

# BACHELOR-THESIS

Fachrichtung Wirtschaftsinformatik

## Chancen und Risiken einer Migration von Webanwendungen zu nativen Systemen

<b>Studiengruppe:</b>	119 WINF
<b>Eingereicht von:</b>	Fabian Reitz Hasseldieksdammer Weg 13 24114 Kiel +49 175 6392445 fabian.reitz@stud.dhsh.de
<b>Erstgutachter DHSH:</b>	Prof. Dr. Alexander Paar Hans-Detlev-Prien-Straße 10 24106 Kiel +49 431 3016255 alexander.paar@dhsh.de
<b>Gutachter des Betriebes:</b>	Marc Köster Mittelstraße 7   Hinterhaus 24103 Kiel +49 431 53015400 koester@stadtwerk.org
<b>Zweitgutachter DHSH:</b>	Prof. Dr. Michael Sachtler Hans-Detlev-Prien-Straße 10 24106 Kiel +49 431 3016170
<b>Abgabetermin:</b>	16.05.2022

## Inhaltsverzeichnis

<b>Inhaltsverzeichnis</b>	<b>II</b>
<b>Abkürzungsverzeichnis</b>	<b>IV</b>
<b>Abbildungsverzeichnis</b>	<b>V</b>
<b>Tabellenverzeichnis</b>	<b>VI</b>
<b>1 Einführung</b>	<b>1</b>
1.1 Einleitung	1
1.2 Problemstellung	4
1.3 Zielsetzung	6
1.4 Aufbau und Vorgehensweise	7
<b>2 Möglichkeiten zur Entwicklung von Anwendungen</b>	<b>8</b>
2.1 Vor- und Nachteile von Webanwendung	8
2.1.1 Update-Frequenz	10
2.1.2 Entwicklung	10
2.1.3 Monetarisierung	14
2.1.4 Schaffen und Konsumieren von Inhalten	14
2.1.5 User Experience	16
2.1.6 Performanz	19
2.2 Vor- und Nachteile von nativen Anwendungen	21
2.2.1 Update-Frequenz	23
2.2.2 Entwicklung	24
2.2.3 Monetarisierung	27
2.2.4 Schaffen und Konsumieren von Inhalten	27
2.2.5 User Experience	28
2.2.6 Performanz	28
2.3 Cross-Plattform	30
2.3.1 Unterstützte Plattformen	30
2.3.2 Gängige Frameworks	31
2.3.2.1 React Native	31
2.3.2.2 Electron	32
2.3.2.3 Flutter	32
<b>3 Praktischer Vergleich von Cross-Plattform-Lösungen</b>	<b>33</b>
3.1 Zielvorgabe	33

3.2 Vergleich der Umsetzungen . . . . .	35
<b>4 Diskussion . . . . .</b>	<b>35</b>
4.1 Beleg der Hypothese . . . . .	35
4.2 Limitation . . . . .	35
4.3 Ausblick . . . . .	35
<b>5 Fazit . . . . .</b>	<b>35</b>
<b>Liteaturverzeichnis . . . . .</b>	<b>VI</b>
<b>Eidesstattliche Erklärung . . . . .</b>	<b>IX</b>
<b>Anhang . . . . .</b>	<b>X</b>
Anhang 1: Tabellen . . . . .	X
Anhang 1.1: stackoverflow Developer Surveys 2015 bis 2021 . . . . .	X
Anhang 1.2: Hard- und Softwarespezifikationen des Testsystems . . . . .	XI
Anhang 1.3: Ergebnisse des Tests zur Erfassung der Browser-Responsivität von Safari und Chrome . . . . .	XI
Anhang 1.4: Hardware des Windows-Systems . . . . .	XII
Anhang 2: Abbildungen . . . . .	XIII
Anhang 2.1: facebook empfiehlt native Smartphone-App . . . . .	XIII

## Abkürzungsverzeichnis

API	Application Programming Interface deutsch: Programmierschnittstelle
CP	Cross-Plattform deutsch: plattformübergreifend
CSS	Cascading Style Sheets
DOM	Document Object Model
HTML	Hypertext Markup Language
IDE	Integrated Development Environment deutsch: integrierte Entwicklungsumgebung
UI	User Interface deutsch: Nutzungsoberfläche
W3C	World Wide Web Consortium deutsch: Internet Konsortium
Webapp	Web Application deutsch: Webanwendung
WLAN	Wireless Local Area Network deutsch: drahtloses lokales Netzwerk

## Abbildungsverzeichnis

1	Anteil der Full-Stack Entwicklerinnen und Entwickler von 2013 bis 2021 . . . . .	4
2	Nutzende von facebook auf dem Smartphone nach Plattform .	16
3	Ausgewählte Endgeräte mit unregelmäßigen Displayformaten .	25
4	Das Display von Apples MacBook Pro 2021 . . . . .	26
5	MockUp der zu entwickelnden Anwendung . . . . .	35
6	Während der ersten Schritte der Registrierung auf facebook . .	XIII

## Tabellenverzeichnis

1	Ausgewählte Standards und zugehörige Status der W3C . . . .	11
2	Ausgewählte Browser mit Unterstützung für asynchrone JavaScript-Funktionen . . . . .	12
3	Ausgewählte Browser mit Unterstützung für CSS Masks . . . .	13
4	Ausgewählte Browser mit Unterstützung für Web Notifications .	18
5	Ausgewählte Browser mit Unterstützung für das <code>contextmenu</code> -Event . . . . .	19
6	Ausgewählte Betriebssysteme und die zugehörigen Programmiersprachen für native Entwicklung . . . . .	22
7	Ausgewählte Cross-Plattform-Frameworks und ihre Zielplattformen . . . . .	33
8	Versionen der Zielbetriebssysteme . . . . .	34
9	Anzahl der Antworten der stackoverflow Developer Surveys 2015 bis 2021 . . . . .	X
10	Hard- und Softwarespezifikationen des Testsystemes für den Vergleich der Leistungsfähigkeit von Safari und Chrome . . . .	XI
11	Ergebnisse des Tests zur Erfassung der Browser-Responsivität von Safari und Chrome mittels <i>speedometer</i> . . . . .	XI
12	Hardwarespezifikationen des Windows-Systems zum Testen der entwickelten Anwendung . . . . .	XII

# 1 Einführung

## 1.1 Einleitung

Während es unmöglich ist, den Ursprung des Internets auf einen exakten Zeitpunkt festzulegen, lässt sich jedoch mit Gewissheit sagen, dass die Entwicklung des Internets einen bedeutsamen Wendepunkt in der Geschichte der Menschheit darstellt (Kleinrock 2010: 26). Floridi (2010) sieht in dem Internet als „Infosphere“ (Floridi 2010: 9) die vierte Revolution in einer Reihe von weltverändernden Wandlungen. Zu diesen Wandlungen gehören die Kopernikus-Revolution, die Darwin'sche Revolution und die Freud'sche Revolution. Diese Wenden veränderten das grundlegende Verständnis der Menschen sowohl über ihre Umwelt als auch über sich selbst (Floridi 2010: 8f.).

Das Internet befindet sich in einem stetigen Wandel. Der Beginn des modernen Internets wird im Kontext dieser Arbeit auf den Zeitpunkt datiert, als Sir Tim Berners-Lee die ersten Webkomponenten im Jahr 1990 entwickelte. Berners-Lee arbeitete zu diesem Zeitpunkt bei *CERN* in der Schweiz. Er entwickelte ein System zur Verwaltung von unternehmensinternen Information mittels des ersten Webbrowsers. Dieser war in der Lage, HTML-Dokumente von einem durch Berners-Lee entwickelten Webserver abzurufen und darzustellen (Berners-Lee 1990).

In der Geschichte des Internets finden sich einige Trends und Entwicklungen, welche als Meilensteine gesehen werden. Diese teilen das Internet historisch in Web 1.0, Web 2.0, Web 3.0 und Web 4.0 auf (Kollmann 2020: 133).

Den Beginn der Geschichte des Internets bildet das Web 1.0. Dieses basiert überwiegend auf den Ideen von Berners-Lee und wird durch die Etablierung des Internets in der Gesellschaft erweitert. Somit besteht das Web 1.0 lediglich aus statischen HTML-Seiten, welche den Nutzenden in einem Webbrowser angezeigt wird. Der dominierende Webbrowser der Neunzigerjahre ist der *Netscape Navigator* des Unternehmens *Netscape Communications* (O'Reilly 2005). Besonders für das Web 1.0 ist die binäre Rollenverteilung der Nutzenden, wie Kollmann beschreibt:

„Zum einen gab es aktive Ersteller von Web-Inhalten, die, teils kommerziell, teils privat, Informationen einstellten und publizier-

ten. Zum anderen gab es passive Konsumenten, die sich lediglich die bereitgestellten Inhalte ansehen konnten und auch gar keine andere Option hatten, als die Informationen zu empfangen und zu konsumieren“ (Kollmann 2020: 134).

Dieser Passivität der Konsumierenden sind sich die Unternehmen der Zeit des Web 1.0 ebenfalls bewusst. So zeigen sich im Web 1.0 die ersten Ansätze von ausgefeilten E-Commerce-Strategien, welche darauf abzielen, Produkte und Dienstleistungen auf diesem neuen Markt zu vertreiben (Kollmann und Lomberg 2010: 1204).

Das auf das Web 1.0 folgende Web 2.0 entstand um 2005 und markierte somit die Zeit, als die dot-com-Blase geplatzt ist. Diese stetige Wende ist aus einer Konferenz zwischen den Unternehmen *O'Reilly* und *MediaLive International*. Die Idee einer nächsten Evolutionsstufe des Internets kam den Unternehmen bei einem Brainstorming. Dieses Brainstorming brachte letztendlich die *Web 2.0 Conference* hervor. Hierbei muss Erwähnung finden, dass Unternehmen das Buzzword *Web 2.0* als Marketing-Element missbrauchen. Das erschwert die Einordnung des Web 2.0 umso mehr, da viele dieser Unternehmen nichts mit den Definitionsansätzen der *Web 2.0 Conference* gemein haben. Die *Web 2.0 Conference* versucht sich an einer Definition über zentrale Aspekte dieser neuen Iterationsstufe des Internets. Dazu gehören die Ansätze *Web as a Platform* und *User-Generated Content*. Die Rolle der passiven Konsumierenden des Web 1.0 veränderte sich demnach zu den aktiven Teilnehmenden des Web 2.0. Erste soziale Medien ermöglichen eine Interaktion mit anderen Nutzenden, Bewertungen auf E-Commerce-Seiten verschaffen Käuferinnen und Käufern eine Stimme und digitale Enzyklopädien laden zum Teilen des eigenen Wissens ein. Zentrale Plattformen des Web 2.0 sind somit *Facebook*, *eBay* und *Wikipedia*. Anbieter einer Rich User Experience, vor allem *Google*, lösen die Riesen des Web 1.0, beispielsweise *Netscape*, ab (O'Reilly 2005). Dabei spielen die sieben Grundprinzipien des Web 2.0 eine zentrale Rolle: Globale Vernetzung, Kollektive Intelligenz, Datengetriebene Plattformen, Perpetual Beta, Leichtgewichtige Architekturen, Geräteunabhängigkeit und Reichhaltige Oberfläche (Kollmann und Häsel 2007, zitiert nach Kollmann 2020: 137).

Die Fortführung, der Ausbau und die allzeitliche Zugänglichkeit des Web 2.0

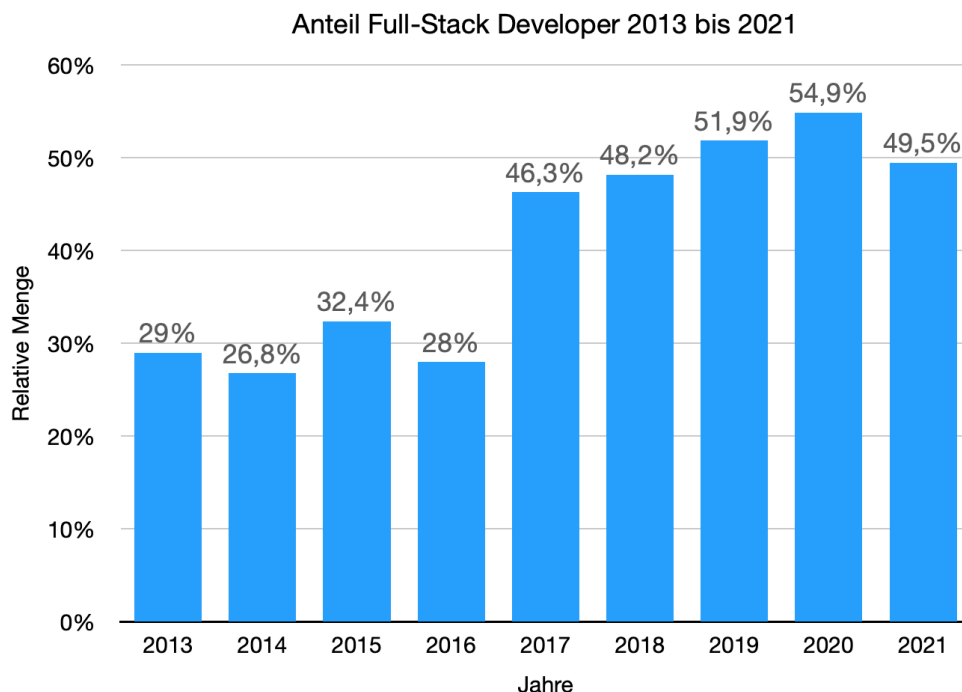


machen das Internet zu der modernen Infosphere, die Floridi 2010 beschrieb. Durch die Möglichkeiten, die Entwicklerinnen und Entwicklern gegeben werden, ist es mit wenig Aufwand möglich, ganze Anwendungen über einen Webbrowser zugänglich zu machen. Insbesondere Anwendungen, welche ursprünglich für native Systeme entwickelt wurden, finden ihren Weg in die Cloud. Welche Vor- und Nachteile dieser Trend hat und ob eine Remigration zu nativen Systemen unter Verwendung moderner Technologien Sinn ergibt, wird in dieser Arbeit näher beleuchtet.

## 1.2 Problemstellung

Die Entwicklung von Anwendungen für das Web wird eine immer beliebtere Alternative zu nativen Anwendungen, wie beispielsweise installierbare Desktop-Anwendungen. Deutlich erkennbar wird der Trend bei einem Vergleich der *stackoverflow Developer Surveys* aus den Jahren 2015 bis 2021. Betrachtet man nun die Antworten *Full-Stack Developer* und trägt diese als Funktion der Jahre auf, lässt sich folgender Graph erkennen:

Abbildung 1: Anteil der Full-Stack Entwicklerinnen und Entwickler von 2013 bis 2021



Der Graph zeigt den Anteil von Full-Stack Entwicklerinnen und Entwicklern an der Gesamtmenge (siehe Anhang 1.1) der Antworten der jährlichen *stackoverflow Developer Survey* (2015 *Developer Survey* o.D.; *Developer Survey Results 2016* o.D.; *Developer Survey Results 2017* o.D.; *Developer Survey Results 2018* o.D.; *Developer Survey Results 2019* o.D.; 2020 *Developer Survey* o.D.; 2021 *Developer Survey* o.D.).

Der Anteil an Full-Stack Entwicklerinnen und Entwicklern nahm im Jahr 2017 stark zu und erreichte einen relativen Wert von 46,3 %. Der Anteil stieg weiter und erreichte im Jahr 2020 den Höchststand mit 54,9 %. Im Folgejahr fiel der Wert gering. Mit einer zeitweise absoluten Mehrheit von Web-Developern unter den Teilnehmenden der Umfrage lässt sich eine Beliebtheit und Re-

levanz dieser Technologie schlussfolgern. Die steigende Anzahl zeigt auch, dass Web-Anwendungen über die Jahre an Relevanz gewonnen haben. Hier sei jedoch zu erwähnen, dass Full-Stack Developer nur einen Anteil der Teilnehmenden abbildet, die mit Webtechnologien arbeiten. Berufsbezeichnungen wie Front-End Developer oder Back-End Developer gehören ebenfalls zu den Webentwicklerinnen und -entwicklern, finden jedoch zur Minimierung der Komplexität hier keine Beachtung.

Webanwendungen bieten zahlreiche Vorteile gegenüber konventionellen nativen Anwendungen, sowohl aus Sicht der Entwickelnden als auch aus Nutzenden-Sicht. Hier sei beispielsweise die universelle Erreichbarkeit von jedem Endgerät zu erwähnen. Jedoch müssen ebenfalls die Nachteile bedacht werden, sollte sich für die Entwicklung von Software als Webanwendung entschieden werden. Hier kann als Beispiel die Vielzahl von Webbrowsern mit uneinheitlichen Standards erwähnt werden. Entwicklerinnen und Entwickler müssen sich über die Implikationen der Wahl zwischen nativen Anwendungen und Webanwendungen im Klaren sein, um den Entwicklungsaufwand gering und die Nutzungserfahrung positiv zu gestalten. Welche Aspekte Front-End Developer bei dieser Wahl beachten müssen und welche Lösungen moderne Technologien für dieses Problem bieten, wird in dieser Arbeit aufgezeigt. Zentrale Betrachtung findet dabei der Gedanke einer Remigration von Webanwendungen zu nativen Anwendungen vor dem Hintergrund der großen Zahl an Webentwicklerinnen und -entwicklern.

### **1.3 Zielsetzung**

Ziel dieser Arbeit ist die Dokumentation der Möglichkeiten, die sich Entwicklenden von Webanwendungen bieten, sollten sie eine Migration zu nativen Anwendungen anstreben. Abschließend soll der Versuch einer Vorhersage als begründete Vermutung getätigt werden, ob sich native Anwendungen im Anbetracht moderner Möglichkeiten im Vergleich zu Webanwendungen zukünftig durchsetzen werden.

#### **1.4 Aufbau und Vorgehensweise**

Um die Möglichkeiten der Entwicklung von Anwendungen bestmöglich zu vergleichen, werden zunächst die Browser mit den größten Marktanteilen verglichen. Die Betrachtung findet dabei sowohl aus Sicht der Nutzerinnen und Nutzer als auch aus der von Entwickelnden statt. Dabei wird ein besonderes Augenmerk auf die Unterschiede der Technologien gelegt. Darauf folgend werden die Betriebssysteme und ihre Eigenheiten verglichen, für welche native Software entwickelt werden soll. Auch hier werden die Marktführer der Betriebssysteme für Desktop und Smartphone beziehungsweise Tablet verglichen. Als letzter Bestandteil des Literature Reviews werden moderne Cross-Plattform-Technologien verglichen. Hierbei werden die Stärken und Schwächen der Lösungen verglichen, wie auch die Programmiersprache vor dem Hintergrund eines Webentwickelnden.

Im Fokus des Praxisteils dieser Arbeit steht der Vergleich der Technologien aus praktischer Sicht. Hierzu werden mithilfe der betrachteten Technologien Anwendungen erstellt, welche vorab definierte Anforderungen erfüllen. Die Ausführungen werden daraufhin anhand der Entwicklungserfahrung verglichen. Der praktische Teil erfolgt dabei vor dem Hintergrund eines Webentwickelnden.

## 2 Möglichkeiten zur Entwicklung von Anwendungen

### 2.1 Vor- und Nachteile von Webanwendung

Die steigende Zahl der Full-Stack Entwicklenden der letzten Jahre zeigt den Bedarf nach Webanwendungen aus Sicht der Unternehmen (*2020 Developer Survey* o.D.). Aus diesem Trend lässt sich die Vermutung ableiten, dass die Relevanz von Webanwendungen für die Nutzenden und Kunden dieser Unternehmen ebenfalls zunimmt. Diese Vermutung wird dadurch bestätigt, dass die zwanzig am häufigsten besuchten Websites aus November 2021 Webanwendungen sind (Clement 2022). Beim Betrachten derartiger Statistiken stellt sich die Frage, ab wann eine Website eine Webanwendung ist. Zur Klärung folgen nun Definitionen der gängigen Begriffe für diese Arbeit:

**Website:** Eine Website ist eine Sammlung von aufrufbaren Webseiten. Sie bildet einen Internetauftritt ab.

**Webseite:** Eine Webseite ist ein einzeln aufrufbarer Bestandteil einer Website und bildet einen Zustand einer Ansicht im Webbrowser ab.

Eine Webanwendung ist eine besondere Form einer Website. Eine Website ist im ursprünglichen Ansatz eine Sammlung von HTML-Dokumenten. Diese werden von einem Server über das HTTP-Protokoll an einen Client, meist ein Webbrowser, übertragen. So besteht hier nur eine Richtung des Datenaustauschs: von dem Server zum Client. Eine Webanwendung setzt einen Server voraus, welcher Businesslogik besitzt. Dazu werden Schnittstellen, sogenannte APIs, von dem Client angesprochen. Die APIs eines Webservers ermöglichen eine bidirektionale Kommunikation. So kann eine nutzende Person Daten von dem Server empfangen und an diesen übermitteln. Der Server verarbeitet die Daten und bietet dem Nutzenden eine Webseite mit dynamischem Inhalt an.

Praktische Beispiele von Webanwendungen sind jederzeit im Internet einsehbar, so auch die eingangs erwähnten meistbesuchten Websites im November 2021. Diese umfassen *google.com* auf dem ersten Platz mit 45,41 Milliarden monatlichen Aufrufen, gefolgt von *youtube.com* und *facebook.com* (Clement 2022).

Ein Frontend einer Webanwendung bedient sich den drei grundsätzlichen Programmier- und Markup-Sprachen des Webs: *HTML*, *CSS* und *JavaScript*. Es ist möglich, jedoch ineffizient, eine Webanwendung nur mit diesen Technologien zu entwickeln. Moderne Frameworks und Libraries vereinfachen Probleme und Herausforderungen und sorgen so für eine angenehmere Entwicklungserfahrung und eine höhere Entwicklungsgeschwindigkeit. Zu den bekanntesten Technologien gehören unter anderem *React.js*, *Angular* und *Vue.js* (Clement 2022). Webframeworks und -libraries vereinfachen den Aufwand der Entwicklung durch die zentrale Nutzung von *JavaScript* und *TypeScript*, wobei *CSS* und *HTML* durch technologiespezifische Aspekte erweitert werden.

Ziel der Nutzung von Webframeworks und -libraries ist eine einfache Kommunikation mit einem Backend oder anderen APIs, die Abdeckung einer Vielzahl von Browsern und letztendlich das Rendern von *HTML*, *CSS* und clientseitigem *JavaScript* im Browser der nutzenden Person. Welche Vor- und Nachteile Webanwendungen verglichen mit nativen Technologien mit sich bringen, wird im Folgenden erläutert.

Jobe (2013: 27) begründet den rapiden Fortschritt von Webanwendungen mit der Etablierung der Technologie *HTML5*. *HTML5* und die verwandten Technologien *CSS3* und *JavaScript* sorgen dafür, dass Webapps mit nativen Anwendungen in den Punkten Funktionalität, Design, Interaktion und Einsatz von Multimedia konkurrieren können. Als besondere Vorteile erwähnt Jobe (2013: 28) die Update-Frequenz und die Entwicklung als Ganzes. Der Autor sieht in der Monetarisierung, verglichen mit nativen Anwendungen, Nachteile, da für das Web keine einheitlichen Monetarisierungsstrategien existieren. Da sich dieser Aspekt seit der Veröffentlichung der Arbeit jedoch weiterentwickelt hat, wird dieser Punkt ebenfalls betrachtet.

Nachteile sieht Jobe (2013: 28) in den Punkten Schaffen und Konsumieren von Inhalten, User Experience und Performanz. Hierbei ist auffällig, dass die Anzahl der Nachteile überwiegt. Wie kritisch diese Nachteile jedoch sind, und ob die Vorteile dennoch überwiegen, wird im Folgenden unter modernen Gesichtspunkten erläutert.

### 2.1.1 Update-Frequenz

Jobe (2013: 28) gibt die Frequenz, mit welcher Updates für eine Webanwendung geliefert werden nicht direkt als Vorteil an, sondern vergleicht lediglich den formalen Update-Vorgang von nativen Programmen über beispielsweise App-Stores mit dem informalen Update-Möglichkeiten von Webanwendungen. Bei nativen Anwendungen muss in diesem Kontext Erwähnung finden, dass der Nutzerin oder dem Nutzer in der Regel die Entscheidung überlassen wird, ob die installierte Software aktualisiert werden soll. Diese Wahl impliziert jedoch die Gefahr, Sicherheitsupdates zu verpassen. Aus Sicht eines Entwicklers sollte darauf geachtet werden, dass der Nutzende auf wichtige Sicherheitsupdates aufmerksam gemacht wird. Sollte sich für eine native Softwarelösung basierend auf App-Stores entschieden werden, besteht die Möglichkeit, einen Changelog zu führen und so die Nutzenden vor einer Aktualisierung über Änderungen zu informieren.

Webanwendungen hingegen können jederzeit aktualisiert werden, ohne dass Nutzende informiert oder um Genehmigung gebeten werden müssen. Vorteile dieses Ansatzes sind beispielsweise die User Experience einer stets aktuellen Software, es werden keine Daten heruntergeladen und installiert, was abhängig von der Bandbreite und der Speichergeschwindigkeit längere Zeit in Anspruch nehmen kann, und zuletzt profitieren Nutzenden schnell von Sicherheitsupdates. Bei äußerst kritischen Problemen der Software, kann der Zugang zu den Webservern temporär gesperrt werden während Wartungsarbeiten durchgeführt werden. Auf diese Weise kann sichergestellt werden, dass in dem Moment keine Person Zugriff auf die Software hat.

### 2.1.2 Entwicklung

Wie bereits eingangs erwähnt, bilden die Standards *HTML5*, *CSS3* und *JavaScript* die Grundlage der modernen Webentwicklung. Sollte sich demnach für die Entwicklung der Anwendung als Webapp entschieden werden, sollten die Entwickelnden diese Technologien beherrschen. Für besondere Anwendungsfälle pflegt das *World Wide Web Consortium*, kurz W3C, eine Liste von verfügbaren Standards und Technologien auf dem Weg zur Standardisierung (*STANDARDS* o.D.). Im Folgenden wird eine Auswahl von anwendungsbezogenen Webtechnologien gezeigt, welche die Erfahrungen für Nutzende und Entwickelnde verbessern können. Diese sind nach ihren Status durch die



W3C aufgeteilt:

Tabelle 1: Ausgewählte Standards und zugehörige Status der W3C

W3C Recommendations	Proposed Recommendations	Candidate Recommendations	Working Draft
MathML	Payment Request API	WebXR Device API	Picture-in-Picture
SVG	Geolocation API	Media Capture and Streams	Web GPU
Server-Sent Events		Accelerometer	Geolocation Sensor
HTML5 Web Messaging		Gyroscope	

Die aufgelisteten Technologien haben sehr spezielle Anwendungsgebiete.

*MathML* ist beispielsweise eine Markup-Sprache zum Erstellen mathematischer Formeln, etwa wie mit  $\text{\LaTeX}$ . Im Kontext dieser Arbeit sind jedoch native APIs, wie *Accelerometer* oder *Gyroscope*, von besonderer Wichtigkeit, da sie die Fähigkeiten und Einsatzgebiete von Webanwendungen mit denen von nativen Anwendungen verschwimmen lassen (*STANDARDS* o.D.).

Wie die Standards und die als Standard antizipierten Technologien des W3C zeigen, werden Merkmale nativer Software ihren Weg in den Webbrowser finden. Die Abstraktion dieser über HTML, CSS und JavaScript vereinfacht dabei die plattform- und browserunabhängige Entwicklung der gewünschten Software, vorausgesetzt die Browser unterstützen den Standard.

Hierbei zeigt sich ein weiterer Gesichtspunkt von Webanwendungen: Browserkompatibilität. Trotz der Versuche des W3C, das Internet und seine Technologien zu standardisieren, sind nicht alle Browser gleich. Unterstützte Technologien und die Umsetzung dieser unterscheiden sich nicht nur von Browser zu Browser, sondern sogar von Version zu Version. Als Beispiel seien hier asynchrone JavaScript-Funktionen zu betrachten.

Asynchronität wird von JavaScript genutzt, um Code mit Zeitversatz auszu-

führen, wie zum Beispiel API-Abfragen. Da ungewiss ist, wann die Antwort von der API zurückgegeben wird, wird die Abfrage in eine asynchrone Funktion geschrieben, gekennzeichnet durch die Schlüsselworte `async` und `await`. Da diese Abfrage ein `Promise` zurückgibt, ermöglicht eine asynchrone Funktion das Behandeln der `Promises` als synchrones Objekt. Die Unterstützung dieser Funktionalität kann nun über beispielsweise *CanIUse* überprüft werden:

Tabelle 2: Ausgewählte Browser mit Unterstützung für asynchrone JavaScript-Funktionen

Browser	Version	wird unterstützt
Internet Explorer	11	nein
Edge	101	ja
Firefox	99	ja
Chrome	101	ja
Safari	15.4	ja
Safari on iOS	15.4	ja
Android Browser	101	ja

Schon bei häufig genutzten JavaScript-Schlüsselworten, wie beispielsweise `async` oder `await`, unterscheiden sich die Browser (*Async Functions* o.D.).

Wird nun der gestalterische Aspekt über CSS einer Webanwendung betrachtet, werden ebenfalls Unterschiede offensichtlich. Als Beispiel lässt sich die CSS-Methode `Masks` betrachten. Sie ermöglicht das Einbinden von Bildern als Maske über einem Hintergrund. Die Unterstützung der Browser wird im Vergleich deutlich:

Tabelle 3: Ausgewählte Browser mit Unterstützung für CSS Masks

Browser	Version	wird unterstützt
Internet Explorer	11	nein
Edge	101	teilweise
Firefox	99	ja
Chrome	101	teilweise
Safari	15.4	ja
Safari on iOS	15.4	ja
Android Browser	101	teilweise

Dieser Anwendungsfall zeigt, wie einige Browser, in diesem Fall *Edge* und *Chrome*, Standards nur teilweise unterstützen (*CSS Masks* o.D.). Für Browser mit partieller Unterstützung muss *WebKit* als HTML-Rendering-Engine gesondert angesprochen werden.

Sollten während der Entwicklung Technologien genutzt werden, für welche die angestrebten Browser keinen nativen Support bieten, muss eine Alternative über sogenannte *Polyfills* gefunden werden. Polyfills sind Code-Bausteine, welche moderne HTML-, CSS- oder JavaScript-Funktionalitäten über Umwege für Browser anbieten, welche diese nativ nicht unterstützen. Der Entwickler oder die Entwicklerin kann diese Polyfills in den geschriebenen Code einarbeiten (*Polyfills: Code-Bausteine für moderne Web-Features* 2020). Dies bedeutet jedoch Mehraufwand in der Entwicklung, sollte die Zielgruppe einen Browser nutzen, welche die gewünschten Technologien nicht unterstützt.

Eine Abstraktion der Entwicklung über Frameworks und Libraries, wie beispielsweise React, Angular oder Vue.js, vereinfacht die Entwicklung. Die Technologien setzen dabei auf JavaScript oder TypeScript, sowie wahlweise ein CSS-Präprozessor. React beispielsweise gibt an, dass alle modernen Browser unterstützt werden. Lediglich ältere Browserversionen benötigen Polyfills. Die Dokumentation von React weist jedoch explizit darauf hin, dass Browser ohne Unterstützung für *ES5* nicht von React unterstützt werden, so auch Internet Explorer (*ReactDOM* o.D.).

Ein abschließender Vorteil der Webentwicklung ist die Plattformunabhängig-

keit. Es wird keine proprietäre Soft- oder Hardware für die Entwicklung benötigt. Emulationen und virtuelle Maschinen zeigen das Verhalten der Anwendung unter bestimmten Umständen und sind in der Regel kostenfrei.

### **2.1.3 Monetarisierung**

Der Aspekt der Monetarisierung ist für Entwickelnde ein oftmals kritisch zu betrachtender Punkt. Das Handhaben von monetären Mitteln und den dazugehörigen empfindlichen Daten kann abschreckend wirken. Jobe (2013: 28) gibt an, dass für die Monetarisierung von Webanwendungen keine klare und einheitliche Strategie vorliegt. Dieser Punkt hat sich jedoch seit der Veröffentlichung von Jobe (2013)s Arbeit entwickelt. Anbieter für Zahlungsdienstleistungen haben sich auf Internet-Transaktionen und das Bereitstellen von APIs spezialisiert. Hier sei einer der Marktführer besonders hervorzuheben: *Stripe*.

Stripe ist ein Anbieter von Transaktionsverwaltung und bietet APIs für eine Integration in die Anwendung an. Sämtliche Businesslogik wird seitens Stripe serverseitig gehandhabt, sodass keine zusätzlichen datenschutzrelevanten Informationen von den Entwickelnden berücksichtigt werden müssen. Den Nutzenden wird dann über Stripe die Wahl gelassen, welche der gängigen Zahlungsmethoden genutzt werden soll (*Neue Chancen, aber unkompliziert: Unsere APIs* o.D.). Dieser Ansatz ist für den Verkauf von Waren und Dienstleistungen besonders interessant. Soll die Webanwendungen jedoch auf passive Art monetarisiert werden, bietet sich das Schalten von Werbung an.

Die Monetarisierung von Webanwendung ist nach Jobe (2013: 28) noch immer uneinheitlich, jedoch ist dies auf die Vielzahl der Anwendungsfälle zurückzuführen.

### **2.1.4 Schaffen und Konsumieren von Inhalten**

Das Schaffen und Konsumieren von Inhalten bestimmt den Alltag von Nutzenden von Webanwendungen. Jobe (2013: 28) sieht insbesondere mobile Webanwendungen als weniger geeignet für das Schaffen von Inhalten, sondern mehr für das Konsumieren. Dies liegt vermutlich in der oftmals kleinen Größe des Bildschirms von Smartphones begründet. Werde jedoch Webanwendun-

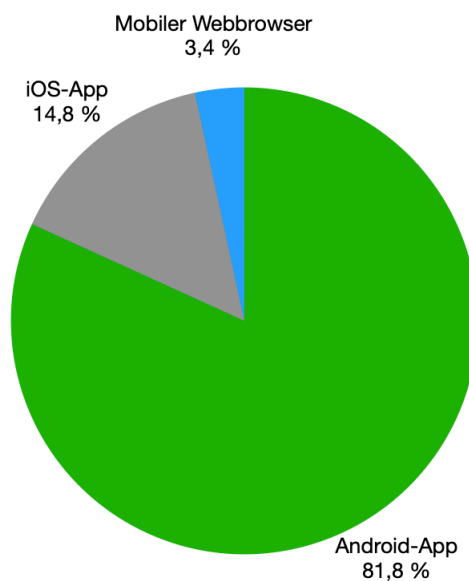
gen in einem Desktop-Browser betrachtet, können diese durchaus im Hinblick auf das Schaffen von Inhalten mit nativen Anwendungen konkurrieren. Hier sei beispielsweise *Office365*, die *Google Docs Suite* oder auch *vscode.dev* zu erwähnen. Anhand dieser Software ist die Migration von nativen Systemen hin zu Webanwendungen erkennbar. *Office365* beispielsweise verfügt über alle Funktionalitäten der nativen Version, kommt jedoch ohne Installation von zusätzlicher Software aus.

Das Konsumieren von Inhalten ist jedoch gesondert zu betrachten. Jobe (2013: 28) gibt an, dass mobile Webanwendungen im selben Maße für den Konsum von Inhalten geeignet sind, wie native Anwendungen. Bezogen auf diese Arbeit ist jedoch ein konkreter Vergleich von Vorteil. Hierzu wird sich einer Statistik über die Nutzung der Plattform facebook bedient, da facebook eine stetig steigende Zahl von Nutzenden aufweist (Statista Research Department 2022). Ziel der Statistik im Zusammenhang mit dieser Arbeit ist die Beantwortung der Frage:

Wird Inhalt auf facebook primär über die native App konsumiert oder präferieren die Nutzenden den Zugriff auf die Plattform über einen Webbrowser?

Von facebook's weltweit 2.910.000 aktiven Nutzenden im dritten Quartal 2021 Statista Research Department (2022) nutzen 81,8% der Volljährigen ausschließlich das Smartphone für den Zugang zu facebook. Lediglich 1.5% der volljährigen Nutzenden nutzen ausschließlich einen Laptop oder Desktop-Computer Kemp (2022). Da diese Aussage allein keinen Bezug zu der Nutzung von nativen Anwendungen herstellt, wird das Verhalten der mobilen Nutzenden gesondert betrachtet:

Abbildung 2: Nutzende von facebook auf dem Smartphone nach Plattform



Das Diagramm zeigt den deutlich dominierenden Anteil der Nutzenden von facebook's nativen Smartphone-Apps. Kombiniert kommen die Betriebssysteme Android und iOS auf einen Anteil von 96,6%. Lediglich 3,4% nutzen facebook auf dem Smartphone über einen Webbrowser (Kemp 2022).

Der große Anteil von Smartphone-Nutzenden im Vergleich zu Nutzenden von Desktops ist auf die Allgegenwärtigkeit von Smartphones zurückzuführen. Der Grund, weshalb Smartphone-Nutzende die native Anwendung präferieren, kann auf zwei Punkte zurückgeführt werden: Empfehlungen seitens facebook (siehe Anhang 2.1) und technische Vorteile. Der zweite Punkt lässt sich über beispielsweise Push-Mitteilungen erklären, welche aus dem Webbrowser nicht möglich sind. So wird den Konsumierenden eine direkte Kommunikation mit weiteren Nutzenden der Plattform vereinfacht. Da dieser Aspekt eng mit der User Experience der Nutzenden zusammenhängt, wird in dem gleichnamigen Absatz tiefer darauf eingegangen.

#### 2.1.5 User Experience

Wie schon im Absatz zum Schaffen und Konsumieren von Inhalten, steht hier der Nutzende im Zentrum der Betrachtung. Als User Experience wird die Erfahrung eines Nutzenden zum Zeitpunkt der Interaktion mit der Software bezeichnet. Der Entwickelnde des Software kann über gezielten Einsatz von

UI-Elementen und Plattformscheidungen eine gute User Experience begünstigen, jedoch nicht erzwingen. Die User Experience als solches ist subjektiv und stark von dem Nutzenden und seiner Situation abhängig.

Jobe (2013: 28) gibt an, dass die Erfahrung der Nutzung der Software als Webanwendung aufgrund der limitierten Integrationsmöglichkeiten beschnitten wird. Er gibt weiterhin an, dass ohne externe Frameworks keine konkurrierende Nutzungserfahrung im Vergleich zu nativen Anwendungen geschaffen werden kann. Dies sieht der Autor in dem ungehinderten Zugang der nativen Software zur Hardware des Geräts, zu Interaktions- und Interface-Optionen der Hard- und Software begründet.

Angenommen, eine zu entwickelnde Webanwendung basiert auf Echtzeitkommunikation mit dem Nutzenden. Der Nutzende soll möglichst schnell über eine wichtige Information in Kenntnis gesetzt werden, weshalb sich die von Smartphones bekannte Interaktionsmöglichkeit der Push-Mitteilung anbieten würde. Tatsächlich existiert eine solche Möglichkeit über sogenannte Web Notifications.

Web Notifications erlauben die Interaktion mit dem Nutzenden über die native Benachrichtigungs-API des Betriebssystems, sogar wenn die Webanwendung nicht geöffnet ist. Um diese Technologie nutzen zu können, muss der Nutzende zunächst die Erlaubnis dazu geben (*Notifications API* o.D.). Dies ist jedoch an die Voraussetzung geknüpft, dass der Nutzende einen unterstützten Webbrowser benutzt:

Tabelle 4: Ausgewählte Browser mit Unterstützung für Web Notifications

Browser	Version	wird unterstützt
Internet Explorer	11	nein
Edge	101	ja
Firefox	99	ja
Chrome	101	ja
Safari	15.4	ja
Safari on iOS	15.4	nein
Android Browser	101	teilweise

Die für native Anwendungen selbstverständliche Funktion, dem Nutzenden Push-Mitteilungen zukommen zu lassen, gestaltet sich für Webanwendungen eher schwierig. Begründet liegt dies in der mangelhaften Unterstützung insbesondere durch mobile Browser der Betriebssysteme iOS und Android (*Web Notifications* o.D.).

Prinzipiell ist diese Annäherung von Webanwendungen an Funktionalitäten nativer Anwendungen als positiver Aspekt für Webanwendungen zu sehen, jedoch können insbesondere Smartphones nicht einheitlich von diesem Feature profitieren. Um diese Funktion möglichst vorteilhaft zu nutzen, muss die Hard- und Software der Zielgruppe genau bestimmt werden. Sollte die Zielgruppe überwiegend Smartphones mit dem Browser *Safari on iOS* nutzen, muss aufgrund von nicht vorhandenen Polyfills gänzlich auf dieses Feature der Webanwendung verzichtet werden.

Als weiterer Aspekt einer guten Nutzungserfahrung kann die Integration eines Kontextmenüs in die Webanwendung gesehen werden. Kontextmenüs sind ein natives Feature von vielen Desktop-Betriebssystemen und spiegeln ein gelerntes Verhalten des Nutzenden wieder. Eine Integration des Menüs in die Webanwendungen lässt die Grenzen zwischen nativen Anwendungen und Webanwendungen weiterhin verschwimmen. Auf diese Weise müssen Nutzende im Produktiveinsatz der Webanwendung keine neuen Arbeitsschritte lernen und profitieren von dem bereits gelernten Verhalten. Jedoch ist auch dieser Punkt uneinheitlich in den gängigen Webbrowsern integriert:



Tabelle 5: Ausgewählte Browser mit Unterstützung für das `contextmenu`-Event

Browser	Version	wird unterstützt
Internet Explorer	11	ja
Edge	101	ja
Firefox	100	ja
Chrome	101	ja
Safari	15.4	ja
Safari on iOS	15.4	nein
Android Browser	101	ja

Die in Desktop-Browsern etablierte API für das `contextmenu`-Event wird sogar von Internet Explorer unterstützt, welcher von Entwicklenden aufgrund seiner unzureichenden Feature-Integration in der Regel lediglich durch Polyfills unterstützt wird. Einzig Safari on iOS unterstützt die genannte API nicht (*Element API: contextmenu event* o.D.). Dies hat zur Folge, dass das gelernte Verhalten der Nutzenden von Desktop-Systemen nicht auf die Nutzung von einem iPhone oder iPad projiziert werden kann.

Zur Schaffung einer möglichst positiven User Experience müssen sich Entwickelnde von Webanwendungen demnach vor der Entwicklung erkundigen, welche Hardware, beziehungsweise Software die Zielgruppe nutzt, um die Nutzungserfahrung über alle Browser gleich zu gestalten.

#### 2.1.6 Performanz

Die Performanz von Webanwendungen bezeichnet Jobe (2013: 28) als abhängig von den genutzten Redering-Engines für HTML und JavaScript. Weiterhin sieht er die Leistung von Webanwendungen im Vergleich zu nativen Anwendungen durch den limitierten Zugriff auf die Hardware beschnitten. Hier erwähnt der Autor ausdrücklich die mangelhafte Leistung von Webbrowsern auf Smartphones.

Die Performanz einer Webanwendung ist im Wesentlichen durch die Performanz des genutzten Webbrowsers und nicht zuletzt durch die verfügbare Internetverbindung beschränkt. Somit scheint der Vergleich von nativen Anwen-

dungen mit Webanwendungen in Hinblick auf die Leistungsfähigkeit nicht adäquat. Um dennoch den Evaluierungsprozess der Entwicklung anzudeuten, wird im Folgenden die Performanz der Rendering-Engines von Google Chrome und Safari verglichen.

Google Chrome nutzt für die Verarbeitung von JavaScript die sogenannte V8-Engine (*V8 JavaScript engine* o.D.). Zur Darstellung von HTML nutzt Chrome ein Derivat der WebKit-Engine, die sogenannte Blink-Engine (Barth 2013).

Safari hingegen nutzt für HTML und JavaScript die bereits erwähnte WebKit-Engine (*Companies and Organizations that have contributed to WebKit – WebKit* o.D.).

Um die Performanz der Browser zu testen und zu vergleichen, wird die Webanwendung *speedometer* von *browserbench* genutzt. Diese misst die Responsivität von einfachen Webanwendungen in dem genutzten Browser. Unter anderem nutzt die Webanwendungen *Vue.js*, *React*, *Angular*, *jQuery* und *VanillaJS*.

Als Ergebnis des Tests erreicht Chrome durchschnittlich 104,9 Durchläufe pro Minute bei einer Standardabweichung von 3,39 über insgesamt 10 Versuchsdurchläufen. Safari hingegen erreicht durchschnittlich 129,1 Durchläufe pro Minute bei einer Standardabweichung von 4,25 über ebenfalls 10 Versuchsdurchläufe (siehe Anhang 1.3). Durchgeführt wird der Test auf einem MacBook Pro 2020 (siehe Anhang 1.2).

Als Resultat des Tests ist Safari performanter als Chrome. Diese Aussage ist jedoch nicht universell gültig, da hier lediglich ein Test genutzt wird, welcher auf lediglich einem Gerät durchgeführt wird.

## **2.2 Vor- und Nachteile von nativen Anwendungen**

Als nativ werden Anwendungen bezeichnet, welche speziell für eine Plattform entwickelt werden. Die zu verwendende Programmiersprache, Frameworks und Libraries sind maßgeblich von dem gewählten System abhängig. So sollte sich zum Beginn der Entwicklung die Frage gestellt werden, welche Plattform die angestrebte Zielgruppe nutzt:

Tabelle 6: Ausgewählte Betriebssysteme und die zugehörigen Programmiersprachen für native Entwicklung

Betriebssystem	Programmiersprache	Besonderheiten
Windows	Visual Basic, C#, F#, C++/CLI	.NET Framework zur Entwicklung und Ausführung ( <i>Erste Schritte mit .NET Framework</i> o.D.)
MacOS, iOS und iPadOS	Swift	benötigt einen Mac zum Entwickeln ( <i>Swift</i> o.D.)
Android	Kotlin	Java als Programmiersprache wird noch unterstützt, jedoch wird davon abgeraten ( <i>Develop Android apps with Kotlin</i> o.D.)
Linux	z.B. Python, C, C++, Rust, JavaScript	Stark abhängig von der gewählten Distribution, Einzelfallbetrachtung nötig ( <i>Make a Linux App</i> o.D.)

Native Anwendungen sollten möglichst nahe an dem Hostsystem geschrieben werden, um von möglichst großer Leistung zu profitieren. Die systemeigenen Sprachen ermöglichen dies ohne die Notwendigkeit für eine Runtime-Umgebung.

Einheitliche Eigenschaften von nativen Anwendungen sind uneingeschränkter Zugriff auf die vom Betriebssystem genehmigte Hardware und die Unterstützung für alle möglichen Interface- und Interaktions-Optionen des Geräts, beziehungsweise des Betriebssystems (Jobe 2013: 28). Eine weitere Gemeinsamkeit ist die Voraussetzung der Installation der Anwendung, um

diese nutzen zu können.

Welche Vor- und Nachteile die Entwicklung und Nutzung von nativen Anwendungen mit sich bringt, wird in den folgenden Absätzen erläutert. Hierzu wird sich erneut der Methodik von Jobe (2013: 28) bedient, welcher die Vor- und Nachteile nach den Gesichtspunkten Update-Frequenz, Entwicklung, Monetarisierung, Schaffen und Konsumieren von Inhalten, User Experience und Performanz. Im Kontext dieser Arbeit ist jedoch die Differenzierung zwischen mobilen Anwendungen und Desktop-Anwendungen besonders zu beachten. Letztere können von jeder Person entwickelt und veröffentlicht werden. Insbesondere iOS-Anwendungen hingegen können mit originaler iOS-Software lediglich aus dem App Store herunter geladen werden. Auf diese Weise wirkt Apple Kontrolle und Regulierungen aus, um beispielsweise Schadsoftware für iOS-Nutzende unzugänglich zu machen (*App Store Review Guidelines* 2022).

### **2.2.1 Update-Frequenz**

Begonnen mit der Betrachtung von Anwendungen, welche über einen Store, wie beispielsweise Apples App Store, Googles Play Store oder der Microsoft Store, veröffentlicht werden, sind Aktualisierungen der Anwendungen formal über die integrierte Funktion des Stores (Jobe 2013: 28). Als stellvertretendes Beispiel wird hier Apples App Store angeführt. Die Aussagen gelten dabei für mobile Apps unter iOS und iPadOS, aber auch für Desktop-Anwendungen über den MacOS App Store. Diese verfügen unter anderem über einen ausführlichen Workflow für das Aktualisieren einer Anwendung. Dabei kann die Entwicklerin oder der Entwickler einen Change-Log führen, den Nutzenden über Änderungen in den Nutzungsbedingungen informieren oder Plattform-Unterstützung direkt auf der Seite der Anwendung im Store angeben (*MANAGE APPS AND VERSIONS - Create a new version* 2022).

Diese Dienstleistung seitens Apple ist für Nutzende und für Entwickelnde von Vorteil. Nutzende sehen auf einen Blick Änderungen, Neuigkeiten und unterstützte Plattformen der Anwendung. Entwickelnde müssen so keine gesonderten rechtlichen Texte, Change-Logs oder Ähnliches in die Anwendung integrieren, sondern können sich auf das Schaffen einer angenehmen User Experience fokussieren.

Anwendungen hingegen, welche nicht über Stores veröffentlicht werden, können beispielsweise über das Internet heruntergeladen werden. Diese sind in der Regel nicht reguliert. Daraus resultierend haben Entwickelnde dieser Anwendungen keinerlei Verpflichtungen gegenüber einer Plattform. Dadurch werden uneinheitliche Praktiken angewandt, wodurch die User Experience noch vor der Installation verglichen mit einem Download aus einem Store vergleichsweise gering ausfallen kann.

### **2.2.2 Entwicklung**

Die Entwicklung von nativen Anwendungen ist äußerst abhängig von der Plattform. Die Wahl von Frameworks, Libraries und Programmiersprachen kann erst nach der Entscheidung für die Plattform getroffen werden. Welche Hindernisse eine Entwicklerin oder ein Entwickler dabei bewältigen muss, wird in diesem Abschnitt näher beleuchtet.

Ein ausschlaggebender Zeitpunkt im Entwicklungsprozess ist die Wahl der Plattform. Entwickelnde können sich bei der Wahl beispielsweise folgende Fragen stellen:

- Soll die Anwendung über einen der bereits erwähnten Stores zur Verfügung gestellt werden?
- Soll die Anwendung nur auf einem bestimmten Gerät ausführbar sein?
- Unterstützt die angestrebte Hardware, beziehungsweise das Betriebssystem die Anforderungen der Anwendungen?

Benötigen Entwickelnde beispielsweise einen hardwareseitig verbauten Gyroskopsensor und die zugehörige API für die Anwendung, so sollten sie ein Smartphone als Endgerät in Betracht ziehen. Hat das Entwicklungsteam keinen Zugriff auf einen Mac, so sollte auf die Entwicklung der Anwendung für Apples Betriebssysteme verzichtet werden.

Ebenfalls zu berücksichtigen sind die zahlreichen Formfaktoren der Displays moderner Endgeräte. Die Displays von modernen Smartphones sind nicht länger rechteckig, weisen Aussparungen für Kameras und Sensoren auf und sind teilweise sogar faltbar.

Abbildung 3: Ausgewählte Endgeräte mit unregelmäßigen Displayformaten



Bei den hier gezeigten Geräten handelt es sich um das Apple iPhone 13, das Apple iPad mini und das Samsung Galaxy S20 (*Samsung Galaxy S20 Datenblatt* o.D.). Alle Geräte verfügen über abgerundete Ecken und die Smartphones zeigen zusätzliche Aussparungen im Display für Kameras und Sensorik.

Dieser Trend von unregelmäßigen Displayformaten ist spätestens mit der Veröffentlichung von Apples MacBook Pro 2021 auf ein Notebook übersprungen.

Abbildung 4: Das Display von Apples MacBook Pro 2021



Apple etabliert mit der Veröffentlichung des MacBook Pro 2021 abgerundete Ecken und Aussparungen im Display ihrer Notebooks (*MacBook Pro (14", 2021) - Technische Daten (AT) 2022*). Besonders zu beachten ist, dass nur die oberen Ecken des Displays abgerundet sind. Die unteren Ecken verbleiben rechteckig.

Inwiefern dies die User Experience verändert, wird in einem späteren Abschnitt beleuchtet. Der direkte Einfluss auf die Entwicklung ist jedoch unabstreitbar. Es kann nicht länger von einem rechteckigen zweidimensionalen Viewport ausgegangen werden. Faltbare Smartphones, gebogene Displays unregelmäßige Aussparungen gehören zu modernen Smartphones dazu. Die etablierten Entwicklungsumgebungen, am Beispiel von Xcode, erleichtern das Berücksichtigen dieser Sonderfälle. Dies hat jedoch zur Folge, dass UI-Elemente entweder verdeckt und unerreichbar sind, oder sie für jedes unterstützte Gerät neu positioniert werden müssen. Vergleichbar ist diese Problematik mit responsive Design von Websites.

In dem Fall einer Webentwicklung haben Entwickelnde die Möglichkeit, über das CSS-Feature `@media` CSS-Regeln anzuwenden, wenn eine gegebene Bedingung wahr ist. So lassen sich beispielsweise Höhe und Breite des Viewports abfragen, wodurch das Design der Webanwendung sich einem kleinen Smartphone-Viewport oder einem großen Desktop-Viewport anpassen kann (*Media Queries Level 3 2022*).

Ein weiterer, nicht zu vernachlässigender Aspekt der nativen Entwicklung ist der Mehraufwand bei einer Unterstützung von mehr als einer Plattform. Soll



die Anwendung für Nutzerinnen und Nutzer auf verschiedenen Betriebssystemen verfügbar sein, muss die Anwendung mehrmals nativ entwickelt und veröffentlicht werden. Dazu sind unterschiedliche Programmiersprachen, Frameworks und IDEs nötig, was zusätzlichen Lernaufwand bedeutet, sollten die Grundlagen hierzu erst erschlossen werden müssen.

Als finaler Punkt sei hier die ungleichmäßige Integration von Hardware-APIs angeführt. Insbesondere Smartphones profitieren von einer Vielzahl von Sensoren, auf welche eine Anwendung Zugriff haben kann. Stellvertretend sei hier Apples iPhone 13 Pro aufgeführt, welches über Apples eigene Gesichtserkennungstechnologie Face ID, einen LiDAR Scanner, ein Barometer, einen 3-Achsen Gyrosensor, einen Beschleunigungssensor und mehr verfügt (*iPhone 13 Pro und iPhone 13 Pro Max - Technische Daten* o.D.). Diese Sensoren sind jedoch in älteren Smartphones von Apple nicht verbaut, wodurch die potenzielle Zielgruppe schrumpft. Als Folge dessen müssen Entwickelnde nicht nur Hardware-Abfragen bei Smartphones unterschiedlicher Hersteller beachten, sondern ebenfalls die Klientel mit älteren Smartphones einer identischen Marke berücksichtigen.

### **2.2.3 Monetarisierung**

Die Monetarisierung von nativen Anwendungen in Stores ist über die Stores selbst möglich (*App Store Connect* o.D.). Dabei wird den Entwickelnden selbst überlassen, ob die Anwendung einmalig Geld kosten soll, eine wiederkehrende Summe verlangt wird oder Werbung geschaltet wird.

Werden hingegen native Anwendungen betrachtet, die nicht durch Stores verwaltet werden, müssen sich Entwickelnde selbst um eine Zahlungsintegration bemühen. Hier genießen Entwickelnde, vergleichbar mit der Monetarisierung von Webanwendungen, vollkommene Freiheit.

### **2.2.4 Schaffen und Konsumieren von Inhalten**

Jobe (2013: 28) sieht native Anwendungen im Aspekt des Schaffens von Inhalten im Vorteil. In dem vorigen Abschnitt zu Webanwendungen und deren Möglichkeiten zum Schaffen und Konsumieren von Inhalten wird festgestellt, dass native Anwendungen sowohl das Schaffen als auch das Konsumieren von Webanwendungen vereinfachen, beziehungsweise Nutzende die native

Option von Anwendungen Webanwendungen vorziehen. Genauer wird dieser Aspekt in dem Abschnitt Performanz beleuchtet.

### **2.2.5 User Experience**

Die Nutzungserfahrung von nativen Anwendungen wird nicht zuletzt durch den Aspekt der nativen Performanz und direktem Hardware-Zugriff begünstigt. Jobe (2013: 28) begründet die Überlegenheit von nativen Anwendungen im Punkto User Experience mit einer nahtlosen Integration der Software in das Betriebssystem. So ist es möglich, dass Entwickelnde User-Interface-Elemente des Betriebssystems nutzen können. Das gelernte Verhalten dieser Elemente fördert eine positive Erfahrung der Nutzenden, sodass diese keine neuen Design-Sprachen lernen müssen.

Ma et al. (2018: 1001) beleuchten in ihrer Arbeit die Performanz von nativen Anwendung und Webanwendungen basierend auf dem Android Betriebssystem. Hier erwähnen die Autoren, dass die Möglichkeiten nativer Anwendungen einen Vergleich dieser Ansätze ungerecht aussehen lassen. So verfügen native Anwendungen über die Möglichkeit, Einstellungen der Geräte kontextbezogen anzupassen. Als Beispiele werden Limitation des Stromkonsums bei schwacher Batterieleistung, Einschränkung des Datenverkehrs bei geringer Konnektivität und das softwareseitige Verbieten von umfangreichen Downloads ohne WLAN-Verbindung.

Diese Möglichkeiten von nativen Anwendungen können die User Experience verbessern. Native Anwendungen dominieren in diesem Aspekt Webanwendungen aufgrund des uneingeschränkten Zugriffs auf verbaute Sensoren und das Betriebssystem.

### **2.2.6 Performanz**

Native Anwendungen ziehen ihre Vorteile nicht nur aus dem Zugriff auf Informationen der Sensoren, sondern auch aus der hohen Performanz im Vergleich zu Webanwendungen. Wie bereits in der Betrachtung von Webanwendungen erwähnt, ist die Performanz von Webanwendungen durch die Performanz der durch den Browser genutzten Rendering Engines beschränkt. Native Anwendungen hingegen werden nicht von einer solchen Abstraktionsschicht eingeschränkt, wodurch ihre Performanz lediglich durch das Betriebssystem und die verbaute Hardware limitiert ist Jobe (2013: 28).

In der Arbeit von Ma et al. (2018: 999) wird dieser Aspekt im Kontext von Smartphones genauer beleuchtet. Die Autoren finden heraus, dass native Anwendungen verglichen mit Webanwendungen im allgemeinen weniger Daten übertragen. Dies sehen die Autoren in der Möglichkeit der programmatischen Limitierung der Anzahl und Größe der heruntergeladenen Ressourcen durch native Entwickelnde begründet. Weiterhin wird nativen Entwickelnden die Freiheit gelassen, selbst über Daten im Cache zu verwalten. Diese Option wird im Fall von Webanwendungen gänzlich dem Browser überlassen. Ma et al. (2018: 999) bewerten die Cache-Funktion von insbesondere mobilen Browsern als nicht zufriedenstellend. Als Gründe geben die Autoren unter anderem redundante Übertragungen von Daten an. Sie beschreiben weiterhin die mangelhafte Optimierung von Websites als einen primären Grund für die mangelhafte Cache-Performanz von Webanwendungen. Mittels kontextbezogener Anpassungen der Anfragen nach Ressourcen der Webanwendungen, lassen sich diese, so die Autoren, auf dieselbe Cache-Performanz bringen, wie sie native Anwendungen aufweisen.

Die stellvertretend betrachtete Cache-Performanz von nativen Smartphone-Anwendungen überwiegt demnach in der Regel die von mobilen Webanwendungen, wie auch Malavolta (2016: 1) bestätigt.

## 2.3 Cross-Plattform

Die Entwicklung von Cross-Plattform-Anwendungen ist kein neues Phänomen. Der Ansatz von Cross-Kompilierung erlaubt Entwicklenden, die gewünschte Anwendung in einer kompilierten Programmiersprache zu entwickeln und diese unabhängig für unterschiedliche Plattformen zu kompilieren. Dies reduziert den Aufwand, den kompilierten plattform-nativen Code individuell zu entwickeln (Dickson 2013: 7).

Prinzipiell besitzen CP-Anwendungen vergleichbare Vorteile gegenüber Webanwendungen, wie sie native Anwendungen gegenüber Webanwendungen aufweisen. Begründet liegt dies in dem bereits erwähnten kompilierten nativen Code für die jeweilige Plattform. Werden jedoch CP-Anwendungen mit nativ Entwickelten Anwendungen verglichen, werden Unterschiede deutlich. Dickson (2013: 7) erwähnt eine geringe Performanz bei einem Vergleich mit nativen Anwendungen. Dies sieht der Autor in der Menge von Libraries begründet, welche in dem Speicher abgelegt werden müssen. Dieser Vergleich hängt jedoch stark von dem verwendeten CP-Framework ab, wie Heitkötter et al. (2013: 134) bei dem Vergleich der Performanz von nativen mobilen Anwendungen mit Anwendungen des CP-Frameworks *PhoneGap* angeben.

Wie bei jeder Entwicklung müssen sich Entwickelnde zunächst über die Zielgruppe informieren. Die genutzte Hard- und Software der Zielgruppe entscheidet maßgeblich über die Wahl des Frameworks und somit auch über die zu nutzende Programmiersprache. Die Betrachtung der hier aufgeführten Frameworks wird vor dem Hintergrund eines Web-Entwickelnden mit einer Expertise in dem Web-Framework React durchgeführt. Dieser Hintergrund wird im Praxis-Teil dieser Arbeit erneut aufgegriffen, wenn die Frameworks zur Entwicklung von spezifischen Anwendungen genutzt werden.

### 2.3.1 Unterstützte Plattformen

Wie bereits erwähnt, ist die Plattform der Zielgruppe maßgeblich bestimmend für das zu nutzende CP-Framework. Ist die Zielgruppe eher auf mobilen Systemen aktiv, sollte ein Framework wie React Native in Betracht gezogen werden, welches primär auf iOS/iPadOS und Android ausgelegt ist (*React Native · Learn once, write anywhere* o.D.). Sollte sich die Zielgruppe eher an Desktop-Systemen finden, lohnt sich die Evaluierung des Frameworks Electron (*Plattformübergreifende Desktop-Anwendungen mit JavaScript, HTML*

und CSS entwickeln. o.D.). Electron nutzt die Webtechnologien HTML, CSS und JavaScript und kompiliert für MacOS, Windows oder Linux eine native Anwendung aus einer gemeinsamen Code-Basis. Kennen die Entwickelnden ihre Zielgruppe nicht, so lohnt sich eine Betrachtung des Frameworks Flutter. Flutter ist ein CP-Framework zur Erstellung von nativen Anwendungen aus einer gemeinsamen Code-Basis für eine große Menge unterschiedlicher Plattformen. Dazu zählen mobile Plattformen, das Web, Desktop-Plattformen und sogar Embedded Systems (*Flutter - Build apps for any screen* o.D.). Die hier erwähnten Frameworks werden im Folgenden genauere Betrachtung finden.

### **2.3.2 Gängige Frameworks**

React Native und Flutter belegen in der Stackoverflow Umfrage aus 2021 den sechsten und siebten Platz der beliebtesten Frameworks von etwas unter 60.000 Befragten (*2021 Developer Survey* o.D.). Um ebenfalls ein rein desktop-orientiertes CP-Framework einzubringen, wird Electron hier Erwähnung finden.

#### **2.3.2.1 React Native**

React Native ist ein von Facebook im Jahr 2015 veröffentlichtes CP-Framework zur Entwicklung von mobilen Anwendungen. React Native setzt, wie React auch, auf einen abgewandelten JavaScript-Syntax namens JSX. Zusätzlich kann CSS, beziehungsweise ein CSS-Preprozessor für Styling angewendet werden. React als Web-Framework kompiliert den geschriebenen Code und gibt die Komponenten an das Browser DOM weiter. So wird der geschriebene Code in HTML, CSS und clientseitiges JavaScript ausgegeben. React Native hingegen respektiert und berücksichtigt die nativen UI-Elemente und UX-Entscheidungen der Betriebssysteme. So werden JSX-Komponenten über eine sogenannte Bridge zu nativen Komponenten des jeweiligen Betriebssystems kompiliert (Danielsson 2016: 10f.).

Als Resultat dessen weisen Anwendungen, welche in React Native entwickelt werden, eine gute Integration in die Design-Sprache des Host-Systems auf.

#### **2.3.2.2 Electron**

Electron ist ein Framework, mit welchem Entwickelnde native Desktop-Anwendungen mithilfe von Webtechnologien entwickeln können. Zu den Ziel-Betriebssystemen zählen MacOS, Windows und Linux. Bekannte Programme, welche Electron nutzen, sind beispielsweise Visual Studio Code, Microsoft Teams oder WhatsApp (*Plattformübergreifende Desktop-Anwendungen mit JavaScript, HTML und CSS entwickeln*. o.D.). Electron basiert auf Chromium und Node.js und vereint diese Technologien in einem Framework (Jasim 2017, zitiert nach Kredpattanakul und Limpiyakorn 2019: 572).

#### **2.3.2.3 Flutter**

Flutter sticht als Framework in dieser Aufzählung etwas heraus. Primär liegt dies daran, dass Webentwickelnde für dieses Framework nicht die üblichen Technologien HTML, CSS und JavaScript nutzen können, sondern sich gegebenenfalls den Dart-Syntax aneignen müssen. Wie eingangs erwähnt, ist Flutter ein Cross-Plattform-Framework zur Entwicklung von nativen Anwendungen auf mobilen Systemen, Desktop-Systemen, Embedded Systems und sogar Webanwendungen (*Flutter - Build apps for any screen* o.D.). Damit deckt Flutter in dieser Aufzählung die meisten Plattformen ab, was den Grund für die Aufnahme des Frameworks in diese Arbeit darstellt.

### 3 Praktischer Vergleich von Cross-Plattform-Lösungen

Die erarbeitete Theorie wird an Beispielen von hierfür entwickelten Anwendungen gezeigt. Dazu werden die bereits beleuchteten Cross-Plattform-Frameworks React Native, Electron und Flutter genutzt. Hierbei ist zu erwähnen, dass sich der Hintergrund des Autors fast ausschließlich auf die Entwicklung von Webanwendungen bezieht. Sämtliches Wissen über die Frameworks wird sich speziell für diese Arbeit zugelegt. Dabei spielt Flutter durch die Dart-Programmiersprache eine besondere Rolle, da hierfür die Grundlagen des Frameworks und der Sprache erlernt werden müssen.

#### 3.1 Zielvorgabe

Es sollen Anwendungen entwickelt werden, welche auf die Plattformen macOS, Windows, Linux, iOS, Android und das Web ausgelegt sind. Dazu sollen die von den Frameworks „out-of-the-box“ unterstützen Plattformen gezielt angesprochen werden. Externe Packages, welche Unterstützung für weitere Plattformen nachrüsten, werden hier nicht betrachtet. Für React Native wird Expo genutzt, um die Entwicklung zu vereinfachen. Tabellarisch ausgedrückt unterstützen die jeweiligen CP-Frameworks die folgenden Plattformen:

Tabelle 7: Ausgewählte Cross-Plattform-Frameworks und ihre Zielplattformen

	MacOS	Windows	Linux	iOS	Android	Web
<b>React Native</b>	x	x	x	✓	✓	x
<b>Electron</b>	✓	✓	✓	x	x	x
<b>Flutter</b>	✓	✓	✓	✓	✓	✓

Unterstützung der Cross-Plattform-Frameworks für die einzelnen Plattformen. Flutter unterstützt zuzüglich Embedded Systems, welche jedoch aufgrund von mangelnder Relevanz keine Betrachtung finden.

Die Zielsysteme werden aufgrund mangelnder Geräte teilweise virtualisiert, beziehungsweise simuliert. Die folgenden Versionen der Plattformen werden genutzt:

Tabelle 8: Versionen der Zielbetriebssysteme

Betriebssystem	Version	virtualisiert/simuliert
MacOS	MacOS Monterey Version 12.3.1 (Hardware siehe Anhang 1.2)	<i>x</i>
Windows	Windows 10 Home Version 21H2 (Hardware siehe Anhang 1.4)	<i>x</i>
Linux	Ubuntu 22.04 LTS	✓
iOS	iOS 15.4	✓
Android	Android 8.1 Oreo	✓
Web	Chrome Version 101.0.4951.54 for MacOS	<i>x</i>

Für den Test mit iOS wird ein simuliertes Apple iPhone 13 Pro Max und für Android ein simuliertes Google Pixel 4 genutzt. Die Virtualisierung der Linux-Distribution wird mithilfe von VirtualBox Version 6.1 durchgeführt.

Folgende Elemente sollen in die zu entwickelnden Anwendungen integriert werden:

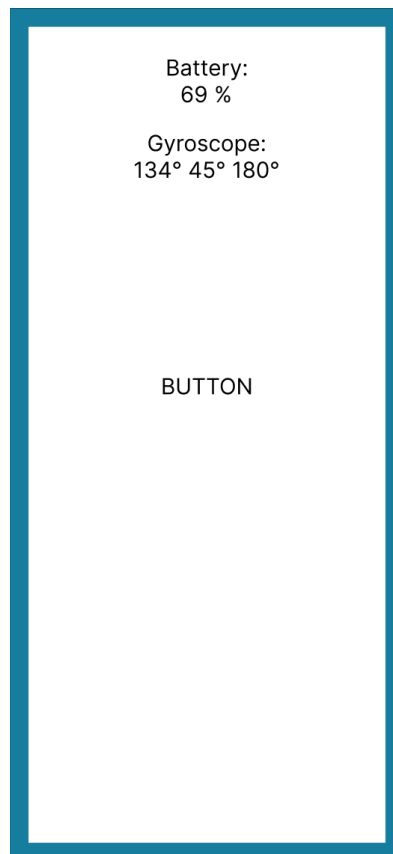
- eine Textanzeige über den aktuellen Status der Energieversorgung des Gerätes
- eine Textanzeige über die aktuelle Ausrichtung des Gerätes
- eine Umrandung des Viewports
- ein nativer Button, welcher eine Mitteilung über das native Mitteilungssystem auslöst

Über die Textanzeigen wird der Zugriff der Anwendungen auf die Sensor-APIs geprüft. Die Umrandung des Viewports dient der Visualisierung einer möglichen Integration nicht-rechteckiger Displays in das Framework, der Button



prüft die Verfügbarkeit nativer UI-Elemente und die Mitteilung testet die Integration nativer Mitteilungen in dem Framework. Das Styling der Anwendungen wird auf ein Minimum reduziert, da der Fokus auf der Funktionalität liegt. Zur genaueren Visualisierung soll folgendes MockUp dienen:

Abbildung 5: MockUp der zu entwickelnden Anwendung



Das MockUp zeigt die grundlegend umzusetzenden Anforderungen. Werte und Anordnung dienen hier als Orientierung und können sich während der Entwicklung ändern.

### **3.2 Vergleich der Umsetzungen**

## **4 Diskussion**

### **4.1 Beleg der Hypothese**

### **4.2 Limitation**

### **4.3 Ausblick**

## **5 Fazit**

## Literaturverzeichnis

- 2015 Developer Survey* (o.D.): stackoverflow, [online] <https://insights.stackoverflow.com/survey/2015#work-occupation> [abgerufen am 12.04.2022].
- 2020 Developer Survey* (o.D.): stackoverflow, [online] <https://insights.stackoverflow.com/survey/2020#developer-roles> [abgerufen am 12.04.2022].
- 2021 Developer Survey* (o.D.): stackoverflow, [online] <https://insights.stackoverflow.com/survey/2021#section-developer-roles-developer-type> [abgerufen am 12.04.2022].
- App Store Connect* (o.D.): Apple Developers, [online] [https://developer.apple.com/support/app-store-connect/#/apple\\_ref/doc/uid/TP40011225-CH26-SW1](https://developer.apple.com/support/app-store-connect/#/apple_ref/doc/uid/TP40011225-CH26-SW1) [abgerufen am 08.05.2022].
- App Store Review Guidelines* (2022): Apple Developers, [online] <https://developer.apple.com/app-store/review/guidelines/> [abgerufen am 08.05.2022].
- Async Functions* (o.D.): CanIUse, [online] <https://caniuse.com/?search=async> [abgerufen am 03.05.2022].
- Barth, A. (2013): *Blink: A rendering engine for the Chromium project*, Chromium Blog, [online] <https://blog.chromium.org/2013/04/blink-rendering-engine-for-chromium.html> [abgerufen am 06.05.2022].
- Berners-Lee, T. (1990): *Information Management: A Proposal*, CERN, [online] <https://www.w3.org/History/1989/proposal.html> [abgerufen am 08.04.2022].
- Clement, J. (2022): *Most popular websites worldwide as of November 2021, by total visits*, statista, [online] <https://www.statista.com/statistics/1201880/most-visited-websites-worldwide/> [abgerufen am 22.04.2022].
- Companies and Organizations that have contributed to WebKit – WebKit* (o.D.): WebKit, [online] <https://trac.webkit.org/wiki/Companies%20and%20Organizations%20that%20have%20contributed%20to%20WebKit> [abgerufen am 06.05.2022].
- CSS Masks* (o.D.): CanIUse, [online] <https://caniuse.com/?search=mask> [abgerufen am 03.05.2022].
- Danielsson, W. (2016): *React Native application development*, Linköping: Linköpings universitet.
- Develop Android apps with Kotlin* (o.D.): Android Developers, [online] <https://developer.android.com/kotlin> [abgerufen am 06.05.2022].
- Developer Survey Results 2016* (o.D.): stackoverflow, [online] <https://insights.stackoverflow.com/survey/2016#developer-profile-developer-occupations> [abgerufen am 12.04.2022].
- Developer Survey Results 2017* (o.D.): stackoverflow, [online] <https://insights.stackoverflow.com/survey/2017#developer-profile--developer-type> [abgerufen am 12.04.2022].
- Developer Survey Results 2018* (o.D.): stackoverflow, [online] <https://insights.stackoverflow.com/survey/2018#developer-profile--developer-type> [abgerufen am 12.04.2022].
- Developer Survey Results 2019* (o.D.): stackoverflow, [online] <https://insights.stackoverflow.com/survey/2019#developer-profile--developer-type> [abgerufen am 12.04.2022].

- Dickson, J. (2013): *Xamarin Mobile Development*, Grand Valley State University: ScholarWorks.
- Element API: contextmenu event* (o.D.): CanIUse, [online] [https://caniuse.com/mdn-api\\_element\\_contextmenu\\_event](https://caniuse.com/mdn-api_element_contextmenu_event) [abgerufen am 06.05.2022].
- Erste Schritte mit .NET Framework* (o.D.): Microsoft, [online] <https://docs.microsoft.com/de-de/dotnet/framework/get-started/> [abgerufen am 06.05.2022].
- Floridi, L. (2010): *Information - A Very Short Introduction*, New York: Oxford University Press Inc.
- Flutter - Build apps for any screen* (o.D.): Flutter, [online] [flutter.dev/](https://flutter.dev/) [abgerufen am 10.05.2022].
- Heitkötter, H., Hanschke, S. und Majchrzak, T. (2013): „Evaluating Cross-Platform Development Approaches for Mobile Applications“. In: *Web Information Systems and Technologies*. Berlin, Heidelberg: Springer. DOI: 10.1007/978-3-642-36608-6\_8.
- iPhone 13 Pro und iPhone 13 Pro Max - Technische Daten* (o.D.): Apple, [online] <https://www.apple.com/de/iphone-13-pro/specs/> [abgerufen am 08.05.2022].
- Jobe, W. (2013): Native Apps vs. Mobile Web Apps, in: Bd. 7, Nr. 4, S. 6, [abgerufen am 28.04.2022].
- Kemp, S. (28. Feb. 2022): *FACEBOOK STATISTICS AND TRENDS, DATA-REPORTAL*, [online] <https://datareportal.com/essential-facebook-stats> [abgerufen am 05.05.2022].
- Kleinrock, L. (2010): An Early History of the Internet, in: *History of Communications*, Bd. 48, Nr. 8, S. 26–36, [online] <https://ieeexplore.ieee.org/document/5534584> [abgerufen am 08.04.2022].
- Kollmann, T. (2020): *Handbuch Digitale Wirtschaft*, Wiesbaden: Springer Gabler.
- Kollmann, T. und Lomberg, C. (2010): Web 1.0, Web 2.0 and Web 3.0: The Development of E-Business, in: *Encyclopedia of E-Business Development and Management in the Global Economy*, Bd. 1, S. 1203–1210, [online] <https://www.igi-global.com/book/encyclopedia-business-development-management-global/37278> [abgerufen am 09.04.2022].
- Kredpattanakul, K. und Limpiyakorn, Y. (2019): *Transforming JavaScript-Based Web Application to Cross-Platform Desktop with Electron*, Singapore: Springer Singapore. [abgerufen am 10.05.2022].
- Ma, Y., Liu, X., Liu, Y., Liu, Y. und Huang, G. (2018): 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*, Bd. 17, Nr. 5, S. 990–1003, [online] <https://ieeexplore.ieee.org/abstract/document/8051099>.
- MacBook Pro (14", 2021) - Technische Daten (AT)* (2022): Apple, [online] [https://support.apple.com/kb/SP854?viewlocale=de\\_AT&locale=de\\_AT](https://support.apple.com/kb/SP854?viewlocale=de_AT&locale=de_AT) [abgerufen am 08.05.2022].
- Make a Linux App* (o.D.): Make a Linux App, [online] <https://makealinuxapp/#/> [abgerufen am 06.05.2022].
- Malavolta, I. (2016): „Beyond native apps: web technologies to the rescue! (keynote)“. In: *SPLASH '16: Conference on Systems, Programming, Languages, and Applications: Software for Humanity*. Amsterdam Netherlands, S. 1–2. ISBN: 978-1-4503-4643-6. DOI: 10.1145/3001854.3001863. [online] <https://dl.acm.org/doi/10.1145/3001854.3001863> [abgerufen am 09.05.2022].

- MANAGE APPS AND VERSIONS - Create a new version* (2022): App Store Connect Help, [online] <https://help.apple.com/app-store-connect/#/dev480217e79> [abgerufen am 08.05.2022].
- Media Queries Level 3* (2022): W3C, [online] <https://www.w3.org/TR/mediaqueries-3/> [abgerufen am 08.05.2022].
- Neue Chancen, aber unkompliziert: Unsere APIs* (o.D.): stripe, [online] <https://stripe.com/de> [abgerufen am 03.05.2022].
- Notifications API* (o.D.): whatwg, [online] <https://notifications.spec.whatwg.org> [abgerufen am 05.05.2022].
- O'Reilly, T. (2005): *What Is Web 2.0*, O'Reilly, [online] <https://www.oreilly.com/pub/a/web2/archive/what-is-web-20.html> [abgerufen am 09.04.2022].
- Plattformübergreifende Desktop-Anwendungen mit JavaScript, HTML und CSS entwickeln*. (o.D.): Electron, [online] <https://www.electronjs.org/> [abgerufen am 10.05.2022].
- Polyfills: Code-Bausteine für moderne Web-Features* (2020): IONOS, [online] <https://www.ionos.de/digitalguide/websites/web-entwicklung/polyfill/> [abgerufen am 03.05.2022].
- React Native · Learn once, write anywhere* (o.D.): React Native, [online] <https://reactnative.dev/> [abgerufen am 10.05.2022].
- ReactDOM* (o.D.): React, [online] <https://reactjs.org/docs/react-dom.html> [abgerufen am 03.05.2022].
- Samsung Galaxy S20 Datenblatt* (o.D.): inside digital, [online] <https://www.inside-digital.de/handys/samsung-galaxy-s20> [abgerufen am 08.05.2022].
- STANDARDS* (o.D.): W3C, [online] <https://www.w3.org/standards/> [abgerufen am 02.04.2022].
- Statista Research Department (28. Apr. 2022): *Number of monthly active Facebook users worldwide as of 1st quarter 2022*, statista, [online] <https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/> [abgerufen am 05.05.2022].
- Swift* (o.D.): Apple Developers, [online] <https://developer.apple.com/swift/> [abgerufen am 06.05.2022].
- V8 JavaScript engine* (o.D.): V8, [online] <https://v8.dev/> [abgerufen am 06.05.2022].
- Web Notifications* (o.D.): CanIUse, [online] <https://caniuse.com/notifications> [abgerufen am 05.05.2022].

## **Eidesstattliche Erklärung**

Ich erkläre hiermit und versichere an Eides Statt, dass ich meine Bachelor-Thesis / mein Praxisprojekt „Chancen und Risiken einer Migration von Webanwendungen zu nativen Systemen“ selbständig und ohne fremde Hilfe angefertigt habe und dass ich alle von anderen Autoren wörtlich übernommenen Stellen wie auch die sich an Gedankengänge anderer Autoren eng anlehnenden Ausführungen meiner Arbeit besonders gekennzeichnet und die Quellen zitiert habe.

Des Weiteren erkläre ich, dass sämtliche Ausführungen nicht bereits inhaltlicher Bestandteil einer anderen Prüfungsleistung (z.B. Praxisprojekte, Semesterarbeiten) gewesen sind.

Mir ist bekannt, dass bei einem Verstoß gegen diese Erklärung die Prüfungsleistung mit der Note „nicht ausreichend“ bewertet wird.

Kiel, den 16.05.2022

---

Fabian Reitz

## Anhang

### Anhang 1: Tabellen

#### Anhang 1.1: stackoverflow Developer Surveys 2015 bis 2021

Tabelle 9: Anzahl der Antworten der stackoverflow Developer Surveys 2015 bis 2021

Jahr	Absolute Anzahl aller Antworten	Relativer Anteil Full-Stack Developer
2013	8.218 Antworten	29 %
2014	7.346 Antworten	26,8 %
2015	22.148 Antworten	32,4 %
2016	49.525 Antworten	28 %
2017	36.125 Antworten	46,3 %
2018	92.098 Antworten	48,2 %
2019	81.335 Antworten	51,9 %
2020	49.370 Antworten	54,9 %
2021	66.484 Antworten	49,5 %

Die Tabelle zeigt die Anzahl aller Antworten pro Jahr der *stackoverflow Developer Survey*, sowie den relativen Anteil von Full-Stack Entwicklern an diesen Antworten (*2015 Developer Survey* o.D.; *Developer Survey Results 2016* o.D.; *Developer Survey Results 2017* o.D.; *Developer Survey Results 2018* o.D.; *Developer Survey Results 2019* o.D.; *2020 Developer Survey* o.D.; *2021 Developer Survey* o.D.).

## Anhang 1.2: Hard- und Softwarespezifikationen des Testsystemes

Tabelle 10: Hard- und Softwarespezifikationen des Testsystemes für den Vergleich der Leistungsfähigkeit von Safari und Chrome

Aspekt	System
Betriebssystem	macOS Monterey Version 12.3.1
Chrome Version	Version 100.0.4896.127 (Offizieller Build) (x86_64)
Safari Version	Version 15.4 (17613.1.17.1.13)
Modell	MacBook Pro (13 Zoll, 2020, Vier Thunderbolt 3 Anschlüsse)
Prozessor	2 GHz Quad-Core Intel Core i5
Speicher	32 GB 3.733 MHz LPDDR4X
Grafikkarte	Intel Iris Plus Graphics 1.536 MB

Das hier aufgeschlüsselte System wird zum Testen der Browser-Responsivität genutzt.

## Anhang 1.3: Ergebnisse des Tests zur Erfassung der Browser-Responsivität von Safari und Chrome

Tabelle 11: Ergebnisse des Tests zur Erfassung der Browser-Responsivität von Safari und Chrome mittels *speedometer*

Durchlauf	Chrome	Safari
1. Durchlauf	108 ( $\pm 1,6$ )	131 ( $\pm 3,4$ )
2. Durchlauf	107 ( $\pm 2,3$ )	125 ( $\pm 3,4$ )
3. Durchlauf	109 ( $\pm 5,8$ )	139 ( $\pm 3,5$ )
4. Durchlauf	106 ( $\pm 1,6$ )	123 ( $\pm 9,0$ )
5. Durchlauf	111 ( $\pm 5,3$ )	130 ( $\pm 1,4$ )
6. Durchlauf	103 ( $\pm 3,1$ )	128 ( $\pm 2,5$ )
7. Durchlauf	103 ( $\pm 2,9$ )	129 ( $\pm 2,4$ )
8. Durchlauf	99 ( $\pm 4,7$ )	127 ( $\pm 2,1$ )
9. Durchlauf	103 ( $\pm 1,8$ )	130 ( $\pm 3,1$ )
10. Durchlauf	100 ( $\pm 3,4$ )	129 ( $\pm 4,0$ )
Mittelwert	104,9 (gesamt $\pm 3,93$ )	129,1 (gesamt $\pm 4,25$ )

Dies sind die Testergebnisse des Browser-Responsivität, ermittelt mit *speedometer*. Die Angaben sind in der Einheit Durchläufe pro Minute angegeben. Der Test wird insgesamt zehnmal wiederholt und die Mittelwerte werden gebildet.

#### Anhang 1.4: Hardware des Windows-Systems

Tabelle 12: Hardwarespezifikationen des Windows-Systems zum Testen der entwickelten Anwendung

Aspekt	System
Prozessor	4 GHz Octa-Core AMD FX-8350
Speicher	16 GB 1.600 MHz DDR3
Grafikkarte	NVIDIA GeForce GTX 1060 6GB

Das hier aufgeschlüsselte System wird zum Testen der mithilfe eines CP-Frameworks entwickelten Anwendungen genutzt.



## Anhang 2: Abbildungen

### Anhang 2.1: facebook empfiehlt native Smartphone-App

Abbildung 6: Während der ersten Schritte der Registrierung auf facebook



Facebook empfiehlt als einer der ersten Schritte nach der Registrierung via Smartphone-Webbrowser das Herunterladen der eigenen App.