Vergleich von Frameworks zur Entwicklung hybrider Applikationen

Jannik Dürr[1], Dominic Joas[2], Ruben Kalmbach[3]

Abstract: Hybride Applikationen sind ein moderner Ansatz zur Vereinfachung der Softwareentwicklung. Basierend auf einer gemeinsamen Codebasis wird die Anwendung durch native Wrapper zu verschiedenen Betriebssystemen kompatibel gemacht. Aufgrund der großen Anzahl an Frameworks für die Entwicklung hybrider Applikationen werden in diesem Beitrag verschiedene Frameworks gegenübergestellt. Zunächst werden der Aufbau und die Eigenschaften hybrider Anwendungen beschrieben, sowie Vergleichskriterien und relevante Merkmale hybrider Frameworks zusammengefasst. Dann werden drei Anwendungsgebiete definiert: rechenintensiv, formularlastig und sensorlastig. Die Vergleichskriterien werden in einem paarweisen Vergleich für jedes Anwendungsgebiet separat gewichtet. Anschließend werden die einzelnen Frameworks mithilfe von Literatur und eigenen Untersuchungen betrachtet und anhand der Kriterien bewertet. Durch die Berechnung der Punktzahlen für jedes Framework können Empfehlungen für jedes Anwendungsgebiet gegeben werden.

Keywords: Hybride Applikation; Hybride Frameworks; Mobile Applikation; Native Applikation

1 Einleitung

In diesem Kapitel wird eine Einleitung in den Beitrag in Form einer Motivation gegeben. Anschließend werden die verwendeten Methoden erläutert, gefolgt von einem Überblick über das Vorgehen und die Ziele.

1.1 Motivation

Applikationen auf mobilen Geräten nehmen im Alltag vieler Menschen eine große Rolle ein. In den App-Stores von Google und Apple stehen mehrere Millionen Apps zur Verfügung, die jeden Bereich des täglichen Lebens abdecken. Bei vielen dieser Apps handelt es sich um native Applikationen, seit einigen Jahren gewinnen jedoch hybride Applikationen ebenfalls immer mehr an Popularität und kommen z. B. bei Social-Media-Apps wie Twitter erfolgreich zum Einsatz.

[1] DHBW Stuttgart Campus Horb, Studiengang Informatik, Florianstraße 15, 72160 Horb am Neckar, Deutschland, i20007@hb.dhbw-stuttgart.de
[2] DHBW Stuttgart Campus Horb, Studiengang Informatik, Florianstraße 15, 72160 Horb am Neckar, Deutschland, i20015@hb.dhbw-stuttgart.de
[3] DHBW Stuttgart Campus Horb, Studiengang Informatik, Florianstraße 15, 72160 Horb am Neckar, Deutschland, i20018@hb.dhbw-stuttgart.de

Hybride Applikationen versuchen, die Vorteile von nativen Applikationen und Web-Applikationen zu kombinieren. Native Applikationen sind Anwendungen, die speziell für ein konkretes Betriebssystem entwickelt und optimiert wurden. Web-Applikationen sind hingegen vollständig browserbasiert und unabhängig von konkreten Betriebssystemen, allerdings ist der Zugriff auf die Sensoren des Geräts stark eingeschränkt. Einen guten Kompromiss bilden die bereits angesprochenen hybriden Applikationen, die wie Web-Anwendungen mit Webtechnologien entwickelt werden, allerdings anschließend in native Container, auch Wrapper genannt, verpackt werden. So sind sie kompatibel zu mehreren Betriebssystemen und können dabei auf native Schnittstellen zugreifen. Der Arbeitsaufwand, der bei der Entwicklung von Apps ein zentraler Faktor ist, kann so stark reduziert werden, da die Entwicklung nicht für jedes Betriebssystem separat erfolgen muss.

Für die Entwicklung von hybriden Applikationen stehen eine Vielzahl an Frameworks zur Verfügung. Abbildung 1 stellt diesbezüglich dar, welche hybriden Frameworks weltweit von Entwicklern bevorzugt verwendet werden. Allerdings ist es für Entwickler, die sich nicht intensiv mit der Entwicklung hybrider Applikationen beschäftigt haben, oft schwierig zu entscheiden, welches Framework für bestimmte Anwendungsfälle verwendet werden sollte. Daher werden in dieser Arbeit die wichtigsten hybriden Frameworks genauer betrachtet und anhand definierter Kriterien miteinander verglichen. So können Empfehlungen abgegeben werden, welches Framework abhängig von den Anforderungen eingesetzt werden sollte.

1.2 Methoden

Folgende Methoden werden neben der Literaturarbeit verwendet:

- **Paarweiser Vergleich**
  Durch einen paarweisen Vergleich werden die Vergleichskriterien systematisch paarweise gegenübergestellt und so ermittelt, welches der Kriterien wichtiger ist bzw. ob sie gleich wichtig sind. Aus der numerischen Interpretation der Gegenübergestellungen ergeben sich die Rangfolge und die relative Gewichtung der Vergleichskriterien.
- **Prototyping**
  Durch die Entwicklung von Prototypen mit möglichst identischem Funktionsumfang werden Erfahrungen mit den Frameworks gesammelt und der Entwicklungsaufwand bewertet.
- **Benchmarking**
  Durch die Durchführung eines Software-Benchmarks mit identischen Algorithmen werden Ausführungszeiten und CPU-Auslastung der Prototypen ermittelt.

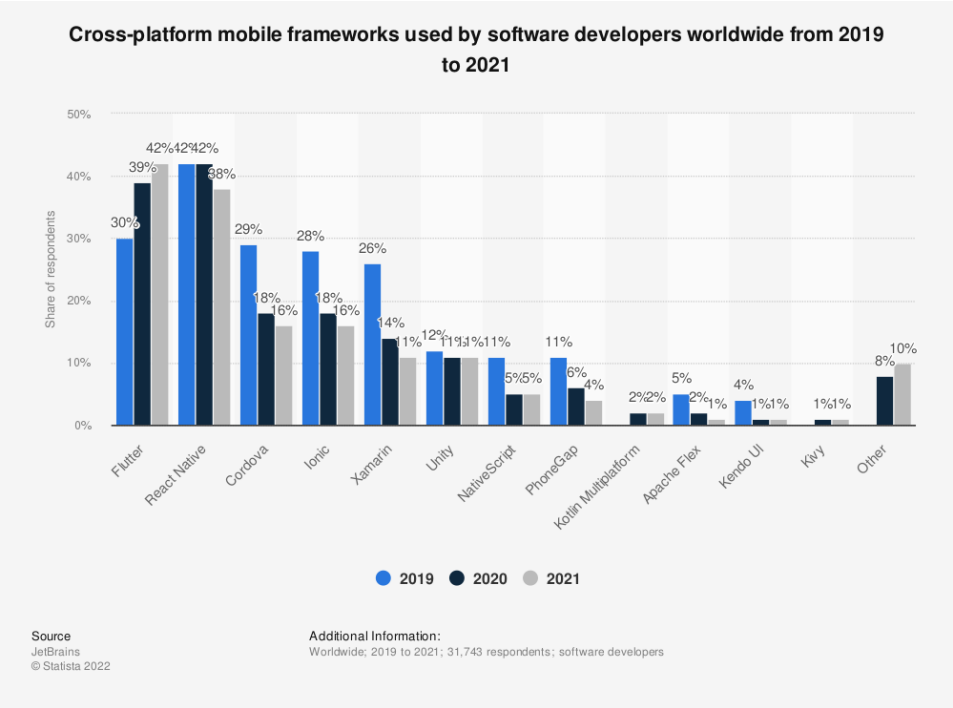

Abb. 1: Umfrage, welche hybriden Frameworks von Entwicklern verwendet werden [Statista 2021]

1.3 Vorgehen und Ziele

Das Ziel des Beitrags ist der Vergleich von Frameworks zur Entwicklung hybrider Applikationen. Zu diesem Zweck werden hybride Applikationen zunächst definiert und gegen mobile und native Applikationen abgegrenzt. Anschließend werden Vergleichskriterien ermittelt und Anwendungsgebiete für hybride Applikationen bestimmt, wobei den Kriterien jeweils eine Gewichtung zugeordnet wird. Dann werden sechs relevante, hybride Frameworks basierend auf Literatur und eigenen Erkenntnissen anhand der ermittelten Kriterien untersucht. Anschließend findet der Vergleich der Frameworks statt, wobei die hybriden Frameworks zunächst paarweise anhand einzelner Kriterien verglichen werden. Mithilfe von Prototypen und Benchmarks wird außerdem die Performance der Frameworks gemessen und ausgewertet. Anschließend werden die Ergebnisse zusammengefasst und es werden Empfehlungen gegeben, welches Framework sich für welches Anwendungsgebiet eignet.

2 Hybride Applikationen

In diesem Kapitel werden hybride Applikationen zunächst definiert und gegen mobile und native Applikationen abgegrenzt. Dabei wird außerdem der Aufbau behandelt. Anschließend werden Eigenschaften sowie Vor- und Nachteile von hybriden Applikationen genauer betrachtet.

2.1 Definition und Aufbau

Um eine Definition für hybride Applikationen zu finden, ist es notwendig, ein einheitliches Verständnis über native Applikationen, Web-Applikationen und Cross-Plattform-Applikationen zu schaffen. Daher werden diese Begriffe im Folgenden erläutert und gegeneinander abgegrenzt.

Native Applikation

Eine native Applikation ist eine App, die für genau ein Betriebssystem von mobilen Endgeräten entwickelt wird und in der Regel auch nur dort ausführbar ist. Für die Entwicklung muss die Programmiersprache des jeweiligen Betriebssystems (z.B. Java/Kotlin für Android) verwendet werden. Da native Applikationen speziell für das Betriebssystem entwickelt werden, fällt es auch leicht, Sensoren zu nutzen, die das jeweilige Endgerät anbietet. Durch die native Ausführung wird die App außerdem so optimiert, dass sie ressourcenschonend und leistungsstark ist.

Web-Applikation

Web-Applikationen werden in der Regel nicht lokal auf einem mobilen Endgerät installiert, sondern werden über eine Cloud oder einen Server bereitgestellt und sind über eine URL abrufbar. Dies bringt den Vorteil mit sich, dass man Web-Apps plattformübergreifend nutzen kann. Die App muss daher nur einmal entwickelt werden und nicht für jede Plattform neu. Durch die Ausführung auf einem Web-Server sind die Berechtigungen und Funktionalitäten der App aber eingeschränkt. Daher können die Sensoren des jeweiligen Endgerätes nicht genutzt werden und man kann nur eingeschränkt auf den Speicher des Geräts zugreifen. Hat ein Smartphone keinen Internet-Zugriff, so kann nicht auf den Web-Server zugegriffen und somit auch die App nicht genutzt werden.

Cross-Plattform Applikation

Eine Cross-Plattform Applikation ist eine Applikation, welche über eine gemeinsame Codebasis entwickelt wurde und auf verschiedene Betriebssysteme übertragen werden kann. Dies kann entweder durch die Entwicklung einer Web-Applikation (vgl. oben) oder die Entwicklung einer hybriden Applikation erreicht werden.

Hybride Applikation

Im Gegensatz zu einer Web-Applikation basiert eine hybride App nicht zwangsweise auf Web-Technologien. Bei der Entwicklung hybrider Apps können stattdessen auch andere Technologien und Programmiersprachen (z. B. Dart oder C#) zum Einsatz kommen. Hybride Applikationen basieren auf einer gemeinsamen Code-Basis, die in nativen Code für verschiedene Betriebssysteme kompiliert wird. So kann im Gegensatz zu Web-Applikationen auch auf native Funktionen der jeweiligen Betriebssysteme wie z. B. Sensoren zugegriffen werden.

> „Generally speaking, hybrid app is programmed in browser-supported language and wrapped as native app. It is called hybrid because it combines the features of both native and web applications.“ [Denko et al. 2021]

Die einzelnen Applikationsarten sind in Abbildung 2 entsprechend ihrer Flexibilität und ihres Funktionsumfangs angeordnet. Es ist erkennbar, dass native Apps den größten Funktionsumfang, aber gleichzeitig auch die geringste Flexibilität bieten. Web-Applikationen hingegen haben eine enorme Flexibilität, unterstützen allerdings nur einen geringen Funktionsumfang. Hybride Applikationen sind in der Mitte beider Applikationsarten angeordnet und bieten einen Mittelweg zur Entwicklung mit sowohl relativ hohem Funktionsumfang als auch relativ hoher Flexibilität.

Aufbau

Eine hybride App besteht in der Regel aus mehreren Teilen. Es gibt einen Kern, in dem sich der Quellcode für die eigentliche Applikation befindet und für Android und iOS jeweils einen Teil, in dem der native Code generiert wird. Der Kern der App wird häufig mit Web-Technologien erstellt. Dabei kann es sich z. B. um HTML, CSS und JavaScript handeln. Es können allerdings auch abweichende Sprachen (z. B. Dart oder C#) zum Einsatz kommen.

Frameworks für hybride Apps liefern eine JavaScript-Bridge mit, welche den eigentlichen Quellcode in nativen Code umwandelt. Dieser wird dann zur Erstellung von Android- bzw. iOS-Apps genutzt. Wurde der native Code erstellt, kann man diesen über die jeweilige Entwicklungsumgebung öffnen und wie den Quellcode einer nativen App verwenden.

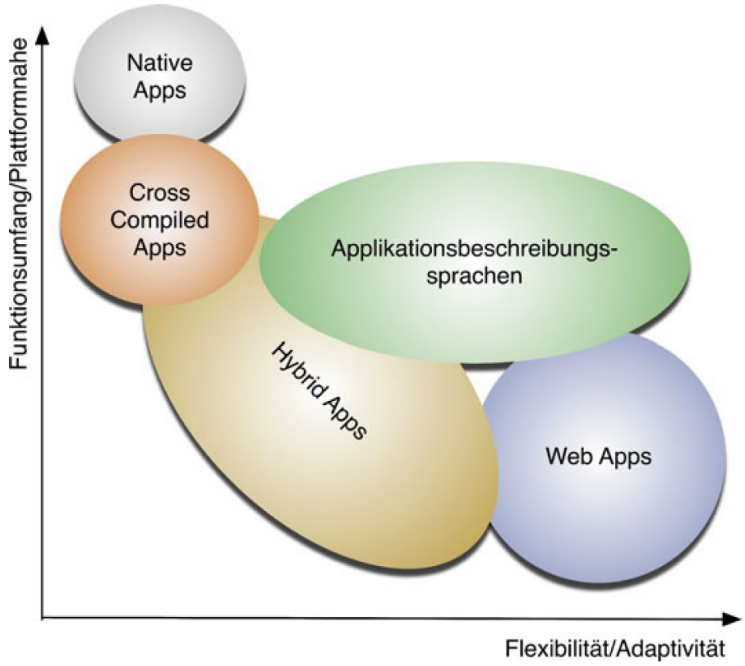

Abb. 2: Übersicht über Arten von Applikationen nach Flexibilität und Funktionsumfang [Willnecker et al. 2012, S.405]

Da eine hybride App oft unter dem Slogan „write once, run everywhere“ beworben wird, ist es notwendig, die Applikation unabhängig vom konkreten Betriebssystem auf gleiche Art und Weise anzuzeigen. Um dies zu erreichen, wird häufig in der App eine Webview angezeigt, welche den JavaScript-Code wiedergibt.

> „Hybrid app looks and performs in native way but works like web app, so hybrid app possesses the great user experience of native app and cross-platform characteristic of web app at the same time.“
> [Que et al. 2016]

2.2 Vor- und Nachteile

Hybride Applikationen bringen einige Vor- und Nachteile mit sich, die in diesem Abschnitt genauer betrachtet werden. Abschließend wird ein Fazit getroffen und die wichtigsten Aussagen zusammengefasst.

Vorteile

Der zentrale Vorteil hybrider Applikationen besteht in der Kompatibilität zu verschiedenen Betriebssystemen. Gemäß dem „Write once, run anywhere“-Paradigma ist die gemeinsame Codebasis mit den meistverwendeten, mobilen Betriebssystemen kompatibel [Denko et al. 2021]. Dabei handelt es sich Stand 2022 zu über 99% um die Betriebssysteme Android und iOS [Statista 2022]. Somit sind auch die Entwicklungskosten deutlich geringer als bei nativen Applikationen, da ein Entwicklungsteam für alle Betriebssysteme zeitgleich entwickeln kann [Kleinschrod 2020]. Dies bietet sich besonders für Start-Up-Unternehmen an, die mit wenigen Mitarbeitern schnell eine funktionsfähige Applikation entwickeln wollen. Da hybride Applikationen somit auch in den beiden App Stores von Google und Apple angeboten werden können, kann eine höhere Anzahl potentieller Nutzer erreicht werden als bei nativen Applikationen, die auf eine konkrete Plattform beschränkt sind [Tyshchenko 2020].

Da die Codebasis hybrider Applikationen mit Webtechnologien entwickelt wird, kommen im Entwicklungsprozess vergleichsweise einfach zu erlernende Programmiersprachen wie HTML, CSS und JavaScript zum Einsatz. Native Applikationen werden mit Java oder Kotlin für Android und Swift oder Objective-C für iOS entwickelt, wobei es sich jeweils um deutlich komplexere Programmiersprachen handelt [Tyshchenko 2020]. Der Zugriff auf die Sensoren des Endgeräts (z. B. GPS, Kamera oder Beschleunigungssensor) ist bei hybriden Applikationen durch Verwendung einer Bridge zu den Schnittstellen des Betriebssystems möglich [Que et al. 2016], wobei es sich um einen klaren Vorteil gegenüber Web-Applikationen handelt. Außerdem können hybride Applikationen genau wie native Applikationen Daten in lokalen Datenbanken abspeichern [Denko et al. 2021]. Sie sind somit auch offline verfügbar, was bei Web-Applikationen nicht der Fall ist.

Nachteile

Hybride Applikationen haben grundsätzlich eine geringere Performance als native Applikationen und nutzen die Leistungsfähigkeit des Endgeräts nicht optimal aus [Kleinschrod 2020]. Dies hat die Ursache, dass es durch die notwendige Kommunikation zwischen Applikation und nativen Komponenten zu Geschwindigkeitseinbußen kommt. Zudem ist es bei nativen Applikationen einfacher, die UX (User Experience) besser für jedes Betriebssystem zu optimieren, da direkt mit nativen Widgets gearbeitet werden kann [Cowart 2012]. Bei hybriden Applikationen ist dies nicht der Fall, sodass die Entwicklung einer Benutzeroberfläche, die auf allen Plattformen gut aussieht, anspruchsvoll sein kann. Außerdem sind hybride Applikationen tendenziell anfälliger für Fehler und Ausfälle als native Applikationen [Tyshchenko 2020].

Ein weiterer Aspekt der Entwicklung hybrider Applikationen ist der Testprozess. Dieser ist grundsätzlich aufwändiger als bei nativen Applikationen, da die Applikation für alle

Plattformen getestet werden muss. Dies erschwert auch die Testautomatisierung, da die Skripte einen größeren Funktionsumfang abdecken müssen [Tyshchenko 2020]. Neben den Funktionstests (z. B. Ressourcenzugriff und Verhalten), Oberflächentests (z. B. korrekte Darstellung) und Leistungstests (z. B. Lasttest und Stresstest) gibt es zwei weitere Testarten, die besonders im Testprozess hybrider Applikationen relevant sind. Dabei handelt es sich um Kompatibilitätstests und Verbindungstests [Tyshchenko 2020]. Kompatibilitätstests sind sinnvoll, um die korrekte Ausführung der hybriden Applikation auf mehreren Betriebssystemen sicherzustellen. Durch Verbindungstests wird das korrekte Verhalten der Applikationen online und offline sichergestellt.

Fazit

Hybride Applikationen sind ein guter Kompromiss zwischen nativen Applikationen und Web-Applikationen. Sie kombinieren die wichtigsten Vorteile der beiden Applikationsarten und versuchen, daraus entstehende Nachteile bestmöglich zu reduzieren. Zusammengefasst kann man sagen, dass hybride Applikationen die schnelle, einfache und kostengünstige Entwicklung einer Applikation für mehrere Betriebssysteme ermöglichen und dafür eine leicht reduzierte Performance, eine etwas erschwerte Optimierung der Benutzeroberfläche und einen etwas aufwändigeren Testprozess in Kauf nehmen.

2.3 Potential

Trotz der einzugehenden Kompromisse gegenüber nativen Applikationen sind beliebte und vielverwendete hybride Applikationen bereits Bestandteil unseres Alltags und im Fokus von großen Unternehmen. Bekannte Beispiele für hybride Applikationen sind Twitter, Instagram, Evernote, Uber und Remote POS.

Diese Apps demonstrieren, dass die Leistungsfähigkeit von hybriden Applikationen durchaus den derzeitigen Anforderungen des Markts gewachsen sind und ein Interesse in der Verwendung und Weiterentwicklung von hybriden Frameworks besteht [Dharmwan 2021].

Die Möglichkeit, eine Applikation in den App Stores gängiger Anbieter zu veröffentlichen, ohne dass die Entwicklungskosten proportional zu der Anzahl der gewünschten Zielplattformen wachsen, ist ein Potential, das Firmen bei der Entwicklung ihrer Applikationen nutzen können und möchten.

3 Vergleichskriterien

In diesem Kapitel werden Vergleichskriterien ermittelt, anhand derer die Frameworks untersucht werden. Anschließend werden Anwendungsgebiete bestimmt und die Relevanz der Vergleichskriterien je Anwendungsgebiet gewichtet.

3.1 Bestimmung

Folgende Vergleichskriterien werden bei der Untersuchung der Frameworks berücksichtigt.

Plattformspezifische Funktionen
Die Zugriffsmöglichkeiten des Frameworks auf plattformspezifische Funktionen wird untersucht. Dies umfasst z. B. die Speicherverwaltung, die sich je nach Betriebssystem unterscheidet. Außerdem wird der Zugriff auf verschiedene Sensoren betrachtet. Dazu gehört u. A. die Standortermittlung durch GPS, der Umgebungslichtsensor oder das Magnetometer.

Performance
Die Performance des Frameworks wird u. A. mithilfe von Literatur bewertet. Dabei werden Geschwindigkeitseinbußen bei der Kommunikation zwischen Codebasis und dem nativen Wrapper, der plattformspezifische Funktionen realisiert, berücksichtigt. Zur detaillierten Bestimmung der Leistungswerte für die definierten Anwendungsgebiete sind Prototyping und Benchmarking notwendig. Ein weiterer Aspekt ist u. A. die Dateigröße des Builds für die jeweiligen Zielplattformen.

Benutzeroberfläche
Die Vorgehenweise bei der Erstellung der Benutzeroberfläche wird untersucht. Dabei wird betrachtet, in welchem Umfang grafische Elemente zur Verfügung stehen und durch welche strukturellen Elemente die responsive Anordnung der Elemente umgesetzt ist. Außerdem wird untersucht, welche Technologien (z. B. HTML & CSS oder native Elemente) zur Entwicklung der Benutzeroberfläche eingesetzt werden.

Erste Schritte
Die Einrichtung des Frameworks wird betrachtet. Dabei wird untersucht, wie aufwändig und schwierig die Installation ist und ob weitere Abhängigkeiten zu dritten Programmen bestehen, die ebenfalls installiert werden müssen. Außerdem wird die Komplexität der Einarbeitung in das Framework bewertet, wobei z. B. Einstiegsschwierigkeiten hervorgehoben werden.

Entwicklungsunterstützung
Die Unterstützung bei der Softwarentwicklung mit dem Framework wird bewertet. Dabei wird untersucht, welche IDEs verwendet werden können und ob dabei für die kommerzielle Verwendung kostenpflichtige Lizenzen benötigt werden. Außerdem wird betrachtet, ob es IDE-Erweiterungen für das Framework gibt und ob Templates für die Entwicklung existieren.

Dokumentation
Die Dokumentation des Frameworks wird bewertet. Dabei wird Wert auf den Umfang und die Nutzerfreundlichkeit der Dokumentation gelegt. Außerdem wird die Übersichtlichkeit und die Verfügbarkeit betrachtet.

Ökosystem
Das Ökosystem des Frameworks wird untersucht. Dabei wird die Vielzahl der zur Verfügung stehenden Klassenbibliotheken betrachtet, die den Funktionsumfang des Frameworks erweitern. Außerdem wird die Kompatibilität zu externen Diensten untersucht.

Lebenszyklus & öffentliches Interesse
Der Lebenszyklus des Frameworks wird betrachtet. Es wird bewertet, wie populär und modern das Framework ist und an welchem Punkt des Lebenszyklus es sich befindet. So kann bestimmt werden, wie wahrscheinlich eine zeitnahe Einstellung der Weiterentwicklung des Frameworks ist und ob sich ein Umstieg auf die langfristige Softwareentwicklung mithilfe des Frameworks empfiehlt.

Außerdem werden folgende Aspekte betrachtet, die jedoch nicht Teil des anschließenden Vergleichs sind.

Programmiersprachen Programmiersprachen, in denen die Softwareentwicklung erfolgt

Zielplattformen Zielplattformen, für die Software entwickelt werden kann

Wartbarkeit Architektur-Patterns, die mithilfe des Frameworks umgesetzt werden können

Datenhaltung DBMS, die vom Framework unterstützt werden

Lizenz & Kosten Lizenz, unter der das Framework steht und eventuell anfallende Kosten

3.2 Anwendungsgebiete

Um die Frameworks bestmöglich zu testen, müssen Anwendungsgebiete definiert werden, für die in den jeweiligen Frameworks beispielhaft hybride Applikationen erstellt werden. Diese Anwendungsgebiete werden im Folgenden näher beschrieben.

Formularlastige Anwendung

Eine formularlastige Anwendung ist eine Anwendung, in welche Benutzer*innen Informationen über Formulare eingeben können. Diese Daten sollen beispielhaft in einer lokalen Datenbank abgespeichert und wieder daraus geladen werden. Außerdem wird bei der Entwicklung der Funktionsumfang und der Aufwand zur Erstellung der Formulare betrachtet. Auch die Usability soll ermittelt werden, indem notiert wird, was beim Ausfüllen der jeweiligen Formulare auffällt und was positiv bzw. negativ hervorzuheben ist.

Sensorlastige Anwendung

Damit ein Smartphone mit der Umwelt kommunizieren kann, haben aktuelle Geräte eine große Anzahl an Sensoren eingebaut. Ein aktuelles Smartphone der Mittelklasse hat üblicherweise mindestens folgende Sensoren eingebaut:

Gyroskop Misst die Rotation des Geräts.

Beschleunigungssensor Misst, ob das Gerät in Bewegung ist.

Lichtsensor Misst die Helligkeit der Umgebung.

GPS-Sensor Bestimmt die Position des Geräts.

Kamera Aufnahme von Bildern und Videos.

Fingerabdrucksensor Registriert den Fingerabdruck der Benutzer*in.

Um zu ermitteln, welche Sensoren das jeweilige Framework nutzen kann, soll eine Applikation entwickelt werden, welche diese Sensoren im Einsatz zeigt. Hierbei soll auch berücksichtigt werden, wie einfach es ist, die Messungen der Sensoren in der Applikation zu berücksichtigen.

Rechenintensive Anwendung

Um die Leistung der Frameworks zu messen, sollen rechenintensive Funktionen implementiert werden. Hierbei soll ermittelt werden, wie lange das Framework für die Berechnungen benötigt und wie stark die CPU ausgelastet wird. Außerdem soll ermittelt werden, wie das Framework mit Extremsituationen wie einem hohen Leistungsbedarf zurechtkommt und ob es vielleicht sogar ab einem gewissen Grad abstürzt.

3.3 Gewichtung

Die Anforderungen an eine Applikation variieren von Anwendungsgebiet zu Anwendungsgebiet. Um dies im Vergleich zu berücksichtigen, erfolgt eine individuelle Gewichtung der Vergleichskriterien je Anwendungsgebiet. Im Rahmen eines paarweisen Vergleichs werden die Vergleichskriterien systematisch gegenübergestellt und entschieden, welches der betrachteten Kriterien wichtiger ist. Die Summe der gewonnenen Vergleiche wird durch die Anzahl der Vergleichskriterien geteilt, um eine prozentuale Gewichtung des betrachteten Vergleichskriteriums zu erhalten.

Im Folgenden werden die paarweise Vergleiche und die daraus resultierende Gewichtung der Vergleichskriterien für die Anwendungsgebiete dargestellt und Auffälligkeiten aufgezeigt.

3.3.1 Formularlastige Anwendung

als / wichtiger	Plattformspezifische Funktionen	Performance	Benutzeroberfläche	Erste Schritte	Entwicklungsunterstützung	Dokumentation	Ökosystem	Lebenszyklus & öffentliches Interesse	Summe	%
Plattformspezifische Funktionen		0	0	1	0	0	1	0	2	7%
Performance	1		0	1	0	1	1	0	4	14%
Benutzeroberfläche	1	1		1	1	1	1	1	7	25%
Erste Schritte	0	0	0		1	0	0	0	1	4%
Entwicklungsunterstützung	1	1	0	0		1	1	0	4	14%
Dokumentation	1	0	0	1	0		1	0	3	11%
Ökosystem	0	0	0	1	0	0		0	1	4%
Lebenszyklus & öffentliches Interesse	1	1	0	1	1	1	1		6	21%
									Prüfsumme	100,00%

Abb. 3: Gewichtung der Kriterien in formularlastigen Anwendungen

In Abbildung 3 ist der paarweise Vergleich und die daraus resultierende Gewichtung der Vergleichskriterien für das Anwendungsgebiet der formularlastigen Anwendungen dargestellt. Die Benutzeroberfläche wurde wichtiger als alle sieben verbleibenden Vergleichskriterien eingestuft und erhält dadurch eine relative Gewichtung von 25%. Der Lebenszyklus und das öffentliche Interesse erhält eine relative Gewichtung von 21%. Performance und Entwicklungsunterstützung erhalten jeweils eine relative Gewichtung von 14%. Erste Schritte und Ökosystem erhalten jeweils eine relative Gewichtung von 4%.

3.3.2 Sensorlastige Anwendung

als / wichtiger	Plattformspezifische Funktionen	Performance	Benutzeroberfläche	Erste Schritte	Entwicklungsunterstützung	Dokumentation	Ökosystem	Lebenszyklus & öffentliches Interesse	Summe	%
Plattformspezifische Funktionen		1	1	1	1	1	1	1	7	25%
Performance	0		1	1	0	0	1	0	3	11%
Benutzeroberfläche	0	0		0	0	0	0	0	0	0%
Erste Schritte	0	0	1		0	0	0	0	1	4%
Entwicklungsunterstützung	0	1	1	1		0	0	1	4	14%
Dokumentation	0	1	1	1	1		0	0	4	14%
Ökosystem	0	0	1	1	1	1		0	4	14%
Lebenszyklus & öffentliches Interesse	0	1	1	1	0	1	1		5	18%
									Prüfsumme	100,00%

Abb. 4: Gewichtung der Kriterien in sensorlastigen Anwendungen

In Abbildung 4 ist der paarweise Vergleich und die daraus resultierende Gewichtung der Vergleichskriterien für das Anwendungsgebiet der sensorlastigen Anwendungen darge-

stellt. Die plattformspezifische Funktionen wurde wichtiger als alle sieben verbleibenden Vergleichskriterien eingestuft und erhalten dadurch eine relative Gewichtung von 25%. Der Lebenszyklus und das öffentliche Interesse erhält eine relative Gewichtung von 18%. Entwicklungsunterstützung, Dokumentation und Ökosystem erhalten jeweils eine relative Gewichtung von 14%. Alle Vergleichskriterien wurden wichtiger als die Benutzeroberfläche eingestuft, diese erhält dadurch eine relative Gewichtung von 0%.

3.3.3 Rechenintensive Anwendung

wichtiger \ als	Plattformspezifische Funktionen	Performance	Benutzeroberfläche	Erste Schritte	Entwicklungsunterstützung	Dokumentation	Ökosystem	Lebenszyklus & öffentliches Interesse	Summe	%
Plattformspezifische Funktionen		0	1	1	0	1	0	0	3	11%
Performance	1		1	1	1	1	1	1	7	25%
Benutzeroberfläche	0	0		0	0	0	0	0	0	0%
Erste Schritte	0	0	1		0	0	0	0	1	4%
Entwicklungsunterstützung	1	0	1	1		1	0	0	4	14%
Dokumentation	0	0	1	1	0		1	0	3	11%
Ökosystem	1	0	1	1	1	0		0	4	14%
Lebenszyklus & öffentliches Interesse	1	0	1	1	1	1	1		6	21%
									Prüfsumme	100,00%

Abb. 5: Gewichtung der Kriterien in rechenintensiven Anwendungen

In Abbildung 5 ist der paarweise Vergleich und die daraus resultierende Gewichtung der Vergleichskriterien für das Anwendungsgebiet der rechenintensiven Anwendungen dargestellt. Die Performance wurde wichtiger als alle sieben verbleibenden Vergleichskriterien eingestuft und erhält dadurch eine relative Gewichtung von 25%. Der Lebenszyklus und das öffentliche Interesse erhält eine relative Gewichtung von 21%. Entwicklungsunterstützung, sowie Ökosystem, erhalten jeweils eine relative Gewichtung von 14%. Alle Vergleichskriterien wurden wichtiger als die Benutzeroberfläche eingestuft, diese erhält dadurch eine relative Gewichtung von 0%.

3.3.4 Auffälligkeiten

Bei der Betrachtung der Ergebnisse fällt auf, dass Lebenszyklus und öffentliches Interesse bei jedem Anwendungsgebiet als zweitwichtigstes Kriterium gewichtet ist. Außerdem ist erkannbar, dass die Benutzeroberfläche hauptsächlich bei formularlastigen Anwendungen relevant und bei rechenintensiven und sensorlastigen Anwendungen zu vernachlässigen ist. In jedem Anwendungsgebiet dominiert ein Vergleichskriterium mit 25%, welches repräsentativ für die Anforderungen des Anwendungsgebiets ist.

4 Hybride Frameworks

In diesem Kapitel werden Frameworks zur Entwicklung hybrider Applikationen untersucht. Dabei werden die Kriterien aus dem vorherigen Kapitel berücksichtigt.

4.1 Flutter

Flutter ist ein modernes, hybrides Framework, das von Google entwickelt und 2018 veröffentlicht wurde. Als Open-Source-Framework steht unter der BSD-Lizenz und ist somit in kommerziellen Projekten nutzbar. Die Entwicklung erfolgt in der objektorientierten Programmiersprache Dart, die als Nachfolger von JavaScript entwickelt wurde und viele der charakteristischen JavaScript-Funktionalitäten implementiert, sich dabei allerdings eher an der Syntax von Java orientiert [Wu 2018]. Außerdem können Bibliotheken in den Programmiersprachen C und C++ eingebunden werden. Mit Flutter können Applikationen für Android, iOS, Windows, Linux, macOS und Browser wie Firefox, Chrome oder Edge entwickelt werden.

Obwohl Flutter ein vergleichsweise neues und modernes Framework ist, zeichnet es sich durch eine detaillierte und umfangreiche Dokumentation aus. Diese deckt den gesamten Entwicklungsprozess ab und leitet den Entwickler von der Installation bis zum Deployment der Flutter-Applikationen. Außerdem sind Beispielprojekte und Videoanleitungen auf dem Youtube-Kanal von Flutter zu finden. Die Dokumentation ist online [4] einsehbar und steht bei Bedarf ebenfalls als Download [5] zur Verfügung. Zum aktuellen Zeitpunkt ist sie ausschließlich auf Englisch verfasst.

Flutter ist für die Plattformen Windows, macOS, Linux und ChromeOS verfügbar. Zur Installation muss das FlutterSDK inklusive der enthaltenen Dart Virtual Machine heruntergeladen, extrahiert und im gewünschten Verzeichnis abgelegt werden. Flutter benötigt mehrere Abhängigkeiten zu anderen Programmen. Diese können mithilfe des Befehls *flutter doctor* in der Konsole abgefragt werden. Konkret werden Android SDK, Android Studio, Visual Studio, Visual Studio Code, IntelliJ IDEA und Chrome benötigt. Außerdem werden sogenannte Host Devices benötigt, auf denen das Debugging der Applikationen stattfinden kann. Für Android können diese im Android Emulator von Android Studio konfiguriert werden. Dazu ist es jedoch notwendig, die VM Hardware Acceleration zu aktivieren, was einen gewissen Aufwand mit sich bringt. Genaue Anweisungen sind in der Flutter Dokumentation zu finden.

Die Entwicklung von Flutter-Applikationen kann in verschiedenen IDEs (Integrierte Entwicklungsumgebung) erfolgen. Visual Studio Code ist die populärste IDE, da sie auch in kommerziellen Projekten kostenlos nutzbar ist und mit der *Flutter Extension* weitere Funktionalitäten zur Bearbeitung, Refactoring und Ausführung der Applikationen hinzugefügt

[4] https://docs.flutter.dev/
[5] https://api.flutter.dev/offline/flutter.docset.tar.gz

werden können. Weitere IDEs sind z. B. Android Studio oder IntelliJ IDEA. Besonders hervorgehoben werden muss die Hot-Reload-Funktion von Flutter, die ein schnelles Nachladen bei Änderungen während der Ausführung der Applikation ermöglicht und auch bei größeren Änderungen noch zuverlässig und schnell funktioniert. Dies geschieht durch Injection der modifizierten Code-Dateien in die laufende Dart Virtual Machine [Zammetti 2019]. Außerdem stehen für Flutter hunderte Templates auf diversen Websites [6] zur Verfügung, die die Entwicklung deutlich beschleunigen können und viele Anwendungsgebiete bereits abdecken.

Flutter ermöglicht einen einfachen Zugriff auf spezifische Funktionen der jeweiligen Zielplattform. Mithilfe von Paketen (z. B. *sensors_plus* oder *flutter_sensors*) kann auf die Sensoren des Endgeräts wie z.B. Beschleunigungssensor, Gyroskop, Magnetometer, Näherungssensor und viele mehr zugegriffen werden. Außerdem ist es möglich, den Zugriff auf plattformspezifische Schnittstellen selbst in Dart zu implementieren, um z. B. den Akkustand abzufragen. Die Speicherverwaltung der Applikationen liegt nicht in der Hand des Entwicklers, sondern wird von der Dart Virtual Machine übernommen, die u. A. einen Garbage Collector für die Freigabe von nicht mehr benötigtem Speicherplatz umfasst.

Die Benutzeroberfläche wird in Flutter modular, d. h. in Form von Komponenten, entwickelt. Dabei verwendet Flutter nicht wie die meisten Frameworks OEM (Original Equipment Manufacturer)-Widgets, die vom Betriebssystem des Endgeräts zur Verfügung gestellt werden, sondern generiert eigene Widgets. Dies ermöglicht ein einzigartiges Design der Widgets und erhöht die Erweiterbarkeit und Flexibilität [xster 2017]. Für die Umsetzung der Benutzeroberfläche wird die Komponente *MaterialApp* als Top-Level-Widget verwendet, da sie z. B. Kopfzeile und Navigator bereits enthält. Die Oberfläche ist üblicherweise als Grid-Struktur organisiert, sodass die Anordnung der Widgets in Reihen (Row) und Spalten (Column) möglich ist. Für das Styling ist kein CSS (Cascading Style Sheets) notwendig, da die Gestaltung mithilfe von Decoration- und Style-Elementen direkt in Dart erfolgen kann. Zu diesem Zweck stehen auch weitere Elemente wie Center(), Positioned() oder Transform() zur Verfügung, um die Widgets innerhalb der Grid-Zelle auszurichten.

Da Flutter nicht die OEM-Widgets des Betriebssystems verwendet, kann in der Regel eine höhere Performance als bei anderen Frameworks erreicht werden. Dies hat den Hintergrund, dass Geschwindigkeitseinbußen, die durch die Kommunikation zwischen Applikation und nativen Komponenten entstehen würden, vermieden werden können [Helios Blog 2020]. Stattdessen nutzt Flutter seine eigene Hochleistungsengine Skia zum Rendern der benötigten Widgets [Wu 2018]. Man spricht daher davon, dass das Rendering von der Systemebene in die Applikationsebene verlagert wird. Dies führt allerdings dazu, dass die Build-Dateien größer als bei anderen Frameworks sind, da die eigenen Widgets und Renderer ebenfalls in der releasten Applikation enthalten sein müssen [Wu 2018]. Grundsätzlich ist Dart als eine Programmiersprache mit hoher Performance zu betrachten, da sowohl JIT (Just-in-Time)-Kompilierung für die Entwicklung als auch AOT (Ahead-of-Time)-Kompilierung

[6] z. B. https://flutterawesome.com/tag/templates/

für den Release der Applikation eingesetzt werden [Dart Documentation 2022]. AOT-kompilierter Code ermöglicht einen schnellen Start des Programms und zuverlässig geringe Ausführungszeiten während des gesamten Programmablaufs [Obinna 2020].

Im Flutter-Ökosystem steht eine große Anzahl an Paketen zur Verfügung. Dabei ist nahezu jeder Bereich von Datenbanken über WebSockets und Audio bis hin zu Animationen abgedeckt. Zur Organisation wird das offizielle Paket-Repository pub.dev verwendet. Von dort können die Pakete über den Pub Package Manager einfach in eigene Projekte integriert werden, indem die Dependency mit der benötigten Version in einer Projektdatei hinterlegt wird und dann beim nächsten Build aufgelöst wird. Anschließend kann das Paket mithilfe des import-Befehls verwendet werden.

Für die Datenhaltung steht eine recht begrenzte, aber wachsende Auswahl an Datenbanksystemen zur Verfügung. Die Pakete sqflite und moor erlauben den Zugriff auf das relationale Datenbanksystem SQLite, indem sie als Wrapper speziell für Flutter fungieren und SQL-Queries sowohl in SQL als auch direkt in Dart ermöglichen. Der Zugriff auf NoSQL Datenbanksysteme ist ebenfalls möglich. Zu diesem Zweck können die Pakete hive und firebase verwendet werden, die Unterstützung für die gleichnamige Key-Value-Datenbank und JSON-Datenbank bieten [Greenrobot 2021].

Flutter steht seit seinem Release in Konkurrenz mit dem Framework React Native. Es ist nicht eindeutig absehbar, welches Framework sich langfristig durchsetzen wird. Allerdings gewinnt Flutter in den letzten Jahren immer weiter Marktanteile und ist seit 2021 das meistverwendete, hybride Framework am Markt [Statista 2021]. Der Umstieg von Unternehmen auf die Softwareentwicklung mit Flutter scheint daher langfristig sinnvoll, da es sich voraussichtlich weiterhin als eines der meistverwendeten Frameworks etablieren wird.

4.2 React Native

React Native wurde im Jahr 2015 von Meta entwickelt und bis zum Jahr 2020 so weiterentwickelt, dass es ausgereift und für den Einsatz im Produktivumfeld geeignet ist. React Native begann als internes Hackathon-Projekt, dabei war das eigentliche Ziel des Projekts, den Entwicklungsprozess von Android und iOS zu vereinheitlichen [Niemeier 2022]. Das Framework ist Open-Source und steht unter der MIT-Lizenz. Aufgrund dieser Lizenz kann das Framework auch kommerziell verwendet werden. Es unterstützt die wichtigsten mobilen Plattformen Android und iOS genauso wie Windows, macOS und AndroidTV. Nicht zuletzt unterstützt es natürlich auch das Entwickeln von Webanwendungen. Viele namhafte Applikationen setzen bereits React Native ein, darunter sind Anwendungen wie Instagram, Skype oder auch Uber Eats [Chandratre 2020].

Die Entwicklung einer App in React Native erfolgt auf Basis von Webtechnologien wie HTML, CSS und JavaScript [Zammetti 2019]. Das Framework setzt unter Anderem auch die verwandte JavaScript-Bibliothek ReactJS ein, welche auch von Meta entwickelt wurde.

Der Unterschied zwischen React und React Native ist, dass React auf die Entwicklung von Web-Applikationen spezialisiert ist, während React Native für die Entwicklung von nativen Apps optimiert ist [Lestal 2020]. Entwickler, die bereits in React Anwendungen entwickelt haben, können diesen Code auch weiterhin nutzen [Krypczyk et al. 2021].

React Native kann schnell und einfach eingerichtet werden. Für die Entwicklungsumgebungen von Jetbrains gibt es Plugins, welche ganz bequem über das Plugin-Portal installiert werden können. Genauso gibt es auch für Visual Studio Code aus dem Hause Microsoft Unterstützung durch Plugins. Um React Native zu installieren, muss zuerst NodeJS und NPM installiert werden. NodeJS ist ein Framework, welches die Entwicklung von JavaScript-Anwendungen erleichtert. Über NPM können andere Bibliotheken per Kommandozeilenbefehl installiert werden. Für React Native gibt es auch ein sogenanntes Command Line Interface. Über dieses Interface kann man das System auf einfache Weise aktualisieren oder Erweiterungen installieren. Ist NPM installiert, so kann durch Aufrufen bestimmter Befehle in der Kommandozeile das Framework installiert und eine Anwendung eingerichtet werden. Da der Code von React Native in nativen Code umgewandelt wird, müssen für Android und iOS ebenso auch die nativen Entwicklungsumgebungen installiert werden. Dies wären Android Studio für Android und AppCode für iOS. Über diese nativen Entwicklungsumgebungen kann man auch Emulatoren der jeweiligen Betriebssysteme starten und so die App auf den Geräten testen [7].

Projekte in React Native bestehen aus Code, welcher in JavaScript erstellt wird. Dieser ist in Views unterteilt, welche die Oberfläche widerspiegeln und Services, welche die Fachlogik enthalten. Oberflächen werden in React entwickelt, die Geschäftslogik wird in JavaScript verfasst. Nach dem Build der Anwendung werden Views in native Views umgewandelt. Services nutzen über eine Bridge die Schnittstellen der nativen Plattformen [Krypczyk et al. 2021]. React Native unterstützt den Hot-Reloading Mechanismus, sodass Änderungen im Programmcode zur Laufzeit durchgeführt werden können, ohne dass die Anwendung neu gestartet werden muss. Jede React Native Anwendung besteht hauptsächlich aus zwei unterschiedlichen Arten von Threads. Dabei kümmert sich einer der Threads als Haupt-Thread um die Anzeige der Elemente in der Benutzeroberfläche, während der andere Thread für die Ausführung des JavaScript-Codes verantwortlich ist. Er definiert außerdem die Funktionalitäten der Elemente auf der Benutzeroberfläche [Niemeier 2022].

Da React Native Open-Source ist, gibt es viele Bibliotheken und Erweiterungen von anderen Entwicklern. Daher gibt es auch Bibliotheken, welche React Native um die Nutzung von Sensoren erweitern [Schmidt 2018]. So werden z. B. Geschwindigkeitssensor, Gyroskop, Magnetometer und Barometer unterstützt. Über eine gut dokumentierte, native Schnittstelle werden auch native Elemente für die Oberfläche unterstützt. Dabei kann es sich z. B. um diverse Views in Android handeln.

In einer Datei können verschiedene Views hinterlegt werden, welche die Oberflächen der Applikation definieren. Views können sich auch je nach Plattform unterscheiden, sodass

[7] https://www.tutorialspoint.com/react_native/react_native_environment_setup.htm

eine App auf Android später anders aussehen kann als auf iOS. Die einzelnen Views können modular aufgebaut werden und aus verschiedenen Dateien bestehen.

React Native unterstützt sehr viele Technologien und Schnittstellen, welche häufig bei der App-Entwicklung benötigt werden. So kann grundsätzlich auf native Oberflächenelemente zugegriffen werden. Der grundlegende Funktionsumfang kann beliebig durch eine der zahlreichen Erweiterungen ergänzt werden. Zum Beispiel kann durch Erweiterungen die Unterstützung für folgende Datenbank-Systemen ermöglicht werden: Firebase, SQLite, Realm, PouchDB, AsyncStorage und Weitere.

Zusammenfassend kann festgehalten werden, dass React Native voraussichtlich noch länger unterstützt wird, da hinter dem Framework eine namhafte Firma wie Meta steht. Das Projekt wird auf Open-Source-Basis entwickelt und kann so potenziell von jedem Entwickler weiterentwickelt werden. React Native hat einen großen Funktionsumfang und kann durch zusätzliche Erweiterungen dynamisch ausgebaut werden. Applikationen werden einmal gebaut und können für eine Vielzahl von Plattformen entwickelt werden. Um native Apps zu testen, müssen allerdings auch Entwicklungsumgebungen für die jeweiligen Plattformen installiert sein.

4.3 .NET MAUI

.NET MAUI (Multi-Platform App UI) ist ein Framework von Microsoft, mit dem seit 2022 plattformunabhängige Applikationen entwickelt werden können. MAUI ist der offizielle Nachfolger von Xamarin und verwendet bei linuxbasierten Zielplattformen ebenfalls die Laufzeitumgebung Mono [Davidbritch 2022d].

Mit MAUI können Applikationen für Android, iOS, Windows, macOS und Tizen entwickelt werden. Die gemeinsame Codebasis wird in der objektorientierten Programmiersprache C# geschrieben. Die Benutzeroberfläche kann wahlweise über die XML-basierte Beschreibungssprache XAML (Extensible Application Markup Language) oder unter Verwendung des Single-Page Web-Frameworks .NET Blazor erstellt werden. Bei letzterem spricht man von MAUI Blazor oder auch Blazor Hybrid, welches die Zielplattform um die gängigen Browser wie Chrome, Firefox und Edge erweitert.

Die Benutzeroberfläche wird in MAUI Blazor modular, d. h. in Form von wiederverwenbaren, dynamischen Komponenten entwickelt. Bei einer Razor-Komponente handelt es sich um einen eigenständigen Teil der Benutzeroberfläche und die dazugehörige Verarbeitungslogik. Die Komponenten werden in einer Kombination aus C#, HTML-Markup und optionalem JavaScript implementiert. C# ersetzt hierbei das bei Web-Frameworks üblicherweise benötigte JavaScript, sodass keine Kenntnisse über JavaScript benötigt werden. Die Darstellungsvorgaben können über CSS beliebig definiert werden. Sowohl einzelne Seiten als auch die komplette Blazor Applikation lassen sich über ein BlazorWebView-Steuerelement in der MAUI Applikation hosten. Die Web-UI wird anschließend in dem eingebetteten

Steuerelement gerendert und die Komponenen werden im systemeigenen Prozess ausgeführt [Davidbritch 2022a].

MAUI unterliegt der MIT-Lizenz und Blazor unterliegt der Apache-2.0-Lizenz, somit handelt es sich um Open-Source Software.

Obwohl MAUI und Blazor vergleichsweise neue Frameworks sind, zeichnen sie sich durch eine detaillierte und umfangreiche Dokumentation aus. Diese deckt den gesamten Entwicklungsprozess ab und leitet Entwickler*innen von der Installation bis zum Deployment der hybriden Applikation. Außerdem sind Beispielprojekte und Videoanleitungen auf der Lernplatform [8] und dem Youtube-Kanal von Microsoft und dotnet zu finden. Die Dokumentation ist online [9] einsehbar und verweist auf weitere nützliche Ressourcen. Zum aktuellen Zeitpunkt ist sie auf Englisch und auf Deutsch verfasst.

Die Entwicklung von MAUI Applikationen kann in verschiedenen IDEs erfolgen. Visual Studio 2022, Visual Studio 2022 für Mac, Visual Studio Code und Rider sind namhafte Vertreter. Mit wenigen Schritten kann die bestehende Visual Studio Installation um den .NET MAUI Entwicklungsworkload erweitert werden. Hierzu werden weder Konsolenbefehle noch die manuelle Installation von weiteren Programmen benötigt. Der in Visual Studio enthaltene Android Emulator ermöglicht das Verwalten und Debuggen der simulierten Geräte, ohne die Entwicklungsumgebung zu verlassen. Um die Zielplattformen macOS und iOS zu Debuggen, wird ein Mac benötigt, welcher remote in die Entwicklungsumgebung eingebunden werden kann. Visual Studio ermöglicht ebenfalls Hot Reloading. Durch die Verwendung von Templates kann der Entwicklungsprozess beschleunigt werden [Davidbritch 2022b].

Für MAUI und Blazor als Teile des .NET-Ökosystems gibt es viele Bibliotheken und Erweiterungen von anderen Entwicklern. Diese können unkompliziert über die Paketverwaltung NuGet eingebunden werden. Durch Blazors Interoperabilität zu JavaScript ist es ebenfalls möglich, JavaScript-Bibliotheken einzubinden. Für Sensoren, die weder über die MAUI-API noch über Bibliotheken angesprochen werden können, besteht die Möglichkeit, plattformspezifischen Code aus der gemeinsamen Codebasis aufzurufen. Für diese plattformbedingte Kompilierung werden entsprechende Kentnisse über die Schnittstellen der Zielplattform benötigt [Davidbritch 2022c].

MAUI vereint plattformspezifische Frameworks in einer einzigen API. Die geteilte Codebasis interagiert überwiegend mit der zielplattformunabhängigen MAUI-API, welche anschließend die spezifischen APIs der Zielplattformen konsumiert. Ein direkter Aufruf von Schnittstellen der Zielplattformen ist ebenfalls möglich. Abhängig von der Zielplattform wird wahlweise Ahead-of-Time (AOT) und Just-in-Time (JIT) Kompilierung ermöglicht [Davidbritch 2022d]. In Abbildung 6 ist die Architektur einer MAUI Applikation dargestellt.

Zusammenfassend kann festgehalten werden, dass die Entwicklung des Frameworks noch

[8] https://dotnet.microsoft.com/en-us/learn/maui
[9] https://learn.microsoft.com/de-de/aspnet/core/blazor/hybrid/?view=aspnetcore-6.0

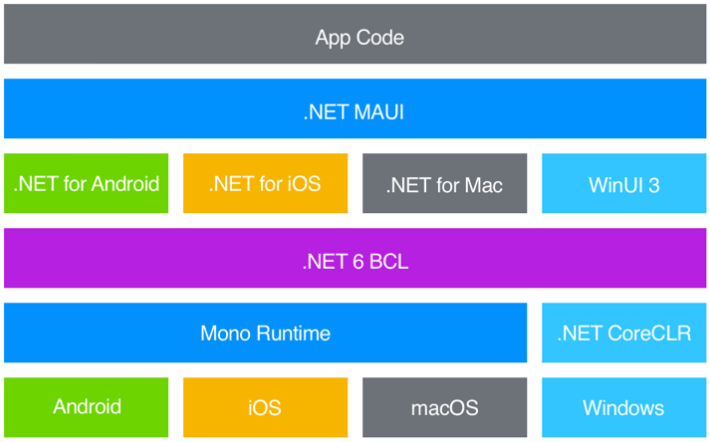

Abb. 6: Übersicht über die Architektur einer MAUI Applikation nach [Davidbritch 2022d]

weiter voran geht und als Nachfolger von Xamarin auch entsprechend lange unterstützt wird. MAUI hat einen großen Funktionsumfang und kann durch zusätzliche Erweiterungen dynamisch ausgebaut werden. Durch die Möglichkeit, XAML oder Blazor einzusetzen, richtet sich MAUI sowohl an Webentwickler*innen, als auch an Entwickler*innen mit Erfahrung in Desktop-Applikationen.

5 Vergleich der Frameworks

Im folgenden Kapitel wird der Vergleich der Frameworks durchgeführt. Dabei wird zunächst jedes Framework anhand der ermittelten Vergleichskriterien bewertet. Zur Bestimmung der Performance findet ein Benchmarking statt, bei dem Berechnungszeiten und CPU-Auslastung für rechenintensive Funktionen dokumentiert werden. Anschließend kann mit den Bewertungen und den zuvor bestimmten Gewichtungen für die Kriterien in jedem definierten Anwendungsgebiet eine Punktzahl errechnet werden, anhand derer die Frameworks sortiert werden können.

5.1 Prototypen

Im Rahmen des Vergleichs werden für jedes Framework drei unterschiedliche Prototypen umgesetzt (vgl. Abschnitt 3.2). Zur Demonstration einer formularlastigen App wird eine To-Do-Listen Applikation umgesetzt. Diese nimmt Aufgaben per Formular auf und speichert sie in einer lokalen Datenbank ab. Außerdem können Aufgaben gelöscht oder als „erledigt“ markiert werden.
Die zweite Applikation ist eine sensorlastige App. Hierbei werden die gängigsten Sensoren über das Framework angesprochen und die Daten ausgelesen. Wichtig sind hierbei Sensoren wie z. B. der Beschleunigungssensor, das Gyroskop oder das Magnetometer.
Um die Performance zu vergleichen, wurden bei der rechenintensiven App eine rekursive und eine iterative Funktion implementiert. Bei der rekursiven Funktion handelt es sich um die Ackermann-Funktion. Diese ist dafür bekannt, sehr stark anzusteigen und die Rechenkapazität schnell zu erschöpfen. Die Ackermann-Funktion ist im Folgenden definiert [Tutego 2015].

```
a(0,m) = m + 1
a(n,0) = a(n - 1, 1)
a(n,m) = a(n - 1, a(n, m - 1))
```

Als iterative Funktion dient eine Implementierung der Potenzierung, bei der die Berechnung der potenzierten Zahl iterativ erfolgt. Hierbei werden die Basis und der Exponent als Parameter übergeben.

5.2 Benchmarks

Um die rechenintensive App zu testen, werden die implementierten Funktionen (vgl. Kapitel 5.1) ausgeführt und dabei die Zeit gemessen, die die Applikation für die Ausführung bzw. Berechnung benötigt. Außerdem wurde die CPU-Auslastung dokumentiert. Zur Ausführung und Messung wird das Samsung Galaxy S20 FE verwendet.

Bei der Ackermann-Funktion wurden für die Parameter n und m folgende Wertebelegungen gemessen: (n=3, m=8) und (n=3, m=12). Die Messergebnisse sind in Abbildung 7 dargestellt. Bei der Durchführung der Messungen wurde festgestellt, dass Flutter und MAUI nur geringe

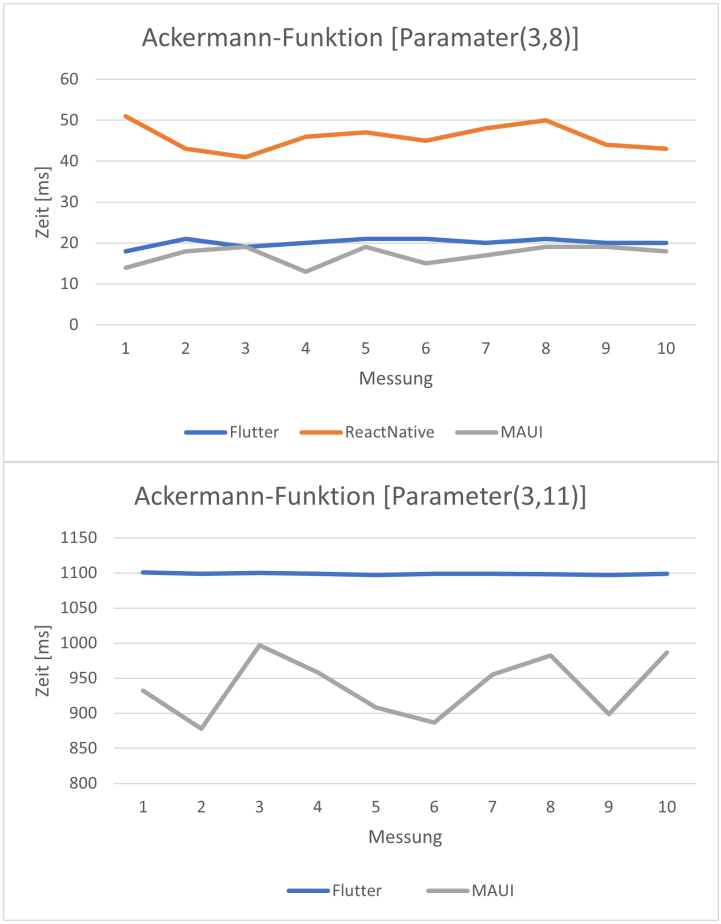

Abb. 7: Analyse der Frameworks anhand der Ackermann-Funktion

Unterschiede bei der Ausführungszeit aufweisen. MAUI berechnet die Ackermann-Funktion bei beiden Varianten etwas schneller als Flutter. Konkret benötigt MAUI jeweils rund 15% weniger Rechenzeit. Deutlich abgeschlagen ist in beiden Fällen das dritte Framework React Native. Bei den Parametern (n=3, m=8) benötigte es 2,5x länger als Flutter, um das Ergebnis der Ackermann-Funktion zu berechnen. Bei (n=3, m=12) konnte React Native kein Ergebnis errechnen und brach schon vorher mit einem StackOverflow-Fehler ab.

Bei der iterativen Potenzierung wurde mit der Basis 12 und dem Exponenten 9 gemessen. Die Messergebnisse sind in Abbildung 8 dargestellt. Dabei ist derselbe Trend wie bei der

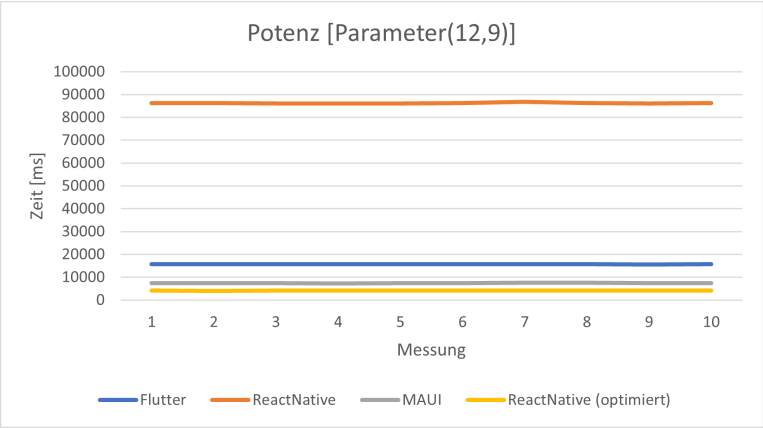

Abb. 8: Analyse der Frameworks anhand der Iterativen Potenzierung

Ackermann-Funktion erkennbar. MAUI hat erneut die höchste Performance und ist konkret 50% schneller als Flutter. Außerdem ist React Native zeitlich erneut weit von den anderen Frameworks entfernt. Es benötigt rund 5,5x länger als Flutter zur Berechnung der Potenz. Durch Ausführung eines nativen Moduls war es möglich, ReactNative zu optimieren und ebenfalls eine hohe Performance zu erreichen.

Die Auslastung der CPU wurde bei einer rechenintensiven Berechnung über einen Zeitraum von 30 Sekunden gemessen. Die Messergebnisse sind in Abbildung 9 dargestellt. Es ist

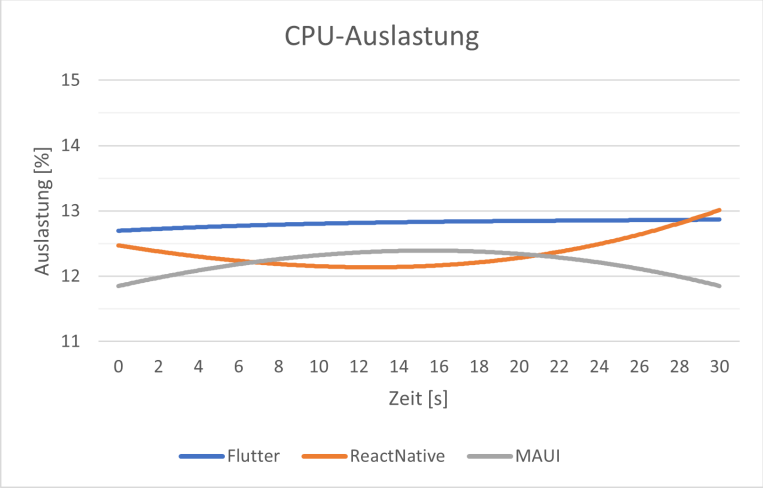

Abb. 9: CPU-Auslastung der Frameworks

sichtbar, dass alle Frameworks annähernd dieselben Auslastung generieren. Konkret lastet

MAUI die CPU mit geringem Abstand am wenigsten aus, während Flutter tendenziell eine leicht höhere Auslastung herbeiführt.

5.3 Bewertung der Frameworks

In diesem Abschnitt werden die Frameworks für die ermittelten Vergleichskriterien bewertet. Jedes Kriterium kann mit minimal 0 Punkten und maximal 5 Punkten bewertet werden. Dabei werden die ermittelten Eigenschaften der Frameworks aus der Literaturarbeit in Kapitel 4 verwendet. Außerdem werden die Ergebnisse des Benchmarkings in Abschnitt 5.2 und die subjektiven Eindrücke bei der Erstellung der Prototypen miteinbezogen.

5.3.1 Flutter

Flutter erhält aufgrund der angegebenen Faktoren die folgenden Bewertungen.

Plattformspezifische Funktionen (5/5) Es ist ein einfacher Zugriff auf Sensoren des Endgeräts gegeben. Außerdem ist eine Möglichkeit zur eigenständigen Implementierung beim Zugriff auf plattformspezifische Schnittstellen vorhanden.

Performance (4/5) Es liegt eine hohe Performance durch u. A. AOT-Kompilierung vor. Da Flutter laut Benchmarking minimal langsamer als MAUI ist, ist ein kleiner Punktabzug notwendig.

Benutzeroberfläche (4/5) Flutter hat ein einzigartiges Design, da keine OEM-Widgets verwendet werden, sondern mit einer eigenen Renderengine gearbeitet wird. Es ist ein kleiner Punktabzug notwendig, da im Gegensatz zu den anderen Frameworks keine Gestaltung per CSS möglich ist.

Erste Schritte (2/5) Es ist aufgrund der vergleichsweise aufwändigen Installation ein starker Punktabzug notwendig, da viele Abhängigkeiten zu anderen Programmen vorliegen. Die Einarbeitung in Dart kann Zeit benötigen, da der Aufbau der Anwendung teilweise nicht intuitiv ist.

Entwicklungsunterstützung (4/5) Es ist eine Unterstützung für viele verschiedene IDEs inklusive Erweiterungen vorhanden. Durch Hot Reload wird eine schnelle Entwicklung ermöglicht. Es ist ein kleiner Punktabzug notwendig, da die Unterstützung beim Anlegen von Widgets besser sein könnte.

Dokumentation (5/5) Die Dokumentation ist online und offline verfügbar und sehr umfangreich. Außerdem sind Videoanleitungen auf Youtube vorhanden.

Ökosystem (4/5) Pakete können einfach über den Pub Package Manager eingebunden werden. Es ist eine umfangreiche Paketauswahl vorhanden. Ein kleiner Punktabzug ist notwendig, da kein vergleichbar großes Ökosystem wie z. B. bei .NET vorliegt.

Lebenszyklus & öffentliches Interesse (5/5) Flutter ist das meistverwendete, hybrides Framework im Jahr 2021 mit steigenden Tendenzen. Es ist sehr modern und noch am Beginn des Lebenszyklus, da der Release erst 2018 war.

5.3.2 React Native

React Native erhält aufgrund der angegebenen Faktoren die folgenden Bewertungen.

Plattformspezifische Funktionen (5/5) Es ist ein einfacher Zugriff auf Sensoren des Endgeräts gegeben. Außerdem sind Bibliotheken vorhanden, die Dateizugriffe übernehmen und einfach gestalten. Es gibt die Möglichkeit, über Android z. B. eigene Module zu implementieren.

Performance (2/5) Es ist ein starker Punktabzug notwendig, da React Native eindeutig die geringste Performance der drei Frameworks hat. Die Optimierung durch native Module wiegt dies nicht auf.

Benutzeroberfläche (4/5) Native Widgets des jeweiligen Systems werden verwendet. Das Design erfolgt über spezielles CSS, somit ist die Gestaltung sehr variabel. Ein kleiner Punktabzug ist notwendig, da keine eigenen, einzigartigen Widgets benutzt werden.

Erste Schritte (3/5) Es ist ein kleiner Punktabzug notwendig, da die Installation etwas aufwändiger ist. Nach der Installation genügen allerdings einfache Befehle zur Erstellung von Apps.

Entwicklungsunterstützung (4/5) Viele verschiedene IDEs inklusive Erweiterungen werden unterstütz. React Native ist z. B. in der IDE Webstorm von JetBrains bereits bei Installation vorhanden. Es ist ein kleiner Punktabzug notwendig, da die Unterstützung beim Anlegen von Widgets besser sein könnte.

Dokumentation (5/5) Die Dokumentation ist online umfangreich verfügbar. Außerdem sind zahlreiche Bücher erhältlich.

Ökosystem (4/5) Über NPM können Module einfach per Terminal hinzugefügt werden. Eine umfangreiche Paketauswahl ist ebenfalls vorhanden. Ein kleiner Punktabzug ist notwendig, da kein vergleichbar großes Ökosystem wie z. B. bei .NET vorliegt.

Lebenszyklus & öffentliches Interesse (5/5) React Native wird für viele bekannte Apps verwendet und durch Meta weiterentwickelt. Das Framework ist ausgereift, da es bereits seit 2015 auf dem Markt ist.

5.3.3 MAUI

MAUI erhält aufgrund der angegebenen Faktoren die folgenden Bewertungen.

Plattformspezifische Funktionen (4/5) Der Zugriff auf Sensoren des Endgeräts ist durch Pakete gegeben. Eine Möglichkeit zur eigenständigen Implementierung in C# beim Zugriff auf plattformspezifische Schnittstellen ist vorhanden. Es ist ein kleiner Punktabzug notwendig, da der Zugriff auf Sensoren aufwändiger als in den anderen Frameworks ist.

Performance (5/5) MAUI hat u. A. durch die AOT-Kompilierung eine sehr hohe Performance. Es ist minimal schneller als Flutter und deutlich schneller als React Native.

Benutzeroberfläche (4/5) Die Gestaltung der Oberfläche erfolgt über HTML und CSS. Bereits erstellte Webanwendungen können wiederverwendet werden. Ein kleiner Punktabzug ist notwendig, da keine eigenen, einzigartigen Widgets benutzt werden.

Erste Schritte (5/5) Die Installation und Einrichtung ist sehr einfach, da Visual Studio den Großteil der Arbeit abnimmt und Abhängigkeiten nachinstalliert. Der Aufbau der Anwendungen ist intuitiv und orientiert sich an anderen Microsoft-Produkten.

Entwicklungsunterstützung (4/5) Unterstützung für viele verschiedene IDEs ist vorhanden. Die Hot Reload-Funktion ermöglicht eine schnelle Entwicklung und funktioniert zuverlässig. Der Android Emlator ist direkt in Visual Studio integriert. Ein kleiner Punktabzug ist notwendig, da die Unterstützung beim Anlegen von Widgets besser sein könnte.

Dokumentation (4/5) Die Dokumentation ist online umfangreich verfügbar. Außerdem sind Videoanleitungen und Lernplattformen vorhanden. Es gibt einen kleinen Punktabzug, da Verwechslungsgefahr zwischen klassischem MAUI (via XAML) und MAUI Blazor besteht.

Ökosystem (5/5) Der NuGet Package Manager kann zum Einbinden von Paketen entweder im Terminal oder auf einer grafischen Oberfläche verwendet werden. Das .NET-Ökosystem ist sehr umfangreich und es werden viele Anwendungsbereiche unterstützt.

Lebenszyklus & öffentliches Interesse (3/5) MAUI ist ein sehr junges Framework mit großem Interesse im .NET-Umfeld. Es steht am Beginn des Lebenszyklus, da der Release erst 2022 war. Es ist ein kleiner Punktabzug notwendig, da noch kleinere Fehler bzw. „Kinderkrankheiten“ im Framework vorhanden sind.

5.4 Vergleich und Ergebnisse

Im Folgenden werden die in Abschnitt 5.3 ermittelten Bewertungen der Frameworks unter Berücksichtigung der in Abschnitt 3.3 definierten Gewichtungen der Vergleichskriterien je Anwendungsgebiet miteinander verglichen.

5.4.1 Formularlastige Anwendung

		React		Flutter		MAUI	
	Gewichtung	Bewertung	Wert	Bewertung	Wert	Bewertung	Wert
Plattformspezifische Funktionen	7%	5	0,36	5	0,36	4	0,29
Performance	14%	2	0,29	4	0,57	5	0,71
Benutzeroberfläche	25%	4	1,00	4	1,00	4	1,00
Erste Schritte	4%	3	0,11	2	0,07	5	0,18
Entwicklungsunterstützung	14%	4	0,57	4	0,57	4	0,57
Dokumentation	11%	5	0,54	5	0,54	4	0,43
Ökosystem	4%	4	0,14	4	0,14	5	0,18
Lebenszyklus & öffentliches Interesse	21%	5	1,07	5	1,07	3	0,64
	Summe		**4,07**		**4,32**		**4,00**

Abb. 10: Bewertung für formularlastige Anwendungen

Im Anwendungsgebiet der formularlastigen Anwendungen setzt sich Flutter mit einer Gesamtbewertung von 4,32 gegen React Native mit 4,07 und MAUI mit 4,00 durch. Alle Frameworks erreichen eine Wertung von größer oder gleich 4 Punkten von den maximal erreichbaren 5 Punkten. In dem für dieses Anwendungsgebiet am höchsten gewichteten Vergleichskriterium schneiden alle Frameworks gleich gut ab. Das Bewertungsschema der einzelnen Vergleichskriterien je Anwendungsgebiet und Framework, sowie die Gesamtbewertung eines jeden Frameworks ist in Abbildung 10 dargestellt.

5.4.2 Rechenintensive Anwendung

Im Anwendungsgebiet der rechenintensiven Anwendungen setzt sich Flutter mit einer Gesamtbewertung von 4,36 gegen MAUI mit 4,21 und React Native mit 3,89 durch. Lediglich Flutter und MAUI erreichen eine Wertung von größer oder gleich 4 Punkten von den maximal erreichbaren 5 Punkten. In dem für dieses Anwendungsgebiet am höchsten gewichteten Vergleichskriterium schneidet MAUI am besten ab, dennoch erreicht Flutter eine höhere Gesamtwertung. React Native schneidet bei diesem Vergleichskriterium am schlechtesten ab. Das Bewertungsschema der einzelnen Vergleichskriterien je Anwendungsgebiet und

		React		Flutter		MAUI	
	Gewichtung	Bewertung	Wert	Bewertung	Wert	Bewertung	Wert
Plattformspezifische Funktionen	11%	5	0,54	5	0,54	4	0,43
Performance	25%	2	0,50	4	1,00	5	1,25
Benutzeroberfläche	0%	4	-	4	-	4	-
Erste Schritte	4%	3	0,11	2	0,07	5	0,18
Entwicklungsunterstützung	14%	4	0,57	4	0,57	4	0,57
Dokumentation	11%	5	0,54	5	0,54	4	0,43
Ökosystem	14%	4	0,57	4	0,57	5	0,71
Lebenszyklus & öffentliches Interesse	21%	5	1,07	5	1,07	3	0,64
	Summe		**3,89**		**4,36**		**4,21**

Abb. 11: Bewertung für rechenintensive Anwendungen

Framework, sowie die Gesamtbewertung eines jeden Frameworks ist in Abbildung 11 dargestellt.

5.4.3 Sensorlastige Anwendung

		React		Flutter		MAUI	
	Gewichtung	Bewertung	Wert	Bewertung	Wert	Bewertung	Wert
Plattformspezifische Funktionen	25%	5	1,25	5	1,25	4	1,00
Performance	11%	2	0,21	4	0,43	5	0,54
Benutzeroberfläche	0%	4	-	4	-	4	-
Erste Schritte	4%	3	0,11	2	0,07	5	0,18
Entwicklungsunterstützung	14%	4	0,57	4	0,57	4	0,57
Dokumentation	14%	5	0,71	5	0,71	4	0,57
Ökosystem	14%	4	0,57	4	0,57	5	0,71
Lebenszyklus & öffentliches Interesse	18%	5	0,89	5	0,89	3	0,54
	Summe		**4,32**		**4,50**		**4,11**

Abb. 12: Bewertung für sensorlastige Anwendungen

Im Anwendungsgebiet der sensorlastigen Anwendungen setzt sich Flutter mit einer Gesamtbewertung von 4,50 gegen React Native mit 4,32 und MAUI mit 4,11 durch. Alle Frameworks erreichen eine Wertung von größer oder gleich 4 Punkten von den maximal erreichbaren 5 Punkten. In dem für dieses Anwendungsgebiet am höchsten gewichteten Vergleichskriterium erhalten Flutter und React Native die maximal erreichbare Punktzahl.

MAUI schneidet beim bei diesem Vergleichskriterium etwas schlechter ab. Das Bewertungsschema der einzelnen Vergleichskriterien je Anwendungsgebiet und Framework, sowie die Gesamtbewertung eines jeden Frameworks ist in Abbildung 12 dargestellt.

5.4.4 Ergebnis

In Abbildung 13 ist das Ergebnis des Vergleichs übersichtlich zusammengefasst. Es ist erkennbar, dass Flutter insgesamt die beste Bewertung erreicht hat. Dahinter folgen mit einigem Abstand MAUI und ReactNative.

Framework / Anwendungsgebiet	Flutter	MAUI	ReactNative
Formularlastig	4,32	4,00	4,07
Rechenintensiv	4,36	4,21	3,89
Sensorlastig	4,50	4,11	4,32
	13,18	**12,32**	**12,28**

Abb. 13: Übersicht über Punktzahlen je Framework und Anwendungsgebiet

6 Fazit

Alle drei Frameworks erleichtern die Entwicklung von mobilen Applikationen, indem der Kern der Applikation nur einmal entwickelt werden muss. So kann viel Zeit gespart werden, die sonst in die Entwicklung separater Anwendungen in unterschiedlichen Programmiersprachen für jedes einzelne Betriebssystem investiert werden müsste. Bei allen getesteten Frameworks wird die Erstellung einer Web-App, einer Android-App und einer iOS-App unterstützt. Außerdem ist abhängig vom individuellen Framework die Entwicklung für weitere Systeme möglich. So kann man z. B. mit Maui und React Native Windows-Store-Apps erstellen, während Flutter die Erstellung von Windows Executables ermöglicht.

Alle Frameworks überzeugen durch eine gute, ausführliche Online-Dokumentation. Bei Flutter stellte sich die Einrichtung jedoch als etwas komplizierter heraus, da viele Abhängigkeiten zu anderen Programmen bestehen. Außerdem erlauben alle Frameworks eine große Vielfalt an Gestaltungsmöglichkeiten, da z. B. die Gestaltung durch eine eingeschränkte Form von CSS möglich ist. Sowohl React Native als auch Maui bringen zudem den Vorteil mit sich, dass der Code von bereits existierenden Web-Apps übernommen werden kann. Da hinter allen Frameworks namhafte Firmen wie Microsoft, Google und Meta stehen, kann man davon ausgehen, dass die Frameworks auch in Zukunft weiterentwickelt werden und weiter unterstützt werden.

Bei der Durchführung des Benchmarkings konnte ermittelt werden, dass die Prototypen der Frameworks Flutter und Maui sehr performant sind. Das Framework React Native ist hingegen deutlich weniger performant. Die CPU-Auslastung ist bei allen Frameworks nahezu identisch.

Mit einer formularlastigen Anwendung, einer sensorlastigen Anwendung und einer rechenintensiven Anwendung wurden drei Anwendungsgebiete definiert. Durch die Ermittlung von Vergleichskriterien, einer individuellen Gewichtung je Anwendungsgebiet und einer anschließenden Bewertung konnten je Framework und Anwendungsgebiet Punktzahlen berechnet werden. Dabei konnte sich Flutter in jedem Anwendungsgebiet vor MAUI und React Native durchsetzen. Das abschließende Ergebnis des Vergleichs ist in Abbildung 13 dargestellt. Es ist ersichtlich, dass Flutter insgesamt die beste Bewertung erreicht hat. Auf dem zweiten Platz befindet sich MAUI, auf dem dritten Platz folgt React Native.

Literatur

[Chandratre 2020] Ruchir Chandratre. „Liste der Dinge, die Sie beachten sollten, bevor Sie mit der Entwicklung mobiler Apps mit React Native beginnen“. In: *Cisin* (28. Jan. 2020). URL: https://www.cisin.com/coffee-break/de/enterprise/list-of-things-you-should-keep-in-mind-before-you-start-developing-mobile-apps-with-react-native.html (besucht am 04. 11. 2022).

[Cowart 2012] Jim Cowart. *What is a Hybrid Mobile App?* Hrsg. von Telerik Blogs. 2012. URL: https://www.telerik.com/blogs/what-is-a-hybrid-mobile-app- (besucht am 18. 10. 2022).

[Dart Documentation 2022] Dart Documentation. *Dart Overview*. 2.11.2022. URL: https://dart.dev/overview#platform (besucht am 02. 11. 2022).

[Davidbritch 2022a] Davidbritch. *Hosten einer Blazor-Web-App in einer .NET MAUI-App mit BlazorWebView - .NET MAUI*. 4.12.2022. URL: https://learn.microsoft.com/de-de/dotnet/maui/user-interface/controls/blazorwebview?view=net-maui-7.0 (besucht am 04. 12. 2022).

[Davidbritch 2022b] Davidbritch. *Installieren von Visual Studio 2022 zum Entwickeln plattformübergreifender Apps mit .NET MAUI - .NET MAUI*. 4.12.2022. (Besucht am 04. 12. 2022).

[Davidbritch 2022c] Davidbritch. *NET MAUI invoking platform code - .NET MAUI*. 6.11.2022. URL: https://learn.microsoft.com/en-us/dotnet/maui/platform-integration/invoke-platform-code (besucht am 06. 11. 2022).

[Davidbritch 2022d] Davidbritch. *Was ist .NET MAUI? - .NET MAUI*. 4.12.2022. URL: https://learn.microsoft.com/de-de/dotnet/maui/what-is-maui?view=net-maui-7.0 (besucht am 04. 12. 2022).

[Denko et al. 2021] 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* 13.1 (2021), S. 78–90. ISSN: 2643-7937. DOI: 10.4018/IJSPPC.2021010105.

[Dharmwan 2021] Subodh Dharmwan. „7 examples of hybrid apps that have taken businesses to the next level“. In: *Cynoteck Technology Solutions* (27. Apr. 2021). URL: https://cynoteck.com/de/blog-post/hybrid-apps-that-have-taken-businesses-to-the-next-level/ (besucht am 04. 12. 2022).

[Greenrobot 2021] Greenrobot. *What is the best Flutter Database?* 2021. URL: https://greenrobot.org/news/flutter-databases-a-comprehensive-comparison/ (besucht am 04. 11. 2022).

[Helios Blog 2020] Helios Blog. *Flutter vs. React Native: Which one should you opt for in 2020?* 2020. URL: https://www.heliossolutions.co/blog/flutter-vs-react-native-which-one-should-you-opt-for-in-2020/ (besucht am 02. 11. 2022).

[Kleinschrod 2020] Bernd Kleinschrod. *Native App vs Web App vs Hybrid App*. Hrsg. von Webraketen. 2020. URL: https://webraketen.io/app-native-web-hybrid/ (besucht am 19. 10. 2022).

[Krypczyk et al. 2021] Veikko Krypczyk und Dirk Mittmann. *React Native im Überblick*. 2021. URL: https://entwickler.de/react/react-native-im-uberblick (besucht am 04. 11. 2022).

[Lestal 2020] Justin Lestal. „React vs React Native: What's the difference?" In: *Devskiller* (12. Aug. 2020). URL: https://devskiller.com/react-vs-react-native-whats-the-difference/ (besucht am 04. 11. 2022).

[Niemeier 2022] Susan Niemeier. *React Native*. 4.11.2022. URL: https://www.tenmedia.de/de/glossar/react-native (besucht am 04. 11. 2022).

[Obinna 2020] Onuoha Obinna. „How does JIT and AOT work in Dart? - Onuoha Obinna - Medium". In: *Medium* (7. Apr. 2020). URL: https://onuoha.medium.com/how-does-jit-and-aot-work-in-dart-cab2f31d9cb5 (besucht am 02. 11. 2022).

[Que et al. 2016] Peixin Que, Xiao Guo und Maokun Zhu. „A Comprehensive Comparison between Hybrid and Native App Paradigms". In: *2016 8th International Conference on Computational Intelligence and Communication Networks. CICN 2016 : 23-25 December 2016, THDC Institute of Hydropower Engg and Technology, Bhagirathipuram, Tehri, India : proceedings*. 2016 8th International Conference on Computational Intelligence and Communication Networks (CICN) (Tehri, India). Hrsg. von G. S. Tomar. Piscataway, NJ: IEEE, 2016, S. 611–614. ISBN: 978-1-5090-1144-5. DOI: 10.1109/CICN.2016.125.

[Schmidt 2018] Daniel Schmidt. „Using Sensors in React Native - React Native Training - Medium". In: *React Native Training* (14. Mai 2018). URL: https://medium.com/react-native-training/using-sensors-in-react-native-b194d0ad9167 (besucht am 04. 11. 2022).

[Statista 2021] Statista. *Cross-platform mobile frameworks used by global developers 2021. The State of Developer Ecosystem 2021*. Hrsg. von JetBrains. 2021. URL: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/ (besucht am 21. 10. 2022).

[Statista 2022] Statista. *Mobile Betriebssysteme - Marktanteile Internetnutzung weltweit bis September 2022*. Hrsg. von StatCounter. 2022. URL: https://de.statista.com/statistik/daten/studie/184335/umfrage/marktanteil-der-mobilen-betriebssysteme-weltweit-seit-2009/ (besucht am 06. 11. 2022).

[Tutego 2015] Tutego. *Die Ackermann-Funktion*. de. Copyright: Christian Ullenboom, www.tutego.de. 10.02.2015. URL: http://www.tutego.de/java/articles/Ackermann-Funktion.html (besucht am 29. 11. 2022).

[Tyshchenko 2020] Andrew Tyshchenko. „Testing Native and Hybrid Mobile Apps. Whats the Difference?“ In: *Medium* (19. Juni 2020). URL: https://medium.com/@andrew.tishchenko/testing-native-and-hybrid-mobile-apps-whats-the-difference-4ca98a71afc1 (besucht am 05. 11. 2022).

[Willnecker et al. 2012] Felix Willnecker, Damir Ismailović und Wolfgang Maison. „Architekturen mobiler Multiplattform-Apps“. In: *Smart mobile apps. Mit Business-Aps ins Zeitalter mobiler Geschäftsprozesse*. Hrsg. von Stephan Verclas und Claudia Linnhoff-Popien. Xpert.press. Dordrecht: Springer, 2012, S. 403–417. ISBN: 978-3-642-22258-0. DOI: 10.1007/978-3-642-22259-7_26.

[Wu 2018] Wenhao Wu. *React Native vs Flutter, Cross-platforms mobile application frameworks*. 2018. URL: https://www.theseus.fi/bitstream/handle/10024/146232/thesis.pdf?sequence=1.

[xster 2017] xster. „Why Flutter doesn't use OEM widgets“. In: *Medium* (16. Nov. 2017). URL: https://medium.com/flutter/why-flutter-doesnt-use-oem-widgets-94746e812510 (besucht am 02. 11. 2022).

[Zammetti 2019] Frank W. Zammetti. *Practical flutter. Improve your mobile development with google's latest Open-Source SDK*. eng. Springer eBook Collection. Zammetti, Frank W. (VerfasserIn). Berkeley, CA: Apress, 2019. 396 S. ISBN: 978-1-4842-4971-0. DOI: 10.1007/978-1-4842-4972-7.