Eruierung von Kriterien und Best Practices zur Entwicklungsentscheidung zwischen mobiler nativer App, Web-App und hybrider App auch unter Berücksichtigung einer generellen Akzeptanz-Entwicklung

Bachelor-Thesis

im Studiengang Medieninformatik

vorgelegt von

Benjamin Mehl

Matrikelnummer: 43405

am 27. Februar 2025

an der Hochschule der Medien Stuttgart

Erstprüfer: Prof. Dr.-Ing. Oliver Kretzschmar

Zweitprüfer: Dr. Eberhard Huber

Bearbeitungszeitraum: 2. Dezember 2024 bis 3. März 2025

Erklärung

Ich, Benjamin Mehl, erkläre eidesstattlich, dass ich die Arbeit selbstständig angefertigt, keine anderen als die angegebenen Hilfsmittel benutzt und alle aus ungedruckten Quellen, gedruckter Literatur oder aus dem Internet im Wortlaut oder im wesentlichen Inhalt übernommenen Formulierungen und Konzepte gemäß den Richtlinien wissenschaftlicher Arbeiten zitiert, durch Fußnoten gekennzeichnet bzw. mit genauer Quellenangabe kenntlich gemacht habe.

Diese Erklärung wurde wortwörtlich aus [27] übernommen.

Stuttgart, 26.02.2025 — Ort, Datum

B. Mehl — Unterschrift

Kurzfassung

Diese Arbeit untersucht die Entscheidungsfindung bei der Entwicklung mobiler Anwendungen. Im Fokus stand die Evaluierung der Vor- und Nachteile verschiedener Entwicklungsansätze, wie nativ, hybrid/cross-platform sowie webbasierten Lösungen unter Berücksichtigung technologischer, wirtschaftlicher und nutzerzentrierter Aspekte. Zur Erfassung dieser Faktoren wurde eine umfassende Literaturrecherche durchgeführt und durch eine quantitative Nutzerstudie mit 40 Teilnehmenden ergänzt, wodurch empirische Einblicke in das Nutzerverhalten und die Präferenzen gewonnen wurden.

Die Analyse ergab, dass native und hybride Apps tendenziell eine tiefere Systemintegration sowie eine höhere Performance und Offline-Fähigkeit bieten, während moderne Web-Technologien diese Lücke zunehmend schließen und durch Progressive Web Apps eine flexiblere Distribution ermöglichen. Wirtschaftlich zeigen sich native Lösungen durch parallele Codebasen kostenintensiver, während Cross-Plattform- und Web-Ansätze effizienter umgesetzt werden können. Zudem verdeutlicht die Nutzerstudie unter anderem, dass viele Anwender*innen Apps nur installieren, wenn sie diese regelmäßig nutzen, während Gelegenheitsanwendungen bevorzugt über mobile Webseiten abgerufen werden.

Auf Grundlage dieser und weiterer Erkenntnisse wurde ein strukturierter Kriterienkatalog entwickelt, der mithilfe gewichteter Fragestellungen eine Einschätzung der Eignung der jeweiligen Entwicklungsstrategie ermöglicht. Ein interaktives Tool unter https://appdecide.netlify.app/, welches diese Methode implementiert, liefert darüber hinaus konkrete Handlungsempfehlungen zur Optimierung der Technologieauswahl. Diese Arbeit liefert somit Entscheidungsträger*innen eine fundierte Grundlage zur Wahl der am besten geeigneten Entwicklungsstrategie, um eine hohe Nutzerakzeptanz und wirtschaftlichen Erfolg mobiler Anwendungen sicherzustellen.

Abstract

This thesis examines the decision-making process in mobile application development. The focus was on evaluating the advantages and disadvantages of the different development approaches of native, hybrid/cross-platform, and web-based solutions while considering technological, economic, and user-centered aspects. To assess these factors, a comprehensive literature review was conducted and supplemented by a quantitative user study with 40 participants, providing empirical insights into user behavior and preferences.

The analysis revealed that native and hybrid apps tend to offer deeper system integration, higher performance, and better offline capabilities, while modern web technologies are closing this gap and enabling more flexible distribution through Progressive Web Apps. From an economic perspective, native solutions are more cost-intensive due to parallel codebases, whereas cross-platform and web-based approaches can be implemented more efficiently. Additionally, the user study highlights that many users only install apps if they use them regularly, while occasional applications are often accessed via mobile websites.

Based on these and other findings, a structured criteria catalog was developed to assess the suitability of different development strategies using weighted questions. An interactive tool available at https://appdecide.netlify.app/ implements this methodology and provides concrete recommendations for optimizing the technology selected. This thesis offers a structured foundation that helps decision-makers select the most suitable development strategy while considering key factors. It serves as a guideline to support informed choices, aiming to maximize user acceptance and economic success for mobile applications.

Inhaltsverzeichnis

KI-Disclaimer

In dieser Bachelor-Arbeit wurde Künstliche Intelligenz in Form von Large Language Models (LLMs) zur sprachlichen Verfeinerung und Verbesserung der Ausdrucksweise benutzt. Fachliche Inhalte, Analysen und Schlussfolgerungen basieren jedoch auf eigenständiger Recherche und wissenschaftlichen Quellen. Es wurde sichergestellt, dass kein inhaltliches Wissen aus dem Kontext der LLMs übernommen, sondern ausschließlich bestehende Argumentationen klarer formuliert und relevante wissenschaftliche Quellen identifiziert wurden. Die entsprechenden Texte, welche mit der Unterstützung von KI verfasst wurden sind klar gekennzeichnet.

Akronyme

APIs Application Programming Interfaces
CSS Cascading Style Sheets
HTML Hypertext Markup Language
IP Internet Protocol
ROI Return on Investment
SDKs Software Development Kits
URL Uniform Resource Locator

Abbildungsverzeichnis

Tabellenverzeichnis

1 Überblick

Im Rahmen der Entwicklung einer mobilen Applikation stellt sich nach der initialen Anforderungsanalyse und der konzeptionellen Gestaltung unweigerlich die Frage nach der optimalen technologischen Umsetzung. Angesichts der Vielzahl an verfügbaren Entwicklungsansätzen – darunter native, hybride und webbasierte Lösungen – ist eine fundierte Entscheidung essenziell, da jede dieser Technologien spezifische Vor- und Nachteile hinsichtlich Leistungsfähigkeit, Entwicklungskosten, Wartungsaufwand und Nutzererfahrung aufweist.

Diese Arbeit verfolgt das Ziel, die technischen Möglichkeiten und Grenzen der verschiedenen Entwicklungsansätze systematisch zu analysieren und sie mit wirtschaftlichen Anforderungen sowie dem Nutzerverhalten in Beziehung zu setzen. Durch diesen interdisziplinären Vergleich sollen relevante Zusammenhänge identifiziert und Best Practices für eine technologiegestützte Entscheidungsfindung herausgearbeitet werden.

Grundsätzlich lassen sich drei Hauptarten technischer Umsetzungen unterscheiden: native Apps, hybride bzw. Cross-Plattform-Apps und Web-Apps. Um ein besseres Verständnis für diese Begriffe zu bekommen, werden sie im Folgenden erklärt.

Eine native App ist eine Anwendung, die speziell für ein bestimmtes Betriebssystem entwickelt wird, beispielsweise für iPhones mit iOS oder Android-Smartphones. Sie ist direkt auf die jeweilige Plattform zugeschnitten, sodass sie dort besonders gut funktioniert. Ein Beispiel dafür wäre WhatsApp, welches direkt über den App Store oder Google Play Store heruntergeladen wird und für das jeweilige Betriebssystem spezifisch entwickelt wurde.

Eine hybride oder Cross-Plattform-App hingegen wird so entwickelt, dass sie auf mehreren Geräten mit unterschiedlichen Betriebssystemen funktioniert, ohne dass für jede Plattform eine komplett eigene Version programmiert werden muss. Das spart Aufwand und Kosten, kann aber in manchen Fällen zu kleinen Einschränkungen führen. Ein Beispiel für eine solche App ist Instagram, das auf verschiedenen Geräten nahezu identisch aussieht und funktioniert.

Die dritte Variante sind Web-Apps. Sie müssen nicht aus einem App-Store heruntergeladen werden, sondern funktionieren direkt über den Webbrowser. Bekannte Beispiele sind YouTube oder Facebook, die man sowohl als herkömmliche App als auch einfach über den Browser nutzen kann. Der Vorteil von Web-Apps ist, dass sie unabhängig vom Gerät funktionieren, allerdings können sie nicht alle Funktionen bieten, die eine App direkt auf dem Smartphone ermöglicht.

1.1 Forschungsfrage und Aufgabenstellung

Die vorliegende Arbeit untersucht die Entscheidungskriterien für die Entwicklung mobiler Anwendungen und erörtert, inwiefern technologische, wirtschaftliche und nutzerbezogene Aspekte die Wahl zwischen nativen, hybriden und webbasierten Lösungen beeinflussen. Die zentrale Forschungsfrage lautet:

"Welche technologischen, wirtschaftlichen und nutzerbezogenen Kriterien sind für die Wahl zwischen nativer App, Web-App und hybrider App entscheidend, und welche Best Practices lassen sich daraus ableiten?"

Zur Beantwortung dieser Frage wurde zunächst eine umfassende Literaturrecherche durchgeführt, um bestehende Erkenntnisse zu technischen Möglichkeiten, wirtschaftlichen Rahmenbedingungen und Nutzerpräferenzen zu analysieren. Ergänzend wurde eine empirische Studie mit 40 Teilnehmenden durchgeführt, um aktuelle Nutzererwartungen und Akzeptanzfaktoren besser zu verstehen. Auf Grundlage dieser Erkenntnisse werden Handlungsempfehlungen in Form einer methodischen Berechnung entwickelt, die anhand der Anforderungen für eine potenzielle App die Wahrscheinlichkeiten der jeweiligen Entwicklungsstrategien ermitteln. Darauf aufbauend wird ein interaktives Tool implementiert, welches Entscheidungsträger*innen bei der Auswahl des optimalen Entwicklungsansatzes unterstützt.

1.2 Struktur der Arbeit

Um ein fundiertes Verständnis der aktuellen technologischen Entwicklungen und ihrer geschäftlichen Relevanz zu gewinnen, wird zunächst die gegenwärtige Marktsituation analysiert. Hierbei werden zentrale Kennzahlen zu mobilen Anwendungen vorgestellt, um die Bedeutung einer fundierten Entwicklungsentscheidung zu verdeutlichen. Anschließend erfolgt eine umfassende Untersuchung der technischen Aspekte verschiedener Entwicklungsansätze, einschließlich ihrer Unterschiede und Einschränkungen. Dabei werden nicht nur technische Limitationen im Endprodukt betrachtet, sondern auch Herausforderungen in der Entwicklung, Distribution und Bereitstellung von Updates analysiert.[1]

Aufbauend auf dieser technischen Grundlage werden wirtschaftliche sowie nutzerbezogene Anforderungen untersucht, um Schnittstellen zwischen technologischen Möglichkeiten und geschäftlichen Bedürfnissen zu identifizieren. Durch eine empirische Nutzerstudie wurden Erkenntnisse zum Nutzerverhalten gewonnen und durch eine Recherche bestehender Fachliteratur ergänzt, um weiterführende Zusammenhänge zu identifizieren und einzuordnen.

[1] Dieser Text wurde mit Unterstützung von KI geschrieben.

Basierend auf den gewonnenen Erkenntnissen werden in Kapitel 5 konkrete Handlungsempfehlungen formuliert, die sich an zuvor erarbeiteten Kriterien orientieren. Abschließend wird die Forschungsfrage auf Grundlage dieser Empfehlungen beantwortet und ein Fazit gezogen.

2 Zielsetzung und Ist-Situation

2.1 Zielsetzung

Das Ziel dieser Arbeit ist die systematische Erarbeitung eines Kriterienkatalogs zur Entscheidungsfindung zwischen nativen Apps, hybriden Apps und Web-Apps. Basierend auf einer umfassenden Analyse technischer, wirtschaftlicher und nutzerzentrierter Aspekte sollen Best Practices abgeleitet werden, die eine fundierte Entscheidungsfindung für Entwickler*innen und Unternehmen ermöglichen. Dabei wird insbesondere untersucht, inwieweit Nutzerakzeptanz und technologische Einschränkungen die Wahl der Entwicklungsstrategie beeinflussen.

2.2 Ist-Situation und Business Case

Die technologische Entwicklung und wirtschaftlichen Trends der letzten Jahre zeigen eine klare Verschiebung hin zu einer stärkeren Nutzung mobiler Geräte und einem steigenden wirtschaftlichen Potenzial im Bereich mobiler Apps. Diese Entwicklungen unterstreichen die zunehmende Bedeutung von mobilen Anwendungen und der Entscheidung für die geeignete Art der Entwicklung.

Laut Statistiken haben mobile Geräte im Jahr 2024 einen Marktanteil bei Internetzugriffen von etwa 63%, während Desktop-Geräte nur noch rund 35% erreichen. Dieser Unterschied ist das Ergebnis eines klaren Trends hin zu vermehrter Smartphone-Nutzung in den letzten Jahren [68]. Die zunehmende ökonomische Bedeutung mobiler Geräte wird durch das starke Wachstum der weltweiten App-Einnahmen deutlich. Diese stiegen von 467 Milliarden US-Dollar im Jahr 2023 auf 522 Milliarden US-Dollar im Jahr 2024 [71].

Auch die Anzahl der App-Downloads zeigt ein signifikantes Wachstum. Im Jahr 2023 wurden weltweit 257 Milliarden Apps heruntergeladen. Ein deutlicher Anstieg gegenüber 204 Milliarden Downloads im Jahr 2019 [70].

Ein weiterer Indikator für die wachsende Bedeutung mobiler Apps ist die gestiegene Bildschirmzeit. In den USA betrug die durchschnittliche tägliche Nutzung mobiler Geräte im Jahr 2022 etwa 4 Stunden und 30 Minuten pro Person. Dies stellt eine deutliche Steigerung gegenüber den 3 Stunden und 45 Minuten im Jahr 2019 dar [72].

Die obigen Entwicklungen zeigen, dass mobile Anwendungen ein essenzieller Bestandteil moderner digitaler Strategien sind. Eine fundierte Entscheidung über die Art der Entwicklung, sei es nativ, hybrid oder als Web-App, wird daher immer wichtiger. Falsche Entscheidungen können nicht nur die Wettbewerbsfähigkeit eines Unternehmens gefährden, sondern auch die zukünftige Skalierbarkeit und Anpassungsfähigkeit der Anwendung stark beeinträchtigen.

Für zahlreiche Unternehmen ist eine mobile App von entscheidender Bedeutung, um ihre Kunden oder Zielgruppen schnell und unkompliziert zu erreichen. Die Entwicklung einer mobilen App erfordert jedoch Entscheidungen zwischen einer Vielzahl unterschiedlicher technischer Ansätze. Da die Wahl der zugrunde liegenden Technologie erhebliche Auswirkungen auf Aspekte wie Skalierbarkeit, Wettbewerbsfähigkeit, Kosten sowie zukünftige Anpassungen haben kann, sollen in den folgenden Kapiteln die relevanten Entscheidungsfaktoren erörtert und Ansätze für eine fundierte Technologieauswahl aufgezeigt werden.[2]

[2] Dieser Text wurde mit Unterstützung von KI geschrieben.

3 Technische Aspekte von nativen, hybriden und Web-Technologien

3.1 Einführung in die Entwicklungsansätze

Im Folgenden werden die Funktionsweisen und die technischen Hintergründe der verschiedenen Ansätze erläutert, um eine fundierte Grundlage für die späteren Abschnitte zu schaffen.

Native Apps

Native Anwendungen werden speziell für ein bestimmtes Betriebssystem (z. B. Android oder iOS) entwickelt. Dies bedeutet, dass sie auf die jeweiligen nativen Application Programming Interfaces (APIs) eines Betriebssystems, wie beispielsweise die Kontrolle über die Lautstärke des Gerätes, zugreifen können. Die Entwicklung erfolgt in Programmiersprachen, die für die jeweilige Plattform vorgesehen sind, wie Java oder Kotlin für Android [24] und Swift oder Objective-C für iOS [49].

Der Entwicklungsprozess nativer Apps beginnt mit der Erstellung des Quellcodes in der entsprechenden Sprache. Im Anschluss wird dieser mithilfe eines Compilers in maschinenlesbaren Code übersetzt und in Form einer .apk (Android) oder .ipa (iOS) Datei gebundelt. Die fertige Anwendung wird dann über App-Stores wie den Google Play Store oder den Apple App Store verbreitet. Dieser Prozess wird in Abbildung 3.1 dargestellt.

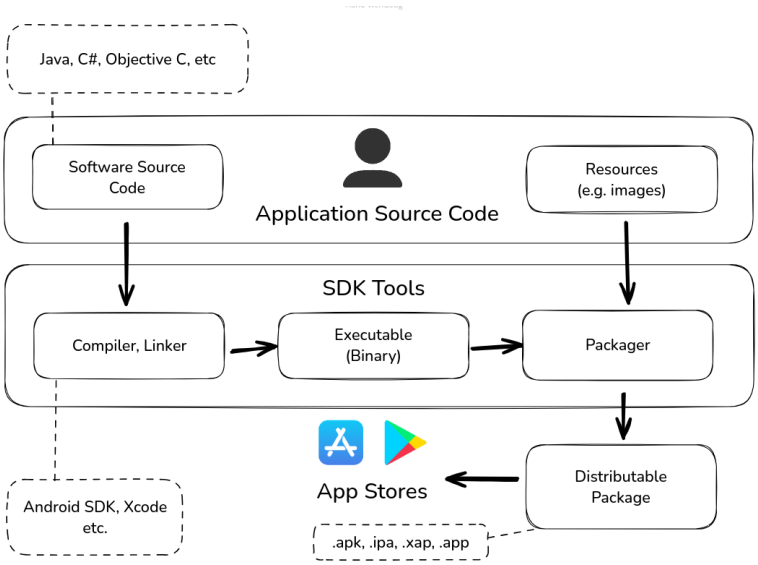

Abbildung 3.1: Darstellung des Entwicklungsprozesses nativer Apps vom Quellcode bis zur Bereitstellung in App-Stores (In Anlehnung an Abbildung aus [52])

Web-Apps

Im Gegensatz zu nativen Apps werden Web-Apps nicht direkt auf dem Gerät installiert, sondern über einen Browser, wie Google Chrome, Safari oder Mozilla Firefox, ausgeführt. Die Anwendungen basieren auf Webtechnologien wie HTML, CSS und JavaScript und sind plattformunabhängig, da sie lediglich einen modernen Webbrowser erfordern [42]. Web-Apps werden typischerweise auf einem Server gehostet und beim Aufruf durch Nutzer*innen geladen.

Der Entwicklungsprozess von Web-Apps ist weniger komplex in Bezug auf die Verteilung: Nach Fertigstellung des Frontends, welches die dargestellte Seite an sich beinhaltet, wird die Anwendung über einen Webserver bereitgestellt und ist sofort über eine Uniform Resource Locator (URL) oder IP-Adresse zugänglich [35]. Anders als native Apps durchlaufen Web-Apps keine App-Store-Prüfung und müssen nicht explizit installiert werden. Die Architektur von Web-Apps ist in Abbildung 3.2 dargestellt.

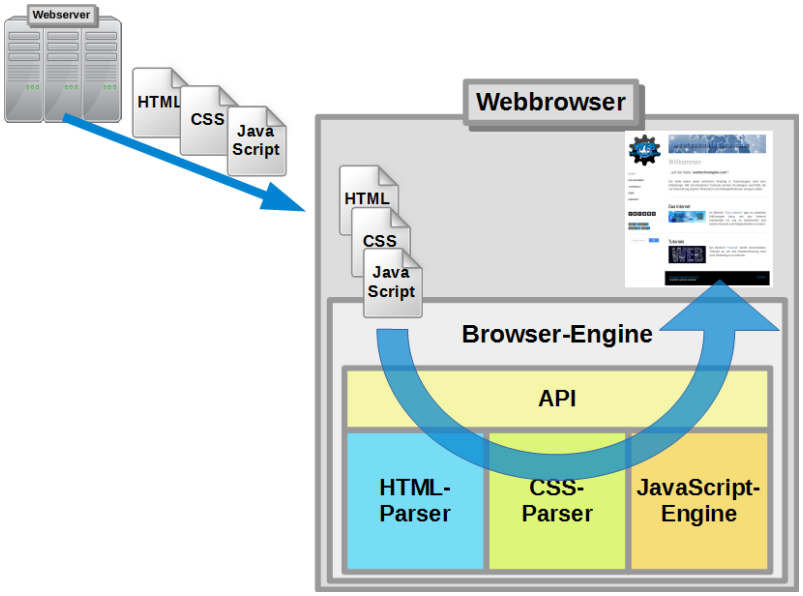

Abbildung 3.2: Ablauf der Bereitstellung von Web-Apps (Abbildung aus [43])

Hybride Apps

Hybride Anwendungen stellen keinen einheitlichen Entwicklungsansatz dar, sondern umfassen verschiedene Frameworks, deren Hauptziel es ist, die Entwicklung von Anwendungen für mehrere Plattformen gleichzeitig zu ermöglichen. Die Umsetzungsansätze dieser Frameworks unterscheiden sich jedoch erheblich und können sowohl die technischen Eigenschaften als auch die Leistungsfähigkeit der resultierenden Anwendungen beeinflussen.

Um den Rahmen dieser Arbeit nicht zu überschreiten, konzentriere ich mich auf die drei am häufigsten verwendeten Technologien und Frameworks für hybride Entwicklung. Diese werden im Rahmen der Evaluation umfassend analysiert und in den Vergleich der Entwicklungsansätze eingebunden. Laut aktuellen Statistiken aus dem Jahr 2023 gehören Flutter, React Native und Cordova zu den am häufigsten verwendeten Frameworks für die Entwicklung hybrider und cross-plattform Anwendungen. Dabei weisen Flutter und React Native einen signifikant höheren Nutzungsgrad auf, wobei der Abstand zu Cordova deutlich erkennbar ist. [78]

Das Framework Flutter verwendet Dart als Programmiersprache und zeichnet sich durch die Nutzung einer eigenen Grafikengine zur Darstellung der Benutzeroberfläche aus. Es verwendet eine eigene Rendering-Engine namens Impeller für Android und iOS und Skia für die restlichen Plattformen, die direkt mit dem Canvas des Betriebssystems interagiert [30]. Dies ermöglicht Flutter, die gesamte Benutzeroberfläche selbst zu zeichnen, ohne auf native UI-Komponenten zurückzugreifen. Diese Architektur ermöglicht eine plattformunabhängige und konsistente Darstellung, da die Benutzeroberfläche direkt gerendert wird und nicht auf nativen Komponenten basiert [28].

React Native basiert hingegen auf JavaScript und dem React-Framework, welches aus der Web-Entwicklung stammt. Die zentrale Komponente war ehemals (bis Oktober 2024) eine sogenannte “Bridge” [75], die es erlaubt, auf native UI-Komponenten der jeweiligen Zielplattform zuzugreifen. Diese Architektur sorgt dafür, dass React Native-Anwendungen ein nahezu identisches Look-and-Feel wie native Apps aufweisen, da sie direkt auf die nativen Elemente des Betriebssystems zugreifen. Dabei werden kritische Anwendungsbereiche wie Benutzeroberfläche und Performance weitgehend auf nativer Ebene ausgeführt, während die Geschäftslogik weiterhin in JavaScript verbleibt (siehe Abbildung 3.3). React Native wird von Meta (ehemals Facebook) aktiv entwickelt und ist aufgrund seiner umfangreichen Community und der breiten Palette an Drittanbieter-Bibliotheken eine der führenden Lösungen für hybride App-Entwicklung [28, 67].

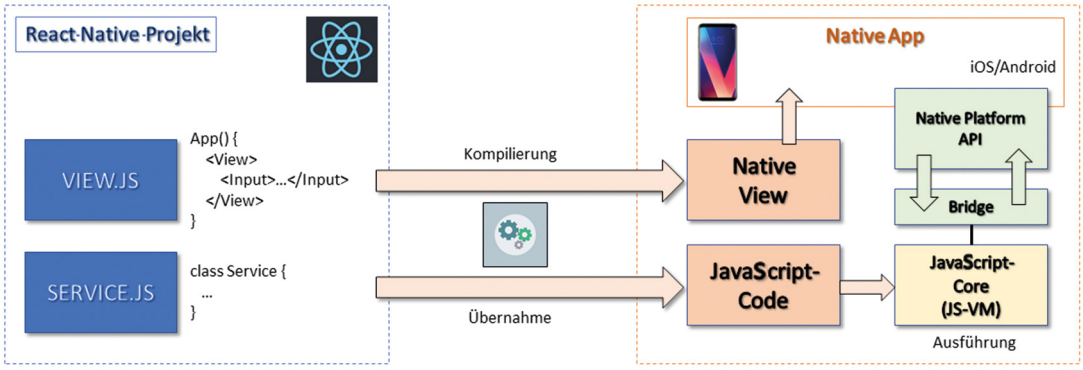

Abbildung 3.3: vereinfachte Darstellung der Arbeitsweise von React Native (Abbildung aus [26])

Apache Cordova ist ein Open-Source-Framework, welches die Erstellung plattformübergreifender Anwendungen durch den Einsatz von Webtechnologien wie HTML, CSS und JavaScript ermöglicht. Dadurch können Entwickler*innen bisherige Webanwendungen, welche aus statischen HTML-, CSS- und JavaScript-Dateien bestehen, relativ einfach in mobile Anwendungen umwandeln [28]. Beim Cordova-Framework wird die Webanwendung in einen plattformspezifischen nativen Container integriert, der als Schnittstelle zwischen der Anwendung und der jeweiligen mobilen Plattform fungiert. Cordova setzt hierbei eine WebView-Komponente ein, um die Webanwendung innerhalb der nativen Umgebung darzustellen, wodurch die App in der Lage ist, auf verschiedene Betriebssysteme zuzugreifen, ohne für jede Plattform gesondert entwickelt werden zu müssen [13].

In den folgenden Abschnitten werden die Unterschiede in der Entwicklung, den Endprodukten, den Distributionswegen und der Sicherheit detailliert beschrieben, um ein umfassendes Verständnis der Vor- und Nachteile sowie der Hintergründe der verschiedenen Entwicklungsansätze zu vermitteln.

3.2 Unterschiede im Endprodukt

Die Unterschiede zwischen nativen, hybriden und Web-Apps manifestieren sich vor allem im Endprodukt, das letztlich Nutzer*innen erreicht. Während sich alle drei Entwicklungsansätze darauf konzentrieren, funktionale und benutzerfreundliche Anwendungen zu erstellen, beeinflusst die gewählte Technologie die Performance und den Zugriff auf gerätespezifische Funktionen sowie die Nutzererfahrung. In diesem Abschnitt werden die zentralen Differenzen hinsichtlich Performance, Funktionalität und der Möglichkeit zur Nutzung nativer Schnittstellen analysiert. Ziel ist es, die jeweiligen Stärken und Schwächen der verschiedenen Entwicklungsansätze herauszuarbeiten und damit ein umfassendes Verständnis für die Auswirkungen auf das Endprodukt zu schaffen.[3]

Der erste Schwerpunkt liegt auf dem Vergleich zwischen nativen Apps und Webanwendungen, da sich dabei die Unterschiede in Bezug auf Performance, Zugriffsmöglichkeiten und Benutzererfahrung am deutlichsten erkennen lassen. Ein deutlicher Unterschied zwischen mobilen Webanwendungen und nativen Apps liegt in der Art und Weise, wie sie geöffnet und genutzt werden. Webanwendungen werden meist über einen Browser oder eine Suchmaschine aufgerufen, indem eine URL durch das Anklicken eines Links oder Lesezeichens sowie das direkte Eingeben dieser, geöffnet wird. Im Gegensatz dazu erfordern native Apps eine vorherige Installation über einen App-Store, bevor sie direkt vom Gerät aus geöffnet werden können. Dieser Unterschied führt dazu, dass Webanwendungen insbesondere beim ersten Zugriff häufig schneller verfügbar sind, während native Apps nach der Installation einen schnelleren und direkteren Zugriff ohne Zwischenschritte ermöglichen.

Gerätespezifische Schnittstellen

Ein wesentlicher Unterschied zwischen nativen Anwendungen und Webanwendungen zeigt sich auch in der Integration und Nutzung gerätespezifischer Funktionen. Native Apps werden gezielt für ein bestimmtes Betriebssystem und dessen Plattform entwickelt, wodurch sie umfassenden Zugriff auf eine Vielzahl von Funktionen und Sensoren des Geräts erhalten. Dies erfolgt über sogenannte APIs, die es ermöglichen, unter anderem auf Hardwareressourcen wie Kamera, Mikrofon, Fingerabdrucksensoren oder auch die Kontaktliste zuzugreifen. Diese enge Verbindung mit der Plattform erlaubt es, eine optimale Performance sowie eine bessere Benutzererfahrung zu gewährleisten [63].

Moderne Webbrowser wie Google Chrome, Safari und Mozilla Firefox haben in den letzten Jahren jedoch signifikante Fortschritte gemacht und bieten mittlerweile ebenfalls eine Vielzahl von APIs an, die auf native Gerätefunktionen zugreifen können. [39] Aktuelle Beispiele

[3] Dieser Text wurde mit Unterstützung von KI geschrieben.

hierfür sind die Web Authentication API [40], die es ermöglicht, sich mit biometrischen Daten oder anderen Authentifizierungsmethoden ohne Passwort anzumelden, sowie die MediaDevice API [36], die Eingabegeräte wie Mikrofone und Kameras unterstützt. Trotz dieser Entwicklungen bleibt der Zugriff auf einige tiefergehende Funktionen und Sensoren häufig noch nativen Anwendungen vorbehalten [10], weshalb sie in spezifischen Anwendungsfällen weiterhin bevorzugt werden. Es werden jedoch kontinuierlich neue Web APIs entwickelt, die Webanwendungen attraktiver und funktional vielseitiger gestalten könnten. Einige dieser neuen APIs, wie die Unterstützung für Bluetooth im Browser [41], befinden sich bereits in der Entwicklungs- und frühen Implementierungsphase. Dennoch werden sie derzeit nicht von allen Browsern vollständig unterstützt, was die Nutzbarkeit in der Breite noch einschränkt.

Bei hybriden und Cross-Plattform-Frameworks sind die Möglichkeiten zur Nutzung gerätespezifischer Funktionen häufig auf externe Community-Plugins angewiesen [65, 34, 76]. Dies bedeutet, dass Gerätefunktionen wie Geolocation (GPS) oder NFC-Scanning nicht standardmäßig durch das Framework selbst bereitgestellt werden [66, 33, 15]. Stattdessen müssen Entwickler*innen auf von der Community entwickelte Plugins zurückgreifen, die sich in das Framework integrieren lassen. Während diese Plugins oft eine breite Palette an Funktionalitäten abdecken, kann ihre Qualität und Stabilität variieren, was zu Einschränkungen in der Leistungsfähigkeit und Konsistenz sowie Entwicklung der Anwendungen führen kann. Dennoch bieten diese Plugins eine Lösung zur Erweiterung hybrider Apps und tragen dazu bei, Lücken in der nativen Funktionalität zu schließen.

Um gerätespezifische Funktionen zu nutzen, die nicht durch Community-Plugins abgedeckt werden, ist es erforderlich, plattformspezifischen nativen Code zu entwickeln, welcher über eine Schnittstelle mit dem restlichen Cross-Plattform Code erreichbar ist. Dies bedeutet, dass für jede unterstützte Plattform meist individuelle Implementierungen notwendig sind, um die gewünschten Funktionalitäten bereitzustellen [32]. Mit steigender Anzahl der Zielplattformen wächst der Entwicklungsaufwand deutlich, was die Komplexität und den zeitlichen Aufwand der Anwendungsentwicklung stark erhöhen kann. Dieser zusätzliche Aufwand kann die Wartung und Skalierbarkeit der Anwendung erschweren, insbesondere wenn sich die nativen Schnittstellen der Plattformen weiterentwickeln oder verändern.

Verlinkung mit URLs

Nicht nur Webanwendungen übernehmen zunehmend Funktionen und Merkmale nativer Apps, sondern auch der umgekehrte Fall lässt sich beobachten. Ein Beispiel hierfür ist die einfache Teilbarkeit von Webanwendungen durch URLs (Uniform Resource Locators). Dies erlaubt nicht nur die Verlinkung der Startseite einer Anwendung, sondern auch die

direkte Navigation zu spezifischen Unterseiten oder Abschnitten innerhalb der Anwendung. [11] Dies erfolgt beispielsweise durch die Verwendung von Anker-IDs in der URL, wie zum Beispiel “http://example.com/<id>”.

Webanwendungen bieten durch die Möglichkeit der Konfiguration und Personalisierung über URL-Parameter eine erhebliche Flexibilität. Informationen und Filter können direkt in die URL eingebettet und gezielt weitergegeben werden, was eine präzise Navigation erlaubt. Ein Beispiel hierfür wäre die Weitergabe einer URL wie beispielsweise “http://example.com/?filter=xxx” [11]. Diese direkte Anpassbarkeit verbessert die Benutzerfreundlichkeit erheblich, da personalisierte Inhalte schnell und unkompliziert übermittelt werden können.

Native Apps bieten inzwischen ähnliche Möglichkeiten zur Datenweitergabe, jedoch erfordert die Implementierung dieser Funktionalität explizite Entwicklungsschritte. Während Webanwendungen diese Flexibilität standardmäßig unterstützen, müssen Entwickler*innen bei nativen Anwendungen spezielle Metadaten definieren, um ähnliche Ergebnisse zu erzielen [23]. Daraus ergibt sich für Webanwendungen ein klarer Vorteil hinsichtlich Flexibilität und Zugänglichkeit.

Performance, Arbeitsspeicherbedarf, Energieverbrauch

Neben den sichtbaren Funktionen und Features spielen auch technische Aspekte wie Performance, geringer Arbeitsspeicherverbrauch und niedriger Energiebedarf eine entscheidende Rolle für die optimale Nutzung einer App. Ein aussagekräftiger Vergleich erfordert die Analyse identischer Anwendungen auf unterschiedlichen Plattformen, beispielsweise Spotify als native Android-App im Vergleich zur Spotify-Web-App.

In der Regel weisen native Anwendungen eine geringere CPU-Auslastung auf und schneiden hinsichtlich der Performance besser ab als ihre webbasierten Pendants. [48] Dies liegt unter anderem daran, dass native Apps direkt auf die Hardware des Geräts zugreifen können, während Web-Apps innerhalb einer Browser-Umgebung operieren und dadurch zusätzliche Verarbeitungsschichten durchlaufen müssen.

Allerdings zeigen Studien und Benchmarks, wie beispielsweise Untersuchungen aus dem Jahr 2018 auch, dass in 31 Prozent der Fälle, insbesondere bei großen Plattformen wie Google, Facebook und Amazon, Web-Apps hinsichtlich der Performance, gemessen in Antwortzeit, Datenverbrauch und Energieverbrauch, ihre nativen Gegenstücke übertreffen [53]. Beim Arbeitsspeicherverbrauch ergibt sich ein unterschiedliches Bild: Native Anwendungen tendieren dazu, weniger RAM (Random-Access Memory) zu beanspruchen als Web-Apps, da sie effizienter auf gerätespezifische Ressourcen zugreifen können. Dieser

geringere Ressourcenverbrauch spiegelt sich auch im Energiebedarf wider. Native Apps verbrauchen insgesamt weniger Arbeitsspeicher und CPU (Central Processing Unit), was zu einer signifikant längeren Akkulaufzeit der Geräte führen kann [48].

Somit können native Anwendungen vor allem in ressourcenintensiven und performancekritischen Szenarien empfohlen werden, während Web-Apps in anderen Bereichen, wie der plattformübergreifenden Verfügbarkeit und schnelleren Bereitstellung der Anwendung, ihre Vorteile sichtbar sind.

Bei den Cross-Plattform Frameworks gibt es auch Unterschiede zwischen Performance und Ressourcennutzung. Im Vergleich zwischen React Native und Flutter zum Beispiel benutzt React Native auf der Plattform iOS deutlich mehr Arbeitsspeicher bei vielen Funktionen wie Bilder-Animationen [12]. Auch in der Nutzlast der CPU braucht React Native auf Android und iOS meist signifikant mehr Rechenleistung als Flutter.
Da Cordova, wie zuvor erläutert, eine Web View zur Darstellung der Anwendung verwendet, entspricht die Performance weitgehend der einer klassischen Webanwendung. [13] Allerdings können durch die Integration von Plugins, die in nativem Code verfasst sind, gezielt Performance-Engpässe überwunden werden.

Offline Fähigkeiten und Speicherverbrauch

Offline-Funktionalitäten sind entscheidend für die Nutzbarkeit und Zuverlässigkeit von Anwendungen, insbesondere in Regionen mit instabiler oder eingeschränkter Internetverbindung. Native und hybride Apps bieten häufig umfassende Offline-Funktionen, da sie in der Lage sind, Daten lokal auf dem Gerät zu speichern und unabhängig von einer aktiven Internetverbindung zu arbeiten. Dieser Vorteil ergibt sich aus der tiefen Integration nativer und hybrider Anwendungen in das jeweilige Betriebssystem, wodurch sie direkten Zugriff auf lokale Speichermedien und Ressourcen erhalten.

Webanwendungen hingegen laden traditionell bei jedem Zugriff die erforderlichen Ressourcen aus dem Internet, was ihre Funktionsfähigkeit in Offline-Szenarien erheblich einschränkt. Ein bedeutender Fortschritt in diesem Bereich sind Progressive Web Apps (PWAs) [38], die durch den Einsatz von Service-Workern gezielt Inhalte cachen und so eine eingeschränkte Offline-Nutzung ermöglichen. PWAs können auf dem Gerät installiert und ähnlich wie native Apps verwendet werden, was die Lücke zwischen Web- und nativen Anwendungen weiter schließt. Die Implementierung von Offline-Funktionen in PWAs erfordert jedoch explizite Entwicklungsmaßnahmen, insbesondere durch die Programmierung von Service-Workern in JavaScript und einer Manifest-Datei für die Definition von Meta-Daten für die PWA [37]. Daher ist diese Funktionalität nicht automatisch bei jeder Web-Anwendung vorhanden. Jedoch rüsten zunehmend große Plattformen wie Spotify, Tinder, Pinterest und Google Maps [55] ihre Webanwendungen mit umfangreichen Offline-

Funktionen und einer PWA aus, um die Benutzerfreundlichkeit auch ohne Internetzugang sicherzustellen.

Der Speicherbedarf und die Größe von Anwendungen variieren je nach Entwicklungsansatz erheblich. Native Apps sind in der Regel optimiert und auf die jeweilige Plattform zugeschnitten, was zu geringerem Speicherbedarf führt. Dies liegt daran, dass native Anwendungen direkt in der jeweiligen Systemsprache geschrieben werden und keine zusätzlichen Abstraktionsschichten oder Laufzeitumgebungen benötigen.

Im Gegensatz dazu weisen hybride und Cross-Plattform-Frameworks wie Flutter, React Native oder Cordova aufgrund ihrer Architektur häufig einen höheren Speicherbedarf auf. Frameworks wie Flutter verwenden beispielsweise eine eigene Rendering-Engine, die zusätzliche Ressourcen beansprucht, was die initiale Größe der App im Vergleich zu nativen Anwendungen ansteigen lässt [25]. Solche Unterschiede in der Speichergröße könnten insbesondere für Anwendungen von Bedeutung sein, bei denen die verfügbare Speicherkapazität der Endgeräte begrenzt ist oder die Größe der App direkten Einfluss auf die Nutzerakzeptanz und die Download-Raten hat. Im Abschnitt zur Nutzerakzeptanz und -verhalten wird dieser Aspekt weiter vertieft.

3.3 Unterschiede in der Entwicklung

Die Programmierung und Entwicklung stellen zentrale Aspekte bei der Erstellung von Anwendungen dar und bilden die Grundlage für die spätere Funktionalität und Nutzererfahrung. In diesem Abschnitt wird detailliert untersucht, wie sich die Entwicklungsansätze für native Apps, Web-Apps und Cross-Plattform-Apps unterscheiden. Dabei wird auf die spezifischen Herausforderungen und Vorteile eingegangen, die sich aus der Wahl der jeweiligen Technologie ergeben. Ziel ist es, ein fundiertes Verständnis für die Entwicklungsprozesse zu schaffen und aufzuzeigen, welche Faktoren bei der Entscheidungsfindung für die geeignete Entwicklungsstrategie berücksichtigt werden sollten.[4]

Entwicklungsumgebungen

Die Entwicklungsumgebung beschreibt die Gesamtheit der Software- und Hardwarevoraussetzungen, die für die Entwicklung einer Anwendung auf dem Rechner der Entwickler*innen erforderlich sind. Wesentliche Bestandteile sind hierbei integrierte Entwicklungsumgebungen (IDEs), welche bei der Erstellung, Verwaltung und Fehlerbehebung von Quellcode unterstützen. Die Wahl und Konfiguration der Entwicklungsumgebung variieren je nach Art der Anwendung und können die Effizienz der Entwicklung und die Qualität der resultierenden Anwendung beeinflussen.

[4] Dieser Text wurde mit Unterstützung von KI geschrieben.

Für die Entwicklung von Web-Anwendungen gestaltet sich die Auswahl der Entwicklungsumgebung vergleichsweise flexibel. In der Regel genügt ein Code-Editor mit Syntax-Highlighting und Unterstützung für HTML, JavaScript und CSS. Zusätzlich können Erweiterungen und Plugins für Frameworks wie React oder Angular integriert werden, um die Produktivität zu steigern. Beispiele für populäre Editoren sind Visual Studio Code [54] und Sublime Text [74].

Im Gegensatz dazu sind die Anforderungen an Entwicklungsumgebungen für native und hybride Anwendungen spezifischer. Für die Entwicklung nativer Android-Apps wird in der Regel Android Studio [44] verwendet, während für die Entwicklung nativer iOS-Anwendungen Xcode [22] erforderlich ist. Diese Entwicklungsumgebungen beinhalten nicht nur Editoren, sondern auch umfangreiche Tools zur Projektverwaltung, Debugging und Performance-Analyse. Darüber hinaus sind sie eng mit den entsprechenden Software Development Kits (SDKs) verknüpft, die für das Kompilieren und Ausführen der Anwendungen unerlässlich sind. Dies kann die Entwicklung einerseits erleichtern, da umfassende Unterstützung bereitgestellt wird, andererseits jedoch auch die Flexibilität einschränken und die Einstiegshürde erhöhen.

Ein signifikanter Unterschied besteht darin, dass Xcode ausschließlich auf Apple-Computern im Mac-App-Store, wie MacBooks oder iMacs, installiert werden kann, was potenziell zusätzliche Hardwarekosten verursacht. Android Studio hingegen ist plattformübergreifend verfügbar und kann auf Windows, macOS und Linux betrieben werden. Beide Entwicklungsumgebungen bieten Emulatoren, die es Entwickler*innen ermöglichen, die Anwendung auf virtuellen Geräten zu testen, ohne physische Hardware anschließen zu müssen. Diese Emulatoren simulieren verschiedene Gerätekonfigurationen und Betriebssystemversionen, wodurch eine Testabdeckung während des Entwicklungsprozesses gewährleistet wird.

Bei der Entwicklung mit Cross-Plattform-Frameworks ist die Nutzung von Emulatoren und SDKs ebenfalls erforderlich, um Anwendungen unter realitätsnahen Bedingungen auf Plattformen wie Android und iOS zu testen. Diese Tools ermöglichen es Entwickler*innen, die plattformspezifischen Verhaltensweisen zu simulieren und somit eine konsistente Nutzererfahrung sicherzustellen. Besonders wichtig ist dies, da Cross-Plattform-Anwendungen potenziell unterschiedliche Implementierungen für jede Plattform aufweisen können, wodurch eine umfassende Teststrategie unerlässlich wird.

Entwicklungseinstieg

Für Entwickler*innen ohne Vorerfahrung in der Entwicklung mobiler Anwendungen kann die Lernkurve der verschiedenen Frameworks und Entwicklungsansätze eine zentrale Rolle spielen, da die Einarbeitungszeit maßgeblich die fristgerechte Fertigstellung eines Projekts beeinflussen kann. Besonders bei Cross-Plattform-Frameworks wie React Native und Flutter gibt es Unterschiede in der Einstiegshürde. React Native gilt im Vergleich als besonders zugänglich für Entwickler*innen mit Vorerfahrung in der Webentwicklung [9], insbesondere durch die Verwandtschaft mit dem weit verbreiteten React-Framework. Eine Umfrage von Stack Overflow aus dem Jahr 2024, die auf 48.503 Antworten basiert, zeigt, dass React mit einem Nutzungsgrad von 39,5 Prozent zu den meistgenutzten Web-Technologien gehört [57]. Diese Vertrautheit mit React kann den Übergang zu React Native erheblich erleichtern und den Lernaufwand minimieren, was zur Effizienz der Entwicklung beiträgt.

Der Einstieg in die Entwicklung mit Flutter kann für Entwickler*innen, die bereits Erfahrung mit React haben, langsamer verlaufen, da Flutter auf der Programmiersprache Dart basiert, die zunächst erlernt werden muss. Dennoch bietet Flutter zahlreiche Ressourcen für Einsteiger, darunter umfassende Dokumentationen, Tutorials und Beispielprojekte, die den Lernprozess strukturieren und erleichtern. Für Entwickler*innen, die keine Vorkenntnisse in jeglicher Entwicklung besitzen, kann es von Vorteil sein, dass Flutter klare Empfehlungen zur Einrichtung der Entwicklungsumgebung gibt [31]. Dies reduziert den initialen Aufwand zur Auswahl geeigneter Tools und beschleunigt die ersten Schritte im Entwicklungsprozess. Die direkte Integration von Entwicklungswerkzeugen in die offiziellen Dokumentationen gewährleistet eine einheitliche und optimierte Entwicklungsumgebung, was besonders für Anfänger eine erhebliche Erleichterung darstellt.

Bei der Entwicklung von Web-Anwendungen und mit Cordova sind vergleichbare Lernkurven zu erwarten, da beide Ansätze auf etablierten Web-Technologien basieren. Web-Anwendungen profitieren von einer Vielzahl an Frameworks, darunter jQuery [51], React [60] und Angular [5], die das Erstellen und Verwalten von Anwendungen erleichtern sollen. Jedes dieser Frameworks weist spezifische Stärken und Anwendungsgebiete auf, was den Auswahlprozess für Entwickler*innen jedoch erschweren kann.

React hat sich als eines der am weitesten verbreiteten Frameworks etabliert und bietet durch seine große Community und umfangreiche Dokumentationen zahlreiche Ressourcen für Entwickler*innen. Diese Verfügbarkeit von Lernmaterialien und Drittanbieter-Bibliotheken kann den Einstieg in die Entwicklung erheblich erleichtern. Allerdings verfolgt React einen unopinionated-Ansatz [60], der keine festen Best Practices vorgibt. Dies erfordert von Entwickler*innen, insbesondere bei komplexeren Projekten, zusätzliche Entscheidungen hinsichtlich der Auswahl und Integration von State-Management-Lösungen.

Während dies Flexibilität und Anpassungsfähigkeit fördert, kann es zugleich die Einarbeitungszeit verlängern und die Komplexität der Architektur von der Applikation erhöhen.

Der Einstieg in die native Entwicklung für iOS und Android setzt spezifische Kenntnisse der jeweiligen Programmiersprachen und Entwicklungsumgebungen voraus [4, 21]. Die Einarbeitung in diese Programmiersprachen kann insbesondere für Entwickler*innen, die primär aus der Webentwicklung kommen, eine Herausforderung darstellen, da sich die Konzepte, Architektur und Tooling deutlich von denen webbasierter Frameworks unterscheiden. Ein wesentlicher Bestandteil des nativen Entwicklungsprozesses ist der Umgang mit den offiziellen integrierten Entwicklungsumgebungen. Die Plattformen bieten eine Vielzahl von Funktionen zur effizienten Fehlerbehebung und Gestaltung von Benutzeroberflächen, die gezielt auf die spezifischen Anforderungen der jeweiligen Umgebung zugeschnitten sind. Dadurch können Entwickler*innen unmittelbar mit der Arbeit beginnen, ohne umfangreiche Konfigurationen selbst vornehmen zu müssen.

Apple und Google stellen umfangreiche Dokumentationen, Tutorials und Entwickler*innen-Communities zur Verfügung, die den Lernprozess unterstützen [17, 2]. Dennoch kann die Vielzahl an APIs und plattformspezifischen Anforderungen zu Beginn überwältigend wirken, was die Einarbeitungszeit verlängert.

3.4 Unterschiede bei Update- und Distributionsprozessen

Wie weiter oben im Text erläutert, unterscheiden sich Web-Anwendungen grundlegend von mobilen Apps in dem Punkt, dass mobile Apps aus einem App Store wie Google Play oder dem Apple Store heruntergeladen werden. Für Entwickler*innen und Produzenten bedeutet dies eine Abhängigkeit vom App Store. Die App muss den Richtlinien [18] entsprechen und wird geprüft, was initial Zeit in Anspruch nehmen kann. Zudem fallen Kosten für ein Konto zur Veröffentlichung an – im Apple Store sind dies jährlich 99 US-Dollar [19], während im Google Play Store eine einmalige Gebühr von 25 US-Dollar [3] anfällt. Updates können durch das Hochladen einer neuen Version in den App Store bereitgestellt werden. Nutzer*innen können ihre App manuell oder automatisch aktualisieren. Allerdings können ältere Versionen weiterhin genutzt werden, was eine Versionierung und das Pflegen älterer Versionen der zugehörigen Backend-Schnittstellen erforderlich machen kann.

Bei Web-Anwendungen erfolgt die Distribution über Server, die für Nutzer*innen über das Internet zugänglich sind. Diese Server können sowohl lokal betrieben werden als auch in großangelegten Rechenzentren von Hosting-Anbietern sein. Die Wahl der Infrastruktur hängt von der Skalierung der Anwendung und den Anforderungen an Verfügbarkeit und Leistung ab. Mit steigender Rechenleistung und zunehmendem Speicherbedarf wachsen in der Regel auch die damit verbundenen Kosten [50]. Im Gegensatz zu nativen Anwendungen ist die Verteilung von Web-Anwendungen nicht an zentrale Plattformen wie App Stores

gebunden. Dies ermöglicht es Entwickler*innen, Änderungen und Updates ohne externe Prüfprozesse direkt zu veröffentlichen. Mehr dazu später in den wirtschaftlichen Aspekten. Ein entscheidender Vorteil von Web-Anwendungen liegt in der Aktualisierung der Anwendung. Bei jedem erneuten Zugriff auf die Web-Applikation wird den Nutzer*innen automatisch die aktuellste Version bereitgestellt. Dies reduziert die Notwendigkeit für manuelle Updates und minimiert Kompatibilitätsprobleme, da ältere Versionen auf den Geräten der Nutzer in der Regel nicht persistiert werden. Die Fähigkeit, kontinuierliche und unmittelbare Updates bereitzustellen, trägt hierbei zur Effizienz der Wartung und Weiterentwicklung bei.

4 Markt- und Nutzer-Anforderungen

4.1 Nutzerakzeptanz und -verhalten

Im folgenden Abschnitt wird die Nutzerakzeptanz als entscheidender Faktor für den Erfolg von Anwendungen analysiert. Dabei wird untersucht, wie Benutzer*innen auf verschiedene Entwicklungsansätze reagieren und welche Faktoren ihre Nutzung und Zufriedenheit beeinflussen. Aspekte wie Usability, User Experience und Vertrauen spielen hierbei eine zentrale Rolle und werden anhand aktueller Studien und Umfragen näher betrachtet. Ziel ist es, Einblicke in das Verhalten und die Präferenzen der Nutzer*innen zu gewinnen, um die Wahl der geeigneten Entwicklungsstrategie fundiert zu unterstützen.[5]

4.1.1 Studienkonzeption

Zur Erhebung eines fundierten Verständnisses der Nutzerakzeptanz wurde im Rahmen dieser Bachelorarbeit eine Umfrage mit 11 gezielten Fragen entwickelt und durchgeführt. Insgesamt nahmen 40 Personen aus verschiedenen Altersgruppen und mit unterschiedlichen beruflichen Hintergründen an der Befragung teil, um eine möglichst heterogene Datengrundlage zu gewährleisten. Die konzipierte und durchgeführte Umfrage sowie die entsprechenden Ergebnisse sind im Anhang A dokumentiert.

Vor der finalen Durchführung der Umfrage wurde ein Test der Umfrage mit einer Gruppe von fünf Personen durchgeführt, um potenzielle Missverständnisse oder Unklarheiten im Fragebogen zu erkennen. Das Feedback aus diesem Testlauf diente als Grundlage für die Präzisierung und Vereinfachung der Formulierungen. Ergänzend wurden Bilder und Screenshots integriert, um den Befragten zusätzlichen Kontext zu bieten und die Verständlichkeit der Fragestellungen zu erhöhen. Dieses Vorgehen trug wesentlich zur Optimierung und Verständlichkeit der Umfrage bei und gewährleistete eine höhere Validität der Ergebnisse.

4.1.2 Analyse der Studienergebnisse

In den folgenden Abschnitten wird auf verschiedene Aspekte der Studienergebnisse eingegangen und diese hinsichtlich der Entscheidung zwischen den Entwicklungsstrategien interpretiert und bewertet.

Anwendungsfälle von mobilen Webseiten

Zur Ermittlung der bevorzugten Anwendungsarten für verschiedene mobile Aktivitäten wurde die Frage formuliert: “Welche Art von mobilen Aktivitäten nutzen Sie bevorzugt als mobile Website anstelle einer App?”. Ziel dieser Fragestellung war es, Erkenntnisse

[5] Dieser Text wurde mit Unterstützung von KI geschrieben.

darüber zu gewinnen, in welchen Bereichen mobile Websites möglicherweise gegenüber nativen Apps bevorzugt werden. Die Ergebnisse zeigen, dass 66,7 Prozent der Befragten Shopping- und E-Commerce-Aktivitäten bevorzugt über mobile Websites abwickeln. Darüber hinaus gaben 52,8 Prozent der Teilnehmenden an, Wetterinformationen lieber über eine mobile Website als über eine App zu beziehen.

Das hohe Ergebnis bei Shopping- und E-Commerce-Aktivitäten lässt sich möglicherweise dadurch erklären, dass Nutzer*innen plattformunabhängig und ohne die Bindung an eine Installation bequem auf Online-Shops über das Web zugreifen können. Insbesondere für Personen, die nur selten in einem bestimmten Online-Shop einkaufen, stellt dies eine geringere Hürde dar, da der Aufwand zur Installation einer App entfällt. Diese Verringerung von Hürden ist für Online-Shops entscheidend, da sie eine geringere Absprungrate ermöglicht und damit eine höhere Conversion-Rate fördern kann. Die Annahme, dass die Installation einer App für Benutzer*innen eine Hürde im Nutzungserlebnis darstellt, wird durch die im nächsten Abschnitt dargestellten Befragungsergebnisse weiter gestützt.

Hürden bei der App-Installation

Um ein besseres Verständnis der Gründe zu erlangen, die Nutzer*innen dazu veranlassen, auf die Installation einer App zu verzichten, wurde die Frage gestellt: “Welche dieser Gründe sind für Sie die zwei wichtigsten, wenn Sie sich gegen die Installation einer App entscheiden?” (siehe Abbildung 4.1). Die Ergebnisse zeigen, dass die Antwortoption “Ich würde die App nur selten nutzen und möchte sie daher nicht installieren” mit einem deutlichen Vorsprung und 72,5 Prozent aller Teilnehmer*innen am häufigsten gewählt wurde. An zweiter Stelle steht mit 57,5 Prozent die Aussage “Ich möchte nicht zu viele Apps auf meinem Handy haben, um den Überblick zu behalten”.

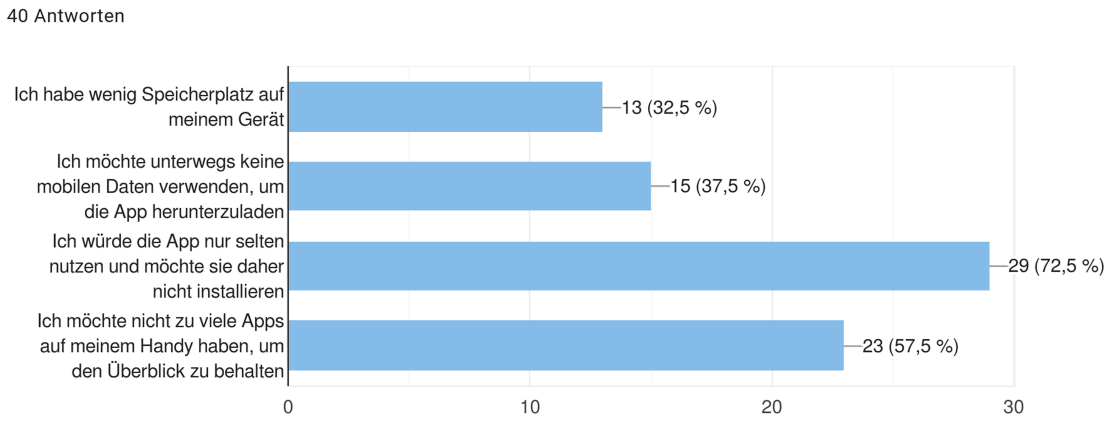

Abbildung 4.1: Auswertung der Antworten aus der Nutzerstudie auf die Frage “Welche dieser Gründe sind für Sie die zwei wichtigsten, wenn Sie sich gegen die Installation einer App entscheiden?”

Die Ergebnisse zeigen, dass Nutzer*innen eher auf Übersichtlichkeit und schnelle Zugänglichkeit der von ihnen bevorzugten Anwendungen achten als auf die Ressourcennutzung wie Speicherplatz oder mobile Daten. Eine mögliche Erklärung hierfür liegt in der begrenzten Anzahl von Apps, die ein durchschnittlicher Nutzer oder eine durchschnittliche Nutzerin regelmäßig verwendet. Eine Studie von Simform (2022) mit 3.756 Teilnehmer*innen stützt diese Annahme, indem sie zeigt, dass durchschnittliche Smartphone-Nutzer*innen etwa 40 Apps installiert haben, aber 89 Prozent ihrer Zeit auf nur 18 dieser Apps entfallen [64]. Ebenso verdeutlicht eine dreimonatige Umfrage in den USA aus dem Jahr 2017, dass 51 Prozent der Befragten innerhalb eines Monats keine neue App installiert haben [16].

Dieser Fokus auf die Reduzierung von genutzten Anwendungen und der Verzicht auf neue Installationen könnte darauf hinweisen, dass Nutzer*innen ihre Geräte eher funktional organisieren und sich auf essentielle Tools konzentrieren, um eine Überforderung durch zu viele Apps zu vermeiden. Für die Entwicklungsentscheidung bedeutet dies, dass Applikationen, die als essenziell wahrgenommen werden und einen klaren regelmäßigen Nutzen bieten, eine höhere Wahrscheinlichkeit haben, installiert zu werden.

Akzeptanz von PWA

Da Progressive Web Apps (PWAs) noch eine relativ neue Technologie darstellen, wurde in der Umfrage die Frage gestellt: “Haben Sie schon einmal von Progressive Web Apps (PWAs) gehört? Falls ja, haben Sie schon einmal eine solche App auf Ihrem Gerät installiert?” Ziel dieser Fragestellung war es, die Bekanntheit und Verbreitung dieser Art von Web-Anwendungen zu untersuchen. Die Ergebnisse zeigen, dass 75 Prozent der Befragten noch nie von PWAs gehört haben. 10 Prozent der Teilnehmenden gaben an, zwar davon gehört, aber noch nie eine solche App installiert zu haben. Lediglich 15 Pro-

zent der Befragten haben bereits eine PWA genutzt. Die Ergebnisse verdeutlichen, dass Progressive Web Apps (PWAs) vor allem bei Nutzer*innen ohne technischen Hintergrund kaum bekannt sind. So gaben 92 Prozent der Befragten, die nicht in einem technischen Bereich tätig sind, an, noch nie von PWAs gehört zu haben.

Die Ergebnisse machen deutlich, dass Progressive Web Apps (PWAs) in der breiten Öffentlichkeit noch nicht weit verbreitet oder bekannt sind. Viele Nutzer*innen assoziieren Web-Anwendungen bisher weder mit Offline-Funktionalitäten noch mit der Möglichkeit zur Installation. Eine Veränderung dieser Wahrnehmung könnte jedoch durch die verstärkte Implementierung von PWAs bei bekannten Plattformen wie YouTube, Spotify oder Pinterest erfolgen [55]. Interessanterweise ergab die Umfrage, dass 37,5 Prozent der Befragten bereits mobile Webseiten als Shortcut auf ihrem Homescreen gespeichert haben. Da der Installationsprozess von PWAs auf mobilen Browsern identisch mit dem Speichern eines Shortcuts ist, könnten viele Nutzer*innen zukünftig unwissentlich PWAs installieren, ohne sich deren Funktionsumfang bewusst zu sein. Ein solcher Übergang würde nicht nur die Akzeptanz von PWAs fördern, sondern könnte auch die Benutzerfreundlichkeit für Personen verbessern, die häufig auf bestimmte Webseiten zugreifen möchten.

Relevanz von plattformübergreifender Verfügbarkeit

Zur Bewertung der Relevanz einer plattformübergreifenden Verfügbarkeit von Anwendungen wurde die Frage gestellt: “Wie wichtig ist Ihnen die plattformübergreifende Verfügbarkeit einer App (z. B. auf iOS, Android und Desktop)?” Die Ergebnisse der Umfrage zeigen, dass nur 20 Prozent der Teilnehmenden die plattformübergreifende Verfügbarkeit als “eher unwichtig” einschätzen. Der verbleibende Anteil bewertete die Verfügbarkeit auf mehreren Plattformen entweder als “neutral – nicht so wichtig, aber ein Vorteil, wenn es möglich ist” bis hin zu “sehr wichtig – ich möchte die App auf allen meinen Geräten nutzen können.” Insbesondere 20 Prozent der Befragten stuften diese Verfügbarkeit als “sehr wichtig” ein.

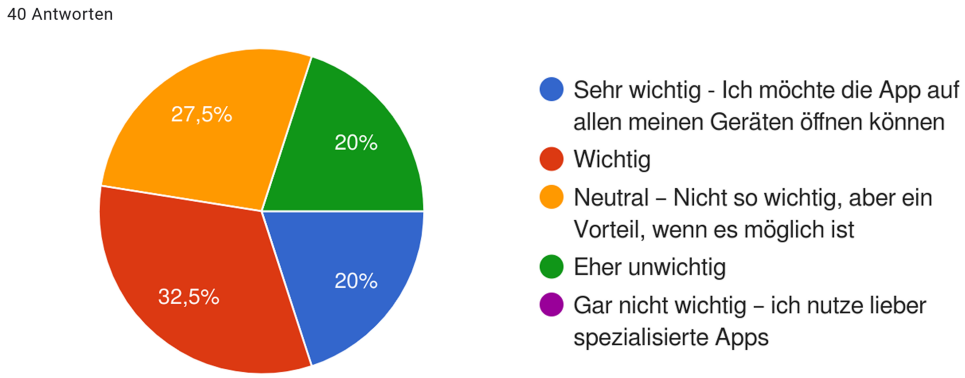

Abbildung 4.2: Auswertung der Antworten aus der Nutzerstudie auf die Frage “Wie wichtig ist Ihnen die plattformübergreifende Verfügbarkeit einer App (z. B. auf iOS, Android und Desktop)?”

Diese Daten unterstreichen die Bedeutung einer plattformübergreifenden Verfügbarkeit, da sie nicht nur die Nutzerfreundlichkeit erheblich verbessert, sondern auch den Zugriff deutlich flexibler gestaltet, da man nicht an ein spezifisches Gerät oder eine bestimmte Plattform gebunden ist. Für Entwickler*innen bietet dies eine klare Orientierung, plattformübergreifende Lösungen zu priorisieren, um den Bedürfnissen und Erwartungen der Nutzer*innen gerecht zu werden und eine größere Reichweite zu erreichen.

Ein weiterer Aspekt bei der Entwicklung mobiler Anwendungen ist die Frage, wie wichtig Nutzer*innen ein plattformabhängiges Design sowie die Nutzung nativer UI-Komponenten ist. In der durchgeführten Umfrage wurde daher die Frage gestellt: “Wie wichtig ist Ihnen, dass das Design und die Bedienung einer App an das jeweilige Betriebssystem (z. B. iOS oder Android) angepasst sind (z. B. typische Pop-ups, Buttons oder Kopfzeilen, die sich wie auf Ihrem Betriebssystem anfühlen)?” Die Ergebnisse der Umfrage zeigen, dass 17,5 Prozent der Befragten diesen Aspekt als “gar nicht wichtig” oder “eher unwichtig” bewerten. Der Großteil der Teilnehmenden stuft die Bedeutung eines plattformangepassten Designs jedoch als hoch ein: 37,5 Prozent bewerteten diesen Aspekt als „wichtig“ und weitere 12,5 Prozent als “sehr wichtig”. Besonders hervorzuheben ist, dass unter iOS-Nutzer*innen rund 61 Prozent ein solches Design als “wichtig” oder “sehr wichtig” einstuften, während dieser Anteil bei Android-Nutzer*innen lediglich 35 Prozent betrug.

Ein möglicher Erklärungsansatz für diese Unterschiede könnte in der unterschiedlichen Erwartungshaltung und Wahrnehmung der Benutzererfahrung zwischen den beiden Nutzergruppen liegen. iOS zeichnet sich durch ein einheitliches und stark reguliertes Design aus [20], das von Nutzer*innen als Teil der gewohnten Benutzererfahrung wahrgenommen wird. Android hingegen bietet größere Anpassungsfreiheit, weshalb die Erwartung einer einheitlichen Oberfläche geringer ausgeprägt sein könnte.

Für die Entwicklung bedeutet dies, dass bei iOS-Anwendungen besonderes Augenmerk auf native UI-Komponenten und die Einhaltung von Design-Richtlinien gelegt werden sollte, um die Nutzerzufriedenheit zu maximieren. Bei Android kann der Fokus stärker auf Funktionalität und Flexibilität liegen, ohne jedoch die Konsistenz zu vernachlässigen. Zielgruppen und Plattformpräferenzen sollten frühzeitig analysiert werden, um fundierte Designentscheidungen zu treffen.[6]

Speicherplatz

Der Speicherbedarf von Anwendungen kann je nach gewähltem Entwicklungsansatz variieren, weshalb in der Umfrage untersucht wurde, inwiefern dieser Faktor die Entscheidung zur Installation einer App beeinflusst. Die Ergebnisse zeigen, dass 52,5 Prozent der Befragten angaben, dass sie den Speicherbedarf einer App bei ihrer Entscheidung zur Installation “etwas” berücksichtigen (siehe Abbildung 4.3). Ein signifikanter Anteil von 42,5 Prozent gab jedoch an, dass der Speicherbedarf keinerlei Einfluss auf ihre Entscheidung hat. Lediglich 5 Prozent der Teilnehmenden – entsprechend zwei Personen – äußerten, dass sie “sehr stark” auf den Speicherbedarf achten und Apps mit hohem Speicherverbrauch vermeiden.

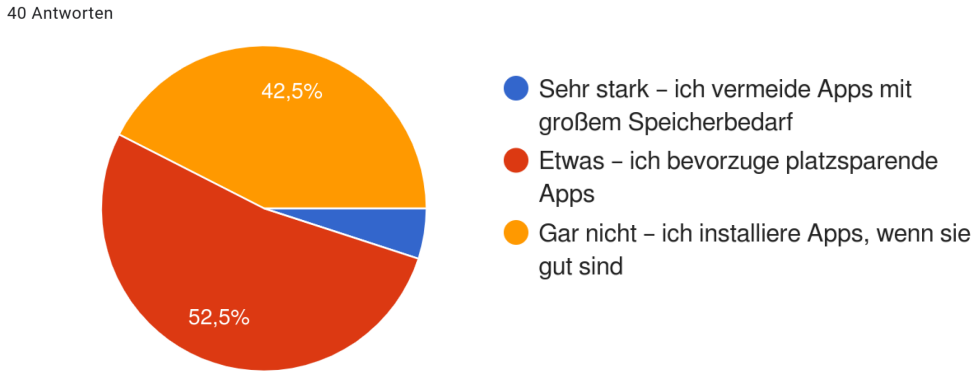

Abbildung 4.3: Auswertung der Antworten aus der Nutzerstudie auf die Frage “Wie stark beeinflusst der Speicherplatzbedarf Ihre Entscheidung, eine App zu installieren?”

Diese Ergebnisse lassen darauf schließen, dass der Speicherbedarf zwar eine gewisse Rolle spielt, jedoch für die Mehrheit der Nutzer*innen nicht der entscheidende Faktor ist. Insbesondere zeigt der geringe Anteil an Nutzer*innen, die stark auf den Speicherbedarf achten, dass andere Faktoren wie Funktionalität, Usability oder spezifische Anforderungen der App vermutlich höhere Priorität haben.

[6] Dieser Text wurde mit Unterstützung von KI geschrieben.

4.2 Wirtschaftliche Aspekte

Die wirtschaftlichen Aspekte der App-Entwicklung spielen eine entscheidende Rolle bei der Wahl der geeigneten Entwicklungsstrategie. Unternehmen und Entwickler*innen stehen vor der Herausforderung, begrenzte Ressourcen wie Zeit, Budget und Personal effektiv zu nutzen, während sie gleichzeitig eine qualitativ hochwertige App entwickeln, die den Anforderungen der Zielgruppe entspricht. Die verschiedenen Entwicklungsmethoden – native, hybride und webbasierte Ansätze – bringen jeweils spezifische Kosten- und Nutzenfaktoren mit sich, die bei der strategischen Planung berücksichtigt werden müssen.[7]

Eine fundierte wirtschaftliche Analyse ist essenziell, um sicherzustellen, dass die geplanten Investitionen in die App-Entwicklung einen maximalen Return on Investment (ROI) generieren. Aspekte wie die Entwicklungs- und Wartungskosten sind hierbei zentrale Entscheidungskriterien.

4.2.1 Vergleich von Entwicklungs- und Wartungskosten

Um die Entwicklungskosten verschiedener Methoden miteinander vergleichen zu können, ist es essenziell, zunächst die durchschnittlichen Gehälter der Entwickler*innen in den jeweiligen Spezialisierungsbereichen zu analysieren. Ein signifikanter Unterschied zeigt sich insbesondere bei der nativen Entwicklungssprache Swift, die für iOS-Anwendungen verwendet wird, im Vergleich zu TypeScript, welches in der Web-Entwicklung sowie in Cross-Plattform-Lösungen wie React Native genutzt wird.

Laut einer Umfrage von Stack Overflow beträgt das durchschnittliche Jahresgehalt von Swift-Entwicklerinnen 75.184 US-Dollar, während TypeScript-Entwickler*innen im Durchschnitt 65.907 US-Dollar pro Jahr verdienen [58]. Dieser Gehaltsunterschied könnte auf die geringere Verfügbarkeit von Swift-Entwickler*innen im Vergleich zu TypeScript- oder JavaScript-Entwickler*innen zurückzuführen sein. In der gleichen Umfrage zeigte sich, dass nur 4,7 Prozent der 60.171 Befragten regelmäßig in Swift programmieren, während 62,3 Prozent JavaScript und 38,5 Prozent TypeScript nutzen [57]. Dies deutet darauf hin, dass die Spezialisierung auf Swift seltener ist, was das Gehalt dieser Entwickler*innen durch das geringe Angebot erhöht.

Der signifikante Unterschied in den Gehältern der Entwickler*innen sowie die Notwendigkeit, native Anwendungen separat für jede Plattform zu entwickeln, kann zu deutlich höheren initialen Erstellungs- und Entwicklungskosten für mobile Apps führen (siehe Tabelle 4.1).

[7] Dieser Text wurde mit Unterstützung von KI geschrieben.

Criterion	Native	Cross-Platform	PWA
Initial Development	25,000$ - 150,000$ per platform	20,000$ - 100,000$	10,000$ - 50,000$
Monthly Maintenance	1,000$ - 5,000$ per platform	500$ - 2,500$	200$ - 2,000$
Development Time	Long	Medium	Short
Required Developers	Specialized iOS and Android developers	React Native or Flutter developers	Web developers

Tabelle 4.1: Vergleich der Kosten zwischen den Entwicklungsmethoden (Tabelle aus [29])

Es ist wichtig zu betonen, dass die in Tabelle 4.1 dargestellten Entwicklungskosten als allgemeine Orientierung dienen und insbesondere für kleinere bis mittelgroße Applikationen gelten können. Die Grafik basiert nicht auf umfassenden, empirischen Studien, sondern soll vielmehr eine grobe Vorstellung der Kostenunterschiede zwischen verschiedenen Entwicklungsansätzen vermitteln. Die tatsächlichen Kosten können je nach Projektumfang, Anforderungen, geografischem Standort der Entwickler*innen und spezifischen Technologien erheblich variieren.

Da in Q4 von 2024 99,53 Prozent der weltweit genutzten Smartphone-Betriebssysteme Android und iOS sind [69], wurde die Analyse der nativen App-Entwicklung auf diese beiden Plattformen beschränkt. Da die Entwicklung mit Cordova technisch nicht wesentlich von der Webentwicklung unterscheidet, kann diese Art der hybriden Entwicklung in den Bereich der Webentwicklung eingeordnet werden.

Neben den Stundensätzen der Entwickler*innen gibt es eine Vielzahl weiterer Faktoren, welche die Kosten der App-Entwicklung beeinflussen können. Einer der zentralen Aspekte ist die Entwicklungszeit, die je nach gewähltem Entwicklungsansatz erheblich variieren kann. Wie in Tabelle 4.1 dargestellt, benötigen native Apps in der Regel mehr Zeit für die Entwicklung als Cross-Plattform- oder Web-Apps. Dies liegt vor allem daran, dass native Anwendungen für Android und iOS jeweils separate Codebasen erfordern, die unabhängig voneinander erstellt und gepflegt werden müssen.

Diese Spezialisierung in verschiedenen Programmiersprachen wie Kotlin und Swift führt meist zu einer Trennung innerhalb des Teams, was wiederum die Koordination und Kommunikation zwischen Entwickler*innen für Android und iOS entscheidend macht. Ohne eine effektive Abstimmung besteht das Risiko, dass Funktionalitäten oder Designaspekte auf den beiden Plattformen unterschiedlich umgesetzt werden, was die Konsistenz der Nutzererfahrung beeinträchtigen kann. Diese zusätzlichen Anforderungen wirken sich nicht nur auf die direkten Entwicklungskosten aus, sondern auch auf die administrativen Aufwände. Projektmanagerinnen und technische Leiterinnen müssen mehr Zeit und Ressourcen investieren, um die Zusammenarbeit zwischen den Plattformteams zu

koordinieren, gemeinsame Standards zu etablieren und kontinuierliche Abstimmungen sicherzustellen.

Kostenfaktoren von Web-Anwendungen

Mobile Web-Anwendungen können in der Regel schneller und kosteneffizienter als native und hybride Anwendungen entwickelt werden. Dies lässt sich zum einen darauf zurückführen, dass es, wie zuvor erläutert, eine große Anzahl an Entwickler*innen in diesem Bereich gibt, die aufgrund des breiten Angebots im Durchschnitt niedrigere Stundensätze verlangen. Zum anderen weist die Web-Entwicklung einen hohen Anteil an Full-Stack-Entwickler*innen auf, die sowohl Frontend- als auch Backend-Aufgaben übernehmen können. Eine Umfrage aus dem Jahr 2024 mit 58.950 Teilnehmenden ergab, dass 30,7 Prozent der Befragten sich als Full-Stack-Entwickler*innen identifizieren [56].

Die hohe Anzahl an Full-Stack-Entwickler*innen in der Web-Entwicklung kann unter anderem darauf zurückgeführt werden, dass JavaScript sowohl im Frontend als auch im Backend weit verbreitet ist. Durch Laufzeitumgebungen wie Node.js können Entwicklerinnen mit einer einzigen Programmiersprache einen Großteil der Anwendungsarchitektur abdecken, was den Einstieg in die Full-Stack-Entwicklung erleichtert. Diese enge Verbindung von Frontend- und Backend-Technologien führt nicht nur zu einer effizienteren Entwicklung, sondern trägt auch dazu bei, dass Web-Anwendungen schneller und kostengünstiger entwickelt werden können. Unternehmen profitieren davon, da sie nicht zwingend separate Teams für die beiden Bereiche aufbauen müssen, sondern auf Entwickler*innen zurückgreifen können, die in beiden Bereichen kompetent sind.

Kostenfaktoren zwischen React Native und Flutter

Da die Entwicklerkosten für React Native und Flutter in Deutschland weitgehend vergleichbar sind [1], rücken andere wirtschaftliche Aspekte wie Entwicklungszeit und Skalierbarkeit in den Fokus. Flutter ermöglicht durch seine umfassende Sammlung vorgefertigter UI-Widgets eine effiziente Entwicklung, insbesondere für Minimal Viable Products (MVPs), wodurch es laut einigen Quellen als kosteneffiziente Lösung für frühe Entwicklungsphasen gilt [1, 62]. Allerdings kann die Notwendigkeit, plattformspezifischen Code für komplexe UI-Elemente zu entwickeln, welche nicht mit dem Framework abgebildet werden können, die Entwicklungszeit verlängern und somit die Kosten erhöhen. React Native hingegen zeichnet sich durch eine breite Entwicklerbasis und eine bewährte Skalierbarkeit aus, die durch den erfolgreichen Einsatz in komplexen Anwendungen wie Microsoft Teams, Instagram, Amazon Shopping und Pinterest belegt wird [77]. Beide Frameworks bieten kosteneffiziente Entwicklungsansätze, wobei Flutter eine schnelle MVP-Entwicklung ermöglicht und React Native eine erprobte Grundlage für skalierbare Anwendungen bietet.

4.2.2 Monetarisierungsstrategien und deren Umsetzbarkeit

Die Wahl der Monetarisierungsstrategie hat einen erheblichen Einfluss auf die technische Umsetzung einer mobilen Anwendung und damit auf die Entscheidung zwischen nativer, cross-platform oder webbasierter Entwicklung. Besonders bei den verbreiteten Monetarisierungsmodellen – direkter Verkauf, In-App-Käufe, Abonnementmodelle und werbebasierte Finanzierungsstrategien – können sich signifikante Unterschiede in der technischen Realisierbarkeit sowie den wirtschaftlichen Implikationen der jeweiligen Plattform ergeben.[8]

Direkter Verkauf (Paid Apps)

Native und Cross-Plattform Apps profitieren hierbei von der zentralen Vertriebsplattform der App-Stores, die eine einfache Zahlungsabwicklung und hohe Sichtbarkeit gewährleisten kann. Allerdings fallen dabei Transaktionsgebühren an, was die Gewinnmargen reduziert. Die Umsatzbeteiligung durch die App-Store-Betreiber variiert abhängig von der Umsatzhöhe des Entwicklers oder der Entwicklerin. Für kleinere Unternehmen mit einem Jahresumsatz von weniger als 1 Million US-Dollar beträgt die Provision sowohl im Apple App Store als auch im Google Play Store 15 Prozent. Überschreitet der Jahresumsatz diese Grenze, erhöht sich die Umsatzbeteiligung auf 30 Prozent [7, 46].

Webbasierte Anwendungen bieten hingegen die Möglichkeit, Zahlungen direkt über externe Zahlungsdienstleister wie Stripe oder PayPal abzuwickeln, wodurch die bei nativen und cross-plattform Applikationen anfallenden Umsatzbeteiligungen der App-Store-Betreiber umgangen werden können [59, 73]. Allerdings fehlt ihnen die Sichtbarkeit in den etablierten App-Stores, wodurch potenzielle Nutzende aktiv auf die entsprechende Website geleitet werden müssen, was die Reichweite und Nutzerakquise einschränken kann.

In-App-Käufe (Freemium-Modell)

Die Monetarisierungsstrategie der In-App-Käufe oder auch Freemium-Modell genannt, ist für alle Apps relevant, welche dem Nutzer oder der Nutzerin die Möglichkeit bieten wollen, in der App Informationen, Dienstleistungen oder andere Produkte kaufen zu können. Plattformspezifische und cross-platform Apps bieten hierbei tiefgehende Integrationen mit den Zahlungsinfrastrukturen von Apple und Google, wodurch Nutzer*innen mit gespeicherten Zahlungsinformationen besonders einfach Transaktionen durchführen können [8, 45]. Native Apps können mit solchen Erleichterungen einen Vorteil bei der Nutzerbindung aufweisen, was sich indirekt auf die Konversionsrate auswirken kann. Eine Studie, welche die Teilnahme von Nutzer*innen an aufeinanderfolgenden Erhebungswellen innerhalb

[8] Dieser Text wurde mit Unterstützung von KI geschrieben.

einer Panelstudie untersuchte, ergab, dass App-Nutzer*innen seltener absprangen als Nutzer*innen mobiler Browser [61]. Dieser Vorteil wird jedoch durch die obligatorischen Umsatzbeteiligungen der Plattformbetreiber getrübt [7, 46]. Web-Apps haben hier den Vorteil, dass sie alternative Zahlungsanbieter flexibel integrieren können und nicht an die Store-Richtlinien gebunden sind.

Abonnementmodelle

Der Trend zu abonnementbasierten Anwendungen zeigt sich insbesondere bei Cross-Plattform und nativen Apps, da App-Stores automatische Verlängerungen sowie eine komfortable Verwaltung der Abos ermöglichen [6, 47]. Dies kann zu stabileren und vorhersehbaren Einnahmen führen. Web-Apps bieten in der Gestaltung von Abonnementmodellen auch wieder eine höhere Flexibilität, da sie nicht an zentrale App-Stores gebunden sind. Allerdings erfordert dies den Aufbau einer eigenständigen Zahlungsinfrastruktur. Zudem können Web-Apps für Nutzer*innen eine höhere Einstiegshürde darstellen, da sie nicht von einer zentralisierten, automatisierten Abonnementverwaltung wie in App-Stores profitieren.

Werbefinanzierung

Werbebasierte Monetarisierungsstrategien sind besonders effektiv, wenn Nutzer*innen eine Anwendung über einen längeren Zeitraum hinweg nutzen, da dadurch mehr Werbeanzeigen ausgespielt werden können. Daher ist es essenziell, das Nutzungsverhalten verschiedener Anwendungsarten zu analysieren. Durchschnittlich verbringen Nutzer*innen 87 Prozent ihrer Bildschirmzeit in nativen Apps, was diesen einen deutlichen Vorteil verschafft [14]. Zudem liegt die Retention Rate, also der Anteil der Nutzer*innen, die eine Anwendung nach einem bestimmten Zeitraum weiterhin aktiv nutzen, nach 90 Tagen bei nativen Apps bei etwa 20 Prozent, während sie bei mobilen Webseiten auf lediglich 5 Prozent sinkt [14].

Web-Apps bieten zwar eine leichtere Integration mit etablierten Werbenetzwerken wie Google AdSense, weisen jedoch, wie zuvor beschrieben, geringere Interaktionsraten auf. Darüber hinaus stehen sie vor der Herausforderung von Werbeblockern in mobilen Browsern. Laut einer Studie aus dem Jahr 2024 haben 11 Prozent der Nutzer*innen auf ihren mobilen Geräten einen Werbeblocker installiert [79], wodurch die Einnahmen durch Werbeanzeigen zusätzlich abnehmen können.

Zusammenfassend lässt sich feststellen, dass native und Cross-Plattform Apps durch ihre tiefe Integration mit App-Store-Ökosystemen Monetarisierungsstrategien wie In-App-Käufe oder Abonnements besonders effizient und einfach umsetzen können, jedoch mit hohen Umsatzbeteiligungen und Richtlinienbeschränkungen konfrontiert sind. Web-Apps bieten mehr Flexibilität in der Preisgestaltung und Zahlungsabwicklung, stehen jedoch

vor Herausforderungen in der Nutzerkonversion und der Distributionsstrategie. Die Wahl der geeigneten Monetarisierungsstrategie sollte daher nicht isoliert betrachtet, sondern in enger Abstimmung mit der gewählten Entwicklungsplattform getroffen werden.[9]

[9] Dieser Text wurde mit Unterstützung von KI geschrieben.

5 Konzeption von Handlungsempfehlungen

Auf Grundlage der in dieser Arbeit gewonnenen Erkenntnisse und Schlussfolgerungen können nun konkrete Handlungsempfehlungen formuliert werden, die sich aus den spezifischen Anforderungen an die Anwendung ableiten. Dabei ist es essenziell, eine ganzheitliche Perspektive einzunehmen, die das Zusammenspiel technischer, nutzerzentrierter und wirtschaftlicher Faktoren berücksichtigt. Nur durch eine integrative Betrachtung dieser Aspekte kann eine fundierte Entscheidungsgrundlage für die Entwicklung einer Anwendung geschaffen werden.[10]

5.1 Kriterienkatalog

Wie in dieser Arbeit dargelegt, existiert eine Vielzahl an Kriterien, die für die Wahl einer geeigneten Entwicklungsstrategie entscheidend sind. Angesichts dieser Erkenntnis erscheint es sinnvoll, einen strukturierten Kriterienkatalog zu erstellen. Dieser soll Entwicklern und Entscheidungsträgern als Orientierungshilfe dienen, indem er relevante Kriterien in Form gezielter Fragestellungen zusammenfasst. Durch die systematische Beantwortung dieser Fragen können fundierte Einschätzungen darüber getroffen werden, welche Entwicklungsstrategie, sei es eine native, hybride oder webbasierte Lösung, in einem spezifischen Anwendungskontext am besten geeignet ist.

Darüber hinaus ist es für potenzielle App-Entwickler*innen von entscheidender Bedeutung, sich frühzeitig über mögliche Herausforderungen und Einschränkungen der jeweiligen Entwicklungsstrategie bewusst zu werden. Ein solcher Kriterienkatalog kann dabei helfen, kritische Aspekte zu identifizieren und potenzielle Sackgassen in der Entwicklung zu vermeiden. Dadurch wird das Risiko minimiert, in späteren Phasen der Entwicklung auf schwer überwindbare Hürden zu stoßen, die zu unerwarteten Mehraufwänden oder gar einem Scheitern des Projekts führen könnten.

Um Entscheidungsträger*innen eine Entscheidungsgrundlage zu bieten, ist es essenziell, dass der entwickelte Kriterienkatalog eine Einschätzung darüber ermöglicht, wie gut eine Entwicklungsstrategie zu den Anforderungen einer App passt. Zu diesem Zweck wird jeder Antwort innerhalb des Kriterienkatalogs ein spezifisches Gewicht für die drei möglichen Entwicklungsstrategien zugewiesen.

[10] Dieser Text wurde mit Unterstützung von KI geschrieben.

Fragen	**Antworten**	**Gewichte (N/CP/W)**
Do you need to use the following device-specific features: Access Contacts, NFC, Bluetooth, or SMS?	Yes, these features are required	8/3/0
	No, these features are not required	5/5/5
Do you want to build a quick prototype first, or start developing the final application immediately?	Build a prototype for rapid validation	1/5/8
	Develop the full application from the start	5/5/5
What type of app are you planning to create?	Social Media	5/8/3
	Banking or Finance	8/5/2
	E-Commerce	3/5/8
	Newsletter or Information-based	1/3/8
How frequently do you expect users to use your app?	Daily	8/5/3
	Weekly	5/5/5
	Monthly or less	1/3/8
What is your budget for app development?	€5,000 - €50,000	1/5/8
	€50,000 - €100,000	3/8/5
	€100,000 or more	5/5/5
What is your development team's existing technical expertise?	Web (JavaScript/TypeScript) background	2/5/8
	No prior development experience	3/8/5
	Native mobile development	8/5/5
What is your primary monetization approach?	App store subscriptions/in-app purchases	8/8/5
	Advertising-based revenue	8/8/5
	Direct payment (Paid Apps)	5/5/8
Does your application need to function offline without an internet connection?	Yes, offline functionality is critical	8/8/3
	No, internet connectivity is always assumed	5/5/5
How frequently do you anticipate needing to deploy updates?	Frequent updates (weekly or more often)	1/1/8
	Moderate updates (monthly)	5/5/8
	Rare updates (quarterly or less)	5/5/5

Tabelle 5.1: Fragen, Antworten und deren Gewichte bezüglich der Entscheidung von App Entwicklungsstrategien (N=Native, CP=Cross-Platform, W=Web)

Die in der letzten Spalte genannten Gewichte wurden als Ergebnis der bisherigen Arbeit heuristisch ermittelt. Hohe Gewichte bedeuten, dass eine bestimmte Entwicklungsstrategie besonders gut für die jeweilige Anforderung geeignet ist. Je geringer das Gewicht, desto weniger entspricht die Strategie den Anforderungen. Ein Gewicht von null bedeutet, dass die Entwicklungsstrategie für die entsprechende Anforderung ungeeignet ist.

Die Auswahl der Fragen, Antworten und Gewichte basiert heuristisch auf den Analysen in den folgenden Abschnitten: Frage 1: 3.2.0.1, Frage 2: 4.2.1, Frage 3: 4.1.2, Frage 4: 4.1.2.2, Frage 5: 4.1, Frage 6: 3.3.0.2, Frage 7: 4.2.2, Frage 8: 3.2.0.4, Frage 9: 3.4

Die Gewichtung basiert dabei auf der Fibonacci-Zahlenreihe bis zur Zahl 8. Die Wahl dieser Zahlenfolge ist nicht zufällig, sondern beruht auf einer Eigenschaft der Fibonacci-Zahlenreihe: Mit zunehmenden Werten wachsen die Abstände zwischen den Zahlen exponentiell. Dies sorgt dafür, dass besonders wichtige Kriterien deutlich stärker ins Gewicht fallen als weniger entscheidende Faktoren. Eine lineare Skalierung der Gewichte könnte dazu führen, dass bedeutende Aspekte verwässert würden. Durch die exponentielle Natur der Fibonacci-Skala hingegen wird sichergestellt, dass zentrale Anforderungen einen entsprechend höheren Einfluss auf das Gesamtergebnis haben, während weniger relevante Kriterien nur in einem angemessenen Maß berücksichtigt werden.[11]

Zudem erleichtert die Verwendung der Fibonacci-Zahlenreihe die Zuweisung der Gewichte in der Praxis. Anstatt feine Abstufungen wie zwischen den Werten 5 und 6 treffen zu müssen, erfolgt die Entscheidung zwischen Werten mit deutlich mehr Abstand, wie beispielsweise 5 und 8. Dies reduziert subjektive Unsicherheiten bei der Gewichtung, da größere Abstände zwischen den Zahlen eine klarere Einordnung der Bedeutung eines Kriteriums ermöglichen können.

Um diese Methode der Entscheidungsfindung möglichst vielen Entscheidungsträger*innen zugänglich zu machen, wurden die Fragen in Englisch formuliert. Zudem erfolgte eine Vorabtestung der Fragen in einer kleineren Gruppe, um deren Verständlichkeit sicherzustellen. Dabei zeigte sich, dass in einigen Fällen nicht ausreichend Informationen zu den Anforderungen vorlagen, um eine fundierte Antwort geben zu können. Daher wurde entschieden, jeder Frage die zusätzliche Antwortmöglichkeit „I don't know“ hinzuzufügen und mit den Gewichten 5/5/5 zu versehen, da in diesem Fall alle Entwicklungsstrategien als gleichermaßen wahrscheinlich betrachtet werden.

Zudem zeigte die Testphase, dass bestimmte Anforderungen je nach Anwendung eine höhere Relevanz besitzen als andere Kriterien und daher individuell stärker gewichtet werden sollten. Infolgedessen wurde in der finalen Berechnungsmethode eine zusätzliche Gewichtung für besonders relevante Fragen integriert, sodass diese mit einem doppelten Faktor in die Entscheidungsfindung einfließen. Diese zweite Art der Gewichtung wird individuell durch Nutzer*innen der Methode bestimmt.

Auf Grundlage dieser Erkenntnisse wurde die folgende Formel zur Berechnung der prozentualen Eignung einer Entwicklungsstrategie abgeleitet.

[11] Dieser Text wurde mit Unterstützung von KI geschrieben.

$$Score_s = \left\lfloor \frac{100}{W_{\text{total}}} \sum_{q \in Q} (w_{q,o,s} \cdot d_q) \right\rfloor$$

wobei:

$W_{\text{total}} = \sum_{q \in Q} (8 \cdot d_q)$ (maximal erreichbare Punktzahl)

$$d_q = \begin{cases} 2 & \text{wenn } q \in Q_{double} \\ 1 & \text{sonst} \end{cases}$$

$w_{q,o,s} \in F_{\leq 8} = \{1, 2, 3, 5, 8\}$ (Fibonacci-Zahlen bis 8)

$w_{q,o,s} =$ vordefinierte Gewichte für Strategie s in Frage q mit gewählter Option o

$Q =$ Menge aller Fragen

$Q_{double} =$ Menge der Fragen mit doppelter Gewichtung durch Nutzer*innen

$s \in \{$Nativ, Cross-Platform, Web-App$\}$

Beispiel:

Wenn $Q = \{q_1, q_2, q_3\}$ und $Q_{double} = \{q_1\}$:

$W_{total} = (8 \cdot 2) + (8 \cdot 1) + (8 \cdot 1) = 32$

$\text{Score}_{\text{nativ}} = \left\lfloor \frac{100}{32} (\cdot S_{q_1,o,\text{nativ}} \cdot 2 + S_{q_2,o,\text{nativ}} \cdot 1 + S_{q_3,o,\text{nativ}} \cdot 1) \right\rfloor$

Um die Berechnung für Entscheidungsträger*innen in der App-Entwicklung zu vereinfachen und intuitiver zu gestalten, wurde im Rahmen dieser Arbeit eine eigene Anwendung entwickelt. Diese Anwendung setzt die formulierten Evaluationsfragen inklusive der oben eingeführten Bewertungsformeln in eine intuitiv nutzbare App um. Zudem zeigt die Anwendung potenzielle kritische Punkte in Bezug auf die gewählte oder empfohlene Entwicklungsstrategie. Um die Bewertung der Anforderungen in Bezug auf die Entwicklungsstrategien nachvollziehbar zu gestalten, werden zu jeder Frage kurze Begründungen bereitgestellt. Außerdem werden zu jeder Begründung auch Quellen angezeigt, wie zum Beispiel Abschnitte dieser Arbeit, sodass Nutzer*innen gezielt weiterführende Informationen aufrufen können. Darüber hinaus werden konkrete Handlungsempfehlungen gegeben, die es ermöglichen, die identifizierte Strategie optimal umzusetzen und das Potenzial der Anwendung bestmöglich umzusetzen.

Eine prototypische Umsetzung dieser Methode ist unter folgendem Link verfügbar: https://appdecide.netlify.app/. Der entsprechende Code dafür ist öffentlich unter https://github.com/BenniBM/AppDecide einsehbar. Zudem ist der kompilierte Code im Anhang B zu finden und mit einem lokalen Web-Server ausführbar.

Die entwickelte Anwendung wurde zunächst selbst anhand des erstellten Fragenkatalogs evaluiert, um die geeignetste Entwicklungsstrategie zu bestimmen. Dabei ergab die Berechnung anhand der Tabelle eine prozentuale Eignung von 85 Prozent für die Wahl einer Web-App. Um eine möglichst breite Abdeckung unterschiedlicher Nutzungsszenarien zu gewährleisten, wurde die Anwendung durch geringen Aufwand zusätzlich als Progressive Web App (PWA) entwickelt, sodass sie auch offline nutzbar und installierbar ist.

Abbildung 5.1 zeigt ein Beispiel, wie das Ergebnis für eine Evaluation von Anforderungen einer App aussehen kann.

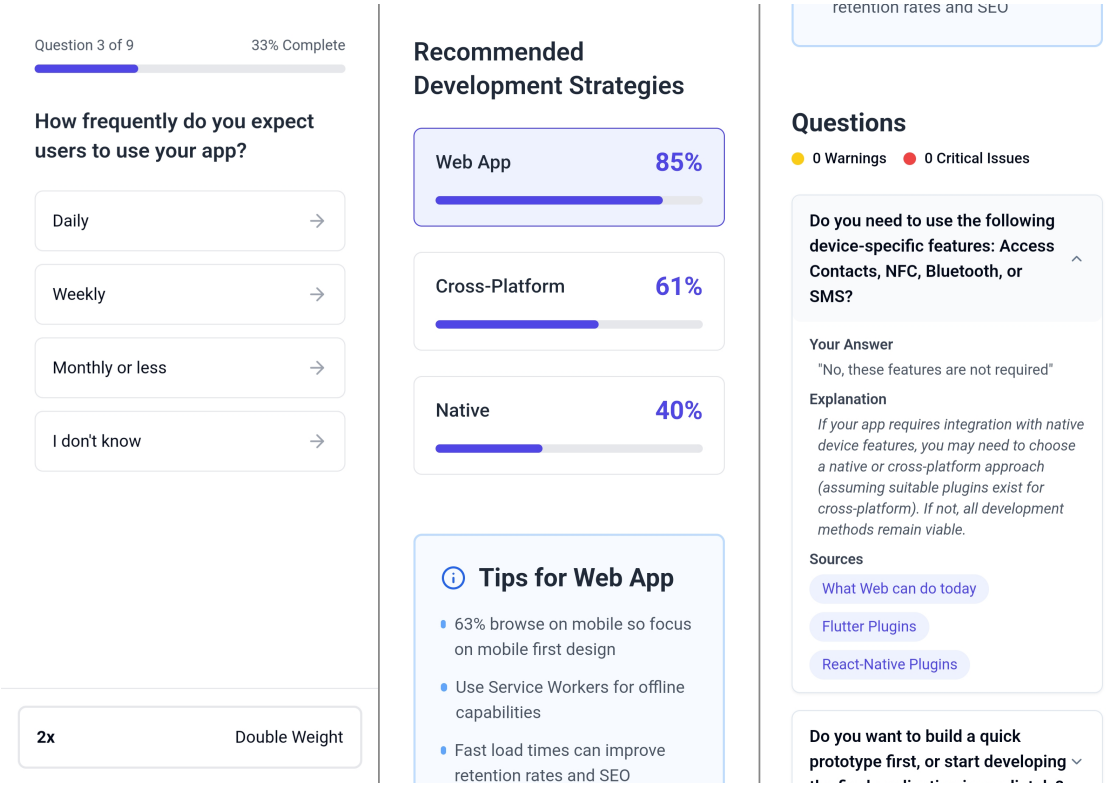

Abbildung 5.1: Beispiel für die Anzeige des Ergebnisses einer Evaluation von Entwicklungsstrategien in der Evaluations-App

Es ist wichtig zu betonen, dass nicht alle in dieser Arbeit betrachteten Faktoren zur Entscheidungsfindung zwischen den verschiedenen Entwicklungsstrategien in einer quantitativen Bewertung ideal erfasst werden können. Einige Aspekte, insbesondere solche mit subjektiven oder kontextabhängigen Auswirkungen, lassen sich nicht vollständig in ein standardisiertes Berechnungsmodell überführen. Daher sollte die hier vorgestellte

methodische Bewertung als unterstützendes Instrument wahrgenommen werden, welches eine erste Einschätzung bietet. Entscheidungsträger*innen sollten diese erste Analyse als Orientierungshilfe nutzen, sich jedoch zusätzlich mit den spezifischen Anforderungen ihrer Anwendung befassen.

6 Fazit und Ausblick

Die Wahl der optimalen Entwicklungsstrategie für mobile Applikationen stellt eine zentrale Herausforderung dar, da sie Auswirkungen auf die Entwicklungskosten, die Leistungsfähigkeit, den Wartungsaufwand und die Nutzererfahrung haben kann. Trotz der Vielzahl an verfügbaren Technologien fehlt es vielen Entscheidungsträgern an einer strukturierten Methode, um eine fundierte Auswahl zu treffen. Diese Arbeit behandelt dieses Problem durch eine systematische Untersuchung zentraler Entscheidungskriterien anhand einer quantitativen Nutzerstudie und einer umfassenden Literaturrecherche. Dabei werden Best Practices abgeleitet und eine Methode zur Wahrscheinlichkeitsberechnung entwickelt, welche die am besten geeignete Entwicklungsstrategie ermittelt. Ergänzend wurde ein interaktives Tool implementiert, das die Analyse erleichtert und zugleich konkrete Handlungsempfehlungen liefert.

6.1 Zusammenfassung der Ergebnisse

Diese Arbeit zeigt, dass die Wahl der Entwicklungsstrategie unter anderem stark von technologischen Faktoren beeinflusst wird. Native und hybride Apps bieten tiefere Systemintegration, insbesondere durch den Zugriff auf native Schnittstellen, während moderne Web-Technologien diesen Rückstand zunehmend aufholen. Gleichzeitig ermöglichen Web-Anwendungen durch die standardmäßige Nutzung von URLs eine einfache Teilbarkeit und Speicherung von Inhalten. Im Thema Performance sind native Apps oft im Vorteil, da sie in der Regel eine geringere CPU-Auslastung und einen niedrigeren Arbeitsspeicherverbrauch haben. Dennoch gibt es Szenarien, in denen Web-Anwendungen effizienter arbeiten als ihre Gegenstücke, insbesondere im Hinblick auf Daten- und Energieverbrauch. Auch bei der Offline-Fähigkeit liegen native und hybride Apps im Vorteil, da sie Inhalte standardmäßig lokal auf dem Endgerät speichern. Web-Anwendungen können jedoch mithilfe von und Service Workern ebenfalls offline nutzbar gemacht werden, was mit vergleichsweise geringem zusätzlichen Entwicklungsaufwand realisierbar ist.

Zudem hat sich gezeigt, dass die Entwicklung nativer, hybrider und webbasierter Anwendungen unterschiedliche Entwicklungsumgebungen und Ansätze erfordert. Während Web-Apps flexibel mit gängigen Code-Editoren erstellt werden können, benötigen native Apps spezifische Tools wie Android Studio oder Xcode, die eng mit den jeweiligen Plattform-SDKs verknüpft sind. Cross-Plattform-Frameworks wie Flutter und React Native ermöglichen eine gemeinsame Codebasis, erfordern aber dennoch in manchen Fällen plattformspezifische Anpassungen. Auch die Lernkurve variiert. Webentwickler*innen können oft schneller in React Native einsteigen, während Flutter eine Einarbeitung in Dart erfordert. Bei schnellen Updates sind Web-Apps klar im Vorteil, da Änderungen sofort für alle Nutzer*innen verfügbar sind, während native Apps meist App-Store-Prüfungen

durchlaufen müssen. Zwar ermöglichen automatische Updates eine nahtlose Aktualisierung, doch ältere App-Versionen können weiterhin genutzt werden, was eine langfristige Backend-Kompatibilität erfordert.

Ein weiterer Aspekt der Wahl zwischen den Entwicklungsmethoden ist das Nutzerverhalten. Die Analyse der Nutzerakzeptanz zeigt, dass die Wahl der Entwicklungsmethode maßgeblich von der Nutzungsfrequenz, der Zielplattform und der Nutzererwartung abhängt. Die durchgeführte Studie mit 40 Teilnehmenden verdeutlicht, dass Nutzer*innen bei selten genutzten Anwendungen wie E-Commerce mobile Websites bevorzugen, um Installationshürden zu umgehen. Hauptgründe gegen App-Installationen sind die seltene Nutzung (72,5 %) und der Wunsch nach Übersichtlichkeit (57,5 %). Dies spricht für webbasierte Lösungen oder auch Progressive Web Apps (PWAs), deren Installationsprozess Nutzer*innen bereits durch Shortcuts vertraut ist (37,5 %), obwohl PWAs selbst nur 15 % der Befragten bekannt sind.

Plattformübergreifende Verfügbarkeit ist für 80 % der Teilnehmenden relevant, wobei iOS-Nutzer*innen stärker auf natives Design achten (61 % vs. 35 % bei Android). Dies unterstreicht die Bedeutung nativer UI-Komponenten für iOS-Apps. Der Speicherplatzbedarf beeinflusst nur 5 % der Nutzer*innen entscheidend, sodass Leistungsfähigkeit und Funktionalität priorisiert werden können.

Die wirtschaftliche Analyse zeigt, dass die Wahl der Entwicklungsmethode erheblich von Budget, Skalierungsbedarf und Monetarisierungsstrategie abhängt. Native Apps verursachen die höchsten Initialkosten, bedingt durch spezialisierte Entwickler*innen (Swift: 75.184 USD/Jahr vs. TypeScript: 65.907 USD/Jahr) und parallele Codebasen. Cross-Platform-Lösungen reduzieren Kosten durch Code-Wiederverwendung, während PWAs dank Web-Technologien und Full-Stack-Entwickler*innen (30,7 % im Web-Bereich) am günstigsten umsetzbar sind. Wartungskosten folgen diesem Trend.

Bei der Monetarisierung bieten native und Cross-Platform-Apps Vorteile durch App-Stores wie Reichweite und integrierte Zahlungen, jedoch mit Umsatzbeteiligungen von 15 bis 30 Prozent. Web-Apps umgehen Store-Gebühren, benötigen aber eigenes Marketing. In-App-Käufe und Abonnements sind in nativen Apps meist effizienter und einfacher umsetzbar. Eine höhere Retention-Rate bei nativen Apps begünstigt ein werbebasiertes Monetarisierungsmodell (20 % bei nativen und Cross-Plattform-Apps vs. 5 % bei Web-Apps). Gleichzeitig wird Werbung in Web-Apps durch Ad-Blocker eingeschränkt, die von 11 Prozent der Nutzer*innen verwendet werden.

Um geeignete Entwicklungsstrategien für einen individuellen Anforderungskatalog zu empfehlen, hat diese Arbeit gezeigt, dass es entscheidend ist, eine Vielzahl relevanter Aspekte systematisch zu berücksichtigen und zu bewerten. Aus diesem Grund wurde

in dieser Arbeit eine Methode entwickelt, die es ermöglicht, eine fundierte Analyse der verschiedenen Optionen durch Kriterien vorzunehmen. Der Ansatz berücksichtigt nicht nur technische, sondern auch wirtschaftliche und nutzerzentrierte Perspektiven, die hierbei gleichermaßen in die Entscheidungsfindung einfließen.[12]

6.2 Beantwortung der Forschungsfrage

Die in dieser Arbeit entwickelte Methodik, basierend auf einem strukturierten Kriterienkatalog der zuvor genannten Faktoren, bietet eine fundierte Entscheidungsgrundlage. Sie ermöglicht es, Best Practices abzuleiten, die Unternehmen und Entwickler*innen bei der Wahl der optimalen Entwicklungsstrategie unterstützen und dabei sowohl technische als auch wirtschaftliche sowie nutzerzentrierte Anforderungen berücksichtigen. Damit beantwortet diese Arbeit die zentrale Forschungsfrage “Welche technologischen, wirtschaftlichen und nutzerbezogenen Kriterien sind für die Wahl zwischen nativer App, Web-App und hybrider App entscheidend, und welche Best Practices lassen sich daraus ableiten?”, indem sie die entscheidenden Kriterien nennt und praxisorientiert umsetzt. Für die konkrete Entscheidungsfindung sollten jedoch stets individuelle Faktoren wie zum Beispiel branchenspezifische Regularien, Team-Expertise oder langfristige Roadmaps zusätzlich analysiert werden, die in dieser Arbeit nicht berücksichtigt wurden. Die vorgestellte Systematik bietet somit eine fundierte erste Einschätzung, ersetzt aber keine projektabhängige, umfassende Recherche.

6.3 Ausblick

Die Ergebnisse dieser Arbeit liefern eine Grundlage für die technologische Entscheidungsfindung und verdeutlichen zudem die vielen Faktoren, die dabei eine Rolle spielen. In einer zunehmend digitalisierten Welt, in der eine große Menge von Unternehmen und Entwickler*innen vor der Herausforderung stehen, ihre Anwendungen für eine breite Nutzerbasis optimal bereitzustellen, trägt diese Forschung dazu bei, zukunftssichere Strategien abzuleiten.

Die rasante Entwicklung im Bereich der Web- und App-Technologien wird die Entscheidungsfindung zwischen nativen, hybriden und webbasierten Anwendungen weiterhin beeinflussen. Fortschritte in der Browser-Technologie und die zunehmende Verbreitung von Progressive Web Apps (PWAs) könnten die Lücke zur nativen Performance weiter verringern. Gleichzeitig entwickeln sich Cross-Plattform-Frameworks stetig weiter, um die Balance zwischen plattformspezifischer Optimierung und effizienter Entwicklung zu verbessern. Hierbei stellt sich zudem die Frage, ob sich die Entwicklung nativer Apps in Zukunft noch lohnt, wenn Cross-Plattform- und Web-Lösungen die technologischen

[12] Dieser Text wurde mit Unterstützung von KI geschrieben.

Unterschiede zunehmend ausgleichen und mit einem geringeren Entwicklungsaufwand verbunden sind.

Zukünftige Forschung könnte sich darauf konzentrieren, wie KI-gestützte Entwicklungsansätze oder neue Webstandards den Entscheidungsprozess weiter verändern. Besonders relevant wird die Frage, inwiefern sich Web- und Cross-Plattform-Technologien weiter an native Anwendungen annähern und welche Auswirkungen dies auf langfristige Entwicklungsstrategien hat. Letztlich bleibt die Wahl der richtigen Technologie ein dynamischer Prozess, der sich an den technischen Fortschritt und wandelnde Nutzererwartungen anpassen muss.

Anhang

Anhang A: Umfrage Nutzerverhalten

A.1: Umfrageergebnisse Nutzerverhalten

Die gesammelten Umfrageergebnisse sind in der Datei Anhang/Umfrage-Nutzerverhalten-Ergebnisse.csv zu finden.

A.2: Entworfene Umfrage Nutzerverhalten

Die entworfene Umfrage in Bezug auf Nutzerverhalten ist in der Datei Anhang/Umfrage.pdf zu finden.

Anhang B: Tool zur Entwicklungsentscheidung

Die vollständige kompilierte Version der entwickelten Web-Anwendung befindet sich im Ordner Anhang/tool/ und kann mit einem lokalen Web-Server ausgeführt werden.

Quellenverzeichnis

[1] IT Portal 24: Flutter vs React Native – Was ist besser für 2025? https://www.itportal24.de/ratgeber/flutter-vs-react-native. Abgerufen am 17.1.2025.

[2] android: Create your first Android app. https://developer.android.com/codelabs/basic-android-kotlin-compose-first-app#0. Abgerufen am 5.1.2025.

[3] android: Erste Schritte mit der Play Console. https://support.google.com/googleplay/android-developer/answer/6112435?hl=de#zippy=schritt-bezahlen-sie-die-registrierungsgebuehr. Abgerufen am 6.1.2025.

[4] android: Kotlin and Android. https://developer.android.com/kotlin?hl=de. Abgerufen am 5.1.2025.

[5] angular: angular. https://angular.dev/. Abgerufen am 5.1.2025.

[6] Apple: Abrechnung und Abonnements. https://support.apple.com/de-de/billing. Abgerufen am 29.1.2025.

[7] Apple: App Store Small Business Program. https://developer.apple.com/app-store/small-business-program/. Abgerufen am 19.1.2025.

[8] Apple: Zahlungsmethode zu deinem Apple Account. https://support.apple.com/de-de/118429. Abgerufen am 28.1.2025.

[9] Sriram Santosh Aripirala, Pranay Airan und Diwakar Reddy Peddinti: The Polyglot's Playground: Navigating KMP, Flutter, and React Native in Cross-Platform Ecosystems. In: International Journal for Research in Applied Science and Engineering Technology, S. 176. (2024). DOI: 10.22214/ijraset.2024.64492.

[10] Adam Bar: What Web Can Do Today. https://whatwebcando.today/. Abgerufen am 28.12.2024.

[11] Tim Berners-Lee, Roy T. Fielding und Larry M Masinter: Uniform Resource Identifier (URI): Generic Syntax. In: Request for Comments 3986, S. 16–18. RFC Editor (Jan. 2005). DOI: 10.17487/RFC3986.

[12] Natesh Bhat: Developing packages plugins. https://nateshmbhat.medium.com/flutter-vs-react-native-performance-benchmarks-you-cant-miss-ïÿŕ-2e31905df9b4. Abgerufen am 1.1.2025.

[13] Stefan Bosnic, István Papp und Sebastian Novak: The development of hybrid mobile applications with Apache Cordova. In: 2016 24th Telecommunications Forum (TELFOR), S. 2. (Nov. 2016). DOI: 10.1109/TELFOR.2016.7818919.

[14] Bryj: Mobile Apps: The Gold Standard for User Experience and Better ROI. https://www.bryj.ai/mobile-apps-the-gold-standard-for-user-experience-and-better-roi/. Abgerufen am 29.1.2025.

[15] Michał Chudziak: react-native-community/geolocation. https://www.npmjs.com/package/@react-native-community/geolocation. Abgerufen am 02.01.2025.

[16] comScore: Number of app downloads per month of smartphone users in the United States as of June 2017. https://www.statista.com/statistics/325926/monthly-app-downloads-of-us-smartphone-users/. Abgerufen am 11.1.2025.

[17] Apple Developer: App Dev Tutorials. https://developer.apple.com/tutorials/app-dev-training/. Abgerufen am 5.1.2025.

[18] Apple Developer: App Review Guidelines. https://developer.apple.com/app-store/review/guidelines/. Abgerufen am 6.1.2025.

[19] Apple Developer: Choosing a Membership. https://developer.apple.com/support/compare-memberships/. Abgerufen am 6.1.2025.

[20] Apple Developer: Human Interface Guidelines. https://developer.apple.com/design/human-interface-guidelines. Abgerufen am 10.1.2025.

[21] Apple Developer: Swift. https://developer.apple.com/swift/. Abgerufen am 5.1.2025.

[22] Apple Developer: XCode - Apple Developer. https://developer.apple.com/xcode/. Abgerufen am 3.1.2025.

[23] Android Google Developers: Android App Links. https://developer.android.com/training/app-links?hl=de. Abgerufen am 31.12.2024.

[24] Android Google Developers: Plattformarchitektur. https://developer.android.com/guide/platform?hl=de. Abgerufen am 27.12.2024.

[25] Nidhi Dhiman, Ankush Choudhary und Satish Chaudhary: Review on Comparative Study of Flutter App and Android App. In: International Journal for Research in Applied Science and Engineering Technology, S. 2081. (2023).

[26] Veikko Krypczyk Dirk Mittmann: React Native im Überblick. https://entwickler.de/react/react-native-im-uberblick. Abgerufen am 27.12.2024.

[27] finito e.U.: Eidesstaatliche Erklärung. https://www.finito24.de/eidesstattliche-erklaerung/. Abgerufen am 26.2.2025.

[28] Taras Fedynyshyn, Olha Mykhaylova und Ivan Opirskyy: Security Implications of Mobile Development Frameworks: Findings from Static Analysis of Android Apps. In: S. 1–4. (Okt. 2024). DOI: 10.1109/TCSET64720.2024.10755684.

[29] feedbax: Native App vs. Cross-Platform App vs. PWA: Which is the right choice for you? https://feedbax.ai/blog/native-app-vs-cross-plattform-app-vs-pwa. Abgerufen am 15.1.2025.

[30] Flutter: Flutter architectural overview. https://docs.flutter.dev/resources/architectural-overview. Abgerufen am 30.12.2024.

[31] Flutter: Start building Flutter Android apps on Windows. https://docs.flutter.dev/get-started/install/windows/mobile. Abgerufen am 5.1.2025.

[32] Google Flutter: Developing packages plugins. https://docs.flutter.dev/packages-and-plugins/developing-packages. Abgerufen am 31.12.2024.

[33] Google Flutter: Flutter Geolocator Plugin. https://pub.dev/packages/geolocator. Abgerufen am 02.01.2025.

[34] Google Flutter: The official package repository for Dart and Flutter apps. https://pub.dev/. Abgerufen am 31.12.2024.

[35] Mozilla Foundation: HTTP. https://developer.mozilla.org/de/docs/Web/HTTP. Abgerufen am 27.12.2024.

[36] Mozilla Foundation: MediaDevies - Web APIs. https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices. Abgerufen am 28.12.2024.

[37] Mozilla Foundation: PWAs installierbar machen. https://developer.mozilla.org/de/docs/Web/Progressive_web_apps/Guides/Making_PWAs_installable. Abgerufen am 3.1.2025.

[38] Mozilla Foundation: Was ist eine progressive Web-App? https://developer.mozilla.org/de/docs/Web/Progressive_web_apps/Guides/What_is_a_progressive_web_app. Abgerufen am 3.1.2025.

[39] Mozilla Foundation: Web APIs. https://developer.mozilla.org/en-US/docs/Web/API. Abgerufen am 28.12.2024.

[40] Mozilla Foundation: Web Authentication API - Web APIs. https://developer.mozilla.org/en-US/docs/Web/API/Web_Authentication_API. Abgerufen am 28.12.2024.

[41] Mozilla Foundation: Web Bluetooth API - Web APIs. https://developer.mozilla.org/de/docs/Web/API/Web_Bluetooth_API. Abgerufen am 28.12.2024.

[42] Mozilla Foundation: Web-Technologien für Entwickler. https://developer.mozilla.org/de/docs/Web. Abgerufen am 27.12.2024.

[43] Gebhard-Webdesign: Einführung Web Technologien. https://www.webtechnologien.com/basic-tutorials/html/einfÃijhrung/. Abgerufen am 27.12.2024.

[44] Google: Android Studio. https://developer.android.com/studio?hl=de. Abgerufen am 3.1.2025.

[45] Google: Eine Google Play-Zahlungsmethode hinzufügen, entfernen oder bearbeiten. https://support.google.com/googleplay/answer/4646404?sjid=15007005276253071761-EU. Abgerufen am 28.1.2025.

[46] Google: Service fees. https://support.google.com/googleplay/android-developer/answer/112622?sjid=4175626678853687051-EU. Abgerufen am 19.1.2025.

[47] Google: Zahlungen und Abos. https://myaccount.google.com/payments-and-subscriptions. Abgerufen am 29.1.2025.

[48] Ruben Horn u. a.: Native vs Web Apps: Comparing the Energy Consumption and Performance of Android Apps and their Web Counterparts. In: S. 7–8. (Mai 2023). DOI: 10.1109/MOBILSoft59058.2023.00013.

[49] Apple Inc.: Develop apps for Apple platforms. https://developer.apple.com/tutorials/app-dev-training/. Abgerufen am 27.12.2024.

[50] Vatche Isahagian: Strategic and operational services for workload management in the cloud. In: S. 3–4. (2013).

[51] jQuery: jQuery. https://jquery.com/. Abgerufen am 5.1.2025.

[52] Mohamed Lachgar und Abdali Abdelmounaim: Decision Framework for Mobile Development Methods. In: International Journal of Advanced Computer Science and Applications 8, S. 111. (Feb. 2017). DOI: 10.14569/IJACSA.2017.080215.

[53] Yun Ma u. a.: A Tale of Two Fashions: An Empirical Study on the Performance of Native Apps and Web Apps on Android. In: IEEE Transactions on Mobile Computing 17.5, S. 991. (Mai 2018). DOI: 10.1109/TMC.2017.2756633.

[54] Microsoft: Visual Studio Code. https://code.visualstudio.com/. Abgerufen am 3.1.2025.

[55] Onilab: 20+ PWA Examples That Will Inspire You to Build Your Own in 2024. https://onilab.com/blog/20-progressive-web-apps-examples. Abgerufen am 3.1.2025.

[56] Stack Overflow: Developer Roles - Developer Types. https://survey.stackoverflow.co/2024/developer-profile/#developer-roles. Abgerufen am 16.1.2025.

[57] Stack Overflow: Most popular technologies. https://survey.stackoverflow.co/2024/technology#most-popular-technologies-webframe. Abgerufen am 5.1.2025.

[58] Stack Overflow: Top paying Technologies. https://survey.stackoverflow.co/2024/technology/#top-paying-technologies. Abgerufen am 15.1.2025.

[59] Paypal: PayPal Developer. https://developer.paypal.com/home/. Abgerufen am 20.1.2025.

[60] React: React. https://react.dev/. Abgerufen am 5.1.2025.

[61] Caroline Roberts u. a.: Response Burden and Dropout in a Probability-Based Online Panel Study – A Comparison between an App and Browser-Based Design. In: Journal of Official Statistics 38.4, S. 1009. (2022). DOI: 10.2478/jos-2022-0043.

[62] Droids on Roids: Flutter vs. React Native – Which is Better for Your Project? https://www.thedroidsonroids.com/blog/flutter-vs-react-native-comparison. Abgerufen am 17.1.2025.

[63] Amazon Web Services: Was ist der Unterschied zwischen Webanwendungen, nativen Apps und Hybrid-Apps? https://aws.amazon.com/de/compare/the-difference-between-web-apps-native-apps-and-hybrid-apps/. Abgerufen am 28.12.2024.

[64] simform: App Usage Statistics 2022 that'll Surprise You (Updated). https://www.simform.com/blog/the-state-of-mobile-app-usage. Abgerufen am 11.1.2025.

[65] Daniel Sogl: Cordova Community Plugins. https://danielsogl.gitbook.io/awesome-cordova-plugins. Abgerufen am 31.12.2024.

[66] Daniel Sogl: Geolocation. https://danielsogl.gitbook.io/awesome-cordova-plugins/geolocation. Abgerufen am 02.01.2025.

[67] Meta Open Source: React Native Landing Page. https://reactnative.dev/. Abgerufen am 27.12.2024.

[68] StatCounter: Desktop vs Mobile vs Tablet vs Console Market Share Worldwide. https://gs.statcounter.com/platform-market-share#monthly-202301-202401. Abgerufen am 10.12.2024.

[69] StatCounter: Market share of mobile operating systems worldwide from 2009 to 2024, by quarter. https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/. Abgerufen am 16.1.2025.

[70] Statista: Number of mobile app downloads worldwide from 2016 to 2023. https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/. Abgerufen am 10.12.2024.

[71] Statista: Revenue of mobile apps worldwide 2019-2027. https://www.statista.com/forecasts/1262892/mobile-app-revenue-worldwide-by-segment. Abgerufen am 10.12.2024.

[72] Statista: Time spent with nonvoice activities on mobile phones every day in the United States from 2019 to 2024. https://www.statista.com/statistics/1045353/mobile-device-daily-usage-time-in-the-us/. Abgerufen am 10.12.2024.

[73] Stripe: Stripe | Online Bezahldiesnt. https://stripe.com/de. Abgerufen am 20.1.2025.

[74] sublime: Sublime Text Editor. https://www.sublimetext.com/. Abgerufen am 3.1.2025.

[75] The React Team: New Architecture is here. https://reactnative.dev/blog/2024/10/23/the-new-architecture-is-here. Abgerufen am 30.12.2024.

[76] The React Team: React Native Directory. https://reactnative.directory/. Abgerufen am 31.12.2024.

[77] The React Team: Who is using React Native? https://reactnative.dev/showcase. Abgerufen am 17.1.2025.

[78] Lionel Sujay Vailshery: Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2023. https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/. Abgerufen am 22.12.2024.

[79] YouGov: 48 market research – Two in five consumers don't use ad blockers. https://business.yougov.com/content/49122-48-market-research-two-in-five-consumers-dont-use-ad-blockers. Abgerufen am 29.1.2025.