Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Studiengang Medieninformatik

Bachelorarbeit

von

Christoph Schuster

Gegenüberstellung von Ansätzen für Progressive Web Apps zum Entwurf von hybriden Benutzeroberflächen am Beispiel einer Reiseziele-Anwendung

Comparison of Progressive Web App Approaches for the Conception of Hybrid User Interfaces Using the Example of a Travel-Application

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Studiengang Medieninformatik

Bachelorarbeit

von

Christoph Schuster

Gegenüberstellung von Ansätzen für Progressive Web Apps zum Entwurf von hybriden Benutzeroberflächen am Beispiel einer Reiseziele-Anwendung

Comparison of Progressive Web App Approaches for the Conception of Hybrid User Interfaces Using the Example of a Travel-Application

Bearbeitungszeitraum: von 01. April 2022
bis 31. August 2022

1. Prüfer: Prof. Dr.-Ing. Christoph P. Neumann

2. Prüfer: Prof. Dr.-Ing. Dominikus Heckmann

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Ostbayerische Technische Hochschule
Amberg-Weiden

Bestätigung gemäß § 12 APO

Name und Vorname
der Studentin/des Studenten: **Schuster, Christoph**

Studiengang: **Medieninformatik**

Ich bestätige, dass ich die Bachelorarbeit mit dem Titel:

Gegenüberstellung von Ansätzen für Progressive Web Apps zum Entwurf von hybriden Benutzeroberflächen am Beispiel einer Reiseziele-Anwendung

selbstständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Datum: 10. September 2022

Unterschrift:

Danksagung

An dieser Stelle möchte ich mich bei allen Personen bedanken, die zum Gelingen dieser Arbeit beigetragen haben.
Ein besonderer Dank gilt hierbei Herrn Prof. Neumann, der meine Bachelorarbeit stets mit viel Engagement betreut hat. Besonders möchte ich dabei die hilfreichen Anregungen, sowie das stets schnelle und konstruktive Feedback während der Erstellung dieser Arbeit hervorheben.
Außerdem möchte ich mich bei meiner Familie, insbesondere meinen Eltern bedanken. Vielen Dank für Euren motivierenden Beistand, sowie Eure Unterstützung während meines Studiums in jeglicher Hinsicht.

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Ostbayerische Technische Hochschule
Amberg-Weiden

Bachelorarbeit Zusammenfassung

Studentin/Student (Name, Vorname): **Schuster, Christoph**
Studiengang: Medieninformatik
Aufgabensteller, Professor: Prof. Dr.-Ing. Christoph P. Neumann
Durchgeführt in (Firma/Hochschule): OTH Amberg-Weiden

Ausgabedatum: 01. April 2022 Abgabedatum: 31. August 2022

Titel:

Gegenüberstellung von Ansätzen für Progressive Web Apps zum Entwurf von hybriden Benutzeroberflächen am Beispiel einer Reiseziele-Anwendung

---

Zusammenfassung:

Die vorliegende Arbeit beschäftigt sich mit dem Konzept der Progressive Web App. Ähnlich zu Web Apps sind diese plattformunabhängig nutzbar. Gleichzeitig sollen sich Progressive Web Apps dabei aber wie eine native Anwendung verhalten und dieser auch äußerlich in ihrer Benutzeroberfläche gleichen.
Ziel dieser Arbeit ist herauszufinden, welche Aspekte und Richtlinien beachtet werden müssen, um eine größtmögliche Ähnlichkeit der Benutzeroberfläche einer Progressive Web App zu nativen Anwendungen auf dem Desktop und mobilen Plattformen zu erreichen. Inwiefern Webframeworks Funktionalitäten anbieten, um dies umzusetzen, wird ebenfalls untersucht. Des Weiteren soll die Arbeit das aktuelle Potential von Progressive Web Apps prüfen, andere Anwendungskonzepte in Zukunft zu ersetzen.
Zur Klärung dieser Fragen wurde unter anderem ein Webframework-Vergleich anhand eines Kriterienkatalogs auf Basis einer reinen Literaturrecherche durchgeführt. Darüber hinaus erfolgte eine weitere Literaturrecherche, um Prinzipien und Richtlinien zur Gestaltung von Benutzeroberflächen nativer Anwendungen auf dem Desktop und mobilen Plattformen, im Speziellen für Android und IOS, aufzuzeigen. Zudem wurden dabei Informationen gesammelt, um das Konzept der Progressive Web App mit anderen Anwendungskonzepten zu vergleichen.
Um das Ergebnis des Webframework-Vergleichs zu prüfen, wurde eine Progressive Web App mit dem Webframework entwickelt, welches das beste Ergebnis in diesem Vergleich erzielte. Zudem kamen bei der Entwicklung der Progressive Web App die recherchierten Design-Prinzipien zum Einsatz, um den Mehrwert dieser bei der Gestaltung einer Benutzeroberfläche zu prüfen.

Der Webframework-Vergleich zeigt, dass sich das Framework Ionic am besten für die Entwicklung von Progressive Web Apps eignet, die je nach Plattform ihre Benutzeroberfläche ändern und entsprechend den Richtlinien der jeweiligen Plattform anpassen.
Die Umsetzung der Progressive Web App macht des Weiteren deutlich, dass die Ergebnisse des Webframework-Vergleichs in Bezug auf Ionic zutreffen. Auch der Einsatz der recherchierten Design-Prinzipien hilft, neben den Funktionalitäten von Ionic, die Benutzeroberfläche der Progressive Web App an eine jeweilige Plattform weitestgehend anzupassen.
Die Untersuchung, ob Progressive Web Apps andere Anwendungskonzepte in Zukunft ersetzen könnten, lässt zudem den Entschluss zu, dass Progressive Web Apps zwar über die letzten Jahre vor allem durch Google mehr an Bedeutung gewonnen haben, zeitgleich sich aber Progressive Web Apps, vor allem gegenüber nativen Anwendungen durch Apples mangelnde Unterstützung, auch in naher Zukunft nicht durchsetzen können.

Schlüsselwörter: Progressive Web App, PWA, App-Like, Design, Webframework, Ionic, React, Vue, Benutzeroberfläche, Native App, Web App, Desktop, Mobile

Abstract

This bachelor thesis deals with the concept of the progressive web app. Similar to web apps, these can be used platform-independent. At the same time, however, they should behave like native applications and also resemble them in their user interface.
The goal of this thesis is to find out which aspects and guidelines have to be considered for this similarity of the user interface to native applications on the desktop and mobile platforms and to what extent web frameworks offer functionalities to implement this similarity to native applications. Furthermore, the work examines the current potential of progressive web apps to replace other application concepts in the future.
To clarify these questions, a web framework comparison was performed using a criteria catalog based on a literature research. Furthermore, an additional literature research was conducted to show principles and guidelines for the design of user interfaces of native applications on the desktop and mobile platforms. For mobile platforms in particular for Android and IOS. Also the additional literature research was used to compare the concept of the progressive web app against other application concepts.
To test the result of the web framework comparison, a progressive web app was developed using the web framework that achieved the best result of this comparison. In addition, the researched design principles were used in the development of the progressive web app.
The web framework comparison shows that the Ionic framework is the best fitting for developing progressive web apps that change their user interface and adapt it according to the guidelines of the respective platform.
Furthermore, the implementation of the progressive web app makes it clear that the results of the web framework comparison apply in regard to Ionic. Also, the use of the researched design principles, in addition to the functionalities of Ionic, helps to adapt the user interface of the progressive web app to the particular platform from which the user accesses it.
The research into whether progressive web apps could replace other application concepts in the future leads to the conclusion that progressive web apps have gained more importance over the last few years, especially through Google. But, at the same time progressive web apps will not be able to replace native applications in the near future due to the lack of support from Apple.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Abkürzungsverzeichnis

PWA	Progressive Web App
UI	User Interface
JSX	JavaScript XML
XML	Extensible Markup Language
SFC	Single File Components
CLI	Command-Line Interface
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
HTTPS	Hypertext Transfer Protocol Secure
JSON	JavaScript Object Notation
IDE	Integrated Development Environment

1. Einleitung

Endgeräte erlauben es, auf Anwendungen oder das Internet zuzugreifen und so den Alltag zu erleichtern. Diese gibt es dabei in verschiedensten Ausführungen und Variationen, welche ein Benutzer abhängig seiner Bedürfnisse auswählt und nutzt. Befindet sich ein Benutzer zu Hause, so verwenden dieser beispielsweise eher einen Computer. Ist dieser unterwegs, greift er häufiger auf ein mobiles Endgerät, wie das Smartphone, zurück.
Die gebotene Palette an Endgeräten bringt dabei aber auch jeweils eine eigene Auswahl an Betriebssystemen mit sich. So nutzen einige Computer das Windows-Betriebssystem von Microsoft, manche Smartphones verwenden das Android-Betriebssystem, andere das IOS von Apple. Diese Begebenheit stellt vor allem Entwickler vor die Entscheidung, für welches Betriebssystem beziehungsweise welche Plattform sie ihre Applikation im Speziellen entwickeln sollen. Möchte ein Entwickler aber beispielsweise eine Android-Anwendung zusätzlich auf einer zweiten Plattform wie IOS anbieten, muss dieser die Anwendung entsprechend neu implementieren und anpassen, was einen weiteren Entwicklungs- und Kostenaufwand erfordert.

Für die Lösung dieses Problems, Anwendungen plattformübergreifend anzubieten und diese dadurch so vielen Benutzern als möglich zugänglich zu machen, existieren bereits verschiedenste Konzepte. Eines von diesen Konzepten, ist das Konzept der *Progressive Web App* (PWA).
Progressive Web Apps werden grundsätzlich über das Internet in einem Browser aufgerufen und ausgeführt, wodurch sie plattformunabhängig nutzbar sind. Dennoch bieten Progressive Web Apps die Möglichkeit, sie lokal auf einem Endgerät zu installieren und offline zu verwenden. Die Installation erfolgt dabei über ein sogenanntes *Web App Manifest* – eine JSON-Datei, welche die hierzu benötigten Informationen und Ressourcen für die PWA beinhaltet. Die Offline-Fähigkeit stellt ein *Service Worker* zur Verfügung. Dabei handelt es sich um eine JavaScript-Datei, die im Hintergrund eines Browsers ausgeführt wird.
Des Weiteren können Entwickler Funktionalitäten, die eigentlich nativen Anwendungen vorbehalten sind, in Progressive Web Apps einbauen und somit auf Hardware-Komponenten eines Endgerätes, wie die Smartphone-Kamera, zugreifen. Native Anwendungen sind hierbei Applikationen, die Entwickler speziell für ein Betriebssystem entwickeln.
Progressive Web Apps sind somit in der Lage, sich weitestgehend dem Konzept nativer Anwendungen anzunähern, obwohl sie keiner spezifischen Plattform angehören.

Das Konzept der PWA definiert zudem verschiedene Kriterien, deren Anforderungen weitestgehend zu erfüllen sind, damit eine Anwendung das Potential einer Progressive Web App gänzlich ausschöpfen kann. Das Kriterium *App-Like* beschreibt beispielsweise die Anforderung an eine PWA, weitestgehend wie eine native Anwendung auszusehen und sich wie eine zu verhalten.
Die Kombination plattformunabhängiger und plattformspezifischer Aspekte innerhalb des PWA-Konzeptes erweckt mit Sicherheit das Interesse in Entwicklerkreisen. Vor allem Googles Unterstützung gegenüber diesem Konzept, welches seit 2015 maßgeblich an der Formung von Progressive Web Apps beteiligt ist, verdeutlicht, welches Potential und welche Möglichkeiten Progressive Web Apps in der App-Landschaft bieten können. [Russel, 2015]

Dieses Potential von Progressive Web Apps gegenüber anderen Anwendungskonzepten durch beispielsweise ihrer Performanz oder ihrem Zugriff auf native Funktionalitäten, wurde bereits durch einige Forschungen untersucht. [Sander and Ackermann, 2018] [Diekmann and Eggert, 2021]
Wie speziell sichergestellt werden kann, dass Progressive Web Apps sich mit nativen Anwendungen vor allem in ihrer Gestaltung gleichen, wird dabei nicht direkt beantwortet. Auch befassen sich die Forschungsarbeiten hauptsächlich mit Progressive Web Apps auf mobilen Plattformen. Wie eine Progressive Web App native Aspekte auf einem Desktop aufgreifen kann, bleibt daher offen, obwohl Benutzer eine PWA auch auf diesem über den Browser aufrufen können.

1.1 Zielsetzung

Ziel dieser Arbeit ist herauszufinden, wie sichergestellt werden kann, dass sich eine Progressive Web App vor allem mit ihrer Benutzeroberfläche weitestgehend an die native Umgebung der jeweiligen Plattform anpasst, in der die PWA ausgeführt wird. Der App-Like-Aspekt einer PWA ist dabei zu erfüllen. Insbesondere soll dabei ein Vergleich zwischen der Gestaltung, sowie dem Aufbau von Desktop-Anwendungen und Anwendungen auf mobilen Plattformen erfolgen. In Bezug auf die mobilen Plattformen sollen zudem zusätzliche Unterschiede zwischen IOS und Android aufgegriffen werden.
Zur Veranschaulichung der Ergebnisse dieses Diskurses soll eine PWA mit einer hybriden Benutzeroberfläche in ihrer Gestaltung vorab geplant und anschließend implementiert werden. Hybrid bedeutet in diesem Kontext, zwei plattformspezifische Benutzeroberflächen (Desktop & Mobil) in einer einzigen Benutzeroberfläche zusammenzuführen, die sich dann je nach verwendeter Plattform in ihrem Aussehen anpasst. Die Grundfunktionalitäten einer PWA sind dabei zudem zu erfüllen. Auch native Funktionalitäten sollen in dieser Progressive Web App nutzbar sein.
Zur Umsetzung der Progressive Web App soll ein Webframework genutzt werden. Um welches Webframework es sich dabei handelt, ist zudem im Rahmen dieser Arbeit zu klären. Hierbei sollen mehrere Webframeworks zur Auswahl gestellt und deren Schwächen und Stärken zur Entwicklung von Progressive Web Apps aufgezeigt werden.

Es soll des Weiteren im Rahmen dieser Arbeit ein aktueller Überblick gegeben werden, ob sich Progressive Web Apps zum jetzigen Zeitpunkt in Zukunft gegenüber anderen Anwendungskonzepten durchsetzen könnten.

1.2 Methodik

Um das Konzept der Progressive Web App im Vergleich zu anderen Anwendungskonzepten abgrenzen zu können, erfolgt die Definition der Konzepte *Web App*, *Native App* und *Hybride App*. Anschließend rückt das Konzept der PWA mit einer entsprechenden Definition und einem Überblick des aktuellen technologischen Stands in den Fokus.
Die Wahl des Webframeworks für die Entwicklung der Progressive Web App erfolgt mit Hilfe eines Webframework-Vergleichs. Dieser basiert auf reiner Literaturrecherche. Hierbei kommt ein zuvor definierter Kriterienkatalog zum Einsatz, welcher als Beurteilungsgrundlage dient. Das Ergebnis des Vergleichs legt anschließend fest, welches Webframework sich am besten für die Entwicklung der PWA eignet.
Zur Gestaltung einer hybriden Benutzeroberfläche werden in einer weiteren Literaturrecherche Design-Prinzipien und Richtlinien für Webseiten und Benutzeroberflächen von Desktop-Anwendungen und mobilen Anwendungen analysiert. Dadurch, dass Webseiten plattformunabhängig agieren, helfen diese verschiedene Möglichkeiten zu erörtern, um Elemente einer Benutzeroberfläche an verschiedene Bildschirmgrößen anzupassen.
Mit Hilfe eines PWA-Prototypen werden anschließend die gesammelten Design-Prinzipien und Richtlinien beispielhaft für die Gestaltung der hybriden Benutzeroberfläche praktisch angewendet. Die Implementierung der PWA erfolgt dabei mit dem festgelegten Webframework aus dem Webframework-Vergleich. Die Grundfunktionen einer PWA und zusätzliche native Funktionalitäten werden dabei auch beim Prototyp mit aufgegriffen.
Zum Schluss erfolgt eine kurze Analyse des PWA-Konzepts gegenüber den anderen, aufgezeigten Anwendungskonzepten und inwiefern Progressive Web Apps diese in Zukunft ersetzen könnten.

1.3 Aufbau der Arbeit

Die Arbeit teilt sich in sieben Kapitel auf. Nach der Hinführung zur Thematik in Kapitel 1 erfolgt in Kapitel 2.1 eine Beleuchtung des PWA-Konzepts, sowie anderer Anwendungskonzepte. Anschließend wird in Kapitel 2.2 das Konzept der Progressive Web App genauer dargestellt und der aktuelle technologische Stand geklärt.
In Kapitel 3 erfolgt der Webframework-Vergleich anhand eines Kriterienkatalogs, der am Ende das Webframework festlegt, welches zur Entwicklung der PWA zum Einsatz kommt.
Kapitel 4 behandelt die Planung der Progressive Web App für die anschließende Implementierung. Dabei wird in Kapitel 4.1 die Architektur der PWA erläutert. Das Kapitel 4.2 beleuchtet anschließend Design-Prinzipien zur Gestaltung von Benutzeroberflächen auf dem Desktop, mobilen Plattformen, sowie im speziellen Gestaltungsrichtlinien

für die mobilen Plattformen Android und IOS. Auch die Erstellung von Wireframes und Mockups, welche die Gestaltung und den Aufbau der zu entwickelnden hybriden Benutzeroberfläche definieren, wird hier vorgenommen.
Die Darstellung der Implementierung der Progressive Web App erfolgt in Kapitel 5 und gibt dabei einen Überblick, wie eine hybride Benutzeroberfläche mit dem gewählten Webframework umgesetzt werden kann. Auch Beispiele zu eingesetzten Design-Prinzipien innerhalb der umgesetzten PWA werden dabei aufgeführt, sowie der Zugriff auf native Funktionalitäten durch die Progressive Web App.
In Kapitel 6 erfolgt die Evaluation. Diese beinhaltet in Kapitel 6.1 die Bewertung der PWA auf technischer Ebene. Anschließend wird die Umsetzung der PWA in Kapitel 6.2 mit dem verwendeten Webframework in Bezugnahme auf den in Kapitel 3 durchgeführten Webframework-Vergleich evaluiert und geprüft, ob dem Ergebnis aus diesem zugestimmt werden kann.
Am Ende des Kapitels erfolgt ein kurzer Diskurs, ob zum aktuellen Zeitpunkt Progressive Web Apps andere Anwendungskonzepte ersetzen könnten.
Zum Schluss der Arbeit sind in Kapitel 7 das Fazit und ein kurzer Ausblick auf weitere mögliche Forschungsfragen, sowie zukünftige Entwicklungen der PWA, einzusehen.

2. Grundlagen

Im folgenden Kapitel erfolgt ein Überblick über verschiedene Anwendungskonzepte, sowie eine genauere Erörterung des PWA-Konzeptes selbst. Dabei rückt der aktuelle technologische Stand von Progressive Web Apps in den Mittelpunkt, sowie die Kompatibilität der modernen Browser eine Progressive Web App auszuführen.

2.1 Definition existierender Konzepte

2.1.1 Web App

Eine Web App ist eine Anwendung, die Entwickler mit Hilfe von HTML, CSS und JavaScript oder TypeScript implementieren. Sie ist wie eine Webseite über eine URL erreichbar und wird entsprechend in einem Browser ausgeführt, wodurch eine Installierung auf dem verwendeten Endgerät entfällt. Web Apps sind somit unabhängig von einem Betriebssystem nutzbar.
Sind Web Apps zudem responsiv gestaltet, also dass sich deren Inhalt in Anordnung und Skalierung an verschiedene Bildschirmgrößen anpasst, können diese sowohl auf dem Smartphone, dem Tablet oder dem Desktop genutzt werden, ohne dass bestimmte Bereiche der Anwendung durch den jeweiligen Bildschirmrand abgeschnitten werden. [Bühler et al., 2019, S.77] [Luntovskyy and Gütter, 2020, S.347]

2.1.2 Native App

Im Gegensatz zu Web Apps werden Native Apps für spezielle Plattformen und dem dabei einhergehenden Betriebssystem entwickelt. Die zu verwendende Programmiersprache, sowie die zu verwendende Entwicklungsumgebung zur Umsetzung einer nativen Anwendung, hängt dabei auch von der gewählten Plattform ab. So nutzen Programmierer zur Entwicklung einer nativen IOS-Anwendung beispielsweise die Programmiersprache Swift und die Entwicklungsumgebung *Xcode*. Für native Android-Anwendungen kommt hingegen Java und *Android Studio* zum Einsatz. Anwendungen für Microsoft Windows nutzen C#, C oder C++ und die IDEs *Visual Studio* oder *Visual Studio Code*.
Dadurch, dass native Anwendungen an spezielle Plattformen gebunden sind, können diese Hardware-Komponenten des jeweiligen Endgerätes uneingeschränkt nutzen. [Luntovskyy and Gütter, 2020, S.346]

Um auf native Anwendungen zugreifen zu können, müssen Benutzer auf den jeweiligen Store des Betriebssystems, wie den Google Play Store (Android) oder den App Store (IOS), zugreifen. Entwickler müssen dabei, um ihre Anwendung dort anbieten zu können, vorher bestimmte Richtlinien und Vorgaben der jeweiligen Stores erfüllen. Insbesondere die strengeren Richtlinien von Apple können dabei den Veröffentlichungsprozess einer Anwendung verzögern. [Bühler et al., 2019, S.76]

2.1.3 Hybride App

Hybride Apps sind eine Kombination aus Web Apps und nativen Anwendungen. Wie bei einer Web App erfolgt die Entwicklung einer hybriden Anwendung zu Beginn mit Hilfe von HTML, CSS und JavaScript oder TypeScript. Anschließend wird mit Hilfe eines Frameworks, wie *Ionic* oder *PhoneGap*, die Applikation von einem nativen Wrapper umgeben, mit diesem hybride Anwendungen auf Hardware-Komponenten der jeweiligen Plattform zugreifen können. Hybride Anwendungen werden somit plattformunabhängig entwickelt und für das jeweilige Betriebssystem anschließend kompiliert. Entwickler können zudem hybride Applikationen über den jeweiligen Store der Plattform anbieten. Voraussetzung hierfür ist, dass ihre Anwendung, ähnlich zu einer nativen Anwendung, die Vorgaben und Richtlinien der jeweiligen Plattform erfüllt. [Bühler et al., 2019, S.77] [Luntovskyy and Gütter, 2020, S.348]

2.1.4 Progressive Web App

Der Begriff *Progressive Web App* wurde 2015 von Google durch Alex Russell und Frances Berriman ins Leben gerufen. [Russel, 2015] Das Konzept der PWA selbst existiert aber bereits seit 2007 und wurde von Steve Jobs präsentiert. [divante, 2019]

Progressive Web Apps sind ein weiterführendes Konzept von Web Apps und werden grundsätzlich in einem Browser geladen und ausgeführt, wodurch man sie unabhängig eines Betriebssystems implementieren und nutzen kann. Des Weiteren sind Progressive Web Apps zugleich lokal auf einem Endgerät installierbar und können native Funktionalitäten anbieten, die an sich nativen Applikationen vorbehalten sind. Dazu zählt zum Beispiel der Zugriff auf die Kamera oder auf den Standort des jeweiligen Endgerätes. [Digital Guide Ionos, 2019]
Das Konzept der Progressive Web App beruht zudem auf *Progressive Enhancement*.

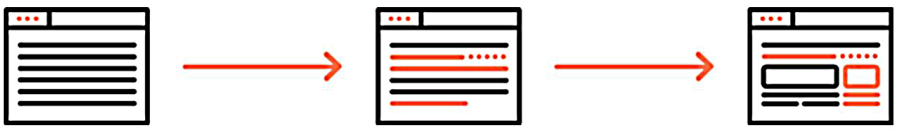

Abbildung 2.1: Progressive Enhancement [Sheppard, 2017, S.7]

Progressive Enhancement ist eine Strategie zur Entwicklung von Webseiten. Bei dieser wird versucht, dass Webseiten sowohl für moderne Browser und leistungsfähigere Geräte, als auch für veraltete Browser und weniger leistungsstarke Endgeräte nutzbar bleiben.

Dabei sollen die Kernfunktionalitäten beziehungsweise der Kerninhalt einer Webseite auf allen Endgeräten verwendbar und sichtbar sein. Benutzern mit moderneren Browsern und leistungsfähigeren Geräten wird dann eine verbesserte Version dieser Webseite zur Verfügung gestellt, wie es auch beispielhaft Abbildung 2.1 darstellt. Auf der linken Seite der Abbildung ist dabei eine Webseite zu sehen, die nur ihre Kerninhalte dem Benutzer präsentiert. Rechts neben dieser befinden sich zwei ihrer verbesserten Versionen mit zusätzlicher Gestaltung durch CSS und weiteren Funktionalitäten durch JavaScript, die je nach Leistung des Browsers und Endgerätes dem Benutzer zur Verfügung gestellt werden. [Sheppard, 2017, S.6]

2.2 Technologischer Stand von Progressive Web Apps

Damit eine Progressive Web App funktioniert, müssen drei technologische Anforderungen erfüllt werden. Diese umfassen einen *Service Worker*, ein *Web App Manifest* und die Nutzung des Kommunikationsprotokolls *HTTPS*.

Service Worker

Ein Service Worker ist der Kern einer PWA. Der Service Worker ist eine JavaScript-Datei, die im Hintergrund eines Browsers ausgeführt wird, also unabhängig des Hauptthread. Dadurch sind Service Worker nutzbar, auch wenn die Anwendung, welche den Service Worker verwendet, inaktiv ist. Push-Benachrichtigungen können so beispielsweise vom Service Worker ohne aktive PWA empfangen und vom jeweils verwendeten Browser dem Benutzer dargestellt werden.
Des Weiteren ermöglichen Service Worker eine Progressive Web App offline zu verwenden. Dabei ist beim erstmaligen Aufrufen der PWA eine Internetverbindung notwendig, damit der Service Worker installiert und initialisiert werden kann. Des Weiteren werden dabei benötigte Inhalte und Ressourcen der PWA vom entsprechenden Server lokal durch beispielsweise *IndexedDB*, einer clientseitigen Datenbank, innerhalb des Browsers abgelegt.
Nach einem Neustart der Progressive Web App ist der Service Worker für die PWA einsatzbereit und lädt Daten und Ressourcen für diese aus dem Zwischenspeicher in die Anwendung, falls keine Internetverbindung vorhanden sein sollte. Dadurch kann ein Benutzer die Anwendung offline nutzen und deren zwischengespeicherte Inhalte einsehen. [MDN Web Docs, 2022c]

Web App Manifest

Das Web App Manifest ist eine JSON-Textdatei und soll die Installierbarkeit der PWA auf dem Startbildschirm (engl. Homescreen) eines Endgerätes ermöglichen. Diese Textdatei beinhaltet die hierzu erforderlichen Informationen zur PWA. Dazu zählt unter anderem der Name der PWA, die Version der PWA und das zu verwendende Icon, das auf dem Homescreen des Endgerätes darzustellen ist. [Microsoft, 2022]

HTTPS

Da ein Service Worker unter anderem Daten aus einem Cache beziehungsweise Zwischenspeicher innerhalb des Browsers lädt und verhindert werden soll, dass sensible Daten an Dritte gelangen, ist aus Sicherheitsgründen zur Nutzung eines Service Workers das Kommunikationsprotokoll HTTPS (Abk. für HyperText Transfer Protocol Secure) erforderlich. [MDN Web Docs, 2022c]

2.2.1 Anforderungen an eine Progressive Web App

Seit dem Aufkommen des Konzeptes der Progressive Web App wurden bestimmte Anforderungen beziehungsweise Charakteristiken definiert, die eine PWA erfüllen sollte, um besagtem Konzept gerecht zu werden. Diese Anforderungen zählen auch zu den wesentlichen Vorteilen von Progressive Web Apps. [Microsoft, 2022] [MDN Web Docs, 2022a]

Discoverable

Progressive Web Apps besitzen wie eine Webseite oder eine Web App eine URL und werden somit von Suchmaschinen indiziert. Über diese Suchmaschinen ist eine PWA somit durch eine einfache Suche von Benutzern durch bestimmte Schlüsselwörter auffindbar. [MDN Web Docs, 2022a]

Installable

Durch das Web App Manifest von Progressive Web Apps können Benutzer eine PWA lokal auf ihren Endgeräten installieren. Über eine Verknüpfung auf dem Homescreen wird dabei ein App-Icon dargestellt, über welches der Benutzer die Progressive Web App im Browser öffnen kann, ohne eine URL in der Suchleiste eingeben zu müssen. [MDN Web Docs, 2022a]

Linkable

Eine Progressive Web App ist mit einer bestimmten URL verknüpft. Um die Progressive Web App nutzen zu können, ist somit nur die URL erforderlich, wodurch man sie verlinken und mit anderen Benutzern teilen kann, ohne dabei auf einen App Store zugreifen zu müssen. [MDN Web Docs, 2022a]

Network Independent

Auch bei einer unzuverlässig arbeitenden oder fehlenden Internetverbindung soll es möglich sein, eine Progressive Web App nutzen zu können, sobald diese bereits einmal zuvor vom Benutzer erfolgreich aufgerufen wurde. Dies ermöglicht unter anderem der Service Worker der PWA, der durch Caching bereits geladene Inhalte erneut wiedergeben kann. Auch eine clientseitige Datenbank, wie IndexedDB oder Web Storage helfen dabei, Daten der Progressive Web App lokal im Browser zu speichern und offline zugänglich zu machen. [MDN Web Docs, 2022a]

Der Designansatz *Offline Fist* (auch *Cache Fist* genannt) definiert diese Vorgehensweise, Inhalte offline zwischenzuspeichern und aus dem Zwischenspeicher bereitzustellen, sobald diese einmal heruntergeladen wurden. Inhalte werden nur dann vom Server heruntergeladen, wenn diese nicht im Zwischenspeicher vorhanden sind. [MDN Web Docs, 2022b]

Progressively Enhanced

Progressive Web Apps basieren auf Progressive Enhancement, damit Benutzer mit älteren Browsern und Benutzer mit moderneren Browsern die PWA gleichermaßen nutzen können. Dabei wird die Progressive Web App für ältere Browser auf ihre Kernfunktionen und wichtigsten Inhalte heruntergebrochen. Bei moderneren Browsern wird dann der volle Funktionsumfang der PWA ausgeschöpft. [MDN Web Docs, 2022a]

Re-engageable

Bei der Charakteristik *Re-engageable* einer PWA geht es darum, die Aufmerksamkeit eines Benutzers immer wieder für die Anwendung zu gewinnen. PWAs können hierfür auf Push-Benachrichtigungen zugreifen und wie eine native Anwendung an den Benutzer eine Nachricht senden, ohne dass dieser das Endgerät aktiv benutzen oder die PWA geöffnet haben muss. So kann er über neue Inhalte und Updates auf dem Laufenden gehalten werden. [MDN Web Docs, 2022a]

Responsive

Die Benutzeroberfläche und Inhalte der Progressive Web App passen sich an die verschiedenen Bildschirmgrößen der einzelnen Endgeräte an. So ist die PWA sowohl auf dem Desktop, dem Tablet und dem Smartphone nutzbar, ohne dass Inhalte auf kleineren Bildschirmen angeschnitten werden oder ganz verschwinden. [MDN Web Docs, 2022a]

Safe

Eine Progressive Web App muss das HTTPS Kommunikationsprotokoll nutzen, um eine verschlüsselte und sichere Übertragung zu gewährleisten. [MDN Web Docs, 2022a]

App-Like

Eine Progressive Web App soll dem Benutzer bei der Nutzung das Gefühl geben, dass er in diesem Moment eine native Anwendung auf seinem Endgerät ausführt. Die Offline-Fähigkeit, das App-Icon auf dem Homescreen oder die nativen Funktionalitäten, wie Push-Benachrichtigungen, spielen dabei eine größere Rolle.
[Tandel and Jamadar, 2018, S.9440] Da native Anwendungen speziell für bestimmte Betriebssysteme beziehungsweise Plattformen zu entwickeln sind, kann es zudem hilfreich sein, die Benutzeroberfläche einer PWA an die jeweils verwendete Plattform anzupassen, um eine native Anwendung bis zu einem gewissen Grad nachzuahmen.

2.2.2 Kompatibilität von Progressive Web Apps

In welchem Maß ein Benutzer eine Progressive Web App nutzen kann, ist stark davon abhängig, bis zu welchem Grad der jeweils genutzte Browser diesbezügliche Unterstützungen bereithält.
Damit eine Progressive Web App ihre Kernfunktionalitäten in einem Browser generell anbieten kann, muss dieser Service Worker unterstützen. Wie die Abbildung 2.2 zeigt, unterstützen zum jetzigen Zeitpunkt beinahe alle modernen Browser in ihrer aktuellsten Version vollständig oder zum Teil Service Worker.
Der Internet Explorer, Opera Mini und der KaiOS Browser sind dabei die einzigen Browser, welche Service Worker nicht unterstützen. Da die Entwicklung des Internet Explorers 2015 von Microsoft aufgrund des Microsoft Edge Browsers eingestellt wurde, wird für diesen auch in Zukunft keine Unterstützung für Service Worker zu erwarten sein. Im Juni 2022 wird die Unterstützung des Internet Explorers 11 außerdem gänzlich eingestellt. Es soll dennoch innerhalb von Microsoft Edge einen *Internet Explorer Mode* geben, mit welchem Legacy-Anwendungen und Webseiten weiterhin genutzt werden können, die noch auf den Internet Explorer angewiesen sind. [Microsoft, 2021] Legacy-Anwendungen sind dabei veraltete Programme, die auf moderneren Browsern oder Betriebssystemen nicht mehr richtig ausführbar sind. [TechTarget and ComuputerWeekly, 2020]

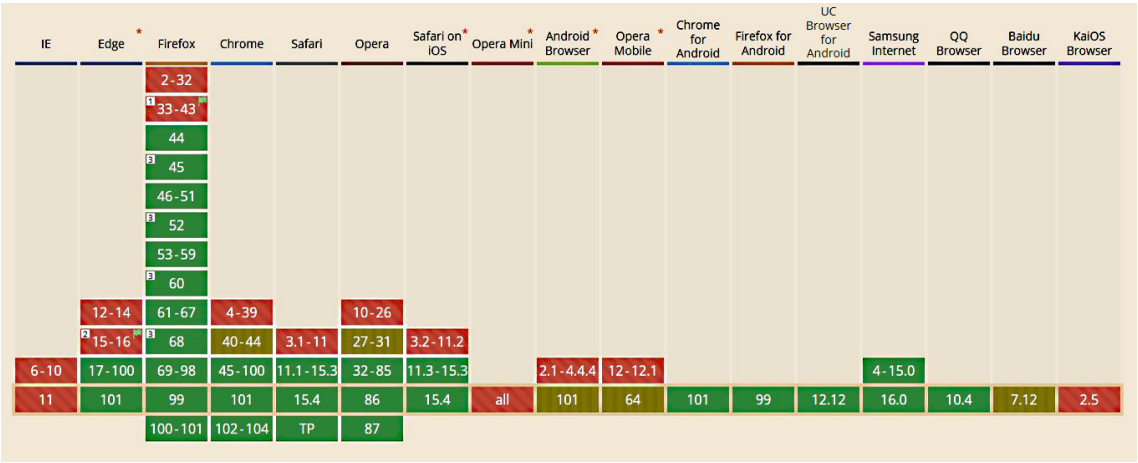

Abbildung 2.2: Unterstützung von Service Workern in den verschiedenen Browsern [Deveria, 2022]

Firefox bietet außerdem, trotz Unterstützung des Service Workers auf dem Desktop, nicht die Möglichkeit eine PWA auf dieser Plattform zu installieren. [Klose, 2021] Des Weiteren ist es browserabhängig, ob ein Benutzer native Funktionalitäten, die eine PWA eigentlich bereitstellt, auch tatsächlich nutzen kann. Während beispielsweise der Google Chrome Browser Push-Benachrichtigungen mit Hilfe von Web APIs ermöglicht, sind diese im Safari Browser auf dem IOS-Betriebssystem nicht nutzbar. Generell bietet der Safari Browser auf dem IOS-Betriebssystem vergleichsweise weniger Unterstützung gegenüber Web APIs, als beispielsweise der Chrome Browser auf dem Android-Betriebssystem. [Bar, 2022] [Sheppard, 2017, S.8]

APIs sind dabei Programmierschnittstellen, über diese Entwickler von Dritten (beispielsweise von Google) bestimmte Funktionalitäten und Dienste in ihre Anwendungen einbinden können. Web APIs sind im Speziellen quelloffen beziehungsweise kostenlos nutzbar und werden über das Internet mittels dem HTTP Protokoll in webbasierten Anwendungen verwendet. [Kulkarni, 2021]
Durch den Ansatz des Progressive Enhancements, den Progressive Web Apps verfolgen, kann eine PWA dennoch in einem Browser ausgeführt werden, auch wenn bestimmte Funktionen der Progressive Web App von diesem nicht unterstützt werden.

3. Webframework-Vergleich

Zum heutigen Zeitpunkt existiert eine große Auswahl an verschiedenen Webframeworks, welche die Entwicklung von dynamischen Webanwendungen, Webseiten und anderen webbasierten Konzepten mit unterschiedlichen Ansätzen unterstützen und um einiges vereinfachen. Bei der Wahl eines Webframeworks für ein Projekt ist es dabei wichtig zu wissen, inwiefern dessen angebotene Funktionalitäten das grundlegende Konzept des zu entwickelnden Projektes bereichern.
Da diese Arbeit hauptsächlich das Konzept der Progressive Web App beleuchtet, soll im Folgenden ein Vergleich zwischen Webframeworks erfolgen, der die Unterstützung von Progressive Web Apps durch das jeweilige Framework diskutiert. Der Vergleich erfolgt zwischen den Webframeworks *React*, *Vue*, *Ionic*, sowie *Ionic React* und *Ionic Vue* und basiert auf einer reinen Literaturrecherche.
Im Folgenden ist eine kurze Erläuterung der einzelnen Webframeworks einzusehen.

React Jordan Walke entwickelte zusammen mit dem Technologieunternehmen Facebook (2021 in Meta Platforms umbenannt) React – auch ReactJs genannt.
[Meta Platforms, 2021]
Obwohl React auf Grund seiner Funktionalitäten häufig als Webframework angesehen wird, handelt es sich offiziell um eine komponentenbasierte, quelloffene JavaScript-Bibliothek zum Erstellen von webbasierten Benutzeroberflächen. [Meta Platforms, 2022] Im Jahr 2011 verwendete Facebook zum ersten Mal React für den Facebook Newsfeed. [Bastakoti, 2022, S.1] Im Jahr 2013 veröffentlichte Facebook React als quelloffene Software. [Lay, 2019, S.6] Zum jetzigen Zeitpunkt ist die am aktuellsten verfügbare Version von React v18.0.0.

Vue Das Framework Vue – oder auch Vue.js genannt – wurde von Evan You entwickelt und 2014 auf GitHub als quelloffene Software veröffentlicht. [arocom, 2019] Es ist ein komponentenbasiertes progressives Webframework zur Erstellung von webbasierten Benutzeroberflächen und Single Page Webanwendungen. Der Begriff *progressiv* bedeutet in diesem Fall, dass Vue einem Entwickler nicht vorschreibt, in welchem Umfang dieser das Webframework nutzen soll.
So kann ein Entwickler es als einfache JavaScript-Bibliothek in einem bestehenden Projekt einbinden oder es aber als komplettes Framework für ein neues Projekt verwenden. [Vuejs, 2022a]
Vue3 ist seit 2020 die aktuelle Version des Webframeworks.

Ionic Von dem Unternehmen Drifty entwickelt, wurde Ionic 2013 für hybride und cross-plattform Applikationen veröffentlicht. Drifty wurde dabei 2012 von Max Lynch und Ben Sperry gegründet. [Ionic, 2022b]
Ionic ist ein komponentenbasiertes, quelloffenes UI Framework, das auch den Zugang zu nativen Funktionalitäten eines Endgerätes ermöglicht. Es basiert auf AngularJS und Apache Cordova und verfolgt das Prinzip „One codebase, running everywhere“[Ionic, 2022b].
Neben seiner grundlegenden Kombination mit Angular besteht auch die Möglichkeit, Ionic als standalone Framework zu nutzen. Auch ermöglicht Ionic eine Integration mit React (Ionic React) oder Vue (Ionic Vue). Im Rahmen des Webframework-Vergleichs wird die standalone Version des Frameworks nicht berücksichtigt, da Ionic standardmäßig mit Angular genutzt wird. Ionics aktuell veröffentlichte Version ist Ionic 6. [Ionic, 2022b]

Ionic React Ionic React ist seit 2019 die Integration von Ionic in das React Ökosystem. So sind React spezifische Funktionalitäten zusammen mit Ionic nutzbar. Es ist kompatibel mit React-Versionen ab v16.8. [Ionic, 2022b]

Ionic Vue Ionic Vue wurde 2020 veröffentlicht und basiert auf Vue3. [Ionic, 2022b] Es wurde bereits 2019 an Ionic Vue gearbeitet. Die Entwicklung pausierte allerdings, da Entwickler von Vue zu diesem Zeitpunkt bereits an Vue3 arbeiteten. Das Ionic-Team wollte entsprechend die Ionic Vue Integration für die neueste Version bereitstellen, statt für die damals noch aktuelle Version Vue2. [DeBeasi, 2020]

Für die Gegenüberstellung der Webframeworks erfolgt die Erstellung eines Kriterienkatalogs. Jedes Webframework erhält dabei zu jedem aufgeführten Kriterium eine Bewertung. Die Symbole in der Tabelle 3.1 entsprechen je einer möglichen Bewertung für ein Kriterium.

Tabelle 3.1: Verwendete Bewertungssymbole im Kriterienkatalog

Zeichen	Bezeichnung
+ +	trifft zu
+	trifft ziemlich zu
o	trifft teilweise zu
-	trifft kaum zu
- -	trifft nicht zu
k.W.	keine Wertung

Das Webframework, welches am Ende des Vergleichs am besten abschneidet, wird zur beispielhaften Entwicklung einer Progressive Web App im Laufe dieser Arbeit eingesetzt.

3.1 Kriterienkatalog

Der Kriterienkatalog besteht aus insgesamt sieben Kriterien. Drei dieser Kriterien fokussieren sich auf das Konzept der Progressive Web App und sollen Auskunft darüber geben, inwiefern das jeweilige Webframework dieses unterstützt. Zu diesen drei Kriterien gehört das Kriterium *UI Gestaltungsmöglichkeiten*, das Bewertungskriterium *Einfache Nutzung nativer Funktionen* und das Kriterium *PWA Unterstützung*.
Die restlichen vier Kriterien *Ausführliche Dokumentation*, *Große Community*, *Einarbeitungsaufwand* und *Testen* sollen eine allgemeinere Bewertung der Webframeworks zulassen.
Im Folgenden erfolgt eine nähere Erläuterung der einzelnen Bewertungskriterien.

Ausführliche Dokumentation

Eine Dokumentation dient vor allem in der Softwareentwicklung dazu, Informationen festzuhalten, welche Dritten unter anderem die Funktionen, den Ablauf und die Nutzung eines Systems vermitteln und näherbringen soll. Diese soll die Einarbeitung in das Webframework erleichtern.

Das Kriterium *Ausführliche Dokumentation* soll entsprechend die Ausführlichkeit und die Übersichtlichkeit der jeweiligen Dokumentation eines Webframeworks bewerten. Auch, ob das Webframework seine Dokumentation mit aktuellen Informationen pflegt und ob es Beispiele, wie Codeschnipsel, zur näheren Erläuterung von Funktionalitäten gibt, soll hierbei in das Bewertungskriterium mit einfließen.
Da im Rahmen der Arbeit das Konzept der Progressive Web App im Fokus steht, ist zudem zu prüfen, ob ein Leitfaden zur Implementierung dieser existiert.

Große Community

Die Community eines Webframeworks ist vor allem wichtig, um Fragen zu klären, die ein Entwickler beispielsweise nicht mit Hilfe der Dokumentation klären konnte. Auch Tutorials, die eine Community zu verschiedenen Themen stellt, bilden hierbei eine Stütze, das Framework besser zu verstehen.
Dadurch, dass einige Frameworks als quelloffene Software angeboten werden, tragen deren Communities außerdem einen wesentlichen Teil zu dem Wachstum des jeweiligen Webframeworks bei, beispielsweise in Form neuer Funktionalitäten, die als Bibliotheken nutzbar sind.

Das Kriterium *Große Community* soll die Größe der Community des jeweiligen Webframeworks wiederspiegeln. Dabei sollen *npm-Downloads* (Abk. für Node Package Manager) und die Anzahl an gestellten Fragen auf *Stack Overflow* zu den betreffenden Webframeworks als Maß zum Vergleich der Größe der jeweiligen Communities dienen.
Npm ist dabei das größte Software Register, über welches Entwickler Pakete herunterladen und installieren können. Über npm sind auch die Frameworks dieses Vergleichs zu installieren.
Stack Overflow ist eine Internetplattform mit der größten Community zur Erstellung und Beantwortung von Fragen.

Einarbeitungsaufwand

Bevor ein Entwickler mit der Umsetzung eines Projektes mit Hilfe eines Webframeworks startet, sollte dieser sich die Frage stellen, wie viel Zeit er in die Einarbeitung des Webframeworks investieren muss beziehungsweise möchte.

Wie gut und schnell sich ein Programmierer in die grundlegenden Funktionen des jeweiligen Webframeworks einarbeiten könnte, soll das Kriterium *Einarbeitungsaufwand* klären.

UI Gestaltungsmöglichkeiten

Neben der fehlerfreien Ausführbarkeit einer Anwendung ist es wichtig, dass diese dem Benutzer eine ansprechende und aussagekräftige Benutzeroberfläche bietet. Wie diese gestaltet ist, beeinflusst maßgeblich die Benutzererfahrung, also wie ein Benutzer die Bedienung der Anwendung im Gesamten wahrnimmt. [Digital Guide Ionos, 2018] Insbesondere Progressive Web Apps sollen dem Benutzer bei deren Verwendung das Gefühl einer nativen Anwendung vermitteln (Kriterium App-Like).

Die Auswahl an Gestaltungsmöglichkeiten für die Benutzeroberfläche, die ein Webframework einem Entwickler bietet, stellt also ein wichtiges Kriterium dar. Fragen, ob das jeweilige Framework Animationen unterstützt oder ob vorgefertigte Komponenten für Benutzeroberflächen zur Verfügung stehen, soll das Bewertungskriterium *UI Gestaltungsmöglichkeiten* beatworten.
Eine Komponente stellt dabei einen abgekapselten Block aus HTML-Elementen, CSS Styling und JavaScript dar, der innerhalb einer Anwendung beliebig oft wiederverwendbar ist. [Digital Guide Ionos, 2020]
Neben diesen Punkten ist auch zu prüfen, ob das Aussehen nativer Anwendungen weitestgehend, vor allem für Progressive Web Apps, durch das Webframework nachgeahmt werden kann.

Einfache Nutzung nativer Funktionen

Da Progressive Web Apps das Gefühl nativer Applikationen vermitteln sollen, sollten diese auch Benutzern native Funktionalitäten zur Verfügung stellen, die native Anwendungen definieren. Zu diesen zählen beispielsweise Push-Benachrichtigungen oder der Zugriff auf die Kamera oder das Mikrofon des jeweiligen Endgerätes.

Inwiefern das jeweilige Webframework den Zugang zu nativen Funktionen ermöglicht und vereinfacht, soll dieses Bewertungskriterium klären.

Testen

Um sicherzustellen, dass eine entwickelte Anwendung und deren Funktionen den Anforderungen entsprechend funktioniert, sollten Entwickler Tests durchführen.

Das Bewertungskriterium *Testen* prüft, ob das Webframework Testarten wie Unittests, Integrationstests oder End-to-End-Tests unterstützt. Hierbei soll auch beachtet werden, ob entsprechende Leitfäden und Tipps zur Durchführung von Tests in der Dokumentation des betreffenden Webframeworks existieren.
Hierzu noch eine kurze Erläuterung, über die bereits erwähnten Teststrategien.
Unittests dienen dazu, einzelne Komponenten einer Anwendung unabhängig voneinander zu testen und sollten vor den Integrationstests durchgeführt werden. [Yanes, 2021] Unittests sollten den Großteil der Testabdeckung einer Anwendung ausmachen. [Vasilev, 2021, S.9]
Integrationstests prüfen, wie einzelne Komponenten eines Systems zusammenarbeiten und ob diese ihre Aufgaben so ausführen, wie es gewollt ist.
End-to-End-Tests überprüfen die Gesamtheit der Anwendung auf Fehler anhand realer Anwendungsszenarien. Im Vergleich zu Unittests und Integrationstests sind End-to-End-Tests aber um einiges aufwändiger umzusetzen. [Yanes, 2021]

PWA Unterstützung

Dieses Bewertungskriterium prüft, inwiefern das jeweilige Webframework das Konzept der Progressiven Web App unterstützt. Hierbei fließt in die Bewertung mit ein, ob eine Progressive Web App in ihren Kernfunktionalitäten mit dem betroffenen Webframework implementiert werden kann und ob Leitfäden in der Dokumentation zur Erstellung einer PWA existieren.
Auch die Bewertungskriterien *Einfache Nutzung nativer Funktionen* und *UI Gestaltungsmöglichkeiten* nehmen auf diesen Bewertungspunkt Einfluss, da diese wichtige Anforderungen an das Konzept von Progressive Web Apps aufgreifen.

Unberücksichtigtes Bewertungskriterium Performance

Im Rahmen dieser Arbeit wird die Performance der einzelnen Webframeworks nicht als Bewertungskriterium in den Vergleich mit aufgenommen. Grund hierfür ist der Mangel an vorhandener Literatur und Last-Tests zur Performance der einzelnen Frameworks. Während zu React und Vue einige Last-Tests existieren, die deren Performance vergleichen, konnten insbesondere in Bezug auf Ionic React und Ionic Vue keine genauen Ergebnisse auf Grund der reinen Literaturrecherche gefunden werden. Aus diesen Gründen ist es nicht möglich, eine klare Aussage darüber zu treffen, welches Webframework vergleichsweise eine bessere oder schlechtere Performance liefert.

3.2 Durchführung des Vergleichs

Der Vergleich erfolgt, wie bereits erwähnt, zwischen den Webframeworks React, Vue, Ionic und deren Integrationen Ionic React und Ionic Vue.

Auswertung Ausführliche Dokumentation

Das Kriterium *Ausführliche Dokumentation* soll Aussage darüber geben, wie gut und ausführlich die Dokumentation eines Webframeworks gehalten ist und ob sie Entwicklern als grundlegende Stütze für Fragen und aufkommende Probleme dienen kann.

React React erklärt in seiner Dokumentation die wichtigsten Grundkonzepte seiner Funktionalitäten. Auch Themen für bereits fortgeschrittenere React-Entwickler sind dabei einzusehen. [Saks, 2019, S.11] Codeschnipsel dienen zum besseren Verständnis und als Orientierung für die Implementierung.
Als Einstiegs-Tutorial für angehende React-Entwickler besteht die Möglichkeit, ein kleineres Spiel durchzuarbeiten und zu implementieren, um sich mit den grundlegenden Funktionen des Frameworks vertraut zu machen. [Meta Platforms, 2022]
Im Vergleich zu anderen Frameworks ist die Dokumentation von React recht kurz und knapp gehalten. Zu einigen Themen, wie den Progressive Web Apps, ist kein Leitfaden zu finden. Des Weiteren führt die rasche Weiterentwicklung des Frameworks und die Veröffentlichung von neuen Funktionalitäten und Bibliotheken zum Teil zu veralteten Stellen in der Dokumentation. [Lay, 2019, S.23] [Bastakoti, 2022, S.8]

Vue Vue besitzt im Vergleich zu React eine bessere Dokumentation. [Saks, 2019, S.38] Laut der *State of Vue.js 2021* Umfrage für das Jahr 2021, in welcher 1635 Vue-Softwareentwickler teilnahmen, gaben 76 % der Befragten an, dass Vue eine großartige Dokumentation zur Verfügung stellt. [Monterail and Vue.js, 2021, S.34]
Die Dokumentation ist strukturiert und prägnant. Ähnlich zu React führt Vue Codeschnipsel zur näheren Erläuterung als Beispiele auf.
Entwickler können außerdem ein Tutorial, welches interaktiv durch die Grundkonzepte des Frameworks leitet, im Browser aufrufen und durchführen.
Zu Progressive Web Apps wird allerdings nichts Näheres erläutert.

Ionic Die Dokumentation von Ionic ist allgemein übersichtlich und ausführlich gehalten. Es wird alles auf einer Hauptwebseite gebündelt. So können Entwickler auch zwischen den einzelnen Ionic Versionen wechseln, um sich die hierzu jeweils passende Dokumentation anzeigen zu lassen. [altexsoft, 2019] Wie Vue und React führt auch Ionic Codeschnipsel zum besseren Verständnis in der Dokumentation auf.
Durch die mögliche Integration des Frameworks mit Angular (Standard), React und Vue existiert des Weiteren zu jedem einzelnen Framework eine extra Sparte in der Dokumentation. Für jede Framework-Kombination stellt Ionic zudem ein Tutorial mit Video zur Erstellung einer Photo Gallery App zur Verfügung.

Ionic bietet außerdem zu jeder Integration einen angepassten Leitfaden zur Implementierung einer Progressive Web App. Dieser klärt die wichtigsten technischen Anforderungen, um eine Progressive Web App zu entwickeln. [Ionic, 2022b]

Ionic Vue Wie bereits erwähnt, existieren für die einzelnen Integrationen von Ionic jeweils eigene Sparten in der Dokumentation. Die vorhandene Dokumentation zu Ionic Vue erklärt die wichtigsten Konzepte, um mit der Umsetzung eines Projektes starten zu können. Auch hier können Entwickler das Tutorial zur Erstellung einer Photo Gallery App – abgestimmt auf Vue – einsehen und umsetzen.
Im Vergleich zur Dokumentation von Ionic React führt Ionic Vue die zusätzlichen Reiter *Utility Functions* und *Troubleshooting*. Utility Functions liefert nützliche Hinweise zu Funktionen, die bestimmte Aufgaben vereinfachen sollen. So wird eine Alternative zu dem von Ionic Vue verwendeten Vue Router geboten. Auch Hinweise, um den *Hardware Back Button* eines Android-Gerätes zu nutzen oder auf die Tastatur eines Endgerätes zuzugreifen, sind hier unter anderem aufgeführt.
Der Reiter Troubleshooting gibt Entwicklern eine Übersicht über häufig auftretende Probleme, die während der Entwicklung eines Ionic Vue Projektes auftreten könnten, mit hierzu passenden Lösungen.
Im Vergleich zu Ionic und Ionic React weist die Dokumentation von Ionic Vue an einigen Stellen Lücken auf. Unter der Rubrik *Testing* beispielsweise sind keinerlei Informationen aufgeführt. [Ionic, 2022a]
An dieser Stelle ist noch zu erwähnen, dass die Dokumentation von Vue3 selbst zu Rate gezogen werden kann, um weitere Fragen zu Kernfunktionalitäten von Vue zu klären.

Ionic React Ionic Reacts Dokumentation ähnelt in ihrem Aufbau der Dokumentation zu Ionic Vue. Die wichtigsten Grundkonzepte werden erklärt. Als Einstiegs-Tutorial dient auch hier die Photo Gallery App, wobei dieses auf React abgestimmt ist.
Im Vergleich zu Ionic Vue existiert für Ionic React ein Leitfaden zum Testen eines Ionic React Projekts.
Des Weiteren werden im zusätzlichen Reiter *Overlays* zwei Techniken näher erläutert, um Overlay-Komponenten wie *Modals* – Dialogfelder, die vorübergehend die Interaktion mit der Anwendung durch den Benutzer unterbinden, bis die Applikation diese wieder verwirft – oder Warnhinweise in einem Projekt zu nutzen.
Für Ionic React können Entwickler auch zusätzlich die React Dokumentation zu Rate ziehen. [Ionic, 2022c]

Fazit Ausführliche Dokumentation Alle Webframeworks bieten Codeschnipsel und Tutorials zum besseren Verständnis. Während die Dokumentation von Ionic, Ionic React und Ionic Vue zusätzliche Informationen zur Umsetzung einer Progressive Web App aufführen, konzentrieren sich Vue und React hauptsächlich auf ihre gebotenen Funktionalitäten. Da der Fokus dieser Arbeit auf dem Konzept der PWA liegt, erhalten Vue und React auf Grund eines fehlenden PWA-Leitfadens eine schlechtere Wertungsstufe.
React steht im Vergleich mit seiner Dokumentation auf Grund der teilweise veralteten

Ausführungen zu verschiedenen Tools und Bibliotheken zudem hinter Vue. Dennoch können sich Entwickler einen guten Überblick über die grundlegenden Funktionen des Frameworks verschaffen.
Ionic Vue weist im Vergleich zu der Dokumentation von Ionic React Lücken auf, weshalb Ionic Vue im Vergleich zu diesem eine schlechtere Wertungsstufe erhält.
Obwohl beide Dokumentationen von Ionic React und Ionic Vue weitestgehend ihre Grundfunktionalitäten abdecken, empfiehlt es sich dennoch, die Dokumentationen von React beziehungsweise Vue zusätzlich einzusehen, da beispielsweise die HTML-ähnliche Syntax *JSX* von React in der Ionic React Dokumentation oder die *SFC* (Single File Components) Syntax von Vue in der Ionic Vue Dokumentation nicht im Detail erläutert wird. Auf Grund dessen, dass eine zusätzliche Dokumentation jeweils zu Rate gezogen werden sollte, muss man beiden Integrationen in diesem Vergleich einen Punkt abziehen. Folgende Tabelle 3.2 zeigt das Endergebnis dieser Kategorie.

Tabelle 3.2: Kriterium Ausführliche Dokumentation

React	Vue	Ionic	Ionic React	Ionic Vue
o	+	+ +	+	o

Auswertung Große Community

Zur Bestimmung der Größe der Community dienen in dieser Arbeit als Maß die npm-Downloads. Das folgende Diagramm der Abbildung 3.1 zeigt die Anzahl heruntergeladener Pakete für das jeweilige Webframework über einen Zeitraum der letzten zwei Jahre, im Abstand von je einer Woche. Die Farbe Rot repräsentiert dabei React, die Farbe Gelb Vue, blau Ionic mit seiner Standardintegration Angular, orange steht für Ionic React und grün für Ionic Vue.

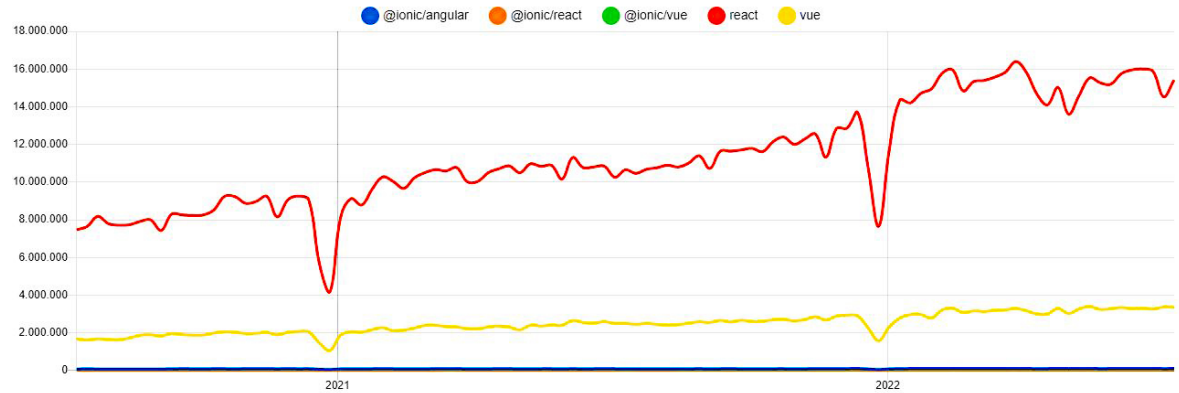

Abbildung 3.1: Wöchentliche npm-Downloads über die letzten 2 Jahre [npmtrends, 2022]

Des Weiteren dienen die gezählten Tags auf Stack Overflow als weitere Messgröße zur Bestimmung der Größe der jeweiligen Community. Die Tags dienen auf Stack Overflow zur Kategorisierung von Fragen. Ein Tag entspricht einer gestellten Frage auf der Plattform. Die Tags wurden zum 01.04.2022 ausgelesen und in die Tabelle 3.3 eingetragen.

Tabelle 3.3: Tag-Anzahl auf Stack Overflow zum jeweiligen Webframework am 01.04.2022 [Stack Exchange, 2022]

reactjs	vue.js	ionic-framework	ionic-react	ionic-vue
375.194	91.781	44.348	205	57

An dieser Stelle ist noch zu erwähnen, dass es im Speziellen auf Stack Overflow keinen Tag für Ionic Angular gibt. Vermutlich, da, wie bereits erwähnt, Ionic auf Angular basiert und Angular die Standardintegration für Ionic ist.

React React besitzt eine sehr große Community. [Lay, 2019, S.6] Die wöchentlichen React npm-Downloads in der Abbildung 3.1, welche sich über die vergangenen zwei Jahre zwischen acht Millionen und 16 Millionen Downloads bewegten, untermauern dies. Auch auf Stack Overflow wurden bei den verglichenen Webframeworks bisher die meisten Fragen zu React mit einer Anzahl von 375.194 gestellt.
Die hohe Anzahl an Entwicklern, die React nutzen, helfen das quelloffene Framework weiterzuentwickeln, wodurch eine große Auswahl an Bibliotheken von Dritten zur Verfügung steht, die bereits vorgefertigte Lösungen für aufkommende Probleme bieten. [Vasilev, 2021, S.17] Durch Reacts große Community existiert auch eine breitere Auswahl an Tutorials, die als Leitfäden zur Lösung von Problemen dienen.
Im Vergleich zu anderen Framework-Communities tritt die Community von React aber fragmentierter auf. Es gibt kein offizielles Forum, in dem gebündelt alle Themen behandelt werden. Stattdessen müssen sich Entwickler entscheiden, in welchem Forum sie ihre Frage stellen wollen. Beispiele für React Foren wären *DEV's React community* oder *Hashnode's React community*. [Monterail and Vue.js, 2021, S.79]

Vue Laut der Abbildung 3.1 besitzt Vue mit größtenteils mehr als zwei Millionen wöchentlichen npm-Downloads die zweitgrößte Community von den verglichenen Frameworks, die auch stetig am Wachsen ist. Auch mit 91.781 Tags auf Stack Overflow liegt Vue auf Platz zwei.
Vues Community ist somit kleiner als Reacts. Dadurch, dass Vue von einem chinesischen Unternehmen entwickelt wird, ist ein beträchtlicher Teil der Community zudem aus dem nicht englischsprachigen Raum. Einige Einträge im Forum oder Tutorials könnten so auf Grund der Sprachbarriere für den ein oder anderen Entwickler nutzlos sein. [Vuejs, 2022c] [Patel, 2019] Im Vergleich zu React ist die Community aber nicht so stark fragmentiert. Die offizielle Webseite von Vue, die auch die Dokumentation führt, verlinkt auf ein Hauptforum. [Vuejs, 2022a] Des Weiteren existiert ein offizieller Community Guide, der vor allem Neueinsteigern parallel zur offiziellen Dokumentation des Frameworks als Hilfestellung dienen soll. [Vuejs, 2022b]

Ionic Laut der offiziellen Webseite von Ionic nutzen im Gesamten (inklusive React und Vue Integration) mehr als fünf Millionen Entwickler das Framework. [Ionic, 2022b] Im Vergleich zu React und Vue ist die Community von Ionic aber um einiges kleiner. Die wöchentliche npm-Download-Rate lag laut des Diagramms in der Abbildung 3.1 über die vergangenen zwei Jahre, speziell bei Ionic mit seiner Standardintegration Angular, zwischen 73.000 und 120.000. Auch die Stack Overflow Tags sind mit 44.348

Fragen um mehr als die Hälfte geringer als von Vue.
Dennoch soll Ionic eine aktive Community haben, die hilft, das Framework samt seinen möglichen Integrationen zu verbessern und voran zu bringen. Es existiert ein gebündeltes Forum, auf dem Entwickler ihre Fragen stellen können, sowie Tutorials, die bei bestimmten Problemen Abhilfe schaffen sollen. [Ionic, 2022b]

Ionic Vue Die Community von Ionic Vue ist im Vergleich sehr klein. Die Download-Rate der über npm heruntergeladenen Pakete zur Nutzung von Ionic Vue liegt seit den letzten zwei Jahren pro Woche stets unter 10.000. Auch die Anzahl gestellter Fragen zu Ionic Vue auf Stack Overflow ist mit 57 Tags sehr gering. Hier sollte man aber beachten, dass teilweise Fragen zu Ionic Vue auch nur unter den Tags *ionic-framework* und/oder *vue.js* gestellt werden, weshalb die Anzahl an gestellten Fragen zu dieser Integration höher sein muss.
Da Ionic Vue Funktionalitäten von Vue nutzt, können aber Entwickler auch Einträge und Tutorials aus Foren der Vue Community zu Rate ziehen.

Ionic React Mit einer npm-Download-Anzahl von 20.000 bis 30.000 pro Woche über die vergangenen zwei Jahre muss die Community hinter Ionic React größer sein, als die Community hinter Ionic Vue. Auch der Tag *ionic-react* zeigt mit 205 gestellten Fragen auf Stack Overflow eine höhere Zahl, als es bei Ionic Vue der Fall ist. Dabei sollte ähnlich zu Ionic Vue beachtet werden, dass Entwickler Fragen zu Ionic React auch unter den Tags *ionic-framework* und/oder *reactjs* stellen.
Da Ionic React Funktionen des React Frameworks nutzt, können auch hier Entwickler die Tutorials und Foreneinträge der React Community weitestgehend nutzen, um aufgetretene Probleme zu lösen.

Fazit Große Community Bei dem Bewertungskriterium *Große Community* liegt React mit den meisten npm-Downloads pro Woche und der Tag-Anzahl auf Stack Overflow auf Platz eins, gefolgt von Vue.
Ionic hat im Vergleich zu diesen beiden Frameworks eine viel kleinere Community. Dennoch soll die gesamte Ionic Community aktiv an der Entwicklung des Frameworks beteiligt sein.
Ionic React und Ionic Vue bilden auf Grundlage der npm-Downloads und Stack Overflow Tags das Schlusslicht. Es könnten zwar die Foren von Vue und React nützlich sein, um aufgetretene Fragen zu klären, die spezifisch die Funktionalitäten dieser Frameworks betreffen. Dennoch sind die Communities der beiden noch recht jungen Integrationen für Ionic im Vergleich um einiges kleiner und müssen auf Grund der gering ausfallenden Zahlen der beiden gewählten Vergleichsgrößen eine schlechtere Bewertung erhalten, die in Tabelle 3.4 einzusehen ist.

Tabelle 3.4: Kriterium Große Community

React	Vue	Ionic	Ionic React	Ionic Vue
+ +	+	o	-	-

Auswertung Geringer Einarbeitungsaufwand

Das Kriterium *Geringer Einarbeitungsaufwand* diskutiert, wie bereits im Unterkapitel *Kriterienkatalog* erläutert, wie aufwändig es ist, die grundlegenden Funktionalitäten des jeweiligen Webframeworks zu erlernen.

React React basiert auf HTML, CSS und JavaScript. TypeScript wird ebenfalls unterstützt. Sollte ein Entwickler in diesen Programmiersprachen gute Kenntnisse besitzen, soll der Einarbeitungsaufwand in React mit einer geringen Lernkurve einhergehen. [Rawat and Mahajan, 2020, S.699]
React nutzt JSX (Abk. für JavaScript XML) zur Erstellung von Komponenten. Diese React spezifische Syntax ist eine Erweiterung von JavaScript und ermöglicht es innerhalb von JavaScript HTML-Elemente zu verwenden. Die Einarbeitung in JSX wirkt sich stark auf die Lernkurve aus, da sie für den ein oder anderen Entwickler unübersichtlich und komplex wirken könnte und so das Verständnis erschwert. [Lay, 2019, S.23]
Obwohl es kein Muss ist, JSX in einem React Projekt zu verwenden, ist es dennoch empfehlenswert, da es eine gewisse Übersichtlichkeit über den Quellcode liefert und das Erstellen von Komponenten erleichtert, da HTML und JavaScript in einer Datei gebündelt verwendet werden und einzusehen sind. [Meta Platforms, 2022]
Das Erstellen eines neuen Projektes wird durch die Toolchain namens *Create React App* erleichtert. Diese integriert aber keinen Router in das React Projekt, der hilft zwischen Komponenten und einzelnen Seiten der Anwendung zu navigieren. Ein Entwickler muss einen Router somit selbst dem Projekt hinzufügen. [Facebook, 2022]
Die fehlende Integration des Routers soll unter anderem Entwicklern die Freiheit geben selbst entscheiden zu können, mit welchen Tools und Bibliotheken diese ihr Projekt aufbauen möchten. Bei den meisten Bibliotheken für das React Framework handelt es sich um Bibliotheken von Dritten, so auch beim React Router. [Code-Boost, 2020]
Vor allem Neueinsteiger müssen sich dadurch einen Überblick über die wichtigsten existierenden Bibliotheken verschaffen, wie ausgereift diese sind und ob diese zu ihrem aktuellen Projekt passen und in dieses integrierbar sind.
Im Bewertungskriterium *Ausführliche Dokumentation* wurde bereits die rasche Weiterentwicklung des Webframeworks angesprochen, die unter anderem durch die große React Community vorangetrieben wird. Dadurch müssen sich Entwickler stets über aktuelle Änderungen auf dem Laufenden halten, um weiterhin mit den Technologien des Frameworks effizient arbeiten zu können. [Kugler, 2022]
Auch die dadurch teilweise veraltete Dokumentation kann sich negativ auf den Einarbeitungsaufwand auswirken. Im Gegensatz dazu existieren durch die Größe der React Community viele Tutorials und Leitfäden, um Fragen zu klären.

Vue Vue basiert wie React auf HTML, CSS und JavaScript. Vor allem seit Vue3 wird aber auch TypeScript unterstützt. Ähnlich zu Reacts Create React App nutzt Vue den *Vue CLI*, um die Erstellung eines Projektes zu vereinfachen. Einen Router müssen dabei Entwickler ebenfalls selbst dem Projekt hinzufügen. Dieser wird aber anders als von React von Vue selbst zur Verfügung gestellt.
Vue bietet ähnlich zu React die Möglichkeit, ein Projekt durch verschiedene Bibliotheken den eigenen Wünschen entsprechend aufzubauen. Hierbei ist aber auch die

Kompatibilität mit dem eigenen Projekt zu beachten.
Der Fokus des Webframeworks liegt im Vergleich zu React nicht auf JavaScript, sondern auf HTML und CSS, was die Einarbeitung in das Framework zusätzlich erleichtern soll.
Zur Erstellung von Komponenten können Entwickler die SFC (Abk. für Single File Components) Syntax von Vue nutzen. Diese bündelt in einer einzigen Datei das Template (HTML), das Styling (CSS) und die Logik (JavaScript) einer Komponente. Die Vermischung von HTML-Elementen mit JavaScript, wie es bei JSX vorgesehen ist, ist dabei aber nicht möglich. Es besteht aber die Möglichkeit, diese SFC Syntax nicht zu nutzen. Des Weiteren kann JSX ebenfalls bei Bedarf in Vue verwendet werden. [Vuejs, 2022a] Die Dokumentation des Webframeworks dient für die Einarbeitung in das Webframework als gute Stütze. Auch die Größe der Community kann durch die größere Auswahl an Tutorials helfen, den Einarbeitungsaufwand zu verringern.

Ionic Ionic nutzt keine Framework-spezifische Syntax. Es basiert auf HTML, CSS und JavaScript und unterstützt zudem TypeScript, wobei TypeScript bei der Erstellung eines Projektes als Standard dient. Bevorzugt ein Entwickler JavaScript, ist dies entsprechend anpassbar.
Ionic selbst hat durch den Verzicht auf eine eigene Syntax zur Entwicklung von Komponenten eine geringe Lernkurve. [WebInfoMart, 2019] Da es auf AngularJS basiert, können Kenntnisse mit diesem zusätzlich hilfreich sein, um den Einarbeitungsaufwand weiter zu minimieren.
Den *Ionic CLI* können Entwickler als Hilfswerkzeug zur Erstellung eines Projektes nutzen, ohne zudem gleich zu Beginn einen Router oder andere Bibliotheken zusätzlich integrieren zu müssen. Auch die ausführliche Dokumentation des Frameworks hilft dabei, die grundlegenden Funktionen von Ionic schnell verstehen zu können. [Ionic, 2022b] Durch die vergleichsweise kleinere Community könnte es allerdings schwer sein, weitere hilfreiche Tutorials und Leitfäden zu finden, um Fragen zu klären.

Ionic Vue Durch die Kombination von Ionic und Vue in Ionic Vue basiert auch dieses auf HTML, CSS und JavaScript beziehungsweise TypeScript. TypeScript wird dabei auch als Standard bei der Erstellung eines Projektes über den Ionic CLI verwendet. Die Nutzung von JavaScript statt TypeScript ist aber weiterhin möglich und muss bei gewünschter Änderung entsprechend im Projekt angepasst werden.
Der Einarbeitungsaufwand in Ionic Vue hängt stark von den Kenntnissen des Entwicklers zu dem Vue Framework ab. Ist ein Entwickler bereits mit Vue vertraut, ist die Lernkurve weiterhin relativ gering. Muss dieser sich aber noch beispielsweise mit dem Vue Router, der von Ionic Vue verwendet wird, oder anderen Funktionen Vues befassen, kann der Aufwand steigen.
Die Dokumentation von Ionic Vue bietet genügend Informationen, um die grundlegenden Funktionen zu verstehen. Für weitere Ausführungen müsste aber, wie bereits im Bewertungskriterium *Ausführliche Dokumentation* erwähnt, die Dokumentation von Vue selbst zu Rate gezogen werden. [Ionic, 2022a] Die kleine Community hinter Ionic Vue könnte es Entwicklern zudem erschweren, passende Tutorials und Leitfäden bei zusätzlichen Fragen zu finden.

Ionic React Wie Ionic Vue nutzt auch Ionic React als Standard TypeScript. Die Nutzung von JavaScript ist, zusammen mit HTML und CSS, aber ebenfalls möglich.
Ionic React bietet bei der Erstellung eines Projektes ohne zusätzliche Einbindungen mehr Funktionen als React. So wird gleich bei der Erstellung eines Projektes mittels Ionic CLI der React Router eingebunden, was es vor allem unerfahreneren Entwicklern erleichtert, mit der Implementierung eines Projektes zu beginnen.
Dadurch, dass Ionic React grundlegend dieselben Funktionen wie React anbietet, kann auch JSX optional für das Erstellen von Komponenten zum Einsatz kommen. Hierbei geht, wie bereits bei React dargelegt, ein größerer Einarbeitungsaufwand einher. Ähnlich zu Ionic Vue hilft die Dokumentation von Ionic React bei der Einarbeitung die Grundfunktionen des Webframeworks zu verstehen. Für nähere Erläuterungen, beispielsweise zu JSX, müssen Entwickler aber in der React Dokumentation nachschlagen. [Ionic, 2022c] Die kleinere Community könnte auch hier, ähnlich zu Ionic Vue, die Suche nach passenden Leitfäden zur Klärung von zusätzlichen Fragen erschweren.

Fazit Geringer Einarbeitungsaufwand Alle Webframeworks stellen ein Hilfswerkzeug beziehungsweise ein Tool zur Verfügung, das helfen soll, ein Projekt zu erstellen. Ionics Ionic CLI bindet dabei bereits alle notwendigen Bibliotheken ein. Bei Vue und React ist hingegen beispielsweise noch die Integration eines Routers nötig. Die Suche nach passenden Bibliotheken für Vue und React kann vor allem für Neueinsteiger zu Beginn schwierig sein, weshalb den beiden Webframeworks an dieser Stelle eine Bewertungsstufe abgezogen wird.
Vue und Ionic gehören in diesem Vergleich zu den Frameworks mit der geringsten Lernkurve. Vue fokussiert sich hauptsächlich auf HTML und CSS. Auch seine SFC Syntax soll relativ leicht zu verstehen sein. Laut einer Umfrage aus dem Jahr 2020, in der Angular, Vue und React verglichen wurden, soll Vue von diesen dreien am schnellsten erlernbar sein. [Salomaa, 2020, S.29] Ionic hat im Vergleich zu Vue keine besondere Syntax aufzuweisen. Entwickler können sich somit durch grundlegendes Wissen in HTML, CSS und JavaScript/TypeScript schnell in das Framework einarbeiten, wobei zusätzliche Hintergrundkenntnisse zu Angular für das Verständnis von Ionic von Vorteil sein können. Die kleinere Communitygröße und die dadurch kleinere Auswahl an Tutorials und Leitfäden könnte bei zusätzlichen Fragen allerdings die Einarbeitung erschweren, weshalb Ionic an dieser Stelle eine schlechtere Wertungsstufe erhält.
React bildet bei dem Einarbeitungsaufwand das Schlusslicht. Reacts optional nutzbare Syntax JSX erleichtert zwar die Implementierung von Komponenten und gestaltet diese übersichtlicher, dennoch müssen sich Entwickler mit dieser zuvor genügend vertraut machen. Vor allem die teils veraltete Dokumentation kann die Einarbeitung in das Framework erschweren. [Lay, 2019, S.23]
Die größere Auswahl an Tutorials und Leitfäden durch die größere Community könnte dem gut entgegenwirken, auch wenn ein Entwickler dazu erst nach passenden und guten Leitfäden suchen muss.
Der Einarbeitungsaufwand von Ionic Vue und Ionic React hängt stark davon ab, wie ausgereift die Kenntnisse des jeweiligen Entwicklers in Vue oder React sind. Ionic CLI bietet zwar die nötige Unterstützung ein neues Projekt aufzusetzen, dennoch

muss man weitestgehend die Funktionen des integrierten Frameworks verstehen und bei Bedarf die Dokumentation von React oder Vue selbst zu Rate ziehen. Die sehr kleinen Communities hinter den beiden Integrationen lassen zudem einen Mangel an passenden Tutorials speziell für diese vermuten.
Aus besagten Gründen und da anhand der Literaturrecherche selbst keine explizite Aussage über den Einarbeitungsaufwand der beiden Integrationen gefunden werden konnte, wird diesen als Ergebnis *ohne Wertung* in die Tabelle 3.5 für das Kriterium *Geringer Einarbeitungsaufwand* eingetragen.

Tabelle 3.5: Kriterium Geringer Einarbeitungsaufwand

React	Vue	Ionic	Ionic React	Ionic Vue
o	+	+	k.W.	k.W.

Auswertung UI Gestaltungsmöglichkeiten

Wie bereits im Unterkapitel *Kriterienkatalog* erläutert, soll dieses Bewertungskriterium prüfen, welche Auswahl an Gestaltungsmöglichkeiten das jeweilige Webframework einem Entwickler bietet, um eine ansprechende Benutzeroberfläche zu erstellen. Auch, ob das Aussehen nativer Anwendungen umsetzbar ist, ist vor allem für das Konzept der Progressive Web App wichtig.

React Für React existiert eine große Auswahl an Bibliotheken, welche vorgefertigte UI-Komponenten beinhalten. Eine empfohlene Bibliothek ist *Material UI*, die auch starke Unterstützung von der React Community erfährt. [Rawat and Mahajan, 2020] Auch für Animationen existieren verschiedene Bibliotheken, wie *React Motion* oder *React Reveal*. An sich hat React auch eine eigene Animations-Bibliothek namens *React Transition Group*, die ein Entwickler zusätzlich seinem Projekt hinzufügen kann. Animationen können dabei auch selbst implementiert werden.
Für die Gestaltung des nativen Aussehens einer Anwendung können Entwickler die *Onsen-UI*-Bibliothek einbinden. Diese erlaubt es, die Benutzeroberfläche automatisch an das Betriebssystem des Endgerätes anzupassen, auf welchem die Anwendung zum Einsatz kommt. So wird auf einem Android-Gerät die Benutzeroberfläche anders dargestellt, als auf einem IOS-Gerät. [Onsen UI, 2022]

Vue Vue bietet ähnlich zu React die Möglichkeit Bibliotheken einzubinden, die vorgefertigte Komponenten für die Benutzeroberfläche zur Verfügung stellen. Als Beispiel dienen hier *Vant* oder *Vuetify*.
Grundlegende Animationen können Entwickler in Vue selbst implementieren. Zudem können sie auch Bibliotheken nutzen, um bestimmte vorgefertigte Animationen zu verwenden. Onsen UI ist ebenfalls – ähnlich zu React – in einem Vue Projekt nutzbar und bietet dabei dieselben Funktionalitäten. [Onsen UI, 2022]

Ionic Ionic selbst stellt bereits vorgefertigte UI-Komponenten zur Erstellung von Benutzeroberflächen zur Verfügung. Auch auf Icons und Animationen kann ein Entwickler zugreifen.

Ionic Animations ermöglicht die Umsetzung von Animationen. Gestengesteuerte Animationen, die beispielweise durch einen Doppelklick oder ein Wischen nach rechts oder links ausgelöst werden, sind zudem mit *Ionic Gestures* umsetzbar.
Ähnlich zur Onsen-UI-Bibliothek, bietet Ionic selbst die Funktionalität an, die Benutzeroberfläche einer Anwendung entsprechend der aktuell verwendeten Plattform anzupassen. Ionic nennt diese Funktion *Adaptive Styling*. [Ionic, 2022b]

Ionic Vue Da Ionic Vue eine Kombination aus Ionic und Vue ist, können Entwickler entsprechend Bibliotheken und Funktionalitäten, wie das Adaptive Styling, zur Erstellung von Benutzeroberflächen aus beiden Frameworks verwenden.
Dabei muss aber beachtet werden, dass die einzubindenden Bibliotheken des Vue Frameworks die Vue3 Version unterstützen, da Ionic Vue auf Vue3 basiert. [Ionic, 2022a]

Ionic React Ähnlich zu Ionic Vue können auch hier Entwickler Funktionalitäten und Bibliotheken aus React und Ionic nutzen. Die Kompatibilität der Bibliotheken mit Ionic React ist dabei ebenfalls zu beachten.
Eine große Auswahl an vorgefertigten Komponenten zur Gestaltung von Benutzeroberflächen liegt also auch hier vor. Auch das Adaptive Styling und Animationen sind somit nutzbar. [Ionic, 2022c]

Fazit UI Gestaltungsmöglichkeiten Alle Webframeworks bieten die Möglichkeit, vorgefertigte Komponenten zu nutzen. Bei Vue und React steht dabei eine große Auswahl an Bibliotheken zur Verfügung, die Entwickler in ihr Projekt einbinden können. Ionic hat bereits eine große Auswahl an vorgefertigten UI-Komponenten integriert.
Ionic, Ionic Vue und Ionic React können durch das Adaptive Styling das Aussehen der Benutzeroberfläche an die jeweilige Plattform, auf der die Anwendung ausgeführt wird, anpassen. Dies ist auch bei Vue und React möglich, allerdings müssen dazu Entwickler noch eine zusätzliche Bibliothek, wie Onsen UI, einbinden. Die zusätzliche Einbindung von Bibliotheken und der dabei einhergehende Aufwand für Entwickler, ist für dieses Kriterium allerdings nicht von Belang und nimmt somit keinen Einfluss auf die Bewertung.
Alle Webframeworks bieten im Großen und Ganzen eine große Auswahl an Möglichkeiten, um eine Benutzeroberfläche entwickeln, gestalten und plattformspezifisch anpassen zu können, weshalb jedes Webframework in Tabelle 3.6 die bestmögliche Wertung erhält.

Tabelle 3.6: Kriterium UI Gestaltungsmöglichkeiten

React	Vue	Ionic	Ionic React	Ionic Vue
+ +	+ +	+ +	+ +	+ +

Auswertung Einfache Nutzung nativer Funktionen

Die Nutzung und der Zugriff auf native Funktionalitäten, wie die Kamera eines Endgerätes, ist vor allem für Progressive Web Apps wichtig, um einem Benutzer das Gefühl einer nativen Anwendung zu vermitteln.

Inwiefern ein Webframework es Entwicklern ermöglicht, diese nativen Funktionen in ihrer Anwendung möglichst leicht einzubinden, wird im Folgenden erörtert.

React Um bei React native Funktionen nutzen zu können, sind entsprechende Bibliotheken einzubinden. Hierbei müssen Entwickler natürlich darauf achten, dass die jeweilige Bibliothek mit der aktuellen React Version kompatibel ist. Um auf die Kamera eines Endgerätes zuzugreifen, kann beispielsweise *React-webcam* zum Einsatz kommen. Mit der Bibliothek *Satelize* kann das GPS eines Gerätes genutzt werden oder *howler.js* ermöglicht den Zugriff auf das Mikrofon. [Bastakoti, 2022, S.10,S.12,S.16] Neben der Einbindung von Bibliotheken ist React aber auch mit anderen Frameworks integrierbar, die native Funktionalitäten anbieten, wie zum Beispiel Cordova oder Ionic. Zudem ist die Integration von Web APIs möglich, deren Funktionalität allerdings von der Unterstützung durch den jeweiligen Browser abhängt, in dem die Anwendung ausgeführt wird.

Vue Ähnlich zu React müssen Entwickler auch bei Vue auf Bibliotheken zurückgreifen, um native Funktionen nutzen zu können. Für Push-Benachrichtigungen existiert beispielsweise *Vue-snotify*. Da aber die Community von Vue kleiner ist als Reacts, ist es schwerer, passende Bibliotheken für native Funktionen zu finden.
Dennoch können Entwickler auch Vue mit anderen Webframeworks verknüpfen, wie zum Beispiel mit Cordova oder Ionic, oder Web APIs nutzen, die native Funktionalitäten anbieten.

Ionic Da Ionic unter anderem auf Apache Cordova basiert, bietet das Ionic Framework selbst den Zugriff auf native Funktionalitäten an. Hierzu entwickelte das Ionic-Team *Capacitor*, welches native Plugins zur Verfügung stellt.
Capacitor kann über den Ionic CLI genutzt werden und bietet unter anderem Funktionalitäten wie den Kamerazugriff, Push-Benachrichtigungen, oder Geolocation an. Neben Capacitor wäre auch die Nutzung von Web APIs möglich.
Im Vergleich zu React und Vue führt Ionic in seiner offiziellen Dokumentation Hinweise zur Nutzung von nativen Funktionen auf. [Ionic, 2022b]

Ionic Vue Ionic Vue hat Zugriff auf Ionics Capacitor und kann somit auf native Funktionen zugreifen. In der offiziellen Dokumentation von Ionic Vue existiert zudem ein Leitfaden, Capacitor in das Projekt einzubinden. [Ionic, 2022a] Auch Web APIs sind nutzbar.

Ionic React Ionic React hat wie Ionic Vue Zugriff auf Capacitor und kann somit ebenfalls auf native Funktionen zugreifen. Auch in der offiziellen Dokumentation von Ionic React existiert ein Leitfaden, wie Capacitor in das Projekt einzubinden ist. [Ionic, 2022c] Ebenfalls sind auch hier Web APIs nutzbar.

Fazit Einfache Nutzung nativer Funktionen Ionic, Ionic Vue und Ionic React bieten mit Ionics Capicator die komfortabelste Möglichkeit für Entwickler, auf native Funktionalitäten eines Endgerätes zuzugreifen. Dabei werden gleich mehrere native

Funktionen angeboten, wie Push-Benachrichtigungen oder der Zugriff auf die Kamera. Es müssen keine zusätzlichen Bibliotheken oder Web APIs für jede einzelne native Funktionalität gesucht werden, wie es bei Vue und React der Fall ist. Auch der Aufwand, ein weiteres Framework wie Ionic oder Cordova in ein Projekt einzubinden, erübrigt sich mit Ionic, Ionic Vue und Ionic React.
Da vor allem in diesem Kriterium der Fokus darauf liegt, dass Entwickler möglichst ohne großen Aufwand native Funktionalitäten in ihrer Anwendung einbinden können, muss React und Vue durch die nötige Suche und Einbindung von Bibliotheken oder Web APIs eine schlechtere Wertung zugesprochen werden. Auch die fehlenden Leitfäden zu nativen Funktionalitäten in der Dokumentation ist hier als Minuspunkt zu werten. Die folgende Tabelle 3.7 zeigt die Bewertungen der einzelnen Frameworks in dieser Kategorie.

Tabelle 3.7: Kriterium Einfache Nutzung nativer Funktionen

React	Vue	Ionic	Ionic React	Ionic Vue
o	o	+ +	+ +	+ +

Auswertung Testen

Dieses Bewertungskriterium prüft, ob das jeweilige Webframework Teststarten wie Unittests, Integrationstest oder End-To-End Tests unterstützt. Auch inwiefern Leitfäden für Entwickler existieren, soll beachtet werden.

React React ermöglicht die Umsetzung von Unittests, Integrationstests und End-to-End-Tests.
In React wird das Test-Framework namens *Jest* und die *React-Testing*-Bibliothek für Unittests und Integrationstest verwendet, wobei sich die React-Testing-Bibliothek besser für Integrationstests eignet, als Unittests. [Vasilev, 2021, S.30]
Enzyme oder *React Test Renderer* sollen sich besser für Unittests in React eignen, da diese im Vergleich zur React-Testing-Bibliothek die React Komponenten isolierter testen. [Vasilev, 2021, S.36]
Jest und die React-Testing-Bibliothek werden automatisch dem React Projekt hinzugefügt, wenn dieses mit Hilfe von Create React App erstellt wird. [Vasilev, 2021, S.21]
Ansonsten müssen Jest und die React-Testing-Bibliothek selbst eingebunden werden, falls diese zum Testen zum Einsatz kommen sollen.
React führt zur Erstellung und Ausführung von Tests Leitfäden samt Beispielen in der Dokumentation zur Verfügung. Auch weitere Test-Frameworks und Bibliotheken werden neben Jest und der React-Testing-Bibliothek aufgelistet, die verwendet werden können, um Tests umzusetzen.
Für End-to-End-Tests empfiehlt React das Test-Framework *Cypress*, das Test-Framework *Playwright* oder die Bibliothek *Puppeteer*. [Meta Platforms, 2022]

Vue Vue führt ebenfalls in der Dokumentation Teststrategien und Beispiele auf, um Vue Anwendungen zu testen. Es werden zu jedem Test-Typ Empfehlungen ausgesprochen, welche Frameworks und Bibliotheken am besten zu nutzen sind.

Vitest wird als Test-Framework für Unittests empfohlen. Es kann auch für Integrationstests genutzt werden. Vitest wurde von Vue selbst entwickelt und lässt sich somit einfach in Vue Projekten nutzen. Vitest ist aber noch relativ neu und es wird auch noch weiter an diesem Framework entwickelt, weshalb es noch nicht gänzlich stabil ist.
Als Alternative zu Vitest empfiehlt Vue *Peeky* zur Durchführung von Unittests.
Des Weiteren kann für das Testen von Komponenten die *Vue Test Utils* verwendet werden – eine Bibliothek, die entsprechende Testing-Werkzeuge bereitstellt. Vue empfiehlt das Test-Framework Cypress für End-to-End-Tests. [Vuejs, 2022a]

Ionic Ionic hat ebenfalls in seiner Dokumentation Hinweise mit Beispielen zum Testen von Applikationen aufgeführt. Ionic nutzt die Bibliothek *Karma* und das Framework *Jasmine* für Unittests und Integrationstests, wobei Karma die Tests ausführt und mit Hilfe von Jasmine die Tests geschrieben werden. [Johnson, 2022] Bei webbasierten Anwendungen ohne native Funktionalitäten wird für End-to-End-Tests Cypress oder *Selenium* genutzt.
Obwohl Ionic den Zugriff auf native Funktionen für eine Anwendung vereinfacht, soll das Testen dieser Funktionalitäten recht aufwändig sein. Es können zum Testen dieser *Appium* und *WebdriverIO* genutzt werden, um Funktionen der Anwendung auf verschiedenen Plattformen wie Android oder IOS zu testen. [Lynch, 2022]
WebdriverIO automatisiert dabei Tests. Appium simuliert native Umgebungen, um native Funktionalitäten zu testen. [Ionic, 2022b]
Die Konfiguration dieser Werkzeuge in Ionic Projekten ist aber nicht intuitiv und einfach. Auf Grund dieser Schwierigkeit, native Funktionen relativ einfach zu testen, bietet Ionic, vor allem für End-to-End-Tests, noch keine starke Test-Kultur innerhalb seiner Community. Ionic möchte dies aber in Zukunft ändern und durch *Ionic E2E* das Testen von nativen Funktionen mittels End-to-End-Tests vereinfachen. Ionic E2E kann bereits genutzt werden. [Lynch, 2022]

Ionic Vue Ionic Vue führt keinerlei Leitfäden zu Teststrategien in der Dokumentation auf. Für End-to-End-Tests wird aber Cypress empfohlen. [ionic-team, 2022] Inwiefern beispielsweise Vue Test Utils mit Jest oder Vitest zur Durchführung von Unittests in Ionic Vue genutzt werden können, ist durch die Literaturrecherche nicht feststellbar. Es sollte aber möglich sein, solange Vue3 von diesen unterstützt wird.

Ionic React Ionic React führt in der Dokumentation Hinweise zum Testen einer Ionic React Anwendung auf.
Wenn ein Ionic React Projekt mit Hilfe von Ionic CLI erstellt wird, nutzt dieses automatisch Jest und die React-Testing-Bibliothek für Unittests und Integrationstests. Zusätzlich bietet Ionic React mit *Ionic React Test Utils* Funktionalitäten, die helfen sollen, im speziellen Ionic Komponenten innerhalb der Ionic React App zu testen. [Ionic, 2022c] Für End-to-End-Tests findet sich nichts in der Dokumentation.

Fazit Testen Alle Frameworks bieten grundsätzlich die Möglichkeit, Unittest, Integrationstests und End-to-End-Tests durchzuführen. Dabei benötigen die Webframeworks meist zusätzliche Bibliotheken oder Test-Frameworks, die speziell für das Testen von

Anwendungen ausgelegt sind. Vue bietet Entwicklern mit Vitest dabei ein eigenes Test-Framework an, um Unittests und Integrationstests durchzuführen.
Bis auf Ionic Vue existieren in jedem Framework Leitfäden zum Testen einer Anwendung. Am wenigsten ausführlich ist dabei der Leitfaden in Ionic Reacts Dokumentation, vor allem bezüglich End-to-End-Tests. Deshalb erhält Ionic React eine schlechtere Wertungsstufe.
An sich müsste Ionic Vue durch den fehlenden Leitfaden zum Testen einer Anwendung eine schlechtere Wertung als Ionic React erhalten. Trotz des fehlenden Leitfadens kann auch davon ausgegangen werden, dass Entwickler in Ionic Vue alle drei Testarten nutzen können. Da man aber durch den Mangel an vorhandener Literatur für Ionic Vue keine eindeutige und sichere Aussage darüber treffen kann, inwiefern dieses Webframework Unittests und Integrationstests tatsächlich unterstützt oder welche Bibliotheken oder Frameworks Entwickler hierzu nutzen sollten, erhält Ionic Vue in der Tabelle 3.8 keine Wertung.

Tabelle 3.8: Kriterium Testen

React	Vue	Ionic	Ionic React	Ionic Vue
+ +	+ +	+ +	+	k.W.

Auswertung PWA Unterstützung

Da in dieser Arbeit das Konzept der Progressive Web App im Fokus steht, gibt dieses Bewertungskriterium Auskunft darüber, wie gut die jeweiligen Webframeworks dieses Konzept unterstützen.

React Entwickler können Progressive Web Apps mit React durch zusätzliche Einbindungen von Bibliotheken und anderen Werkzeugen entwickeln. Mit Hilfe von Create React App kann direkt ein Projekt für eine Progressive Web App erstellt werden. [Devathon Team, 2020] React führt in seiner offiziellen Dokumentation, wie bereits im Bewertungskriterium *Ausführliche Dokumentation* erwähnt, keine Leitfäden zur Erstellung einer Progressive Web App.
React kann zwar native Funktionen für Progressive Web Apps durch die zusätzliche Einbindung von Bibliotheken bereitstellen, allerdings ist es aufwändig, passende Bibliotheken zu finden. Die Nutzung eines zusätzlichen Frameworks oder von Web APIs für native Funktionalitäten ist, wie im Bewertungskriterium *Einfache Nutzung nativer Funktionen* erwähnt, aber ebenfalls möglich.
Die Benutzeroberfläche einer Progressive Web App ist durch Einbindung zusätzlicher Bibliotheken je nach Plattform anpassbar, um das Aussehen einer nativen Anwendung nachzuahmen.
Aus dem React Ökosystem kann an dieser Stelle auch *Next.js* erwähnt werden. Next.js basiert auf React und soll helfen, die Entwicklung von Projekten zu vereinfachen, indem es vorgefertigte Funktionalitäten, beispielsweise für das Routing in einer Anwendung, zur Verfügung stellt. Auch die Entwicklung von Progressive Web Apps soll dieses Framework vereinfachen. [Vercel, 2022] Da es sich bei Next.js aber um ein eigenständiges Framework handelt, wird dieses für den Vergleich nicht mit in die Bewertung aufgenommen.

Vue Auch Vue führt, wie bereits im Bewertungskriterium *Ausführliche Dokumentation* erläutert, keine näheren Leitfäden zu Progressive Web Apps.
Dennoch können über die Hilfswerkzeuge Vue CLI oder *Vite* Progressive Web Apps erstellt werden. [Vuejs, 2022a] Native Funktionalitäten sind durch zusätzliche Bibliotheken und Web APIs nutzbar.
Die Benutzeroberfläche einer Progressive Web App ist auch bei Vue durch die Einbindung zusätzlicher Bibliotheken je nach Plattform anpassbar.
Ähnlich zu React existieren auch im Vue Ökosystem Alternativen, um eine Progressive Web App zu erstellen, wie das Framework *Quasar*. Dieses ist ähnlich wie Ionic dazu geeignet, plattformübergreifende Anwendungen zu entwickeln. Inwiefern sich Quasar und Ionic voneinander unterscheiden, wird aber in dieser Arbeit nicht näher beleuchtet.
Eine weitere Möglichkeit stellt die Nutzung von *Nuxt.js* dar. Es soll ähnlich zu Reacts Next.js Funktionalitäten bereitstellen, um Vue Projekte einfacher zu entwickeln. [Vuejs, 2022a] Da es sich bei Quasar und Nuxt.js ebenfalls um eigenständige Frameworks handelt, werden auch diese bei der Bewertung dieses Kriteriums nicht mit einbezogen.

Ionic Ionic unterstützt offiziell Progressive Web Apps und bietet entsprechende Leitfäden an. Mit Hilfe des Ionic CLI können Entwickler Projekte zur Umsetzung von Progressive Web Apps erstellen. [Ionic, 2022b]
Native Funktionalitäten, wie Geolocation oder der Zugriff auf die Kamera, sind vor allem durch Ionics Capacitor für Progressive Web Apps nutzbar.
Auch die auf IOS und Android abgestimmten, vorgefertigten Komponenten, sowie das Adaptive Styling von Ionic erlauben es, die Benutzeroberfläche der Progressive Web App ähnlich einer nativen Anwendung aufzubauen und je nach Plattform anzupassen. [Ionic, 2022b]

Ionic Vue Eine Progressive Web App ist bei Ionic Vue ähnlich zu Vue mit der Vue CLI erstellbar.
Ionic Vue erlaubt es, von beiden Frameworks Funktionalitäten zu beziehen, die das Entwickeln einer Progressive Web App erleichtern.
Ionics Capacitor liefert dabei den Zugriff auf native Funktionalitäten. Auch das Adaptive Styling von Ionic, sowie Bibliotheken aus Vue3, können dabei genutzt werden.
In der Dokumentation von Ionic Vue existiert zudem ein Leitfaden zur Erstellung einer Progressive Web App. [Ionic, 2022a]

Ionic React Bei Ionic React wird eine Progressive Web App über den Ionic CLI erstellt.
In Ionic Reacts Dokumentation existiert außerdem ein Leitfaden zur Entwicklung einer Progressive Web App. Wie Ionic Vue kann auch Ionic React aus Ionic und React Funktionalitäten zur Entwicklung einer Progressive Web App nutzen. Dazu gehören unter anderem auch hier die nativen Funktionalitäten durch Capacitor, das Adaptive Styling oder zusätzliche Bibliotheken aus dem React Framework, die Entwickler bei Bedarf einbinden können. [Ionic, 2022c]

Fazit PWA Unterstützung Alle Webframeworks des Vergleichs ermöglichen es Entwicklern, eine Progressive Web App in ihren Grundfunktionen zu entwickeln. Bereitgestellte Werkzeuge dienen dabei als Hilfestellung, diese schnell und einfach zu erstellen. Ionic liefert mit seinem Adaptive Styling und den nativen Funktionalitäten durch Capacitor eine komfortablere Möglichkeit für Entwickler, eine Progressive Web App entsprechend dem Abbild einer nativen Anwendung umzusetzen. Auch Ionic Vue und Ionic React können diese Funktionalitäten nutzen.
React und Vue benötigen zusätzliche Einbindungen, um Funktionalitäten, ähnlich dem Adaptive Styling, oder jeweilige native Funktionen nutzen zu können. Da im Kriterium *UI Gestaltungsmöglichkeiten* die nötige, zusätzliche Einbindung von Bibliotheken nicht in die Bewertung mit eingeflossen ist, erfolgt dies hier ebenfalls nicht. Für den Punkt der nativen Funktionalitäten muss aber, ähnlich zum Kriterium *Einfache Nutzung nativer Funktionen*, für die nötige, zusätzliche Einbindung von Bibliotheken oder Web APIs eine Wertungsstufe abgezogen werden.
Vue und React führen zudem keine PWA-Leitfäden in ihren Dokumentationen, was ebenfalls eine schlechtere Bewertung dieser beiden Webframeworks erfordert. Ionic, Ionic Vue und Ionic React stellen im Gegensatz dazu einen Leitfaden zur Verfügung.
Ionic, Ionic Vue und Ionic React bieten bei diesem Vergleich auf Grund der Literaturrecherche die beste Unterstützung für Progressive Web Apps. Vue und React erhalten in der folgenden Tabelle 3.9 im Vergleich schlechtere Wertungen.

Tabelle 3.9: Kriterium PWA Unterstützung

React	Vue	Ionic	Ionic React	Ionic Vue
o	o	+ +	+ +	+ +

3.3 Ergebnis des Vergleichs

Im Folgenden zeigt die Tabelle 3.10 den vollständig ausgewerteten Kriterienkatalog.

Tabelle 3.10: Ausgewerteter Kriterienkatalog

Bewertungskriterium	React	Vue	Ionic	Ionic React	Ionic Vue
Ausführliche Dokumentation	o	+	+ +	+	o
Große Community	+ +	+	o	-	-
Geringer Einarbeitungsaufwand	o	+	+	k.W.	k.W.
UI Gestaltungsmöglichkeiten	+ +	+ +	+ +	+ +	+ +
Einfache Nutzung nativer Funktionen	o	o	+ +	+ +	+ +
Testen	+ +	+ +	+ +	+	k.W.
PWA Unterstützung	o	o	+ +	+ +	+ +

React, Ionic React und Ionic Vue erhielten jeweils in insgesamt drei Kategorien die Wertung *trifft zu (+ +)*. React konnte vor allem mit seiner großen Community punkten. Vue stach zusammen mit Ionic in der Kategorie *Geringer Einarbeitungsaufwand* hervor. Hauptsächlich wegen seiner JSX Syntax und der teilweise veralteten Dokumentation, erhielt React im Bewertungskriterium *Geringer Einarbeitungsaufwand* eine seiner schlechtesten Bewertungen mit *trifft teilweise zu (o)*. Diese Bewertung wurde React auch in den Kriterien *Ausführliche Dokumentation*, *Einfache Nutzung nativer Funktionen* und *PWA Unterstützung* zugesprochen. Auch Vue schnitt in den Kategorien *Einfache Nutzung nativer Funktionen* und *PWA Unterstützung* am schlechtesten ab. Ionic lag in der Kategorie *Große Community* hinter React und Vue.
Die schlechtesten Wertungen *trifft kaum zu (-)* erhielten Ionic React und Ionic Vue im gesamten Vergleich in der Kategorie *Große Community*.
Ionic React und Ionic Vue erhielten im Kriterium *Geringer Einarbeitungsaufwand* keine Wertung.
Auch im Kriterium *Testen* wurde Ionic Vue keine Wertung zugesprochen. Auf Grund der mangelnden Auswahl an aussagekräftiger Literatur, konnte man hier keine passenden Bewertungsentscheidungen treffen.
Ionic erhielt in diesem Vergleich fünf Wertungen mit *trifft zu (+ +)* und liegt deshalb auf Platz eins. Insbesondere in den Bewertungskriterien *UI Gestaltungsmöglichkeiten*, *Einfache Nutzung nativer Funktionen* und *PWA Unterstützung*, die besonders wichtig für die Entwicklung von Progressive Web Apps sind, schnitt Ionic sehr gut ab.

Wie bereits zu Beginn des Vergleichs erwähnt, soll mit dem Webframework, das am besten im Vergleich abschneidet, beispielhaft eine Progressive Web App entwickelt werden. Da Ionic den Vergleich für sich entschieden hat, erfolgt somit die Entwicklung der PWA mit diesem.

4. Planung der Progressive Web App

Bei der Progressive Web App, die im Rahmen dieser Arbeit entwickelt wird, handelt es sich um eine Reiseziele-Anwendung namens *memorest*. Neben den grundlegenden Funktionen einer PWA soll diese Anwendung folgende Funktionalitäten beinhalten:

1. Durchgeführte Reisen dokumentieren/einfügen
2. Einträge bearbeiten oder löschen
3. Zugriff auf die Kamera des jeweiligen Endgerätes ermöglichen (ausgenommen Desktop)

Der Schwerpunkt bei der Entwicklung dieser Anwendung bezieht sich auf die Gestaltung der hybriden Benutzeroberfläche. Dabei werden Design-Prinzipien für Benutzeroberflächen auf dem Desktop, sowie für Benutzeroberflächen auf mobilen Plattformen mit einbezogen. Auf diese Prinzipien wird im Unterkapitel 4.2 *Gestaltung der hybriden Benutzeroberfläche* näher eingegangen.

4.1 Architektur der Progressive Web App

Aufgrund des Ergebnisses des Webframework-Vergleichs erfolgt die Entwicklung der Progressive Web App mit Hilfe des Frameworks Ionic. Dabei wird die Programmiersprache TypeScript verwendet.
Zusätzlich wird die dokumentenbasierte Datenbank *PouchDB* genutzt. Diese ermöglicht es, nach dem Offline-First-Prinzip Daten clientseitig abzuspeichern und offline zugänglich zu machen. PouchDB stellt außerdem asynchrone APIs zur Verfügung, um Daten serverseitig abspeichern zu können.
Um es dem Benutzer zu ermöglichen, seine durch PouchDB im aufgerufenen Browser lokal gespeicherten Daten auch über andere Browser und Endgeräte abzurufen, müssen diese zusätzlich auf einem Server abgelegt werden. Für diese serverseitige Speicherung kommt die dokumentenbasierte Datenbank *CouchDB* zum Einsatz.
Der Vorteil in einem Projekt PouchDB und CouchDB zu verwenden ist, dass diese beiden Datenbanken ihre Daten von sich aus untereinander synchronisieren können. Fügt ein Benutzer neue Inhalte in der PWA ein, werden diese dabei zuerst lokal durch PouchDB abgespeichert, egal ob eine Internetverbindung vorhanden ist oder nicht. Anschließend erfolgt bei einer vorhandenen Internetverbindung die Synchronisation der

Inhalte mit CouchDB, wodurch der Zugriff auf die Daten über verschiedene Endgeräte und Browser gewährleistet werden kann.
Wie das Datenbankmodell für die PWA memorest aufgebaut sein soll, zeigt die folgende Abbildung 4.1 in Form eines Entity-Relationship-Modells (in der original Chen-Notation, zzgl. Min-Max-Kardinalitäten).

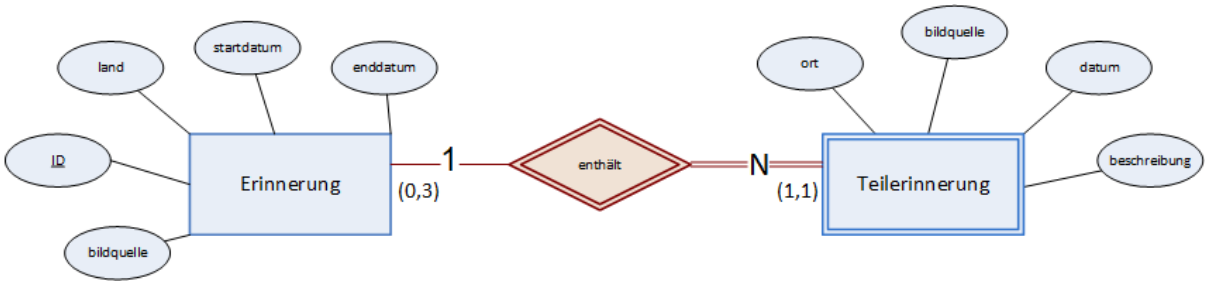

Abbildung 4.1: ER-Modell (in der original Chen-Notation, zzgl. Min-Max-Kardinalitäten), Quelle: Eigene Darstellung

Dabei existiert eine Entität namens *Erinnerung*, welche einen Primärschlüssel als Identifikationsnummer zur eindeutigen Identifikation und vier weitere Attribute annimmt. Diese vier Attribute sollen eine Reiseerinnerung in der Anwendung im Kern definieren. Hierzu zählt das Land, in dem die Reise stattfand, das Startdatum und Enddatum der Reise, sowie ein Bild dieser Reise. Diese Entität identifiziert die schwache Entität *Teilerinnerung* in einer identifizierenden Beziehung. Dabei kann eine Teilerinnerung ohne eine Entität Erinnerung nicht existieren. Die Entität Erinnerung hingegen kann ohne eine Teilerinnerung existieren oder bis zu maximal drei Teilerinnerungen zugeordnet bekommen. Die schwache Entität Teilerinnerung hat vier Attribute, in diese näheres zu der jeweiligen Teilerinnerung eingetragen werden soll. Zu diesen Attributen zählt zum einen der Ort, an dem die Teilerinnerung stattfand, das Datum, eine Beschreibung zu dieser Teilerinnerung und ein Bild. Da die schwache Entität Teilerinnerung von der Entität Erinnerung identifiziert wird, benötigt diese keinen eigenen Primärschlüssel.
Da CouchDB und PouchDB dokumentenbasierte Datenbanken sind, erfolgt die eigentliche Abspeicherung von Datenbankeinträgen im JSON-Format. Ein diesbezügliches Beispiel ist im Anhang A vorzufinden. Dieses hält die Erinnerung an eine Norwegenreise mit vier Teilerinnerungen zu vier verschiedenen norwegischen Städten fest. Das Bild der JSON-Datei ist mit Hilfe eines *Online JSON Viewer* erstellt worden.

4.2 Design-Prinzipien zur Gestaltung hybrider Benutzeroberflächen

Progressive Web Apps sind über den Browser eines Endgerätes aufrufbar und sind dadurch sowohl auf dem Smartphone, als auch dem Desktop nutzbar. Wie bereits im Kapitel *Grundlagen* erläutert, sollten Progressive Web Apps hierbei das Kriterium App-Like erfüllen und auf den Benutzer dadurch wie eine native Anwendung der jeweils verwendeten Plattform wirken. Dies kann zum einen durch die Bereitstellung nativer Funktionalitäten erreicht werden, wie dem Zugriff auf die Kamera des jeweiligen

Endgerätes oder dessen Standort. Zum anderen kann die Benutzeroberfläche der Progressive Web App dazu beitragen, dass die Anwendung als nativ oder annähernd nativ wahrgenommen wird. Die Progressive Web App muss hierfür ihre Benutzeroberfläche an die jeweils verwendete Plattform anpassen. So soll sich die Benutzeroberfläche der PWA auf dem Desktop von der Benutzeroberfläche auf mobilen Plattformen wie IOS oder Android unterscheiden, wodurch sich eine notwendige, hybride Gestaltung der Benutzeroberfläche einer Progressive Web App ergibt.
Das charakteristische Design und der Aufbau von Webseiten und Anwendungen auf dem Desktop, so wie auf mobilen Plattformen, ist dabei jeweils durch verschiedene Design-Prinzipien und Richtlinien definiert. Diese sollen sicherstellen, dass Anwendungen und Webseiten auf ihrer Plattform für Benutzer gut nutzbar sind. Auch IOS und Android haben jeweils ihre diesbezüglichen Empfehlungen in eigenen *Styleguides* definiert.

4.2.1 Allgemeine Grundlagen

Eine Benutzeroberfläche ist eine Schnittstelle zwischen Mensch und Maschine, also zwischen dem Benutzer und dem Endgerät, mit dem er interagieren möchte.
Das *User Interface Design* beschäftigt sich dabei mit der Problemstellung, die Funktionalitäten einer Anwendung oder Webseite dem Benutzer einfach und verständlich visuell mittels einer Benutzeroberfläche zugänglich zu machen. Durch richtiges User Interface Design kann man den Blick des Benutzers manipulieren und leiten und ihn dazu anregen, Aktionen innerhalb einer Anwendung auszuführen. Außerdem ist es möglich den Fokus auf wichtige Funktionalitäten zu setzen, damit diese schneller wahrgenommen werden. [Panzarella, 2022, S.75]

Der Aufbau und die Gestaltung einer Benutzeroberfläche kann maßgeblich die Benutzererfahrung – auch *User Experience* genannt – mit der jeweiligen Anwendung beeinflussen und darüber entscheiden, ob ein Benutzer erneut auf die Funktionalitäten der Anwendung zurückgreift oder nicht. [Hendrick, 2022, S.7]
Die User Experience umfasst somit die Gesamtheit der subjektiven Wahrnehmungen und Erfahrungen, die ein Benutzer mit einer Anwendung macht und wie sich diese auf die Beziehung des Benutzers zu der jeweiligen Anwendung auswirken. Auch, ob eine Anwendung die Erwartungen eines Benutzers an diese erfüllt, spielt hier eine Rolle. [Mulligan, 2021, S.17]
Das *User Experience Design* befasst sich deshalb damit, diese subjektive Wahrnehmung eines Benutzers so positiv als möglich ausfallen zu lassen und erfolgt vor dem User Interface Design. Durch das User Experience Design wird bestimmt, wie die Benutzeroberfläche funktioniert. Dazu zählt zum Beispiel, wie die Navigation innerhalb einer Anwendung erfolgt und in welchem Abschnitt einer Anwendung welche Funktionen zugänglich sein sollen. Dabei werden meist grobe Skizzen mit den wichtigsten Kernelementen – sogenannte *Wireframes* – erstellt, die als Blaupause für das spätere User Interface Design dienen sollen. [Hendrick, 2022, S.7, S.8, S.21]

Durch eine Benutzeroberfläche soll letzten Endes die technische Ebene einer Anwendung, also wie genau deren Funktionen implementiert wurden und funktionieren,

vor dem Benutzer weitestgehend verborgen bleiben. Alan Cooper nennt in seinem Buch *About Faces* die technische Ebene einer Anwendung *Implementation Model*. Im Gegensatz zu diesem steht das *Mental Model*. Dieses beschreibt die Denkweise des Benutzers, welcher die technischen Funktionen zum besseren Verständnis meist um einiges abstrahiert und vereinfacht, um eine Interaktion durchführen zu können. Cooper nennt in seinem Buch dabei das Beispiel eines Anrufs über das Smartphone. Der Benutzer stellt sich vor, dass eine unsichtbare Verbindung, ähnlich einer Leitung, zwischen ihm und dem Anrufer liegt und es ihnen so erlaubt, miteinander zu kommunizieren. Wie genau aber diese Verbindung auf technischer Ebene zustande kommt, muss er in diesem Moment nicht wissen, um einen Anruf durchführen zu können. [Cooper et al., 2014, S.16-21]
Das Implementation Model muss sich somit für intuitive Benutzerinteraktionen weitestgehend dem Mental Model des Benutzers annähern. Sei es durch die Aufteilung einer technischen Funktion auf mehrere Schritte, die dem Benutzer dargestellt werden oder durch die Möglichkeit, nötige Dateneingaben nicht nur über Eingabefelder zu tätigen, sondern zusätzliche Auswahllisten oder Dropdown-Menüs anzubieten. Diese Annäherungs- und Abstraktionsschritte zu dem Mental Model entsprechen verschiedenen *Representative Models*. Je mehr sich dabei das Representative Model dem Mental Model gleicht, desto einfacher ist es für den Benutzer, die Anwendung zu nutzen. Eine visuelle Darstellung des Representative Models zeigt folgende Abbildung 4.2.

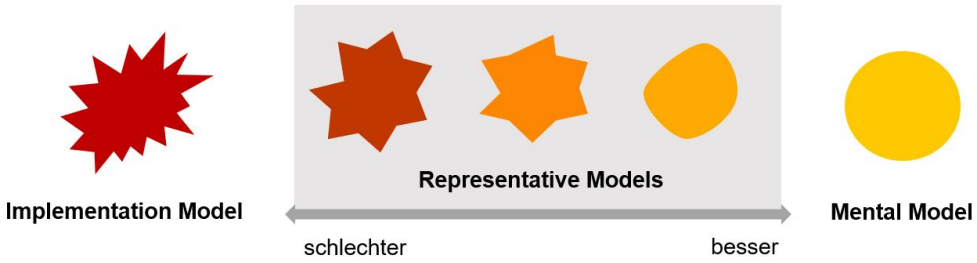

Abbildung 4.2: Zusammenspiel der drei Modelle. Abbildung in Anlehnung an [Cooper et al., 2014]

Ist es für einen Benutzer frustrierend eine bestimmte Anwendung zu nutzen, liegt dies meist daran, dass in dieser das Mental Model vernachlässigt wurde. [Cooper et al., 2014, S.16-21]

Ein weiterer wichtiger Punkt in der Gestaltung von Benutzeroberflächen oder Webseiten ist es sicherzustellen, dass diese barrierefrei genutzt werden können. Hierbei geht es darum, dass alle Benutzer, unabhängig von physischen, psychischen oder technologischen Einschränkungen eine Anwendung nutzen können. In den *Web Content Accessibility Guidelines 2.0* der *Web Accessibility Initiative* existieren diesbezügliche Empfehlungen mit den Konformitätsstufen A, AA und AAA. Die Stufe A stellt dabei das Minimum an notwendigen Anforderungen für eine barrierefreie Nutzung einer Anwendung dar und ist deshalb nur für einen kleinen Anteil der eingeschränkten Benutzer ausgelegt. Die Konformitätsstufen AAA stellt im Gegensatz dazu sicher,

dass alle Benutzer mit einer möglichen Einschränkung die Anwendung in vollem Umfang nutzen können. Die Stufe AA entspricht einem Mittelmaß zwischen den Konformitätsstufen A und AAA und wird von den meisten Unternehmen angestrebt. [AllAccessible, 2022]

4.2.2 Design-Prinzipien für Benutzeroberflächen

Die Beurteilung, ob eine Benutzeroberfläche übersichtlich und einfach zu nutzen ist, hängt letzten Endes von der subjektiven Meinung des jeweiligen Benutzers ab. Dennoch existieren verschiedene Prinzipien, die diese Meinungsbildung positiv beeinflussen sollen. Diese werden im Folgenden erörtert.

Simplicity

Die Nutzung einer Anwendung soll für den Benutzer so einfach und übersichtlich wie möglich gestaltet werden. Denn je mehr Auswahlmöglichkeiten diesem zeitgleich zur Verfügung stehen, desto länger braucht dieser, um eine Entscheidung für eine Interaktion zu treffen. Dieses Prinzip wurde von William Edmund Hick und Ray Hyman im Jahr 1952 in *Hick's Law* (auch Hick-Hyman Law) definiert. Auf diesem basiert auch das Design-Prinzip *K.I.S.S* (Abk. für Keep It Short and Simple), welches besagt, dass Einfachheit (Simplicity) der beste Weg ist, damit ein System am besten genutzt werden kann. [Soegaard, 2021]
Durch einfach gehaltene Benutzeroberflächen, die sich nur auf die wesentlichen Funktionen fokussieren, ist es zudem möglich, den Einarbeitungsaufwand gering zu halten, den ein Benutzer aufbringen muss, um eine Anwendung zu verstehen. Eine zu lange Einarbeitungszeit kann zu Frust führen und demotivierend wirken, die Anwendung weiter zu nutzen. Das Simplicity-Prinzip soll dies verhindern. [Panzarella, 2022, S.88] [Hendrick, 2022, S.26]

Progressive Disclosure

Das Prinzip *Progressive Disclosure* greift den Ansatz des Simplicity-Prinzips auf und dient dazu, die Komplexität innerhalb einer Benutzeroberfläche zu reduzieren. Dabei wird versucht nur die nötigsten und wichtigsten Informationen und Funktionen in der Benutzeroberfläche darzustellen, während unwichtigere Elemente ausgelagert und über ein Menü oder eine zusätzliche Interaktionsfläche aufrufbar sind. Auch die Aufteilung einer komplexeren Funktionalität auf mehrere Schritte beziehungsweise auf mehrere aneinandergereihte Anwendungsabschnitte, wie zum Beispiel bei einem Registrierungsprozess, gehört zu diesem Prinzip.
Informationen und Funktionen können dabei in die folgenden drei Kategorien aufgeteilt werden.
Essenzielle Informationen sind Informationen, ohne diese der Benutzer die Anwendung nicht weiter nutzen kann. Deshalb sollten diese immer sichtbar und einsehbar sein.
Wichtige Informationen beschreiben Informationen, mit denen der Benutzer abwägen kann, ob er eine Aktion ausführen soll oder nicht. Im Gegensatz zu den essenziellen

Informationen, kann ein Benutzer eine Anwendung weiterhin nutzen, selbst wenn wichtige Informationen fehlen. Dazu zählen beispielsweise Informationen zu möglichen Vorteilen, die ein Benutzer erhält, falls dieser sich in einer Anwendung registriert. **Unwichtige Informationen** sind weder notwendig zur Nutzung der Anwendung, noch regen sie den Benutzer dazu an, eine Aktion auszuführen. Deshalb sind diese durch beispielsweise Menüs, Dialogboxen, Drop-Down-Menüs oder auf einen zweiten Abschnitt innerhalb der Anwendung, der über eine zusätzliche Interaktionsfläche erreichbar ist, auszulagern, um die Komplexität der Benutzeroberfläche zu reduzieren. [Panzarella, 2022, S.93]

Alignment

Das *Alignment* beschreibt die Platzierung der sichtbaren Elemente auf der Darstellungsfläche und vermittelt den Bezug der Elemente untereinander. Durch das Einhalten des Alignment-Prinzips wirken Benutzeroberflächen geordneter und leserlicher. So ist linksbündiger Text beispielsweise besser zu lesen, als zentrierter Text, da bei zentrierten Texten die Satzanfänge jeweils an unterschiedlichen Positionen beginnen und so das Auge des Lesers hin und her springen muss.
Um Elemente entsprechend ausrichten zu können, empfiehlt es sich, Bezugslinien zu nutzen. Bei diesen wird von einer beliebigen Kante oder dem Rand eines Objektes eine Linie entsprechend nach oben, unten, links oder rechts gezogen. An diese Linie können dann weitere Elemente angeordnet werden. [Panzarella, 2022, S.110]
In der folgenden Abbildung 4.3 sind beispielhaft lila eingezeichnete Bezugslinien auf der Webseite www.prosieben.de zu sehen. Es ist zu erkennen, dass durch die Nutzung dieser die visuellen Elemente passend angeordnet sind und die Webseite übersichtlich und geordnet wirkt.

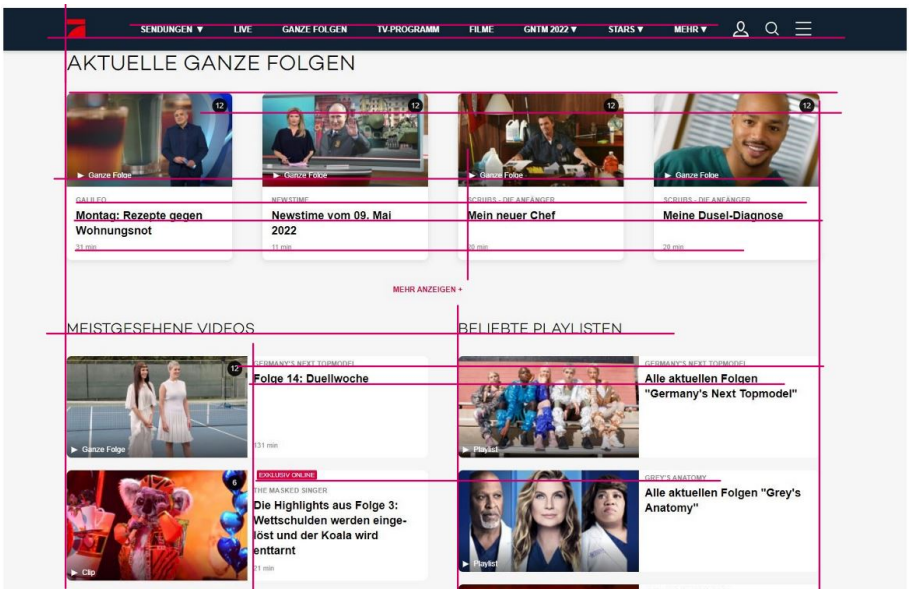

Abbildung 4.3: Beispiel: Eingezeichnete Bezugslinien auf der Webseite www.prosieben.de, Quelle: Eigene Darstellung

Auch die Nutzung eines Gitters, das über die Gestaltungsfläche der Benutzeroberfläche gelegt wird, kann dabei hilfreich sein, um Objekte korrekt anzuordnen. Auf dem Desktop empfiehlt es sich dabei, ein Gitter mit zwölf Spalten, auf dem Tablet mit acht Spalten und für mobile Benutzeroberflächen mit vier Spalten zu nutzen. [Hendrick, 2022, S.24]
Worauf bei der Umsetzung von mobilen Benutzeroberflächen geachtet werden muss, wird im Laufe dieses Kapitels noch näher erläutert.

Proximity

Proximity definiert die Distanz einzelner Elemente auf einer Darstellungsfläche zueinander. Elemente, die näher zusammenliegen, werden als eine Gruppe angesehen. Dabei ist es egal, welche Farbe oder Form die Elemente einer solchen Gruppe haben. [Panzarella, 2022, S.141] Nimmt man an dieser Stelle ebenfalls als Beispiel die Abbildung 4.3, werden sowohl die Blockelemente auf der linken Seite unter der Überschrift *Meistgesehene Videos* als eine Gruppe angesehen, als auch die Blockelemente auf der rechten Seite unter der Überschrift *Beliebte Playlisten*. Diese Gruppierung erfolgt dabei nur auf Grund der Tatsache, dass die jeweiligen Elemente einer Gruppe näher zusammenliegen.
Der Vorteil durch dieses Gestaltungsprinzip liegt somit darin, Elemente nach inhaltlichen Gemeinsamkeiten gruppieren und so Informationsblöcke bilden zu können. Ist ein Element eines solchen Blockes oder eine Überschrift, die diesen Block beschreibt, für den Benutzer inhaltlich uninteressant, kann er somit den gesamten Block überspringen.

Farbe

Farben können bei der Gestaltung von Benutzeroberflächen auf verschiedene Weise zum Einsatz kommen. Diese können Elemente und Funktionen hervorheben, um deren Wichtigkeit für die Anwendung auszudrücken. Des Weiteren vermitteln Farben durch biologische und kulturelle Begebenheiten und Entwicklungen verschiedene Nachrichten oder auch ein bestimmtes Gefühl an den Benutzer. [Lundberg, 2020]
So kommt die Farbe Rot häufig für Warnhinweise zum Einsatz. Tritt beispielsweise bei der Nutzung einer Anwendung ein Fehler auf oder führt der Benutzer eine Funktion falsch aus, wird dies meist mit Hilfe der Farbe Rot signalisiert. Dabei drückt rot auch eine gewisse Dringlichkeit aus, was den Benutzer dazu bringen soll, auf den Warnhinweis zu reagieren. Im Vergleich dazu signalisiert die Farbe Grün Gelassenheit. Diese kommt meist zum Einsatz, wenn eine Aktion eines Benutzers erfolgreich ausgeführt wurde, wodurch der Benutzer weiß, dass er alles richtig gemacht hat.
Diese Farbassoziationen sollten Entwickler bei der Gestaltung von Benutzeroberflächen beachten. Im Anhang B ist eine Tabelle B.1 mit einigen Farbassoziationen aufgeführt. Auch Beispiele, für welche Themengebiete oder Funktionen bestimmte Farben meist zum Einsatz kommen, werden dabei aufgegriffen.

Neben den eher positiven Assoziationen können Farben auch negative Gefühle hervorrufen. Negative Assoziationen treten häufig durch einen zu dominanten Einsatz bestimmter Farben auf, weshalb ein sparsamer Einsatz von diesen von Nöten ist.

Zu viel schwarz kann beispielsweise düster wirken, während zu viel gelb oder rot oft unangenehm anzusehen ist. Die neutrale Farbe Weiß eignet sich hingegen gut für Hintergründe und kann demnach großflächig zum Einsatz kommen.
Des Weiteren sollten nicht zu viele verschiedene Farben genutzt werden, drei bis fünf sollten hierbei ausreichen. Bei drei gewählten Farben sollten diese in folgendem Verhältnis in der Benutzeroberfläche zur Verwendung kommen.
60 % der Anwendung sollte die gewählte Hauptfarbe oder Primärfarbe einnehmen, 30 % der Anwendung sollte die Sekundärfarbe abdecken. Die dritte Farbe soll mit zehn-prozentigem Anteil als Akzentfarbe dienen und die wichtigsten Elemente hervorheben. Die Akzentfarbe sollte dabei am auffälligsten sein, wie beispielsweise die Farbe Rot oder die Farbe Orange. [Mulligan, 2021, S.124]

Die Bedeutung der jeweiligen Farbe kann sich je nach Kulturraum, in der die Anwendung eingesetzt wird, zudem ändern. [Mulligan, 2021, S.125] In China beispielweise wird rot mit Glück verbunden, während in Südafrika die Farbe Rot für Trauer steht. [Lundberg, 2020]

Neben der Verwendung bestimmter Farben auf Grund ihrer Assoziationen, werden Farben auch zur Signalisierung von möglichen Interaktionen eingesetzt. Da meist für Hintergründe neutrale Farben wie weiß ihren Einsatz finden, können kräftigere Farben eine mögliche Interaktion signalisieren.
Des Weiteren können Farben mit bestimmten Funktionalitäten einer Anwendung verknüpft werden. Die Farbe Grün signalisiert häufig in sozialen Medien, dass ein jeweiliger Benutzer gerade die Anwendung nutzt. Panzarella nennt in seinem Buch *UI + UX: web design simply explained* als Beispiel für die Farbe Blau den Messaging-Dienst Whatsapp. Zwei blaue Haken zeigen dabei dem Benutzer an, dass seine Nachricht vom Empfänger gelesen wurde. [Panzarella, 2022, S.148]

Als letzter Punkt bei dem Gestaltungsprinzip Farbe ist der Ansatz der Barrierefreiheit zu berücksichtigen. Da etwa 10 % der Weltbevölkerung Farben nicht richtig erkennen können, sollten zusätzliche Texte oder Icons zum Einsatz kommen, um farblich gekennzeichnete Funktionen zu beschreiben und Missverständnisse zu vermeiden. [Panzarella, 2022, S.151] [W3C Web Accessibility Initiative, 2022]

Kontrast

Der Kontrast bezeichnet den visuellen Unterschied zwischen Farben und Objekten, meist zwischen hell und dunkel. Wie stark ein Kontrast ist, hängt dabei stark von den verwendeten Farbtönen und vor allem den Sättigungen und den Helligkeiten der gewählten Farben ab. Die Sättigung und die Helligkeit nehmen im *HSB*- beziehungsweise *HSV*-Farbraum (Hue, Saturation, Brightness/Value) Werte von 0 % bis 100 % (oder Werte zwischen 0 und 1) an.
Die Sättigung (Saturation) bestimmt dabei die Intensität der Farbe. Eine niedrige Sättigung bedeutet somit, dass eine Farbe blasser ist.
Die Helligkeit (Brightness/Value) dunkelt bei einem niedrigen Wert die Farbe ab. Ein höherer Helligkeitswert hellt die Farbe entsprechend auf. Hue steht für den Farbton

und nimmt je nach dessen Wahl einen Wert von 0 bis 360 Grad an, da die Farbtöne im HSB/HSV-Farbraum auf einem Farbkreis liegen.
Die Folgende Abbildung 4.4 soll zum besseren Verständnis des HSV-Farbraums dienen.

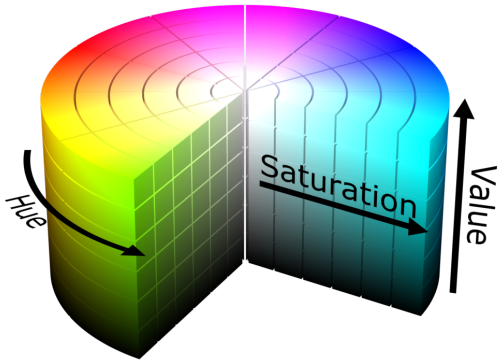

Abbildung 4.4: HSV-Farbraum [Faizunnabi, 2020]

Der Kontrast zwischen zwei Farben wird durch das Kontrastverhältnis definiert, welches die Kontraststärke zwischen der Vordergrundfarbe und Hintergrundfarbe berechnet. Für die Berechnung des Kontrastverhältnisses existieren verschiedene online-Werkzeuge, wie zum Beispiel der *Kontrast Checker* von der Webseite WebAim oder das *Barrierefreiheits-Tool* von Adobe Color. Hierbei gibt man die Farbe für den Vordergrund und den Hintergrund an. Liegt beispielsweise auf einer Webseite schwarzer Text auf weißem Hintergrund, muss in dem online-Tool für den Vordergrund die Farbe Schwarz eingespeist werden und für den Hintergrund die Farbe Weiß. Das Ergebnis des Kontrastverhältnisses beträgt dabei 21:1, was dem höchstmöglichen Wert eines Kontrastverhältnisses entspricht.

Grundsätzlich sind starke Kontrastverhältnisse besser und schneller wahrnehmbar als schwächere, wodurch Entwickler wichtigere Elemente hervorheben können. Des Weiteren erleichtern stärkere Kontraste das Lesen, weshalb meistens schwarzer Text auf weißem Hintergrund abgebildet wird. [Panzarella, 2022, S.154]
Da zudem manche Menschen Kontrastverhältnisse unterschiedlich wahrnehmen, sollte für die barrierefreie Nutzung einer Anwendung ein gewisser Gradient für das Kontrastverhältnis, vor allem für essenzielle Informationen, nicht unterschritten werden. [Hendrick, 2022, S.25]
In den Web Content Accessibility Guidelines 2.0 soll für die Erfüllung der Konformitätsstufe AA mindestens ein Kontrastverhältnis von 4,5:1 (Maximum ist 21:1) in der Anwendung vorliegen, was bedeutet, dass im Verhältnis die hellere Farbe 4,5-mal mehr Leuchtkraft besitzen muss, als die dunklere Farbe. Die Konformitätsstufe AAA benötigt mindestens ein Kontrastverhältnis von 7:1. [W3C Web Accessibility Initiative, 2018]

Visuelle Hierarchie

Das Prinzip *Visuelle Hierarchie* beschreibt den hierarchischen Aufbau einer Benutzeroberfläche nach der Wichtigkeit der jeweils dargestellten Elemente. Nach dieser Hierarchie soll der Benutzer dann durch die verschiedenen Funktionen geleitet werden, was ihm auch helfen soll, die Anwendung besser zu verstehen. [Hendrick, 2022, S.27]
An dieser Stelle kann das Gestaltungsprinzip auch in Zusammenhang mit Hick's Law betrachtet werden. Häufig ist die Auswahl und Komplexität an Funktionalitäten in einer Anwendung ab einem gewissen Punkt nicht weiter zu vereinfachen (Prinzip Simplicity). [Soegaard, 2021]
Um zu verhindern, dass die Auswahlmöglichkeiten einer komplexeren Benutzeroberfläche den Benutzer überfordern, sollen deshalb zusätzlich die wichtigsten Funktionen hervorgehoben werden. Ist ein Element größer oder hat es eine auffälligere Farbe, wirkt es automatisch dominanter und fällt als erstes dem Benutzer ins Auge. Weniger dominante Elemente bewirken das Gegenteil. [Panzarella, 2022, S.103] Auch stärkere Kontrastverhältnisse können dabei helfen, eine visuelle Hierarchie innerhalb einer Anwendung zu bilden.

Mit größeren Elementen können zudem Benutzer einfacher interagieren. Paul Fitts beschreibt in seinem Gesetz *Fitts's Law*, wie schnell und genau eine Person durch eine physikalische Bewegung eine bestimmte Fläche oder ein bestimmtes Element erreichen und antippen kann, abhängig davon, wie groß diese Fläche ist und wie weit diese von der Person entfernt liegt. Dabei sagt dieses Gesetz aus, dass eine Fläche umso schwerer zu erreichen ist, je weiter weg sich diese von der Person befindet und je kleiner die Fläche dabei ist. Größere Flächen mit derselben Entfernung zu einer Person, wie eine kleine Fläche, sind somit einfacher zu erreichen.
Überträgt man dies auf die digitale Welt und die Nutzung von Benutzeroberflächen, ist ähnliches zu beobachten. Benutzer benötigen mehr Zeit, um mit kleineren Flächen zu interagieren, die weiter weg von der aktuellen Position ihres Mauszeigers liegen, als mit größeren Flächen mit derselben Entfernung. Entsprechend sind vor allem die wichtigsten Funktionalitäten mit größeren Buttons und Klickflächen zu versehen, um die Interaktion zu erleichtern, die Aufmerksamkeit des Benutzers zu erregen und somit eine visuelle Hierarchie zu schaffen. [Goktürk, 2015]

Typographie

Die *Typographie* ist die Lehre der Schriftgestaltung. Diese kann dazu eingesetzt werden, die Aufmerksamkeit des Benutzers zu lenken und eine visuelle Hierarchie in einem Darstellungsbereich zu schaffen. Zu der Gestaltung von Schriften gehören beispielweise Schriftgrößen, Zeilenabstände, Kontrast zum Hintergrund und welche Schriftfamilien zum Einsatz kommen. Eine *Schriftfamilie* ist dabei eine Gruppe ähnlicher Schriftstile/Schriftschnitte einer Schriftart, die sich dabei in Schriftbreite (schmale oder breite Buchstaben/Zeichen), Schriftlage (normale oder kursive Buchstaben/Zeichen) und Schriftstärke (dünne oder fette Buchstaben/Zeichen) unterscheiden. So gehören zu der Schriftfamilie der Schriftart *Arial* beispielsweise unter anderem die Schriftschnitte *Arial Bold* (fett), *Arial Italic* (kursiv) und *Arial Bold Italic* (fett und kursiv). [Beinert, 2022]

Ähnlich zum sparsamen Umgang mit verschiedenen Farben sollten auch nicht mehr als zwei verschiedene Schriftfamilien innerhalb eines Layouts oder einer Benutzeroberfläche zum Einsatz kommen. [Hendrick, 2022, S.25] Die gewählten Schriftfamilien sollten sich dabei ähneln, zeitgleich dennoch genügend Kontraste zueinander aufweisen, dass ein tatsächlicher Unterschied in deren Aussehen erkennbar ist. Ähneln sich die Schriftfamilien zu sehr, könnte ein Benutzer es unter Umständen als Fehler im Design empfinden. [Hendrick, 2022, S.51, S.72]
Die einzelnen Schriftstile einer Schriftfamilie selbst sind problemlos kombinierbar. Aus diesem Grund empfiehlt es sich Schriftfamilien zu wählen, die mindestens vier Schriftstile (regular, italic/kursiv, bold/fett und bold italic) besitzen, um ein gewisses Auswahlspektrum zu haben. [Hendrick, 2022, S.54]

Um Text gut leserlich für den Benutzer zu gestalten, sollte ein möglichst hohes Kontrastverhältnis zwischen der Schrift und dem Hintergrund sein. Des Weiteren empfiehlt es sich, für Texte mit wichtigen Inhalten eine Schriftgröße zwischen acht und elf Punkten zu wählen. Der Zeilenabstand sollte dabei 1,25 bis 1,5-mal größer sein als die gewählte Schriftgröße. [Hendrick, 2022, S.64]
Wie bereits im Design-Prinzip Alignment erläutert, ist linksbündiger Text außerdem leichter zu lesen, da in vielen Sprachen auch von links nach rechts gelesen wird. Rechtsbündiger Text kann vor allem bei längeren Abschnitten im Gegensatz dazu schwerfälliger zu lesen sein. Bei rechtsbündigem Text ist außerdem darauf zu achten, keine Satzzeichen am Ende einer Zeile zu setzen, damit der Text auch tatsächlich rechtsbündig ist.
Zentrierte Texte können eine gewisse Dynamik vermitteln, wobei diese ähnlich zu rechtsbündigen Texten auch schwerer zu lesen sind.
Blocksatztexte wirken modern und ordentlich, da sie zu beiden Seiten schlüssig sind. Bei der Nutzung des Blocksatzes ist darauf zu achten, dass die einzelnen Zeilen nicht zu viele und zu große Lücken aufweisen. [Stan, 2020]

Konsistenz und Vorhersagbarkeit

Das Prinzip der *Konsistenz* beschreibt die einheitliche Gestaltung einer Anwendung oder Webseite von deren Startbildschirm bis hin zur letzten Unterseite oder zum letzten Teilabschnitt. Eine Anwendung in ihrer Gestaltung konsistent zu halten ist wichtig, damit ein Benutzer Elemente in jedem neuen Abschnitt der Anwendung wiedererkennen und sich sofort wieder in der Benutzeroberfläche zurechtfinden kann. [Hendrick, 2022, S.28]
Ein Benutzer soll durch das Prinzip der Konsistenz auch zu einem gewissen Grad vorhersagen können, welche Elemente welche Aktionen nach einer Interaktion ausführen. Auch generelle, übergreifende Design-Standards sollten dabei eingehalten werden, um eine gewisse Vorhersagbarkeit zu bieten. Bestimmte Icons sind so beispielsweise bereits von Vornherein mit bestimmten Funktionalitäten verknüpft. Ein Hamburger-Icon assoziiert das Öffnen eines Menüs oder ein Zahnrad-Icon öffnet meist die Einstellungen einer Anwendung.
Des Weiteren erwarten Benutzer bereits aus Gewohnheit und Erfahrung mit anderen Anwendungen und Webseiten einige Funktionen oder Elemente an bestimmten Stellen

in der Benutzeroberfläche vorzufinden. Eine Suchleiste oder ein Logo liegt so meist im oberen Bereich des Bildschirms. [Wong, 2021]

Ein Benutzer sollte immer wissen, auf welcher Unterseite einer Webseite oder in welchem Abschnitt einer Anwendung er sich gerade befindet und was bei der Ausführung der nächsten gewählten Aktion passieren wird. Drückt er beispielsweise auf eine Interaktionsfläche, um zur vorherigen Ansicht der Anwendung zu gelangen, sollte auch tatsächlich die vorherige Ansicht dieser aufgerufen werden.
Des Weiteren sollten klickbare Flächen sichtbar und deutlich von nicht klickbaren Flächen differenzierbar sein, damit der Benutzer auch weiß, über welche Elemente in der Benutzeroberfläche er eine Aktion ausführen könnte.

Wird in einer Anwendung keine konsistente Gestaltung vorgenommen, ändern sich beispielsweise Buttons nach einer Aktion und wird ihnen in einem neuen Abschnitt der Anwendung eine andere Funktionalität zugeordnet oder ihr Aussehen verändert, muss sich der Benutzer erneut mit der Benutzeroberfläche vertraut machen, was ihm die Nutzung der Anwendung erschweren kann. [Hendrick, 2022, S.28]
Deshalb sollten nach diesem Prinzip Elemente ihre bereits zugeteilte Funktionen kontinuierlich beibehalten. Entwickler sollten grundsätzlich die gleichen Farben und Schriftfamilien durchgehend in der Anwendung einsetzen. Auch Interaktionsflächen, die ähnliche Funktionen bereitstellen, sollten sich annähernd in ihrem Aussehen gleichen.
Wird zudem eine Anwendung plattformübergreifend entwickelt, sollte dabei das genutzte Design auch plattformübergreifend zum Einsatz kommen. Betreibt beispielsweise eine Firma sowohl eine Webseite, als auch eine Anwendung, so sollten diese in ihrer Gestaltung Gemeinsamkeiten aufweisen, damit es für Kunden oder Benutzer ersichtlich ist, dass diese zusammengehören und von derselben Firma stammen. [Panzarella, 2022, S.137] Auch die Bedienung der jeweiligen plattformspezifischen Anwendung ist dadurch für den Benutzer leichter, da er das Wissen zur Nutzung plattformübergreifend einsetzen kann.

4.2.3 Benutzeroberflächen auf dem Smartphone

Heutzutage existieren viele verschiedene Endgeräte mit unterschiedlichen Bildschirmgrößen. Wie die Abbildung 4.5 zeigt, erfolgten im Jahr 2022 bisher 58 % des weltweiten Internetverkehrs über mobile Endgeräte (grüne Linie). Auf dem Desktop (blaue Linie) waren es 40 %, auf dem Tablet (lila Linie) 2 %.

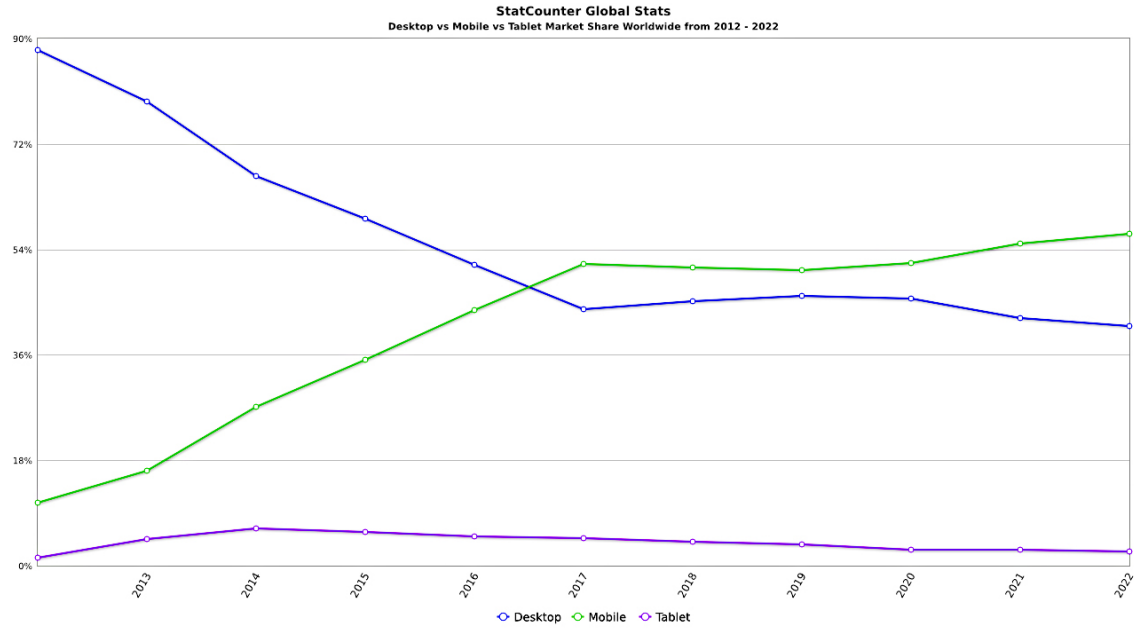

Abbildung 4.5: Weltweiter Internetverkehr über Desktop, Mobile und Tablet über die vergangenen 10 Jahre [StatCounter, 2022]

Vor allem für Webseiten und Benutzeroberflächen von Anwendungen, die von verschiedenen Plattformen aus aufrufbar sind, ist es somit immer wichtiger, diese responsiv zu gestalten. Responsiv bedeutet dabei, dass sich ihre Inhalte automatisch an die jeweilige Bildschirmgröße eines Endgerätes anpassen. Ansonsten würden Bildschirmränder wichtige Informationen und Elemente abschneiden, wodurch es Benutzern schwerer fällt, die jeweilige Anwendung zu nutzen.
Im Vergleich zur Gestaltung von Benutzeroberflächen für Desktop-Applikationen, muss somit folgenden Punkten bei mobilen Anwendungen für das Smartphone Beachtung geschenkt werden.

Mobile First

Smartphones haben einen kleineren Bildschirm und somit eine viel kleinere Darstellungsfläche zur Verfügung, als ein Computer. Während bei einer Desktop-Anwendung die Möglichkeit besteht, den Inhalt auf mehrere Spalten horizontal aufzuteilen und gegebenenfalls dabei größere Bilder mit aufzuführen, bieten Smartphones vor allem in der Horizontalen kaum Platz.
Elemente müssen deshalb hier neu angeordnet werden. Nehmen manche Bestandteile der Benutzeroberfläche zu viel Platz ein, ist es sinnvoll, diese in der mobilen Version

der Benutzeroberfläche herauszunehmen.
Den Design-Prinzipien Simplicity und Progressive Disclosure sollte hierbei noch mehr Beachtung geschenkt werden.

Wird eine Webseite oder plattformübergreifende Anwendung zuerst für den Desktop entwickelt, müssten Entwickler die Benutzeroberfläche anschließend für die mobilen Plattformen herunterbrechen und Inhalte und Elemente entsprechend kürzen.
Um diese Entscheidung, welche Elemente aus der Benutzeroberfläche zu nehmen sind zu umgehen, gibt es den Ansatz des *Mobile First* (siehe Abbildung 4.6). Dabei wird zuerst eine Anwendung oder Webseite für das Smartphone entwickelt. Dadurch ist es von Anfang an erforderlich, nur die wesentlichen Elemente für die Benutzeroberfläche zu nutzen und diese auf der begrenzten Darstellungsfläche passend anzuordnen.

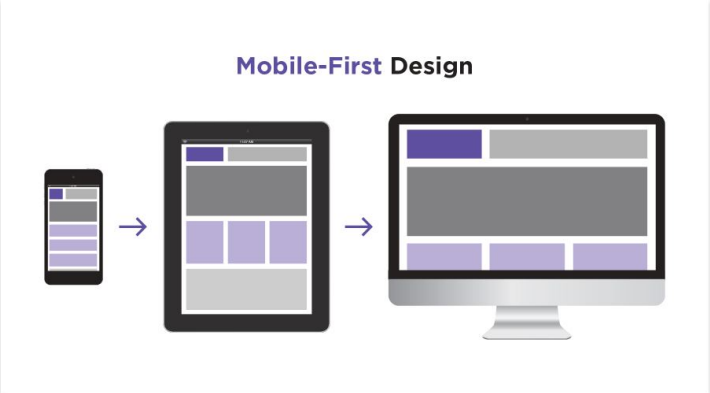

Abbildung 4.6: Mobile First Design-Ansatz [Williamson, 2020]

Aus diesem Grund wird Mobile First oft mit dem Ansatz *Content First* assoziiert. Bei diesem ist der Inhalt wichtiger als die Gestaltung der Benutzeroberfläche selbst und ist auf das Wesentliche zu beschränken. Ein Benutzer soll dabei schnell den Kerninhalt einer Anwendung oder Webseite erkennen.

Außerdem ist es wichtig zu beachten, dass ein Benutzer auf dem Smartphone auf den ersten Blick weniger Elemente und Text sehen kann, als auf dem Desktop. Dadurch muss er auf dem Smartphone mehr Aufwand durch beispielsweise Scrollen betreiben, um an dieselbe Informationsmenge zu gelangen, die er auf dem Desktop auf einen Blick sehen könnte. Essenzielle Informationen sollten deshalb vor allem bei mobilen Webseiten und Anwendungen gleich beim Öffnen dieser dargestellt werden. Dieser Bereich wird *Above the fold* genannt und beschreibt den Ausschnitt einer Webseite, den ein Benutzer ohne zusätzliche Interaktionen direkt einsehen kann. Alle Elemente, die durch weiteres Scrollen auftauchen, befinden sich unterhalb dieses Bereiches, der als *Below the fold* bezeichnet wird. [Panzarella, 2021, S.40]
Ist die mobile Version entwickelt, erfolgt anschließend die Anpassung der Benutzeroberfläche an größere Bildschirme. Da nun bereits die wichtigsten Elemente durch die mobile Version in der Benutzeroberfläche enthalten sind, ist es möglich, den zusätzlichen verfügbaren Platz der größeren Bildschirme mit weiteren Elementen zu befüllen.

Hier ist unbedingt zu beachten ist, dass die mobile Version der Benutzeroberfläche nicht eins zu eins auf die Desktop-Version übertragen werden darf, wie auch eine Benutzeroberfläche vom Desktop nicht für Smartphones geeignet ist.

Interaktion

Benutzern stehen bei der Nutzung von Anwendungen auf dem Desktop zusätzliche Hardware-Komponenten als Eingabegeräte zur Verfügung, welche Interaktionen erleichtern. So sind Texteingaben über die Tastatur oder gezielte Interaktionen mit der Maus schnell und einfach durchführbar. Im Vergleich dazu finden alle Interaktionen auf einem Smartphone auf der Darstellungsfläche selbst statt. So tippt ein Benutzer Text direkt über die virtuelle Tastatur auf dem Bildschirm seines Smartphones in ein vorgesehenes Textfeld. Die Tasten dieser virtuellen Tastatur sind dabei im Vergleich zur Tastatur des Computers um einiges kleiner, was das Risiko sich zu vertippen erhöht und die Eingabe erschwert. Grundsätzlich ist das Tippen mit dem Finger auf den Touchscreen im Vergleich zur Maus weniger präzise, da zudem der Finger das Sichtfeld zwischen Benutzer und Touchscreen zeitweise verdeckt. Dadurch kann sich eine scheinbar einfache Interaktion auf dem Desktop als eine komplexere Funktion auf dem Smartphone herausstellen. Aus diesem Grund sollten vor allem auf mobilen Endgeräten umfangreichere Funktionen und Aktionen, wie das Ausfüllen eines Formulars, in mehrere Schritte unterteilt werden (Design-Prinzip Progressive Disclosure). [Panzarella, 2021, S.34]
Des Weiteren sollten Interaktionselemente, wie Buttons, möglichst vergrößert werden. Auch den Interaktionsbereich über das visuelle Element selbst hinaus zu vergrößern, wäre eine weitere Möglichkeit.
Es sollte zudem genügend Platz zwischen den einzelnen Interaktionselementen vorhanden sein, um zu verhindern, dass zwei Interaktionsbereiche zu nahe aneinander liegen.
Durch die schwerere Interaktion auf Smartphones ist außerdem die virtuelle Tastatur generell nur in geringem Maße zu verwenden. Statt beispielweise zur Filterung von Listeneinträgen Texteingaben zu verwenden, wären Radiobuttons oder Checkboxen eine bessere Alternative. Bei der Angabe eines Datums kann statt einem Textfeld ein interaktiver Kalender genutzt werden.
Kommt die virtuelle Tastatur dennoch zum Einsatz, ist die Eingabe so leicht als möglich zu gestalten. Hierzu gibt es beispielsweise die Autokorrektur. Auch Vorschläge in einem Dropdown-Menü in einer Suchleiste oder die Reduzierung der nötigen Eingabefelder in einem Dialogfeld können dabei hilfreich sein. Während auf der Desktop-Version für eine Registrierung beispielsweise die E-Mail-Adresse, ein Passwort, Vor- und Nachname und eventuell weitere Datenangaben erforderlich sind, um die Anwendung nutzen zu können, sollte man sich auf der mobilen Version nur auf die E-Mail-Adresse und das Passwort beschränken. Sind dennoch mehrere Eingabefelder erforderlich, sollte, wie bereits erwähnt, eine Aufteilung der Interaktionen in mehrere Schritte erfolgen. [Panzarella, 2021, S.65] [Jensen, 2020]

Touch Hierarchie

Während am Desktop Elemente einer Benutzeroberfläche durch die Maus relativ schnell und einfach erreichbar sind, hängt die Bedienbarkeit einer mobilen Anwendung stark von der Anatomie der menschlichen Hand ab und wie ein Benutzer das Smartphone dabei hält. Eine verbreitete Auffassung von Designern ist, dass Benutzer durch die typische und natürliche Bewegung des Daumens bei der Nutzung eines Smartphones hauptsächlich den unteren Bereich des Touchscreens zur Interaktion nutzen und das Smartphone dabei auch nur mit einer Hand halten. [Hoober, 2017]
Die Abbildung 4.7 aus Steven Hoobers Ausarbeitung *Design for Fingers, Touch, and People* zeigt, wie angenehm oder leicht es für einen Benutzer ist, einen bestimmten Bereich des Bildschirms zu erreichen, wenn dieser mit einer Hand das Smartphone hält und bedient.

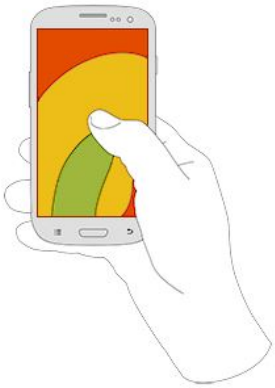

Abbildung 4.7: Nutzung des Bildschirms auf Grundlage der natürlichen Bewegung des Daumens [Hoober, 2017]

Im grünen Bereich liegen dabei Elemente, die mit der natürlichen Bewegung des Daumens von unten nach oben erreichbar sind. Der gelbe Bereich markiert Positionen für Elemente, bei diesen der Benutzer für eine Interaktion seinen Daumen strecken oder einknicken müsste, dies aber noch in einem angenehmen Rahmen. Rot stellt den unangenehmsten Interaktionsbereich auf dem Bildschirm dar, bei diesem ein Benutzer Elemente mit dem Daumen nicht erreichen könnte, ohne vorher zusätzlich die Handposition zu verändern. Deshalb sollte eine Platzierung wichtiger interaktiver Elemente im einfach erreichbaren, unteren Bereich des Touchscreens erfolgen, wohingegen informative Inhalte im oberen Bereich des Bildschirms anzuordnen sind. [Hoober, 2017]

Hoober fand im Gegensatz dazu in einer neueren Feldforschung (2017) heraus, dass die meisten Interaktionen von Benutzern tatsächlich im mittleren Bereich des Touchscreens erfolgen, unabhängig davon, wie diese das Smartphone halten. Des Weiteren scannen Benutzer nicht wie am Desktop mit ihrem Blick Inhalte von oben links nach unten rechts, sondern sehen sich Inhalte lieber mittig auf dem Smartphone an. Benutzer scrollen häufig auch Inhalte zur Mitte des Bildschirms, da diese dort besser einzusehen

sind. Auch Touch-Interaktionen sind im Zentrum des Touchscreens besser und präziser vom Benutzer durchführbar (vgl. Abbildung 4.8).

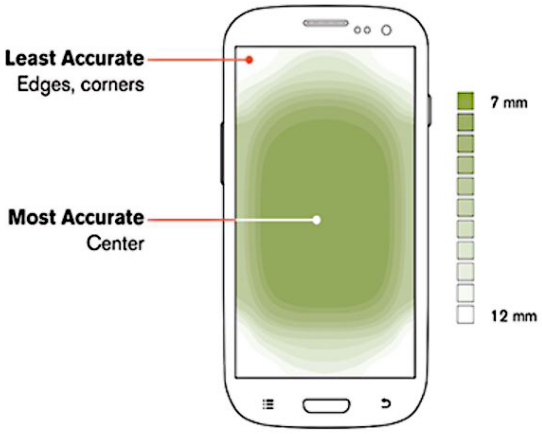

Abbildung 4.8: Interaktionspräzision am Touchscreen nach Hoober [Hoober, 2017]

Laut Hoober sind interaktive Elemente deshalb in der Mitte des Touchscreens kleiner zu gestalten, während Elemente nahe den Rändern, vor allem in den Ecken des Bildschirms, größer darzustellen sind. Auch die wichtigen Kernelemente einer Anwendung sollten Entwickler in der Mitte des Bildschirms platzieren. [Hoober, 2017] Sekundäre Elemente wie *Tabs*, die zur Navigation innerhalb der Anwendung dienen oder Suchfelder, um Inhalte zu suchen, können an den oberen und unteren Rand geschoben werden. Andere, unwichtigere Funktionen sind über ein Menü zugänglich zu machen, das in den Ecken des Touchscreens vorzufinden ist. [Hoober, 2017]

Navigation im Vergleich zum Desktop

Auf dem Desktop aufgerufene Anwendungen und Webseiten haben genügend Darstellungsfläche zur Verfügung, um eine übersichtliche Navigation anzubieten. Die Menüs können sich an den Bildschirmrand schmiegen, ohne dabei wichtige Inhalte einer Anwendung zu verdecken. Meist werden Menüs dabei im oberen Bereich des Bildschirms horizontal platziert.
Drop-Down-Menüs oder *Mega-Drop-Down-Menüs* können dem Benutzer zeitgleich mehrere Kategorien samt Unterkategorien anbieten. Mega-Drop-Down-Menüs bieten sich vor allem an, wenn es viele wichtige Hauptkategorien gibt, auf die ein Benutzer navigieren soll. [Costa, 2020]
Mobile Plattformen haben im Vergleich kaum Platz, um verschachtelte Menüs mit mehreren Unterkategorien anzubieten. Auch horizontal ausgerichtete, umfangreichere Menüs sind meist schwer darzustellen. Für mobile Anwendungen auf dem Smartphone sollten Menüs deshalb vor allem vertikal listenartig ausgerichtet werden, die beispielsweise durch ein Hamburger-Symbol ein- und ausblendbar sind. Anstatt mehrere Unterkategorien zur Verfügung zu stellen, sollten Entwickler sich auf wesentliche Hauptkategorien beschränken und eine flachere Navigationsstruktur nutzen. Eine flachere Navigationsstruktur soll dabei verhindern, dass der Benutzer über mehrere

Interaktionen zu tief in die Anwendung navigieren muss, um bestimmte Informationen zu finden.
Für flache Navigationsstrukturen in mobilen Anwendungen kommen häufig *Tab Bars* zum Einsatz, da Benutzer durch einfache Wisch- oder Tipp-Interaktionen zwischen verschiedenen Hauptkategorien wechseln können. [Costa, 2021]
Navigationsmenüs auf mobilen Anwendungen sollten außerdem mit größeren Klickflächen und Texten ausgestattet werden. Generell sollte man Texte auf mobilen Plattformen größer darstellen, als auf dem Desktop. Empfohlen wird dabei für mobile Anwendungen eine Schriftgröße von 16pt. [Jensen, 2020]

IOS und Android Styleguide im Vergleich

Android und IOS besitzen für native Anwendungen, die auf ihren Plattformen vertrieben werden, eigene Styleguides, welche Empfehlungen zur Gestaltung von Benutzeroberflächen für Entwickler beinhalten. Dadurch sollen sich native Anwendungen in ihrem Aussehen und deren Struktur deutlicher in das Ökosystem der einzelnen Plattformen schmiegen und zudem die Benutzererfahrung verbessern.
Der aktuellste Styleguide für Android von Google heißt *Material Design 3*. Apples Styleguide für IOS wird *Human Interface Design* genannt.
Um das Gefühl einer nativen Anwendung an den Benutzer zu vermitteln und die App-Like-Charakteristik von Progressive Web Apps zu erfüllen, empfiehlt es sich, diese Styleguides in die Entwicklung von Benutzeroberflächen für Progressive Web Apps mit einzubeziehen. Einige dieser Empfehlungen werden im Folgenden erörtert.

Allgemein Apple setzt in seinem Human Interface Design auf Minimalismus. Typographie und gedecktere/mattere Farben stehen deutlich im Mittelpunkt. Es werden außerdem kaum Schattierungen genutzt. [Apple, 2022]
Das Material Design von Google orientiert sich im Gegensatz dazu deutlicher an der realen Welt. Es werden Schatten genutzt, um eine dritte Dimension in der Benutzeroberfläche zu schaffen und mögliche Interaktionen, vor allem wichtige Interaktionen, zu signalisieren und hervorzuheben. So stehen Buttons scheinbar aus der Benutzeroberfläche heraus, wie auch in der realen Welt Buttons aus einer Oberfläche herausstehen würden. Weniger wichtige Funktionen werden aber ähnlich zu IOS ohne Schatten dargestellt. [Google, 2022]

Primäre Navigation Beim Human Interface Design dient eine Tab Bar als primäres Navigationselement. Die Abbildung 4.9 zeigt ein Beispiel dieses Elements. Die Tab Bar sitzt im unteren Bildschirmbereich und besitzt drei bis fünf Tabs, zwischen diesen der Benutzer durch Tippen oder Wischen navigieren kann. Die Labels der einzelnen Tabs sollen eine Schriftgröße von 10 pt haben. Des Weiteren wird empfohlen, innerhalb dieser Tab Bar die Hauptfunktion zentriert zu setzen, eine Suchfunktionalität sollte an das Ende der Tab Bar. Profileinstellungen oder Einstellungen zur Anwendung sollen nicht aufgeführt werden.

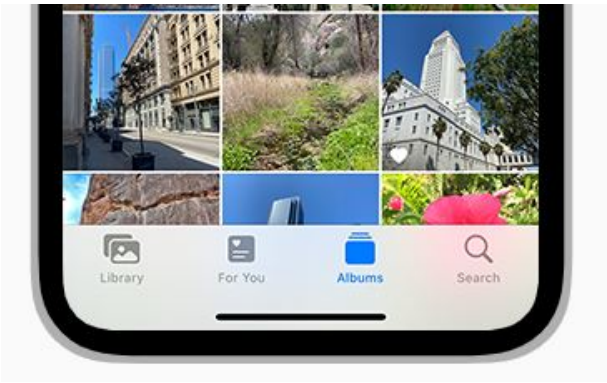

Abbildung 4.9: IOS Tab Bar [Apple, 2022]

Andere, bekannte Third-party IOS-Anwendungen empfehlen im Gegensatz dazu Profilinformationen oder Einstellungen für eine Anwendung ans Ende der Tab Bar zu setzen und die Suchfunktionalität an zweiter Stelle. [Erik D. Kennedy, 2021] [Apple, 2022] Instagram beispielsweise hat seine Navigation nach dieser Empfehlung aufgebaut.
Beim Material Design kann, ähnlich zu IOS, eine Tab Bar im unteren Bereich des Bildschirms mit drei bis fünf Elementen eingesetzt werden, um zwischen den wichtigsten Kategorien zu navigieren. Das Konzept der Tab Bar wird im Material Design aber als *Navigation Bar* bezeichnet. Die Abbildung 4.10 zeigt ein Beispiel dieser Navigation Bar. Die Navigation Bar findet bei Android eine immer häufigere Anwendung. Im Gegensatz dazu, können anstelle einer Navigation Bar auch Tabs im oberen Bereich des Bildschirms zum Einsatz kommen, was vor allem bei weniger als drei *Top-Level*-Kategorien empfohlen wird. Eine Top-Level-Kategorie ist dabei eine der wichtigsten Kategorien einer Anwendung. Dieser kann man dabei wiederum andere Kategorien unterordnen, falls vorhanden. Tabs sollen eine Schriftgröße von 14 sp (entspricht 14 pt von IOS) haben. [Google, 2022]

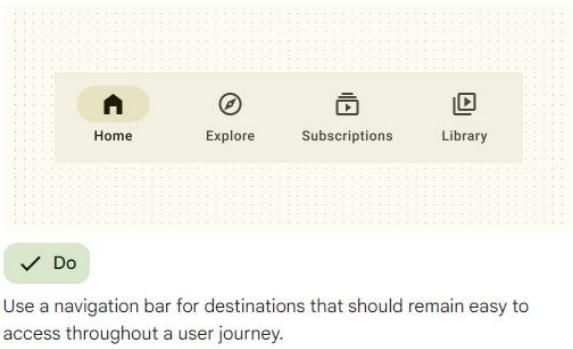

Abbildung 4.10: Android Navigation Bar [Google, 2022]

Sekundäre Navigation Sekundäre Navigationselemente werden beim Human Interface Design ausgelagert, meist in einen Tab mit dem Label *More* innerhalb der Tab Bar oder einem Hamburger-Menü. Falls mehr als fünf Top-Level-Kategorien dargestellt werden müssen, empfiehlt es sich eine *Sidebar* zu nutzen, die vertikal die verschiedenen

Kategorien aufführt und durch besagten More-Tab ein- und ausgeblendet werden kann. [Erik D. Kennedy, 2021]
Auch beim Material Design werden sekundäre Navigationselemente ausgelagert und über ein Hamburger-Menü abrufbar gemacht. Diese Auslagerung sollte zudem bei mehr als fünf Top-Level-Kategorien erfolgen. Ein *Navigation Drawer* kann hierfür genutzt werden, der wie eine Schublade bei Bedarf geöffnet oder geschlossen werden kann und ähnlich zur Sidebar vertikal angeordnet die verschiedenen Kategorien darstellt. [Google, 2022]

Rückwärts Navigation Um innerhalb einer IOS-Anwendung zurück zu navigieren, ist hierfür meist oben links im Bildschirm ein Zurück-Button in einer *Navigation Bar* vorgesehen. Diese Navigation Bar beinhaltet zudem meist den Titel der aktuellen Ansicht innerhalb der Anwendung und einen Button auf der rechten Seite, um mit dem dargestellten Inhalt interagieren zu können.
Auch das Wischen auf dem Touchscreen von links nach rechts ermöglicht es auf der IOS-Plattform in der Navigationshierarchie einen Schritt zurück zu gehen. [Apple, 2022]
Bei Android existierte bis einschließlich Android 9 eine omnipräsente schwarze Navigationsleiste am unteren Bildschirmrand und beinhaltete den Zurück-Button. Ab Android 10 existiert die omnipräsente schwarze Navigationsleiste nicht mehr. Stattdessen ist es möglich, wie bei IOS von links nach rechts zu wischen, um einen Schritt zurück zu gehen. Auch sogenannte *Top App Bars*, die im oberen Bereich des Bildschirms liegen, beinhalten einen Zurück-Button. [Erik D. Kennedy, 2021]

Primärer Aktionsbutton Der Button mit der Kernfunktion einer Anwendung wird nach dem Human Interface Design in einer Tab Bar (mittig) oder in einer Navigation Bar platziert. [Apple, 2022]
Im Material Design wird der wichtigste Button losgelöst von der restlichen Benutzeroberfläche meist unten rechts im Bildschirm dargestellt. [Google, 2022]

Buttons Buttons einer IOS-Anwendung sind flach und minimalistisch ohne Schatten gestaltet. Um bestimmte oder wichtige Buttons hervorzuheben, werden diese farbig ausgefüllt, was Entwickler aber nur vereinzelt und in geringem Maß vornehmen sollten. Die hierarchische Bedeutung und Wichtigkeit von Buttons soll somit nicht durch verschiedene Größen vermittelt werden, sondern durch verschiedene Änderungen in ihrem Design. Des Weiteren soll der Text eines Buttons einzeilig sein. Es können aber zusätzlich Untertitel unter den eigentlichen Button-Text gesetzt werden, dabei aber in einer kleineren Schriftgröße. [Apple, 2022]
Android hingegen nutzt Schatten bei wichtigeren Buttons. Die Beschriftung eines Buttons soll nicht über zwei Zeilen gehen. Die Wichtigkeit einzelner Buttons vermittelt bei Android deren Gestaltung und ihre Platzierung auf dem Bildschirm. [Google, 2022]
Sowohl bei Android, als auch bei IOS, sind Icons innerhalb eines Buttons direkt neben dem Text zu platzieren.

Touch-Bereich Das Human Interface Design empfiehlt für interaktive Elemente mindestens eine Größe von 44x44 pt (Abk. für points) zur Verfügung zu stellen.

[Apple, 2022] Im Material Design wird für interaktive Elemente eine Größe von 48x48 dp (Abk. für device-independent pixels) empfohlen. Die Einheiten pt und dp gleichen sich und sind deshalb 1:1 gegenseitig umzurechnen. [Google, 2022]

Schrift Die Standard-Schriftfamilie bei IOS-Anwendungen ist *San Francisco*. Für Basistexte, beziehungsweise Körpertexte, wird eine Schriftgröße von 17 pt empfohlen, für Überschriften 20 pt bis 34 pt. Buttons sollen eine Schriftgröße von 17 pt haben. [Apple, 2022]
Android nutzt als Schriftfamilie standardmäßig *Roboto*. Basistexte haben eine Schriftgröße von 14 sp bis 16 sp (Abk. für scaleable pixel). Texte für Buttons und Tabs sollten eine Schriftgröße von 14 sp und Überschriften eine Größe von 20 sp haben. [Google, 2022] Die Einheit sp ist dabei 1:1 in pt umzurechnen.

4.3 Gestaltung der hybriden Benutzeroberfläche

Vor der technischen Umsetzung einer Benutzeroberfläche innerhalb einer Anwendung sollte diese konzeptionell in Form von sogenannten *Wireframes* und *Mockups* dargestellt werden, um das bestmögliche Ergebnis für den Aufbau und die Gestaltung der Benutzeroberfläche zu erzielen.
Ein Wireframe entspricht dabei einem Grundgerüst, welches Aufbau und Struktur einer Benutzeroberfläche widerspiegeln soll. Auch welche Interaktionen ein Benutzer durchführen muss, um eine Aktion auszulösen, wird dabei aufgegriffen. Wichtig ist hierbei, dass die Elemente der Benutzeroberfläche vereinfacht darzustellen sind. Das Design, sowie Schriftfamilien und Farben, sind bei Wireframes nicht näher zu beachten.

Ein Mockup basiert auf bereits konzipierten Wireframes und gibt einen detaillierten Blick auf das spätere Aussehen einer Benutzeroberfläche. Dabei wird das Design in den Fokus gerückt und über das Gerüst des Wireframes aufgezogen. Es erfolgt eine farbliche Darstellung der Elemente, Kontraste werden beachtet und Schriftfamilien, sowie verschiedene Schriftgrößen kommen zum Einsatz. [Großkortenhaus, 2020]

Die folgenden Wireframes und Mockups werden mit dem Wireframe Tool von JUSTINMIND erstellt. Die verwendeten Icons stammen von der Webseite *flaticon.com*. Im Folgenden erfolgt nur die Darstellung der Wireframes und Mockups für die Hauptansicht der Anwendung.

4.3.1 Mobile Plattformen

Nach dem Ansatz des Mobile First werden zuerst die Entwürfe für die mobilen Benutzeroberflächen der Progressive Web App memorest erstellt. Für die Benutzeroberfläche auf den mobilen Plattformen erfolgt dabei jeweils die Konzeption eines Wireframes für Android und IOS, unter der Berücksichtigung des jeweiligen Styleguides.

Auf der Android-Version der Benutzeroberfläche werden die Einträge des Benutzers als Karten paarweise dargestellt und zudem nach dem jeweiligen Reisejahr gruppiert.

Karten des gleichen Jahres werden dabei näher zusammenliegend angeordnet (Prinzip Proximity). Gemäß der Prinzipien Simplicity und Progressive Disclosure stellen die Karten eine vereinfachte und reduzierte Ansicht der Reiseerinnerungseinträge dar. Mit einem Tippen auf eine Karte lässt sich die Detailansicht mit den dazugehörigen, ausgelagerten Zusatzinformationen der jeweiligen Reiseerinnerung öffnen. Ein losgelöster Button stellt die Hauptfunktion, einen neuen Eintrag zu erstellen, zur Verfügung. Die Navigation erfolgt über eine Navigation Bar im unteren Bereich des Bildschirms. Im oberen Bereich des Bildschirms liegt eine Suchleiste, um nach bestimmten Einträgen zu filtern. Generell wird nach dem Prinzip Alignment darauf geachtet, Elemente passend und geordnet in der Benutzeroberfläche zu platzieren und ihren Bezug zueinander zu verdeutlichen. Hierbei werden die Karten beispielsweise unabhängig der Gruppierung des jeweiligen Jahres unter- und nebeneinander angeordnet. Folgende Abbildung 4.11 gibt einen visuellen Überblick zur Android-Benutzeroberfläche.

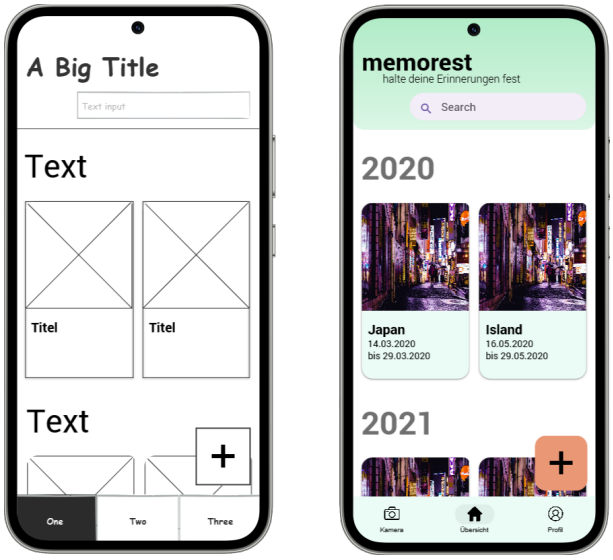

Abbildung 4.11: Wireframe (links) und Mockup (rechts) für die Benutzeroberfläche auf der Android-Plattform, Quelle: Eigene Darstellung

Das Mockup für die Android-Benutzeroberfläche zeigt die Farbgestaltung der Anwendung, sowie Beispieltexte und die Beschriftung und Icons der einzelnen Tabs der Navigation Bar. Die verwendete Schriftfamilie ist Roboto. Zur Sicherstellung der Einhaltung der AA-Konformitätsstufe der Web Content Accessibility Guidelines 2.0, erfolgt die Berechnung und Suche nach dem kleinsten Kontrastverhältnis innerhalb der Benutzeroberfläche. Das kleinste vorliegende Ergebnis liegt dabei zwischen der gewählten Schriftfarbe für die Jahreszahlen (Hexadezimalcode: #707070) und der weißen Hintergrundfarbe (Hexadezimalcode: #FFFFFF). Das Ergebnis beträgt 4,95:1 und erfüllt somit die Vorgabe der AA-Konformitätsstufe (Prinzip Kontrast).
Über den Kamera-Tab (links) kann der Benutzer auf die Kamera zugreifen. Der mittlere Button zeigt die Hauptansicht der Anwendung, die im Wireframe und Mockup (Abbildung 4.11) zu sehen ist. Über den Profil-Tab (rechts) kann der Benutzer seine Daten einsehen. Die Farbe Grün dient als Hauptfarbe für die Progressive Web App, da diese mit der Natur und Gelassenheit assoziiert wird und häufigen Einsatz in der

Tourismusbranche findet (Prinzip Farbe). Der primäre Button erhält eine rötliche bis orange Färbung und fällt durch diese und seine losgelöste Platzierung in der Benutzeroberfläche deutlicher auf (Prinzip Visuelle Hierarchie) und regt zur Interaktion an (Prinzip Farbe).

Die Benutzeroberfläche für IOS folgt einem ähnlichen Aufbau. Im oberen Bereich des Bildschirms sitzt die Suchleiste, der Titel und ein Button für die primäre Funktionalität, neue Einträge zu erstellen. Um die spätere Anpassung der Benutzeroberfläche an die jeweilige Plattform deutlicher differenzieren zu können, werden die Karten auf der IOS-Plattform untereinander und größer angeordnet. Die Gruppierung nach dem jeweiligen Reisejahr erfolgt aber auch hier. Eine Detailansicht der einzelnen Einträge öffnet sich auch über das Tippen auf die jeweilige Karte. Eine Tab Bar liegt im unteren Bereich des Bildschirms.
Für IOS werden dieselben Farben verwendet, die auch Android nutzt (Prinzip der Konsistenz). Entsprechend gleichen sich die Kontrastverhältnisse. Der primäre Button im oberen Bereich des Bildschirms erhält ebenfalls eine farbliche Hervorhebung. Die Anwendung wird größtenteils dennoch in der Farbe Weiß gehalten, um dem minimalistischen und schlichten Design von Apples Human Interface Design Guidelines zu folgen. Die Tab Bar bietet dieselben Funktionen, wie die Navigation Bar der Android-Version. Das Mockup und Wireframe für die IOS-Plattform ist im Anhang C.1 vorzufinden.

4.3.2 Desktop

Der Aufbau und das Design der Wireframes und Mockups der mobilen Benutzeroberflächen werden entsprechend für den Desktop angepasst.
Das Menü liegt horizontal im oberen Bereich des Bildschirms, wie es die Abbildung 4.12 zeigt. Das Menü soll Platz für den Namen der Anwendung, ein Eingabefeld für die Filterungsfunktion und drei weitere Buttons bieten.
Die Einträge des Benutzers reihen sich unterhalb des Menüs auf. Jeder Eintrag wird durch eine eigene Karte repräsentiert, welche aus einem Bild, einem Titel, weiterem Text und einem Button bestehen soll. Über diesen Button ist eine Detailansicht des jeweiligen Reiseeintrags aufrufbar.
Das Mockup für die Benutzeroberfläche des Desktops zeigt zudem die Farbgestaltung der Anwendung, sowie Beispieltexte und die Beschriftung der einzelnen Buttons. Die Farben gleichen sich, nach dem Prinzip der Konsistenz, mit den Farben der mobilen Benutzeroberflächen.
Die hellgraue Farbe innerhalb der Karten ist allerdings neu und nur für den Desktop vorgesehen.
Das kleinste Kontrastverhältnis in der Benutzeroberfläche liegt zwischen der Schriftfarbe (Hexadezimalcode: #9C5551) des Textbuttons *Erinnerung abrufen* und der Hintergrundfarbe (Hexadezimalcode: #F2F2F2) der Karte und hat einen Wert von über 4,9:1 und erfüllt somit die Konformitätsstufe AA.
Der Button *Erinnerung hinzufügen* im Menü stellt die Hauptfunktionalität der Anwendung zur Verfügung, einen neuen Eintrag zu erstellen. Aus diesem Grund wird dieser zusätzlich farblich hervorgehoben, um den Fokus darauf zu legen. Der Button

Profil erlaubt es dem Benutzer seine Daten einzusehen. Über die Suchleiste kann man nach Einträgen suchen. Über den Button *Übersicht* gelangt ein Benutzer zurück zur Hauptansicht der Anwendung, welche im Mockup (Abbildung 4.12) dargestellt wird.

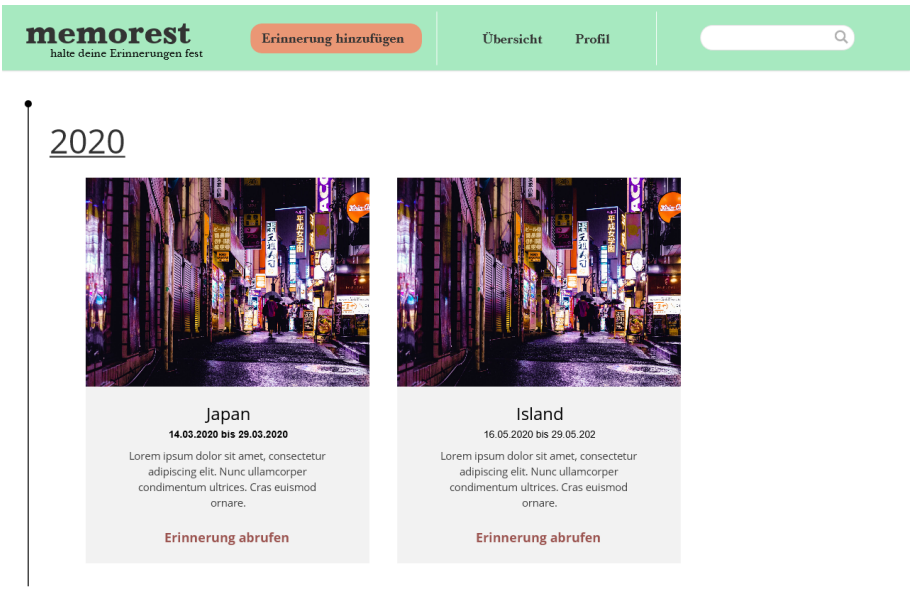

Abbildung 4.12: Mockup für Benutzeroberfläche auf dem Desktop, Quelle: Eigene Darstellung

Die einzelnen Karten der Einträge werden auch hier nach dem jeweiligen Jahr der Reise sortiert und entsprechend der passenden Überschrift zugeordnet. Das Wireframe für die Benutzeroberfläche des Desktops ist im Anhang C.2 zu finden.

5. Umsetzung der Progressive Web App

Wie bereits erläutert, wird im Rahmen dieser Arbeit eine Progressive Web App namens *memorest* entwickelt, welche das Speichern von Reiseerinnerungen ermöglichen soll. Zudem soll die Anwendung dem Benutzer die Grundfunktionen einer Progressive Web App zur Verfügung stellen.

5.1 PWA Grundfunktionalitäten

Die grundlegenden Funktionalitäten einer Progressive Web App werden zu Beginn mit Hilfe des Ionic CLI in das Projekt eingebunden. Das Paket *@angular/pwa* erzeugt dabei automatisch den Service Worker und das Web App Manifest, wodurch die Installierbarkeit und Offline-Fähigkeit der Anwendung gewährleistet wird. Innerhalb des Web App Manifests selbst, ist anschließend der Name der App (Attribute *name* und *short_name*) anzupassen, der am Ende auf dem Startbildschirm des Endgerätes nach einer Installierung der PWA unterhalb des Icons dargestellt werden soll (vgl. Quellcode-Abbildung 5.1). Auch das zu verwendende Icon wird innerhalb dieser JSON-Datei in mehreren Formaten festgelegt. Da für memorest kein eigenes Icon existiert, werden die bereits festgelegten Standard-Icons belassen.

```
1   {
2  "name": "memorest",
3  "short_name": "memorest",
4  "theme_color": "#1976d2",
5  "background_color": "#fafafa",
6  "display": "standalone",
7  "scope": "./",
8  "start_url": "./",
9  "icons": [...]
10  }
```

Quellcode 5.1: Anpassung des Anwendungsnamens im Web App Manifests

5.2 Datenspeicherung

Zur Speicherung der Benutzereingaben wird für die lokale Speicherung PouchDB und für die externe Ablage der Daten CouchDB genutzt.
CouchDB ist dabei nach erfolgreicher Installierung auf dem Entwicklercomputer über die URL *http://localhost:5984/_utils/* erreichbar und öffnet einen Anmeldedialog. Nach erfolgreicher Anmeldung kann man beispielsweise bereits erstellte Datenbanken einsehen oder eigenständig selbst neue anlegen.
Innerhalb der Anwendung behandelt ein Service namens *database.service* Anfragen an PouchDB. Sobald dieser Service aufgerufen wird, wird die Methode *initialiseDB()* ausgeführt, um eine lokale Datenbank namens *memodb* im aktuell verwendeten Browser zu erstellen, falls noch keine lokale Datenbank in diesem Browser mit diesem Namen vorhanden ist. Um eine ausgelagerte Datenbank anschließend in CouchDB zu erzeugen, wird die URL von CouchDB in PouchDB angegeben und angepasst, indem der Datenbankname an diese angehängt wird (*http://localhost:5984/memodb*). Sollte in CouchDB ebenfalls noch keine Datenbank mit diesem Namen existieren, erzeugt CouchDB nach der ersten API-Abfrage von PouchDB diese automatisch. Mit Hilfe dieser URL zu der ausgelagerten Datenbank, synchronisiert PouchDB innerhalb der *syncData()*-Methode enthaltene Datensätze mit CouchDB und umgekehrt.
Neben den bereits genannten Methoden stellt der Service noch folgende, in der Tabelle 5.1 einzusehende Methoden, zur Verfügung.

Tabelle 5.1: Weitere Methoden des database.service

Methodenname	Beschreibung
addEntry()	Fügt neuen Eintrag in Datenbank ein
getEntry()	Liest Eintrag mittels einer id aus der Datenbank aus
getAll()	Liest alle Einträge aus der Datenbank aus
updateEntry()	Aktualisiert Datensatz mittels einer id in der Datenbank
delEntry()	Löscht Datensatz mittels einer id aus der Datenbank

PouchDB benötigt zum Löschen oder zum Aktualisieren von Datensätzen den zu dem jeweiligen Dokument gehörenden *revision marker*, kurz *_rev*. Diese revision marker stellen eine zufällige, von PouchDB generierte id dar, die sich bei jeder Änderung eines Dokuments ebenfalls ändert. PouchDB und CouchDB können so zu jedem Dokument – ähnlich zu Git – eine Revisionshistorie (engl. revision history) in Form eines Baumes erstellen. Dies soll helfen, die Synchronisation zwischen den Datenbanken zu vereinfachen und bei aufkommenden Konflikten diese schneller zu lösen. [Apache Software Foundation, 2022]

5.3 Kamera

Das Kamera-Plugin von Capacitor implementiert die Kamerafunktionalität. Hierzu ist vorab Capacitor in das Projekt einzubinden. Anschließend erfolgt die Installation des Capacitor Kamera-Plugins *@capacitor/camera*. Zusätzlich ist es notwendig, die *PWA Elements Bibliothek* von Ionic über *@ionic/pwa-elements* einzubinden. Diese Bibliothek stellt zu vereinzelten Capacitor Plugins webbasierte Funktionalitäten bereit. Bei dem Kamera Plugin beispielsweise, wird eine interaktive, webbasierte Benutzeroberfläche geladen, über diese die Kamera des jeweiligen Endgerätes gesteuert werden kann, unabhängig des Betriebssystems.
Nach dem Einbinden der nötigen Plugins wird für die Kamerafunktion ein Service namens *photo.service* erstellt. Dabei ermöglicht die Methode *openCamera()* die Kamera des jeweiligen Endgerätes in der erwähnten, webbasierten Benutzeroberfläche zu öffnen. Hierbei ist es wichtig, für das *source*-Attribut den Wert *Camera* des Enums *CameraSource* (*CameraSource.Camera*) festzulegen. Dies ermöglicht es, ein neues Foto mit der Kamera aufzunehmen. Zusätzlich soll das Attribut *saveToGallery* auf *true* gesetzt werden, wodurch das Foto von Anwendung theoretisch lokal in der Gallery des Endgerätes abgelegt werden sollte.
Was sich an dieser Stelle allerdings als Problem herausstellt ist, dass das aufgenommene Foto nicht lokal auf dem Endgerät vorzufinden ist. Auch ein Test auf einem Android-Gerät zeigt, dass keine aufgenommenen Fotos in der Fotogallery des Endgerätes vorzufinden sind. Wieso dies der Fall ist, konnte während der Implementierung nicht herausgefunden werden.
Um dennoch auf eine andere Art aufgenommene Fotos abzuspeichern, könnte die Filesystem API von Capacitor genutzt werden, um die Fotos innerhalb der Anwendung in einer eigenen Fotogallery zu speichern und abrufbar zu machen. Da für die PWA memorest allerdings eine solche Funktionalität, die Fotos in der Anwendung selbst abzulegen, nicht vorgesehen ist, wurde diese Möglichkeit nicht weiterverfolgt.
Neben dem Aufrufen der Kamera ermöglicht der Service *photo.service* mit der Methode *getImage()* in Quellcode-Abbildung 5.2 Bilder vom Endgerät in die Anwendung zu laden.

```
1    public async getImage() {
2    try{
3      const chosenPhoto = await Camera.getPhoto({
4        resultType: CameraResultType.Base64,
5        source: CameraSource.Photos,
6        quality: 80
7      });
8      [...]
9    }
```

Quellcode 5.2: Codeschnipsel für Zugriff auf Bilder des Endgerätes

Dabei wird für das *source*-Attribut der Typ *CameraSource.Photos* festgelegt, um auf die Fotos des Endgerätes zuzugreifen. Das ausgewählte Foto ist anschließend als base64-string (*resultType: CameraResultType.Base64*) von der Anwendung verwendbar.

5.4 Push-Benachrichtigungen

Die Implementierung der Push-Benachrichtigungsfunktionalität erfolgt mit Hilfe des *Firebase Cloud Messaging* (FCM) von Firebase. Firebase ist dabei eine Entwicklungsplattform von Google für mobile Anwendungen und Webanwendungen, welche verschiedene Funktionalitäten und Werkzeuge für Entwickler zur Verfügung stellt.
Grund für die Nutzung des Firebase Cloud Messaging ist, dass dieses sowohl Funktionalitäten für native Anwendungen, als auch für Webanwendungen anbietet und somit auch für Progressive Web Apps.
Auch Capacitor stellt Plugins zur Verfügung, um Push-Benachrichtigungen zu empfangen. Allerdings konnte anhand dessen Dokumentation nicht festgestellt werden, inwiefern sich das Plugin für Progressive Web Apps eignet.
Aus diesem Grund kommt Firebase Cloud Messaging für die Push-Benachrichtigung in memorest zum Einsatz.
Firebase Cloud Messaging nutzt für Push-Benachrichtigungen die Web Push API, welche Push-Benachrichtigungen über den jeweils verwendeten Browser an den Benutzer darstellt und somit plattformunabhängig agiert. Es ist wichtig zu beachten, dass IOS Safari die Web Push API nicht unterstützt und somit keine Push-Benachrichtigungen über diesen Browser empfangen werden können. Für die Implementierung dieser Funktionalität wird der Leitfaden von Simon Grimm – Betreiber der *Ionic Academy* – zu Rate gezogen. [Grimm, 2020]

Zum besseren Verständnis erfolgt eine kurze Zusammenfassung des Leitfadens, in dieser auch zusätzlich vorzunehmende Änderungen zur Sicherstellung der Funktionalität in memorest aufgeführt werden.
Zuerst wird ein Firebase-Konto und anschließend ein Projekt erstellt, über dieses der Firebase Messaging Dienst genutzt werden soll. Danach erfolgt die Einbindung von Angular Fire (*@angular/fire*) in das Projekt. Angular Fire ist die offizielle Bibliothek von Angular für Firebase, um dessen Einbindung in ein Angular Projekt zu erleichtern.
Um Firebase Messaging nutzen zu können, wird in die *environment*-Dateien von Ionic jeweils die Projekt-Konfiguration geschrieben, welche Firebase nach dem Erstellen eines Projektes vorgibt. Anschließend sind noch Änderungen in folgenden Dateien von memorest vorzunehmen, die in der unten aufgeführten Tabelle 5.2 beschrieben sind.

Tabelle 5.2: Zusätzlich vorzunehmende Änderungen für Firebase Cloud Messaging

Dateiname	Änderung
firebase-messaging-sw.js	Neu zu erstellende Datei. Enthält den Service Worker von Firebase, über diesen Push-Benachrichtigungen behandelt werden
combined-sw.js	Neu zu erstellende Datei. Kombiniert den Firebase Service Worker und den bereits zu Beginn eingebundenen Angular Service Worker
angular.json	*firebase-messaging-sw.js* und *combined-sw.js* referenzieren
manifest.webmanifest	GCM ID von Google einfügen ("gcm_sender_id": "103953800507")
app.module	Anstelle des Angluar Service Worker die Datei *combined-sw.js* als Service Worker registrieren, zusätzliche Module importieren

Nach den vorgenommenen Änderungen wird zusätzlich in der *index.html* Datei folgender, in der Quellcode-Abbildung 5.3 zu sehender Codeabschnitt eingefügt, da ansonsten das Web App Manifest keinen Zugriff mehr auf den Angular Service Worker hätte. Dieses Problem ergab sich, nachdem die Push-Benachrichtigungsfunktion fertiggestellt wurde.

```
1  <script>
2    if ('serviceWorker' in navigator) {
3      navigator.serviceWorker.register('ngsw-worker.js')
4        .then(() => console.log('service worker installed'))
5        .catch(err => console.error('Error', err));
6    }
7
8  </script>
```

Quellcode 5.3: Codeschnipsel aus der Datei index.html

Die if-Bedingung prüft, ob der aktuelle Browser Service Worker unterstützt. Falls nicht, werden die Funktionen des Service Worker ausgelassen und die Anwendung ist ohne offline-Funktionalität und ohne Installierbarkeit weiter nutzbar. Werden Service Worker unterstützt, wird der Angular Service Worker (*ngsw-worker.js*) im Browser registriert.

Für die Push-Benachrichtigungen wird in der Anwendung der Service *notification.service* erstellt. Um Push-Benachrichtigungen an den Benutzer senden zu können, muss dieser hierzu vorab die Erlaubnis hierzu geben. Diese Abfrage erfolgt in der Methode *requestPermission()*.
Erteilt der Benutzer die Erlaubnis, registriert dies den Firebase Service Worker im Browser und erzeugt einen Token, der speziell diesem Benutzer zugeordnet wird

und auf einem Server zu dem jeweiligen Benutzerkonto abgespeichert werden sollte. Nachrichten können so als Broadcast an alle Benutzer einer Anwendung oder über den jeweiligen Token an spezielle Benutzer gesendet werden.
Da memorest keine Benutzerkonten implementiert, wird der Token nur lokal in der Anwendung behalten.
Bei Push-Benachrichtigungen wird zwischen Vordergrundbenachrichtigungen und Hintergrundbenachrichtigungen unterschieden. Erstere empfängt ein Benutzer, wenn dieser die Anwendung aktiv bedient und verwendet, beziehungsweise das Endgerät die Anwendung auf dem Bildschirm darstellt. Erhält ein Benutzer eine Push-Benachrichtigung, wenn die Anwendung nicht direkt auf seinem Endgerät zu sehen oder gänzlich geschlossen ist, zählt diese als Hintergrundbenachrichtigung. Beide Arten von Benachrichtigungen müssen unterschiedlich behandelt werden.
Der Service *notification.service* kümmert sich um Push-Benachrichtigen, die eine Anwendung im Vordergrund empfängt. Die *getMessages()*-Methode empfängt dabei eingehende Nachrichten über das *messages*-Objekt von Angular Fire, welches empfangene Nachrichten enthält.

Um Änderungen zu registrieren, also ob eine neue Nachricht angekommen ist, wird *subscribe()* verwendet. Dies ist eine Methode, um bestimmte Abläufe einer Anwendung beobachten zu können und Änderungen zu registrieren. Diese Überwachung von neu eintreffenden Nachrichten erfolgt in der *listenForMessages()*-Methode innerhalb der TypeScript-Datei der Hauptseite der PWA (*tab2.page*).
Die Darstellung der eintreffenden Benachrichtigung in der Benutzeroberfläche erfolgt ebenfalls innerhalb dieser Methode.
Die folgende Quellcode-Abbildung 5.4 zeigt einen Codeausschnitt der *listenForMessages()*-Methode.

```
1  listenForMessages() {
2     this.noteService.getMessages().subscribe(async (msg: any) => {
3       const toast = await this.toastController.create({
4         header: msg.data.title,
5         message: msg.data.body,
6         icon: 'information-circle',
7         position: 'top',
8         duration: 4000,
9
10      });
11      await toast.present();
12    });
13  }
```

Quellcode 5.4: Codeschnipsel zur Darstellung der Vordergrundbenachrichtigungen

Zur Darstellung der Vordergrundbenachrichtigungen in der Benutzeroberfläche werden die erhaltenen Informationen der Nachricht in eine Toast-Nachricht (*toastController*) eingespeist. Der Aufbau, das Aussehen, die Position und die Dauer der Darstellung der Toast-Nachricht wird zudem definiert und anschließend in der Benutzeroberfläche über *toast.present()* dargestellt.

Push-Benachrichtigungen, welche die Anwendung im Hintergrund empfängt, werden in der Datei des Firebase Service Worker (*firebase-messaging-sw.js*) behandelt. Dies erfolgt in der Methode *onBackgroundMessage()*, welche in der Quellcode-Abbildung 5.5 dargestellt ist. Diese Methode erhält einen Parameter, der die Informationen zu der Benachrichtigung enthält, die im Hintergrund empfangen wurde.

```
1 messaging.onBackgroundMessage(function(payload) {
2     return self.registration.showNotification("Memorest Alert",{
3         body: payload.data.body,
4         icon: payload.data.icon,
5       });
6   });
```

Quellcode 5.5: Codeschnipsel für Benachrichtigungen im Hintergrund

Anschließend wird die Push-Benachrichtigung vom jeweiligen Browser mittels *showNotification()* in einem kleinen Fenster außerhalb der Anwendung dem Benutzer dargestellt.

Für das Senden einer Benachrichtigung an die PWA dient die API-Plattform *Postman*. Eine Nachricht ist dabei als JSON-Format anzugeben und wird dann an die Firebase Cloud Messaging API gesendet. Die JSON-Datei hat dabei den in Quellcode-Abbildung 5.6 dargestellten Aufbau.

```
1 {
2  "data": {
3     "title": "Memorest Alert",
4     "body": "Deine Reise Erinnerungen warten nur darauf festgehalten
          zu werden",
5     "icon": "<Link>"
6       },
7  "to": "<Benutzertoken>"
8 }
```

Quellcode 5.6: Aufbau einer Nachricht im JSON-Format in Postman

Wichtig zu erwähnen ist, dass bei Postman in der Benutzeroberfläche im Tab *Headers* der Server-Schlüssel des Firebase-Projektes angegeben werden muss. Zudem muss man die API von Firebase Cloud Messaging (*https://fcm.googleapis.com/fcm/send*) Postman mitteilen und im Dropdown Menü die Methode *POST* auswählen.

5.5 Hybride Benutzeroberfläche

Zur Umsetzung der hybriden Benutzeroberfläche werden in der Anwendung verschiedene Ansätze und Wege angewendet. Zum einen, das Adaptive Styling von Ionic. Zum anderen werden bestimmte Elemente in der Benutzeroberfläche mit Hilfe von gesetzten Bedingungen ein- und ausgeblendet. Auch zusätzlich erzeugte Komponenten oder

Pages für bestimmte Plattformen werden genutzt.
Eine Page ist dabei eine eigene, in sich geschlossene Ansicht innerhalb eines Ionic-Projektes. Ein Ionic-Projekt besteht aus mehreren Pages, zwischen diesen Benutzer am Ende navigieren. So existiert in memorest beispielsweise eine eigene Page für die Detailansicht der jeweiligen Erinnerung oder eine eigene Page für das Formular, über dieses der Benutzer eine Erinnerung einträgt und abspeichert.

5.5.1 Adaptive Styling

Ionic nutzt beim Adaptive Styling verschiedene Modi, charakterisiert durch bestimmte Kürzel, um spezifische Stylings der jeweiligen Plattform zuzuordnen.
Das Kürzel *ios* steht dabei für die Plattform IOS. Führt ein Benutzer die Anwendung auf einem IOS-Gerät aus, wird für alle genutzten Ionic UI-Komponenten in der Anwendung das von Ionic festgelegte Styling für IOS verwendet, welches den Human Interface Design Richtlinien von Apple entspricht. Das Kürzel können Entwickler dabei auch für eigene CSS-Klassen oder zur Anpassung des IOS Stylings einer Ionic UI-Komponente verwenden. Das Kürzel ist dabei vor die jeweilige Klassendefinition zu setzen (siehe Quellcode-Abbildung 5.7). Diese CSS-Klassen mit einem vorangesetzten ios-Kürzel kommen in der Anwendung nur auf IOS-Geräten zum Einsatz.

```
1  .ios ion-button {
2    width: 44px;
3    height: 44px;
4    padding: 4px;
5    --background: var(--ion-color-tertiary);
6  }
```

Quellcode 5.7: Beispiel einer plattformspezifischen CSS-Klasse mit einem ios-Kürzel

Für die Android-Plattform ist das Kürzel *md* zu verwenden. Dieser Modus passt die Ionic UI-Komponenten entsprechend dem Material Design von Google an. Auch Plattformen, die weder Android, noch IOS zugeordnet werden können, wie beispielsweise die Desktop-Plattform, nutzen diesen Modus.

5.5.2 Weitere Anpassungswege der Benutzeroberfläche

Zusätzlich zu dem Adaptive Styling erfolgt eine dynamische Zuordnung von CSS-Klassen innerhalb der HTML-Tags je nach Plattform. Auch bestimmte HTML-Elemente werden je nach Plattform ein- oder ausgeblendet. Hierzu wird mit Hilfe von TypeScript und dem *Platform*-Modul von Ionic vorab geprüft, auf welcher Plattform gerade der Benutzer die Anwendung ausführt. Ein Beispiel zeigt die folgende Quellcode-Abbildung 5.8.

```
1  this.android = platform.is('android');
```

Quellcode 5.8: Beispiel zur Prüfung der aktuellen Plattform (hier Android)

Anschließend stellt ein [class]-Ausdruck innerhalb eines HTML-Tags eine Bedingung. Abhängig davon, ob diese Bedingung zutrifft, wird die angegebene Klasse dem HTML-Tag zugeordnet oder nicht. In der Quellcode-Abbildung 5.9 wird beispielsweise die CSS-Klasse *androidheader* dem HTML-Tag <*ion-header*> übergeben, falls der Benutzer die Anwendung auf einem Android-Gerät ausführt. Auf einer IOS-Plattform erfolgt die Zuordnung der CSS-Klasse *iosheader*.

```
1  <ion-header [translucent]="true" [class.androidheader]="android" [
       class.iosheader]="ios">
```

Quellcode 5.9: Dynamische Zuordnung einer CSS-Klasse je nach erfüllter Bedingung

Um ein HTML-Tag für eine spezifische Plattform auszublenden, wird die strukturelle Direktive **ngIf* von Angular verwendet. Trifft die Bedingung zu, ist das HTML-Tag von der Anwendung darzustellen. Trifft die Direktive nicht zu, wird das HTML-Tag ausgeblendet. Folgende Quellcode-Abbildung 5.10 zeigt hierzu ein Beispiel.

```
1  <ng-container *ngIf="!isDesktop">
2    [...]
3  <ng-container>
```

Quellcode 5.10: Beispiel zur strukturellen Direktive *ngIf

Im obigen Codeschnipsel wird **ngIf* innerhalb eines <*ng-container*> verwendet, der einen HTML-Block umschließt, um diesen an eine Bedingung zu knüpfen. In diesem Fall soll der umschlossene HTML-Block nur auf mobilen Plattformen zu sehen sein. Der Vorteil eines <*ng-container*> zu beispielsweise einem <*div*>-Container ist, dass dieser nicht in der Benutzeroberfläche gerendert wird und so die Struktur des HTML-Dokuments nicht beeinflusst.

Plattformspezifische Pages

Da vereinzelte Pages in ihrem Aufbau nicht für jede Plattform geeignet sind und alleine die Veränderung der CSS-Klassen oder des HTML nicht ausreichen würde, um die Page an eine Plattform anzupassen, werden für memorest zusätzliche, komplett überarbeitete plattformspezifische Pages erstellt. Folgende Tabelle 5.3 liefert einen Überblick über die erstellten Pages in memorest und deren zugeordnete Plattform.

Tabelle 5.3: Pages und ihre vorgesehene Plattform

Page-Name	Plattform	Funktion
tabs.page	Mobil, Desktop	Tab Bar zur Navigation mit drei Tabs
tab1.page	Desktop	Formular zum Erstellen neuer Erinnerungen
tab2.page	Mobil, Desktop	Hauptansicht; Sammlung der eingetragenen Erinnerungen
tab3.page	Mobil, Desktop	Dummy-Ansicht des Profils
formslidepage.page	Mobil	Formular zum Erstellen neuer Erinnerungen
detailpage.page	Mobil, Desktop	Detailansicht einer Erinnerung

Wie man aus der Tabelle 5.3 entnehmen kann, existieren für das Erstellungsformular von neuen Erinnerungen zwei verschiedene Pages. Mobile Plattformen verwenden die *formslidepage.page*, wenn ein Benutzer eine neue Erinnerung erstellen möchte. Auf der Desktop-Version der Anwendung stellt die *tab1.page* das Formular zur Erstellung neuer Erinnerungen zur Verfügung. Die Regelung, auf welche dieser Pages die Anwendung je nach Plattform navigieren soll, erfolgt folgendermaßen: Die PWA memorest besitzt zur Navigation eine Tab Bar (*tabs.page*), die auf mobilen Plattformen im unteren Bereich des Bildschirms sitzt. Die Abbildung 5.1 zeigt diese Tab Bar auf der Android-Version der Benutzeroberfläche. Die Hauptansicht (*tab2.page*) der Anwendung ist über den mittigen Tab erreichbar. In der Hauptansicht der Anwendung sitzt zur Erstellung neuer Einträge ein losgelöster Button. Dieser Button leitet den Benutzer nach einer ausgeführten Interaktion auf die Page *formslidepage.page*, welche ausschließlich für mobile Plattformen ausgerichtet ist.

Abbildung 5.1: Tab Bar und losgelöster Button auf mobiler Plattform (Android), Quelle: Eigene Darstellung

Auf der Desktop-Version schmiegt sich die Tab Bar der Page *tabs.page* an den oberen Bildschirmrand (siehe Abbildung 5.2) und wird durch entsprechende HTML und CSS-Anpassungen zu einem Webseiten-Header umgeformt. Dieser Header enthält einen Button. Über diesen ist das Formular für den Desktop zur Erzeugung einer neuen Erinnerung auf der Page *tab1.page* erreichbar.

Abbildung 5.2: Angepasste Tab Bar auf dem Desktop, Quelle: Eigene Darstellung

Die Anpassung der Tab Bar erfolgt in der HTML-Datei der *tabs.page* über die strukturelle Direktive **ngIf* innerhalb des HTML-Tags <*ion-tab-bar*>, welches die UI-Komponente von Ionic repräsentiert. Je nach erfüllter Bedingung rendert die Anwendung die Tab Bar für den Desktop oder die Tab Bar für die mobile Version.

5.5.3 Umgesetzte Design-Prinzipien

Eine der Anforderungen an memorest ist, dass Design-Prinzipien und Richtlinien aus dem Kapitel 4.2 *Design-Prinzipien zur Gestaltung hybrider Benutzeroberflächen* für die hybride Benutzeroberfläche zum Einsatz kommen.
Gemäß der in Kapitel 4.3 festgelegten Mockups und den dabei erörterten, eingesetzten Design-prinzipien für die Hauptansicht der Anwendung, erfolgt die Gestaltung der hybriden Benutzeroberfläche bei der Umsetzung von memorest.
Um an dieser Stelle ein weiteres Beispiel zu geben, werden im Folgenden die verwendeten Design-Prinzipien innerhalb der Formularansicht zur Erstellung neuer Erinnerungen auf dem Desktop und den mobilen Plattformen aufgeführt. Bilder zu den Formularen in der Anwendung sind im Anhang D einsehbar.

Auf der mobilen Plattform wird das Formular (*formslidepage.page*) gemäß dem Prinzip der Simplicity und Progressive Disclosure auf mehrere Abschnitte der Anwendung aufgeteilt. So kann sich der Benutzer auf die Angabe der jeweiligen Daten besser fokussieren. Die Aufteilung erfolgt dabei mit Hilfe eines *ion-slides*, den Entwickler mit mehreren, einzelnen Slides befüllen können. Ein Slide entspricht dabei in der Anwendung einem Formularabschnitt.
Des Weiteren werden alle erforderlichen Angaben eines Datums über einen interaktiven Kalender ermöglicht, der nach einem Klick auf das Eingabefeld erscheint. So reduziert sich die Nutzung der virtuellen Tastatur auf dem Smartphone für den Benutzer auf die Eingabe des Landes, der Orte und, falls gewünscht, näherer Beschreibungen zu der jeweiligen Teilerinnerung.
Die Buttons innerhalb des Formulars erhalten auf Grund der Konsistenz innerhalb der Anwendung die Farbe Grün. Außerdem wir den Buttons die empfohlene Interaktionsfläche von mindestens 44x44pt zugesprochen und genügend Platz zwischen den

einzelnen Interaktionselementen gesetzt.
Die Eingabeelemente des Formulars sind mittig des Bildschirms angeordnet, um den nach Hoober definierten und von Smartphone Nutzern präferierten Interaktionsbereich einzuhalten.
Gemäß des Design-Prinzips Alignment sind die einzelnen Eingabefelder mit ihren darüberliegenden Instruktionen untereinander und linksbündig angeordnet. Um zu verdeutlichen, welche Instruktion zu welchem Eingabefeld gehört, wird der Abstand zwischen einem Eingabefeld und der dazugehörigen Anweisung verkleinert, wodurch gemäß des Prinzips Proximity die näher zusammenliegenden Elemente einen Einheitsblock bilden.
Das kleinste Kontrastverhältnis liegt zwischen der Schrift (Hexadezimalcode: #000000) und dem Hintergrund der Buttons (Hexadezimalcode: #A8EBBF) und nimmt den Wert von 15,3:1 an (Prinzip Kontrast), was der Konformitätsstufe AAA entspricht. Des Weiteren werden zu falschen Eingaben unterhalb des jeweils betroffenen Eingabefeldes rote Fehlerbenachrichtigungen dargestellt, sobald der Benutzer aus einem Eingabefeld navigiert. Die Farbe Rot soll die Aufmerksamkeit des Benutzers wecken und außerdem signalisieren, dass mit der Eingabe etwas nicht stimmt (Prinzip Farbe).

Das Formular auf dem Desktop nimmt eine größere Darstellungsfläche ein und teilt sich auf zwei Spalten auf. Ähnlich zu der mobilen Version des Formulars werden die Elemente entsprechend der Prinzipien Alignment und Proximity angeordnet. Des Weiteren wird das Formular ebenfalls auf mehrere Teilformulare aufgeteilt, um es einfacher und übersichtlicher zu gestalten. Die Aufteilung erfolgt allerdings nicht über einzelne Slides, sondern über das Ein- beziehungsweise Ausblenden von untereinanderliegenden HTML-Blöcken.
Hat ein Benutzer den ersten Formularabschnitt ausgefüllt und klickt auf den Button *Weiter*, so erfolgt die Einblendung des nächsten HTML-Formularabschnittes über die strukturelle Direktive **ngIf* innerhalb des jeweiligen -HTML-Tags. Für diese Bedingung wird eine Variable namens *formBlockId* definiert und dieser dabei die Zahl Null zugewiesen. Jedes Mal, wenn der Weiter-Button im Formular betätigt wird, erfolgt eine Erhöhung des Variablenwertes um eins, was daraufhin die strukturelle Direktive des entsprechenden HTML-Blocks auslöst. Ein diesbezügliches Beispiel zeigt die folgende Quellcode-Abbildung 5.11.

```
1 <!--Second form block-->
2   <div id="secondpart" *ngIf="formBlockId >= 1">
```

Quellcode 5.11: Strukturelle Direktive für den zweiten Formularabschnitt

Erreicht die Variable *formBlockId* einen Wert größer oder gleich eins, so rendert die Anwendung den zweiten Formularabschnitt.
Das kleinste Kontrastverhältnis innerhalb des Formulars hat, wie bei der mobilen Version, den Wert von 15,3:1. Auch die Darstellung von Fehlermeldungen zu falschen Eingaben, sind unter dem jeweiligen Eingabefeld in der Farbe Rot ersichtlich.

6. Evaluation

Im Folgenden wird zum einen geprüft, ob die entwickelte Progressive Web App memorest die erforderlichen Anforderungen einer PWA erfüllt. Hierzu wird das *Google Lighthouse Check* - Tool verwendet, welches Progressive Web Apps in den fünf Kategorien *Geschwindigkeit* (Perfomance), *Barrierefreiheit* (Accessibility), *Sicherheit* (Best Practice), *SEO* (Suchmaschinenerfassung) und *PWA* testet und bewertet. Des Weiteren wird auf die Arbeit mit dem Webframework Ionic und seiner Standardintegration Angular eingegangen, mit Bezugnahme auf die in Kapitel 3 genannten Vorteile, die Ionic laut der durchgeführten Literaturrecherche bieten soll.
Am Ende des Kapitels erfolgt noch ein kurzer Diskurs, ob das Konzept der Progressive Web App im Vergleich zu nativen Anwendungen, Web Apps und hybriden Anwendungen in Zukunft noch mehr an Bedeutung gewinnt oder nicht.

6.1 Google Lighthouse Check

Der Google Lighthouse Check besteht aus fünf Kategorien. Diese Kategorien sind folgendermaßen beschrieben. [Digital Guide Ionos, 2021b] [Kotthoff, 2020]

Perfomance Das Kriterium Perfomance beschreibt die Geschwindigkeit beziehungsweise Performanz der getesteten PWA. Dabei wird unter anderem geprüft, wie schnell die erste Darstellung eines Elements erfolgt, wann die Kerninhalte der Anwendung vollständig einzusehen sind und wann die PWA komplett interaktionsfähig ist.

Accessibility Das Kriterium der Accessibility prüft, inwiefern man die PWA barrierefrei benutzen kann, also ob diese unabhängig von physischen, psychischen oder technologischen Einschränkungen des jeweiligen Benutzers von diesem in all ihren Funktionalitäten nutzbar ist.

Best Practice Das Kriterium Best Practice analysiert die Sicherheitskriterien einer PWA. Dabei prüft der Google Lighthouse Check beispielweise die PWA auf unsichere oder veraltete, genutzte Bibliotheken oder Pakete. Auch, ob beispielsweise die PWA das Verschlüsselungsprotokoll TLS verwendet oder die genutzten Datenbanken sicher sind, wird berücksichtigt.

SEO (Search Engine Optimization) Das Kriterium SEO untersucht, wie gut Suchmaschinen die Progressive Web App erfassen und finden und somit bei Suchergebnissen anzeigen könnten. Dabei fließt auch der Inhalt der PWA selbst mit in die Bewertung dieses Kriteriums mit ein. Dazu gehört beispielweise, ob Bilder mit einem *alt*-Attribut im HTML versehen sind, welches den darzustellenden Text für das Bild beinhaltet, falls dieses aus bestimmten Gründen nicht gerendert werden kann. Auch, ob sich die PWA weitestgehend an mobile Plattformen anpasst oder ob die Seite einen *Title*-HTML-Tag und Meta-Beschreibungen enthält, fließt in die Prüfung mit ein.

PWA Das letzte Kriterium *PWA* prüft, ob die Progressive Web App tatsächlich die grundlegenden technologischen Anforderungen einer PWA, wie die Installierbarkeit durch das Web App Manifest oder die Offline-Fähigkeit durch den Service Worker, bereitstellt.

Durchführung

Der Google Lighthouse Check wird im Rahmen der Arbeit zwei Mal durchgeführt. Zum einen auf einem lokalen http-Server. Auf diesem bewertet der Lighthouse Check die Progressive Web App memorest in besagten Kategorien folgendermaßen:

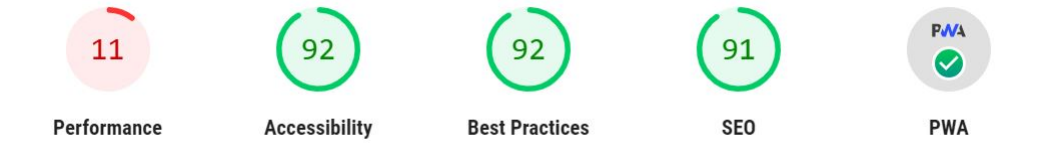

Abbildung 6.1: Google Lighthouse Check Ergebnis zu dem lokalen http-Server

Das Ergebnis in Abbildung 6.1 zeigt, dass memorest in vier der fünf Kategorien sehr gut abschneidet. Die Performance ist mit einer Wertung von elf sehr schlecht bewertet. Google Lighthouse empfiehlt dabei zur Verbesserung der Performanz von memorest unter anderem, auf dem Server *Text Kompression* einzuschalten oder die Verkleinerung von JavaScript Dateien. Da diese Empfehlungen hauptsächlich auf dem jeweiligen Server umzusetzen sind, erfolgt ein weiterer Test von memorest in einer realen Server-Umgebung. Hierzu dient im Rahmen des Tests temporär *Firebase-Hosting*.
Nach erneutem Ausführen des Google Lighthouse Checks erhält memorest für das Kriterium der Performance eine Wertung von 78 Punkten. In beiden Vergleichen erfüllt memorest außerdem die geforderten Grundfunktionalitäten einer Progressive Web App, was an dem grün gefärbten Kriterium *PWA* erkennbar ist. Das Kriterium SEO verliert im Vergleich zum vorherigen Test acht Punkte, wie es in der folgenden Abbildung 6.2 zu sehen ist.

Abbildung 6.2: Google Lighthouse Check Ergebnis von Firebase-Hosting

Grund für den Punkteverlust im SEO-Kriterium ist laut Google Lighthouse, dass memorest keine gültige *robots.txt* Datei enthält, welche dafür zuständig ist, Crawlern vorzugeben, welche Bereiche der Webseite sie durchsuchen dürfen. Crawler sind dabei Bots, die im Internet Informationen sammeln, anhand dieser dann Suchmaschinen passende Webseiten zu bestimmten Suchanfragen dem Benutzer darstellen können. [Digital Guide Ionos, 2021a] Da memorest keine robots.txt Datei enthält, müsste diese zur Verbesserung dieses Kriteriums zusätzlich erstellt werden, was im Rahmen dieser Arbeit aber nicht mehr vorgenommen wird.

6.2 Bewertung von Ionic

Die abgeschlossene Implementierung der Progressive Web App memorest mit Hilfe von Ionic mit seiner Standardintegration Angular ermöglicht es, folgendes Fazit zu dem gewählten Webframework zu ziehen.
Die Einarbeitung in Ionic erfordert, wie bereits als eines seiner Vorteile im Kapitel des Webframework-Vergleichs erörtert, tatsächlich einen geringen Aufwand. Hilfreich dabei war zum einen die Dokumentation von Ionic, auch wenn es an einigen Stellen zum besseren Verständnis notwendig war, zusätzliche Quellen einzusehen. Das offizielle Forum und der Blog von Ionic waren zwei von diesen zusätzlichen Quellen. Zudem waren bei aufkommenden Problemen während der Implementierung zum Großteil hilfreiche Antworten und Lösungswege für Ionic, beispielweise auch auf Stack Overflow, vorzufinden.
Speziell in Bezug auf das Konzept der Progressive Web App waren die PWA Grundfunktionen durch das integrierte PWA Plugin des Ionic CLI innerhalb eines Tages und ohne großen Einarbeitungsaufwand realisierbar.
Zur Gestaltung und Umsetzung der Progressive Web App waren vor allem Ionics vorgefertigte Komponenten aus der UI-Components-Bibliothek hilfreich, wodurch es meist möglich war, innerhalb von ein bis drei Tagen eine Page größtenteils fertigzustellen. Auch das Adaptive Styling und anderweitige, gebotene Funktionalitäten, wie das Überprüfen der Anwendung auf die aktuell genutzte Plattform mit dem Platfom-Modul, waren nützlich, um die hybride Benutzeroberfläche aufzubauen.
Das Testen der hybriden Benutzeroberfläche war im Google Chrome Browser mit Hilfe des Chrome-Entwickler-Tools möglich. In diesem konnten über die *device toolbar* verschiedene IOS- und Android-Geräteemulatoren ausgewählt und die dynamische Anpassung der Benutzeroberfläche nach einem erneuten Aufrufen der PWA beobachtet werden.
Ionic bietet zum Testen des Adaptive Stylings in der Anwendung zudem selbst einen

Android- und IOS-Emulator, der mit Hilfe des Befehls *ionic serve - -lab* für die jeweilige Anwendung aufrufbar ist. Dieser Emulator vermischte allerdings beim Testen von memorest häufig das plattformspezifische Styling der einzelnen Komponenten. Teilweise wurde die Benutzeroberfläche von Android auch in dem vorgesehen IOS-Emulator dargestellt und umgekehrt, weshalb diese Testmöglichkeit auf Grund der fehlerhaften Darstellung nicht weiter genutzt wurde.
Die nativen Funktionalitäten, wie die Kamera, waren an sich ebenfalls leicht durch Capacitor in die Anwendung integrierbar. Allerdings ist die Dokumentation von Capacitor teilweise nicht ausführlich genug formuliert, weshalb die Suche nach zusätzlichen Informationen in anderweitigen Quellen notwendig war. Auch inwiefern sich ein bestimmtes Capacitor Plugin für Progressive Web Apps nutzen lässt, war häufig nicht wirklich erkennbar, weshalb für die Push-Benachrichtigungen letzten Endes Firebase Cloud Messaging zum Einsatz kam. Des Weiteren funktionierte das Kamera-Plugin von Capacitor, wie bereits erläutert, zum Teil nicht, da Fotos nicht lokal auf dem Endgerät gespeichert werden konnten, trotz befolgter Dokumentation und Leitfäden.

Trotz einiger aufgetretenen Schwierigkeiten und Schwächen, zeigt die Implementierung der Progressive Web App dennoch, dass Ionic mit seiner Standardintegration Angular tatsächlich ein gutes Webframework zur Umsetzung von Progressive Web Apps ist und untermauert so das Endergebnis des zuvor durchgeführten Webframework-Vergleichs.

6.3 Das Konzept der Progressiven Web App im Vergleich

Das Konzept der Progressive Web App bietet, wie es im Rahmen dieser Arbeit deutlich wird, vor allem gegenüber den normalen Web Apps mehrere Vorteile. Dazu zählen unter anderem die Offline-Fähigkeit, die Installierbarkeit, der Zugriff auf native Funktionalitäten und die Anpassung der Benutzeroberfläche an die jeweilige Plattform.
Im Vergleich zu nativen Anwendungen bieten Progressive Web Apps die Möglichkeit, diese plattformunabhängig zu nutzen, was die Entwicklungszeit, eine Anwendung an jedes Betriebssystem anzupassen, deutlich verringert und die Produktionskosten senkt.
Laut *topflight* – einer Firma zur Entwicklung von Web und mobilen Anwendungen – können ihren Erfahrungen nach die Entwicklungskosten einer Progressive Web App 30 % der Entwicklungskosten einer nativen Anwendung betragen. [Tuan, 2022]
Grund hierfür ist, dass native Anwendungen je nach Plattform einen neuen Quellcode benötigen. Zudem müssen hierbei für jede Plattform Entwickler mit entsprechenden Fähigkeiten zum Einsatz kommen, da jede Plattform unterschiedliche Programmiersprachen für ihre nativen Anwendungen nutzt. Des Weiteren können durch die spezifischen Programmiersprachen Code-Komponenten der Anwendung für eine andere Plattform nicht wiederverwendet werden. [Marcak, 2021]
Dennoch sind Progressive Web Apps im Vergleich zu nativen oder hybriden Anwendungen, vor allem in Bezug auf den Zugriff der Hardware-Komponenten des jeweiligen Endgerätes, noch im Nachteil. Auch unter Berücksichtigung der Aspekte Sicherheit und Performance liegen nach wie vor native Anwendungen vor Progressive Web Apps. [Marcak, 2021]

Ob eine Progressive Web App eine Hardware-Komponente benutzen kann, hängt stark von der Verfügbarkeit der Web APIs und deren Unterstützung des jeweiligen Browsers ab, in welchem die PWA ausgeführt wird. Vor allem der Browser Safari von IOS unterstützt einige dieser Web APIs nicht. Auch bei der Entwicklung von memorest wird deutlich, dass beispielsweise Push-Benachrichtigungen der Web-Push API nicht mit IOS Safari nutzbar sind. Auch, dass nach wie vor vereinzelte Browser, wie Firefox auf dem Desktop, die Installierbarkeit von Progressive Web Apps nicht unterstützen, ist für Progressive Web Apps nicht von Vorteil. Zudem lässt die mangelnde Unterstützung seitens Apple gegenüber Progressive Web Apps weiterhin native Anwendungen auf der IOS-Plattform im Fokus stehen.
Obwohl Progressive Web Apps im Vergleich zu den nativen Anwendungen in manchen Aspekten im Nachteil sind, fördert vor allem Google weiterhin das Konzept der Progressive Web App. So bietet der Google Play Store auf *Chrome OS* und Android mittlerweile Progressive Web Apps zum Download an und erleichtert es Entwicklern, ihre PWA dort zu vertreiben. Alleine auf Chrome OS, ein von Google entwickeltes Betriebssystem für das Chromebook, ist die Installierungsrate von Progressive Web Apps seit 2021 um 270 % gestiegen. [Gvak and Mistry, 2022]
Auch Microsoft fördert Progressive Web Apps und bietet diese, ähnlich wie der Google Play Store, über den Microsoft Store an. Im App Store von Apple können vereinzelt ebenfalls Progressive Web Apps angeboten werden, allerdings wird dies nicht in dem Umfang unterstützt, wie es beispielsweise beim Google Play Store der Fall ist. [Reshetilo, 2021]
Des Weiteren genießen Progressive Web Apps auch bei größeren Firmen immer mehr Aufmerksamkeit. So nutzen beispielsweise Starbucks, Twitter oder Spotify Progressive Web Apps, um ihre Dienste ihren Kunden anzubieten, woraus geschlossen werden kann, dass Progressive Web Apps heute und auch in naher Zukunft Bestandteil der App-Landschaft bleiben. [ZYMR INC., 2020] Zudem hat Twitter seine native Anwendung im Google Play Store für Chromebooks durch eine PWA ersetzt. Laden Benutzer über diesen Twitter herunter, erfolgt die Installation der Progressive Web App von Twitter auf ihrem Chromebook, anstatt der nativen Anwendung. [Payne, 2020]

Im Großen und Ganzen bieten Progressive Web Apps durch ihre Ähnlichkeit zu nativen Anwendungen eine gute Möglichkeit, eine Anwendung vergleichsweise günstig und mit einem geringeren Entwicklungsaufwand dem Endnutzer über verschiedene Plattformen anzubieten. Ob allerdings Progressive Web Apps tatsächlich eines Tages native Anwendungen ersetzen, ist zu diesem Zeitpunkt nicht eindeutig erkennbar. Vor allem die mangelnde Unterstützung von Progressive Web Apps durch Apple, weist native Anwendungen weiterhin auf IOS eine große Bedeutung zu. Auch der Zugriff und die Unterstützung der nativen Funktionalitäten über Web APIs muss hierzu von den jeweiligen Browsern weiter ausgebaut werden, um das gleiche Funktionsspektrum von nativen Anwendungen zu erreichen.
Auch wenn Progressive Web Apps zum jetzigen Zeitpunkt andere Anwendungskonzepte nicht gänzlich ersetzen können, haben sie doch in der App-Landschaft innerhalb der letzten Jahre stetig an Bedeutung gewonnen und sich bisher in Entwicklerkreisen etabliert.

7. Fazit und Ausblick

Das Fazit fasst im Folgenden die wichtigsten Ergebnisse der Arbeit zusammen. Anschließend erfolgt ein kurzer Ausblick über das Konzept der Progressive Web App.

7.1 Fazit

Die Arbeit befasste sich mit Untersuchungen am Konzept der Progressive Web App, deren Ergebnisse mit Hilfe eines zu entwickelnden PWA-Prototypen namens memorest überprüft wurden. Dabei lag das Kriterium App-Like einer Progressive Web App im Fokus und wie sichergestellt werden kann, dass eine Progressive Web App das Aussehen nativer Anwendungen auf mobilen Plattformen, sowie auf dem Desktop, annimmt. Des Weiteren wurde untersucht, inwiefern sich bestimmte Webframeworks zum jetzigen Zeitpunkt eignen, eine Progressive Web App zu entwickeln und in welcher Hinsicht diese zudem Funktionalitäten anbieten, das App-Like-Kriterium zu erfüllen.
Vorab erfolgte hierzu ein Webframework-Vergleich anhand eines Kriterienkatalogs auf Grundlage einer Literaturrecherche zwischen den Webframeworks React, Vue, Ionic sowie deren Ionic-Integrationen Ionic React und Ionic Vue. Das Ergebnis des Vergleichs zeigte, dass Ionic mit seiner Standardintegration Angular die bestmöglichen Funktionalitäten zur Entwicklung von Progressive Web Apps bietet. Durch das Adaptive Styling von Ionic und seine an Android und IOS angepassten UI-Komponenten, sind Benutzeroberflächen von Progressive Web Apps mit Hilfe des Webframeworks entsprechend dem Aussehen nativer Anwendungen anpassbar. Das von Ionic entwickelte Capacitor ermöglicht zudem Entwicklern eine komfortablere Möglichkeit native Funktionalitäten in eine Progressive Web App einzubinden. Auch die Aufführung von Leitfäden zur Entwicklung von Progressive Web Apps in der Dokumentation und die schnelle Implementierung der Grundfunktionalitäten einer PWA durch den Ionic CLI, sprachen in dem Vergleich für das Framework Ionic.
Zur Klärung, wie sichergestellt werden kann, dass eine Benutzeroberfläche auf der jeweiligen Plattform nativ gestaltet ist, erfolgte eine Erörterung von verschiedenen Design-Prinzipien und Richtlinien zur Gestaltung von Benutzeroberflächen nativer Anwendungen auf mobilen Plattformen und dem Desktop. Die plattformspezifischen Styleguides Material Design von Google für Android und das Human Interface Design von Apple für IOS, sind dabei ebenfalls aufgeführt und näher beleuchtet worden.
Die abgeschlossene Implementierung der Progressive Web App memorest mit Hilfe

von Ionic, bestätigte größtenteils die Ergebnisse zu diesem Webframework aus dem Webframework-Vergleich. Es war ein geringer Einarbeitungsaufwand erforderlich, um mit der Umsetzung des Projektes beginnen zu können. Zudem brachten die im Webframework-Vergleich aufgezeigten Funktionalitäten Ionics für Progressive Web Apps einen tatsächlichen Mehrwert bei der Implementierung der PWA. Allerdings konnten die nativen Funktionalitäten, wie Push-Benachrichtigungen, nicht so einfach wie erwartet, durch Ionics Capacitor umgesetzt werden.
Die recherchierten Design-Prinzipien kamen auch bei der Gestaltung und Entwicklung der hybriden Benutzeroberfläche der Progressive Web App zum Einsatz. Durch die Kombination der Design-Prinzipien mit den angebotenen Funktionalitäten von Ionic war es möglich, eine hybride Benutzeroberfläche zu entwickeln, die sich an die Desktop-Plattform sowie die mobilen Plattformen anpasst und dabei deren native Aspekte aufgreift.
In einem abschließenden Vergleich, inwiefern Progressive Web Apps andere Anwendungskonzepte in Zukunft ersetzen könnten, zeigte sich, dass Progressive Web Apps auf unterschiedlichen Plattformen unterschiedlich stark vertreten sind. Vor allem auf Googles Chrome OS und auf Windows gewinnen Progressive Web Apps immer mehr an Bedeutung. Auch auf Android werden Progressive Web Apps durch den Google App Store unterstützt. Apple hingegen vernachlässigt das Konzept der Progressive Web App, weshalb auf IOS auch native Anwendungen weiterhin starken Bestandteil haben werden.
Zum Schluss konnte keine klare Aussage darüber getroffen werden, ob Progressive Web Apps vor allem native Anwendungen in Zukunft gänzlich ersetzen könnten. Der Vergleich zeigte aber, dass Progressive Web Apps definitiv Teil der App-Landschaft geworden sind. Auch durch die Verwendung dieses Konzepts durch große Unternehmen, wie Twitter oder Starbucks, werden Progressive Web Apps zumindest in naher Zukunft Teil dieser App-Landschaft bleiben.

7.2 Ausblick

Durch die gewonnenen Erkenntnisse dieser Arbeit ergeben sich weitere mögliche Fragestellungen:
Nehmen Benutzer die hybriden Benutzeroberflächen von Progressive Web Apps tatsächlich durch die berücksichtigten Design-Prinzipien als nativ wahr? Treffen die Ergebnisse des Webframework-Vergleichs auf Grundlage der durchgeführten Literaturrecherche auch auf die restlichen Webframeworks zu?
Um diese Fragen zu beantworten, ist jeweils eine Untersuchung erforderlich, die über eine reine Literaturrecherche hinausgeht. Um die Benutzerwahrnehmung zu erforschen, könnte beispielsweise eine Benutzerumfrage anhand verschiedener Prototypen mit unterschiedlich entwickelten hybriden Benutzeroberflächen und eingesetzten Design-Prinzipien durchgeführt werden.
Zur Beantwortung der zweiten Frage könnte, ähnlich zu dieser Arbeit, mit den übrigen Webframeworks des durchgeführten Webframework-Vergleichs jeweils eine Progressive Web App mit einer hybriden Benutzeroberflächen entwickelt und anschließend deren Umsetzungsprozess evaluiert werden.

Generell ist davon auszugehen, dass das Konzept der Progressive Web App auch zukünftig von Bedeutung sein wird. Vor allem Google ist dabei eine treibende Kraft von Progressive Web Apps. Durch die Möglichkeit, Progressive Web Apps über verschiedene App Stores beziehen zu können, ist es für Kunden und Benutzer einfacher, eine PWA zu nutzen. Des Weiteren dürften Entwickler durch die Möglichkeit, vergleichsweise schnell und kostengünstig eine plattformunabhängige Anwendung mit weitestgehend nativen Funktionalitäten zu implementieren, dieses Konzept weiterhin verfolgen. Apples mangelnde Unterstützung zu Progressive Web Apps dürfte hingegen die Bedeutung dieser zumindest auf IOS weiterhin geringhalten. Inwiefern sich diese Begebenheit aber generell auf das Konzept der PWA in Zukunft auswirkt, ist schwer zu sagen. Untersuchungen zu Apples zukünftigen Vorhaben und Entwicklungen gegenüber Progressive Web Apps könnten dabei hilfreich sein, diesbezügliche Rückschlüsse zu treffen.
Sollte Apple dennoch eines Tages seine Unterstützung gegenüber Progressive Web Apps ähnlich zu Google ausbauen, könnten Progressive Web Apps tatsächlich die App-Landschaft weitestgehend dominieren.

Literaturverzeichnis

[AllAccessible, 2022] AllAccessible (2022). WCAG Level A, AA, and AAA What's the Difference? "https://www.allaccessible.org/wcag-level-a-aa-and-aaa-whats-the-difference/". – Abgerufen: 13.05.22.

[altexsoft, 2019] altexsoft (2019). The Good and the Bad of Ionic Mobile Development. "https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-ionic-mobile-develop". – Abgerufen: 26.03.2022.

[Apache Software Foundation, 2022] Apache Software Foundation (2022). PouchDB - The Database that Syncs! "https://pouchdb.com/". – Abgerufen: 02.06.22.

[Apple, 2022] Apple (2022). Human Interface Guidelines. "https://developer.apple.com/design/human-interface-guidelines/". – Abgerufen: 17.05.22.

[arocom, 2019] arocom (2019). Vue.js Fachbegriffe Einfach Erklärt. "https://www.arocom.de/fachbegriffe/webentwicklung/vuejs". – Abgerufen: 16.03.22.

[Bar, 2022] Bar, A. (2022). What Web Can Do Today? "https://whatwebcando.today". – Abgerufen: 05.05.22.

[Bastakoti, 2022] Bastakoti, A. (2022). Using Native Mobile Services In ReactJS. "https://www.theseus.fi/bitstream/handle/10024/703430/Bastakoti_Amrit.pdf?sequence=2". – Abgerufen: 21.03.2022.

[Beinert, 2022] Beinert, W. (2022). Typolexikon. "https://www.typolexikon.de/". – Abgerufen: 14.05.22.

[Bühler et al., 2019] Bühler, P., Schlaich, P., and Sinner, D. (2019). *Digital Publishing*. Springer Vieweg.

[Code-Boost, 2020] Code-Boost (2020). Intro to Declarative Routing with React Router. "https://www.code-boost.com/react-router-intro". – Abgerufen: 11.04.22.

[Cooper et al., 2014] Cooper, A., Reimann, R., Cronin, D., and Noessel, C. (2014). *About Face: The Essentials of Interaction Design 4. Auflage*. Wiley Publishing, Inc.

[Costa, 2020] Costa, R. (2020). Navigation design: Almost everything you need to know. "https://www.justinmind.com/blog/navigation-design-almost-everything-you-need-to-know/". – Abgerufen: 16.05.22.

[Costa, 2021] Costa, R. (2021). Mobile navigation: patterns and examples. "https://www.justinmind.com/blog/mobile-navigation/". – Abgerufen: 16.05.22.

[DeBeasi, 2020] DeBeasi, L. (2020). Announcing Ionic Vue. "https://ionicframework.com/blog/announcing-ionic-vue". – Abgerufen: 25.03.2022.

[Devathon Team, 2020] Devathon Team (2020). 10 Best PWA Frameworks and Tools in 2021. "https://devathon.com/blog/top-10-best-pwa-frameworks/". – Abgerufen: 15.03.22.

[Deveria, 2022] Deveria, A. (2022). Can I use? "https://caniuse.com". – Abgerufen: 05.05.22.

[Diekmann and Eggert, 2021] Diekmann, J. and Eggert, M. (2021). Is a Progressive Web App an Alternative for Native App Development? In *3. Wissenschaftsforum:Digitale Transformation(WiFo21)*. Gesellschaft für Informatik, "https://dl.gi.de/bitstream/handle/20.500.12116/38623/P_319_komplett.pdf?sequence=1&isAllowed=y#page=36". – Abgerufen: 10.03.22.

[Digital Guide Ionos, 2018] Digital Guide Ionos (2018). Grafische Benutzeroberfläche: Was macht ein gutes User- Interface aus? "https://www.ionos.de/digitalguide/websites/webdesign/grafische-benutzeroberflaeche-alles-fuer-ein-gutes-ui". – Abgerufen: 07.04.2022.

[Digital Guide Ionos, 2019] Digital Guide Ionos (2019). Progressive Web-Apps: Was versprechen die progressiven Apps? "https://www.ionos.de/digitalguide/websites/web-entwicklung/progressive-web-apps-welche-vorteile-bieten-sie". – Abgerufen: 04.05.22.

[Digital Guide Ionos, 2020] Digital Guide Ionos (2020). Web Components erklärt. "https://www.ionos.de/digitalguide/websites/web-entwicklung/web-components/". – Abgerufen: 12.03.22.

[Digital Guide Ionos, 2021a] Digital Guide Ionos (2021a). Was ist ein Crawler: Wie die Datenspinnen das Internet optimieren. "https://www.ionos.de/digitalguide/online-marketing/suchmaschinenmarketing/was-ist-ein-crawler/". – Abgerufen: 10.07.22.

[Digital Guide Ionos, 2021b] Digital Guide Ionos (2021b). Was ist Google Lighthouse? "https://www.ionos.de/digitalguide/online-marketing/suchmaschinenmarketing/google-lighthouse/". – Abgerufen: 10.07.22.

[divante, 2019] divante (2019). The history of PWA development. "https://www.divante.com/pwabook/chapter/02-the-history-of-pwas". – Abgerufen: 05.05.22.

[Erik D. Kennedy, 2021] Erik D. Kennedy (2021). IOS vs. Android App UI Design: The Complete Guide. "https://learnui.design/blog/ios-vs-android-app-ui-design-complete-guide.html#primary-nav". – Abgerufen: 17.05.22.

[Facebook, 2022] Facebook (2022). Create React App. "https://create-react-app.dev". – Abgerufen: 21.03.22.

[Faizunnabi, 2020] Faizunnabi, S. (2020). How to Get HSV Values From RGB Colorspace and vice-versa In Python. "https://buzzneers.com/how-to-guides/how-to-get-hsv-values-from-rgb-colorspace-and-vice-versa-in-python/". – Abgerufen: 15.07.22.

[Goktürk, 2015] Goktürk, M. (2015). 37. Fitts's Law. "https://www.interaction-design.org/literature/book/the-glossary-of-human-computer-interaction/fitts-s-law". – Abgerufen: 07.05.22.

[Google, 2022] Google (2022). Meet Material Design 3. "https://m3.material.io/components/navigation-bar/guidelines". – Abgerufen: 18.05.22.

[Grimm, 2020] Grimm, S. (2020). How to Create an Ionic PWA with Web Push Notifications. "https://devdactic.com/ionic-pwa-web-push/". – Abgerufen: 20.06.22.

[Großkortenhaus, 2020] Großkortenhaus, M. (2020). Was ist ein Wireframe? "https://blog.hubspot.de/website/wireframe". – Abgerufen: 25.05.22.

[Gvak and Mistry, 2022] Gvak, N. and Mistry, V. (2022). Powerful apps fueled by the web: How developers are engaging an expanding ChromeOS audience. "https://chromeos.dev/en/posts/powerful-apps-fueled-by-the-web#fn1". – Abgerufen: 10.07.22.

[Hendrick, 2022] Hendrick, T. (2022). *UX/UI Design Complete Guide 2022*. Tabina Hendrick.

[Hoober, 2017] Hoober, S. (2017). Design for Fingers, Touch, and People, Part 1. "https://www.uxmatters.com/mt/archives/2017/03/design-for-fingers-touch-and-people-part-1.php". – Abgerufen: 17.05.22.

[Ionic, 2022a] Ionic (2022a). Ionic Vue Overview. "https://ionicframework.com/docs/vue/overview". – Abgerufen: 30.03.2022.

[Ionic, 2022b] Ionic (2022b). One codebase. Any platform. "https://ionicframework.com". – Abgerufen: 05.04.2022.

[Ionic, 2022c] Ionic (2022c). One Codebase Any Platform Just React. "https://ionicframework.com/docs/react". – Abgerufen: 01.04.2022.

[ionic-team, 2022] ionic-team (2022). @ionic/vue. "https://www.npmjs.com/package/@ionic/vue". – Abgerufen: 06.04.22.

[Jensen, 2020] Jensen, C. (2020). Learn from the best: Mobile Design Principles. "https://uxdesign.cc/learn-from-the-best-mobile-design-principles-f1bdc46bc016". – Abgerufen: 17.05.22.

[Johnson, 2022] Johnson, E. A. (2022). Testing a Page in Ionic 6. "https://www.eaj.io/articles/testing-a-page-in-ionic/". – Abgerufen: 05.04.22.

[Klose, 2021] Klose, A.-C. (2021). Firefox beendet PWA-Support für den Desktop. "https://entwickler.de/javascript/firefox-beendet-pwa-support-fur-den-desktop". – Abgerufen: 05.05.22.

[Kotthoff, 2020] Kotthoff, K. (2020). Die Google Lighthouse Serie Teil 1: Der Lighthouse Score. "https://www.appmatics.de/blog/google-lighthouse-score/". – Abgerufen: 10.07.22.

[Kugler, 2022] Kugler, K. (2022). The Pros and Cons of React JS. "https://www.linkedin.com/pulse/pros-cons-react-js-kip-kugler". – Abgerufen: 22.03.22.

[Kulkarni, 2021] Kulkarni, A. (2021). Web API vs REST API Simplified: 4 Critical Differences. "https://hevodata.com/learn/web-api-vs-rest-api/#web". – Abgerufen: 29.04.22.

[Lay, 2019] Lay, H. T. (2019). Developing A Web Application With ReactJS. Master's thesis, California State Polytechnic University, Pomona, "https://scholarworks.calstate.edu/downloads/6w924d809". – Abgerufen: 18.03.22.

[Lundberg, 2020] Lundberg, A. (2020). Die Bedeutung der Farben und die Kunst, Farbsymbolik anzuwenden. "https://99designs.de/blog/design-tipps/bedeutung-der-farben/". – Abgerufen: 09.05.22.

[Luntovskyy and Gütter, 2020] Luntovskyy, A. and Gütter, D. (2020). *Moderne Rechnernetze*. Springer Vieweg.

[Lynch, 2022] Lynch, M. (2022). Introducing the Ionic End-to-End Testing Reference Example. "https://ionicframework.com/blog/introducing-the-ionic-end-to-end-testing-reference-example/". – Abgerufen: 07.04.22.

[Marcak, 2021] Marcak, M. (2021). Cross-Platform Apps vs Native Apps vs Progressive Web Apps. "https://www.businessofapps.com/insights/cross-platform-apps-vs-native-apps-vs-progressive-web-apps/". – Abgerufen: 04.07.22.

[MDN Web Docs, 2022a] MDN Web Docs (2022a). Introduction to progressive web apps. "https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Introduction#what_is_a_progressive_web_app". – Abgerufen: 04.05.22.

[MDN Web Docs, 2022b] MDN Web Docs (2022b). Making PWAs work offline with Service workers. "https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Offline_Service_workers". – Abgerufen: 04.05.22.

[MDN Web Docs, 2022c] MDN Web Docs (2022c). Using Service Workers. "https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers". – Abgerufen: 04.05.22.

[Meta Platforms, 2021] Meta Platforms (2021). Introducing Meta: A Social Technology Company. "https://about.fb.com/news/2021/10/facebook-company-is-now-meta". – Abgerufen: 07.04.2022.

[Meta Platforms, 2022] Meta Platforms (2022). React. "https://reactjs.org". – Abgerufen: 03.04.2022.

[Microsoft, 2021] Microsoft (2021). Internet Explorer 11 desktop application ending support for certain operating systems. "https://docs.microsoft.com/en-

us/lifecycle/announcements/internet-explorer-11-end-of-support". – Abgerufen: 05.05.22.

[Microsoft, 2022] Microsoft (2022). Overview of Progressive Web Apps (PWAs). "https://docs.microsoft.com/en-us/microsoft-edge/progressive-web-apps-chromium". – Abgerufen: 04.05.22.

[Monterail and Vue.js, 2021] Monterail and Vue.js (2021). State of Vue.js 2021 - How is Vue doing in the JS Landscape? Technical report, Vue.js, "https://www.monterail.com/hubfs/state_of_vue_2021/State_of_vue.js_2021_report_by_Monterail.pdf". – Abgerufen: 08.04.2022.

[Mulligan, 2021] Mulligan, T. (2021). *UX/UI Design 2021-2022 Tutorial for Beginners: The Complete Step by Step Guide to UX/UI Design and Best Practices for designers with no Experience (English Edition)*. Tom Mulligan.

[npmtrends, 2022] npmtrends (2022). @ionic/core vs @ionic/react vs @ionic/vue vs react vs vue. "https://www.npmtrends.com/@ionic/core-vs-@ionic/react-vs-@ionic/vue-vs-react-vs-vue". – Abgerufen: 11.04.22.

[Onsen UI, 2022] Onsen UI (2022). The most beautiful and efficient way to develop HTML5 hybrid and mobile web apps. "https://onsen.io/". – Abgerufen: 28.03.22.

[Panzarella, 2021] Panzarella, L. (2021). *Mobile design: How to design responsive websites and mobile apps that work (Manuals for web designers) (English Edition)*. Luca Panzarella.

[Panzarella, 2022] Panzarella, L. (2022). *UI + UX: web design simply explained (Manuals for web designers) (English Edition)*. Luca Panzarella.

[Patel, 2019] Patel, R. (2019). Pros and Cons of the Vue.js Framework. "https://ronakataglowid.medium.com/pros-and-cons-of-the-vue-js-framework-8015dcbc05ef". – Abgerufen: 27.03.22.

[Payne, 2020] Payne, R. (2020). Twitter's Google Play Store app now installs the PWA by default when on a Chromebook. "https://chromeunboxed.com/twitter-pwa-default-google-play-store-chromebooks-chrome-os/". – Abgerufen: 10.07.22.

[Rawat and Mahajan, 2020] Rawat, P. and Mahajan, A. N. (2020). ReactJS: A Modern Web Development Framework. "https://ijisrt.com/assets/upload/files/IJISRT20NOV485.pdf". – Abgerufen: 18.03.22.

[Reshetilo, 2021] Reshetilo, K. (2021). Progressive Web Apps in 2021: Will They Die or Thrive? "https://dzone.com/articles/progressive-web-apps-in-2021-will-they-die-or-thri". – Abgerufen: 10.07.22.

[Russel, 2015] Russel, A. (2015). Progressive Web Apps:Escaping Tabs Without Losing Our Soul. "https://medium.com/\spacefactor\@m{}slightlylate/progressive-apps-escaping-tabs-without-losing-our-soul-3b93a8561955". – Abgerufen: 04.05.22.

[Saks, 2019] Saks, E. (2019). JavaScript frameworks: Angular vs React vs Vue. "https://www.theseus.fi/bitstream/handle/10024/261970/Thesis-Elar-Saks.pdf?sequence=2". – Abgerufen: 18.03.22.

[Salomaa, 2020] Salomaa, J. (2020). Evaluating JavaScript frameworks. "https://www.diva-portal.org/smash/get/diva2:1461878/FULLTEXT01.pdf". – Abgerufen: 02.04.22.

[Sander and Ackermann, 2018] Sander, N. and Ackermann, E. (2018). Progressive Web Apps als Alternative zuWebapplikationen und Ersatz zu nativen Apps. Master's thesis, Universität Paderborn, "https://cs.uni-paderborn.de/fileadmin/informatik/fg/mci/Masterarbeiten/2018/MA_Ackermann_Eugen__Sander__Nicolas.pdf". – Abgerufen: 10.03.22.

[Sheppard, 2017] Sheppard, D. (2017). *Beginning Progressive Web App Development Creating a Native App Experience on the Web*. apress.

[Soegaard, 2021] Soegaard, M. (2021). Hick's Law: Making the choice easier for users. "https://www.interaction-design.org/literature/article/hick-s-law-making-the-choice-easier-for-users". – Abgerufen: 07.05.22.

[Stack Exchange, 2022] Stack Exchange (2022). Stack Overflow. "https://stackoverflow.com". – Abgerufen: 01.04.22.

[Stan, 2020] Stan, A. (2020). Typografie-Design für Anfänger: Ein Guide zu Regeln und Begriffen. "https://99designs.de/blog/design-tipps/typografie-design/". – Abgerufen: 15.05.22.

[StatCounter, 2022] StatCounter (2022). Desktop vs Mobile vs Tablet Market Share Worldwide. "https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet". – Abgerufen: 17.05.22.

[Tandel and Jamadar, 2018] Tandel, S. S. and Jamadar, A. (2018). Impact of Progressive Web Apps on Web App Development. "https://www.researchgate.net/profile/Sayali-Tandel-2/publication/330834334_Impact_of_Progressive_Web_Apps_on_Web_App_Development/links/5c5605d3a6fdccd6b5dde018/Impact-of-Progressive-Web-Apps-on-Web-App-Development.pdf". – Abgerufen: 04.05.22.

[TechTarget and ComuputerWeekly, 2020] TechTarget and ComuputerWeekly (2020). Legacy-Anwendung (Altanwendung). "https://whatis.techtarget.com/de/definition/Legacy-Anwendung-Altanwendung". – Abgerufen: 05.05.22.

[Tuan, 2022] Tuan, J. (2022). How Much Does it Cost to Build an App in 2022. "https://topflightapps.com/ideas/app-development-costs/". – Abgerufen: 04.07.22.

[Vasilev, 2021] Vasilev, A. (2021). Comparison of React components testing patterns. "https://www.theseus.fi/bitstream/handle/10024/510460/Vasilev_Andrei.pdf?sequence=2". – Abgerufen: 22.03.2022.

[Vercel, 2022] Vercel (2022). The React Frameworkfor Production. "https://nextjs.org/". – Abgerufen: 25.03.22.

[Vuejs, 2022a] Vuejs (2022a). The Progressive JavaScript Framework. "https://vuejs.org". – Abgerufen: 04.04.2022.

[Vuejs, 2022b] Vuejs (2022b). Vue Community Guide. "https://vue-community.org/". – Abgerufen: 12.04.22.

[Vuejs, 2022c] Vuejs (2022c). Vue Forum. "https://forum.vuejs.org". – Abgerufen: 12.04.22.

[W3C Web Accessibility Initiative, 2018] W3C Web Accessibility Initiative (2018). Web Content Accessibility Guidelines (WCAG) 2.1. "https://www.w3.org/TR/WCAG21/". – Abgerufen: 12.05.22.

[W3C Web Accessibility Initiative, 2022] W3C Web Accessibility Initiative (2022). Introduction to Web Accessibility. "https://www.w3.org/WAI/fundamentals/accessibility-intro/". – Abgerufen: 08.05.22.

[WebInfoMart, 2019] WebInfoMart (2019). Differences between Ionic, Angular and React – Ionic vs Angular vs React. "https://webinfomart.com/blog/differences-between-ionic-angular-and-react-ionic-vs-angular-vs-react/". – Abgerufen: 31.03.22.

[Williamson, 2020] Williamson, S. (2020). Mobile-First vs. Responsive Design: Pros, Cons, and Best Practices. "https://www.hudsonintegrated.com/blog/mobile-first-vs-responsive-design-pros-cons-and-best-practices-5293". – Abgerufen: 19.05.22.

[Wong, 2021] Wong, E. (2021). Principle of Consistency and Standards in User Interface Design. "https://www.interaction-design.org/literature/article/principle-of-consistency-and-standards-in-user-interface-design". – Abgerufen: 16.05.22.

[Yanes, 2021] Yanes, A. (2021). Unit vs Integration vs E2E Tests. "https://dev.to/betoyanes/unit-vs-integration-vs-e2e-tests-4b2o". – Abgerufen: 15.04.22.

[ZYMR INC., 2020] ZYMR INC. (2020). Successful progressive app examples that will inspire you to join the league. "https://www.businessofapps.com/insights/successful-progressive-app-examples-that-will-inspire-you-to-join-the-league/". – Abgerufen: 10.07.22.

A. JSON-Format eines Datenbankeintrags in PouchDB

Abbildung A.1: Beispiel eines Datenbankeintrags im JSON-Format für PouchDB und CouchDB, Quelle: Eigene Darstellung

B. Farben und ihre Assoziationen

Tabelle B.1: Farben, ihre Assoziationen und Verwendung [Panzarella, 2022, S.147] [Lundberg, 2020]

Farbe	Assoziation	Häufiger Einsatz
Rot	Energie, Gefahr	Mode, Sport, Unterhaltung, Aufmerksamkeit auf etwas lenken
Schwarz	Autorität, Eleganz, Stärke	Luxusartikel, Mode
Weiß	Unschuld, Reinheit	Gesundheitswesen, High-Tech, Wissenschaft
Blau	Sicherheit, Kompetenz, Ruhe	Gesundheitswesen, High-Tech, Wissenschaft
Gelb	Optimismus, Kompetenz	Sparsame Verwendung von hellerem Gelb kann Benutzer animieren Interaktion durchzuführen
Orange	Wärme, Energie, Vorsicht	Nahrungsmittel, Online-Handel, Verwendung zum Handlungsaufruf
Grün	Natur, Gesundheit, Gelassenheit	Tourismus, Umwelt, Nachhaltigkeit, Signalisierung von erfolgreich abgeschlossener Interaktion
Braun	Natur, Stabilität, Zuverlässigkeit	Nahrungsmittel

C. Wireframes und Mockups

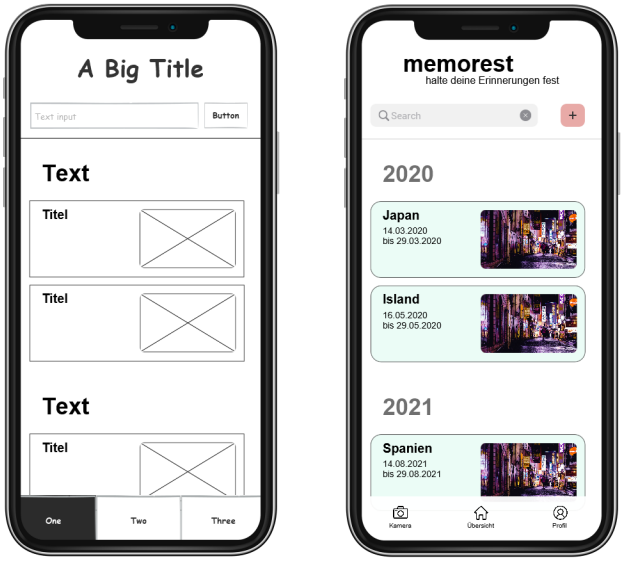

Abbildung C.1: Wireframe (links) und Mockup (rechts) für Benutzeroberfläche auf der IOS Plattform, Quelle: Eigene Darstellung

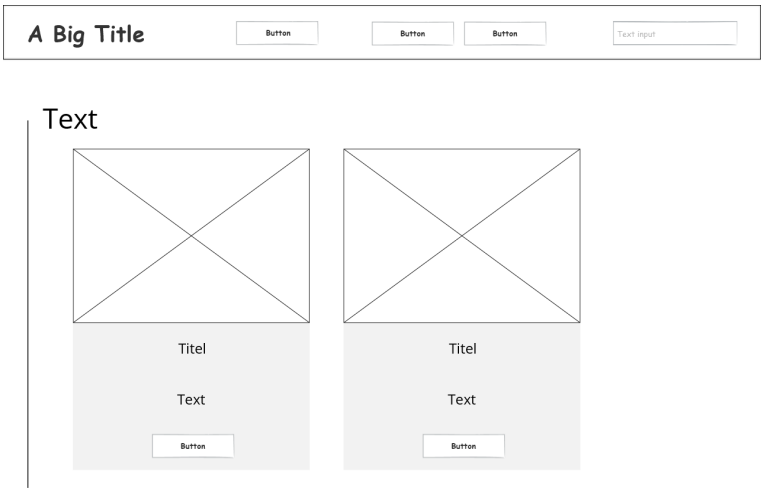

Abbildung C.2: Wireframe für Benutzeroberfläche auf dem Desktop, Quelle: Eigene Darstellung

D. Screenshots zur PWA

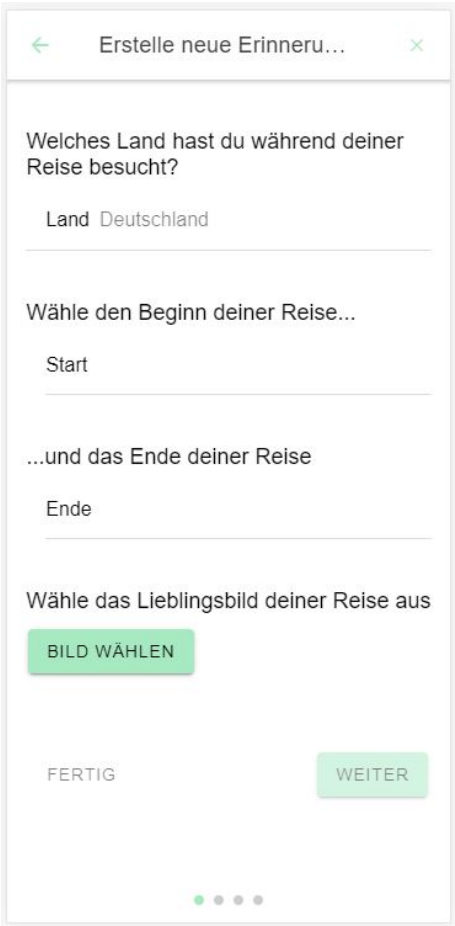

Abbildung D.1: Formular zur Erstellung neuer Einträge auf der Android-Plattform, Quelle: Eigene Darstellung

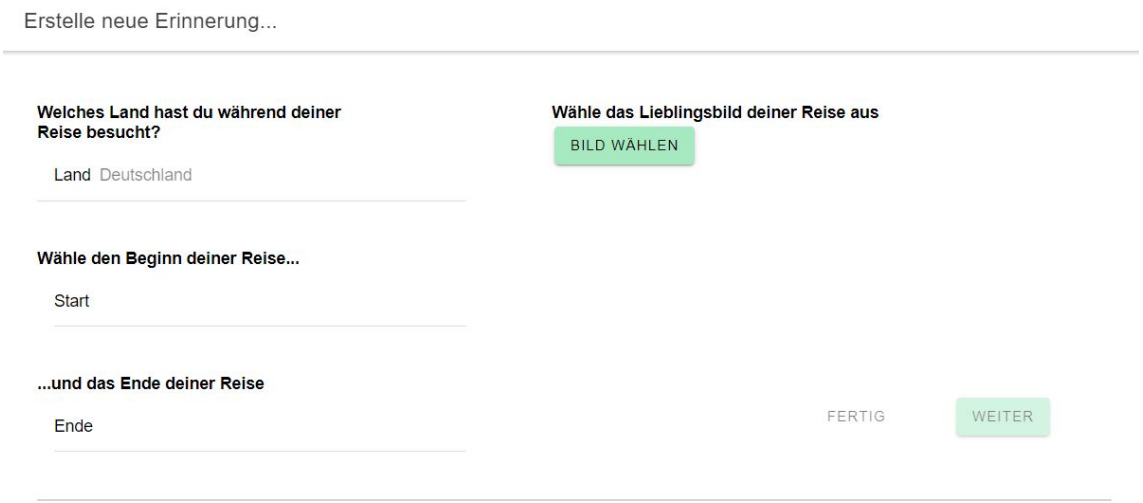

Abbildung D.2: Formular zur Erstellung neuer Einträge auf dem Desktop, Quelle: Eigene Darstellung