklungsmöglichkeiten passend für das Beispiel wäre.

3.1 HARDWARENAHE FUNKTIONALITÄT

Fallbeispiel Nr. 1: Mit Hilfe der App werden über eine Bluetoothverbindung die Einstellungen eines anderen Gerätes ausgelesen und neu gesetzt. Da es sich um ein einfaches Gerät handelt, wird hardwarenah mit Bits gearbeitet. Es bedarf den Zugang zur Bluetooth-Funktion des Gerätes und zu den Bit-Operationen, um den Micro-Controller mit den Befehlen zu füttern.

Ein Beispiel zu einer solchen App ist die Programmierung und Wartung von Radar-Geschwindigkeitsanzeigen im Außendienst.

Vor allem die Native Entwicklung spielt in diesem Fall ihre Stärken aus. Keiner der drei beschriebenen Ansätze ist hardwarenäher als die Native Entwicklung. Es bedarf einer individuellen Ansprache der Bluetooth-Schnittstellen, da es sich nicht um gängige Standards handelt, sondern um eine Eigenentwicklung.

Somit besteht aus dem hybriden Sektor zwar die Möglichkeit auf Bluetooth zuzugreifen, für die Bitoperationen muss jedoch ein eigenes Plugin geschrieben werden. Hier bedarf es spätestens nativer Expertise für Android und/oder iOS.

Die Cross-Compilation kämpft mit dem gleichen Problem und die spezielle Anforderung von Bit-operationen kann nur ein selbst geschriebenes Plugin lösen. Damit wird auch hier native Erfahrung erforderlich.

Die PWA scheidet für ein solches Vorhaben gänzlich aus, da keine Möglichkeit des Zugriffes auf Bluetooth besteht.

Als Fazit dieses Fallbeispiels bleibt festzuhalten, dass es sich hier um ein Paradebeispiel für die Native App-Entwicklung handelt. Hybride- und Cross-Compilation-Ansätze können verfolgt werden, erfordern aber mehr Aufwand, da die Schnittstellen zur nativen Ebene zusätzlich für die jeweilige Plattform geschrieben werden müssen.

3.2 ETABLIERUNG EINES PORTALS

In Fallbeispiel Nr. 2 wird ein Kundenportal erstellt, in dem die Nutzer ihre Stammdaten einsehen und ändern, Verträge inklusive Details abfragen und den Schriftverkehr nachverfolgen können. Zusätzlich stehen Funktionen zur Verfügung, die das Hochladen von Bildern oder Dateien ermöglichen, wie zum Beispiel das Einreichen von Rechnungen. Änderungen an dem Status der Rechnungen werden dem Nutzer per Push Nachricht mitgeteilt.

Das Portal steht möglichst vielen Menschen zur Verfügung und bedient die unterschiedlichen Nutzer-Vorlieben, wie mobile App und Webseite.

Da die Anforderungen eine Webseite beinhalten, scheidet die native Option aus. Der mobile Teil ist zwar erfüllbar, doch für die Webinhalte müsste mit der Entwicklung von vorne begonnen werden.

Die hybride Entwicklung spielt ihre Stärke hingegen voll aus. Da die hybride App-Entwicklung auf Webtechnologien basiert, entsteht die Webseite ganz nebenbei, während der Erstellung der App. Bei der Entwicklung ist darauf zu achten, dass die Web-Version vollumfänglich funktioniert. Doch der Aufwand hält sich – im Verhältnis zu einer Neu-Entwicklung – in Grenzen.

Die PWA bietet viele der benötigten Funktionen die das Fallbeispiel verlangt. Mit anderen Funktionen, wie den Push-Nachrichten unter iOS, kann es zu Problemen kommen. Dies ist weniger der PWA, als den Restriktionen des Betriebssystems von Apple geschuldet. Die PWA eignet sich daher sehr gut für das Vorhaben in Fallbeispiel Nr. 2.

Cross-Compilation hat mit der mobilen Variante, wie die Native Entwicklung, keinerlei Probleme. Mit Flutter lässt sich inzwischen eine PWA erstellen. Somit ist mit dem Framework von Flutter eine Umsetzung möglich, bleibt aber toolabhängig, da Xamarin das Erstellen einer PWA nicht unterstützt.

Als Fazit für Fallbeispiel Nr. 2 bleibt festzuhalten, dass es sich um das denkbar beste Szenario für eine hybride App-Entwicklung bzw. PWA handelt. Egal welche Bildschirmgröße oder Umgebung benötigt wird, beide Ansätze sind – unter Einhaltung des Mobile-First-Prinzips – bedenkenlos einsetzbar.

Eine Native Entwicklung ergibt nur Sinn, wenn die zusätzlichen Kosten, für den bestmöglichen Look & Feel, in Kauf genommen werden.

Cross-Compilation bietet, genau wie die Native Entwicklung, ein sehr gutes Ergebnis für mobile Endgeräte und eine produktiv einsetzbare Webanwendung mit nur einem Tool. So wird, genau wie bei der hybriden Entwicklung und PWA, über alle Geräte hinweg ein Ergebnis ausgeliefert.

3.3 NEWS APP MIT EINHEITLICHER OBERFLÄCHE

Fallbeispiel Nr. 3 beschreibt eine Applikation, die aktuelle News anzeigt, die über verschiedene soziale Kanäle geteilt werden können. Push-Nachrichten informieren den Nutzer auf Wunsch über Neuigkeiten innerhalb einer Rubrik. Für eine hohe Wiedererkennbarkeit ist das Aussehen der Oberfläche auf allen Plattformen identisch. Die native Geschwindigkeit muss sichergestellt werden, da keine Web-Version existiert, dafür jedoch eine Native Desktop-App.

Der native Ansatz kann verfolgt werden, bringt jedoch den Nachteil mit sich, dass der Code zweimal geschrieben werden muss. Dieser einzige Nachteil erhöht die Kosten in der Entwicklung der Anwendung.

Die Anforderung an die native Geschwindigkeit kann zu einem Problem bei der hybriden Entwicklung werden. Davon abgesehen bietet dieser Ansatz den Vorteil der Single Codebase und einer (meist) plattformunabhängig einheitlich dargestellten Oberfläche. Somit bietet der hybride Ansatz eine Möglichkeit der Umsetzung.

Die PWA bringt generell die gleichen Vorteile wie die hybride Entwicklung mit sich, allerdings kommt es plattformabhängig zu Problemen mit einzelnen Funktionen. In diesem Fallbeispiel würde es sich das Problem bei den Push-Nachricht unter iOS bemerkbar machen. Somit wäre die PWA nur bedingt das Mittel der Wahl.

Bei der News App glänzt vor allem die Cross-Compilation, da sie den Vorteil der nativen Geschwindigkeit bei nur einer Codebase mitbringt. So greifen die Vorteile der nativen Welt sowie die Kostenersparnise einer Cross-Plattform-Lösung.

Ein Fallbeispiel zu generieren, in dem die Cross-Compilation als klarer Sieger hervorgeht, ist nicht leicht möglich. Cross-Compilation und hybride Entwicklung liegen nah beieinander, was die Funktionen und Möglichkeiten der App angeht. Wenn das Augenmerk auf einer App ohne weitere Medien (Browser oder Desktop) liegt, entsteht ein Vorteil für Cross-Compilation.

4 FAZIT

Wie aufgezeigt wurde, gibt es nicht den einen Weg, um eine App bereitzustellen. Vielmehr muss der gesamte Kontext, in dem die App stehen soll, klar werden, um individuell aus den verschiedenen Möglichkeiten die beste Option wählen zu können.

Zusätzlich muss berücksichtigt werden, welche Funktionalitäten dem Nutzer bereitstehen sollen, da diese je nach Ansatz nur bedingt oder gar nicht ermöglicht werden können.

Das Feld der App-Entwicklung ist im Verhältnis zu anderen Entwicklungen, wie Java-, C-/C++/C#- oder Web-Anwendungen, vergleichsweise jung. Daher spiegelt dieses Whitepaper nur den aktuellen Stand wider.

Wie zu Beginn des Artikels erwähnt, konzentrieren sich mobile Apps auf das Erledigen einzelner oder sehr weniger Aufgaben. Wenn also der Wunsch besteht, komplexe Business-Prozesse abzubilden, wie Produktionsketten oder logistische Problemstellungen, ist eine Desktop-Anwendung sinnvoll. Sollen jedoch Teilbereiche von Prozessen schnell zugänglich sein, um einzelne Aufgaben separat zu erledigen, kann eine App ihre Stärken ausspielen.

Die folgende Tabelle liefert einen Überblick und Vergleich über die betrachteten Aspekte der Entwicklungsphilosophien:

Entwicklungsparadigma	NATIVE	HYBRID	PWA	CROSS-COMPILATION
Programmiersprache	Java/Kotlin Objective-C/ Swift	JavaScript/ TypeScript	JavaScript/ TypeScript	Toolabhängig z.B. C# oder Dart
Entwicklerverfügbarkeit	Mittel	Hoch	Hoch	Toolabhängig z.B. C#(Mittel) oder Dart(Gering)
Kosten	Hoch	Mittel	Mittel	Mittel
Geräteressourcen	Effizient	Erhöht	Effizient	Gering erhöht
Native API	Problemlos	Geringer bis mittlerer Aufwand	Geringer Aufwand oder nicht verfügbar	Geringer bis mittlerer Aufwand
Aktualität bei Update der zugrundeliegenden Plattform	Direkt mit Veröffentlichung	Zeitlich versetzt	Nicht erforderlich	Zeitlich versetzt
UI / UX	Plattformspezifisch	Plattformunabhängig mit Aufwand plattformspezifisch		Eher plattformspezifisch

Der Punkt Kosten bezieht sich ausschließlich auf die Personalkosten bei der Entwicklung. Die Fixkosten für das Betreiben eines Entwickerprofils für die jeweilige Plattform Android oder iOS sind für alle Entwicklungsstrategien identisch und fallen daher nicht ins Gewicht.

5 QUELLEN

[1] Web-Bluetooth (2021):
Implementation Status for Web-Bluetooth, contribution on github.com.
https://github.com/
letzter Zugriff: 08.06.2021

[2] Ionic (2021):
What is Apache Cordova.
https://ionic.io/
letzter Zugriff: 08.06.2021

[3] Kotlin (2021):
Write the business logic for your iOS and Android Apps just once, in pure Kotlin
https://kotlinlang.org/
letzter Zugriff: 08.06.2021

[4] DZW (2021):
ARD/ZDF-Onlinestudie 2018: Mehr als 90 Prozent der Deutschen online
https://www.dzw.de/
letzter Zugriff: 19.05.2021

[5] ARD-ZDF-Onlinestudie (2020):
Zahl der In-ternetnutzer wächst um 3,5 Millionen
https://www.ard-zdf-onlinestudie.de/
letzter Zugriff: 19.05.2021

[6] digital.ai (2021):
Enterprise app store
https://digital.ai/
letzter Zugriff: 17.05.2021

[7] InVerita (2020):
Flutter vs React Native vs Native: Deep Performance Comparison
https://medium.com/
letzter Zugriff: 17.05.2021

[8] Gupta, Yadav, Chakraborty (11/2016):
Complier bootstrapping and cross-compilation
https://www.currentscience.ac.in/
letzter Zugriff: 08.06.2021

[9] Microsoft (2021):
Xamarin
https://dotnet.microsoft.com/
letzter Zugriff: 08.06.2021

[10] Google Flutter (2021):
Flutter
https://flutter.dev/
letzter Zugriff: 08.06.2021

[11] Google Dart (2021):
Dart
https://dart.dev/
letzter Zugriff: 08.06.2021