HAW HAMBURG

BACHELORTHESIS
Andreas Berks

Konzeption und prototypische Umsetzung einer Progressive-Web-App unter Verwendung von WordPress als Headless Content-Management-System

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Andreas Berks

Konzeption und prototypische Umsetzung einer Progressive-Web-App unter Verwendung von WordPress als Headless Content-Management-System

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Stefan Sarstedt
Zweitgutachter: Prof. Dr. Ulrike Steffens

Eingereicht am: 29. Februar 2020

Andreas Berks

Thema der Arbeit

Konzeption und prototypische Umsetzung einer Progressive-Web-App unter Verwendung von WordPress als Headless Content-Management-System

Stichworte

Progressive-Web-App, Service-Worker, Universelles Rendering, Node.js, TypeScript, WordPress, Headless Content-Management-System, Cloud-Computing

Kurzzusammenfassung

Web-Apps erleichtern durch ihre Plattformunabhängigkeit die Entwicklung mobiler Apps, jedoch auf Kosten der Performanz, der Nutzererfahrung, dem Zugriff auf Gerätefunktionen und der Offline-Fähigkeit. Sogenannte Progressive-Web-Apps (PWAs) versprechen eine Reduzierung dieser Einbußen. Ziel der Arbeit ist die Konzeption und prototypische Umsetzung einer PWA. Der Prototyp soll zeigen, inwieweit sich eine PWA als alternatives Frontend eines WordPress-Blogs eignet.

Andreas Berks

Title of Thesis

Conception and prototypical implementation of a progressive web app using WordPress as a headless content management system

Keywords

Progressive Web App, Service Worker, Universal Rendering, Node.js, TypeScript, WordPress, Headless Content Management System, Cloud Computing

Abstract

Web Apps simplify mobile app development through their platform independence, but at the cost of performance, user experience, access to device features and offline capability. So-called Progressive Web Apps (PWAs) promise to reduce these losses. The objective of this work is the conception and prototypical implementation of a PWA. The prototype is intended to show to what extent a PWA is suitable as an alternative frontend of a WordPress blog.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

In diesem Kapitel werden die Problemstellung und die zu erreichenden Ziele der Arbeit beschrieben. Weiterhin werden die Struktur der Arbeit, die Inhalte der einzelnen Kapitel und die zur Bearbeitung der Arbeit genutzten Werkzeuge dargestellt.

1.1 Motivation

In dem digitalen Zeitalter nimmt die Bedeutung von Smartphones stetig zu. Die meistgenutzte Art von darauf ausgeführten Anwendungen, den sogenannten Apps, sind die nativen Apps. Diese Verbreitung liegt an der optimierten Nutzererfahrung, der hohen Performanz und dem bestmöglichen Zugriff auf die Funktionen des Gerätes.

Der Betriebssystem-Markt teilt sich mit iOS und Android in zwei große Lager. Da die Entwicklung für diese Betriebssysteme in unterschiedlichen Programmiersprachen und unter Einsatz verschiedener Entwicklungswerkzeuge erfolgt, ist bei nativen Apps die Entwicklung und Wartung zweier separater Anwendungen nötig. Betrachtet man darüber hinaus den Desktop-Markt, erhöht sich die Anzahl der separaten Anwendungen zusätzlich. Die Entwicklung nativer Apps stellt daher ein entsprechend aufwendiges Unterfangen dar.

Es existieren viele Entwicklungsansätze, welche sich dieser Problemstellung widmen. Dazu gehören die browserbasierten Web-Apps, welche sich durch eine Plattformunabhängigkeit und die Verwendung ohne vorherige Installation auszeichnen. Beide Vorteile sind jedoch nur auf Kosten der Performanz, Nutzererfahrung und nutzbaren Gerätefunktionen möglich. Durch die Abhängigkeit von einer Netzwerkverbindung besitzen Web-Apps zusätzlich den Nachteil, nicht offline-fähig zu sein.

Bereits in den späten 2000er-Jahren entstanden daher diverse Entwicklungsansätze und Technologien mit dem Ziel, die Vorteile beider Entwicklungsansätze zu kombinieren. Diese orientieren sich unterschiedlich stark an einem der beiden genannten Entwicklungsansätze. Nachteile dieser Entwicklungsansätze sind wie auch bei nativen Apps die notwendige

Installation der App und teilweise auch die vergleichsweise umständlichen Updates über den App-Store der jeweiligen Plattform.

Im Jahr 2015 wurden die Progressive-Web-Apps als eine spezielle Art von Web-Apps vorgestellt. Eine Progressive-Web-App (kurz PWA) ist eine erweiterte Web-App, welche Funktionalitäten bereitstellt, die bisher den anderen Entwicklungsansätzen vorbehalten waren. Dazu gehört die Offline-Fähigkeit, eine (optionale) Installierbarkeit und das Anzeigen von Push-Benachrichtigungen. Trotz alledem besitzen Progressive-Web-Apps alle Vorteile einer Web-App, wie die Plattformunabhängigkeit, den schnellen Zugriff über den Web-Browser und eine Aktualisierung der App, ohne dass der Benutzer Updates über den App-Store der Plattform durchführen muss.

Nicht alle Browser stellen die Web-APIs bereit, welche für eine vollumfängliche PWA erforderlich sind. Daher nutzen PWAs das Prinzip der progressiven Verbesserung, wonach der Funktionsumfang der App mit den Fähigkeiten des Browsers wächst. Der grundlegende Funktionsumfang soll dabei für alle Browser gegeben sein, auch für solche, die nur wenig Funktionalitäten zur Verfügung stellen.

Web-Apps wie die Progressive-Web-Apps lassen eine Auffindbarkeit durch Suchmaschinen zu. Insbesondere für Anwendungen, deren primäres Ziel die Bereitstellung von Inhalten ist, bietet dies einen großen Mehrwert. Ein klassisches Beispiel solcher Anwendungen sind Web-Logs (kurz „Blogs“).

Zur Verwaltung der Inhalte eines solchen Blogs ist ein Content-Management-System nahezu notwendig. Marktführer unter solchen Systemen ist WordPress, ein System, welches mittlerweile über 30 % des Internets mit Inhalten versorgt. Content-Management-Systeme stellen neben dem Backend auch ein Frontend bereit.

Anpassungen des Frontends sind nur in der Programmiersprache PHP möglich, auf welcher WordPress basiert. Die Nutzung moderner Frontend-Technologien wie React, Angular oder Vue ist hingegen nur in einem JavaScript-basierten Frontend möglich. Durch die REST API von WordPress besteht jedoch die Möglichkeit, das Frontend vollständig von dem Backend zu entkoppeln und damit auch zu ersetzen. Durch ein solches „Headless CMS“ kann die größtmögliche Flexibilität in Bezug auf die Technologie-Auswahl für das Frontend geboten werden. Auch eine Skalierung, welche in einem monolithischen WordPress-System erschwert wird, ist durch die Entkopplung bedeutend einfacher.

1.2 Zielsetzung

Ziel der Arbeit ist die Konzeption und prototypische Umsetzung eines Systems, dessen Kern eine Progressive-Web-App darstellt. Das System soll das Frontend eines bestehenden Blogs, welcher Inhalte durch WordPress bereitstellt, vollständig ersetzen. Die Arbeit soll zeigen, inwieweit sich eine Progressive-Web-App für diesen Anwendungsfall eignet, sowie auf die Weiterentwicklung und Einführung in einer Produktionsumgebung vorbereiten.

Die Anforderungen an das System werden anhand des fiktiven Reiseblogs airact erhoben und spezifiziert. airact stellt einen mehrsprachigen WordPress-Blog dar, realisiert durch eine WordPress-Multisite[1]. Die Leserbasis von airact besteht sowohl aus registrierten als auch aus unregistrierten Lesern. Inhaltlich beschäftigt sich airact mit Reise-News und Reise-Deals.

1.3 Aufbau der Arbeit

In Kapitel 1 wird das Thema eingeleitet und auf die Ziele der Arbeit eingegangen. Daraufhin werden in Kapitel 2 die notwendigen theoretischen Grundlagen vermittelt. Kapitel 3 stellt die erhobenen Anforderungen an das System dar, welche in Kapitel 4 näher spezifiziert werden. In Kapitel 5 wird die Architektur des Systems vorgestellt. Die Umsetzung des Systems wird darauffolgend in Kapitel 6 beschrieben. Kapitel 7 beschreibt den Test- und Integrationsprozess. Das entwickelte System wird in Kapitel 8 in Hinblick auf die Anforderungen evaluiert. Abschließend wird in Kapitel 9 ein Fazit gezogen und auf weiterführende Themen eingegangen.

1.4 Zur Bearbeitung der Arbeit genutzte Werkzeuge

- Latex-Editor: `https://www.xm1math.net/texmaker/`
- Literaturverwaltung: `https://www.jabref.org/`
- Erstellung von Diagrammen: `https://www.draw.io/`

[1] *WordPress-Multisites* sind ein Feature von WordPress, welches erlaubt, mehrere WordPress-Websites in einer WordPress-Installation zu betreiben. Die einzelnen WordPress-Websites können jeweils in einer unterschiedlichen Sprache bereitgestellt werden.

- Erstellung von Wireframes: https://www.lucidchart.com/
- Zuschneiden von PDFs: https://www.sejda.com/de/crop-pdf
- Umwandlung von SVG-Grafiken zu PDF: https://cloudconvert.com/svg-to-pdf
- Archivierung von Websites: https://www.web2pdfconvert.com/
- Erstellung von Latex-Tabellen: https://www.tablesgenerator.com/latex_tables
- Website-Archiv: https://web.archive.org/
- Kanban-Board: https://trello.com/

2 Grundlagen

In diesem Kapitel werden die notwendigen theoretischen Grundlagen vermittelt. Dabei wird das Content-Management-System WordPress beschrieben und auf die Entkopplung des Backends und Frontends eingegangen. Anschließend werden verschiedene Arten von Apps betrachtet und mit den sogenannten Progressive-Web-Apps verglichen. Außerdem werden die genutzten Technologien zur Umsetzung des zu entwickelnden Systems beschrieben, wozu die JavaScript-Laufzeitumgebung Node.js, die Container-Plattform Docker und das Container-Verwaltungssystem Kubernetes gehört.

2.1 WordPress

WordPress[1] startete im Jahr 2003, wurde entwickelt von Matt Mullenweg sowie Mike Little und basiert auf dem Blogging-System b2/cafelog[2] (vgl. WordPress Foundation (o. J.e)). Mit der Version 3.0 im Jahr 2010 hat sich WordPress zu einem vollwertigen *Content-Management-System* entwickelt (vgl. WordPress Foundation (2012)).

Content-Management-Systeme (CMS) dienen der Erstellung und Verwaltung von Inhalten einer *Website* (vgl. Q-Success (2019b)). Eine Form von Websites, in welchen CMS Anwendung finden, sind *Blogs* (vgl. Cabot (2018)).

Seit vielen Jahren ist WordPress Marktführer unter den CMS (vgl. Q-Success (2019a)). Im November 2019 besaß WordPress einen Marktanteil von 61,6 %, womit 34,7 % aller Websites WordPress nutzten (vgl. Q-Success (2019b)).

Entwickelt wird WordPress als Open-Source-Projekt in der Programmiersprache PHP (vgl. WordPress Foundation (o. J.t)). WordPress ist unter der GPL (GPLv2 or later) lizenziert (vgl. WordPress Foundation (o. J.g)).

[1] https://wordpress.org/
[2] https://cafelog.com/

WordPress bietet eine Administrationsoberfläche zur Verwaltung der Website sowie deren Inhalte und Benutzer. Die Startseite dieses *(Admin-)Backends* ist das *Dashboard*, welches allgemeine Informationen und Neuigkeiten der Website darstellt. (vgl. Steyer (2016), S. 94f; Abbildung 2.1)

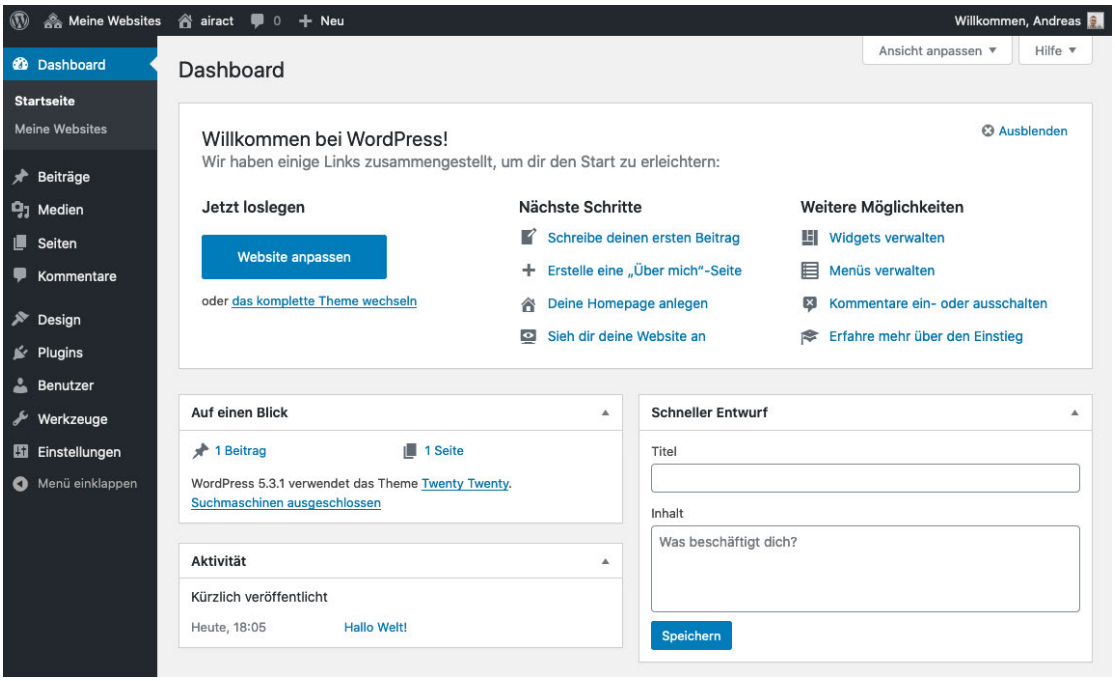

Abbildung 2.1: Dashboard des WordPress-Admin-Backends
(Quelle: Screenshot einer WordPress-Installation)

Mit der Version 5.0 im Jahr 2018 wurde Gutenberg als neuer voreingestellter *Editor* eingeführt. Gutenberg organisiert Inhalte in *Blöcken*, wie etwa Überschriften, Absätze, Bilder, Videos und Listen. Der Editor basiert auf JavaScript und nutzt u. a. die WordPress REST API (vgl. Unterabschnitt 2.1.4). (vgl. WordPress Foundation (o. J.f))

2.1.1 Fachlicher Kontext

Die Inhalte der Website werden in Form von *Beiträgen* und *Seiten* veröffentlicht. Beiträge sind regelmäßig erscheinende Inhalte, welche insbesondere kurz nach der Veröffentlichung relevant sind und daher in umgekehrt chronologischer Reihenfolge aufgelistet werden. Seiten (wie das Impressum oder die „Über uns"-Seite) hingegen stellen langfristig relevante Inhalte dar. Neue Seiten erscheinen im Gegensatz zu neuen Beiträgen eher selten. (vgl. WordPress Foundation (o. J.l))

Beiträge und Seiten werden durch *Taxonomien* strukturiert. Beiträge besitzen standardmäßig die Taxonomien *Kategorie* und *Schlagwort* (vgl. WordPress Foundation (o. J.q)). Seiten können ebenfalls mit Schlagworten versehen werden, die Zuweisung einer Kategorie ist hingegen nicht möglich (vgl. WordPress Foundation (o. J.c)). Unabhängig von Taxonomien können Seiten jedoch baumartig strukturiert werden (beispielsweise kann das Impressum eine Unterseite „Disclaimer“ besitzen) (vgl. WordPress Foundation (o. J.l)).

Ein weiteres Konstrukt zur Strukturierung von Inhalten sind *Menüs*. Ein Menü-Eintrag verlinkt dabei auf einen Beitrag, eine Seite, eine Kategorie oder auf externe Inhalte. (vgl. WordPress Foundation (o. J.b))

Beiträgen und Seiten können *Medien* angehängt werden, wie etwa ein Titelbild. Inhalte einer WordPress-Website besitzen weiterhin *Revisionen*, womit alte oder gerade editierte Versionen der Seite bzw. des Beitrags bezeichnet werden. Diese machen u. a., das Rollback zu einer vorherigen Version möglich. (vgl. WordPress Foundation (o. J.n))

WordPress bietet neben der Inhaltsverwaltung auch eine Benutzerverwaltung. Ein *Benutzer* besitzt dabei u. a. eine Benutzerrolle wie Abonnent, Autor oder Administrator (vgl. WordPress Foundation (o. J.p)). Benutzer einer WordPress-Website können Beiträge und Seiten kommentieren (vgl. WordPress Foundation (o. J.d)).

Es gibt verschiedene Möglichkeiten in WordPress, um Mehrsprachigkeit zu realisieren. Eine davon ist das Feature *WordPress-Multisite*, welches mehrere WordPress-Websites zu einem *Netzwerk* verbindet. Die einzelnen Websites sind dabei autonom und können untereinander verschiedene Sprachen besitzen. (vgl. WordPress Foundation (o. J.j))

Die Benutzer werden durch das Netzwerk verwaltet, müssen jedoch den einzelnen Websites zugewiesen werden. Benutzer können dabei auch mehreren Websites des Netzwerks zugewiesen werden. (vgl. WordPress Foundation (o. J.k))

2.1.2 Aufbau einer WordPress-Installation

Eine WordPress-Installation besteht aus drei wesentlichen Komponenten: dem *WordPress Core*, einem *Theme* und einer Menge von *Plugins*. Während Änderungen am WordPress Core strikt untersagt sind, können Plugins und Themes in der Programmiersprache PHP selbst entwickelt oder von Drittanbietern beschafft werden. (vgl. WordPress Foundation (o. J.i))

Zur Anzeige der Website-Inhalte wird das gerade aktive Theme genutzt, welches das Design und Layout des Frontends definiert. Mit Plugins hingegen kann das Verhalten der Website kontrolliert und neue Funktionen ergänzt werden. (vgl. WordPress Foundation (o. J.s))

WordPress bietet mit der Plugin API[3] die Möglichkeit eigene Plugins zu entwickeln. Diese stellt *Hooks* bereit, mit denen sich in den WordPress Core „eingehackt" werden kann. Hooks werden unterteilt in *Actions* und *Filter*. Einige wenige Hooks sind dabei für das Theme vorgesehen (vgl. WordPress Foundation (o. J.r)). Neben Hooks existieren durch Plugins überschreibbare Funktionen, die sogenannten *Pluggable Functions*. (vgl. WordPress Foundation (o. J.m))

2.1.3 Headless CMS

Traditionelle CMS besitzen eine *monolithische Architektur*, in der das Frontend eng mit dem Backend gekoppelt ist. Dadurch sind Frontend-Entwickler an die Programmiersprache und das Template-System des CMS gebunden. Durch eine Entkopplung von Frontend und Backend sind die eingesetzten Frontend-Technologien hingegen frei wählbar. So können beispielsweise JavaScript-basierte Frontend-Technologien genutzt werden. (vgl. Willmot und Hoyle (2018))

In den letzten Jahren wurden zwei Architektur-Ansätze ausgearbeitet, welche eine solche Entkopplung zum Ziel haben: das *Decoupled CMS* und das *Headless CMS*. Beide Architekturen teilen das Frontend und Backend in separate Teilsysteme, welche durch eine API verbunden werden. Das Backend ist dabei für die Speicherung und Verwaltung der Inhalte zuständig. (vgl. Perfect Sense (2018))

In einem *Decoupled CMS* existiert ein vordefiniertes Frontend. Durch dieses wird der Inhalt für die Darstellung vorbereitet und dann auf verschiedenen Kanälen bereitgestellt. Es handelt sich hierbei um ein proaktives Vorgehen, bei welchem der Inhalt „gepusht" wird. (vgl. Perfect Sense (2018))

Im Gegensatz dazu besitzt ein *Headless CMS* kein definiertes Frontend, sondern stellt den Inhalt lediglich über die API bereit. Die externe Frontend-Anwendung „pullt" den Inhalt von dem Backend, sobald dieser vom Benutzer angefragt wird. Dieses Vorgehen wird auch als reaktiv bezeichnet. (vgl. Perfect Sense (2018))

[3] https://codex.wordpress.org/Plugin_API

Durch das fehlende Frontend ist eine akkurate Live-Vorschau von neuen oder veränderten Inhalten schwerer umzusetzen (vgl. Perfect Sense (2018)). Weiterhin bedeutet die Entkopplung, dass Nutzungsstatistiken (wie Standortdaten) nicht automatisch durch das Backend erfasst werden können (vgl. Klahold und Fathi (2020), S. 43). Hierzu muss entweder eine bidirektionale Kommunikation mit dem Backend erfolgen (vgl. Klahold und Fathi (2020), S. 43) oder es muss auf externe Lösungen wie Google Analytics[4] zurückgegriffen werden.

Ein Headless CMS bietet dafür jedoch die größtmögliche Flexibilität in Bezug auf die Technologie-Auswahl (vgl. Perfect Sense (2018)). Damit lässt sich ein Headless CMS besser in erweiterte Architekturen und Arbeitsabläufe integrieren als ein traditionelles CMS (vgl. Willmot und Hoyle (2018)).

Auch die generellen Vorteile einer Entkopplung sind gegeben. Dazu gehören bessere Wiederverwendbarkeit, bessere Wartbarkeit und eine bessere Verständlichkeit sowie bessere Testbarkeit und bessere Skalierbarkeit (vgl. Gamma u. a. (1995), S. 24f; Danylko (2016)).

2.1.4 WordPress REST API

Ursprünglich als Plugin entwickelt, wurde im Jahr 2015 mit der Version 4.4 die Infrastruktur der WordPress REST API[5] in den WordPress Core integriert (vgl. Mullenweg (2015)). Im Jahr 2016 wurden dann mit der Version 4.7 Endpunkte bereitgestellt, welche Daten der Website als JSON-Objekte zurückliefern, darunter Beiträge, Seiten, Taxonomien und Benutzer (vgl. Hou-Sandi (2016)). Die WordPress REST API kann durch Plugins um neue Endpunkte ergänzt werden (vgl. WordPress Foundation (o. J.a)).

Nicht zu verwechseln ist die WordPress REST API mit der WordPress.com REST API[6]. Diese wird seit dem Jahr 2015 für das WordPress.com Frontend genutzt, welches auch unter dem Namen „Calypso“ bekannt ist (vgl. WordPress Foundation (2015)). Die Entwickler von WordPress sind nach eigenen Angaben bestrebt beide REST APIs anzugleichen (vgl. WordPress Foundation (o. J.h)).

[4] https://analytics.google.com/analytics/web/
[5] https://developer.wordpress.org/rest-api/reference/
[6] https://developer.wordpress.com/docs/api/

2.2 Arten von Apps

Im Jahr 2019 verbrachten Menschen in den USA 70 % ihrer Nutzungszeit digitaler Medien mit der Nutzung eines Smartphones. Dabei dominierten installierte Apps mit einem Nutzungsanteil von 89 % gegenüber der Nutzung des mobilen Webs. Auch weltweit zeigen sich ähnliche Nutzungsanteile. (vgl. Comscore, Inc. (2019))

Der Markt mobiler Betriebssysteme wird von den Betriebssystemen Android und iOS dominiert (vgl. StatCounter (2019c)). Um Apps für eine Masse an Kunden bereitzustellen, sollten beide Betriebssysteme adressiert werden. Da die Betriebssysteme unterschiedliche Programmiersprachen zur Entwicklung von Apps nutzen, haben sich verschiedene Ansätze zur *plattformübergreifenden Entwicklung* gebildet. (vgl. Nunkesser (2018))

Mit einem Nutzungsanteil von 23 % im Jahr 2019 sind auch Desktop-Geräte und Laptops ein wichtiger Teil des Marktes, sowohl in den USA als auch weltweit (vgl. Comscore, Inc. (2019)). Hier dominieren die Betriebssysteme Windows und macOS (vgl. StatCounter (2019b)).

Im Folgenden werden verschiedene Arten mobiler Apps und Desktop-Apps erläutert, darunter auch die *Progressive-Web-Apps*. Die Arten sind aus verschiedenen Entwicklungsansätzen entstanden, wobei die meisten davon plattformübergreifend sind.

2.2.1 Bisherige Arten von Apps

Bereits im Jahr 2007 hat Apple die Möglichkeit vorgestellt, browserbasierte Apps für das iPhone zu entwickeln und diese als *Web-(2.0-)Apps* bezeichnet (vgl. Apple Inc. (2007)). Noch im selben Jahr kündigte Steve Jobs an, dass zukünftig eine Entwicklung *nativer Apps* mithilfe des iPhone SDKs möglich sein wird (vgl. Jobs (2007)). Dieses SDK wurde im Jahr 2008 erstmalig veröffentlicht (vgl. Apple Inc. (2008)).

Im Jahr 2009 wurde in Barney (2009) die Entwicklung von *hybriden Apps* mithilfe von Webtechnologien und dem Framework PhoneGap[7] beschrieben. Hybride Apps stellen dabei eine Zwischenform von nativen Apps und Web-Apps dar.

Inzwischen sind viele Frameworks für die plattformübergreifende (engl. cross-platform) Entwicklung entstanden, welche sich in keine der Kategorien Web-Apps, nativer Apps oder hybrider Apps einordnen lassen (vgl. Nunkesser (2018)). In der Vergangenheit

[7] https://phonegap.com/

wurden in vielen Arbeiten Vorschläge vorgestellt, wie diese Frameworks und die dazugehörigen Arten von Apps kategorisiert werden können. So analysieren Nunkesser (2018) und El-Kassas u. a. (2017) beispielsweise verschiedene Vorschläge und stellen auch eigene Vorschläge vor.

In dieser Arbeit wird aus Gründen der Übersichtlichkeit nur eine zusätzliche Kategorie für Arten von Apps vorgestellt, die nicht in die Kategorien Web-Apps, nativer Apps oder hybrider Apps passen. Diese Kategorie wird im Folgenden als *weitere Cross-Platform-Apps* bezeichnet.

2.2.1.1 Native Apps

Native Apps werden mithilfe von plattformspezifischen *SDKs* entwickelt, welche Werkzeuge und APIs für die Entwicklung bereitstellen. Mithilfe dieser APIs kann auf die *Funktionen* (wie Bluetooth, Kamera oder Standort) und *UI-Komponenten* der Plattform zugegriffen werden. Die Entwicklung der nativen App erfolgt dabei in plattformspezifischen Programmiersprachen, welche in Tabelle 2.1 gezeigt werden. (vgl. IBM Corporation (2012))

Native Apps können in dem *App-Store* der Plattform heruntergeladen und installiert werden (vgl. IBM Corporation (2012)). App-Stores bieten einen stark verbreiteten Vertriebskanal für Apps, in denen auch eine Monetarisierung der App durch Verkäufe möglich ist (vgl. Serrano u. a. (2013)).

Um eine App in einem App-Store bereitstellen zu können, müssen gewisse Leitlinien erfüllt werden, die im Rahmen eines Reviews durch den App-Store-Betreiber geprüft werden (vgl. Google LLC (o. J.h); Apple Inc. (o. J.a)). Zudem ist eine Anmeldung als Entwickler notwendig, welche teilweise auch kostenpflichtig ist (vgl. Apple Inc. (o. J.b)).

Der Zugriff auf die API erfolgt direkt, wodurch keine *Performanz*-Einbußen entstehen (vgl. Xanthopoulos und Xinogalos (2013)). Sämtliche – d. h. auch kürzlich veröffentlichte – Funktionen der Plattform können in der App genutzt werden (vgl. IBM Corporation (2012)). Durch den direkten Zugriff auf UI-Komponenten (wie Buttons und Eingabefelder) der Plattform ist ein natives und konsistentes *Look-and-feel* gegeben (vgl. Apple Inc. (o. J.c); Xanthopoulos und Xinogalos (2013)). Definiert wird dieses durch sogenannte *Designsprachen*, welche für gängige Plattformen im Anhang A.1 exemplarisch dargestellt werden.

Nachteil einer nativen App ist, dass der Code für eine Plattform nicht für eine andere Plattform genutzt werden kann. So erfordern native Apps für verschiedene Plattformen mehrere separate Codebasen, was zu einem hohen Entwicklungs- und Wartungsaufwand führt. Außerdem werden Kenntnisse in den verschiedenen Programmiersprachen und SDKs benötigt. (vgl. Xanthopoulos und Xinogalos (2013))

	Android	**iOS**	**macOS**	**Windows**
Programmier-sprachen	Kotlin, Java	Swift, Objective-C		C#, Visual Basic, C++ und JavaScript
Designsprache	Material Design	iOS Design	Aqua	Fluent (früher Metro)
Entwicklungs-werkzeuge	Android Studio, Android Platform API[8]	Xcode, diverse APIs[9] (z. B. SwiftUI und Core Services)		Visual Studio, WinRT/UWP API[10]
App-Store	Google Play	iOS App Store	Mac App Store	Microsoft Store

Tabelle 2.1: Übersicht über verschiedene Plattformen für Apps

Beispiele für native Apps sind Firefox, SoundCloud, Pinterest und Trello (vgl. Mybridge (2016); Zealous System (2019)).

2.2.1.2 Web-Apps

Web-Apps werden im *Browser* des Benutzers ausgeführt und mit Web-Technologien wie HTML, CSS und JavaScript entwickelt. Der Benutzer kann mit einer URL auf die Web-App zugreifen, ohne dass eine vorherige Installation notwendig ist. (vgl. Raj und Tolety (2012))

Im Gegensatz zu nativen Apps können Web-Apps nicht in einem App-Store vertrieben werden. Damit gehen einerseits ein vielversprechender Vertriebskanal und anderseits eine Möglichkeit der Monetarisierung verloren. (vgl. Raj und Tolety (2012))

Der Browser selbst ist eine native Anwendung, welche direkt auf die Plattform-APIs zugreifen kann. Jedoch stellt der Browser nur einen begrenzten Teil von APIs zur Verfügung. Viele Gerätefunktionen sind daher nicht oder nur teilweise in einer Web-App nutzbar. (vgl. IBM Corporation (2012))

Web-Apps haben keinen direkten Zugang zu nativen UI-Komponenten, wodurch das native Look-and-feel nicht gegeben ist. Es gibt jedoch zusätzliche Technologien, die versuchen, das native Look-and-feel zu reproduzieren. Dazu gehört beispielsweise das Ionic

[8] https://developer.android.com/reference/
[9] https://developer.apple.com/documentation/
[10] https://docs.microsoft.com/en-us/uwp/api/

Framework[11], welches seit der Version 4.0 unabhängig von weiteren Frameworks genutzt werden kann (vgl. Drifty Co. (2019); Drifty Co. (o. J.)).

Weiterhin sind Web-Apps abhängig von einer guten Netzwerkverbindung und sind nicht offline-fähig. Die Performanz leidet außerdem unter der Zeit, die für den Download der Web-App benötigt wird und durch die Zeit, die der Browser im Anschluss benötigt, um die Web-App darzustellen. (vgl. Xanthopoulos und Xinogalos (2013))

Da die genutzten Web-Technologien weitestgehend standardisiert sind, können Web-Apps unabhängig vom Betriebssystem und Gerät des Benutzers eingesetzt werden (vgl. Raj und Tolety (2012)).

Die Web-App wird durch einen *Web-Server* bereitgestellt. Updates können schnell ausgeliefert werden, da diese ebenfalls auf dem Web-Server erfolgen. Auf dem Gerät des Benutzers muss kein Update-Prozess durchlaufen werden. (vgl. Raj und Tolety (2012))

Websites und Web-Apps weisen viele Parallelen auf, wie etwa die Bereitstellung über einen Web-Server. Auch wenn Websites und Web-Apps Unterschiede zueinander aufweisen, sind die Begriffe nicht klar voneinander abgegrenzt. Während Websites in erster Linie statische Inhalte bereitstellen, sind Web-Apps bedeutend interaktiver und nutzen zudem Gerätefunktionen (vgl. Mozilla Foundation (2018)).

Beispiele für Web-Apps sind Gmail, Google Drive, Facebook und Slack (vgl. Mozilla Foundation (2018)).

2.2.1.3 Hybride Apps

Hybride Apps stellen eine Mischung aus nativen Apps und Web-Apps dar. Zur Entwicklung werden, wie bei Web-Apps Technologien, wie HTML, JavaScript und CSS genutzt. Ausgeführt werden hybride Apps in einer sogenannten *Web-View*, einem nativen Container mit Zugriff auf die *Browser-Engine* der Plattform. Wie auch bei den Web-Apps hat diese Ausführung in der Browser-Engine jedoch negative Auswirkungen auf die Performanz. Hybride Apps werden wie native Apps in einem App-Store vertrieben. (vgl. Raj und Tolety (2012))

[11] https://ionicframework.com/

Frameworks wie PhoneGap[12] stellen JavaScript-Schnittstellen bereit, welche auch als *Bridge* bezeichnet werden. Mit dieser plattformübergreifenden Bridge kann auf native Gerätefunktionen zugegriffen werden. (vgl. IBM Corporation (2012))

Wie bei Web-Apps, ist auch bei hybriden Apps das native Look-and-feel nicht standardmäßig gegeben, kann aber mit speziellen Bibliotheken/Frameworks reproduziert werden (vgl. Xanthopoulos und Xinogalos (2013); Unter-Unterabschnitt 2.2.1.2).

Der Code einer hybriden App kann entweder auf einem Web-Server bereitgestellt oder mit der Anwendung selbst ausgeliefert werden. Wird der Code durch einen Web-Server bereitgestellt, können Updates schneller ausgeliefert werden. Jedoch ist die Anwendung dann nicht mehr offline-fähig. Um Vorteile beider Lösungen zu nutzen, ist jedoch ein kombinierter Ansatz möglich. (vgl. IBM Corporation (2012))

Beispiele für Technologien zur Entwicklung hybrider Apps sind PhoneGap, Cordova[13] und Capacitor[14]. Entsprechende Technologien werden beispielsweise von Pacifica, Sworkit, JustWatch und FanReact genutzt (vgl. The Apache Software Foundation (o. J.)).

2.2.1.4 Weitere Cross-Platform-Apps

Cross-Platform-Apps werden in einer Programmiersprache wie JavaScript oder C# entwickelt und stellen native Apps für die gewünschten Plattformen zur Verfügung. Auch Cross-Platform-Apps werden in einen App-Store vertrieben. (vgl. Raj und Tolety (2012))

Es steht eine Reihe von verschiedenen Frameworks zur Verfügung. Je nach Art der technischen Umsetzung variieren diese jedoch stark in Bezug auf den Grad der Plattformunabhängigkeit, der Performanz und dem Grad des nativen Look-and-feels. Im Folgenden werden die grundlegenden Unterschiede aufgezeigt.

Cross-Platform-Apps können entweder *just-in-time* (z. B. React Native[15] unter Android) oder *ahead-of-time* kompiliert (z. B. Xamarin[16]) bzw. zur Laufzeit *interpretiert* werden (z. B. React Native unter iOS) (vgl. Vishal und Kushwaha (2018); Wenjun (2011)). Die ahead-of-time-Kompilierung ist dabei zur Laufzeit meist schneller als die just-in-time-

[12] https://phonegap.com/
[13] https://cordova.apache.org/
[14] https://capacitor.ionicframework.com/
[15] https://facebook.github.io/react-native/
[16] https://dotnet.microsoft.com/apps/xamarin

Kompilierung, welche wiederum schneller als die Ausführung durch einen Interpreter ist (vgl. Vishal und Kushwaha (2018); AltexSoft (2018)).

Weiterhin unterscheiden sich Cross-Platform-Apps in der Art, wie sie mit der nativen Plattform kommunizieren, um etwa die Benutzeroberfläche (UI) darzustellen. Frameworks wie React Native setzen auf eine *Bridge*, welche es ermöglicht, auf native UI-Elemente zuzugreifen (vgl. Eisenman (2015), S. 9). Andere Frameworks wie Flutter[17] hingegen nutzen eine *Rendering-Engine*, welche in der App selbst ausgeführt wird und anschließend die gerenderte Benutzeroberfläche in nativen Canvas-Elementen[18] darstellt (vgl. Google LLC (2019f)).

Beispiele für Apps dieser Kategorie sind Facebook, Instagram, Google Assistent und Microsoft News (vgl. Google LLC (o. J.j), Warcholinski (o. J.) Microsoft Corporation (o. J.c)).

2.2.2 Progressive-Web-Apps

Der Begriff „Progressive-Web-App“ (PWA) wurde im Jahr 2015 von dem Google-Mitarbeiter Alex Russell vorgestellt. PWAs sind keine gänzlich neue Technologie. Vielmehr handelt es sich um eine Web-App, die nach einer Menge von Strategien und Technologien entwickelt wird. Ziel dabei ist es, Vorteile, die bisher nativen Apps vorbehalten waren, zu nutzen. (vgl. Russell (2015))

Eine PWA besitzt folgende Eigenschaften (vgl. Mozilla Foundation (2019d); Osmani (2015)):

- *Progressiv*: Eine PWA basiert auf dem Prinzip der *progressiven Verbesserung* (vgl. Unter-Unterabschnitt 2.2.2.1) und ist daher unabhängig vom Browser des Benutzers.
- *Responsiv*: Eine PWA basiert auf dem Prinzip des *responsiven Designs* (vgl. Unter-Unterabschnitt 2.2.2.2) und wird somit auf Smartphones, Tablets und Desktops/ Notebooks gleichermaßen übersichtlich und benutzerfreundlich dargestellt.

[17] https://flutter.dev/

[18] *Canvas-Elemente* sind Elemente einer Benutzeroberfläche, welche als „Leinwand“ dienen (vgl. Mozilla Foundation (2019b)). Auf dieser „Leinwand“ können Text, Rastergrafiken, Linien und andere grafische Primitive dargestellt werden (vgl. Google LLC (o. J.c)).

- *Netzwerkunabhängig*: Eine PWA funktioniert auch offline oder bei schlechten Netzwerkverbindungen.
- *App-ähnlich*: Die Navigation und Interaktion der PWA orientieren sich an nativen Apps.
- *Aktuell*: Eine PWA befindet sich immer auf der neusten Version.
- *Sicher*: Eine PWA wird über TLS/HTTPS bereitgestellt.
- *Auffindbar*: Eine PWA kann durch Suchmaschinen indexiert werden.
- *Nutzer-bindend*: Eine PWA stellt Mechanismen wie Push-Benachrichtigungen bereit, um Benutzer an sich zu binden.
- *Installierbar*: Eine PWA kann auf dem Startbildschirm des Gerätes hinzugefügt werden.
- *Verlinkbar*: Eine PWA ist über eine URL verlinkbar.

Beispiele für PWAs sind Instagram, Twitter, Telegram und Uber (vgl. Appscope (o. J.)).

2.2.2.1 Progressive Verbesserung

Progressive Verbesserung (engl. progressive enhancement) ist ein Prinzip, bei dem die Bereitstellung von Inhalten im Mittelpunkt steht. Technologien wie CSS und JavaScript dienen demnach nur der Verbesserung der Darstellung und der Interaktivität, sollen aber nicht notwendig sein. (vgl. Champeon und Finck (2003))

Eine Website nach dem Prinzip der progressiven Verbesserung wird zuerst für Browser mit den geringsten Fähigkeiten entwickelt. Anschließend wird die Website im Rahmen der Fähigkeiten moderner Browser optimiert. (vgl. Champeon und Finck (2003))

Von Bedeutung ist die progressive Verbesserung insbesondere für ältere Browser, welche Technologien wie CSS und JavaScript nur eingeschränkt unterstützen, oder für Benutzer, welche diese deaktiviert haben. (vgl. Gustafson (2008))

2.2.2.2 Responsives Design

Responsives Design bedeutet, dass sich das Design dem jeweiligen Anzeigemedium anpasst. Umgesetzt wird responsives Design durch drei Techniken: (vgl. Marcotte (2010))

- *Fließendes Layout*: Die Größe einzelner Elemente des Layouts richtet sich nach der Größe des Anzeigebereiches. Dafür werden Größenangaben nicht in absoluten Größen wie Pixel angegeben, sondern in relativen Größen wie Prozent (vgl. Abbildung 2.2).

Abbildung 2.2: Fließendes Layout
(Quelle: UX Alpaca (2016), Bearbeitungsvermerk:
extrahiert aus einer GIF-Animation)

- *Fließende Bildgrößen*: Wie auch die Größe der Layout-Elemente sollen sich auch die Bildergrößen nach der Größe des Anzeigebereiches richten. In diesem Zusammenhang wird auch von „flexiblen Bildern“ gesprochen.

- *Media-Queries*: Durch Media-Queries können CSS-Regeln oder ganze Stylesheets an eine Bedingung geknüpft werden. Bedingungen können nach Medien-Typ (etwa Drucker oder Bildschirm) und/oder nach Medien-Eigenschaft (wie die Breite des Anzeigebereiches oder des Gerätes) definiert werden. Media-Queries ermöglichen die Umsetzung adaptiver Layouts (vgl. Gustafson (2010); Abbildung 2.3).

Abbildung 2.3: Adaptives Layout
(Quelle: UX Alpaca (2016), Bearbeitungsvermerk:
extrahiert aus einer GIF-Animation)

2.2.2.3 Eingesetzte Technologien

Umgesetzt werden PWAs in erster Linie durch *Service-Worker* und das *Web-App-Manifest* (vgl. Osmani (2015)).

Service-Worker

Service-Worker stellen einen programmierbaren Proxy dar, mit dem das Verhalten bei Anfragen an das Netzwerk kontrolliert werden kann. Dadurch können etwa gecachte Inhalte bereitgestellt werden, selbst wenn keine Netzwerkverbindung besteht. (vgl. Mozilla Foundation (2019e))

Aus Sicherheitsgründen können Service-Worker nur über HTTPS bereitgestellt werden. Dadurch sollen beispielsweise Man-in-the-Middle-Angriffe verhindert werden. (vgl. Mozilla Foundation (2019e))

Service-Worker sind eine Form der *Web-Worker* (vgl. Mozilla Foundation (2019e)). Web-Worker ermöglichen es, Aufgaben in einem separaten Hintergrund-Thread auszuführen, ohne dass dieser den Haupt-Thread der PWA blockiert. Die Kommunikation mit dem Haupt-Thread der PWA erfolgt über *Nachrichten*. Web-Worker sind unabhängig vom HTML-Dokument. Insbesondere haben sie keinen Zugriff auf das DOM[19]. (vgl. Bidelman (2010))

Der Lebenszyklus eines Service-Workers ist unabhängig von der PWA selbst. Service-Worker werden gestoppt, sobald sie nicht mehr genutzt werden und neu gestartet, sobald sie wieder benötigt werden. Daten, welche zwischen den Neustarts verfügbar sein müssen, können beispielsweise mithilfe der IndexedDB API[20] gespeichert werden. (vgl. Gaunt (o. J.))

Zusätzlich können mit Service-Workern Funktionen wie *Push-Benachrichtigungen* und *Hintergrund-Synchronisation* umgesetzt werden. Dafür werden zusätzliche APIs benötigt, wie die Notifications API[21] oder die Web Background Synchronization API[22]. (vgl. Gaunt (o. J.))

[19] Das *Document Object Model* (DOM) dient als Schnittstelle zwischen dem HTML-Dokument der Website und Programmiersprachen wie JavaScript. Das DOM stellt das HTML-Dokument in einer Baumstruktur dar. DOM-Methoden ermöglichen den Zugriff auf diesen Baum. Mit ihnen können die Struktur und der Inhalt des HTML-Dokuments geändert werden und *Ereignisse* registriert werden. (vgl. Mozilla Foundation (2019c))

[20] https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API

[21] https://notifications.spec.whatwg.org/

[22] https://wicg.github.io/BackgroundSync/spec/

Ein Service-Worker muss zunächst durch die PWA registriert werden. Empfohlen wird dies in den meisten Fällen nach dem Laden des HTML-Dokuments (vgl. Posnick (o. J.)). Im Anschluss kann der Service-Worker auf die folgenden Ereignisse reagieren: (vgl. Gaunt (o. J.))

- `install`: Sobald der Service-Worker registriert wurde, wird ein `install`-Ereignis an den Service-Worker gesendet. Der Service-Worker reagiert darauf und nimmt eine Initialisierung vor, etwa indem dieser statische Assets cacht. Das `install`-Ereignis wird auch bei einem Update des Service-Workers (durch Code-Änderung am Service-Worker) an die neue Version gesendet. Nach der Installation wartet der Service-Worker darauf, aktiviert zu werden.
- `activate`: Sobald die alte Version des Service-Workers (sofern vorhanden) nicht mehr verwendet wird, wird ein `activate`-Ereignis an die neu installierte Version des Service-Workers gesendet. Um die Verwendung des alten Service-Workers zu beenden, muss die PWA einmal verlassen werden, ein Aktualisieren der PWA reicht nicht aus. Innerhalb der Behandlung des `activate`-Ereignisses bereinigt der neue Service-Worker dann beispielsweise den Cache der alten Version.
- `fetch`: Das `fetch`-Ereignis wird verwendet, um das Verhalten bei Anfragen an das Netzwerk zu kontrollieren. Statt die Anfrage an das Netzwerk zu leiten, kann ein Element aus dem Cache zurückgeliefert werden.

Web-App-Manifest

Das Web-App-Manifest ist eine JSON-Datei, welche Informationen über die PWA beinhaltet. Dazu gehören u. a. Name und Icon der PWA sowie Attribute, die das Verhalten und das Design der PWA bestimmen. Eingebunden wird das Web-App-Manifest durch ein Link-Tag im HTML-Dokument. (vgl. Gaunt und Kinlan (o. J.))

```
{
  "short_name": "Maps",
  "name": "Google Maps",
  "icons": [
    {
      "src": "/images/icons-192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
```

```
10      {
11        "src": "/images/icons-512.png",
12        "type": "image/png",
13        "sizes": "512x512"
14      }
15    ],
16    "start_url": "/maps/?source=pwa",
17    "background_color": "#3367D6",
18    "display": "standalone",
19    "scope": "/maps/",
20    "theme_color": "#3367D6"
21  }
```

Listing 1: Beispiel eines Web-App-Manifests (Gaunt und Kinlan (o. J.))

Neben einer Auslieferung der App über HTTPS ist ein gültiges Web-App-Manifest Voraussetzung für eine *installierbare Web-App*. Bei einigen Browsern, wie etwa Chrome, ist zusätzlich ein registrierter Service-Worker notwendig, um eine Web-App installierbar zu machen. (vgl. Mozilla Foundation (2019a))

Die Installation kann entweder manuell durch den Benutzer oder proaktiv durch einen *„Add to Home Screen"-Dialog* geschehen (vgl. Mozilla Foundation (2019a)). So senden Browser ein `beforeinstallprompt`-Ereignis, wenn das Benutzerverhalten bestimmte Heuristiken erfüllt (vgl. LePage (o. J.)). Ein einfaches Beispiel für eine solche Heuristik ist die Anzahl der wöchentlichen Besuche. Auf das `beforeinstallprompt`-Ereignis kann reagiert werden, um den Benutzern eine Installation vorzuschlagen (vgl. LePage (o. J.)).

2.2.2.4 Verifizierung

Um zu prüfen, inwieweit die Anforderungen an eine PWA erfüllt sind, stellt Google eine Checkliste bereit. Einige der Punkte können mit dem Werkzeug Lighthouse[23] von Google getestet werden. (vgl. Google LLC (o. J.g))

Lighthouse ist ein sogenanntes Auditing-Werkzeug[24], welches eine Website in folgenden Kategorien bewertet: *Performanz*, *Barrierefreiheit*, *Best-Practices* (beispielsweise keine

[23] https://developers.google.com/web/tools/lighthouse

[24] Ein *Audit* bezeichnet eine unabhängige Bewertung eines Systems, um sicherzustellen, dass Standards, Leitlinien und Spezifikationen erfüllt werden (vgl. Spillner u. a. (2011), S. 236).

Nutzung veralteter oder unsicherer Technologien), *Suchmaschinenoptimierung* und *Progressive-Web-Apps* (vgl. Google LLC (o. J.f)). Zur Bewertung werden Metriken gemessen und Eigenschaften untersucht. Eine Erklärung der gemessenen Metriken und untersuchten Eigenschaften ist im GitHub-Repository von Lighthouse zu finden (vgl. Google LLC (2019i)).

2.2.2.5 Vergleich mit bisherigen Arten von Apps

Wie eine Web-App unterstützt eine PWA alle Plattformen mit einem installierten Browser. Bei einem Blog, welcher bereits über eine Website bereitgestellt wird, kann eine fließende Umstellung auf eine PWA erfolgen. Außerdem werden auch Plattformen unterstützt, welche keinen App-Store bereitstellen (beispielsweise einige eBook-Reader und Spielekonsolen). Inzwischen besteht auch die Möglichkeit, PWAs in den App-Stores von Android (Google Play) und Windows (Microsoft Store) bereitzustellen (vgl. Google LLC (2020c); Microsoft Corporation (2020)). PWAs werden mithilfe plattformunabhängiger Technologien entwickelt, wodurch nur eine Code-Basis gepflegt werden muss.

Der Zugriff auf Gerätefunktionen ist bei einer PWA stark eingeschränkt. Auch die Performanz schneidet im Vergleich zu anderen Arten von Apps relativ schlecht ab. Abstriche in diesen beiden Aspekten sind allerdings bei einer App für einen Blog vertretbar.

PWAs ergänzen Web-Apps um Funktionen, welche bisher den anderen Arten von Apps vorbehalten waren. Dazu gehört die Installierbarkeit der App und das Empfangen von Push-Benachrichtigungen, mit welchen eine hohe Kundenbindung erreicht werden kann. Auch die Offline-Fähigkeit ist ein sehr wichtiger Aspekt, insbesondere für mobile Apps.

Vor allem Google treibt die Entwicklung von PWAs voran. So sind eine Reihe neuer APIs für Chrome in Entwicklung, mit denen beispielsweise auf das native Dateisystem zugegriffen werden kann oder verhindert wird, dass das Gerät in den Sperrzustand verfällt („Wake Lock“) (vgl. Google LLC (2020b)). Prognosen des Marktforschungsinstituts Gartner sagen aus, dass bis 2020 die Hälfte aller kundenorientierten Apps durch eine PWA ersetzt werden (vgl. Gartner, Inc. (2017), zit. n. MacDonald (2018)).

In Tabelle 2.2 wird dieser Vergleich noch einmal verdeutlicht.

[25] stark abhängig vom gewählten Framework
[26] einige Frameworks erfordern plattformspezifische Anpassungen (vgl. Latif u. a. (2016))
[27] https://developer.mozilla.org/en-US/docs/Web/API

	Native Apps	Web-Apps	Hybride Apps	Cross-Platform-Apps	Progressive-Web-Apps
Unterstützte Plattformen	★★★★☆	★★★★★	★★★☆☆	★★☆☆☆–★★★★☆[25]	★★★★★
Plattform-unabhägigkeit	★☆☆☆☆	★★★★★	★★★★★	★★★★☆[26]	★★★★★
Zugriff auf Geräte-funktionen	★★★★★	★☆☆☆☆	★★★☆☆	★★★★☆	★★☆☆☆
Offline-Fähigkeit	★★★★★	★☆☆☆☆	★★★★★	★★★★★	★★★★★
Performanz	★★★★★	★★☆☆☆	★★★☆☆	★★★★☆	★★☆☆☆
Standardmäßig natives Look-and-feel	★★★★★	★☆☆☆☆	★☆☆☆☆	★★★★☆	★☆☆☆☆
Installation/Zugriff über ...	App-Store	Browser	App-Store	App-Store	Browser, Startbildschirm, (App-Store)
Kundenbindung	★★★★★	★☆☆☆☆	★★★★★	★★★★★	★★★★☆
Größe der Community	★★★★★	★★★★★	★★★☆☆	★★★★☆	★★★★☆
Beispiele für Frameworks/APIs	Android SDK, iOS SDK	Web APIs[27]	PhoneGap, Cordova, Capacitor	React Native, Flutter, Xamarin	Web APIs
Beispiele für Apps	Firefox, SoundCloud, Pinterest, Trello	Amazon, Netflix, Twitch, PayPal	Pacifica, Sworkit, JustWatch, FanReact	Facebook, Instagram, Google Assistent, Microsoft News	Instagram, Twitter, Telegram, Uber

Tabelle 2.2: Vergleich zwischen unterschiedlichen Arten von Apps

2.3 Node.js

Node.js[28] ist eine Laufzeitumgebung, mit welcher JavaScript-Code serverseitig ausgeführt werden kann (vgl. Node.js Foundation (2018a)). Das *ereignisgesteuerte* und *nicht-blockierende* Ein-/Ausgabemodell von Node.js verspricht hohe Performanz und Skalierbarkeit. Insbesondere im Vergleich zu threadbasierten Ein-/Ausgabemodellen soll es leichtgewichtig und effizient sein. (vgl. Node.js Foundation (2018b))

Fundament für Node.js ist die von Google entwickelte V8 JavaScript-Engine[29], welche auch durch Chrome genutzt wird (vgl. Node.js Foundation (2018b)). V8 implementiert den JavaScript-Standard ECMAScript[30] (vgl. Google LLC (o. J.k)). Mit der Nutzung aktueller Versionen von V8 stellt Node.js annähernd[31] alle Features der zehnten Edition (Juni 2019) des ECMAScript-Standards bereit (vgl. OpenJS Foundation (o. J.c); Ecma International (2019b)).

ECMAScript ist eine *objekt-orientierte* und *dynamisch-typisierte* Programmiersprache. Auch wenn ECMAScript nicht klassenbasiert ist, bietet ECMAScript klassenähnliche

[28] https://nodejs.org/
[29] https://v8.dev/
[30] https://www.ecma-international.org/publications/standards/Ecma-262.htm
[31] Eine vollständige Liste aller unterstützen Features ist unter https://node.green/ zu finden.

Abstraktionen. Mit diesen können seit der sechsten Edition von ECMAScript (Juni 2015) auf syntaktischer Ebene *Klassen* deklariert werden. (vgl. Ecma International (2019a))

Alternativ können Node.js-Anwendungen in der Programmiersprache *TypeScript* entwickelt werden. TypeScript ergänzt ECMAScript um eine statische Typisierung und liefert zusätzlich verbesserte Klassen- und Interface-Konstrukte. Mithilfe eines Compilers wird TypeScript in den ECMAScript-Standard kompiliert. (vgl. Microsoft Corporation (o. J.b))

Zur Wiederverwendung von Code können in Node.js *Module* exportiert werden und an anderer Stelle bzw. in anderen Anwendungen importiert werden. Node.js unterstützt dabei sowohl Module aus dem ECMAScript-Standard als auch aus dem Modul-Format CommonJS[32]. (vgl. OpenJS Foundation (o. J.d); OpenJS Foundation (o. J.e))

Das Ein-/Ausgabemodell von Node.js wird mithilfe einer *Ereignisschleife* umgesetzt. Innerhalb des Programm-Codes werden dabei zuerst *Rückruffunktionen* für Ein-/Ausgabe-Ereignisse registriert. Die Ereignisschleife verarbeitet in den einzelnen Iterationen eingegangene Ereignisse, indem die registrierten Rückruffunktionen ausgeführt werden. (vgl. OpenJS Foundation (o. J.f))

Eine Node.js-Instanz wird stets in einem einzigen Thread ausgeführt (vgl. OpenJS Foundation (o. J.b)). Um die Vorteile eines Mehrkernsystems zu nutzen, gibt es jedoch folgende Möglichkeiten:

- *Kind-Prozesse*: Durch Kind-Prozesse können Shell-Befehle (entweder direkt oder in einer neuen Shell) in einem neuen Prozess ausgeführt werden. Ein Spezialfall ist die Ausführung eines neuen Node.js-Prozesses, bei welchem auch eine Kommunikation zwischen den Prozessen möglich ist. (vgl. OpenJS Foundation (o. J.a))
- *Cluster*: Mit einem Cluster können mehrere Node.js-Kind-Prozesse erzeugt werden, welche sich denselben Netzwerk-Port teilen. Die Verteilung von Netzwerklast unter den Prozessen ist daher der primäre Anwendungsfall. (vgl. OpenJS Foundation (o. J.b))
- *Worker-Threads*: Ein Worker-Thread ist ein neuer Thread innerhalb des Prozesses. In diesem Thread wird eine neue Node.js-Instanz ausgeführt. Im Gegensatz zu

[32] https://requirejs.org/docs/commonjs.html

Kind-Prozessen können sich die Threads Speicherbereiche teilen. Die Kommunikation zwischen den Threads läuft, wie bei Web-Workern, durch Nachrichten ab. (vgl. OpenJS Foundation (o. J.h))

Zusammen mit Node.js wird der *Paketmanager* NPM[33] ausgeliefert. Die dazugehörige öffentliche *Registry* umfasst über eine Million Pakete und ist damit die weltweit größte Software-Registry. (vgl. OpenJS Foundation (o. J.g))

2.4 Docker

Docker[34] ist eine Plattform zum Builden, Ausliefern und Ausführen von Anwendungen in einem *Container*. Container werden auf einem Host-System ausgeführt. Im Gegensatz zu virtuellen Maschinen teilen sich Container den Betriebssystem-Kernel des Host-Systems untereinander. Container stellen so einen leichtgewichtigen nativen Prozess dar, welcher einen isolierten Kontext für die auszuführende Anwendung bereitstellt. (vgl. Docker Inc. (o. J.c))

Die einzelnen Anwendungen werden in sogenannten *Images* verpackt. Dieses stellt ein privates Dateisystem für die Anwendung bereit, in dem Anwendungscode, Bibliotheken und sonstige Abhängigkeiten enthalten sind. (vgl. Docker Inc. (o. J.c))

Definiert werden Images durch eine Datei mit dem Namen `Dockerfile`, welche beschreibt, wie das private Dateisystem zusammengestellt wird. Außerdem können Befehle für die spätere Ausführung des Images in dem Container festgelegt werden. Listing 2 zeigt ein exemplarisches Dockerfile. (vgl. Docker Inc. (o. J.b))

```
1 FROM node:6.11.5
2
3 WORKDIR /usr/src/app
4 COPY package.json .
5 RUN npm install
6 COPY . .
7
8 EXPOSE 12345
```

[33] https://www.npmjs.com/
[34] https://www.docker.com/

```
9
10  CMD [ "npm", "start" ]
```

Listing 2: Beispiel eines Dockerfiles (vgl. Docker Inc. (o. J.b))

Der Befehl `FROM` definiert ein *Basis-Image*, von welchem das definierte Image abgeleitet wird. Mithilfe des `WORKDIR`-Befehls wird das Arbeitsverzeichnis innerhalb des Containers festgelegt. Die darauffolgenden `COPY`- und `RUN`-Befehle dienen dem Kopieren von Dateien in den Container sowie der Ausführung von Befehlen innerhalb des Containers. Mit dem `EXPOSE`-Befehl können veröffentlichte Ports definiert werden. Durch `CMD` wird festgelegt, welcher Befehl in dem später gestarteten Container ausgeführt werden soll. (vgl. Docker Inc. (o. J.a))

Um Images auf anderen Systemen (etwa einem Server) bereitstellen zu können, werden Images in einer *Registry* hochgeladen. Neben privaten Registries, welche beispielsweise durch Cloud-Computing-Anbieter wie AWS[35] bereitgestellt werden, gibt es auch öffentliche Registries. Die größte öffentliche Registry für Docker-Images ist der Docker Hub[36]. (vgl. Docker Inc. (o. J.d))

2.5 Kubernetes

Kubernetes[37] ist eine Plattform zur Bereitstellung und Verwaltung von containerisierten Anwendungen innerhalb eines *Clusters* (vgl. Lukša (2018), S. 16). Von der zu Grunde liegenden Hardware wird dabei abstrahiert, wodurch Anwendungen ohne Kenntnis über die Hardware bereitgestellt werden können (vgl. Lukša (2018), S. 2). Administrative Prozesse werden durch Kubernetes erleichtert oder automatisiert, beispielsweise durch das automatische Neu-Starten fehlerhafter Container (*Self-Healing*) (vgl. Lukša (2018), S. 85).

Ein Kubernetes-Cluster besteht aus einem replizierbaren *Master* und aus einem oder mehreren *(Worker-)Nodes*. Der Master ist verantwortlich für die Erfüllung eines angestrebten Zustandes. Dieser angestrebte Zustand wird mithilfe einer API definiert und

[35] https://aws.amazon.com/de/ecr/
[36] https://hub.docker.com/
[37] https://kubernetes.io/

beschreibt u. a., welche Anwendungen und Container bereitgestellt werden, wie oft diese repliziert werden und welche Hardware-Ressourcen dafür zur Verfügung stehen sollen. Die Worker-Nodes bearbeiten die entstehende Arbeitslast. (vgl. Kubernetes Contributors (2019a))

Ein physikalisches Cluster kann in mehrere *Namespaces* aufgeteilt werden (vgl. Kubernetes Contributors (2019b)). Namespaces können beispielsweise genutzt werden, um separate Umgebungen für Test und Entwicklung zu schaffen (vgl. Lewis (2015)).

Die kleinsten bereitstellbaren Einheiten in Kubernetes sind die sogenannten *Pods*. Ein Pod ist eine Gruppe von ein oder mehreren Containern, welche sich Hardware-Ressourcen teilen und sich im selben Netzwerk befinden. Pods können als isolierte logische Hosts mit eigener IP-Adresse betrachtet werden. Die Container eines Pods können einander über `localhost` ansprechen. (vgl. Kubernetes Contributors (2019c))

Pods existieren so lange wie die enthaltenen Container und werden nach Terminierung nicht automatisch neu gestartet (vgl. Kubernetes Contributors (2020)). Erstellt und verwaltet werden Pods daher in der Regel durch sogenannte *Controller* (vgl. Kubernetes Contributors (2019c)).

Für zustandslose Anwendungen eignen sich *Deployment*-Controller. Wie auch andere Arten von Controllern stellen diese sicher, dass ausgefallene Pods neu gestartet werden (Self-Healing). Zudem wird das Rollout und Rollback von Updates sowie die Skalierung durch Replikation von Pods ermöglicht (vgl. Google LLC (2019c)). Für zustandsbehaftete Anwendungen (wie etwa eine WordPress-Installation) stehen *StatefulSets* zur Verfügung. (vgl. Google LLC (2019d))

Pods bekommen beim erneuten Starten eine neue IP-Adresse zugewiesen. Daher ergibt es wenig Sinn, diese nicht-permanente IP-Adresse direkt zu nutzen. Abhilfe an dieser Stelle schaffen *Services*, welche eine permanente IP-Adresse und eine Lastverteilung auf replizierte Pods ermöglichen. Durch einen Selektor, welcher sich aus als *Label* bezeichneten Key-Value-Paaren zusammensetzt, findet die Zuordnung zu den Pods statt. (vgl. Google LLC (2019j))

Mithilfe eines *Ingresses* kann der Zugriff von außerhalb des Clusters ermöglicht werden. Ein Ingress definiert Regeln zum Verteilen der externen HTTP(S)-Anfragen an die Services. Die Regeln können dabei auf dem Hostnamen oder dem Anfragepfad basieren. (vgl. Google LLC (2019g))

Eine *Service Discovery* mit DNS ermöglicht die Kommunikation eines Pods mit anderen Services innerhalb des Clusters. Die Hostnamen besitzen folgende Struktur: `<SERVICE>.<NAMESPACE>.<CLUSTER_DOMAIN>`, wobei die Cluster-Domain durch das Cluster definiert wird und etwa `cluster.local` lautet. (vgl. Google LLC (2020a))

Abbildung 2.4 zeigt den beschriebenen Aufbau von Kubernetes. Ressourcen in Kubernetes werden durch YAML-Dateien definiert. Listing 3 zeigt eine Definition für ein Deployment von zwei Backend-Pods und dem dazugehörigen Service. Die Backend-Pods im Beispiel beinhalten einen containerisierten PHP-Server.

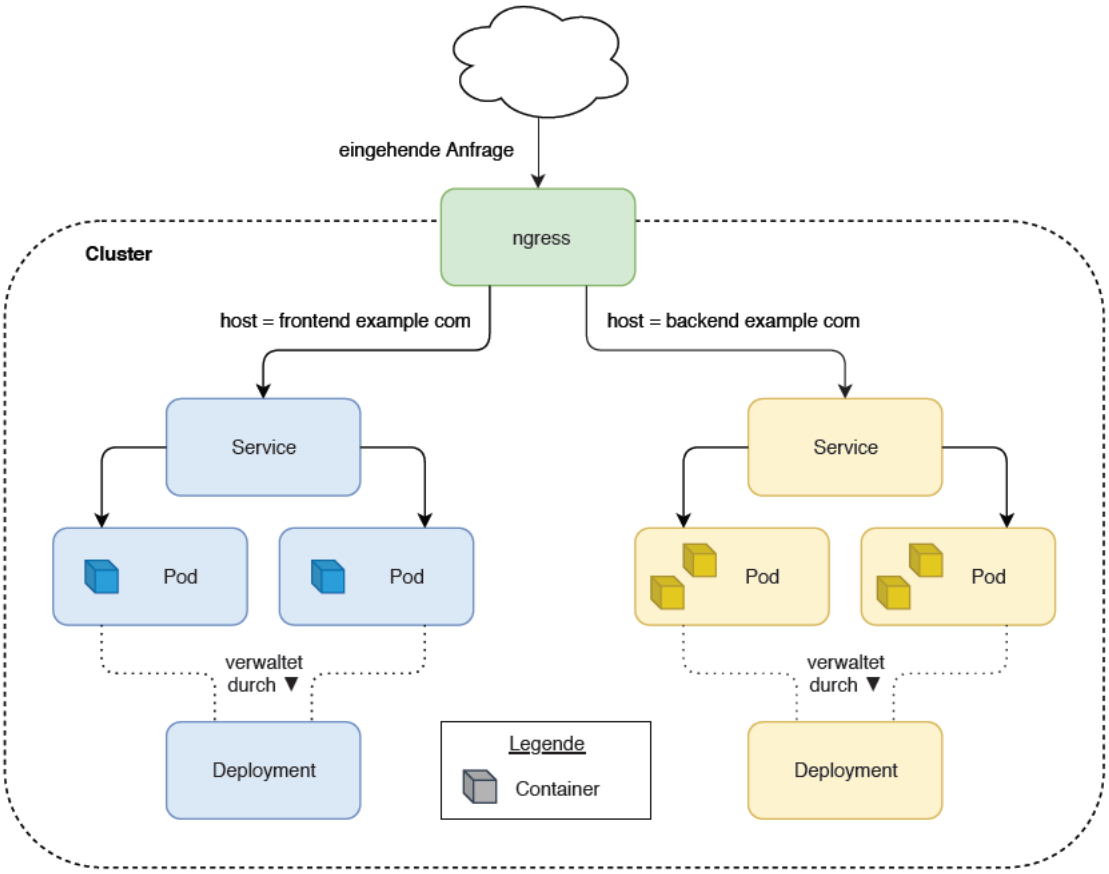

Abbildung 2.4: Aufbau Kubernetes
(Quelle: in Anlehnung an Palmer (o. J.), erstellt mit draw.io)

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-example-deployment
  labels:
    app: backend-example
spec:
  replicas: 2
  selector:
    matchLabels:
      app: backend-example
  template:
    metadata:
      labels:
        app: backend-example
    spec:
      containers:
      - name: backend-example
        image: php:latest
        ports:
        - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: backend-example-service
  labels:
    app: backend-example
spec:
  selector:
    app: backend-example
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
  type: NodePort
```

Listing 3: Beispiel eines Deployments mit dazugehörigem Service

3 Anforderungsanalyse

In der folgenden Anforderungsanalyse werden die Benutzer und deren Erwartungen an das System erfasst. „Ziel der Analyse ist, das bestehende Problem zu durchdringen und zu verstehen“ (Ludewig und Lichter (2013), S. 39). „Die Analyse ist die Vorarbeit und Voraussetzung der Spezifikation“ (Ludewig und Lichter (2013), S. 357), welche in dieser Arbeit in Kapitel 4 dargestellt wird.

3.1 Benutzerrollen

Eine Benutzerrolle ist eine Abstraktion für verschiedene Arten von Benutzern. Definiert werden Benutzerrollen durch die Menge von gemeinsamen Bedürfnissen und Interessen sowie gemeinsamen Verhaltensmustern und Verantwortlichkeiten. (vgl. Wirfs-Brock u. a. (1993), zit. n. Constantine und Lockwood (1999))

- *Leser*: Leser versprechen sich durch das Lesen von Beiträgen und Seiten einen Informationsgewinn. Sie unterscheiden sich nach registrierten Lesern (*Abonnenten*) und *unregistrierten Lesern* sowie nach der Wiederkehr-Häufigkeit (Einsteiger, Gelegenheitsleser und Stammleser).

  Große Unterschiede zeigen sich auch im thematischen Interesse der Leser. Während einige Leser des airact-Blogs lediglich auf der Suche nach neuen Reise-Deals sind, interessieren sich andere für aktuelle Entwicklungen der Flug- und Reisegesellschaften und beobachten diese kontinuierlich. Wiederum andere Leser interessieren sich für die einzelnen Reiseziele und „stöbern“ auf dem airact-Blog, um sich von diesem inspirieren zu lassen.

- *Autor*: Autoren recherchieren Inhalte und verfassen daraus neue Beiträge oder editieren ihre bestehenden Beiträge. Sie verwalten zudem die zu den Beiträgen gehörigen Medien wie etwa Bilder.

- *Redakteur*: Redakteure befassen sich mit der strategischen Planung von Inhalten. Dabei legen sie fest, wann welche Inhalte veröffentlicht werden. Durchgeführte Analysen der Inhalte dienen der Optimierung (wie der Suchmaschinenoptimierung) und ermöglichen, die Beliebtheit von Inhalten zu bewerten. Dabei findet durch entsprechende Recherchen auch stets ein Abgleich mit aktuellen Entwicklungen des Marktes statt.

  Redakteure können Kategorien, Schlagwörter und Inhalte aller Autoren bearbeiten. Bearbeitet ein Redakteur eigene Inhalte oder erstellt er neue Inhalte, nimmt er dabei die Rolle eines Autors ein.

  Oft nehmen Redakteure auch die Rolle eines *Reviewers* ein. Reviewer begutachten Inhalte, welche von einem Autor erstellt wurden und geben diese Inhalte zur Veröffentlichung frei. Das Review umfasst die Prüfung auf Tippfehler, inhaltliche Fehler und fehlerhafte Links/Bilder.

3.2 Funktionale Anforderungen

Funktionale Anforderungen beschreiben, welche Aktionen ein Benutzer mit dem System durchführen kann (vgl. Robertson und Robertson (2013), S. 10). In diesem Abschnitt werden die funktionalen Anforderungen in Form von User Stories und Epics erfasst. *User Stories* sind bewusst einfach gehaltene Beschreibungen von Funktionen, die den Benutzern einen Mehrwert bieten sollen (vgl. Beck u. a. (2001), S. 45ff). Der Aufwand für die Umsetzung der Funktion soll so klein sein, dass mehrere solcher Funktionen in einer Iteration (mit einer Dauer von wenigen Wochen) umgesetzt werden können (vgl. Beck u. a. (2001), S. 47 und 78). Große Stories, welche als *Epics* bezeichnet werden, können in mehrere Stories kleiner Größe aufgespalten werden (vgl. Cohn (2004), S. 6).

- Epic EP1: Als Leser/Autor/Redakteur möchte ich Beiträge/Seiten betrachten.
  - Story EP1-US1: Als Leser/Autor/Redakteur möchte ich eine chronologische Auflistung (absteigend sortiert) aller Beiträge oder der Beiträge einer bestimmten Kategorie betrachten.
  - Story EP1-US2: Als Leser/Autor/Redakteur möchte ich einen einzelnen Beitrag/eine einzelne Seite und den dazugehörigen Inhalt betrachten.

- Epic EP2: Als Autor/Redakteur möchte ich eine Vorschau von Änderungen an einem einzelnen Beitrag/einer einzelnen Seite betrachten.
  - Story EP2-US1: Als Autor möchte ich eine Vorschau von Änderungen an einem der Beiträge/einer der Seiten, die von mir erstellt wurden, betrachten.
  - Story EP2-US2: Als Redakteur möchte ich eine Vorschau von Änderungen an einem der Beiträge/einer der Seiten, die von einem anderen Autor erstellt wurden, betrachten.
- Story EP3-US1: Als Leser möchte ich einen einzelnen Beitrag oder eine einzelne Seite mit anderen Personen teilen.
- Epic EP4: Als Leser/Abonnent möchte ich mein persönliches Benutzerkonto verwalten.
  - Story EP4-US1: Als Leser möchte ich ein neues Benutzerkonto anlegen.
  - Story EP4-US2: Als Abonnent möchte ich mich mit meinem Benutzerkonto einloggen.
  - Story EP4-US3: Als Abonnent möchte ich mich von meinem Benutzerkonto ausloggen.
  - Story EP4-US4: Als Abonnent möchte ich mein bestehendes Passwort und meinen Anzeigenamen ändern.
- Epic EP5: Als Abonnent möchte ich meine persönlichen Angebotsbenachrichtigungen verwalten.
  - Story EP5-US1: Als Abonnent möchte ich die aktuellen Einstellungen meiner Angebotsbenachrichtigungen betrachten.
  - Story EP5-US2: Als Abonnent möchte ich eine neue Reiseklasse zu den Einstellungen meiner Angebotsbenachrichtigungen hinzufügen und eine bestehende Reiseklasse entfernen.
  - Story EP5-US3: Als Abonnent möchte ich ein neues Reiseziel zu den Einstellungen meiner Angebotsbenachrichtigungen hinzufügen und ein bestehendes Reiseziel entfernen.

  - Story EP5-US4: Als Abonnent möchte ich Angebotsbenachrichtigungen aktivieren und deaktivieren.

- Story EP6-US1: Als Abonnent möchte ich Angebotsbenachrichtigungen als Push-Benachrichtigung erhalten.
- Epic EP7: Als Leser möchte ich die Sprache der Benutzeroberfläche und des Inhaltes auswählen.
  - Story EP7-US1: Als Leser möchte ich die Sprache der Benutzeroberfläche ändern.
  - Story EP7-US2: Als Leser möchte ich eine Website aus dem Netzwerk auswählen, von welcher mir der Inhalt angezeigt wird.
- Story EP8-US1: Als Leser möchte ich die PWA zum Startbildschirm meiner Geräte hinzufügen.

3.3 Nicht-funktionale Anforderungen

Nicht-funktionale Anforderungen beschreiben qualitative Anforderungen an das System, welche anhand von Kriterien mess- und testbar gemacht werden (vgl. Robertson und Robertson (2013), S. 10). Beispiel für solche Anforderungen sind Performanz, Bedienbarkeit und Sicherheit (vgl. Robertson und Robertson (2013), S. 10). Die ISO-Norm 9126 und ihre Nachfolgenorm 25010 definieren eine Reihe weiterer nicht-funktionaler Anforderungen (vgl. Raharja und Siahaan (2019)). Die hier genannten Anforderungen ergeben sich auch aus dem Unterabschnitt 2.2.2.

- *Progressiv*: Entwickelt nach dem Prinzip der progressiven Verbesserung, sollen die funktionalen Anforderungen möglichst für alle Browser umgesetzt werden, auch für Browser mit geringem Funktionsumfang. Die PWA soll bei deaktiviertem JavaScript funktionsfähig sein. Weiterhin sollen diese verhältnismäßig verbreiteten Browser unterstützt werden: Internet Explorer 11.0, Safari 5.1 und Opera Mini. Deren Gemeinsamkeit ist, dass elementare JavaScript-Funktionalitäten (wie etwa Promises[1]) nicht unterstützt werden (vgl. Deveria (o. J.a); StatCounter (2019a)).

[1] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

Zusätzlich sollen die Funktionen moderner Browser genutzt werden, wobei die neusten Versionen von Chrome, Edge, Firefox und Safari getestet werden.

- *Responsiv*: Für die verbreitetsten Bildschirmgrößen soll die PWA den verfügbaren Platz optimal ausnutzen. Optimal in diesem Zusammenhang bedeutet, dass kein Platz „verschwendet“ wird und alle Elemente lesbar und übersichtlich dargestellt werden. Folgende Breiten des Anzeigebereiches vp_x sollen unterstützt werden (Angaben in Pixel): $vp_x < 640$ (Smartphone), $640 \leq vp_x < 768$ (Tablet, Portrait), $768 \leq vp_x < 1024$ (Tablet, Landscape), $1024 \leq vp_x < 1280$ (Desktop) und $1280 \leq vp_x$ (Desktop, hochauflösend) (vgl. Tailwind (o. J.)).
- *Netzwerkunabhängig*: Die PWA soll offline verfügbar sein. Bei einer schlechten Netzwerkverbindung sollen zuerst die bereits geladenen Inhalte angezeigt werden, damit dem Benutzer keine leere Seite angezeigt wird.
- *App-ähnlich*: Die Benutzeroberfläche der PWA soll intuitiv bedienbar sein und sich an Designstandards für native Apps orientieren. Im Anhang A.1 werden die Designstandards exemplarisch dargestellt.
- *Aktuell*: Neue Versionen der PWA sollen automatisch (d. h. ohne manuellen Download) zur Verfügung gestellt werden.
- *Sicher*: Die Kommunikation soll über eine gesicherte Verbindung erfolgen (HTTPS) und sensible Daten sollen nicht unverschlüsselt gespeichert werden.
- *Auffindbar*: Die PWA soll durch Suchmaschinen als Anwendung auffindbar und für Suchmaschinen optimiert sein. Bewertet werden soll dieser Aspekt mit einem Auditing-Werkzeug, wie WooRank[2].
- *Verlinkbar*: Die aktuell betrachtete Anzeige soll möglichst genau in der Adressleiste dargestellt werden. Damit soll das Verlinken eines Beitrag/einer Seite und das Setzen eines Lesezeichens für einen Beitrages/eine Seite im Browser möglich sein.
- Die PWA soll in allen Test-Kategorien des Lighthouse-Auditings eine „gute“ Bewertung erhalten (mindestens 90 von 100 Punkten) (vgl. Google LLC (2019i)). Außerdem sollen alle Kriterien der in Google LLC (o. J.g) definierten Checkliste für PWAs erfüllt sein.

[2] https://www.woorank.com/

4 Spezifikation

> „Die in der [Anforderungs-]Analyse festgestellten Anforderungen müssen geordnet, dokumentiert, geprüft, ergänzt und korrigiert werden. Da sich viele andere Arbeiten auf die Spezifikation stützen, ist dieser Schritt besonders wichtig.“ (Ludewig und Lichter (2013), S. 39)

In Abschnitt 4.1 wird der fachliche Kontext spezifiziert. Die erhobenen Anforderungen aus Abschnitt 3.2 werden in Abschnitt 4.2 spezifiziert.

4.1 Fachliches Datenmodell

Die Abbildungen 4.1 und 4.2 zeigen den fachlichen Kontext des entwickelten Systems. Abbildung 4.1 zeigt dabei den Kontext bezogen auf das WordPress-Netzwerk, deren Websites und den dazugehörigen Inhalten. In Abbildung 4.2 werden die für die Benutzer-Interaktion relevanten Aspekte des Systems dargestellt.

Die Abbildung erhebt keinen Anspruch auf Vollständigkeit und stellt nur den für das System relevanten Kontext dar. Die unterschiedlichen Farben dienen zur Erhöhung der Übersichtlichkeit, besitzen jedoch nur entfernt eine Semantik.

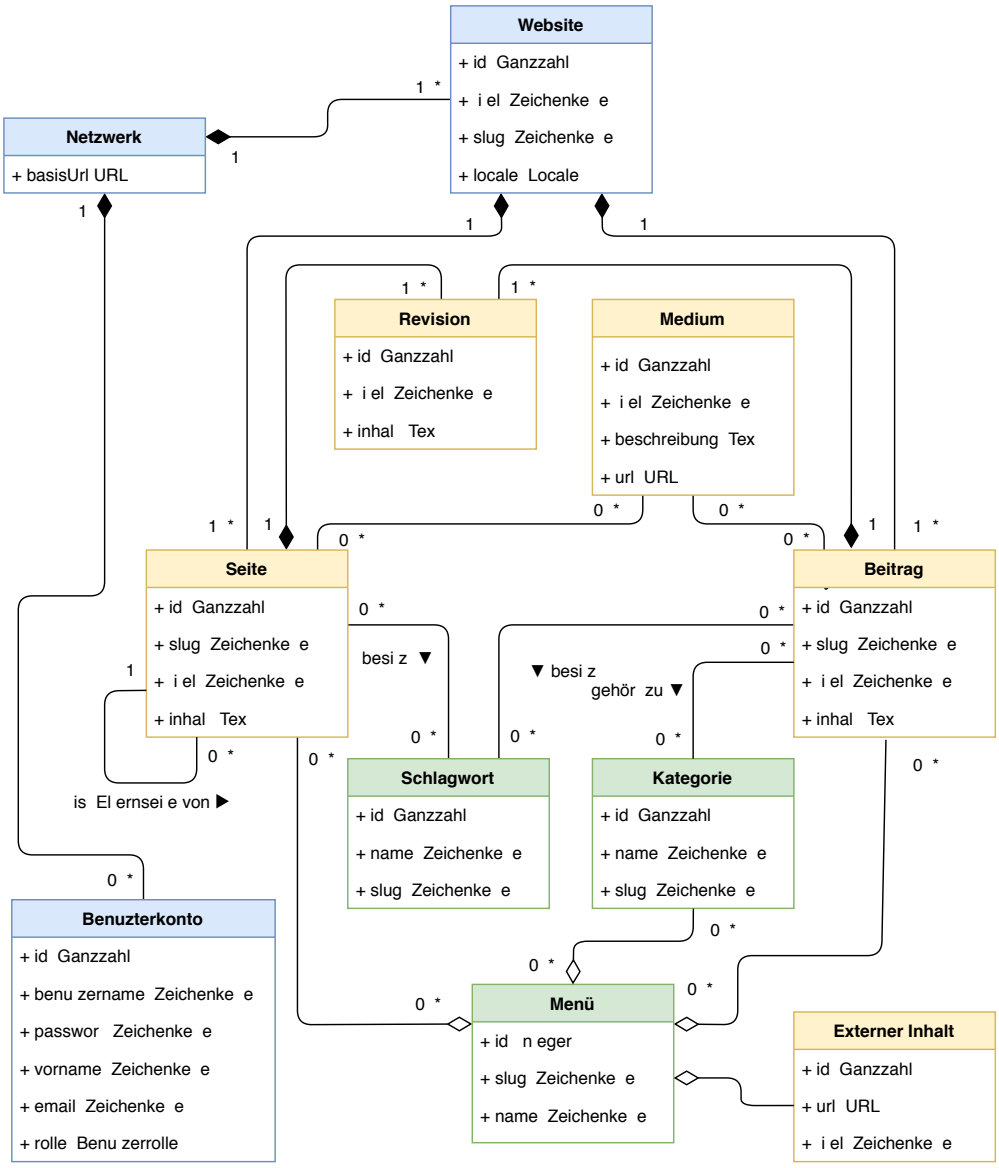

Abbildung 4.1: Fachliches Datenmodell: Netzwerk, Websites und Inhalte (Quelle: eigene Darstellung, erstellt mit draw.io)

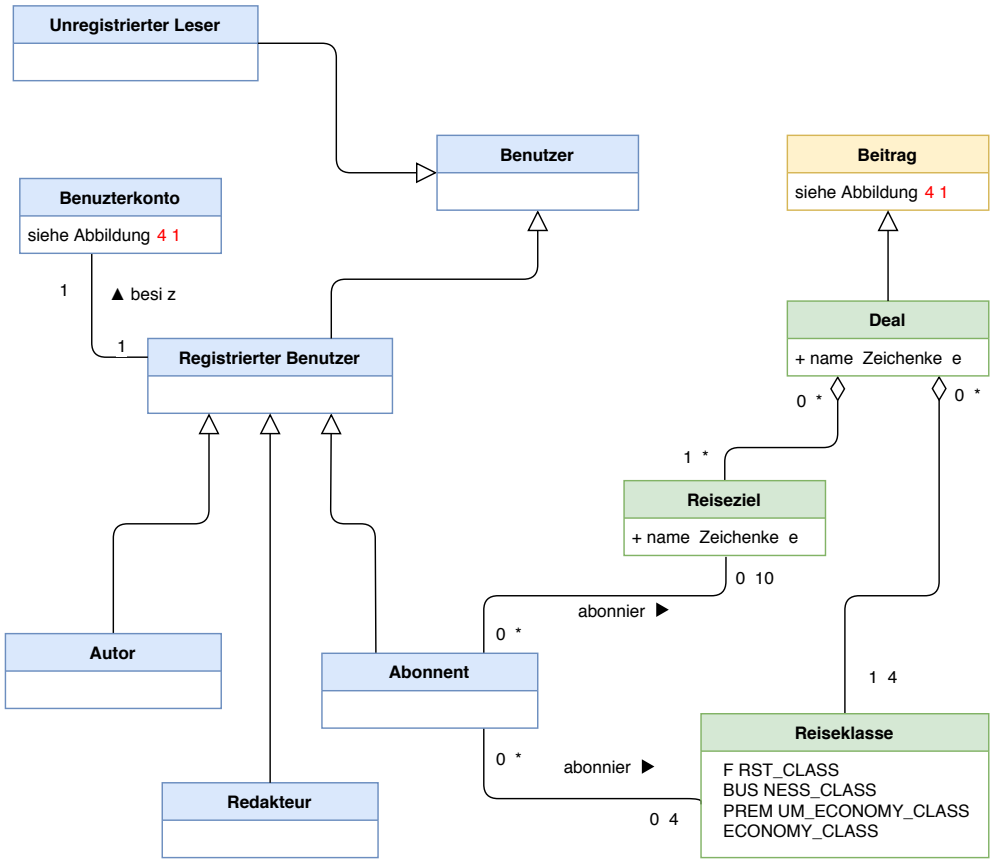

Abbildung 4.2: Fachliches Datenmodell: Interaktion der Benutzer
(Quelle: eigene Darstellung, erstellt mit draw.io)

4.2 Spezifikation der funktionalen Anforderungen

Im Folgenden werden die User Stories aus Abschnitt 3.2 näher spezifiziert. Neben einer textuellen Beschreibung werden dabei Use Cases und Wireframes genutzt.

Die PWA besitzt ein Seitenmenü, welches Zugang zu den einzelnen Funktionalitäten ermöglicht (vgl. Abbildung 4.3). Als Startseite der PWA dient die chronologische Auflistung aller Beiträge aus der User Story EP1-US1, welche im Folgenden spezifiziert wird.

Abbildung 4.3: Wireframe: Seitenmenü
(Quelle: eigene Darstellung, erstellt mit Lucidchart, Bildmaterial von Google LLC (o. J.d))

4.2.1 EP1-US1

> „Als Leser/Autor/Redakteur möchte ich eine chronologische Auflistung (absteigend sortiert) aller Beiträge oder der Beiträge einer bestimmten Kategorie betrachten."

Die Beiträge, die der Benutzer betrachten möchte, sind an die aktuell ausgewählte Website geknüpft (vgl. User Story EP7-US2). Abhängig von dieser, werden alle gewünschten Beiträge angezeigt. Da es sich hierbei um eine große Anzahl an Beiträgen handelt, ist eine sogenannte *Pagination*, d. h. die Aufteilung in mehrere Seiten, notwendig.

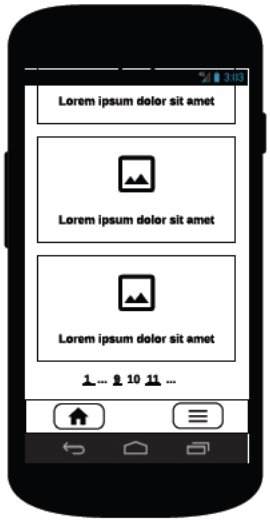

Abbildung 4.4: Wireframe zur User Story EP1-US1
(Quelle: eigene Darstellung, erstellt mit Lucidchart, Bildmaterial von Google LLC (o. J.d))

Die URL innerhalb der PWA soll, bei allen Beiträgen, die der Startseite (`<BASE_URL>/`) entsprechen und bei Beiträgen einer bestimmten Kategorie `<BASE_URL>/categories/<CATEGORY_SLUG>` lauten. Durch die Nutzung der Slug (beispielsweise `deals`) ist eine aussagekräftige URL gegeben, welche sich positiv auf die Bewertung durch Suchmaschinen auswirkt.

4.2.2 EP1-US2

> „Als Leser/Autor/Redakteur möchte ich einen einzelnen Beitrag/eine einzelne Seite und den dazugehörigen Inhalt betrachten.“

Auch hier ist der Beitrag bzw. die Seite an die aktuell ausgewählte Website geknüpft. Neben dem Text-Inhalt sollen auch die eingebundenen Medien dargestellt werden. Die URL innerhalb der PWA soll bei einem Beitrag `<BASE_URL>/posts/<POST_SLUG>` und bei einer Seite `<BASE_URL>/pages/<PAGE_SLUG>` lauten.

Abbildung 4.5: Wireframe zur User Story EP1-US2
(Quelle: eigene Darstellung, erstellt mit Lucidchart, Bildmaterial von Google LLC (o. J.d))

4.2.3 EP2-US1

> „Als Autor möchte ich eine Vorschau von Änderungen an einem der Beiträge/einer der Seiten, die von mir erstellt wurden, betrachten."

Der folgende Use Case beschreibt das Szenario der User Story und ist angelehnt an den Vorlagen von Cockburn (1998) und Ludewig und Lichter (2013), S. 388. Abbildung 4.6 skizziert die darin erwähnte Bearbeitungsseite eines Beitrags/einer Seite.

Abbildung 4.6: Wireframe zur User Story EP2-US1
(Quelle: eigene Darstellung, erstellt mit Lucidchart, Bildmaterial von Flatart (2019))

Bezeichner: UC1
Name: Vorschau betrachten
Ziel: Der Autor betrachtet eine Vorschau des bearbeiteten Beitrags/der bearbeiteten Seite.
Vorbedingung:
Der Autor ist im WordPress-Admin-Backend eingeloggt.
Nachbedingung:
Dem Autor wird der Bearbeitungszustand angezeigt.
Akteuer: Autor

Auslöser:
Der Autor hat einen Beitrag oder eine Seite bearbeitet und möchte den Bearbeitungszustand in einer Vorschau betrachten (entweder sofort oder zu einem späteren Zeitpunkt).

Beschreibung:

1. Der Autor öffnet die Bearbeitungsseite des Beitrages/der Seite im WordPress-Admin-Dashboard.
2. Das System prüft ob Änderungen an dem Beitrag/der Seite vorgenommen wurden.
3. Der Autor wählt die Funktion „Vorschau der Änderungen“.
4. Das System speichert den aktuellen Bearbeitungszustand.
5. Das System öffnet die PWA in einem neuen Reiter.
6. Das System prüft ob der Autor in der PWA eingeloggt ist.
7. Das System zeigt den gespeicherten Bearbeitungszustand in der PWA an.
8. Der Autor betrachtet die Vorschau.

Erweiterungen:

1.a. Der Autor befindet sich bereits auf der Bearbeitungsseite:

1.a.1. Dieser Schritt des Erfolgsszenarios wird übersprungen und es wird mit dem Erfolgsszenario fortgefahren.

6.a. Der Autor ist noch nicht in der PWA eingeloggt:

6.a.1. Ausführung der User Story EP4-US2 (Login). Bei erfolgtem Login wird mit dem Erfolgsszenario fortgefahren. Schlägt der Login fehl, fordert das System zur nochmaligen Eingabe auf.

Sonderfälle:

2.a. Es wurden keine Änderungen an dem Beitrag/der Seite vorgenommen:

2.a.1. Die Funktion „Vorschau der Änderungen“ wird deaktiviert und der Vorgang wird abgebrochen.

Zugehörige User Stories:
User Story EP2-US1

Die angezeigte Vorschau soll der Anzeige aus User Story EP1-US2 entsprechen. Die URL innerhalb der PWA soll bei einem Beitrag `<BASE_URL>/posts/<POST_SLUG>/preview` und bei einer Seite `<BASE_URL>/pages/<PAGE_SLUG>/preview` lauten.

4.2.4 EP2-US2

> „Als Redakteur möchte ich eine Vorschau von Änderungen an einem der Beiträge/einer der Seiten, die von einem anderen Autor erstellt wurden, betrachten.“

Das Erfolgsszenario entspricht dem beschriebenen Szenario aus UC1, jedoch ausgeführt von dem Redakteur. Alternativ soll ein Autor auch direkt die URL der Vorschau an den Redakteur weiterleiten können, ohne dass letzterer die Bearbeitungsseite des Beitrages/der Seite öffnen muss.

4.2.5 EP3-US1

> „Als Leser möchte ich einen einzelnen Beitrag oder eine einzelne Seite mit anderen Personen teilen.“

Zum Teilen eines Beitrages bzw. einer Seite soll – sofern technisch möglich – die native Funktionalität der Plattform genutzt werden. Ist dies technisch nicht möglich, sollen zumindest Links zum Teilen auf verschiedenen Social-Media-Plattformen bereitgestellt werden.

Abbildung 4.7: Wireframe zur User Story EP3-US1
(Quelle: eigene Darstellung, erstellt mit Lucidchart, Bildmaterial von Google LLC (o. J.d))

Abbildung 4.8: Wireframe zur User Story EP3-US1
(Quelle: eigene Darstellung, erstellt mit Lucidchart, Bildmaterial von Google LLC (o. J.d) und Flatart (2019))

4.2.6 EP4-US1

„Als Leser möchte ich ein neues Benutzerkonto anlegen."

Zum Anlegen eines neuen Benutzerkontos gibt der Leser seine persönlichen Daten ein und bestätigt diese. Zu den notwendigen Eingabedaten gehört die E-Mail-Adresse des Lesers sowie das gewünschte Passwort und der gewünschte Anzeigename. Dabei steht es dem Benutzer frei, nur seinen Vornamen oder seinen vollständigen Namen anzugeben. Ein Benutzername wird entgegen dem WordPress-Standard nicht benötigt und stattdessen automatisch generiert.

Abbildung 4.9: Wireframe zur User Story EP4-US1
(Quelle: eigene Darstellung, erstellt mit Lucidchart, Bildmaterial von Google LLC (o. J.d))

4.2.7 EP4-US2

„Als Abonnent möchte ich mich mit meinem Benutzerkonto einloggen."

Das Einloggen soll sowohl mit einem neu angelegten Benutzerkonto, als auch mit einem bestehenden Benutzerkonto möglich sein, welches bereits vor Einführung der PWA erstellt wurde. Bei der Anmeldung wird die E-Mail-Adresse und das gewählte Passwort abgefragt.

Abbildung 4.10: Wireframe zur User Story EP4-US2
(Quelle: eigene Darstellung, erstellt mit Lucidchart, Bildmaterial von Google LLC (o. J.d))

4.2.8 EP4-US3

> „Als Abonnent möchte ich mich von meinem Benutzerkonto ausloggen."

Das Ausloggen wird durch einen Menü-Punkt im Seitenmenü der PWA ermöglicht (vgl. Abbildung 4.3).

4.2.9 EP4-US4

> „Als Abonnent möchte ich mein bestehendes Passwort und meinen Anzeigenamen ändern."

Der Abonnent kann unter Eingabe seines bestehenden Passworts Änderungen an seinem Benutzerkonto vornehmen. Dabei kann das Passwort und der Anzeigename geändert werden, die E-Mail-Adresse hingegen ist unveränderlich.

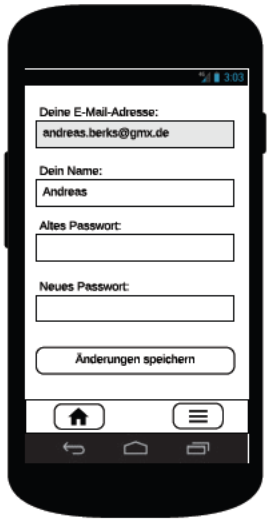

Abbildung 4.11: Wireframe zur User Story EP4-US4
(Quelle: eigene Darstellung, erstellt mit Lucidchart, Bildmaterial von Google LLC (o. J.d))

4.2.10 EP5-US1 – EP5-US4

EP5-US1

„Als Abonnent möchte ich die aktuellen Einstellungen meiner Angebotsbenachrichtigungen betrachten."

EP5-US2

„Als Abonnent möchte ich eine neue Reiseklasse zu den Einstellungen meiner Angebotsbenachrichtigungen hinzufügen und eine bestehende Reiseklasse entfernen."

EP5-US3

„Als Abonnent möchte ich ein neues Reiseziel zu den Einstellungen meiner Angebotsbenachrichtigungen hinzufügen und ein bestehendes Reiseziel entfernen."

EP5-US4

> „Als Abonnent möchte ich Angebotsbenachrichtigungen aktivieren und deaktivieren.“

Der Benutzer hat hierbei die Möglichkeit, favorisierte Reiseziele und -klassen für die Angebotsbenachrichtigungen festzulegen. Letztere können dabei gänzlich aktiviert bzw. deaktiviert werden. Bei Deaktivierung der Angebotsbenachrichtigungen sollen die favorisierten Reiseziele und -klassen erhalten bleiben. Die in Abbildung 4.12 und 4.13 gezeigten Wireframes stellen die Einstellungen dar.

Bei erstmaliger Aktivierung der Angebotsbenachrichtigungen werden die Berechtigungen angefragt, Push-Benachrichtigungen zu senden. Da sich der Benutzer bewusst zur Aktivierung der Angebotsbenachrichtigungen entscheidet, ist die Wahrscheinlichkeit einer Zustimmung zu diesem Zeitpunkt am höchsten.

Innerhalb des Prototyps werden Push-Benachrichtigungen nur an das zuletzt genutzte Gerät gesendet. In einer späteren Anwendung bietet es sich jedoch an, auch mehrere Geräte unterstützen zu können. Eine hierbei mögliche Vorgehensweise wird in dem Ausblick dieser Arbeit dargestellt (vgl. Unterabschnitt 9.2.2).

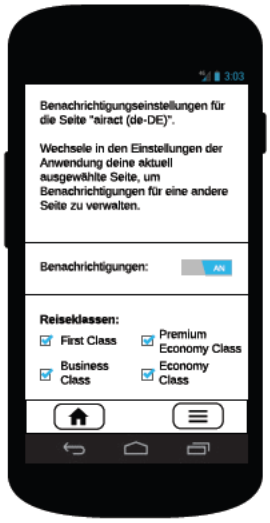

Abbildung 4.12: Wireframe zur User Stories EP5-US1 – EP5-US4
(Quelle: eigene Darstellung, erstellt mit Lucidchart, Bildmaterial von Google LLC (o. J.d))

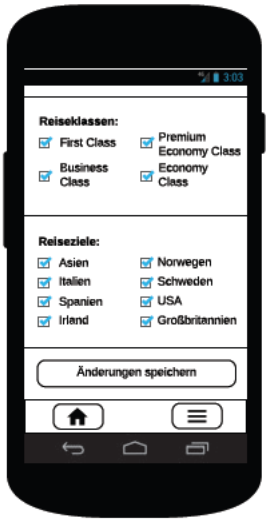

Abbildung 4.13: Wireframe zur User Stories EP5-US1 – EP5-US4
(Quelle: eigene Darstellung, erstellt mit Lucidchart, Bildmaterial von Google LLC (o. J.d))

4.2.11 EP6-US1

> „Als Abonnent möchte ich Angebotsbenachrichtigungen als Push-Benachrichtigung erhalten."

Die Angebotsbenachrichtigung soll den Titel des Beitrages als Text anzeigen. Bei Klick auf die Angebotsbenachrichtigung soll der Benutzer auf die entsprechende URL innerhalb der PWA geleitet werden (vgl. User Story EP1-US2).

4.2.12 EP7-US1 und EP7-US2

EP7-US1

> „Als Leser möchte ich die Sprache der Benutzeroberfläche ändern."

EP7-US2

> „Als Leser möchte ich eine Website aus dem Netzwerk auswählen, von welcher mir der Inhalt angezeigt wird."

Beide Einstellungsmöglichkeiten werden in Abbildung 4.14 gezeigt und sollen jeweils durch ein Drop-Down-Menü mit allen auswählbaren Werten umgesetzt werden.

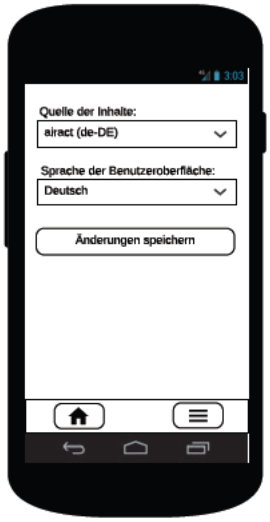

Abbildung 4.14: Wireframe zur User Story EP7-US1
(Quelle: eigene Darstellung, erstellt mit Lucidchart, Bildmaterial von Google LLC (o. J.d))

4.2.13 EP8-US1

> „Als Leser möchte ich die PWA zum Startbildschirm meiner Geräte hinzufügen.“

Das Hinzufügen zum Startbildschirm soll entweder manuell möglich sein oder proaktiv durch den „Add to Home Screen“-Dialog, welcher von dem Browser angezeigt wird, sobald bestimmte Heuristiken im Benutzerverhalten erfüllt sind.

5 Entwurf und Architektur

> „Software-Systeme [...] bestehen aus Modulen oder Komponenten, die miteinander die Gesamtfunktionalität des Systems bieten. Die Gesamtstruktur wird durch die Software-Architektur beschrieben. Die Schnittstellen von Komponenten werden so präzise wie möglich festgelegt, damit die Komponenten parallel entwickelt und am Ende problemlos integriert werden können.“ (Ludewig und Lichter (2013), S. 39)

Nach Starke werden in der Dokumentation einer solchen Software-Architektur verschiedene *Sichten* eingenommen, die folgende Fragen beantworten sollen: (Starke (2015), S. 142)

- „Wie ist der Quellcode des Systems organisiert und strukturiert? Welche „größeren“ Implementierungseinheiten (Bausteine, Strukturelemente) gibt es und welche Beziehungen gibt es dazwischen?“ (siehe Abschnitt 5.6)
- „Wie verhalten sich die Bausteine zur Laufzeit und wie arbeiten sie zusammen?“ (siehe Abschnitt 5.7)
- „Wie fügt sich das System in seine Umgebung ein, insbesondere in seine technische Infrastruktur sowie die Projektorganisation?“ (siehe Abschnitt 5.5, 5.8)
- „Welche grundlegenden Entscheidungen, Technologien oder Konzepte prägen die Lösung, deren Implementierung und Betrieb?“ (siehe Abschnitt 5.1 – 5.4)

Die Schnittstellen der einzelnen Komponenten werden in Abschnitt 5.9 beschrieben.

5.1 Technologie-Auswahl: Frontend

Im Folgenden wird auf die genutzten Technologien in dem Frontend des airact-Systems eingegangen. Dabei werden sowohl Technologien zur Darstellung der Benutzeroberfläche als auch zur Verwaltung des Zustandes betrachtet.

5.1.1 Benutzeroberfläche

Die drei größten Technologien für die Entwicklung von Benutzeroberflächen sind React[1], Angular[2] und Vue[3]. Während der Marktanteil von React und Vue stark zunimmt, zeigt er bei Angular stagnierende bis abnehmende Tendenzen. Dennoch ist Angular, wie auch React, stark verbreitet. Vue hat eine derartige Verbreitung bisher nicht erreicht. Während Angular mit Google und React mit Facebook von großen Firmen entwickelt wird, wird Vue primär durch die Community gestützt (vgl. Martin (2019)). (vgl. Anhang A.2)

React und Vue bieten einen minimalen Funktionsumfang, welcher durch ein umfassendes Ökosystem erweitert werden kann. Angular liefert von Haus aus einen großen Funktionsumfang, wodurch für grundlegende Funktionalitäten keine weiteren Bibliotheken benötigt werden. (vgl. Martin (2019))

Angular bietet umfangreiche Möglichkeiten zur Code-Strukturierung (etwa Module und Dependency Injection) und folgt den Architekturmustern MVC und MVVM (vgl. Google LLC (o. J.a); Google LLC (o. J.b)). Damit wird eine klare Trennung zwischen Repräsentation und Funktion forciert. React und Vue machen wenige Vorgaben zur Architektur. Mit dieser Flexibilität besteht die Gefahr, dass Anwendungen entgegen bewährter Architekturmuster und -prinzipien entwickelt werden.

Angular setzt den Fokus auf die Programmiersprache TypeScript. Eine Entwicklung in JavaScript ist theoretisch möglich, widerspricht jedoch den Standards (vgl. Wellman (2018)). React und Vue werden primär in JavaScript entwickelt, unterstützen jedoch ebenfalls TypeScript (vgl. Microsoft Corporation (o. J.a); Vue.js Contributors (o. J.)).

Während Angular direkte Operationen an dem „echten“ DOM vornimmt, arbeiten React und Vue auf einem virtuellen DOM. Das virtuelle DOM ist ein Abbild des echten DOMs, welches im Arbeitsspeicher gehalten und mit dem echten DOM synchronisiert wird (vgl.

[1] https://reactjs.org/
[2] https://angular.io/
[3] https://vuejs.org/

Facebook Inc. (o. J.e)). Dieses Vorgehen sorgt für eine bessere Performanz. Anhang A.3 zeigt die Unterschiede, gemessen an verschiedenen Metriken. (vgl. Martin (2019))

In der PWA wurde sich für die React-Bibliothek entschieden. Aufgrund der hohen Verbreitung und des hohen Wachstums ist die Bibliothek zukunftssicher. Die hohe Anzahl an Entwicklern führt dazu, dass es viele vorgefertigte Lösungen gibt, die die Entwicklung erleichtern. Dabei muss jedoch stets auf die konsequente Einhaltung von Entwurfsprinzipien geachtet werden. Im Rahmen der folgenden Abschnitte werden dafür verschiedene Leitlinien definiert. Dazu gehört auch die Verwendung von TypeScript.

Elemente der Benutzeroberfläche in React werden durch die Programmiersprache *JSX* beschrieben, einer syntaktischen Erweiterung der Programmiersprache JavaScript bzw. TypeScript. JSX ermöglicht die Definition von *Elementen* in einer HTML-ähnlichen Syntax. (vgl. Facebook Inc. (o. J.d))

Den Kern von React bilden die sogenannten *Komponenten*. Komponenten sind unabhängige und wiederverwertbare Elemente, aus denen sich das spätere HTML-Dokument zusammensetzt. Mithilfe von *Eigenschaften* (Properties) können Komponenten parametrisiert werden. Komponenten lassen sich baumartig schachteln, wobei der Wurzel-Knoten in das HTML-Dokument eingehängt wird (vgl. Listing 5). Seit der Version 0.14 von React können Komponenten auch als Funktionen definiert werden, welche als *Functional Components* bezeichnet werden (vgl. Facebook Inc. (2019b); Listing 4). (vgl. Facebook Inc. (o. J.a))

```tsx
// greeting.tsx
import React from "react";

export interface GreetingProps {
    name?: string;
}

function GreetingComp(props: GreetingProps): JSX.Element {
    return <h1>Hello {props.name || "world"}!</h1>;
}

export default GreetingComp;
```

Listing 4: React-Komponente mit Eigenschaften

```tsx
// index.tsx
import React from "react";
import { render } from "react-dom";
import Greeting from "./greeting"

const root = <Greeting name="Andreas" />;
const mountNode = document.getElementById("root");
render(root, mountNode);
```

Listing 5: Einhängen einer React-Komponente in den Knoten `root`

Elemente, wie etwa Buttons, lösen bei Benutzereingaben *Ereignisse* (Events) aus, welche durch *Ereignis-Listener* verarbeitet werden können (vgl. Facebook Inc. (o. J.b)). Bei der Verarbeitung eines Ereignisses kann beispielsweise der *Zustand* einer Komponente geändert werden. Innerhalb von Functional Components können dafür sogenannte Hooks verwendet werden, welche in der Version 16.8 von React eingeführt wurden (vgl. Facebook Inc. (o. J.c)).

Durch *Hooks* besteht die Möglichkeit, sich in den internen Zustand von React und den Lebenszyklus von Komponenten „einzuhaken“. Hooks sind Funktionen, die mit dem Präfix `use` beginnen. Neben eine Reihe von bestehenden Hooks wie der `useState`-Hook können auch eigene Hooks definiert werden. (vgl. Facebook Inc. (o. J.c))

Listing 6 zeigt eine exemplarische Komponente, welche ein Datum mit Uhrzeit anzeigt und dieses beim Drücken des `Refresh`-Buttons aktualisiert.

```tsx
import React, { useState } from "react";

export interface ClockProps {}

function ClockComp(props: ClockProps): JSX.Element {
    const initialDate = new Date();
    const [date, setDate] = useState(initialDate);

    function handleClick(e) {
        setDate(new Date());
    }

```

```
13      return (
14          <div>
15              <p>Last refreshed at {date.toLocaleTimeString()}.</p>
16              <button onClick={handleClick}>Refresh</button>
17          </div>
18      );
19  }
20
21  export default ClockComp;
```

Listing 6: React-Komponente mit Event-Listener und Zustand

5.1.2 Universelles Rendering

Eine React-Anwendung stellt dem Client ein leeres HTML-Dokument und die dazugehörigen JavaScript-Dateien bereit. Sobald alle JavaScript-Dateien heruntergeladen wurden, füllt der Browser das HTML-Dokument mit Inhalt. Dieses Vorgehen wird als *clientseitiges Rendering* bezeichnet. Durch den JavaScript-Code lässt sich eine flüssige Navigation realisieren, bei welcher nicht die gesamte Benutzeroberfläche, sondern nur veränderte Inhalte aktualisiert werden. (vgl. Grigoryan (2016))

Das Herunterladen und Interpretieren der JavaScript-Dateien sorgt für eine Verzögerung bis zur ersten Darstellung der Benutzeroberfläche (*First Contentful Paint* (FCP)) und des Inhaltes (*First Meaningful Paint* (FMP)). Die genannten Metriken werden in Walton (o. J.) näher beschrieben. Die hohe Abhängigkeit von JavaScript erschwert zudem die Entwicklung nach dem Prinzip der progressiven Verbesserung.

Ein alternatives Vorgehen ist das *serverseitige Rendering*. Hierbei wird der Inhalt des HTML-Dokuments durch den Server generiert, bevor dieses an den Client gesendet wird. Das HTML-Dokument kann dadurch direkt im Browser dargestellt werden. Erst danach werden die ebenfalls bereitgestellten JavaScript-Dateien interpretiert, um den virtuellen DOM aufzubauen und die Website interaktiv zu machen. Auch wenn Suchmaschinen-Crawler heutzutage JavaScript unterstützen, stellt das serverseitige Rendering sicher, dass Suchmaschinen den Inhalt der Website richtig erfassen kann. (vgl. Grigoryan (2016))

Das Rendern durch den Server erhöht die *Time to First Byte* (TTFB), wodurch auch der JavaScript-Code verzögert übertragen wird. Dadurch erhöht sich die *Time to Interactive*

(TTI)), d. h. die Anwendung wird später interaktiv. Die erhöhte Serverlast verringert zudem den Durchsatz. In Abschnitt 7.2 und 7.3 werden diese Aspekte durch entsprechende Tests bewertet. (vgl. Grigoryan (2016))

Mithilfe von Static-Site-Generatoren wie GatsbyJS[4] können die Website und ihre Unterseiten bereits im Vorfeld gerendert werden. Dieses Vorgehen führt zu einer direkten Auslieferung von HTML-Dokumenten mit Inhalt, einem schnelleren FCP und FMP sowie einer schnelleren TTI. (vgl. Nelson (2018))

Aufgrund der Häufigkeit, in der neue und veränderte Inhalte in dem airact-Blog erscheinen, ist ein Static-Site-Generator jedoch nur begrenzt geeignet. Dafür sorgen insbesondere das Erstellen neuer Beiträge und die denkbare Funktion in späteren Versionen der PWA Kommentare zu Beiträgen und Seiten zu verfassen.

Ein Framework für das serverseitige Rendering ist Next.js[5], welches auch in der PWA genutzt wird. Next.js bietet neben dem strikten serverseitigen Rendering auch Funktionen eines Static-Site-Generators, bezeichnet als *statische Optimierung* (vgl. ZEIT, Inc. (o. J.a)). Verglichen mit GatsbyJS ist Next.js leichtgewichtig, stellt dafür jedoch nicht so viel Funktionalität und nur eine begrenzte Anzahl an Plugins[6] bereit. Dennoch lassen sich NPM-Pakete gut integrieren, wie die offiziellen Beispiel-Anwendungen[7] von Next.js zeigen.

Schlussendlich wurde sich in der PWA für ein *universelles Rendering* (auch *isomorphes Rendering*) entschieden. Das universelle Rendering sorgt dafür, dass die Website beim ersten Zugriff serverseitig gerendert wird, um eine schnelle erste Darstellung zu ermöglichen. Nach vollständigem Laden des JavaScript-Codes wird die Website im Weiteren clientseitig gerendert und bietet somit eine flüssige Navigation ohne komplettes Neuladen der Benutzeroberfläche. (vgl. Antonio (2015))

5.1.3 Zustandsverwaltung

Neben der durch React und Next.js bereitgestellten View der PWA ist zusätzlich eine Architektur notwendig, welche auch die Verwaltung von Zuständen und die Handhabung

[4] https://www.gatsbyjs.org/
[5] https://nextjs.org/
[6] https://github.com/zeit/next-plugins
[7] https://github.com/zeit/next.js/tree/master/examples

von komponenten-übergreifenden Ereignissen ermöglicht. Eine weit verbreitete Architektur in React-Anwendungen ist die Redux[8] Architektur (vgl. Anhang A.5). Diese Architektur basiert auf dem Architekturstil Flux[9], welcher von Facebook entwickelt wurde, um deren bisher genutzten MVC-Architekturstil in komplexen Anwendungen zu ersetzen (vgl. Facebook Inc. (2019a)).

Redux nutzt einen globalen Zustand, den sogenannten *Store*, welcher als „single source of truth" dient. Die *View* abonniert den Store, wodurch Zustandsänderungen an die View übermittelt werden. Basierend auf Prinzipien der funktionalen Programmierung ist der Store *immutable*. Bei Benutzereingaben sendet die View Ereignisse, die sogenannten *Actions*, an einen *Reducer*. Der Reducer ist eine Funktion, welche aus dem bisherigen Zustand und dem Ereignis (*Action*) einen neuen Zustand generiert. Abbildung 5.1 zeigt den beschriebenen Datenfluss. (vgl. Redux Contributors (o. J.))

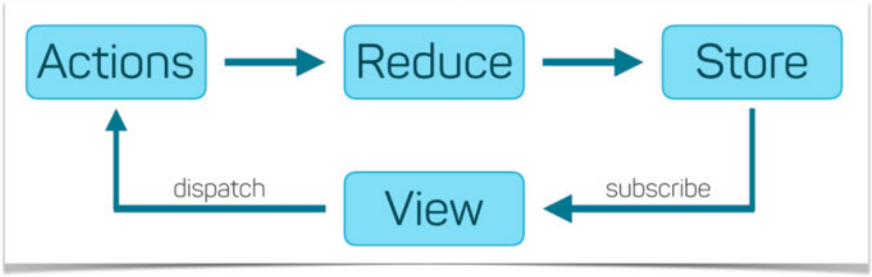

Abbildung 5.1: Datenfluss der Redux-Architektur
(Quelle: Rigau (2018))

Dieses Prinzip sorgt für ein vorhersehbares Ergebnis ohne Seiteneffekte. Durch die damit einhergehende Verständlichkeit, Wartbarkeit und Testbarkeit eignet sich die Redux-Architektur für komplexe Anwendungen und hebt sich damit von alternativen Architekturen, wie der MobX[10]-Architektur, ab. Mit den Redux DevTools[11] steht weiterhin eine nützliche Erweiterung zum Debugging bereit. Diese Vorteile überwiegen den Nachteilen, zu welchen das verhältnismäßig hohe Maß an Boilerplate-Code und eine steile Lernkurve gehört. (vgl. Wieruch (2017))

[8] https://redux.js.org/
[9] https://facebook.github.io/flux/
[10] https://mobx.js.org/
[11] https://github.com/reduxjs/redux-devtools

5.1.4 Komponenten der Benutzeroberfläche

Es stehen viele Bibliotheken für React bereit, welche vorgefertigte Komponenten für die Benutzeroberfläche bieten. Um ein Design umzusetzen, welches sich möglichst an der nativen Plattform orientiert, wurden verschiedene Bibliotheken miteinander verglichen. Untersucht wurden die Bibliotheken Onsen UI[12] und Ionic[13], welche sowohl UI-Elemente für Android als auch für iOS bieten. Ionic wird seit der Version 4 auch unabhängig von dem Ionic-Framework bereitgestellt.

Erste Recherchen konnten nicht bestätigen, dass eine Integration von einer der beiden Bibliotheken mit Next.js möglich ist (vgl. Allonnea (2019); Li (2018)). Auch Versuche des Autors kamen stets zu der Erkenntnis, dass beide Bibliotheken durch die Nutzung des clientseitigen `window`-Objekts die Integration mit Next.js erschweren. Dieses Objekt kann in Next.js zwar durch Features wie dem „dynamic-import“ genutzt werden, jedoch nur durch die Deaktivierung des serverseitigen Renderings (vgl. Neutkens (2019); ZEIT, Inc. (o. J.b)).

Im Rahmen dieser Arbeit wurde das serverseitige Rendering dem Angleich des Designs an die native Plattform vorgezogen. Auch wenn ein Angleich an die native Plattform eine bessere Benutzererfahrung bietet, ist dieser keine Voraussetzung, um die Anforderung der App-Ähnlichkeit zu erfüllen. Statt der beiden genannten Bibliotheken wurde daher die Bibliothek Material-UI[14] gewählt. Diese setzt das in Android genutzte Material-Design um, womit zumindest bei Android-Benutzern ein Design zur Verfügung gestellt wird, welches sich der nativen Plattform annähert. Die app-ähnlichen Komponenten stellen auch für andere Plattformen eine ausreichend gute Benutzererfahrung sicher.

5.2 Technologie-Auswahl: Backend

Express[15] ist ein *Web-Framework* zur Entwicklung von HTTP APIs und damit auch REST APIs. Nach Voss (2019) gilt Express als Industriestandard. Der minimalistische Funktionsumfang von Express kann durch *Middlewares* erweitert werden (vgl. Express Contributors (o. J.b)). Diese werden sowohl von dem Express-Team selbst als auch von der großen Express-Community entwickelt (vgl. Express Contributors (o. J.a)).

[12] https://onsen.io/
[13] https://www.npmjs.com/package/@ionic/react
[14] https://material-ui.com/
[15] http://expressjs.com/

Eine eingehende Anfrage durch den Client wird in Express durch eine Sequenz von Middleware-Aufrufen bearbeitet. Middlewares sind dabei Funktionen, welche Zugriff auf die Anfrage und auf die zu sendende Antwort haben. Die aktuelle Middleware ruft die nächste Middleware auf, es sei denn, die aktuelle Middleware sendet vorzeitig eine Antwort. Dies ist etwa bei einer fehlgeschlagenen Authentifizierung der Fall (vgl. Abbildung 5.2). Anstatt die Anfrage durch folgende Middleware bearbeiten zu lassen, sendet die Middleware beispielsweise eine Antwort mit dem Statuscode 401 („Unauthorized“) an den Client. (vgl. Express Contributors (o. J.b))

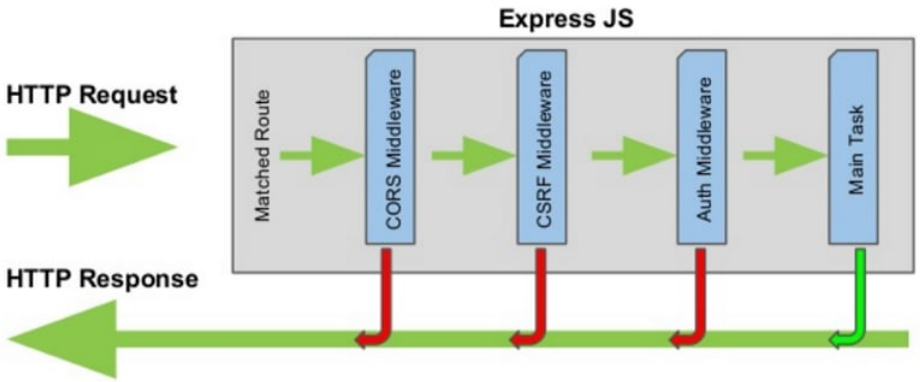

Abbildung 5.2: Middlewares in Express
(Quelle: Kononenko (2018))

5.3 Technologie-Auswahl: Datenbank

Bei der Auswahl eines Datenbank-Management-Systems (DBMS) wurde zwischen relationalen Datenbanken und NoSQL-Datenbanken („Not only SQL“) abgewogen. Unter den Begriff NoSQL fallen verschiedene Arten von Datenbanken wie Dokument-Datenbanken, Key-Value-Datenbanken, Graph-Datenbanken, Suchdatenbanken und Zeitreihen-Datenbanken (vgl. solid IT GmbH (2020)). Insbesondere Dokument-Datenbanken, welche Dokumente beispielsweise im JSON-Format verwalten, bieten einen interessanten Ansatz für das airact-System. Im Folgenden werden relationale Datenbanken im Allgemeinen mit der verbreitetsten Dokument-Datenbank MongoDB[16] verglichen (vgl. solid IT GmbH (2020)).

Dokument-Datenbanken sind schemalos und bieten somit eine hohe Flexibilität sowie die

[16] https://www.mongodb.com/

Möglichkeit, unstrukturierte Daten zu speichern (vgl. Parker u. a. (2013)). Ohne ein festes Schema besteht jedoch die Gefahr, dass invalide Daten in die Datenbank geschrieben werden. In MongoDB-Datenbanken kann dieser Gefahr mit Validierungen des Schemas vorgebeugt werden (vgl. MongoDB, Inc. (o. J.b)).

Datenmodelle in *relationalen Datenbanken* werden typischerweise durch eine *Normalisierung* in mehrere Tabellen aufgeteilt, sodass Redundanzen vermieden werden. Bei einer Abfrage werden resultierende Tabellen durch Joins wieder zusammengeführt. (vgl. Parker u. a. (2013))

Dokument-Datenbanken hingegen aggregieren die Daten einer Entität in einem Dokument. Dies erhöht die Übersichtlichkeit einzelner Datensätze. Hierbei sind allerdings Redundanzen möglich, die u. a. bei Aktualisierungen zu inkonsistenten Zuständen führen können. Auch in MongoDB-Datenbanken können inzwischen Beziehungen zwischen Entitäten durch sogenannte *Lookups* umgesetzt werden (vgl. MongoDB, Inc. (o. J.a)). Während relationale Datenbanken verschiedene Arten von Joins ermöglichen, sind durch *Lookups* jedoch nur Verbünde möglich, die einem LEFT-JOIN entsprechen (vgl. Refsnes Data (o. J.)).

Dokument-Datenbanken haben den Vorteil, besser skalierbar zu sein. Dadurch, dass Datensätze zum Teil im Arbeitsspeicher gehalten werden, sind Dokument-Datenbanken zudem performanter – zumindest bei simplen Abfragen. (vgl. Parker u. a. (2013))

Zur Speicherung eines redundanzfreien Datenbestandes innerhalb des airact-Prototyps sind keine Beziehungen zwischen den Entitäten notwendig. Ein Abonnement für Push-Notifications kann als Liste von IDs der Reiseklassen und der Reiseziele umgesetzt werden. Sowohl die Inhalte, die Reiseklassen als auch die Reiseziele sollen von WordPress verwaltet werden, da sie eine hohe Kohäsion besitzen. Durch die separaten Datenbanken von WordPress und dem Backend ließen sich ohnehin keine Beziehungen zwischen den Abonnements und den dazugehörigen Reiseklasse und Reisezielen umsetzen.

In späteren Versionen der PWA sind jedoch neue Funktionen wie Flug-Buchungen oder personalisierte Werbe-Aktionen denkbar, welche komplexe relationale Daten erfordern, für die relationale Datenbanken besser geeignet sind. Sollten zu einem späteren Zeitpunkt unstrukturierte Daten gespeichert werden müssen, kann dies entweder im begrenzten Maße durch Attribute des Typs JSON oder durch die Nutzung eines hybriden Datenbank-Setups mit sowohl relationalen Datenbanken als auch Dokument-Datenbanken geschehen.

Es wurde sich daher für eine relationale Datenbank entschieden. Gewählt wurde eine PostgreSQL[17]-Datenbank, da diese einen größeren Funktionsumfang als der verbreitetste Konkurrent MySQL[18] bietet und zudem ACID-konformer ist. Entsprechend konfiguriert ist auch die Performanz ähnlich zu einer MySQL-Datenbank. (vgl. Baclit und Sicam (2009), S. 272)

Während PostgreSQL open-source ist, gilt dieser ehemals zutreffende Status bei MySQL als gefährdet. Mit der Übernahme durch Oracle stehen Teile von MySQL nur noch closed-source bereit, weshalb auch Forks wie MariaDB[19] entstanden. (vgl. Endsley (2013))

Zur *objektrelationalen Abbildung* (object-relational mapping, ORM) in Node.js wird das Paket sequelize genutzt. Die dort definierten Modelle werden mithilfe des Pakets sequelize-typescript um deklarative Annotationen ergänzt.

5.4 Technologie-Auswahl: DevOps

Build, Test und Bereitstellung werden mithilfe des Cloud-Computings durchgeführt. In der Cloud gehostete Systeme setzen weniger administrativen Aufwand voraus als etwa bei einem dedizierten Bare-Metal-Server. Hardware-Ressourcen in der Cloud können – zum Teil automatisch – angepasst werden, je nachdem, wie viel Rechenkapazität benötigt wird. (vgl. National Institute of Standards and Technology (2011))

5.4.1 Bereitstellung des Systems

Das airact-System wird primär in einem Kubernetes-Cluster in der Google Cloud[20] bereitgestellt. Die Google Cloud bietet nach der Oracle Cloud[21] die günstigsten Preise für die Bereitstellung eines Kubernetes-Clusters und stellt zudem ein hohes Gratis-Kontingent bereit. Nach AWS[22] und Azure[23] ist die Google Cloud die dritthäufigst genutzte Cloud (vgl. Richter (2019)). Weiterhin bietet die Google Cloud einen großen Marketplace

[17] https://www.postgresql.org/
[18] https://www.mysql.com/de/
[19] https://mariadb.org/
[20] https://cloud.google.com/
[21] https://www.oracle.com/cloud/
[22] https://aws.amazon.com/
[23] https://azure.microsoft.com/

für vorkonfigurierte Anwendungen (sogenannte *Click-to-Deploy*-Lösungen) und umfangreiche Dashboards zur Administration. (vgl. Vasi (2019))

Die WordPress-Installation wird innerhalb des Kubernetes-Clusters durch eine *Click-to-Deploy*-Lösung bereitgestellt. Auch wenn die Lösung eine MySQL-Datenbank für WordPress zur Verfügung stellt, wird in der Produktionsumgebung der Dienst *Cloud SQL* der Google Cloud genutzt. Vorteile liegen hierbei in der verbesserten Administrationsoberfläche, welche automatische Backups und die Erstellung von Lesereplikaten ermöglicht. Dieser Dienst wird ebenfalls für die Backend-Datenbank (PostgreSQL) des airact-Systems genutzt.

5.4.2 Automatisierung von Build- und Testprozessen

Zur kontinuierlichen Integration nutzt das airact-System den Dienst *Cloud Build* der Google Cloud. Cloud Build ermöglicht eine Automatisierung von Build-, Test- und Bereitstellungsprozessen mithilfe von Pipelines. Dafür werden die Quellcode-Repositories des airact-Systems zuerst mit dem Dienst verbunden. Anschließend werden *Trigger* erstellt, welche bei Push-Operationen und Pull-Requests in dem zugehörigen Repository ausgelöst werden. Diese stoßen dann die Ausführung der Befehle für Build, Test und/oder Bereitstellung an, welche innerhalb des Repositories in einer Build-Konfiguration (YAML-Datei) definiert sind. Es stehen dabei diverse Befehle (sogenannte *Cloud-Builder*) bereit, welche innerhalb eines Docker-Containers ausgeführt werden. Innerhalb der Build-Konfiguration können Variablen deklariert werden, deren Werte innerhalb des Triggers gesetzt werden können. (vgl. Google LLC (2019a))

In Abschnitt 7.1 wird die Nutzung der Cloud Build Pipelines in dem airact-System beschrieben. Listing 7 zeigt eine exemplarische Build-Konfiguration.

```
steps:
# build image
- name: 'gcr.io/cloud-builders/docker'
  args: ['build', '-t', '${_IMAGE_NAME}:latest-${_IMAGE_ENVIRONMENT}', '.']

# push image
- name: 'gcr.io/cloud-builders/docker'
  args: ['push', '${_IMAGE_NAME}:latest-${_IMAGE_ENVIRONMENT}']

# trigger intergration testing using the Cloud Build REST API
- name: 'gcr.io/cloud-builders/curl'
  args: ...
```

```
13
14  substitutions:
15    _IMAGE_ENVIRONMENT: '' # set by trigger config (production/staging/development)
16    _IMAGE_NAME: 'gcr.io/airact/airact-frontend'
17
18  images:
19  - '${_IMAGE_NAME}:latest-${_IMAGE_ENVIRONMENT}'
```

Listing 7: Beispiel einer Build-Konfiguration (Cloud Build)

5.4.3 Logging, Monitoring und Alerting

Monitoring bezeichnet das Erheben, Speichern und Verarbeiten von Metriken. Metriken, wie etwa die CPU-Nutzung eines Containers, werden durch ein Monitoring-System abgefragt oder von sogenannten Agents an dieses gesendet. Die erhobenen Metriken werden dann gespeichert, womit die Möglichkeit zur Auswertung besteht. So kann beispielsweise der zeitliche Verlauf einer Metrik als Graph dargestellt werden. Monitoring ermöglicht einen Einblick in das System, um etwaige Fehlerwirkungen zu erkennen oder Grundlagen für Kapazitätsplanungen zu bieten. (vgl. Bass u. a. (2015), S. 127ff)

Beim Logging hingegen werden auftretende *Ereignisse* erfasst. Dies können beispielsweise Fehler oder Zustandsänderungen innerhalb einer Anwendung sein. Neben dem Zeitpunkt der Erfassung und einer Beschreibung des Ereignisses werden hier meist auch kontextuelle Informationen bereitgestellt, wie etwa der auslösende Prozess. Mithilfe der entstehenden Logs können Fehlerwirkungen erkannt und Fehlerzustände lokalisiert werden. (vgl. Bass u. a. (2015), S. 140)

Unter Alerting versteht man das Senden von Alarm-Benachrichtigungen aufgrund kritischer Systemzustände. Dies kann etwa der Fall sein, wenn ein Webserver nicht mehr antwortet oder Metriken einen gewissen Grenzwert überschreiten bzw. eine unerwartet hohe Steigung besitzen. Die Schwellenwerte werden dabei idealerweise so definiert, dass weder unnötige Benachrichtigungen gesendet werden noch das Senden relevanter Benachrichtigungen ausbleibt. (vgl. Bass u. a. (2015), S. 141f)

Zur Umsetzung dieser drei Konzepte wird der Dienst *Stackdriver* der Google Cloud genutzt. Stackdriver unterstützt das Monitoring und Logging für alle genutzten Ressourcen, inkl. der Ressourcen innerhalb des Kubernetes-Clusters (beispielsweise einzelne Pods). Auch ein Alerting auf diversen Kanälen wie etwa Slack ist mit Stackdriver möglich.

5.5 Kontextabgrenzung

Starke definiert die Kontextabgrenzung wie folgt:

> „Die Kontextabgrenzung zeigt das Umfeld eines Systems sowie dessen Zusammenhang mit seiner Umwelt. [...] Sie zeigt das System als Blackbox sowie dessen Verbindungen und Schnittstellen zur Umwelt.“ (Starke (2015), S. 160)

Der entwickelte Prototyp interagiert mit keinen externen Diensten. In einer späteren Version der Anwendung ist jedoch eine Integration mit externen Diensten und Systemen wahrscheinlich. Dazu gehören CRM-Systeme, Marketing-Systeme (etwa für E-Mail- oder Affiliate-Marketing) und Systeme zur Automatisierung von Arbeitsabläufen.

Benutzergruppen, die mit dem Prototyp interagieren, sind Leser, Autoren und Redakteure. Während Autoren und Redakteure Inhalte bereitstellen und bearbeiten, konsumieren Leser entsprechende Inhalte. Zusätzlich verwalten registrierte Leser ihre Benutzerdaten. Abbildung 5.3 stellt die Kontextabgrenzung des Prototyps grafisch dar. Die Pfeile zeigen dabei den Datenfluss (vgl. Starke (2015), S. 161).

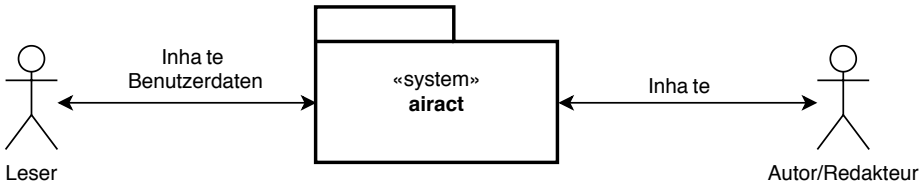

Abbildung 5.3: Kontextabgrenzung
(Quelle: eigene Darstellung, erstellt mit draw.io)

5.6 Bausteinsicht

Die Bausteinsicht stellt die Struktur der Bausteine des Systems dar und zeigt, welche Zusammenhänge zwischen den Bausteinen bestehen. Bausteinsichten werden top-down über mehrere *Level* hinweg verfeinert. Ausgehend von der Kontextabgrenzung (vgl. Abschnitt 5.5) werden dabei Inhalte der Blackboxes aus dem vorherigen Level dargestellt. Die Kontextabgrenzung wird auch als „Level 0“-Bausteinsicht bezeichnet und die darauffolgenden Level jeweils um eins erhöht. (vgl. Starke (2015), S. 163f)

Das System besteht aus drei Teilsystemen: dem Teilsystem `airact-wordpress` zur Verwaltung von Inhalten, dem Teilsystem `airact-frontend` als Anwendung für die

Benutzer und dem Teilsystem `airact-backend`, welches die Anwendungslogik des Systems beinhaltet (vgl. Abbildung 5.4).

Theoretisch wäre die Umsetzung der Anwendungslogik innerhalb des Systems `airact-wordpress` möglich gewesen. Die Aufteilung in die Teilsysteme `airact-wordpress` und `airact-backend` erfolgte, um die Komplexität des Teilsystems `airact-wordpress` zu reduzieren und die Möglichkeit zu bieten, das Backend mit nicht-monolithischen Architekturansätzen zu entwickeln. Das entkoppelte Backend lässt sich durch die Unabhängigkeit von einer WordPress-Installation besser testen, skalieren und in anderen Kontexten wiederverwenden.

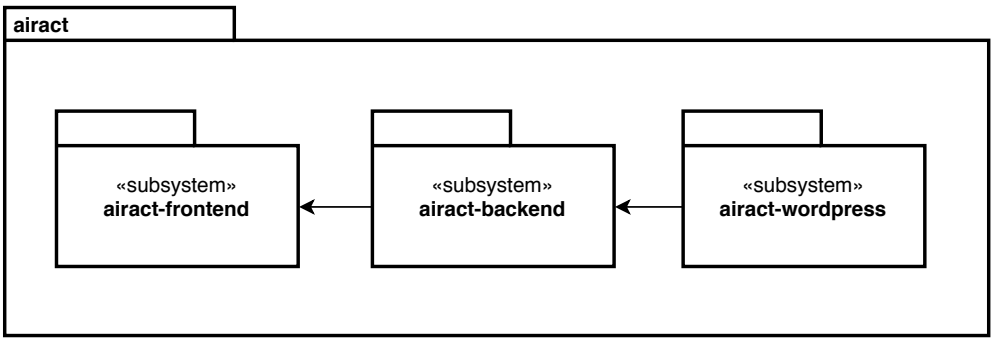

Abbildung 5.4: Bausteinsicht `airact` (Level 1)
(Quelle: eigene Darstellung, erstellt mit draw.io)

Durch die REST API von WordPress stehen die meisten benötigten Schnittstellen für das Backend bereit. Ergänzt werden diese durch das eigens entwickelte Plugin `airact-wordpress-plugin`, welches durch das Plugin `github-updater`[24] bereitgestellt und aktualisiert wird. Das Plugin `airact-wordpress-plugin` bietet zusätzlich eine Schnittstelle, um Test-Daten für Testumgebungen zu erzeugen. Um Benutzer durch Tokens über die REST API zu authentifizieren, wird das Plugin `jwt-authentication-for-wp-rest-api`[25] verwendet. (vgl. Abbildung 5.5)

Das Backend (`airact-backend`) wird mithilfe einer Microservice-Architektur in eine Menge leichtgewichtiger Services unterteilt. Im Gegensatz zu monolithischen Systemen bietet diese Architektur den Vorteil, Änderungen schneller bereitstellen zu können, insbesondere im Zusammenspiel mit entsprechenden automatisierten Prozessen. (vgl. Lewis und Fowler (2014))

[24] https://github.com/afragen/github-updater
[25] https://de.wordpress.org/plugins/jwt-authentication-for-wp-rest-api/

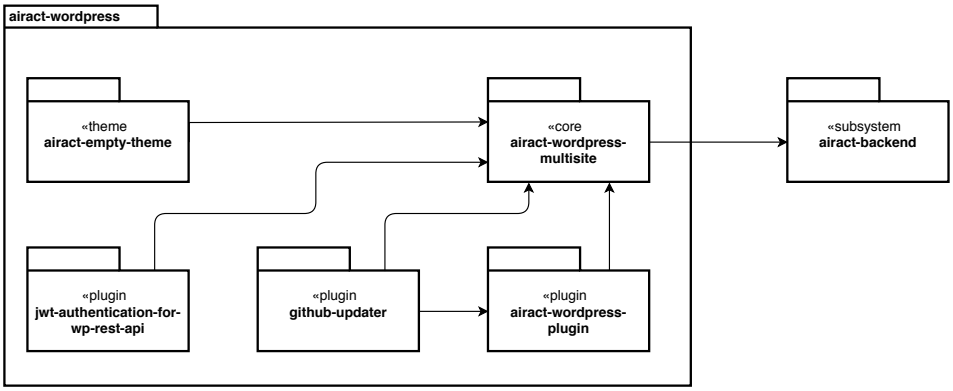

Abbildung 5.5: Bausteinsicht `airact-wordpress` (Level 2)
(Quelle: eigene Darstellung, erstellt mit draw.io)

Durch die klare Verteilung von Zuständigkeiten ist es einfacher, eine modulare Struktur aufzubauen und zu erhalten. Einfache Schnittstellen sorgen dafür, dass die Kopplung zwischen den Services gering gehalten wird und ein fragiles System weniger wahrscheinlich ist. Die isolierten Services ermöglichen zudem eine bessere Skalierbarkeit. (vgl. Lewis und Fowler (2014))

Die Kommunikation zwischen dem Backend und den Teilsystemen `airact-wordpress` und `airact-frontend` erfolgt ausschließlich über REST APIs. Während die Services untereinander direkt kommunizieren und auch eine direkte Kommunikation mit dem WordPress-Teilsystem erfolgt, kommuniziert das Frontend über ein API-Gateway mit den Services. Das API-Gateway bietet einen zentralen Zugriffspunkt für das Frontend, welches Anfragen an die entsprechenden Services weiterleitet.

Wie auch die Services, wird das API-Gateway mit dem Express-Framework entwickelt. Durch entsprechende Middlewares können Funktionalitäten wie CORS[26], Authentifizierung, Caching und Durchsatzbegrenzungen umgesetzt werden. Die Umsetzung innerhalb des API-Gateways bietet dabei den Vorteil, dass die Komplexität der Microservices nicht unnötig erhöht wird. (vgl. Márton (2017))

Durch einen Spring Cloud Config Server[27] werden Konfigurationen zentralisiert verwaltet und bereitgestellt. Definiert werden die Konfigurationen in einem GitHub-Repository, welches von dem Spring Cloud Config Server abgefragt wird. Dieser lässt dabei eine

[26] Das *Cross-Origin Resource Sharing* (CORS) ermöglicht den Zugriff auf Ressourcen eines Servers, ausgehend von anderen Servern. Durch Zugriffsregeln kann definiert werden, welche Server auf entsprechende Ressourcen zugreifen dürfen.
[27] https://cloud.spring.io/spring-cloud-config/reference/html/

Unterteilung der Konfigurationen für unterschiedliche Anwendungen, jeweils mit Profilen für verschiedene Ausführungsumgebungen, zu. In nachfolgenden Teilen dieser Arbeit wird der Spring Cloud Config Server als `airact-configuration-service` bezeichnet.

Das Backend besteht aus folgenden Microservices (vgl. Abbildung 5.6):

- `airact-content-service`: Dient dem Abfragen von Inhalten, welche durch das Teilssystem `airact-wordpress` verwaltet werden. Dieser Service bietet eine einheitliche Schnittstelle, mit der auch die Vorschau von geänderten Inhalten angefragt werden kann. Die Antworten werden in einer minimalen Form bereitgestellt, womit unnötigter Datenverkehr verhindert wird.
- `airact-account-service`: Dient dem Abfragen von Benutzerdaten, welche durch das Teilssystem `airact-wordpress` verwaltet werden. Dabei wird neben der Authentifizierung auch das Abfragen von Berechtigungen und das Erstellen und Bearbeiten von Benutzerkonten ermöglicht.
- `airact-subscription-service`: Verwaltet Abonnements für Push-Benachrichtigungen und versendet entsprechende Benachrichtigungen, sobald im Teilssystem `airact-wordpress` ein neuer Inhalt veröffentlicht wurde. Gespeichert und abgefragt werden entsprechende Daten in der PostgreSQL-Datenbank.

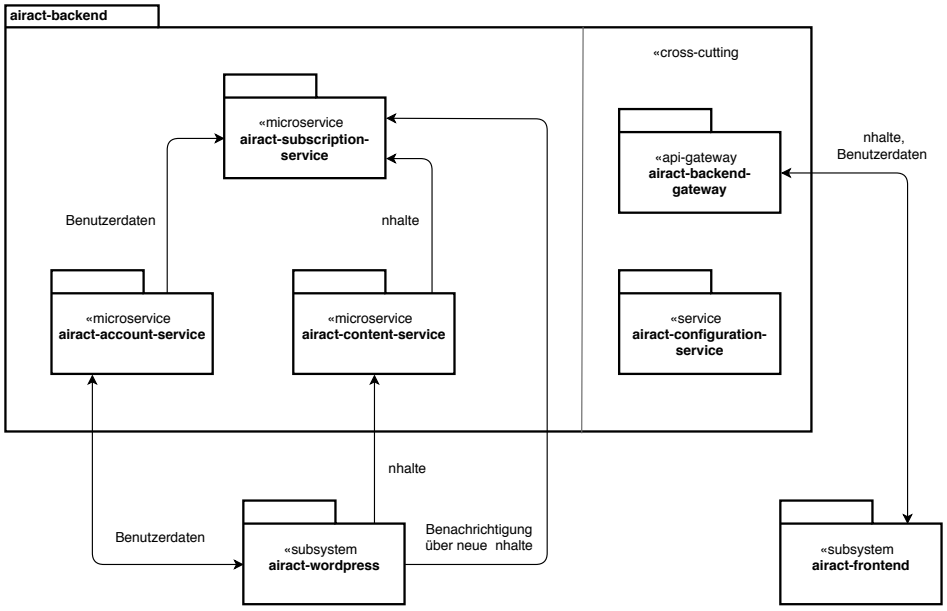

Abbildung 5.6: Bausteinsicht `airact-backend` (Level 2)
(Quelle: eigene Darstellung, erstellt mit draw.io)

5.6.1 Bausteinsicht Backend-Services

Alle Services besitzen eine Fassade, durch welche die REST API bereitgestellt wird. Die Fassade extrahiert und validiert dabei zuerst die Anfragedaten. Die Logik zur Verarbeitung der Anfrage befindet sich in Komponenten, die von der Fassade aufgerufen werden. Die Fassade verarbeitet die Antworten der Komponenten, indem sie diese in ein JSON-Format bringt und an den Client zurückliefert.

Die Komponenten können zur Verarbeitung der Anfrage auf andere Services zugreifen. Dafür werden entsprechende Clients definiert, welche wiederum die Daten der Services extrahieren und validieren.

Um die Implementierungen austauschbar zu machen, wird ein (Dependency-)Injection-Container genutzt, welcher die Abhängigkeiten verwaltet. Durch die Dependency-Injection werden Abhängigkeiten von konkreten Implementierungen entfernt. Dadurch können die Implementierungen ausgetauscht werden, ohne dass Anpassungen an dem Konsumenten notwendig sind. Dies ermöglicht auch eine Injektion von Test-Mocks, wodurch isolierte Tests möglich sind.

Abbildung 5.7 stellt den beschriebenen Aufbau grafisch dar. In diesem abstrakten Modell wurden Platzhalter für Komponenten und Service-Clients genutzt, die sich durch die Domäne ergeben. Die Verbindungen zwischen Paketen der UML-Notation bedeuten beispielsweise, dass eine beliebige Komponente mit einem beliebigen Service-Client kommuniziert. Entsprechende Notationen werden auch in darauf folgenden Abbildungen dieser Arbeit genutzt.

Die Komponenten eines Services bestehen aus einem Controller und einer Menge von (Datenbank-)Modellen mit dazugehörigen Repositories. Der Controller bietet eine Schnittstelle für die Fassade und greift in der Implementierung auf die Repositories zurück, welche eine Schnittstelle zur Datenbank bieten. Andere Services werden mithilfe der Service-Clients aufgerufen.

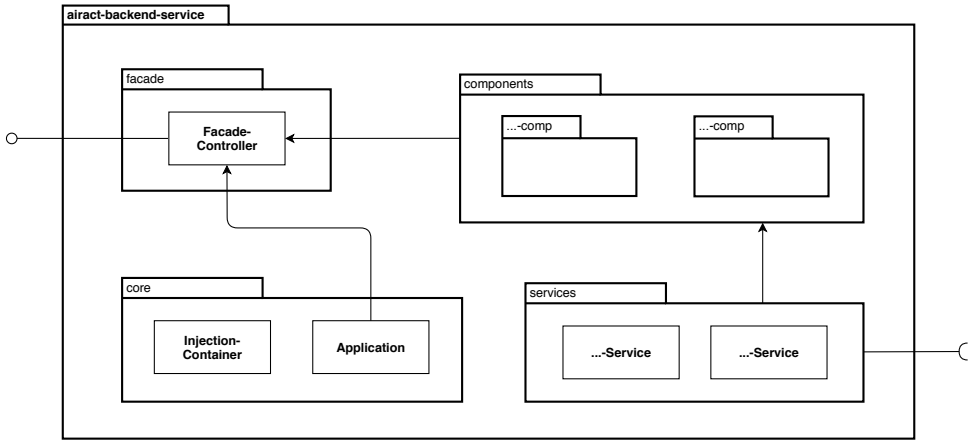

Abbildung 5.7: Bausteinsicht `airact-backend-service` (Level 3)
(Quelle: eigene Darstellung, erstellt mit draw.io)

5.6.2 Bausteinsicht Frontend

Die Grundarchitektur des Frontends orientiert sich an der des Backends. Die Fassade leitet Anfragen jedoch an das Next.js-Framework weiter, welches diese bearbeitet. Next.js organisiert eine Anwendung in Seiten, etwa der Login-Seite oder der Detailansicht eines Beitrages. Diese Seiten bestehen aus wiederverwertbaren React-Komponenten und stellen auch selbst solche dar. Die Komponenten, aus denen sich eine Seite zusammensetzt, werden wiederum durch vorgefertigte Komponenten der Material-UI-Bibliothek realisiert.

Zur Trennung von Anwendung und Repräsentation werden zusätzliche Komponenten definiert, welche die Frontend-Logik bearbeiten und dabei Anfragen an Clients für die Backend-Services stellen. Neben einem Controller beinhalten diese Komponenten der Anwendungslogik meist auch die Redux-Actions und -Reducer, welche zur Verwaltung des Zustands der Anwendung genutzt werden.

Zusätzlich werden statische Assets benötigt, welche von Next.js bereitgestellt werden. Dazu gehören Bilder und JavaScript-Dateien wie der Service-Worker der Anwendung.

Abbildung 5.8 stellt den beschriebenen Aufbau grafisch dar.

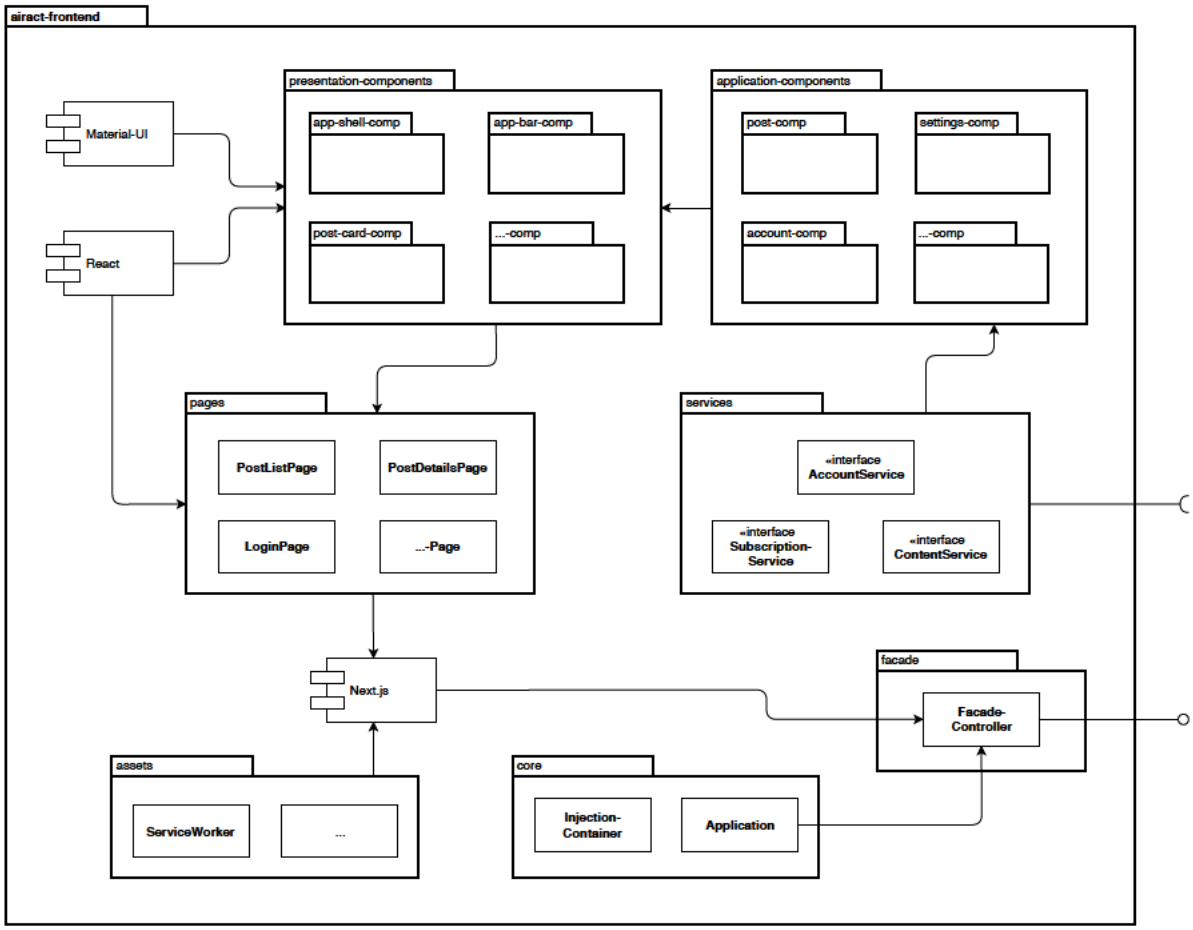

Abbildung 5.8: Bausteinsicht `airact-frontend` (Level 2)
(Quelle: eigene Darstellung, erstellt mit draw.io)

5.7 Laufzeitsicht

„Die Laufzeitsicht beschreibt, welche Bestandteile des Systems zur Laufzeit existieren und wie sie zusammenwirken“ (Starke (2015), S. 169). Die Laufzeitsicht kann mithilfe von UML-Sequenz-, Aktivitäts- oder Kollaborations-/Kommunikationsdiagrammen beschrieben werden (vgl. Starke (2015), S. 171). Entsprechende Diagramme eignen sich zur Darstellung von komplexen Sachverhalten.

Die Verwaltung von Push-Abonnements und das Versenden von Push-Benachrichtigungen stellt einen solchen komplexen Sachverhalt dar. Der Prozess besteht aus vier zeitlich versetzten Abläufen (vgl. Google LLC (o. J.e)):

- Sobald der Benutzer die Push-Benachrichtigungen in den Einstellungen seines Benutzerkontos aktiviert, wird der Benutzer aufgefordert, eine Berechtigung für den Erhalt solcher Benachrichtigungen zu erteilen. (vgl. Abbildung 5.9)
- Darauf erfolgt das Abonnieren des Push-Services, dem Teil des Browsers, welcher Push-Benachrichtigungen verwaltet. Ein solches Abonnement besteht dabei aus einer URL, zu welcher Push-Benachrichtigungen gesendet werden können und einem Public Key zur Verschlüsselung der Benachrichtigungen. Der Push-Service übernimmt die Darstellung auf dem Gerät des Benutzers. Da Abonnements flüchtig sind, müssen sie regelmäßig aktualisiert werden. Innerhalb des Subscript-Services werden die Abonnements empfangen und gespeichert. (vgl. Abbildung 5.10)
- Auch die angefragten Reiseziele werden durch den Subscription-Service verwaltet. Analog dazu findet auch die Verwaltung der angefragten Reiseklassen im Subscription-Service statt. (vgl. Abbildung 5.11)
- Sobald ein neuer Beitrag veröffentlicht wird, wird der Subscription-Service über die Veröffentlichung benachrichtigt. Dieser bestimmt alle Abonnements, die eine Übereinstimmung mit dem Reiseziel und der Reiseklasse des Beitrags haben. Anschließend wird die Benachrichtigung an die URL des Push-Services gesendet, sodass der Abonnent die Push-Benachrichtigung erhält. (vgl. Abbildung 5.12)

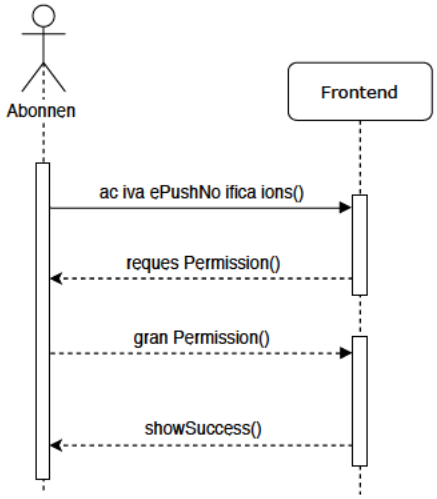

Abbildung 5.9: Aktivierung von Push-Benachrichtigungen
(Quelle: eigene Darstellung, erstellt mit draw.io)

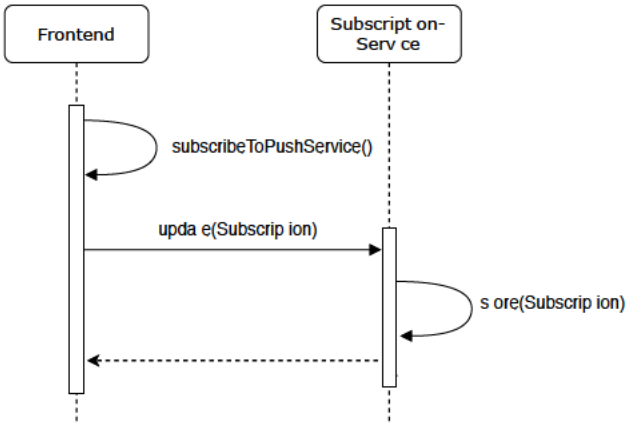

Abbildung 5.10: Abonnieren des Push-Services
(Quelle: eigene Darstellung, erstellt mit draw.io)

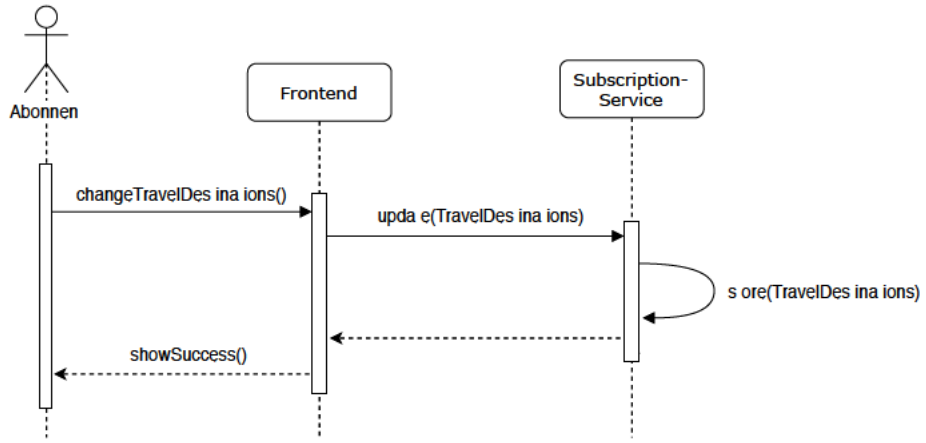

Abbildung 5.11: Änderung der angefragten Reiseziele
(Quelle: eigene Darstellung, erstellt mit draw.io)

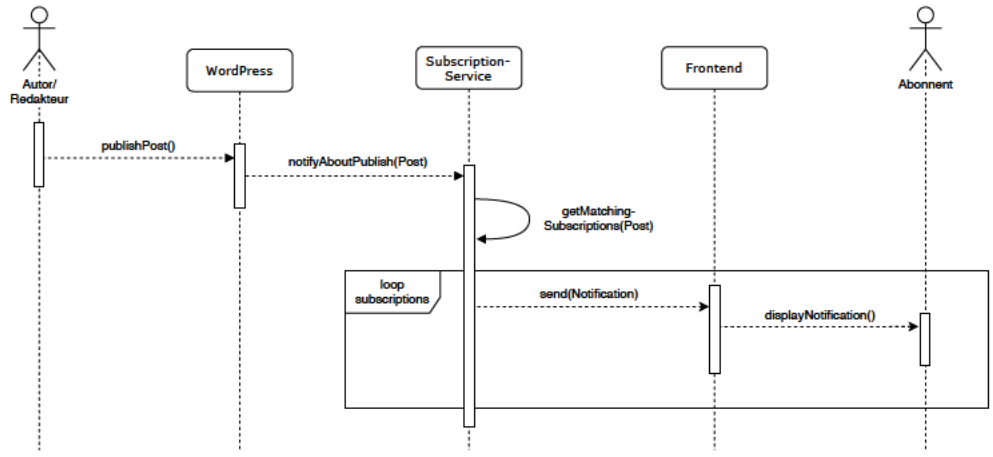

Abbildung 5.12: Senden von Push-Benachrichtigungen
(Quelle: eigene Darstellung, erstellt mit draw.io)

5.8 Verteilungssicht

Die Verteilungssicht soll „eine Landkarte der beteiligten Hardware und der externen Systeme sein“ (Starke (2015), S. 174). In Deployment-Diagrammen der UML-Notation können Knoten, Softwarekomponenten und physikalische Kanäle dargestellt werden. Abbildung 5.13 zeigt die Verteilungssicht des airact-Systems.

Wie bereits in der Technologie-Auswahl beschrieben, findet die Bereitstellung des Systems in einem Kubernetes-Cluster statt (vgl. Abschnitt 5.4). Zur Verarbeitung der Arbeitslast werden zwei Worker-Nodes genutzt, welche ausreichend Hardware-Ressourcen bereitstellen, auch wenn mehrere Namespaces genutzt werden. Vorgesehen ist dabei ein Namespace für die Produktionsumgebung und ein Namespace für die Staging-Umgebung.

Der `airact-configuration-service` bezieht die Konfigurationen aus einem auf GitHub gehosteten Repository. Bereitgestellt wird der `airact-configuration-service`, wie auch die WordPress-Installation und die Microservices, in dem Kubernetes-Cluster. Zum Bereitstellung der Datenbanken wird der Google Cloud SQL Service genutzt (vgl. Abschnitt 5.2).

Das API-Gateway des Backends, die WordPress-Installation und das Frontend werden durch einen Load-Balancer der Google Cloud Compute Engine für Zugriffe außerhalb des Kubernetes-Clusters veröffentlicht. Die dazugehörige statische IP-Adresse und das Let's Encrypt-SSL-Zertifikat werden ebenfalls durch die Google Cloud Compute Engine verwaltet. Die Domain (einschließlich der Subdomains) für das System wird durch das Webhosting von STRATO bereitgestellt. Die DNS-Einträge zeigen dabei auf die statische IP-Adresse des Load-Balancers.

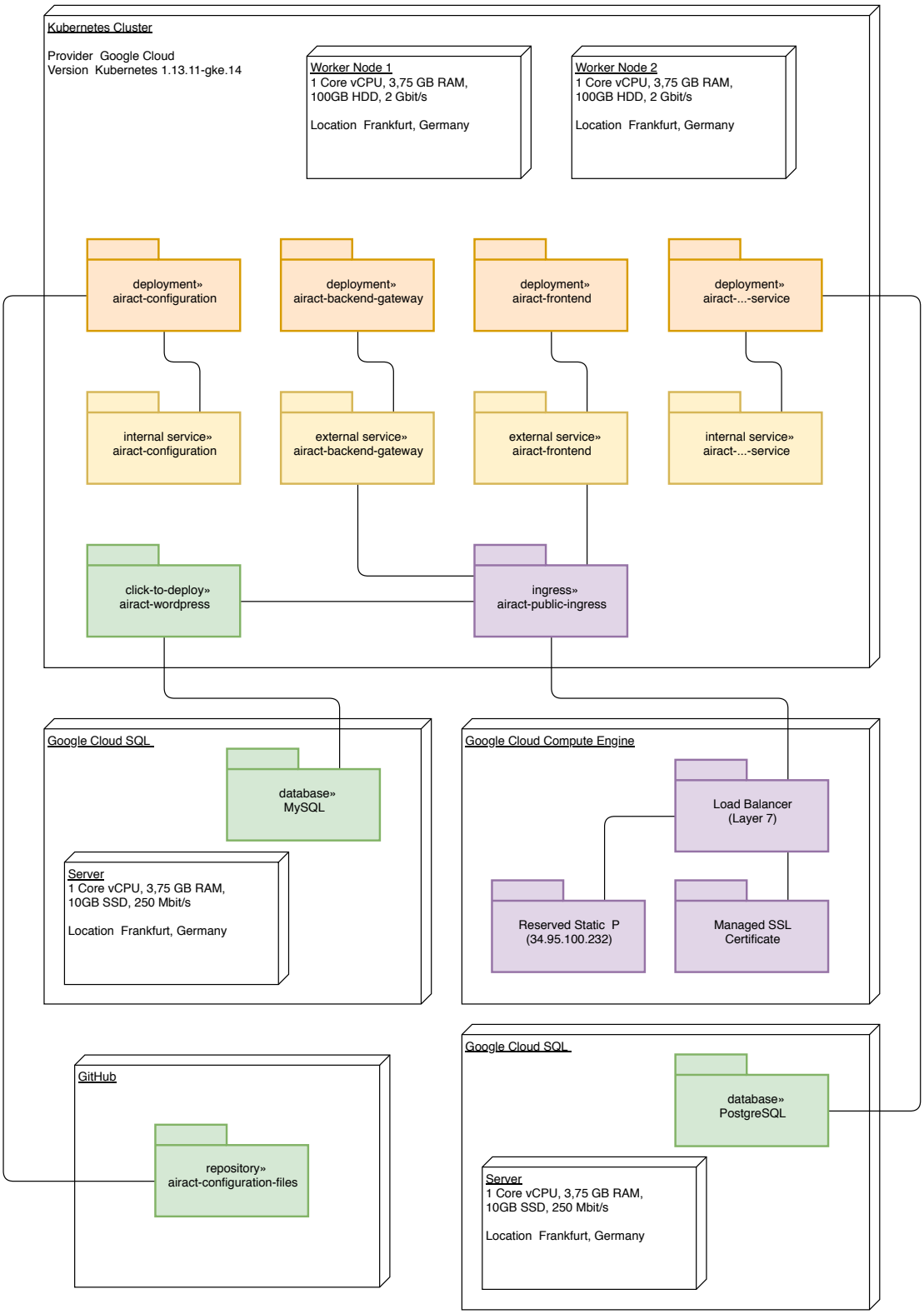

Abbildung 5.13: Verteilungssicht
(Quelle: eigene Darstellung, erstellt mit draw.io)

5.9 REST API

Die Abbildungen 5.14 – 5.16 stellen die REST API der einzelnen Services dar. Abbildung 5.17 zeigt die zusätzlichen Endpunkte der WordPress REST API, die durch das `airact-wordpress-plugin` bereitgestellt werden. Die Endpunkte des WordPress Cores werden in WordPress Foundation (o. J.o) beschrieben. Zusätzliche Endpunkte des Plugins zur Authentifizierung werden in Chavez (o. J.) dokumentiert.

Abbildung 5.14: REST API `airact-content-service` (Quelle: erstellt mit SwaggerHub)

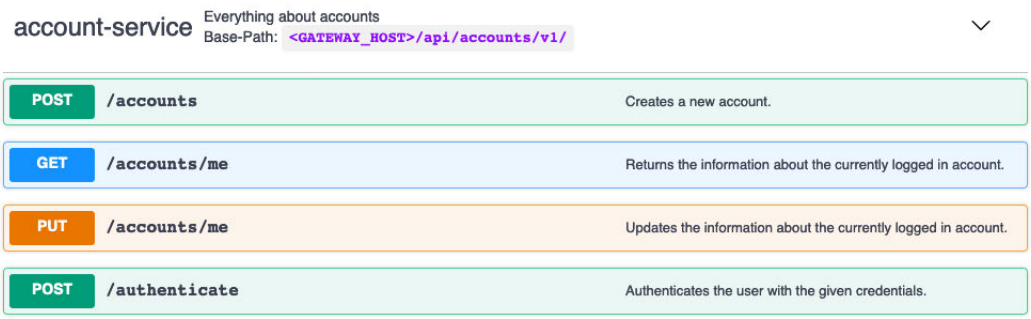

Abbildung 5.15: REST API `airact-account-service` (Quelle: erstellt mit SwaggerHub)

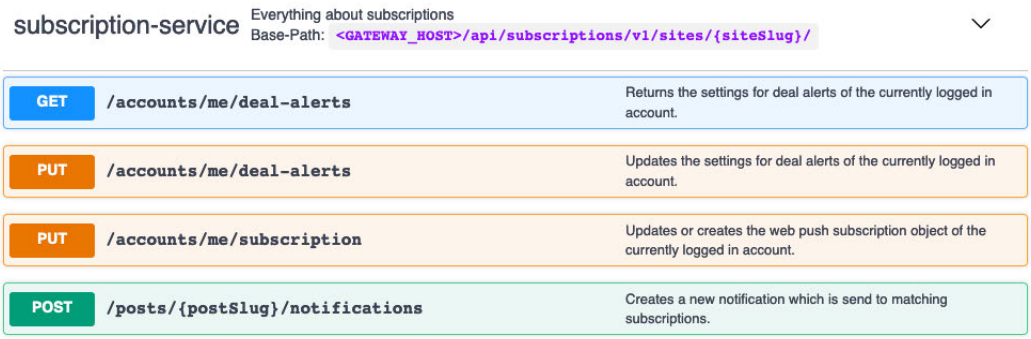

Abbildung 5.16: REST API `airact-subscription-service`
(Quelle: erstellt mit SwaggerHub)

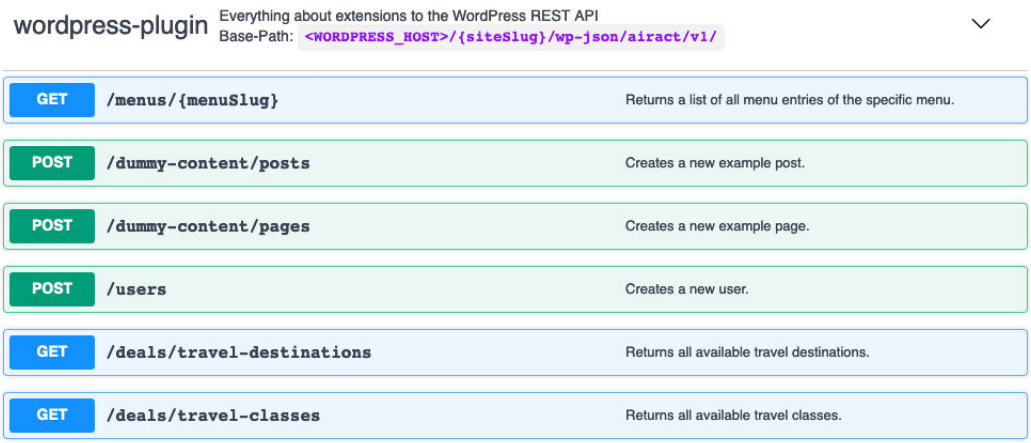

Abbildung 5.17: REST API `airact-wordpress-plugin`
(Quelle: erstellt mit SwaggerHub)

6 Umsetzung

In Abschnitt 6.1 wird das Vorgehen bei der Durchführung des Projektes beschrieben. Abschnitt 6.2 zeigt, wie die Versionierung des Systems erfolgte. Während der Implementierung wurden Werkzeuge zur Verbesserung der Code-Qualität eingesetzt, welche in Abschnitt 6.3 beschrieben werden. In Abschnitt 6.4 wird auf spezielle Implementierungsaspekte eingegangen.

6.1 Vorgehensmodell und Projektphasen

Aufgrund des Projektumfangs und fehlender Erfahrungen im Umgang mit den eingesetzten Technologien wurde ein iteratives Vorgehensmodell gewählt. Ludewig definiert die iterative Software-Entwicklung wie folgt:

> „Software wird in mehreren geplanten und kontrolliert durchgeführten Iterationsschritten entwickelt. Ziel dabei ist, dass in jedem Iterationsschritt – beginnend mit der zweiten Iteration – das vorhandene System auf der Basis der im Einsatz erkannten Mängel korrigiert und verbessert wird. Bei jedem Iterationsschritt werden die charakteristischen Tätigkeiten Analysieren, Entwerfen, Codieren und Testen durchgeführt.“ (Ludewig und Lichter (2013), S. 172)

Das Projekt wurde in mehrere Phasen unterteilt. In der ersten Phase erfolgte die grundlegende Einarbeitung in das Themengebiet sowie die Analyse und Spezifikation der Anforderungen. Anschließend wurden Vorprojekte für die Infrastruktur und für die Teilsysteme durchgeführt. Damit konnten Erfahrungen in den eingesetzten Technologien gesammelt werden und Grundlagen für den darauf folgenden Systementwurf geschaffen werden. Nach dem Entwurf des Systems in der ersten Iteration wurde dieses umgesetzt. Ergebnis hierbei war eine offline-fähige App, welche die funktionalen Anforderungen an das System

erfüllt. Nach der umfassenden Erarbeitung der theoretischen Grundlagen haben sich zusätzliche nicht-funktionale Anforderungen ergeben. Diese wurden in der zweiten Iteration umgesetzt, deren Kern die progressive Verbesserung und das universelle Rendering war. Abschließend wurde das System bezüglich der Anforderungen evaluiert. (vgl. Tabelle 6.1)

Lfd. Nr.	Bezeichnung der Phase	Zielsetzung der Phase
1	Vorrecherche	Skizzierung und Strukturierung der Problemstellung
2	Anforderungsanalyse	Erhebung und Spezifikation der Anforderungen
3	Vorprojekt (Infrastruktur)	Auswahl der eingesetzten Technologien, Aufbau einer groben Architektur und Erarbeitung von Entscheidungsgrundlagen für die Bereitstellung des Systems (vgl. Ludewig und Lichter (2013), S. 104ff)
4	Vorprojekt (Teilsysteme)	Auswahl der eingesetzten Technologien, Aufbau einer groben Architektur und Erarbeitung von Entscheidungsgrundlagen für die Implementierung des Systems (vgl. Ludewig und Lichter (2013), S. 104ff)
5	Entwurf (Iteration 1)	Entwurf und Dokumentation der Systemarchitektur
6	Durchführung (Iteration 1)	Implementierung des Systems
7	Literaturrecherche	Erarbeitung der notwendigen theoretischen Grundlagen
8	Entwurf (Iteration 2)	Überarbeiteter Entwurf mit Ergänzungen aus der Literaturrecherche
9	Umsetzung (Iteration 2)	Überarbeitetes System mit Ergänzungen aus der Literaturrecherche
10	Evaluation	Bewertung des Systems in Hinblick auf die Anforderungen

Tabelle 6.1: Phasen des Projektes

Zur Organisation der Arbeitspakete während der Phasendurchführung wurde ein Kanban-Board genutzt, umgesetzt mit der Anwendung Trello[1]. Das Kanban-Board visualisiert den Workflow durch eine Unterteilung der Arbeitspakete nach Bearbeitungszustand (vgl. Anderson (2010), S. 13ff). Dabei wurden die Bearbeitungszustände „Next“, „Active“, „Review“ und „Done“ definiert (vgl. Abbildung 6.1). Nach Abschluss einer Phase wurden alle abgeschlossenen Pakete durch eine entsprechende Funktion in Trello archiviert.

[1] https://trello.com/

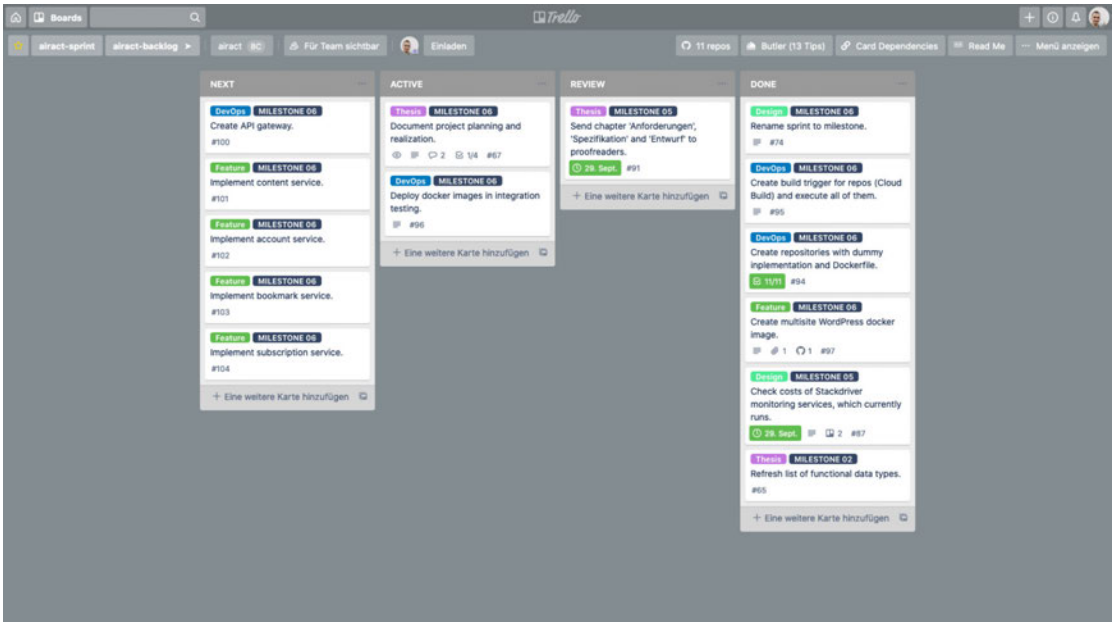

Abbildung 6.1: Kanban-Board
(Quelle: erstellt mit Trello)

Um die Arbeitspakete thematisch zu strukturieren, wurden diese unterschiedlichen Kategorien zugeordnet. Insbesondere durch den Umfang des Backlogs hat sich dies als nützlich erwiesen. Für das Backlog wurde ein eigenes Board erstellt (vgl. Abbildung 6.2). Die Arbeitspakete wurden zu Beginn einer neuen Phase in das Kanban-Board übertragen.

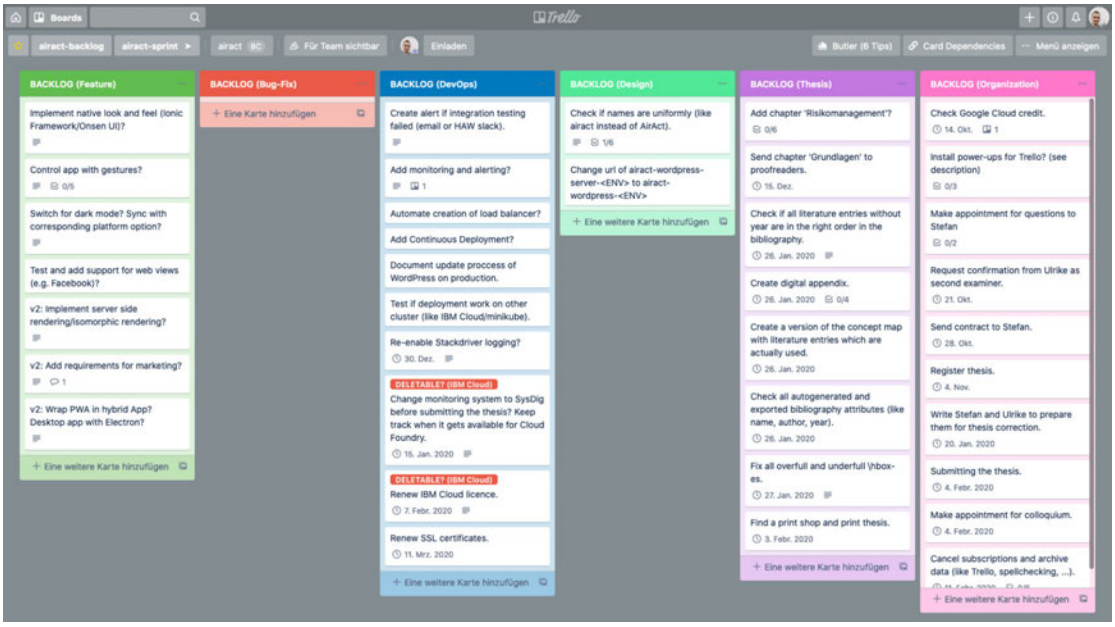

Abbildung 6.2: Backlog-Board
(Quelle: erstellt mit Trello)

6.2 Versionsverwaltung

Zur Versionsverwaltung wurde das Werkzeug Git[2] genutzt. Dabei wurde das Branching-Modell nach Driessen gewählt. Dieses gibt verschiedene Arten von Branches vor und definiert, wie diese zusammenhängen. (vgl. Driessen (2010))

Der `master`-Branch befindet sich stets in einem Zustand, der bereit für die Produktionsumgebung ist. Von diesem leitet sich der `develop`-Branch ab, welcher in den `master`-Branch gemergt werden kann, sobald sich der `develop`-Branch in einem stabilen Zustand befindet. (vgl. Driessen (2010))

Von dem `develop`-Branch gehen wiederum Feature-Branches ab, welche für die Entwicklung der einzelnen Features genutzt werden. Sobald die Entwicklung eines Features abgeschlossen ist, kann der Feature-Branch in den `develop`-Branch gemergt werden. (vgl. Driessen (2010))

Durch einen Release, d. h. durch das Mergen des `develop`-Branch in den `master`-Branch, können zeitkritische Fehlerbehebungen notwendig sein. Diese werden in einem

[2] https://git-scm.com/

Hotfix-Branch getätigt, welcher sich von dem neuen Zustand des `master`-Branches abzweigt und bei abgeschlossener Fehlerbehebung zurück in den `master`-Branch gemergt wird. (vgl. Driessen (2010))

Auf die Verwendung von Release-Branches wurde bei der Entwicklung des Prototyps der Einfachheit halber abgesehen. In einem größeren Projekt mit mehreren Releases sind entsprechende Branches jedoch empfehlenswert. (vgl. Driessen (2010))

6.3 Statische Codeanalyse

Statische Codeanalysen dienen der Verbesserung der Code-Qualität. Im Folgenden werden die dazu genutzten Werkzeuge beschrieben.

Prettier

Zur Formatierung des Quellcodes wird das Werkzeug Prettier[3]. Prettier bietet Unterstützung für alle genutzten Programmiersprachen. Eine Möglichkeit der Konfiguration von Prettier ist die Datei `.prettierrc` innerhalb des Repositories. Diese bietet den Vorteil, dass auch über mehrere Entwickler hinweg eine konsistente Formatierung des Quellcodes sichergestellt wird. Dabei stehen diverse Optionen zur Verfügung (vgl. Prettier Contributors (o. J.a)). (vgl. Prettier Contributors (o. J.b))

ESLint

Eine ergänzende statische Codeanalyse und -optimierung findet mithilfe von ESLint[4] statt. ESLint prüft den Quellcode auf Anomalien und Mängel, etwa auf ungenutzte Variablen. Zudem bietet ESLint eine Vielzahl von Plugins, u. a. für TypeScript, React und auch Prettier. Prettier übernimmt dabei die gesamte Verantwortung für die statische Codeanalyse und auch für die Formatierung des Quellcodes.

Konfigurationen können ähnlich zu Prettier in einer Konfigurationsdatei definiert werden, hier mit dem Namen `.eslintrc.json`. Durch Integrationen für Entwicklungsumgebungen wie VSCode[5] oder WebStorm[6] kann der Quellcode automatisch beim Speichern einer Datei analysiert, formatiert und optimiert werden (vgl. ESLint Contributors (o. J.)).

[3] https://prettier.io/
[4] https://eslint.org/
[5] https://code.visualstudio.com/
[6] https://www.jetbrains.com/webstorm/

6.4 Implementierung

Dieser Abschnitt zeigt spezielle Aspekte für die Implementierung des Systems.

6.4.1 Progressive Verbesserung

Listing 8 zeigt, am Beispiel einer React-Komponente für eine Drop-Down-Liste, die generelle Vorgehensweise zur Umsetzung der progressiven Verbesserung. Diese Komponente findet in der implementierten PWA Anwendung, da die Drop-Down-Liste aus der Bibliothek Material-UI bei deaktiviertem JavaScript nicht funktionsfähig ist.

Standardmäßig wird davon ausgegangen, dass der Browser des Benutzers kein JavaScript unterstützt. Dies wird durch einen Zustand namens `noScript` repräsentiert, welcher initial auf „wahr“ gesetzt wird (vgl. Zeile 10). Sofern der Browser JavaScript und alle benötigten Funktionalitäten unterstützt, wird dieser Zustand auf „falsch“ gesetzt. Dazu wird die React-Hook `useEffect` genutzt, welche clientseitig bei aktiviertem JavaScript ausgeführt wird, sobald die Komponente geladen wird (vgl. Zeile 17). Entsprechend des Zustandswertes wird dann die Anzeige der Komponente vorgenommen (vgl. Zeile 33 – 52).

Bei dem Absenden des Formulars, welches die Drop-Down-Liste beinhaltet, wird ein Ereignis-Handler registriert, welcher nur bei aktiviertem JavaScript ausgeführt wird und dabei prüft, ob alle benötigten Funktionalitäten unterstützt werden. Ist dies der Fall, wird die Bearbeitung des Absendens vorgenommen (vgl. Zeile 29). Werden die benötigten Funktionalitäten nicht unterstützt, wird stattdessen das Standard-Verhalten des Formular-Absendens ausgelöst. Dies sorgt dafür, dass die Seite neu geladen wird und der Wert der Drop-Down-Liste anschließend verarbeitet werden kann (vgl. Zeile 59).

```
1  import React, {useEffect, useState, SyntheticEvent} from 'react';
2  import {Select, MenuItem, Button} from '@material-ui/core';
3  import {useRouter} from 'next/router';
4
5  function handleSubmit(value: string) {
6      // TODO: Handle submit
7  }
8
9  function SelectComponent(props: {}): JSX.Element {
```

```
10      const [noScript, setNoScript] = useState(true);
11      const router = useRouter();
12
13      useEffect(() => {
14          // Disable no-script select if all features
15          // are available in browser
16          if ('someBrowserFeature' in navigator) {
17              setNoScript(false);
18          }
19      });
20
21      const handleSaveClick = (event: SyntheticEvent): void => {
22          // If all features are available in browser, we can
23          // skip form submitting (which reloads the page) by
24          // preventing the default behavior
25          if (!noScript) {
26              event.preventDefault();
27
28              const value = null; // TODO: Get value from select
29              handleSubmit(value);
30          }
31      };
32
33      return (
34          <form action={router.pathname} method="GET">
35              <Select style={noScript ? {display: 'none'} : {}}>
36                  <MenuItem value="value1">Value 1</MenuItem>
37                  <MenuItem value="value2">Value 2</MenuItem>
38              </Select>
39
40              <select
41                  style={!noScript ? {display: 'none'} : {}}
42                  name="submitted_Value"
43              >
44                  <option value="value1">Value 1</option>
45                  <option value="value2">Value 2</option>
46              </select>
47
48              <Button type="submit" onClick={handleSaveClick}>
49                  Save
50              </Button>
51          </form>
52      );
53  }
54
55  Page.getInitialProps = function (context) {
56      // Handle submitted form
57      if (context.router.query['submitted_Value']) {
58          const value = context.router.query['submitted_Value'] as string;
59          handleSubmit(value);
```

```
60          }
61      }
```

Listing 8: Progressive Verbesserung

6.4.2 Service-Worker

Der Service-Worker wird mithilfe der Bibliothek Workbox[7] umgesetzt. Workbox unterstützt verschiedene Caching-Strategien, welche auf Anfrage-Pfade angewendet werden können. Diese Pfade können mithilfe von Zeichenketten oder regulären Ausdrücken beschrieben werden.

Innerhalb des Service-Workers wird primär die „StaleWhileRevalidate“-Strategie genutzt. Bei dieser wird – sofern vorhanden – die angefragte Ressource aus dem Cache zurückgeliefert und parallel eine Netzwerk-Anfrage gestellt, mit deren Antwort der Cache für die nächste Anfrage der Ressource aktualisiert wird (vgl. Google LLC (o. J.l)). Listing 9 zeigt ein Beispiel für das „StaleWhileRevalidate“-Caching aller JavaScript-Ressourcen einer Anwendung.

```
1   const JS_CACHE_NAME = 'airact-cache-js-v1.0';
2
3   workbox.routing.registerRoute(
4       /\.js/,
5       new workbox.strategies.StaleWhileRevalidate({
6           cacheName: JS_CACHE_NAME,
7       })
8   );
```

Listing 9: Workbox Caching

6.4.3 Dependency Injection

Zur Umsetzung der Dependency Injection wird die Bibliothek inversify[8] genutzt. Listing 10 zeigt eine Implementierung anhand eines beispielhaften Services.

[7] https://developers.google.com/web/tools/workbox
[8] http://inversify.io/

Durch inversify wird ein Container für die Dependency Injection bereitgestellt, welcher es ermöglicht, konkrete Implementierungen an ein Bezeichner zu binden (vgl. Zeile 23 – 26). Als Bezeichner werden vordefinierte Symbole genutzt, da dieses Vorgehen weniger fehleranfällig als die Nutzung nicht-konstanter Zeichenketten ist (vgl. Zeile 32). Mit der `unbind`-Methode des Containers besteht die Möglichkeit, eine gebundene Implementierung auszutauschen, beispielsweise durch einen Test-Mock.

Die einzelnen Implementierungen werden mit einer `@injectable()`-Annotation versehen, um sie injizierbar zu machen. Durch die `@inject(<IDENTIFIER>)`-Annotation wird die Abhängigkeit in die konsumierende Klasse injiziert.

```
1  // file: src/main/services/example.ts
2  import {injectable} from 'inversify';
3
4  export interface ExampleService {
5      foo(): string;
6  }
7
8  @injectable()
9  export class ExampleServiceImpl implements ExampleService {
10     foo(): string {
11         return 'foo (implementation)';
12     }
13 }
14
15
16
17 // file: src/main/injection/container.ts
18 import {Container} from 'inversify';
19
20 import {InjectionTypes} from './types';
21 import {ExampleService, ExampleServiceImpl} from '../services/example';
22
23 const injectionContainer = new Container();
24 injectionContainer
25     .bind<ExampleService>(InjectionTypes.ExampleService)
26     .to(ExampleServiceImpl);
27
28
29
30 // file: src/main/injection/types.ts
31 export const InjectionTypes = {
32     ExampleService: Symbol.for('ExampleService'),
33 };
34
```

```
35
36
37  // file: src/main/.../some-consumer.ts
38  export class ExampleConsumer {
39      @inject(InjectionTypes.ExampleService) exampleService: ExampleService;
40
41      foo(): string {
42          const result = this.exampleService.foo();
43          return result.toUpperCase();
44      }
45  }
```

Listing 10: Dependency Injection

7 Test, Integration und Bereitstellung

Das Testen ist ein Teil des Softwareentwicklungsprozesses, welcher die Qualität des entwickelten Systems sicherstellen soll. Auch soll durch Testen geprüft werden, ob das System dem Verwendungszweck und den Bedürfnissen der Benutzer entspricht. Testprozesse sollten dabei bereits während der Entwicklung durchgeführt werden. (vgl. IEEE (2008))

Typischerweise werden zuerst die einzelnen Komponenten getestet („Unit-Tests“). Unter der anschließenden Integration versteht man die Verbindung der einzelnen Komponenten zu einem Ganzen. Durch Integrationstests wird geprüft, ob die Komponenten nach der Integration korrekt zusammenarbeiten. Integrationstests zielen dabei auf die Prüfung der Schnittstellen und der Interaktion zwischen den Komponenten ab. (vgl. Spillner u. a. (2011), S. 50)

Das Testen und die Integration des Systems findet automatisiert statt. Das Vorgehen wird in Abschnitt 7.1 beschrieben. Zur Bewertung des Systems wurden Last-, Performanz- und Stresstests durchgeführt, welche in Abschnitt 7.2 und 7.3 dargestellt werden. Zum Testen nicht-funktionaler Anforderungen werden verschiedene Auditing-Werkzeuge genutzt. Entsprechende Ergebnisse werden in Abschnitt 7.4 dargelegt. Abschnitt 7.5 zeigt, wie die Bereitstellung von neuen Versionen während der Entwicklung erfolgte.

7.1 Kontinuierliche Integration

Getestet und integriert wird das System durch eine kontinuierliche Integration (Continuous Integration). Nach Beck bedeutet dies, die Integration mehrmals täglich vorzunehmen und zwar jedes Mal, wenn eine Aufgabe abgeschlossen wurde (vgl. Beck (2000), S. 54). So können Fehler erkannt werden, direkt nachdem eine Änderung diese ausgelöst hat (vgl. Beck (2000), S. 60). Das System folgt dabei dem Vorschlag von Humble, den Integrationsprozess nach jedem Commit auszuführen, kombiniert mit einem automatischen Build-Prozess (vgl. Humble und Farley (2011), S. 55ff).

Die Integration orientiert sich an den in Grant (2018) gezeigten Vorgehensweisen, vereinfacht diese jedoch:

- Feature-Develop-Test: Änderungen an einem Feature-Branch werden mit den Develop-Branches der anderen Repositories integriert und anschließend getestet. Damit soll geprüft werden, ob der Feature-Branch bereits insoweit lauffähig ist, dass er in den Develop-Branch gemergt werden kann. Der Test wird dabei in einem temporären Namespace des Kubernetes-Clusters durchgeführt, benannt nach dem Muster `temp-<TIMESTAMP>`, etwa `temp-1397392146866`. Dadurch sollen Kollisionen durch parallel durchgeführte Tests vermieden werden.

- Develop-Master-Test: Änderungen an einem Develop-Branch werden mit den Master-Branches der anderen Repositories integriert und anschließend getestet. Damit soll sichergestellt werden, dass der Develop-Branch bereit für einen Merge in den Master-Branch ist. Der Test wird dabei in dem `staging`-Namespace innerhalb des Kubernetes-Clusters durchgeführt.

- Master-Master-Test: Änderungen an einem Master-Branch werden mit den Master-Branches der anderen Repositories integriert und anschließend getestet. Dieser Test wird ausgeführt, um sicherzustellen, dass alle Master-Branches korrekt zusammenarbeiten. Der Test wird dabei ebenfalls in dem `staging`-Namespace innerhalb des Kubernetes-Clusters durchgeführt.

Eine Integration gestaltet sich schwieriger, wenn Änderungen in mehreren Komponenten durchgeführt werden. Dabei kann beispielsweise eine Integration mit einem Feature-Branch anderer Komponenten notwendig sein (vgl. Grant (2018)). Es muss daher ein Weg gefunden werden, Abhängigkeiten zwischen den Branches verschiedener Repositories zu definieren. Da die Entwicklung des Prototyps als Ein-Mann-Projekt und mit im Vorfeld definierten Schnittstellen stattgefunden hat, bestand hierfür noch kein Bedarf. In einer späteren Weiterentwicklung bietet sich ein entsprechendes Vorgehen jedoch an, etwa indem Abhängigkeiten im `airact-configuration-service` in dem in Listing 11 gezeigten Format definiert werden.

```
[
  {
    "object": {
      "repo": "airact-frontend",
      "branch": "feature-new-post-format"
    },
    "relations": [
      {
        "repo": "airact-content-service",
        "branch": "feature-new-post-format",
      }
    ]
  }
]
```

Listing 11: Definition von Abhängigkeiten zwischen Repositories

Bei einem Commit wird unter Nutzung von Google Cloud Build das Docker-Image gebuildet. Dabei werden auch die Unit-Tests ausgeführt. Schlagen diese fehl, wird die Pipeline abgebrochen und eine Benachrichtung an einen Slack-Kanal gesendet (vgl. Google LLC (o. J.i)). Das entstandene Docker-Image wird in die private Docker-Registry gepusht. Anschließend wird mithilfe des Endpunktes `projects.triggers.run` der Cloud Build REST API[1] der Integrationstest getriggert. Die Testfälle werden in einem separaten Repository definiert. (vgl. Abbildung 7.1)

Wurde der Integrationstest getriggert, werden die gepushten Docker-Images in einem Kubernetes-Cluster bereitgestellt. Anschließend wird ein Docker-Image, welches Testfälle des Integrationstests beinhaltet, gebuildet und in die private Docker-Registry gepusht. Dieses wird dann innerhalb des Kubernetes-Clusters ausgeführt, womit auch Integrationstests möglich sind, die auf Cluster-interne Services zugreifen. Schlagen die Integrationstests fehl, wird die Pipeline als fehlgeschlagen gekennzeichnet und eine Benachrichtigung an einen Slack-Kanal gesendet (vgl. Google LLC (o. J.i)). (vgl. Abbildung 7.2)

[1] https://cloud.google.com/cloud-build/docs/api/reference/rest/v1/projects.triggers/run

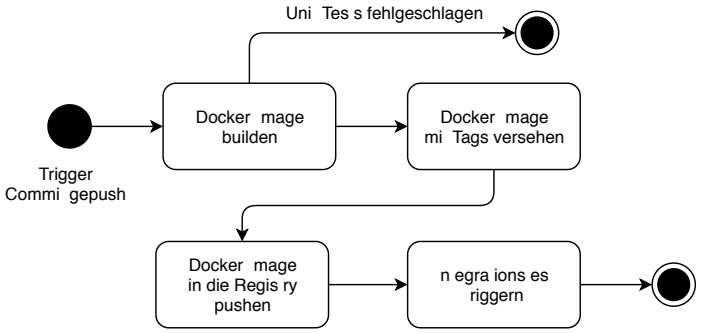

Abbildung 7.1: Kontinuierliche Integration: Build-Prozess
(Quelle: eigene Darstellung, erstellt mit draw.io)

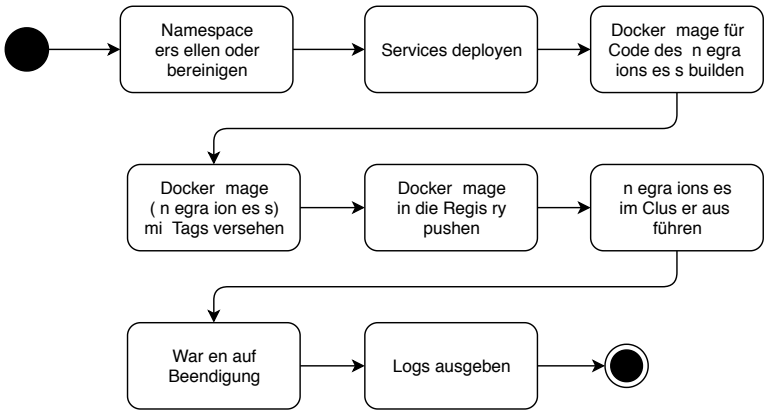

Abbildung 7.2: Kontinuierliche Integration: Integrationstest
(Quelle: eigene Darstellung, erstellt mit draw.io)

7.2 Performanztests

Zur Messung der Antwortzeiten wurden Performanztests mithilfe des Werkzeuges JMeter[2] durchgeführt. Dafür wurde ein Test mit einem virtuellen Benutzer (im Folgenden als Thread bezeichnet) durchgeführt, welcher mehrfach alle sechs Sekunden eine Anfrage sendet.

Die Tests haben gezeigt, dass die WordPress REST API sehr inperformant ist. Während einfache Anfragen (etwa die Abfrage der neusten 10 Beiträge) bereits etwa eine Sekunde benötigten, benötigten komplexere Anfragen (wie die Abfrage der neusten 10 Beiträge inkl. eingebetteter Ressourcen) knapp vier Sekunden. (vgl. Tabelle 7.1)

[2] http://jmeter.apache.org/

Service	Antwortzeit (Durchschnitt)	Antwortzeit (99%-Quantil)	CPU-Nutzung	RAM-Nutzung
WordPress (embed = false)	1184 ms	1231 ms	< 0,01 Kerne	100 MB
WordPress (embed = true)	3777 ms	3977 ms	< 0,01 Kerne	100 MB

Tabelle 7.1: Performanztests ohne Caching der WordPress REST API

Zur Erhöhung der Performanz können die Antworten der WordPress REST API mithilfe von Plugins wie WP REST Cache[3] gecacht werden. Dies führt dazu, dass gecachte Antworten meist innerhalb einer halben Sekunde bereitgestellt werden (vgl. Tabelle 7.2). Vorteil solcher Caching-Plugins innerhalb von WordPress ist, dass der Cache automatisch aktualisiert wird, sobald neue Inhalte veröffentlicht werden.

Service	Antwortzeit (Durchschnitt)	Antwortzeit (99%-Quantil)	CPU-Nutzung	RAM-Nutzung
WordPress (embed = false)	518 ms	568 ms	< 0,01 Kerne	100 MB
WordPress (embed = true)	548 ms	632 ms	< 0,01 Kerne	100 MB

Tabelle 7.2: Performanztests mit Caching der WordPress REST API

Da für die Darstellung einer Seite in der PWA mehrere Anfragen an die Backend-Services gestellt werden, welche wiederum an die WordPress REST API weitergeleitet werden, ist auch eine Antwortzeit von einer halben Sekunde kaum ausreichend. Insbesondere die durchgeführten Audits, beschrieben in Abschnitt 7.4, veranschaulichen dies. Lösungsansätze für diese Problemstellung werden im Rahmen dieser Arbeit im Ausblick dargestellt (vgl. Unterabschnitt 9.2.4).

7.3 Last- und Stresstests

Neben den Performanztests wurden die Teilsysteme einem Last- und Stresstest unterzogen. Die folgenden Tests wurden mit einer bestimmten Anzahl an Threads durchgeführt, welche alle sechs Sekunden eine Anfrage senden (je Thread insgesamt 100 Mal). Um Flaschenhälse zu kompensieren, wurden entsprechende Teilsysteme soweit repliziert, dass alle Anfragen der getesteten Teilsysteme problemlos verarbeitet werden können. Zwischen den Tests wurden die Pods neu gestartet. Die Tests wurden, wie die Performanztests, mit JMeter durchgeführt und gehen von einem virtuellen Server in Berlin aus, welcher

[3] https://wordpress.org/plugins/wp-rest-cache/

ausreichende Spezifikationen besitzt (8-Kern Prozessor, 8 GB Arbeitsspeicher und 1000 MBit/s Netzwerkanbindung).

Durch die Tests mit einer gecachten WordPress REST API haben sich folgende Lastgrenzen ergeben:

- WordPress REST API: etwa 350 Threads (vgl. Tabelle 7.3)
- Content-Service: etwa 650 Threads (vgl. Tabelle 7.4)
- Frontend: etwa 350 Threads (vgl. Tabelle 7.5)

Bei allen Teilsystemen führt eine höhere Anzahl an Threads zu erhöhten Antwortzeiten und zu einer erhöhten Fehlerquote. Durch den Load-Balancer treten bei Anfragen, die länger als 60 Sekunden benötigen, Timeouts auf, welche die Fehlerquote zusätzlich erhöhen. Mit entsprechenden Konfigurationen lässt sich die Grenze für einen Timeout jedoch anpassen. Erhöhte Fehlerquoten führen zu teilweise niedrigeren Antwortzeiten, da entsprechende Anfragen nicht bearbeitet werden können. Dies zeigt sich teilweise auch in der CPU-Nutzung.

Anzahl Threads	Antwortzeit (Durchschnitt)	Antwortzeit (99%-Quantil)	Fehlerrate	CPU-Nutzung	RAM-Nutzung
200	538 ms	594 ms	0,00 %	bis zu 0,15	bis zu 510 MB
300	549 ms	628 ms	0,00 %	bis zu 0,21	bis zu 670 MB
350	559 ms	755 ms	0,00 %	bis zu 0,26	bis zu 710 MB
400	5004 ms	13563 ms	13,96 %	bis zu 0,28	bis zu 720 MB
500	6926 ms	20452 ms	42,64 %	bis zu 0,29	bis zu 640 MB
1000	6294 ms	34571 ms	61,80 %	bis zu 0,36	bis zu 640 MB
2000	4855 ms	34568 ms	74,65 %	bis zu 0,50	bis zu 630 MB

Tabelle 7.3: Lasttest der WordPress REST API (gecacht)

Anzahl Threads	Antwortzeit (Durchschnitt)	Antwortzeit (99%-Quantil)	Fehlerrate	CPU-Nutzung	RAM-Nutzung
500	1165 ms	1658 ms	0,00 %	bis zu 0,58	bis zu 410 MB
600	1340 ms	2099 ms	0,00 %	bis zu 0,68	bis zu 400 MB
650	1449 ms	2693 ms	0,00 %	bis zu 0,66	bis zu 400 MB
700	13603 ms	29115 ms	1,37 %	bis zu 0,70	bis zu 550 MB
1000	14030 ms	43871 ms	2,32 %	bis zu 0,83	bis zu 580 MB
2000	44959 ms	60122 ms	62.45 %	bis zu 0,84	bis zu 800 MB
3000	56598 ms	60164 ms	93,89 %	bis zu 0,91	bis zu 2000 MB

Tabelle 7.4: Lasttest des Content-Services

Anzahl Threads	Antwortzeit (Durchschnitt)	Antwortzeit (99%-Quantil)	Fehlerrate	CPU-Nutzung	RAM-Nutzung
200	4225 ms	4730 ms	0,00 %	bis zu 0,32	bis zu 400 MB
300	4531 ms	5605 ms	0,00 %	bis zu 0,48	bis zu 430 MB
350	4955 ms	6373 ms	0,01 %	bis zu 0,56	bis zu 440 MB
400	10005 ms	16553 ms	1,63 %	bis zu 0,63	bis zu 570 MB
500	13593 ms	24294 ms	6,75 %	bis zu 0,62	bis zu 570 MB
1000	19990 ms	40975 ms	53,96 %	bis zu 0,63	bis zu 840 MB
2000	12205 ms	40972 ms	93,49 %	bis zu 0,46	bis zu 1180 MB

Tabelle 7.5: Lasttest des Frontends (nicht vorgerendert)

Zur Reduzierung der Last des Frontends können Antworten gecacht werden. Dies bietet sich insbesondere für nicht-eingeloggte Benutzer an, da hier keine dynamischen Daten angezeigt werden. Dabei muss jedoch sichergestellt werden, dass der Cache bei neuen und veränderten Inhalten aktualisiert wird.

Erste Tests ohne Synchronisation des Caches zeigen vielversprechende Ergebnisse. So sinken die Antwortzeiten auf ein Minimum und auch die Lastgrenze erhöht sich auf etwa 1250 Threads. (vgl. Tabelle 7.6).

Anzahl Threads	Antwortzeit (Durchschnitt)	Antwortzeit (99%-Quantil)	Fehlerrate	CPU-Nutzung	RAM-Nutzung
1000	108 ms	128 ms	0,00 %	bis zu 0,14	bis zu 270 MB
1200	106 ms	135 ms	0,00 %	bis zu 0,17	bis zu 280 MB
1250	104 ms	130 ms	0,07 %	bis zu 0,29	bis zu 280 MB
1300	101 ms	131 ms	3,59 %	bis zu 0,26	bis zu 290 MB
1500	94 ms	135 ms	15,33 %	bis zu 0,18	bis zu 260 MB
2000	74 ms	136 ms	34,51 %	bis zu 0,20	bis zu 320 MB
3000	265 ms	9066 ms	57,29 %	bis zu 0,20	bis zu 350 MB

Tabelle 7.6: Lasttest des Frontends (vorgerendert)

7.4 Auditing

Zur Bewertung der PWA wurden Audits durchgeführt. Für eine allgemeine Bewertung wurde das Werkzeug Lighthouse genutzt. Bei der Bewertung der Suchmaschinenoptimierung wurde das Werkzeug WooRank eingesetzt.

Das Lighthouse-Audit zeigt, dass die Performance der PWA mangelhaft ist (vgl. Anhang A.6). Dies liegt einerseits an dem universellen Rendering, welches die initiale Anfrage

serverseitig rendert (vgl. Unterabschnitt 5.1.2). Zwar wird dadurch der Inhalt schneller bereitgestellt, jedoch benötigt die PWA eine höhere Zeit, um interaktiv zu werden. Grund dafür ist die Verzögerung bei der Auslieferung des JavaScript-Codes durch das vorherige serverseitige Rendering. Anderseits führen die hohen Antwortzeiten der WordPress REST API dazu, das eine weitere starke Verzögerung auftritt. Lösungsansätze für beide Problemstellungen werden im Rahmen dieser Arbeit im Ausblick dargestellt (vgl. Unterabschnitt 9.2.4).

In den Bereichen „Best Practices“, „Suchmaschinenoptimierung“ und „PWA“ erfüllt die PWA hingegen alle Anforderungen. (vgl. Anhang A.6)

Da die Suchmaschinenoptimierung bei einem Blog ein wichtiges Thema darstellt, wurde dieser Aspekt genauer untersucht. Nach zielgerichteten Optimierungen zeigt auch das Audit von WooRank, dass die PWA suchmaschinenoptimiert ist (vgl. Anhang A.8).

Lediglich durch die Test-Daten der PWA werden einige Aspekte negativ bewertet. Durch den Platzhaltertext Lorem Ipsum wird als Sprache des Inhaltes Latein erkannt und weicht damit von der Sprache des `html`-Tags ab. Weiterhin werden die Copyright-Hinweise für Bilder durch die dicht aufeinander folgenden Links als nicht benutzerfreundlich eingestuft. In einer späteren Produktionsumgebung mit „echten“ Inhalten und entfernten Copyright-Hinweisen sind diese Mängel jedoch nicht mehr vorhanden.

7.5 Bereitstellung

Um neue Versionen auf der Produktionsumgebung bereitzustellen, wird das Rolling-Update von Kubernetes genutzt. Dabei wird die Änderung des Image-Tags vorausgesetzt. Bei der Nutzung eines `latest`-Tag wäre dies nicht möglich. Dieses Vorgehen ist jedoch auch nicht zu empfehlen, da es dadurch schwerer wird zu erkennen, welches Image gerade ausgeführt wird und auch das Rollback von Änderungen erschwert wird (vgl. Google LLC (2019b)). Stattdessen wird der Hash des Docker-Images bestimmt und zur Identifizierung der aktuellen Version genutzt (vgl. Google LLC (2019e); Listing 12).

```
1 #!/bin/bash
2 SERVICE="airact-content-service"
3
4 REPO="gcr.io/airact/${SERVICE}"
```

```
5  TAG="latest-production"
6
7  SHA256=$(docker pull "${REPO}:${TAG}" | sed -n -e 's/Digest: //p')
8  kubectl set image "deployments/${SERVICE}" \
9      "${SERVICE}=${REPO}:${SHA256}" --record
```

Listing 12: Rollout auf Produktionsumgebung

Sollten durch die Änderung Probleme entstanden sein, ist ein Rollback der Änderung möglich (vgl. Google LLC (2019e)). Listing 13 zeigt den entsprechenden Befehl.

```
1  #!/bin/bash
2  SERVICE="airact-content-service"
3
4  kubectl rollout undo "deployments/${SERVICE}"
```

Listing 13: Rollback auf Produktionsumgebung

Ergänzend zu der kontinuierlichen Integration kann auch eine kontinuierliche Bereitstellung erfolgen. Ein mögliches Vorgehen wird im Ausblick beschrieben (vgl. Unterabschnitt 9.2.5).

8 Evaluation

Alle fachlichen Anforderungen wurden erfolgreich innerhalb des Prototyps umgesetzt. Die Funktionalität des Versendens von Push-Benachrichtigungen wurde bereits in der Anforderungsanalyse bewusst minimal gehalten. Dem entsprechend werden zum jetzigen Zeitpunkt der Entwicklung Push-Benachrichtigungen nur an das zuletzt konfigurierte Gerät gesendet. In einer späteren Version der PWA empfiehlt sich jedoch die Unterstützung mehrerer Endgeräte eines Benutzers. Eine hierbei mögliche Vorgehensweise wird im Ausblick dieser Arbeit dargestellt (vgl. Unterabschnitt 9.2.2).

Durch das serverseitige Rendering und die Entwicklung nach dem Prinzip der progressiven Verbesserung ist die Nutzung der PWA auch in Browsern möglich, welche einen geringen Funktionsumfang bieten. Fehlende Funktionalitäten (etwa JavaScript `Promises`) werden durch sogenannte Polyfills ergänzt. Dabei wird das NPM-Package react-app-polyfill genutzt, welches Unterstützung für diverse Browser bietet, u. a. für den Internet Explorer 9. Für den Fall, dass bestimmte Web-APIs, wie etwa die Share-API, nicht bereitstehen, werden entsprechende (nicht-native) Dialoge dargestellt. Weiterhin funktioniert die PWA auch bei deaktiviertem JavaScript.

Die PWA folgt dem Prinzip des responsiven Designs und ist somit für verschiedene Arten von Anzeigemedien optimiert. Bestätigt wurde dies durch Tests auf verschiedenen Anzeigemedien mit unterschiedlicher Größe des Anzeigebereiches sowie durch das WooRank-Audit (vgl. Abschnitt 7.4).

Durch den Service-Worker – sofern von dem Browser unterstützt – ist der Prototyp offline-fähig. Dabei werden zuerst bereits geladene Inhalte dargestellt und anschließend die aktualisierten Inhalte angefragt. Durch die Zustandsverwaltung innerhalb der PWA werden die aktualisierten Daten dann gespeichert und automatisch, d. h. ohne weitere Benutzerinteraktion, dargestellt. Dies führt zu einer optimierten Nutzererfahrung.

Die fließende Navigation und die Orientierung an Design-Standards nativer Apps verstärken diese positive Nutzererfahrung. Insbesondere bei einer installierten App wird

durch den Vollbildmodus der authentische Eindruck einer App vermittelt. Durchgeführte Lighthouse-Audits und die erfolgreiche Umsetzung der Anforderungen aus der Checkliste für PWAs nach Google LLC (o. J.g) bestätigen dies (vgl. Abschnitt 7.4).

Die Performanz erfüllt zwar die minimalen Anforderungen an eine PWA, bietet dennoch keine vollständig zufriedenstellenden Ergebnisse. Lösungsansätze für diese Problemstellung werden im Rahmen dieser Arbeit im Ausblick dargestellt (vgl. Unterabschnitt 9.2.4).

Auch wenn die PWA bereits jetzt eine sehr gute Nutzererfahrung bietet, können auch hier weitere Optimierungen durchgeführt werden. Während der Evaluation der Anforderungen aus der Checkliste für PWAs nach Google LLC (o. J.g) zeigten sich folgende Mängel:

- Eingabefelder der Benutzeroberfläche werden auf mobilen Geräten bei geöffneter Tastatur von letzterer überdeckt.
- Dem Benutzer wird nicht angezeigt, wenn er gerade über keine (ausreichend gute) Netzwerkverbindung verfügt.
- Etwa bei Anfragen von Berechtigungen für Push-Benachrichtigungen und ähnlichen nativen Dialogen wird die angezeigte App nicht verdunkelt.

9 Fazit und Ausblick

In Abschnitt 9.1 werden die Ergebnisse dieser Arbeit abschließend zusammengefasst. Abschnitt 9.2 zeigt aktuelle Entwicklungen sowie mögliche Erweiterungen des Systems und gibt Anregungen, wie diese umgesetzt werden können.

9.1 Fazit

Der entwickelte Prototyp zeigt, dass PWAs eine ernst zu nehmende Alternative zu nativen Apps und anderen Arten von Apps darstellen. Insbesondere die Offline-Fähigkeit, die flüssige Navigation, die Installierbarkeit und die damit einhergehende Anzeige als eigenständige App sorgen für eine sehr gute Nutzererfahrung. Der Prototyp wird durch einen Web-Server bereitgestellt und ist somit über einen Browser erreichbar. Entsprechend kann eine fließende Umstellung auf die PWA als neues Frontend des WordPress-Blogs erfolgen. Insbesondere sind dabei seitens der Benutzer keine weiteren Aktionen, wie etwa eine Installation, notwendig. Die positive Bewertung der PWA durch Suchmaschinen wird durch durchgeführte Optimierungen für ebensolche gewährleistet.

Die Integration mit WordPress verlief durch die bereitgestellte WordPress REST API ohne größeren Aufwand. Lediglich einige Endpunkte mussten der WordPress REST API hinzugefügt werden, um etwa die Authentifizierung und die Abfrage von Menüs möglich zu machen. Dies war durch bestehende Drittanbieter-Plugins und die Anpassung der REST API in einem eigenen Plugin möglich.

Mit der eingesetzten Frontend-Bibliothek React konnten während der Entwicklung schnell erste Ergebnisse erzielt werden. Dabei hat insbesondere die ergänzende Bibliothek Material-UI geholfen, welche vorgefertigte Elemente der Benutzeroberfläche bereitstellt. Eine größere Herausforderung stellte hingegen der Aufbau einer sinnvollen Frontend-Architektur dar. Durch das durchgeführte Vorprojekt konnten jedoch Erfahrungswerte gesammelt

werden, welche zu der umgesetzten Architektur geführt haben. Diese Architektur stellte sich als zuverlässig und erweiterbar heraus, wodurch auch komplexe Anwendungen umsetzbar sind.

Die Umsetzung nach dem Prinzip der progressiven Verbesserung erforderte relativ aufwendige Tests, welche für viele Browser durchgeführt werden mussten. Dies stellt eine generelle Herausforderung in der Web-Entwicklung dar. Durch vorherige Untersuchung der zur Verfügung stehenden Funktionen mithilfe von caniuse[1] konnten viele Hindernisse bereits im Vorfeld abgeschätzt werden. Dennoch empfiehlt sich bei Weiterentwicklungen der PWA die Einführung eines automatisierten Cross-Browser-Testings (vgl. Ausblick 9.2.6).

Durch das universelle Rendering wird für eine schnelle erste Darstellung der PWA gesorgt und bei darauf folgender Navigation eine flüssige Darstellung ermöglicht. Auch die Nutzung der PWA bei deaktiviertem JavaScript kann mithilfe des serverseitige Renderings, welches Teil des universellen Renderings ist, ermöglicht werden.

Die Bereitstellung in einem Kubernetes-Cluster sorgt für einen schnellen Bereitstellungsprozess. Durchgeführte Updates der PWA auf dem Web-Server des Frontends werden den Benutzern automatisch ausgeliefert, sobald sie in der PWA navigieren. Dies stellt insbesondere im Vergleich zu nativen Apps, welche die Durchführung eines Update-Prozesses seitens der Benutzer erfordern, einen großen Mehrwert dar. Durch eine kontinuierliche Bereitstellung kann der Bereitstellungsprozess zusätzlich beschleunigt werden (vgl. Ausblick 9.2.5).

9.2 Ausblick

Im Folgenden wird ein Ausblick über aktuelle Entwicklungen und mögliche Verbesserungen des entwickelten Prototyps sowie der dazugehörigen Prozesse gegeben.

9.2.1 Unterstützung und Funktionen von PWAs

Insbesondere Google treibt die Entwicklung von PWAs voran. So sind eine Reihe neuer APIs für Chrome in Entwicklung, mit denen etwa auf das native Dateisystem zugegriffen

[1] https://caniuse.com/

werden kann oder verhindert wird, dass das Gerät in den Sperrzustand verfällt („Wake Lock“) (vgl. Google LLC (2020b)).

Service-Worker, die Grundvoraussetzung für offline-fähige Web-Apps, werden mittlerweile von allen gängigen Browsern unterstützt (vgl. Deveria (o. J.b)). Andere Funktionen, wie das Hinzufügen zum Startbildschirm per „Add to Home Screen“-Dialog, Push-Benachrichtigungen oder Hintergrundsynchronisation stehen bisher nur teilweise zur Verfügung (vgl. Deveria (o. J.c); Deveria (o. J.d); Deveria (o. J.e)).

Auch andere Browser und Plattformen setzen entsprechende Funktionen jedoch schrittweise um. So wurde mit der iOS Version 12.2 die Unterstützung von PWAs auf Apple-Geräten verbessert (vgl. Firtman (2019)). Auch der Edge-Browser zeigt insbesondere in neueren Versionen, welche auf dem Chromium-Browser basieren, starke Verbesserungen in der Unterstützung moderner Web-APIs. Mit der steigenden Verbreitung von PWAs ist davon auszugehen, dass auch in Zukunft mehr und mehr Funktionen in den gängigen Browsern unterstützt werden.

Inzwischen können PWAs auch über die App-Stores von Android (Play Store) und Windows (Microsoft Store) veröffentlicht werden (vgl. Google LLC (2020c); Microsoft Corporation (2020)). Für den App Store (iOS oder macOS) besteht bisher nur die Möglichkeit, die PWA in eine hybride App einzubetten, beispielsweise mit dem Cordova-Framework[2]. Desktop-Apps für Linux-Betriebssysteme können dabei ergänzend durch das Electron-Framework[3] bereitgestellt werden.

9.2.2 Unterstützung mehrerer Endgeräte pro Benutzer

Zum jetzigen Zeitpunkt der Entwicklung werden Push-Benachrichtigungen nur an das zuletzt konfigurierte Gerät gesendet. Ein möglicher Lösungsansatz, um auch mehrere Geräte zu unterstützen, ist eine Verwaltung von genutzten Geräten innerhalb der Einstellungen für die Angebotsbenachrichtigungen. Dabei kann ein Cookie genutzt werden, welcher die verschiedenen Geräte des Benutzers jeweils eindeutig identifiziert. Der Benutzer sollte dabei zusätzlich einen Namen für das Gerät festlegen können (vgl. Abbildung 9.1). Weiterhin bietet es sich an, mithilfe der Credential Management API[4] die Verwaltung von Logins über mehrere Endgeräte hinweg zu ermöglichen (vgl. Google LLC (o. J.g)).

[2] https://cordova.apache.org/
[3] https://www.electronjs.org/
[4] https://developer.mozilla.org/en-US/docs/Web/API/Credential_Management_API

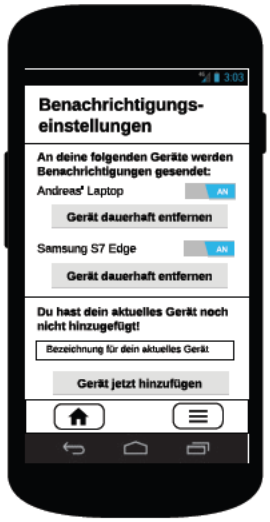

Abbildung 9.1: Wireframe: Geräteverwaltung
(Quelle: eigene Darstellung, erstellt mit Lucidchart, Bildmaterial von Google LLC (o. J.d))

9.2.3 Einsatz von Preact

Preact[5] ist eine Bibliothek, welche verspricht, React mit geringfügigen Änderungen an der Anwendung ersetzen zu können (vgl.Preact Contributors (o. J.b)). Durch den Kompatibilitäts-Layer preact-compat können alle wichtigen React-Funktionalitäten genutzt werden (vgl. Preact Contributors (o. J.a)). Preact ist dabei leichtgewichtig und bietet eine bessere Performanz als React (vgl. Anhang A.3). Auch eine Integration mit Next.js soll durch ein entsprechendes Plugin[6] möglich sein. Gestützt wird Preact durch Firmen wie Groupon, Uber und die New York Times (vgl. Preact Contributors (o. J.c)).

Es bleibt zu evaluieren, ob die Integration mit Next.js und auch mit Material-UI problemlos möglich ist. Durch den Einsatz von Preact können dann auch ungecachte Anfragen an das Frontend schneller verarbeitet werden. Es ist denkbar, dass sich die Last des Frontends damit reduziert und entsprechend mehr gleichzeitige Anfragen möglich sind.

Zu beachten ist, dass Preact zum jetzigen Zeitpunkt noch nicht sehr verbreitet ist, weshalb diese Bibliothek sicherheitshalber auch nicht in dem Prototyp eingesetzt wurde.

[5] https://preactjs.com/
[6] https://github.com/zeit/next-plugins/tree/master/packages/next-preact

Dennoch zeichnen sich stark steigende Tendenzen in der Verbreitung ab. (vgl. Anhang A.2)

9.2.4 Performanz-Optimierungen

Neben einem potenziell möglichen Einsatz von Preact, kann die Performanz der PWA durch folgende Schritte verbessert werden:

1. Caching von Antworten der WordPress REST API: Durch entsprechende Plugins können Antworten der WordPress REST API gecacht werden. Der Vorteil eines Cachings innerhalb von WordPress liegt in der automatischen Synchronisation des Caches mit neuen und veränderten Inhalten.

2. Caching und Komprimierung von Antworten der Backend-Services: Während das Caching der WordPress REST API bereits einen großen Performanz-Gewinn verspricht, lässt dich dieser durch ein Caching der Antworten der Backend-Services zusätzlich erhöhen. Die notwendige Synchronisation des Caches mit neuen und veränderten Inhalten stellt dabei jedoch ein verhältnismäßig aufwendiges Unterfangen dar. Eine ergänzende Komprimierung kann zur Reduzierung der Größe des Antwortobjekts genutzt werden.

3. Nicht-elementare Assets nach dem JavaScript-Code einbinden: Durch eine Einbindung von nicht-elementaren Assets, wie etwa Web-Fonts, am Ende des Antwortobjekts wird die PWA schneller interaktiv.

4. Lazy-Loading von Bildern: Das Laden von Bildern nach der Ausführung des JavaScript-Codes sorgt ebenfalls für eine schnellere Interaktivität der PWA. Dabei muss ein Lösungsweg nach dem Prinzip der progressiven Verbesserung gefunden werden.

5. Caching und Komprimierung von Antworten des Frontends für nicht-eingeloggte Benutzer: Hierdurch wird einerseits die Server-Last des Frontend reduziert und anderseits für noch schnelle Antworten gesorgt. Wie auch bei den Backend-Services stellt die notwendige Synchronisation jedoch ein verhältnismäßig aufwendiges Unterfangen dar.

Anhang A.7 zeigt die Ergebnisse der Lighthouse-Audits nach exemplarischer Umsetzung der genannten Verbesserungen.

9.2.5 Kontinuierliche Bereitstellung

Neben der durchgeführten kontinuierlichen Integration ist auch eine kontinuierliche Bereitstellung möglich, bei der auch die Bereitstellung auf der Produktionsumgebung automatisch erfolgt und so den Prozess zusätzlich beschleunigt. Dabei ist ein automatisches Rollback empfehlenswert, falls Änderungen doch zu unerwarteten Fehlern geführt haben (vgl. Fowler (2006)). Bei der Automatisierung der Bereitstellung sollte zudem darauf geachtet werden, nur solche Services zu deployen, welche auch wirklich geändert wurden.

Die kontinuierliche Bereitstellung kann in Form einer Continuous Delivery oder in Form eines Continuous Deployments erfolgen. Bei beiden Vorgehen werden die Änderungen zuerst automatisiert auf eine Staging-Umgebung deployt und dort getestet. Dabei werden u. a. Systemtests[7] durchgeführt. Während bei dem Continuous Deployment nach erfolgreichen Tests die Änderungen automatisch auf die Produktionsumgebung deployt werden, findet dies bei der Continuous Delivery manuell „per Knopfdruck“ statt (vgl. Abbildung 9.2).

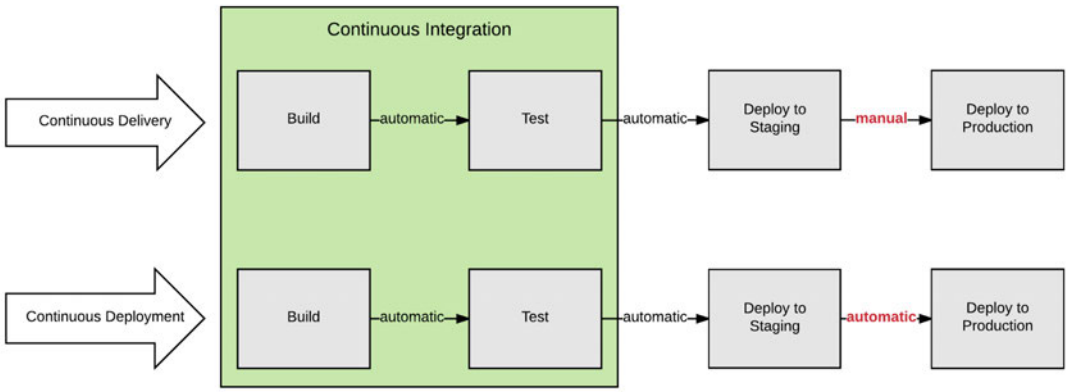

Abbildung 9.2: Kontinuierliche Integration und Bereitstellung
(Quelle: Acetozi (2018), S. 26)

Google LLC (2019h) bietet einen Einstieg in die kontinuierliche Bereitstellung in einem Kubernetes Cluster mit Google Cloud Build und der Continues Delivery-Plattform Spinnaker[8].

[7] Systemtests prüfen das System aus Benutzersicht auf die Erfüllung aller Anforderungen (vgl. Spillner u. a. (2011), S. 58f). Systemtests finden in der Umgebung des Systementwicklers statt (vgl. Spillner u. a. (2011), S. 63). Darauf folgende Akzeptanz-Tests werden in der Kundenumgebung ausgeführt und oft auch durch den Kunden selbst durchgeführt (vgl. Spillner u. a. (2011), S. 62f).

[8] https://www.spinnaker.io/

9.2.6 Cross-Browser-Testing

PWAs und Web-Apps im Allgemeinen werden meist mit dem Anspruch entwickelt, für die meistgenutzten Browser und Plattformen ein gleichbleibendes Verhalten in der Ausführung zu zeigen. Durch die Menge an verschiedenen Browsern und die andauernde Entwicklung von Web-Technologien ist dies jedoch eine große Herausforderung. Inkonsistenzen zwischen den verschiedenen Browsern führen dazu, dass sich eine Web-App je nach Browser anders verhält oder anders dargestellt wird. (vgl. Choudhary u. a. (2010))

Durch Werkzeuge für das Cross-Browser-Testing wie LambdaTest[9] oder CrossBrowserTesting[10] kann hinsichtlich der Konsistenz zwischen verschiedenen Browsern und Plattformen geprüft werden. Neben manuellen Live-Tests auf echten Geräten können mit solchen Werkzeugen auch automatisierte Tests durchgeführt werden, definiert durch Test-Frameworks wie Appium[11] oder Selenium[12].

Lambda Computing Inc. (o. J.) und CrossBrowserTesting.com, LLC (o. J.) bieten einen Einstieg in das Cross-Browser-Testing mit Selenium und Jest.

[9] https://www.lambdatest.com/
[10] https://crossbrowsertesting.com/
[11] http://appium.io/
[12] https://selenium.dev/

Literaturverzeichnis

[Acetozi 2018] Acetozi, Jorge: *Continuous Delivery for Java Apps : Kubernetes and Jenkins in Practice*. Victoria, BC : Leanpub, 2018

[Allonnea 2019] Allonnea, Mickaël: *bug: @ionic/react doesn't work [great] with Next.js · Issue #19975 · ionic-team/ionic*. 2019. – URL https://github.com/ionic-team/ionic/issues/19975. – Zugriffsdatum: 2020-02-06

[AltexSoft 2018] AltexSoft: *Xamarin vs React Native vs Ionic vs NativeScript: Cross-platform Mobile Frameworks Comparison | AltexSoft*. 2018. – URL https://www.altexsoft.com/blog/engineering/xamarin-vs-react-native-vs-ionic-vs-nativescript-cross-platform-mobile-frameworks-comparison/. – Zugriffsdatum: 2019-11-07

[Anderson 2010] Anderson, David J.: *Kanban : successful evolutionary change for your technology business*. Sequim, Washington : Blue Hole Press, 2010

[Antonio 2015] Antonio, Cássio de S.: *Pro React*. Berkeley, CA : Apress, 2015 (SpringerLink)

[Apple Inc. 2007] Apple Inc.: *iPhone to Support Third-Party Web 2.0 Applications - Apple*. 2007. – URL https://www.apple.com/newsroom/2007/06/11iPhone-to-Support-Third-Party-Web-2-0-Applications/. – Zugriffsdatum: 2019-11-17

[Apple Inc. 2008] Apple Inc.: *Apple Announces iPhone 2.0 Software Beta - Apple*. 2008. – URL https://www.apple.com/newsroom/2008/03/06Apple-Announces-iPhone-2-0-Software-Beta/. – Zugriffsdatum: 2019-11-17

[Apple Inc. o. J.a] Apple Inc.: *App Store Review Guidelines - Apple Developer*. o. J.. – URL https://developer.apple.com/app-store/review/guidelines/. – Zugriffsdatum: 2019-12-14

[Apple Inc. o. J.b] Apple Inc.: *How it works - Apple Developer Program*. o. J.. – URL https://developer.apple.com/programs/how-it-works/. – Zugriffsdatum: 2019-12-14

[Apple Inc. o. J.c] Apple Inc.: *Views and Controls | Apple Developer Documentation*. o. J.. – URL https://developer.apple.com/documentation/uikit/views_and_controls. – Zugriffsdatum: 2019-11-24

[Appscope o. J.] Appscope: *Top list - Appscope*. o. J.. – URL https://appsco.pe/toplist. – Zugriffsdatum: 2019-11-09

[Baclit und Sicam 2009] Baclit, Ryan ; Sicam, Chivas: *Foundations of CentOS Linux : enterprise Linux on the cheap*. Berkeley, CA : Apress, 2009 (SpringerLink)

[Barney 2009] Barney, Lee S.: *Developing hybrid applications for the iPhone: using HTML, CSS, and JavaScript to build dynamic apps for the iPhone*. Addison-Wesley Professional, 2009

[Bass u. a. 2015] Bass, Len ; Weber, Ingo ; Zhu, Liming: *DevOps : a software architect's perspective*. New York, NJ : Addison-Wesley, 2015

[Beck 2000] Beck, Kent: *Extreme programming explained : embrace change*. Reading, MA : Addison-Wesley, 2000 (The XP series)

[Beck u. a. 2001] Beck, Kent ; Fowler, Martin ; Kohnke, Jennifer: *Planning extreme programming*. Boston, MA : Addison-Wesley, 2001

[Bidelman 2010] Bidelman, Eric: *The Basics of Web Workers - HTML5 Rocks*. 2010. – URL https://www.html5rocks.com/en/tutorials/workers/basics/. – Zugriffsdatum: 2019-12-06

[Cabot 2018] Cabot, Jordi: WordPress: a content management system to Democratize publishing. In: *IEEE Software* 35 (2018), Nr. 3, S. 89–92

[Champeon und Finck 2003] Champeon, Steven ; Finck, Nick: *Inclusive Web Design*. 2003. – URL http://www.hesketh.com/publications/inclusive_web_design_for_the_future/. – Zugriffsdatum: 2019-11-23

[Chavez o. J.] Chavez, Enrique: *JWT Authentication for WP REST API – WordPress-Plugin | WordPress.org Deutsch*. o. J.. – URL https://de.wordpress.org/plugins/jwt-authentication-for-wp-rest-api/. – Zugriffsdatum: 2020-02-09

[Choudhary u. a. 2010] Choudhary, Shauvik R. ; Versee, Husayn ; Orso, Alessandro: WEBDIFF: Automated identification of cross-browser issues in web applications. In: *2010 IEEE International Conference on Software Maintenance* IEEE (Veranst.), 2010, S. 1–10

[Cockburn 1998] Cockburn, Alistair: *Basic Use Case Template.* 1998. – URL http://cis.bentley.edu/lwaguespack/CS360_Site/Downloads_files/Use%20Case%20Template%20%28Cockburn%29.pdf. – Zugriffsdatum: 2020-01-25

[Cohn 2004] Cohn, Mike: *User stories applied : for agile software development.* Boston, MA : Addison-Wesley, 2004 (Addison-Wesley signature series)

[Comscore, Inc. 2019] Comscore, Inc.: *Global State of Mobile - Comscore, Inc.* 2019. – URL https://www.comscore.com/Insights/Presentations-and-Whitepapers/2019/Global-State-of-Mobile. – Zugriffsdatum: 2019-12-14

[Constantine und Lockwood 1999] Constantine, Larry L. ; Lockwood, Lucy A. D.: *Software for use : a practical guide to the models and methods of usage-centered design.* Boston, MA : Addison-Wesley [u.a.], 1999

[CrossBrowserTesting.com, LLC o. J.] CrossBrowserTesting.com, LLC: *Jest.* o. J.. – URL https://help.crossbrowsertesting.com/selenium-testing/frameworks/jest/. – Zugriffsdatum: 2020-02-14

[Danylko 2016] Danylko, Jonathan: *What Is Loose Coupling? - DZone Integration.* 2016. – URL https://dzone.com/articles/what-is-loose-coupling. – Zugriffsdatum: 2019-12-08

[Deveria o. J.a] Deveria, Alexis: *Can I use... Support tables for HTML5, CSS3, etc.* o. J.. – URL https://caniuse.com/#feat=promises. – Zugriffsdatum: 2020-01-26

[Deveria o. J.b] Deveria, Alexis: *Can I use... Support tables for HTML5, CSS3, etc.* o. J.. – URL https://caniuse.com/#feat=serviceworkers. – Zugriffsdatum: 2020-02-12

[Deveria o. J.c] Deveria, Alexis: *Can I use... Support tables for HTML5, CSS3, etc.* o. J.. – URL https://caniuse.com/#feat=beforeinstallprompt. – Zugriffsdatum: 2020-02-12

[Deveria o. J.d] Deveria, Alexis: *Can I use... Support tables for HTML5, CSS3, etc.* o. J.. – URL https://caniuse.com/#feat=push-api. – Zugriffsdatum: 2020-02-12

[Deveria o. J.e] Deveria, Alexis: *Can I use... Support tables for HTML5, CSS3, etc.* o. J.. – URL https://caniuse.com/#feat=background-sync. – Zugriffsdatum: 2020-02-12

[Docker Inc. o. J.a] Docker Inc.: */ Docker Documentation.* o. J.. – URL https://docs.docker.com/engine/reference/builder/. – Zugriffsdatum: 2020-01-24

[Docker Inc. o. J.b] Docker Inc.: *Containerizing an application | Docker Documentation.* o. J.. – URL https://docs.docker.com/get-started/part2/. – Zugriffsdatum: 2019-12-13

[Docker Inc. o. J.c] Docker Inc.: *Orientation and setup | Docker Documentation.* o. J.. – URL https://docs.docker.com/get-started/. – Zugriffsdatum: 2019-12-13

[Docker Inc. o. J.d] Docker Inc.: *Sharing images on Docker Hub | Docker Documentation.* o. J.. – URL https://docs.docker.com/get-started/part5/. – Zugriffsdatum: 2019-12-13

[Driessen 2010] Driessen, Vincent: *A successful Git branching model » nvie.com.* 2010. – URL https://nvie.com/posts/a-successful-git-branching-model/. – Zugriffsdatum: 2020-01-27

[Drifty Co. 2019] Drifty Co.: *Ionic 4 Roadmap: New Features, Release Schedule, & Goals—Oh My! | The Ionic Blog.* 2019. – URL https://ionicframework.com/blog/ionic-4-roadmap-new-features-release-schedule-goals-oh-my/. – Zugriffsdatum: 2019-11-18

[Drifty Co. o. J.] Drifty Co.: *Ionic Packages - Ionic Documentation.* o. J.. – URL https://ionicframework.com/docs/installation/cdn. – Zugriffsdatum: 2019-11-18

[Ecma International 2019a] Ecma International: *ECMAScript® 2019 Language Specification.* 2019. – URL https://www.ecma-international.org/ecma-262/10.0/index.html. – Zugriffsdatum: 2019-12-13

[Ecma International 2019b] Ecma International: *Standard ECMA-262*. 2019. – URL http://www.ecma-international.org/publications/standards/Ecma-262.htm. – Zugriffsdatum: 2019-12-12

[Eisenman 2015] Eisenman, Bonnie: *Learning react native: Building native mobile apps with JavaScript*. O'Reilly Media, Inc., 2015

[El-Kassas u. a. 2017] El-Kassas, Wafaa S. ; Abdullah, Bassem A. ; Yousef, Ahmed H. ; Wahba, Ayman M.: Taxonomy of cross-platform mobile applications development approaches. In: *Ain Shams Engineering Journal* 8 (2017), Nr. 2, S. 163–190

[Endsley 2013] Endsley, Rikki: *5 Reasons It's Time to Ditch MySQL*. 2013. – URL https://smartbear.com/blog/test-and-monitor/5-reasons-its-time-to-ditch-mysql/. – Zugriffsdatum: 2020-01-22

[ESLint Contributors o. J.] ESLint Contributors: *Integrations - ESLint - Pluggable JavaScript linter*. o. J.. – URL https://eslint.org/docs/user-guide/integrations. – Zugriffsdatum: 2020-01-29

[Express Contributors o. J.a] Express Contributors: *Express middleware*. o. J.. – URL http://expressjs.com/en/resources/middleware.html. – Zugriffsdatum: 2020-01-19

[Express Contributors o. J.b] Express Contributors: *Using Express middleware*. o. J.. – URL https://expressjs.com/en/guide/using-middleware.html. – Zugriffsdatum: 2020-01-19

[Facebook Inc. 2019a] Facebook Inc.: *In-Depth Overview*. 2019. – URL https://facebook.github.io/flux/docs/in-depth-overview/. – Zugriffsdatum: 2020-01-19

[Facebook Inc. 2019b] Facebook Inc.: *react/CHANGELOG.md at master · facebook/react*. 2019. – URL https://github.com/facebook/react/blob/master/CHANGELOG.md. – Zugriffsdatum: 2020-01-21

[Facebook Inc. o. J.a] Facebook Inc.: *Components and Props – React*. o. J.. – URL https://reactjs.org/docs/components-and-props.html. – Zugriffsdatum: 2020-01-21

[Facebook Inc. o. J.b] Facebook Inc.: *Handling Events – React*. o. J.. – URL https://reactjs.org/docs/handling-events.html. – Zugriffsdatum: 2020-01-21

[Facebook Inc. o. J.c] Facebook Inc.: *Hooks at a Glance – React*. o. J.. – URL https://reactjs.org/docs/hooks-overview.html. – Zugriffsdatum: 2020-01-21

[Facebook Inc. o. J.d] Facebook Inc.: *Introducing JSX – React*. o. J.. – URL https://reactjs.org/docs/introducing-jsx.html. – Zugriffsdatum: 2020-01-21

[Facebook Inc. o. J.e] Facebook Inc.: *Virtual DOM and Internals – React*. o. J.. – URL https://reactjs.org/docs/faq-internals.html. – Zugriffsdatum: 2020-01-21

[Firtman 2019] Firtman, Maximiliano: *Switching to Preact (from React) | Preact: Fast 3kb React alternative with the same ES6 API. Components & Virtual DOM.* 2019. – URL https://preactjs.com/guide/v10/switching-to-preact. – Zugriffsdatum: 2020-02-11

[Flatart 2019] Flatart: *Logos and Brands icons by Flatart*. 2019. – URL https://www.iconfinder.com/iconsets/logos-and-brands. – Zugriffsdatum: 2019-12-17

[Fowler 2006] Fowler, Martin: *Continuous Integration*. 2006. – URL https://martinfowler.com/articles/continuousIntegration.html. – Zugriffsdatum: 2020-01-25

[Gamma u. a. 1995] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John M.: *Design patterns : elements of reusable object-oriented software*. Upper Saddle River, NJ : Addison-Wesley, 1995 (Addison-Wesley professional computing series)

[Gartner, Inc. 2017] Gartner, Inc.: *Progressive Web Apps Will Impact Your Mobile App Strategy*. 2017. – URL https://www.gartner.com/en/documents/3645344. – Zugriffsdatum: 2020-02-13

[Gaunt o. J.] Gaunt, Matt: *Service Workers: an Introduction | Web Fundamentals*. o. J.. – URL https://developers.google.com/web/fundamentals/primers/service-workers. – Zugriffsdatum: 2019-12-06

[Gaunt und Kinlan o. J.] Gaunt, Matt ; Kinlan, Paul: *The Web App Manifest | Web Fundamentals | Google Developers*. o. J.. – URL https://developers.google.com/web/fundamentals/web-app-manifest. – Zugriffsdatum: 2019-12-06

[Google LLC 2019a] Google LLC: *Build-Konfiguration – Überblick | Cloud Build-Dokumentation | Google Cloud*. 2019. – URL https://cloud.google.com/cloud-build/docs/build-config. – Zugriffsdatum: 2020-01-23

[Google LLC 2019b] Google LLC: *Configuration Best Practices - Kubernetes*. 2019. – URL https://kubernetes.io/docs/concepts/configuration/overview/. – Zugriffsdatum: 2020-01-27

[Google LLC 2019c] Google LLC: *Deploying a stateless application | Kubernetes Engine Documentation | Google Cloud*. 2019. – URL https://cloud.google.com/kubernetes-engine/docs/how-to/stateless-apps. – Zugriffsdatum: 2020-01-24

[Google LLC 2019d] Google LLC: *Deployment | Kubernetes Engine Documentation | Google Cloud*. 2019. – URL https://cloud.google.com/kubernetes-engine/docs/concepts/deployment. – Zugriffsdatum: 2020-01-24

[Google LLC 2019e] Google LLC: *Deployments - Kubernetes*. 2019. – URL https://kubernetes.io/docs/concepts/workloads/controllers/deployment/. – Zugriffsdatum: 2020-01-27

[Google LLC 2019f] Google LLC: *How is Flutter different for app development - YouTube*. 2019. – URL https://www.youtube.com/watch?v=l-YO9CmaSUM&t=28. – Zugriffsdatum: 2019-11-07

[Google LLC 2019g] Google LLC: *HTTP(S) load balancing with Ingress | Kubernetes Engine Documentation | Google Cloud*. 2019. – URL https://cloud.google.com/kubernetes-engine/docs/concepts/ingress. – Zugriffsdatum: 2020-01-24

[Google LLC 2019h] Google LLC: *Install and Manage Spinnaker on Google Cloud Platform*. 2019. – URL https://cloud.google.com/docs/ci-cd/spinnaker/spinnaker-for-gcp. – Zugriffsdatum: 2020-02-14

[Google LLC 2019i] Google LLC: *lighthouse/scoring.md at master · GoogleChrome/lighthouse · GitHub*. 2019. – URL https://github.com/GoogleChrome/lighthouse/blob/master/docs/scoring.md. – Zugriffsdatum: 2019-12-06

[Google LLC 2019j] Google LLC: *Service | Kubernetes Engine Documentation | Google Cloud*. 2019. – URL https://cloud.google.com/kubernetes-engine/docs/concepts/service. – Zugriffsdatum: 2020-01-24

[Google LLC 2020a] Google LLC: *Service discovery and DNS | Kubernetes Engine Documentation | Google Cloud*. 2020. – URL https://cloud.google.com/kubernetes-engine/docs/concepts/service-discovery. – Zugriffsdatum: 2020-01-24

[Google LLC 2020b] Google LLC: *Unlocking new capabilities for the web | Google Developers*. 2020. – URL https://developers.google.com/web/updates/capabilities. – Zugriffsdatum: 2020-02-11

[Google LLC 2020c] Google LLC: *Using Trusted Web Activities | Google Developers*. 2020. – URL https://developers.google.com/web/updates/2019/02/using-twa. – Zugriffsdatum: 2020-02-11

[Google LLC o. J.a] Google LLC: *Angular - Architecture overview*. o. J.. – URL https://angular.io/guide/architecture. – Zugriffsdatum: 2020-01-21

[Google LLC o. J.b] Google LLC: *Angular - Template Syntax*. o. J.. – URL https://angular.io/guide/template-syntax. – Zugriffsdatum: 2020-01-21

[Google LLC o. J.c] Google LLC: *Custom Drawing | Android Developers*. o. J.. – URL https://developer.android.com/training/custom-views/custom-drawing. – Zugriffsdatum: 2019-11-07

[Google LLC o. J.d] Google LLC: *Icons - Material Design*. o. J.. – URL https://material.io/resources/icons/. – Zugriffsdatum: 2019-12-17

[Google LLC o. J.e] Google LLC: *Introduction to Push Notifications | Web | Google Developers*. o. J.. – URL https://developers.google.com/web/ilt/pwa/introduction-to-push-notifications. – Zugriffsdatum: 2020-02-08

[Google LLC o. J.f] Google LLC: *Lighthouse | Tools for Web Developers | Google Developers*. o. J.. – URL https://developers.google.com/web/tools/lighthouse/. – Zugriffsdatum: 2019-12-06

[Google LLC o. J.g] Google LLC: *Progressive Web App Checklist | Google Developers*. o. J.. – URL https://developers.google.com/web/progressive-web-apps/checklist. – Zugriffsdatum: 2019-12-06

[Google LLC o. J.h] Google LLC: *Publish an app - Play Console Help*. o. J.. – URL https://support.google.com/googleplay/android-developer/answer/6334282?hl=en. – Zugriffsdatum: 2019-12-14

[Google LLC o. J.i] Google LLC: *Sending build notifications | Cloud Build Documentation | Google Cloud*. o. J.. – URL https://cloud.google.com/cloud-build/docs/send-build-notifications. – Zugriffsdatum: 2020-01-27

[Google LLC o. J.j] Google LLC: *Showcase - Flutter*. o. J.. – URL https://flutter.dev/showcase. – Zugriffsdatum: 2019-11-09

[Google LLC o. J.k] Google LLC: *V8 JavaScript engine*. o. J.. – URL https://v8.dev/. – Zugriffsdatum: 2019-12-12

[Google LLC o. J.l] Google LLC: *Workbox Strategies | Google Developers*. o. J.. – URL https://developers.google.com/web/tools/workbox/modules/workbox-strategies. – Zugriffsdatum: 2020-02-27

[Grant 2018] Grant, Pete: *Automated Integration Test Branch Patterns — Cloud BI*. 2018. – URL https://cloudbilimited.com/blog/2018/5/21/automated-integration-test-branch-patterns. – Zugriffsdatum: 2020-01-27

[Grigoryan 2016] Grigoryan, Alex: *The Benefits of Server Side Rendering Over Client Side Rendering*. 2016. – URL https://medium.com/walmartlabs/the-benefits-of-server-side-rendering-over-client-side-rendering-5d07ff2cefe8. – Zugriffsdatum: 2020-01-13

[Gustafson 2008] Gustafson, Aaron: *Understanding Progressive Enhancement – A List Apart*. 2008. – URL https://alistapart.com/article/understandingprogressiveenhancement/. – Zugriffsdatum: 2019-11-04

[Gustafson 2010] Gustafson, Aaron: *CSS Adaptive Layouts with Media Queries*. 2010. – URL https://de.slideshare.net/AaronGustafson/css-adaptive-layouts-with-media-queries. – Zugriffsdatum: 2019-11-24

[Hou-Sandi 2016] Hou-Sandi, Helen: *News – WordPress 4.7 "Vaughan" – WordPress.org*. 2016. – URL https://wordpress.org/news/2016/12/vaughan/. – Zugriffsdatum: 2019-12-15

[Humble und Farley 2011] Humble, Jez ; Farley, David: *Continuous delivery*. Upper Saddle River, NJ : Addison-Wesley, 2011

[IBM Corporation 2012] IBM Corporation: *Native, web or hybrid mobile-app development*. 2012. – URL https://www.computerworld.com.au/whitepaper/371126/native-web-or-hybrid-mobile-app-development/download/. – Zugriffsdatum: 2019-11-06

[IEEE 2008] IEEE: IEEE Standard for Software and System Test Documentation. In: *IEEE Std 829-2008* (2008), S. 1–150

[Jobs 2007] Jobs, Steve: *Apple - Hot News*. 2007. – URL http://web.archive.org/web/20071018221832/http://www.apple.com/hotnews/. – Zugriffsdatum: 2019-11-17. – Archiviert vom Original (http://www.apple.com/hotnews/) am 2007-10-18.

[Klahold und Fathi 2020] Klahold, André ; Fathi, Madjid: *Computer Aided Writing*. Cham : Springer, 2020 (SpringerLink)

[Kononenko 2018] Kononenko, Kevin: *Going out to eat and understanding the basics of Express.js - DEV Community*. 2018. – URL https://dev.to/kbk0125/going-out-to-eat-and-understanding-the-basics-of-expressjs-4d6n. – Zugriffsdatum: 2020-01-19

[Kubernetes Contributors 2019a] Kubernetes Contributors: *Concepts - Kubernetes*. 2019. – URL https://kubernetes.io/docs/concepts/. – Zugriffsdatum: 2020-01-23

[Kubernetes Contributors 2019b] Kubernetes Contributors: *Namespaces - Kubernetes*. 2019. – URL https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/. – Zugriffsdatum: 2020-01-24

[Kubernetes Contributors 2019c] Kubernetes Contributors: *Pods - Kubernetes*. 2019. – URL https://kubernetes.io/docs/concepts/workloads/pods/pod/. – Zugriffsdatum: 2020-01-24

[Kubernetes Contributors 2020] Kubernetes Contributors: *Service - Kubernetes*. 2020. – URL https://kubernetes.io/docs/concepts/services-networking/service/. – Zugriffsdatum: 2020-01-24

[Lambda Computing Inc. o. J.] Lambda Computing Inc.: *Automation Testing With Selenium And Jest | LambdaTest Documentation*. o. J.. – URL https://www.lambdatest.com/support/docs/automation-testing-with-selenium-and-jest/. – Zugriffsdatum: 2020-02-14

[Latif u. a. 2016] Latif, Mounaim ; Lakhrissi, Younes ; Nfaoui, El H. ; Es-Sbai, Najia: Cross platform approach for mobile application development: A survey. In: *2016 International Conference on Information Technology for Organizations Development (IT4OD)* IEEE (Veranst.), 2016, S. 1–5

[LePage o. J.] LePage, Pete: *Add to Home Screen | Web Fundamentals | Google Developers*. o. J.. – URL https://developers.google.com/web/fundamentals/app-install-banners. – Zugriffsdatum: 2019-12-06

[Lewis 2015] Lewis, Ian: *Using Kubernetes Namespaces to Manage Environments - Kubernetes*. 2015. – URL https://kubernetes.io/blog/2015/08/using-kubernetes-namespaces-to-manage/. – Zugriffsdatum: 2020-01-24

[Lewis und Fowler 2014] Lewis, James ; Fowler, Martin: *Microservices*. 2014. – URL https://martinfowler.com/articles/microservices.html. – Zugriffsdatum: 2020-02-07

[Li 2018] Li, Chris: *Can next.js (v5) works with OnsenUI? · Issue #3772 · zeit/next.js*. 2018. – URL https://github.com/zeit/next.js/issues/3772. – Zugriffsdatum: 2020-02-06

[Ludewig und Lichter 2013] Ludewig, Jochen ; Lichter, Horst: *Software Engineering : Grundlagen, Menschen, Prozesse, Techniken*. 3., korrigierte Auflage. Heidelberg : dpunkt.Verlag, 2013

[Lukša 2018] Lukša, Marko: *Kubernetes in Action*. Shelter Island, NY : Manning, 2018

[MacDonald 2018] MacDonald, Allyson: *5 ways the web will evolve in 2018 – O'Reilly*. 2018. – URL https://www.oreilly.com/radar/5-ways-the-web-will-evolve-in-2018/. – Zugriffsdatum: 2020-02-13

[Marcotte 2010] Marcotte, Ethan: *Responsive Web Design – A List Apart*. 2010. – URL https://alistapart.com/article/responsive-web-design/. – Zugriffsdatum: 2019-11-03

[Martin 2019] Martin, Sophia: *Angular vs React vs Vue: Which is the Best Choice for 2019? - By Sophia*. 2019. – URL https://hackernoon.com/angular-vs-react-vs-vue-which-is-the-best-choice-for-2019-16ce0deb3847. – Zugriffsdatum: 2020-01-21

[Microsoft Corporation 2020] Microsoft Corporation: *Progressive Web Apps in the Microsoft Store - Microsoft Edge Development | Microsoft Docs*. 2020. – URL https://docs.microsoft.com/en-us/microsoft-edge/progressive-web-apps/microsoft-store. – Zugriffsdatum: 2020-02-11

[Microsoft Corporation o. J.a] Microsoft Corporation: *React & Webpack · TypeScript*. o. J.. – URL https://www.typescriptlang.org/docs/handbook/react-&-webpack.html. – Zugriffsdatum: 2020-01-13

[Microsoft Corporation o. J.b] Microsoft Corporation: *TypeScript in 5 minutes · TypeScript*. o. J.. – URL https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html. – Zugriffsdatum: 2020-01-23

[Microsoft Corporation o. J.c] Microsoft Corporation: *Xamarin Customer Showcase | .NET*. o. J.. – URL https://dotnet.microsoft.com/apps/xamarin/customers. – Zugriffsdatum: 2019-11-09

[MongoDB, Inc. o. J.a] MongoDB, Inc.: *$lookup (aggregation) — MongoDB Manual*. o. J.. – URL https://docs.mongodb.com/master/reference/operator/aggregation/lookup/. – Zugriffsdatum: 2020-01-21

[MongoDB, Inc. o. J.b] MongoDB, Inc.: *Schema Validation — MongoDB Manual*. o. J.. – URL https://docs.mongodb.com/manual/core/schema-validation/index.html. – Zugriffsdatum: 2020-01-21

[Mozilla Foundation 2018] Mozilla Foundation: *No-Judgment Digital Definitions: App vs Web App | The Firefox Frontier*. 2018. – URL https://blog.mozilla.org/firefox/no-judgment-digital-definitions-app-vs-web-app/. – Zugriffsdatum: 2019-12-15

[Mozilla Foundation 2019a] Mozilla Foundation: *Add to Home screen - Progressive web apps | MDN*. 2019. – URL https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Add_to_home_screen. – Zugriffsdatum: 2019-12-06

[Mozilla Foundation 2019b] Mozilla Foundation: *Canvas - HTML: HyperText Markup Language | MDN*. 2019. – URL https://developer.mozilla.org/de/docs/Web/HTML/Canvas. – Zugriffsdatum: 2019-11-07

[Mozilla Foundation 2019c] Mozilla Foundation: *Document Object Model (DOM) - Web APIs | MDN*. 2019. – URL https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model. – Zugriffsdatum: 2019-12-06

[Mozilla Foundation 2019d] Mozilla Foundation: *Progressive Web-Apps | MDN*. 2019. – URL https://developer.mozilla.org/de/docs/Web/Progressive_web_apps. – Zugriffsdatum: 2019-11-04

[Mozilla Foundation 2019e] Mozilla Foundation: *Service Worker API - Web APIs | MDN*. 2019. – URL https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API. – Zugriffsdatum: 2019-12-06

[Mullenweg 2015] Mullenweg, Matt: *News – WordPress 4.4 "Clifford" – WordPress.org*. 2015. – URL https://wordpress.org/news/2015/12/clifford/. – Zugriffsdatum: 2019-12-15

[Mybridge 2016] Mybridge: *21 Amazing Open Source iOS Apps Written in Swift - Mybridge for Professionals*. 2016. – URL https://medium.mybridge.co/21-amazing-open-source-ios-apps-written-in-swift-5e835afee98e. – Zugriffsdatum: 2019-11-10

[Márton 2017] Márton, Péter: *Building an API Gateway using Node.js | @RisingStack*. 2017. – URL https://blog.risingstack.com/building-an-api-gateway-using-nodejs/. – Zugriffsdatum: 2020-02-07

[National Institute of Standards and Technology 2011] National Institute of Standards and Technology: *NIST SP 800-145, The NIST Definition of Cloud Computing*. 2011. – URL https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf. – Zugriffsdatum: 2020-01-23

[Nelson 2018] Nelson, James K.: *Static vs. Server Rendering*. 2018. – URL https://frontarm.com/james-k-nelson/static-vs-server-rendering/. – Zugriffsdatum: 2020-01-13

[Neutkens 2019] Neutkens, Tim: *window is not defined · Issue #6080 · zeit/next.js*. 2019. – URL https://github.com/zeit/next.js/issues/6080. – Zugriffsdatum: 2020-02-06

[Node.js Foundation 2018a] Node.js Foundation: *Node_JS_Field_Guide_Building_APIs_R1.indd*. 2018. – URL https://foundation.nodejs.org/wp-content/uploads/sites/50/2018/06/NodeJS_FieldGuide_Building_APIs_Final.pdf. – Zugriffsdatum: 2019-12-11

[Node.js Foundation 2018b] Node.js Foundation: *Typeface Use*. 2018. – URL https://foundation.nodejs.org/wp-content/uploads/sites/50/2018/09/Selling_Nodejs_in_Your_Enterprise_2018-1.pdf. – Zugriffsdatum: 2019-12-12

[Nowak 2020a] Nowak, Przemek: *github-stars-history*. 2020. – URL https://stars.przemeknowak.com/. – Zugriffsdatum: 2020-01-12

[Nowak 2020b] Nowak, Przemek: *github-stars-history*. 2020. – URL https://stars.przemeknowak.com/. – Zugriffsdatum: 2020-01-13

[Nowak 2020c] Nowak, Przemek: *github-stars-history*. 2020. – URL https://stars.przemeknowak.com/. – Zugriffsdatum: 2020-01-14

[Nunkesser 2018] Nunkesser, Robin: Beyond web/native/hybrid: a new taxonomy for mobile app development. In: *2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft)* IEEE (Veranst.), 2018, S. 214–218

[OpenJS Foundation o. J.a] OpenJS Foundation: *Child Process | Node.js v13.3.0 Documentation*. o. J.. – URL https://nodejs.org/api/child_process.html. – Zugriffsdatum: 2019-12-11

[OpenJS Foundation o. J.b] OpenJS Foundation: *Cluster | Node.js v13.3.0 Documentation*. o. J.. – URL https://nodejs.org/api/cluster.html. – Zugriffsdatum: 2019-12-11

[OpenJS Foundation o. J.c] OpenJS Foundation: *ECMAScript 2015 (ES6) and beyond | Node.js*. o. J.. – URL https://nodejs.org/de/docs/es6/. – Zugriffsdatum: 2019-12-12

[OpenJS Foundation o. J.d] OpenJS Foundation: *ECMAScript Modules | Node.js v13.3.0 Documentation*. o. J.. – URL https://nodejs.org/api/esm.html. – Zugriffsdatum: 2019-12-13

[OpenJS Foundation o. J.e] OpenJS Foundation: *Modules | Node.js v13.3.0 Documentation*. o. J.. – URL https://nodejs.org/api/modules.html. – Zugriffsdatum: 2019-12-11

[OpenJS Foundation o. J.f] OpenJS Foundation: *The Node.js Event Loop, Timers, and process.nextTick() | Node.js*. o. J.. – URL https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/. – Zugriffsdatum: 2019-12-11

[OpenJS Foundation o. J.g] OpenJS Foundation: *npm | build amazing things*. o. J.. – URL https://www.npmjs.com/. – Zugriffsdatum: 2019-12-12

[OpenJS Foundation o. J.h] OpenJS Foundation: *Worker Threads | Node.js v13.3.0 Documentation*. o. J.. – URL https://nodejs.org/api/worker_threads.html. – Zugriffsdatum: 2019-12-11

[Osmani 2015] Osmani, Addy: *Getting Started with Progressive Web Apps | Google Developers*. 2015. – URL https://developers.google.com/web/updates/2015/12/getting-started-pwa. – Zugriffsdatum: 2019-11-03

[Palmer o. J.] Palmer, Matthew: *Kubernetes Ingress with Nginx Example - Kubernetes Book*. o. J.. – URL https://matthewpalmer.net/kubernetes-app-developer/articles/kubernetes-ingress-guide-nginx-example.html. – Zugriffsdatum: 2020-01-24

[Parker u. a. 2013] Parker, Zachary ; Poe, Scott ; Vrbsky, Susan V.: Comparing nosql mongodb to an sql db. In: *Proceedings of the 51st ACM Southeast Conference*, 2013, S. 1–6

[Perfect Sense 2018] Perfect Sense: *Content Management Systems: Traditional, Decoupled or Headless?* 2018. – URL https://www.brightspot.com/blog/decoupled-cms-and-headless-cms-platforms. – Zugriffsdatum: 2019-12-07

[Posnick o. J.] Posnick, Jeff: *Service Worker Registration | Web Fundamentals | Google Developers*. o. J.. – URL https://developers.google.com/web/fundamentals/primers/service-workers/registration. – Zugriffsdatum: 2019-11-24

[Potter 2020] Potter, John: *mobx vs redux | npm trends*. 2020. – URL https://www.npmtrends.com/mobx-vs-redux. – Zugriffsdatum: 2020-01-14

[Preact Contributors o. J.a] Preact Contributors: *Differences to React | Preact: Fast 3kb React alternative with the same ES6 API. Components & Virtual DOM.* o. J.. – URL https://preactjs.com/guide/v10/differences-to-react/. – Zugriffsdatum: 2020-02-11

[Preact Contributors o. J.b] Preact Contributors: *Switching to Preact (from React) | Preact: Fast 3kb React alternative with the same ES6 API. Components & Virtual DOM.* o. J.. – URL https://preactjs.com/guide/v10/switching-to-preact. – Zugriffsdatum: 2020-02-11

[Preact Contributors o. J.c] Preact Contributors: *Who's using Preact? | Preact: Fast 3kb React alternative with the same ES6 API. Components & Virtual DOM.* o. J..

– URL https://preactjs.com/about/we-are-using. – Zugriffsdatum: 2020-02-11

[Prettier Contributors o. J.a] Prettier Contributors: *Options · Prettier*. o. J.. – URL https://prettier.io/docs/en/options.html. – Zugriffsdatum: 2020-01-27

[Prettier Contributors o. J.b] Prettier Contributors: *What is Prettier? · Prettier*. o. J.. – URL https://prettier.io/docs/en/index.html. – Zugriffsdatum: 2020-01-27

[Q-Success 2019a] Q-Success: *Historical yearly trends in the usage of content management systems, November 2019*. 2019. – URL https://w3techs.com/technologies/history_overview/content_management/all/y. – Zugriffsdatum: 2019-11-03

[Q-Success 2019b] Q-Success: *Usage Statistics and Market Share of Content Management Systems, November 2019*. 2019. – URL https://w3techs.com/technologies/overview/content_management/all. – Zugriffsdatum: 2019-11-03

[Raharja und Siahaan 2019] Raharja, Irit Maulana S. ; Siahaan, Daniel O.: Classification of Non-Functional Requirements Using Fuzzy Similarity KNN Based on ISO/IEC 25010. In: *2019 12th International Conference on Information & Communication Technology and System (ICTS)* IEEE (Veranst.), 2019, S. 264–269

[Raj und Tolety 2012] Raj, CP R. ; Tolety, Seshu B.: A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. In: *2012 Annual IEEE India Conference (INDICON)* IEEE (Veranst.), 2012, S. 625–629

[Redux Contributors o. J.] Redux Contributors: *Data flow*. o. J.. – URL https://redux.js.org/basics/data-flow/. – Zugriffsdatum: 2020-01-21

[Refsnes Data o. J.] Refsnes Data: *SQL Joins*. o. J.. – URL https://www.w3schools.com/sql/sql_join.asp. – Zugriffsdatum: 2020-01-21

[Richter 2019] Richter, Felix: • *Chart: Amazon Dominates Public Cloud Market | Statista*. 2019. – URL https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/. – Zugriffsdatum: 2020-01-23

[Rigau 2018] Rigau, Xavi: *Introduction to Redux in Flutter*. 2018. – URL https://blog.novoda.com/introduction-to-redux-in-flutter/. – Zugriffsdatum: 2020-01-19

[Robertson und Robertson 2013] Robertson, Suzanne ; Robertson, James: *Mastering the requirements process : getting requirements right*. 3. Auflage. Upper Saddle River, NJ : Addison-Wesley, 2013

[Russell 2015] Russell, Alex: *Progressive Web Apps: Escaping Tabs Without Losing Our Soul – Infrequently Noted*. 2015. – URL https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/. – Zugriffsdatum: 2019-11-03

[Serrano u. a. 2013] Serrano, Nicolas ; Hernantes, Josune ; Gallardo, Gorka: Mobile web apps. In: *IEEE software* 30 (2013), Nr. 5, S. 22–27

[solid IT GmbH 2020] solid IT GmbH: *DB-Engines Ranking - popularity ranking of database management systems*. 2020. – URL https://db-engines.com/en/ranking. – Zugriffsdatum: 2020-01-19

[Spillner u. a. 2011] Spillner, Andreas ; Linz, Tilo ; Schaefer, Hans: *Software testing foundations : a study guide for the certified tester exam ; foundation level, ISTQB compliant*. 3. Auflage. Santa Barbara, CA : Rocky Nook, 2011

[Stack Exchange Inc 2020a] Stack Exchange Inc: *Stack Overflow Trends*. 2020. – URL https://insights.stackoverflow.com/trends?tags=angular%2Cangularjs%2Cvue.js%2Creactjs%2Cember.js. – Zugriffsdatum: 2020-01-12

[Stack Exchange Inc 2020b] Stack Exchange Inc: *Stack Overflow Trends*. 2020. – URL https://insights.stackoverflow.com/trends?tags=gatsby%2Cnext.js%2Cnest%2Cnuxt.js. – Zugriffsdatum: 2020-01-13

[Starke 2015] Starke, Gernot: *Effektive Softwarearchitekturen : ein praktischer Leitfaden*. 7., überarbeitete Auflage. München : Hanser, 2015

[StatCounter 2019a] StatCounter: *Browser Version Market Share Worldwide | StatCounter Global Stats*. 2019. – URL https://gs.statcounter.com/browser-version-market-share. – Zugriffsdatum: 2020-01-26

[StatCounter 2019b] StatCounter: *Desktop Operating System Market Share Worldwide | StatCounter Global Stats*. 2019. – URL https://gs.statcounter.

com/os-market-share/desktop/worldwide#monthly-201411-201910. – Zugriffsdatum: 2019-11-23

[StatCounter 2019c] StatCounter: *Mobile Operating System Market Share Worldwide | StatCounter Global Stats*. 2019. – URL https://gs.statcounter.com/os-market-share/mobile/worldwide#monthly-201411-201910. – Zugriffsdatum: 2019-11-23

[Steiner 2018] Steiner, Thomas: What is in a Web View: An Analysis of Progressive Web App Features When the Means of Web Access is not a Web Browser. In: *Companion Proceedings of the The Web Conference 2018* International World Wide Web Conferences Steering Committee (Veranst.), 2018, S. 789–796

[Steyer 2016] Steyer, Ralph: *WordPress : Einführung in das Content Management System*. Wiesbaden : Springer Vieweg, 2016 (SpringerLink)

[Tailwind o. J.] Tailwind: *Responsive Design - Tailwind CSS*. o. J.. – URL https://tailwindcss.com/docs/responsive-design/. – Zugriffsdatum: 2020-02-24

[The Apache Software Foundation o. J.] The Apache Software Foundation: *Apache Cordova*. o. J.. – URL https://cordova.apache.org/. – Zugriffsdatum: 2019-11-09

[UX Alpaca 2016] UX Alpaca: *What is the difference between fixed, fluid, adaptive and responsive layouts and why should I care?* 2016. – URL https://medium.com/@space.alpaca/so-what-exactly-is-the-difference-between-fixed-fluid-adaptive-and-responsive-layouts-and-why-3773272d8481. – Zugriffsdatum: 2019-11-24

[Vasi 2019] Vasi, Ioana: *The State of Kubernetes Cloud Providers in 2019 | Presslabs*. 2019. – URL https://www.presslabs.com/blog/kubernetes-cloud-providers-2019/. – Zugriffsdatum: 2020-01-23

[Vishal und Kushwaha 2018] Vishal, Kumar ; Kushwaha, Ajay S.: Mobile Application Development Research Based on Xamarin Platform. In: *2018 4th International Conference on Computing Sciences (ICCS)* IEEE (Veranst.), 2018, S. 115–118

[Voss 2019] Voss, Laurie: *JavaScript: Who, Where, What, Why and Next*. 2019. – URL https://slides.com/seldo/web-directions-code-leaders#/. – Zugriffsdatum: 2020-01-12

[Vue.js Contributors o. J.] Vue.js Contributors: *TypeScript Support — Vue.js*. o. J.. – URL https://vuejs.org/v2/guide/typescript.html. – Zugriffsdatum: 2020-01-13

[Walton o. J.] Walton, Philip: *User-centric Performance Metrics | Web Fundamentals*. o. J.. – URL https://developers.google.com/web/fundamentals/performance/user-centric-performance-metrics. – Zugriffsdatum: 2020-01-13

[Warcholinski o. J.] Warcholinski, Matt: *10 Famous Apps Built with React Native - Blog Brainhub.eu*. o. J.. – URL https://brainhub.eu/blog/react-native-apps/. – Zugriffsdatum: 2019-11-09

[Wellman 2018] Wellman: *Dan Wellman's answer to Is it mandatory to learn TypeScript while learning Angular 5? - Quora*. 2018. – URL https://www.quora.com/Is-it-mandatory-to-learn-TypeScript-while-learning-Angular-5/answer/Dan-Wellman-2. – Zugriffsdatum: 2020-01-13

[Wenjun 2011] Wenjun, Liu: Performance Study for Java Virtual Machine in Embedded Systems. In: *2011 International Conference on Internet Computing and Information Services* IEEE (Veranst.), 2011, S. 97–99

[Wieruch 2017] Wieruch, Robin: *Redux vs MobX without Confusion - RWieruch*. 2017. – URL https://www.robinwieruch.de/redux-mobx. – Zugriffsdatum: 2020-01-14

[Willmot und Hoyle 2018] Willmot, Tom ; Hoyle, Joe: *Talking to 25% of the Web*. 2018. – URL https://humanmade.com/uploads/sites/3/2018/11/Talking-to-28-of-the-Web.pdf. – Zugriffsdatum: 2019-11-03

[Wirfs-Brock u. a. 1993] Wirfs-Brock, Rebecca ; Wilkerson, Brian ; Wiener, Lauren: *Objektorientiertes Software-Design*. München : Hanser Fachbuchverlag, 1993

[WordPress Foundation 2012] WordPress Foundation: *User:Lastnode/Wordpress CMS « WordPress Codex*. 2012. – URL https://codex.wordpress.org/User:Lastnode/Wordpress_CMS. – Zugriffsdatum: 2019-11-03

[WordPress Foundation 2015] WordPress Foundation: *The Story Behind the New WordPress.com | Developer Resources*. 2015. – URL

https://developer.wordpress.com/2015/11/23/the-story-behind-the-new-wordpress-com/. – Zugriffsdatum: 2019-12-11

[WordPress Foundation o. J.a] WordPress Foundation: *Adding Custom Endpoints | REST API Handbook | WordPress Developer Resources.* o. J.. – URL https://developer.wordpress.org/rest-api/extending-the-rest-api/adding-custom-endpoints/. – Zugriffsdatum: 2019-12-11

[WordPress Foundation o. J.b] WordPress Foundation: *Appearance Menus Screen | WordPress.org.* o. J.. – URL https://wordpress.org/support/article/appearance-menus-screen/. – Zugriffsdatum: 2019-12-10

[WordPress Foundation o. J.c] WordPress Foundation: *Categories — Support — WordPress.com.* o. J.. – URL https://en.support.wordpress.com/posts/categories/. – Zugriffsdatum: 2019-12-10

[WordPress Foundation o. J.d] WordPress Foundation: *Comments in WordPress | WordPress.org.* o. J.. – URL https://wordpress.org/support/article/comments-in-wordpress/. – Zugriffsdatum: 2019-12-10

[WordPress Foundation o. J.e] WordPress Foundation: *FAQ About WordPress « WordPress Codex.* o. J.. – URL https://codex.wordpress.org/FAQ_About_WordPress. – Zugriffsdatum: 2019-12-10

[WordPress Foundation o. J.f] WordPress Foundation: *Frequently Asked Questions | Block Editor Handbook | WordPress Developer Resources.* o. J.. – URL https://developer.wordpress.org/block-editor/contributors/faq/. – Zugriffsdatum: 2019-12-14

[WordPress Foundation o. J.g] WordPress Foundation: *GNU Public License | WordPress.org.* o. J.. – URL https://wordpress.org/about/license/. – Zugriffsdatum: 2019-12-10

[WordPress Foundation o. J.h] WordPress Foundation: *Introducing the New WordPress.com.* o. J.. – URL https://developer.wordpress.com/calypso/. – Zugriffsdatum: 2019-12-11

[WordPress Foundation o. J.i] WordPress Foundation: *Introduction to Plugin Development | Plugin Developer Handbook | WordPress Developer Resources.* o. J.. – URL https://developer.wordpress.org/plugins/intro/. – Zugriffsdatum: 2019-12-11

[WordPress Foundation o. J.j] WordPress Foundation: *Multilingual WordPress | WordPress.org*. o. J.. – URL https://wordpress.org/support/article/multilingual-wordpress/. – Zugriffsdatum: 2019-12-10

[WordPress Foundation o. J.k] WordPress Foundation: *Network Admin Users Screen « WordPress Codex*. o. J.. – URL https://codex.wordpress.org/Network_Admin_Users_Screen. – Zugriffsdatum: 2019-12-10

[WordPress Foundation o. J.l] WordPress Foundation: *Pages | WordPress.org*. o. J.. – URL https://wordpress.org/support/article/pages/. – Zugriffsdatum: 2019-12-10

[WordPress Foundation o. J.m] WordPress Foundation: *Plugin API « WordPress Codex*. o. J.. – URL https://codex.wordpress.org/Plugin_API. – Zugriffsdatum: 2019-12-11

[WordPress Foundation o. J.n] WordPress Foundation: *Post Types | WordPress.org*. o. J.. – URL https://wordpress.org/support/article/post-types/. – Zugriffsdatum: 2019-12-10

[WordPress Foundation o. J.o] WordPress Foundation: *Reference | REST API Handbook | WordPress Developer Resources*. o. J.. – URL https://developer.wordpress.org/rest-api/reference/. – Zugriffsdatum: 2019-12-11

[WordPress Foundation o. J.p] WordPress Foundation: *Roles and Capabilities | WordPress.org*. o. J.. – URL https://wordpress.org/support/article/roles-and-capabilities/. – Zugriffsdatum: 2019-12-10

[WordPress Foundation o. J.q] WordPress Foundation: *Taxonomies | WordPress.org*. o. J.. – URL https://wordpress.org/support/article/taxonomies/. – Zugriffsdatum: 2019-12-10

[WordPress Foundation o. J.r] WordPress Foundation: *Theme Development « WordPress Codex*. o. J.. – URL https://codex.wordpress.org/Theme_Development. – Zugriffsdatum: 2019-12-11

[WordPress Foundation o. J.s] WordPress Foundation: *What is a Theme? | Theme Developer Handbook | WordPress Developer Resources*. o. J.. – URL https://developer.wordpress.org/themes/getting-started/what-is-a-theme/. – Zugriffsdatum: 2019-12-11

[WordPress Foundation o. J.t] WordPress Foundation: *The WordPress Codebase – Make WordPress Core*. o. J.. – URL https://make.wordpress.org/core/handbook/contribute/codebase/. – Zugriffsdatum: 2019-12-10

[Xanthopoulos und Xinogalos 2013] Xanthopoulos, Spyros ; Xinogalos, Stelios: A comparative analysis of cross-platform development approaches for mobile applications. In: *Proceedings of the 6th Balkan Conference in Informatics* ACM (Veranst.), 2013, S. 213–220

[Zealous System 2019] Zealous System: *Learn Why These 5 Popular Startups Rewrote Their Android Apps From Java to Kotlin - By*. 2019. – URL https://hackernoon.com/learn-why-these-5-popular-startups-rewrote-their-android-apps-from-java-to-kotlin-d60ee01ba7f1. – Zugriffsdatum: 2019-11-09

[ZEIT, Inc. o. J.a] ZEIT, Inc.: *Automatic Static Optimization - Documentation | Next.js*. o. J.. – URL https://nextjs.org/docs/advanced-features/automatic-static-optimization. – Zugriffsdatum: 2020-01-21

[ZEIT, Inc. o. J.b] ZEIT, Inc.: *Dynamic Import - Documentation | Next.js*. o. J.. – URL https://nextjs.org/docs/advanced-features/dynamic-import. – Zugriffsdatum: 2020-02-06

A Anhang

A.1 Designsprachen

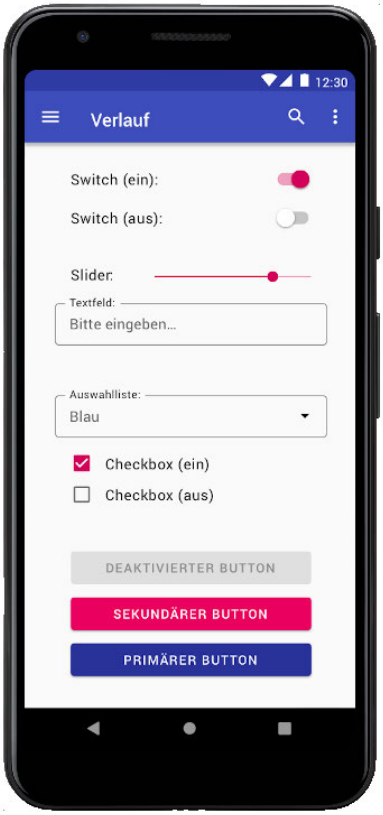

Abbildung A.1: Designsprache Material Design (Android) (Quelle: eigene Darstellung, erstellt mit Android Studio)

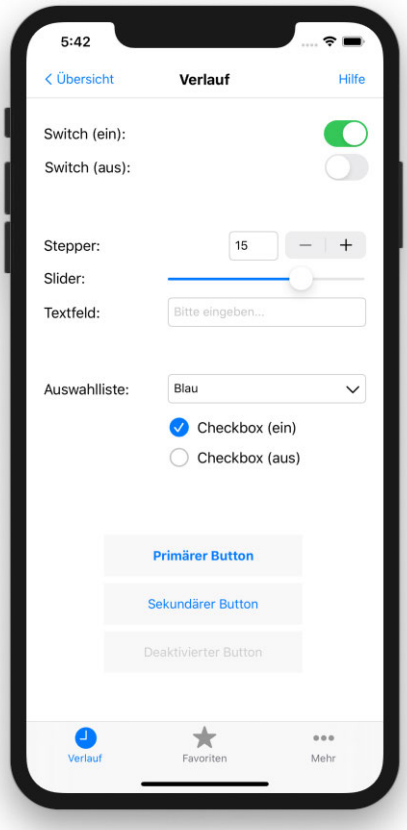

Abbildung A.2: Designsprache iOS (Quelle: eigene Darstellung, erstellt mit Xcode)

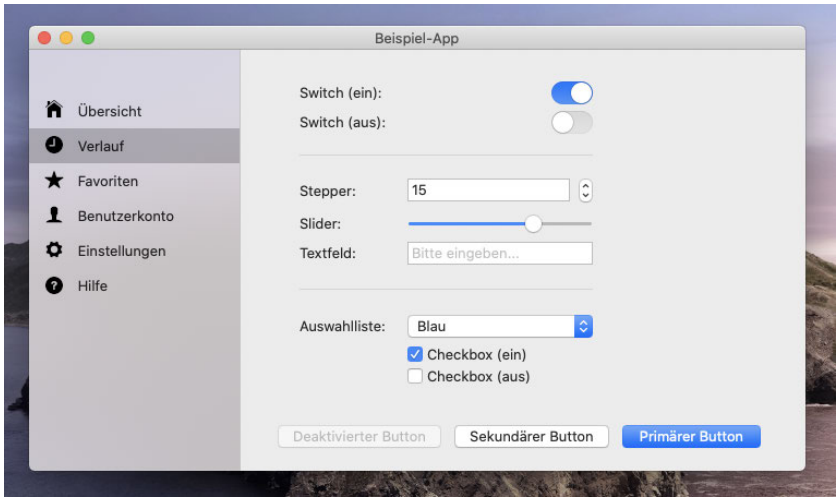

Abbildung A.3: Designsprache Aqua (macOS)
(Quelle: eigene Darstellung, erstellt mit Xcode)

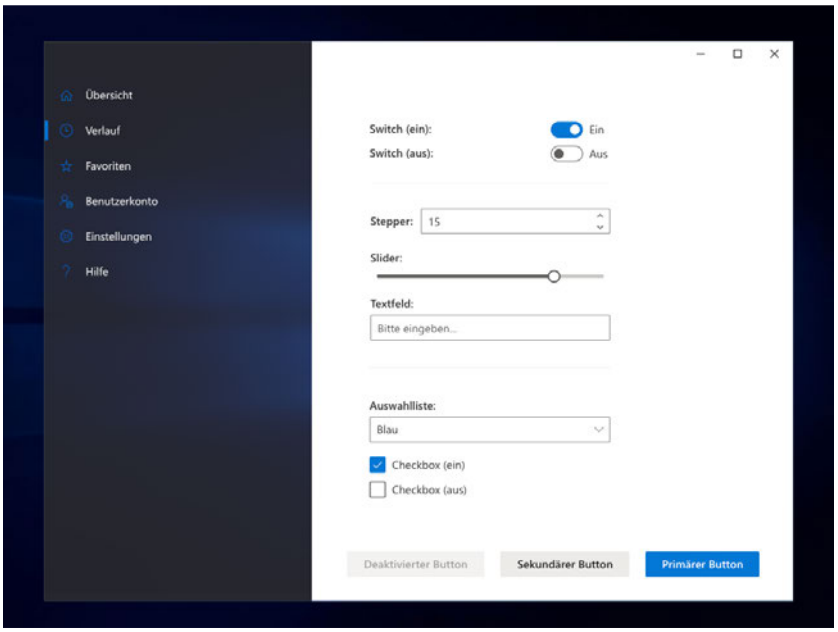

Abbildung A.4: Designsprache Fluent (Windows)
(Quelle: eigene Darstellung, erstellt mit Visual Studio)

A.2 Marktanteile Frontend-Technologien

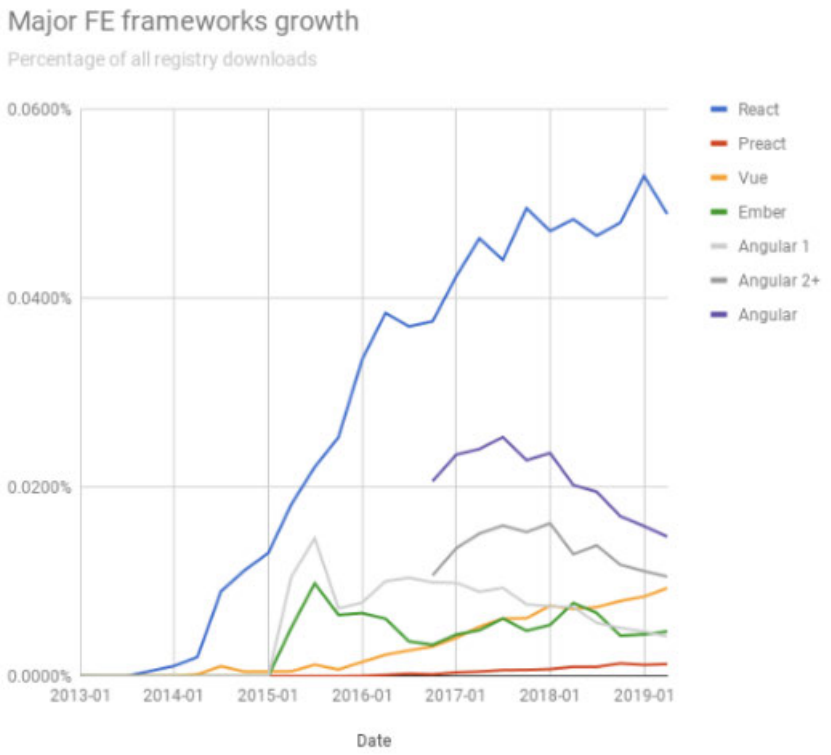

Abbildung A.5: Relative Verbreitung Frontend-Technologien (NPM)
(Quelle: Voss (2019))

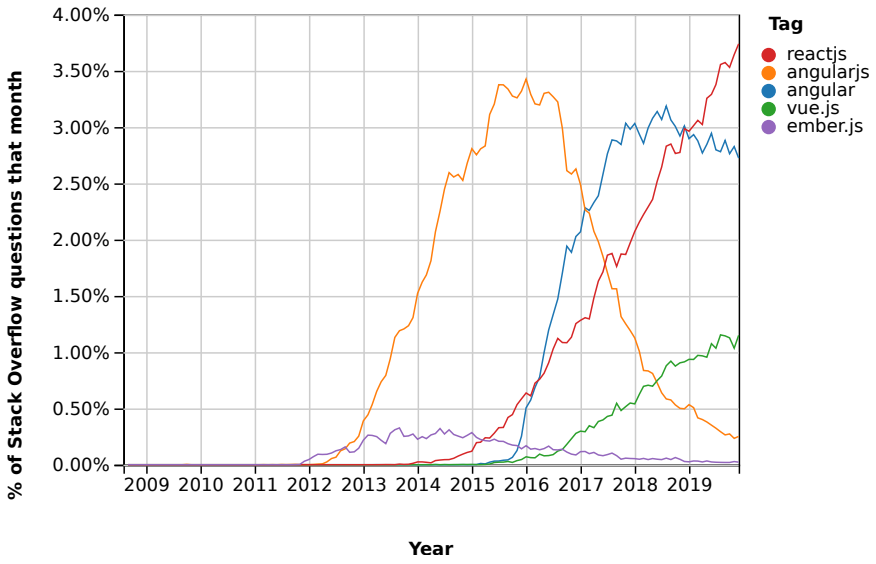

Abbildung A.6: Relative Verbreitung Frontend-Technologien (Stack Overflow) (Quelle: Stack Exchange Inc (2020a))

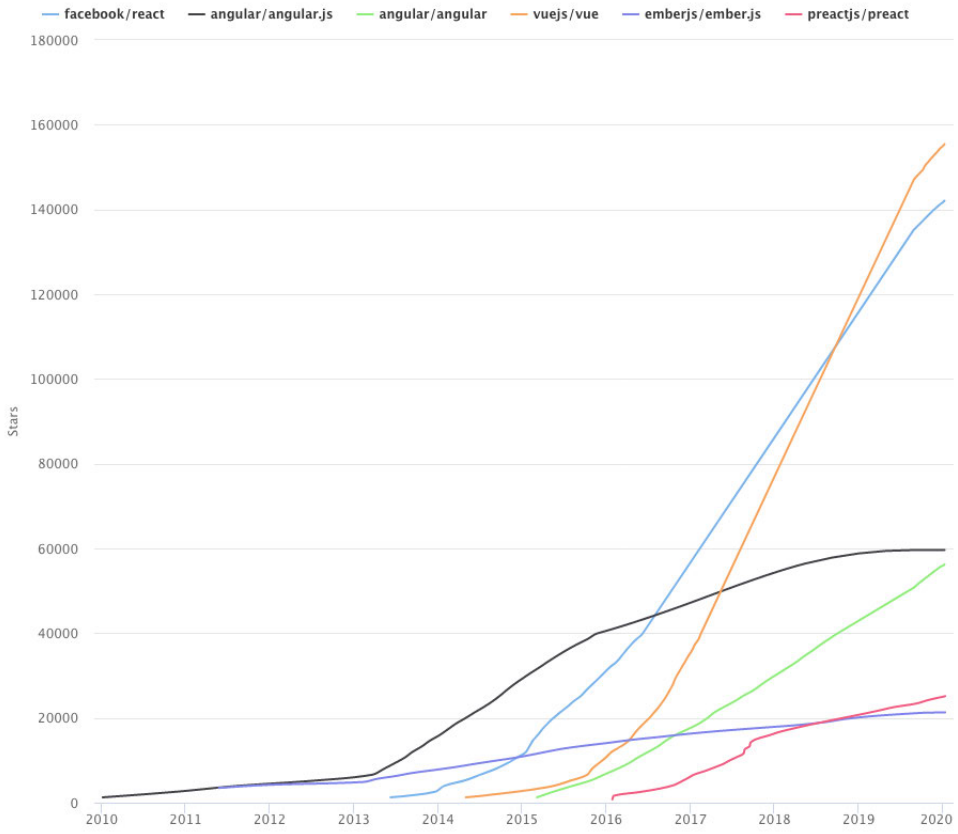

Abbildung A.7: Absolute Download-Zahlen Frontend-Technologien (GitHub) (Quelle: Nowak (2020a))

A.3 Performanz Frontend-Technologien

Duration in milliseconds ± standard deviation (Slowdown = Duration / Fastest)

Name	preact-v8.2.6-keyed	vue-v2.5.16-keyed	angular-v6.1.0-keyed	react-v16.4.1-keyed	angularjs-v1.7.2-keyed	ember-v3.3.0-keyed
create rows Duration for creating 1000 rows after the page loaded.	174.6 ± 9.3 (1.0)	182.1 ± 7.6 (1.0)	185.2 ± 10.2 (1.1)	180.5 ± 7.3 (1.0)	225.5 ± 10.7 (1.3)	406.8 ± 15.8 (2.3)
replace all rows Duration for updating all 1000 rows of the table (with 5 warmup iterations).	157.3 ± 4.5 (1.0)	158.8 ± 2.7 (1.0)	161.2 ± 2.7 (1.0)	157.3 ± 2.0 (1.0)	201.4 ± 6.4 (1.3)	269.1 ± 23.6 (1.7)
partial update Time to update the text of every 10th row (with 5 warmup iterations) for a table with 10k rows.	96.2 ± 3.0 (1.4)	156.4 ± 9.8 (2.3)	68.8 ± 3.7 (1.0)	81.9 ± 2.7 (1.2)	90.0 ± 5.8 (1.3)	134.9 ± 5.9 (2.0)
select row Duration to highlight a row in response to a click on the row. (with 5 warmup iterations).	10.5 ± 3.5 (1.0)	10.6 ± 2.0 (1.0)	7.9 ± 4.3 (1.0)	10.3 ± 2.1 (1.0)	9.7 ± 1.3 (1.0)	8.7 ± 1.9 (1.0)
swap rows Time to swap 2 rows on a 1K table. (with 5 warmup iterations).	23.1 ± 2.9 (1.2)	20.0 ± 2.9 (1.0)	105.8 ± 1.8 (5.3)	106.5 ± 1.9 (5.3)	106.8 ± 1.1 (5.3)	122.0 ± 2.9 (6.1)
remove row Duration to remove a row. (with 5 warmup iterations).	49.3 ± 0.6 (1.0)	54.2 ± 2.2 (1.2)	47.1 ± 3.0 (1.0)	49.6 ± 0.8 (1.1)	50.4 ± 0.8 (1.1)	55.7 ± 1.0 (1.2)
create many rows Duration to create 10,000 rows	1,852.0 ± 51.6 (1.2)	1,603.2 ± 34.8 (1.0)	1,693.9 ± 70.1 (1.1)	1,935.4 ± 33.6 (1.2)	2,126.4 ± 71.4 (1.3)	2,931.9 ± 42.9 (1.8)
append rows to large table Duration for adding 1000 rows on a table of 10,000 rows.	271.7 ± 3.8 (1.1)	342.5 ± 6.0 (1.4)	243.3 ± 6.3 (1.0)	268.6 ± 6.9 (1.1)	305.7 ± 11.6 (1.3)	403.7 ± 32.5 (1.7)
clear rows Duration to clear the table filled with 10.000 rows.	194.1 ± 2.1 (1.1)	191.9 ± 6.1 (1.1)	263.9 ± 3.0 (1.5)	175.4 ± 4.1 (1.0)	419.2 ± 6.9 (2.4)	203.1 ± 3.6 (1.2)
slowdown geometric mean	**1.10**	**1.17**	**1.28**	**1.28**	**1.54**	**1.80**

Abbildung A.8: Vergleich der Performanz von Frontend-Technologien (1/2) (Quelle: Steiner (2018))

Startup metrics (lighthouse with mobile simulation)

Name	preact-v8.2.6-keyed	vue-v2.5.16-keyed	angular-v6.1.0-keyed	react-v16.4.1-keyed	angularjs-v1.7.2-keyed	ember-v3.3.0-keyed
consistently interactive a pessimistic TTI - when the CPU and network are both definitely very idle. (no more CPU tasks over 50ms)	1,952.7 ± 0.4 (1.0)	2,252.7 ± 0.2 (1.2)	3,278.5 ± 4.0 (1.7)	2,477.6 ± 0.6 (1.3)	2,854.0 ± 0.7 (1.5)	5,105.5 ± 0.8 (2.6)
script bootup time the total ms required to parse/compile/evaluate all the page's scripts	16.0 ± 0.0 (1.0)	55.1 ± 1.5 (3.4)	235.2 ± 8.5 (14.7)	65.6 ± 2.3 (4.1)	124.6 ± 2.0 (7.8)	419.9 ± 6.6 (26.2)
main thread work cost total amount of time spent doing work on the main thread. includes style/layout/etc.	316.3 ± 61.8 (1.0)	420.6 ± 67.5 (1.3)	679.3 ± 5.3 (2.1)	466.0 ± 3.9 (1.5)	530.6 ± 2.5 (1.7)	884.2 ± 7.6 (2.8)
total byte weight network transfer cost (post-compression) of all the resources loaded into the page.	155,626.0 ± 0.0 (1.0)	215,445.0 ± 0.0 (1.4)	365,497.0 ± 0.0 (2.3)	251,915.0 ± 0.0 (1.6)	328,568.0 ± 0.0 (2.1)	742,719.0 ± 0.0 (4.8)

Memory allocation in MBs ± standard deviation

Name	preact-v8.2.6-keyed	vue-v2.5.16-keyed	angular-v6.1.0-keyed	react-v16.4.1-keyed	angularjs-v1.7.2-keyed	ember-v3.3.0-keyed
ready memory Memory usage after page load.	2.5 ± 0.2 (1.0)	2.7 ± 0.2 (1.1)	6.0 ± 0.0 (2.4)	2.8 ± 0.2 (1.1)	3.4 ± 0.2 (1.4)	6.2 ± 0.1 (2.5)
run memory Memory usage after adding 1000 rows.	5.2 ± 0.0 (1.0)	7.1 ± 0.0 (1.4)	9.8 ± 0.0 (1.9)	6.7 ± 0.0 (1.3)	10.9 ± 0.0 (2.1)	14.6 ± 0.0 (2.8)
update eatch 10th row for 1k rows (5 cycles) Memory usage after clicking update every 10th row 5 times	5.3 ± 0.0 (1.0)	7.2 ± 0.0 (1.4)	9.8 ± 0.0 (1.9)	7.6 ± 0.0 (1.4)	10.9 ± 0.0 (2.1)	14.7 ± 0.0 (2.8)
replace 1k rows (5 cycles) Memory usage after clicking create 1000 rows 5 times	8.0 ± 0.0 (1.1)	7.2 ± 0.0 (1.0)	10.1 ± 0.0 (1.4)	7.9 ± 0.0 (1.1)	11.4 ± 0.1 (1.6)	15.6 ± 0.0 (2.2)
creating/clearing 1k rows (5 cycles) Memory usage after creating and clearing 1000 rows 5 times	5.4 ± 0.0 (1.8)	3.0 ± 0.0 (1.0)	6.4 ± 0.0 (2.1)	3.8 ± 0.0 (1.3)	3.9 ± 0.0 (1.3)	7.3 ± 0.0 (2.5)

Abbildung A.9: Vergleich der Performanz von Frontend-Technologien (2/2) (Quelle: Steiner (2018))

A.4 Marktanteile Rendering-Frameworks

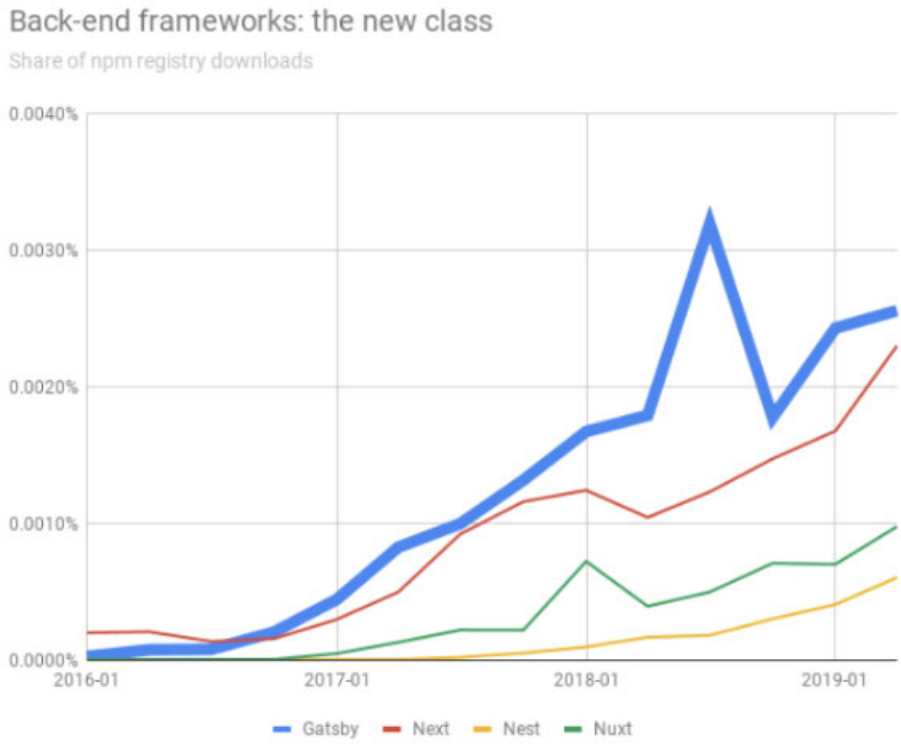

Abbildung A.10: Relative Verbreitung Rendering-Frameworks (NPM) (Quelle: Voss (2019))

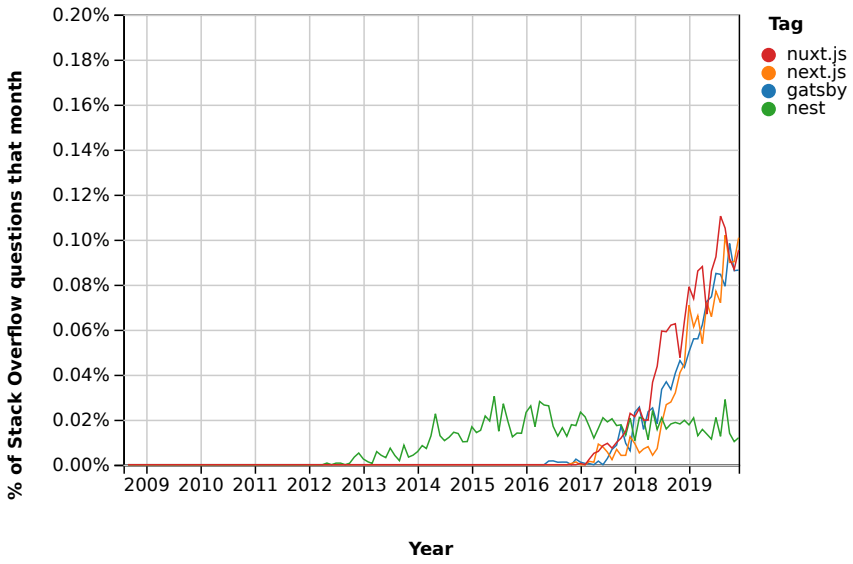

Abbildung A.11: Relative Verbreitung Rendering-Frameworks (Stack Overflow) (Quelle: Stack Exchange Inc (2020b))

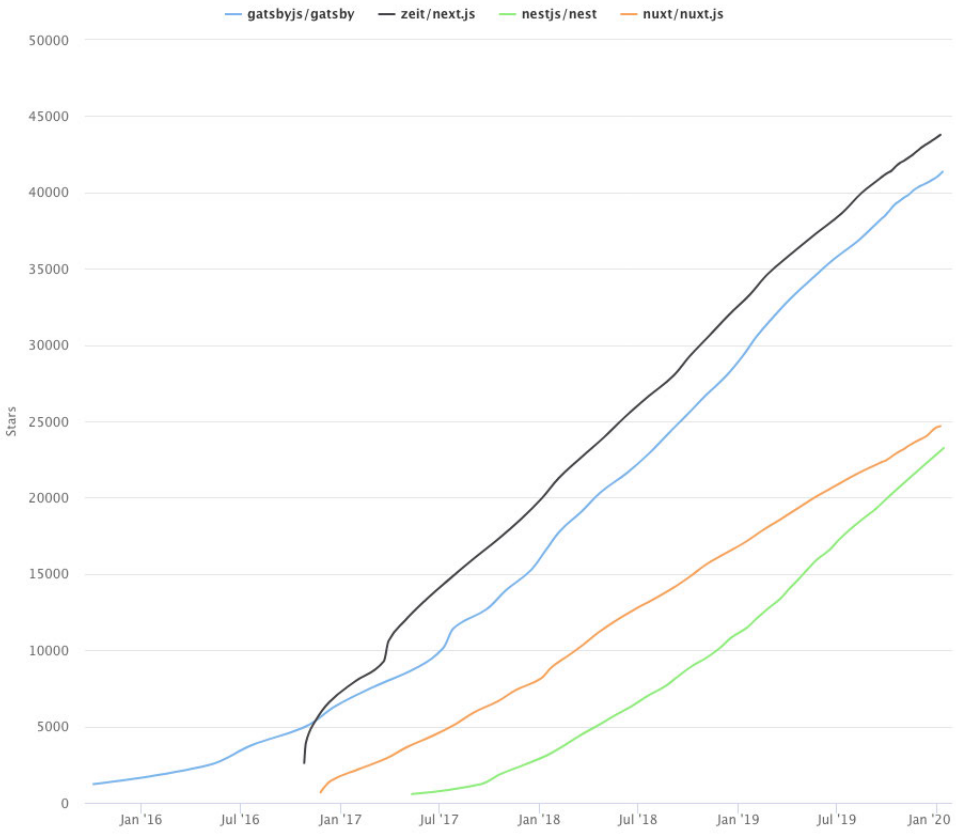

Abbildung A.12: Absolute Download-Zahlen Rendering-Frameworks (GitHub) (Quelle: Nowak (2020b))

A.5 Marktanteile State-Management-Frameworks

Hinweis: Für State-Management-Frameworks stehen in Voss (2019) keine Statistiken bereit. In den Stack Overflow Trends existiert zudem kein Tag für MobX. Aus diesem Grund wird, im Gegensatz zu den anderen Technologie-Arten, nur die Statistik aus Nowak (2020c) aufgeführt. Zusätzlich werden Download-Zahlen der NPM-Registry aus Potter (2020) aufgeführt, zeigen jedoch, im Gegensatz zu den anderen Technologie-Arten, *absolute* Zahlen.

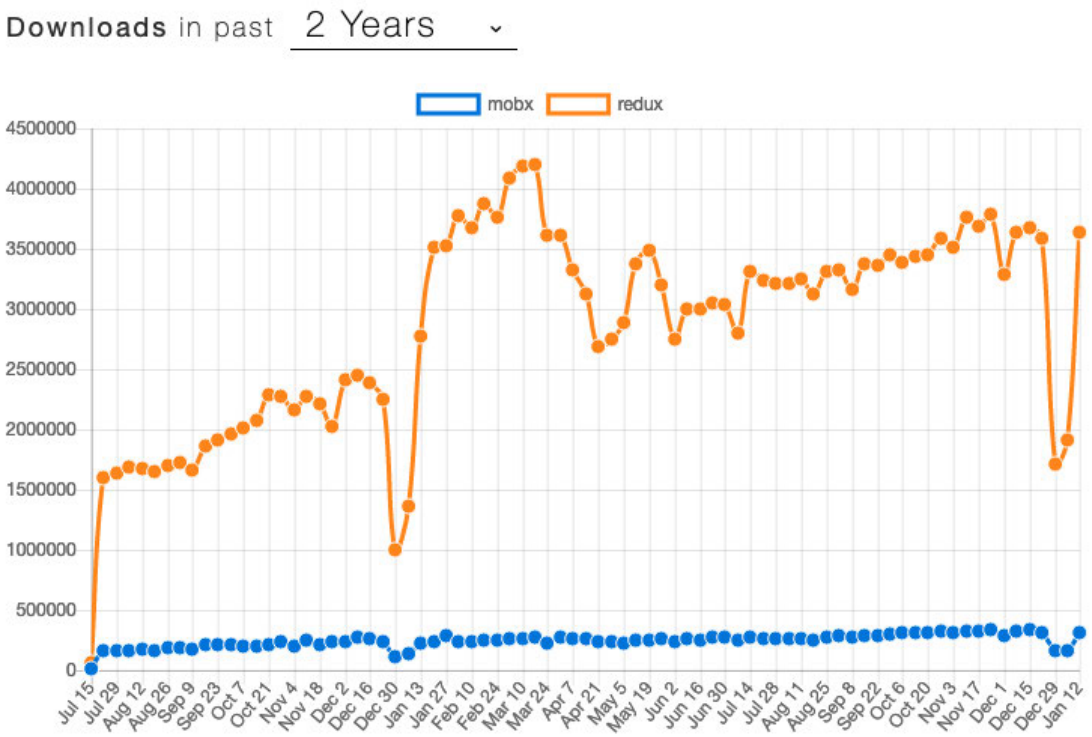

Abbildung A.13: Absolute Download-Zahlen Zustandsverwaltung (NPM) (Quelle: Potter (2020))

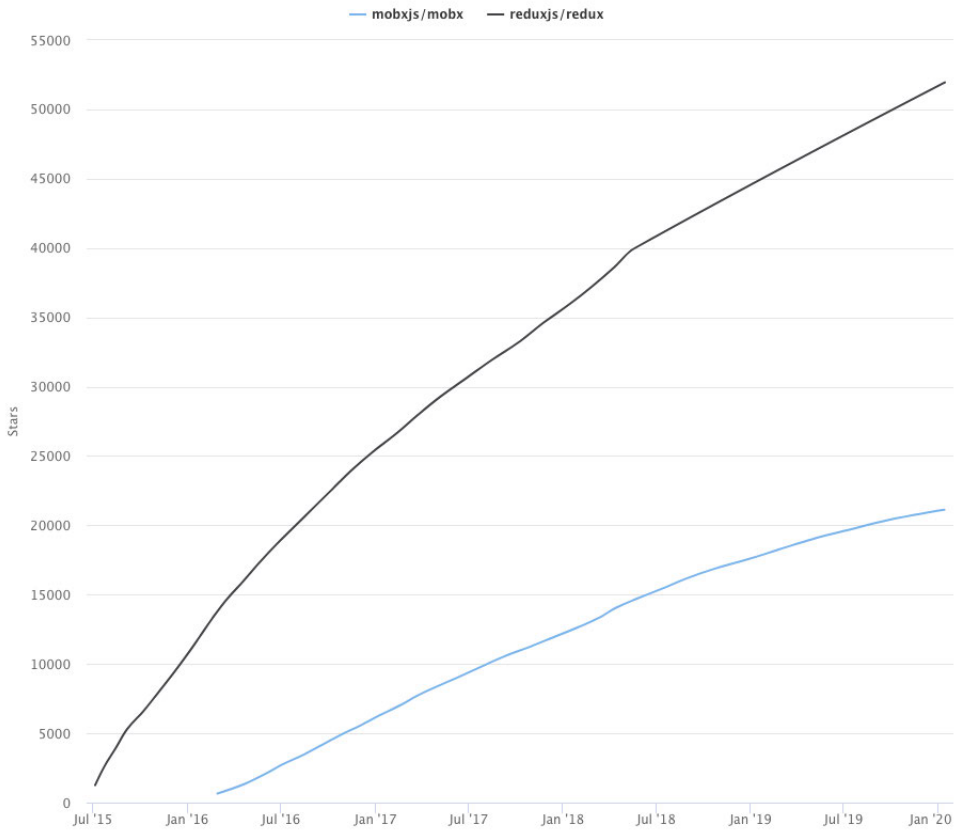

Abbildung A.14: Absolute Download-Zahlen Zustandsverwaltung (GitHub) (Quelle: Nowak (2020c))

A.6 Audit-Ergebnisse Lighthouse vor Performanz-Optimierungen

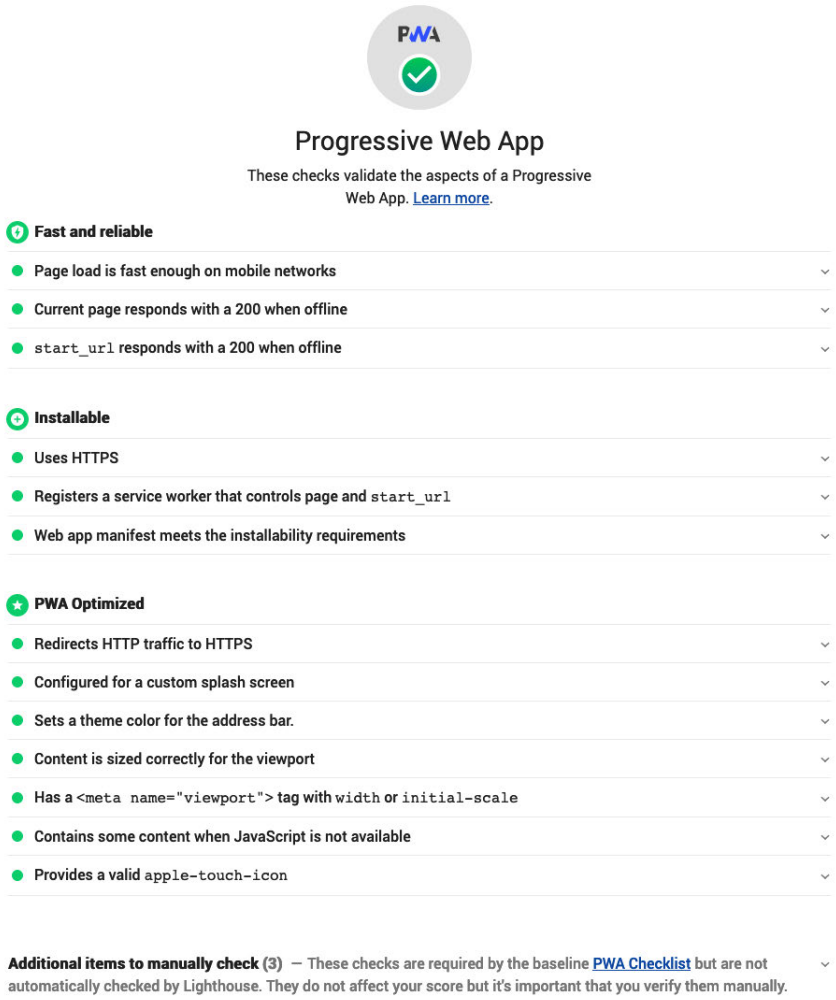

Abbildung A.15: Lighthouse-Audit: Progressive Web App (Quelle: erstellt mit Lighthouse)

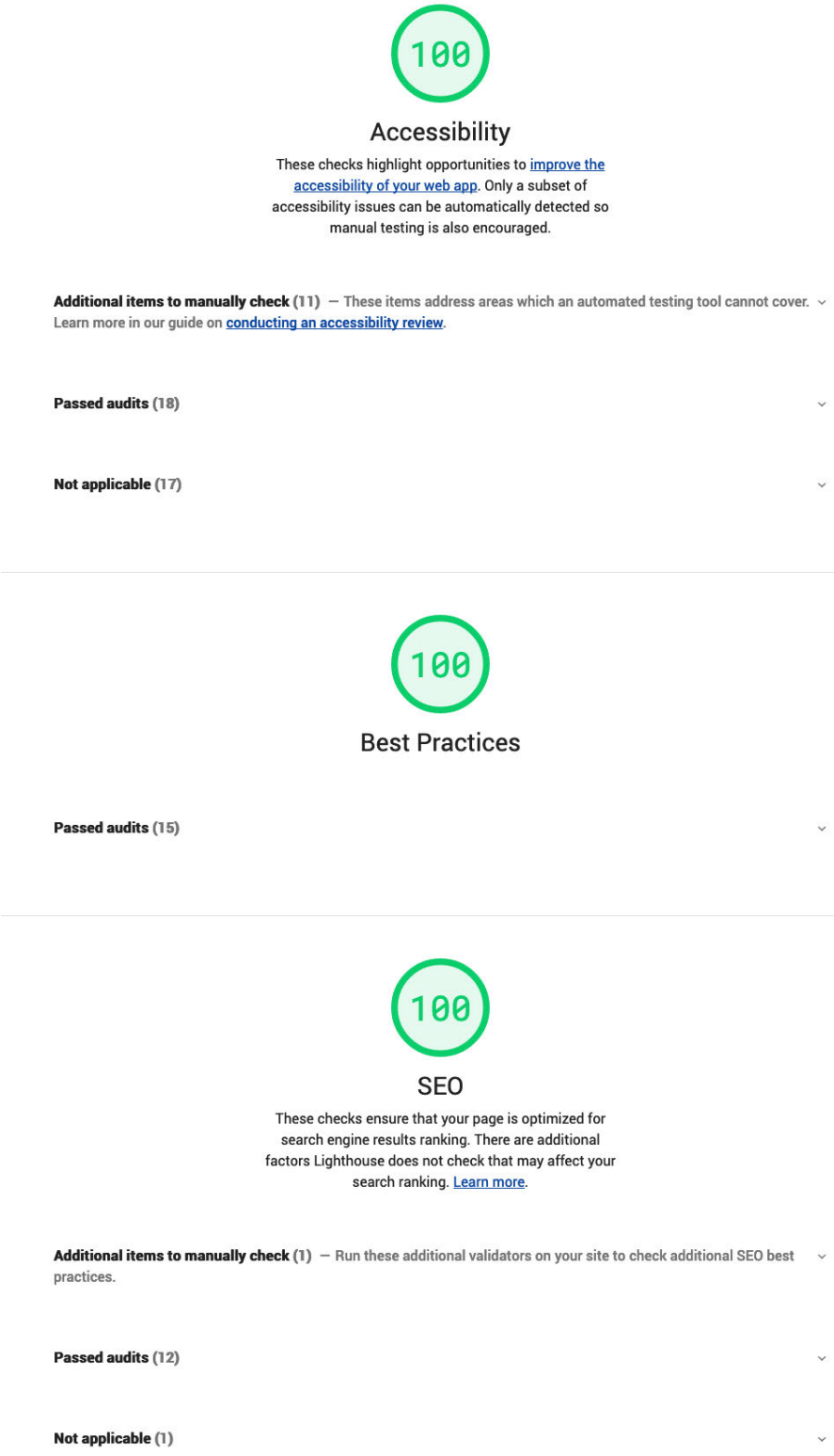

Abbildung A.16: Lighthouse-Audit: Barrierefreiheit, Best Practices und SEO (Quelle: erstellt mit Lighthouse)

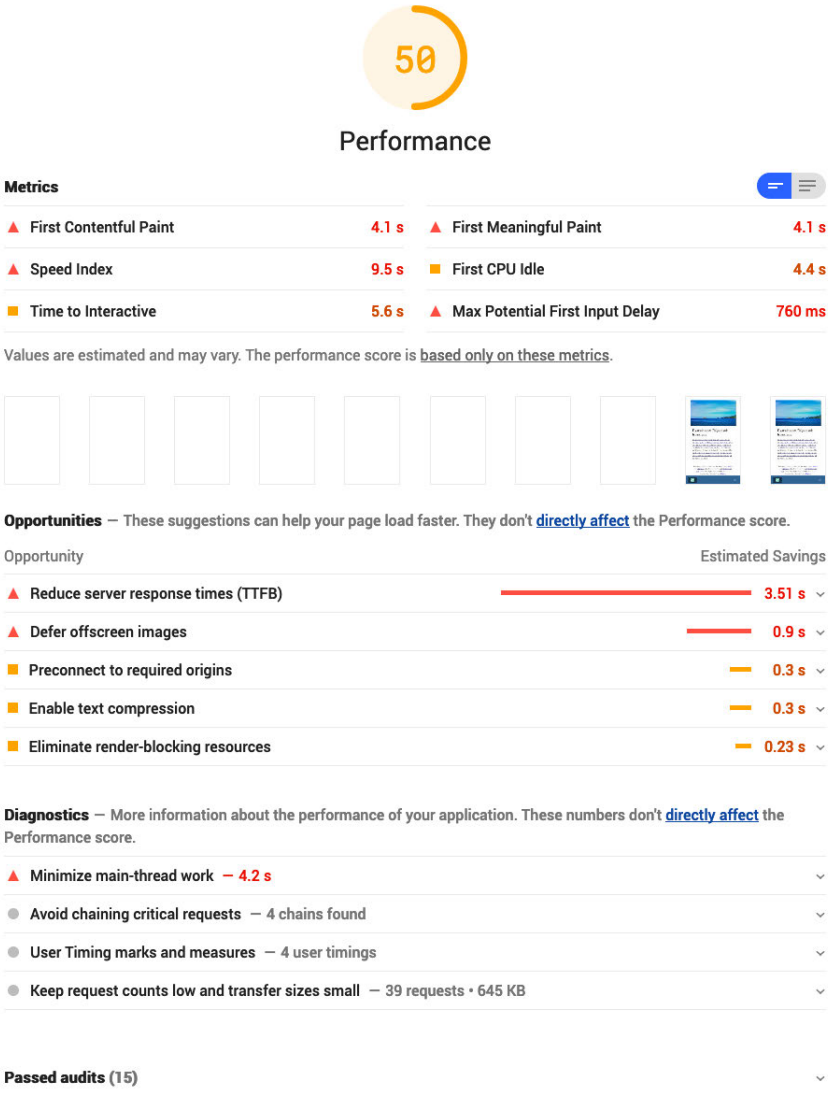

Abbildung A.17: Lighthouse-Audit: Performance
(Quelle: erstellt mit Lighthouse)

A.7 Audit-Ergebnisse Lighthouse nach Performanz-Optimierungen

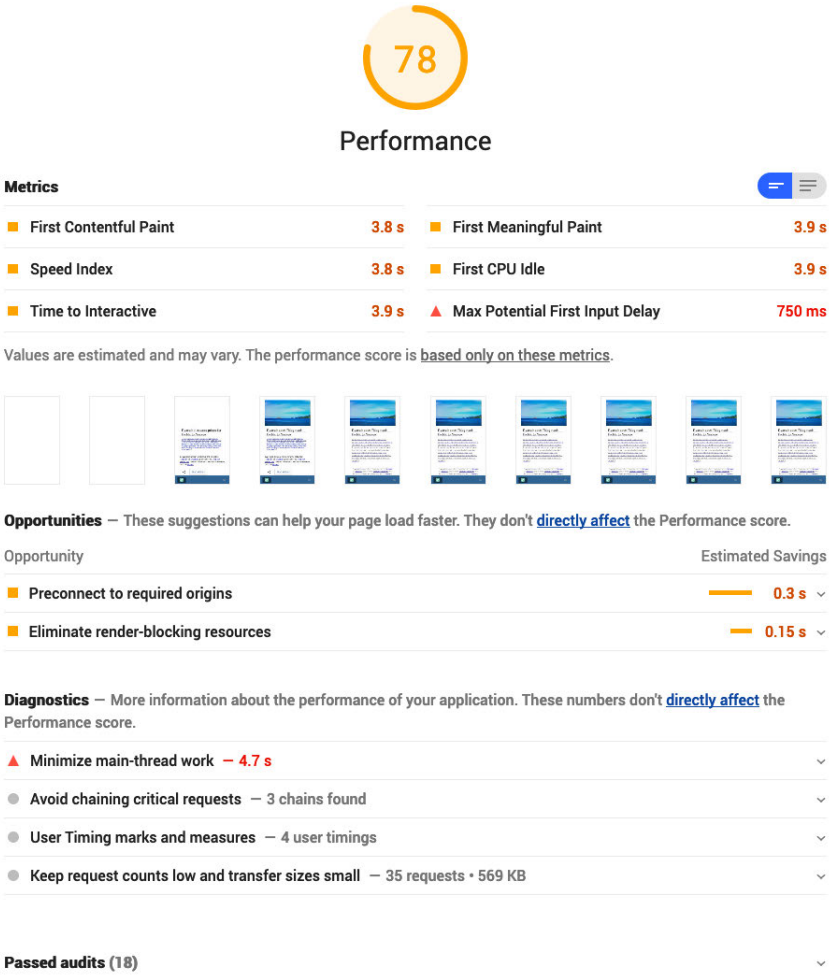

Abbildung A.18: Lighthouse-Audit: Performance nach Verbesserung 1 (vgl. Unterabschnitt 9.2.4, Quelle: erstellt mit Lighthouse)

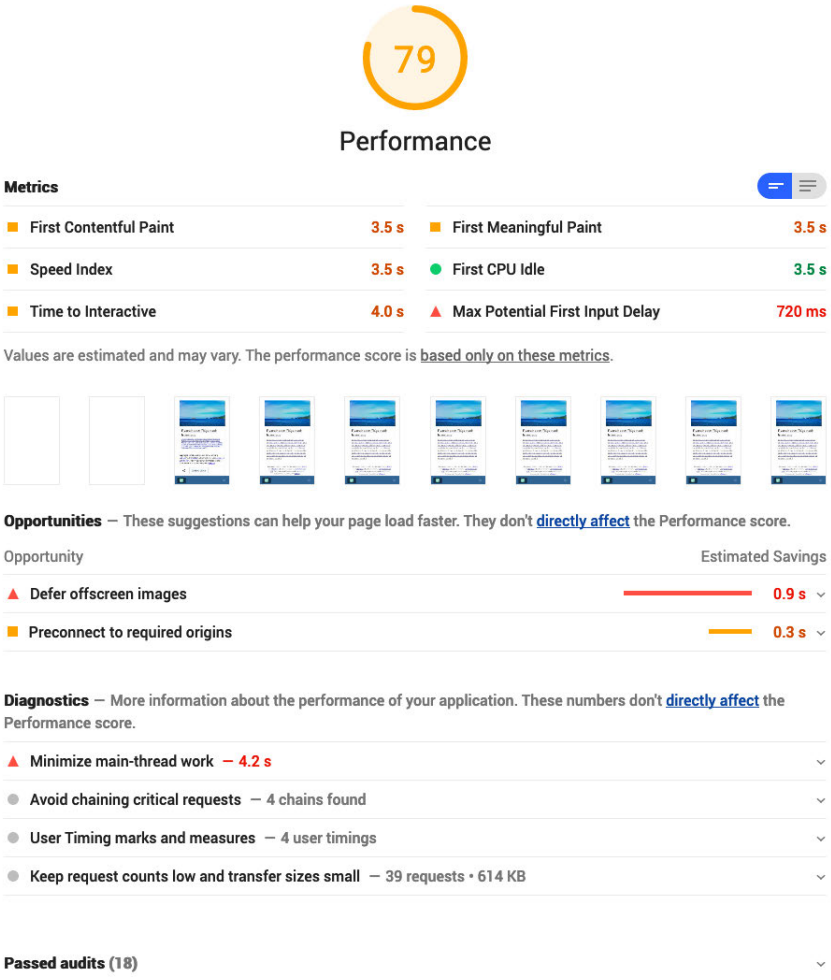

Abbildung A.19: Lighthouse-Audit: Performance nach Verbesserung 2 (vgl. Unterabschnitt 9.2.4, Quelle: erstellt mit Lighthouse)

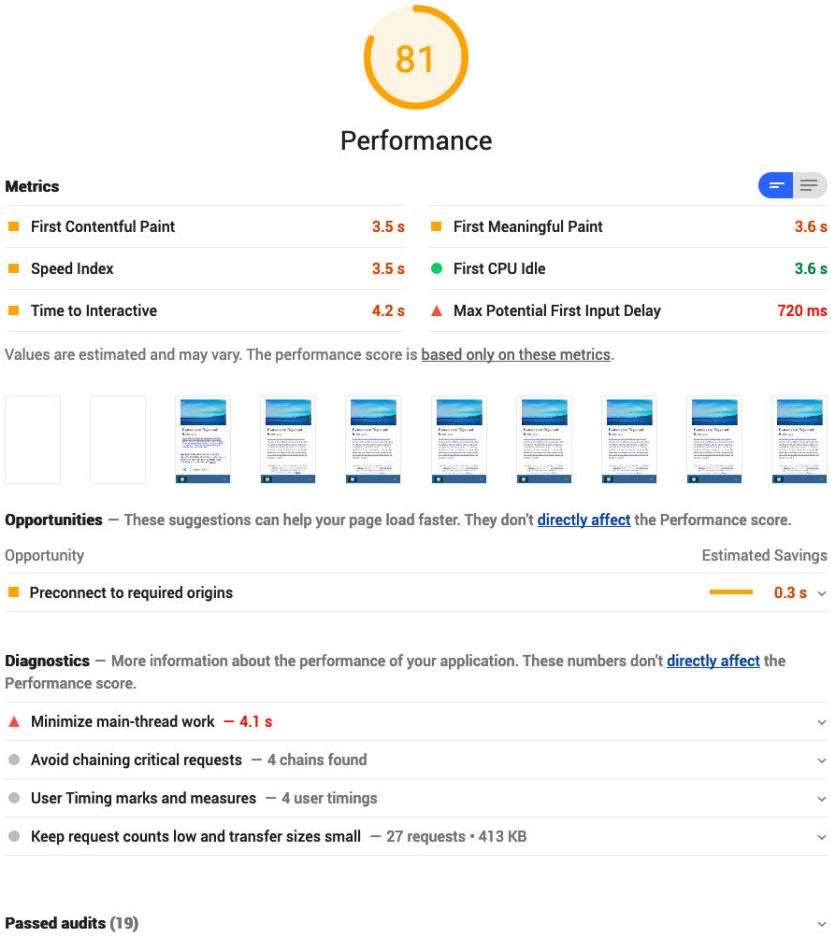

Abbildung A.20: Lighthouse-Audit: Performance nach Verbesserung 3 (vgl. Unterabschnitt 9.2.4, Quelle: erstellt mit Lighthouse)

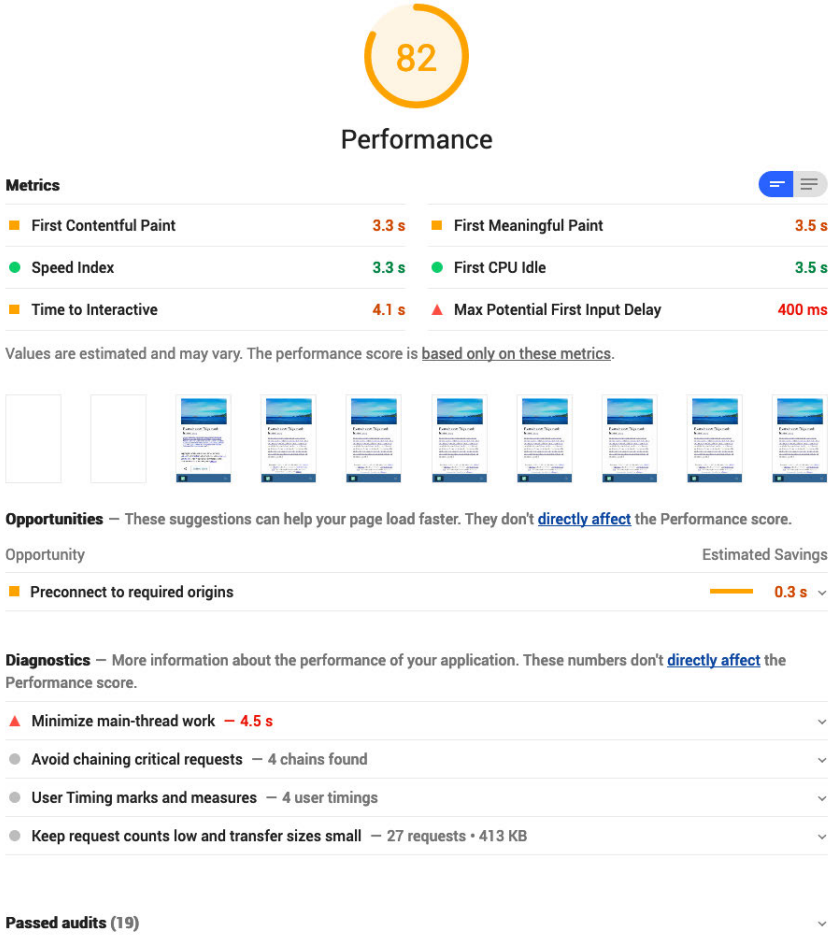

Abbildung A.21: Lighthouse-Audit: Performance nach Verbesserung 4 (vgl. Unterabschnitt 9.2.4, Quelle: erstellt mit Lighthouse)

A.8 Audit-Ergebnisse WooRank

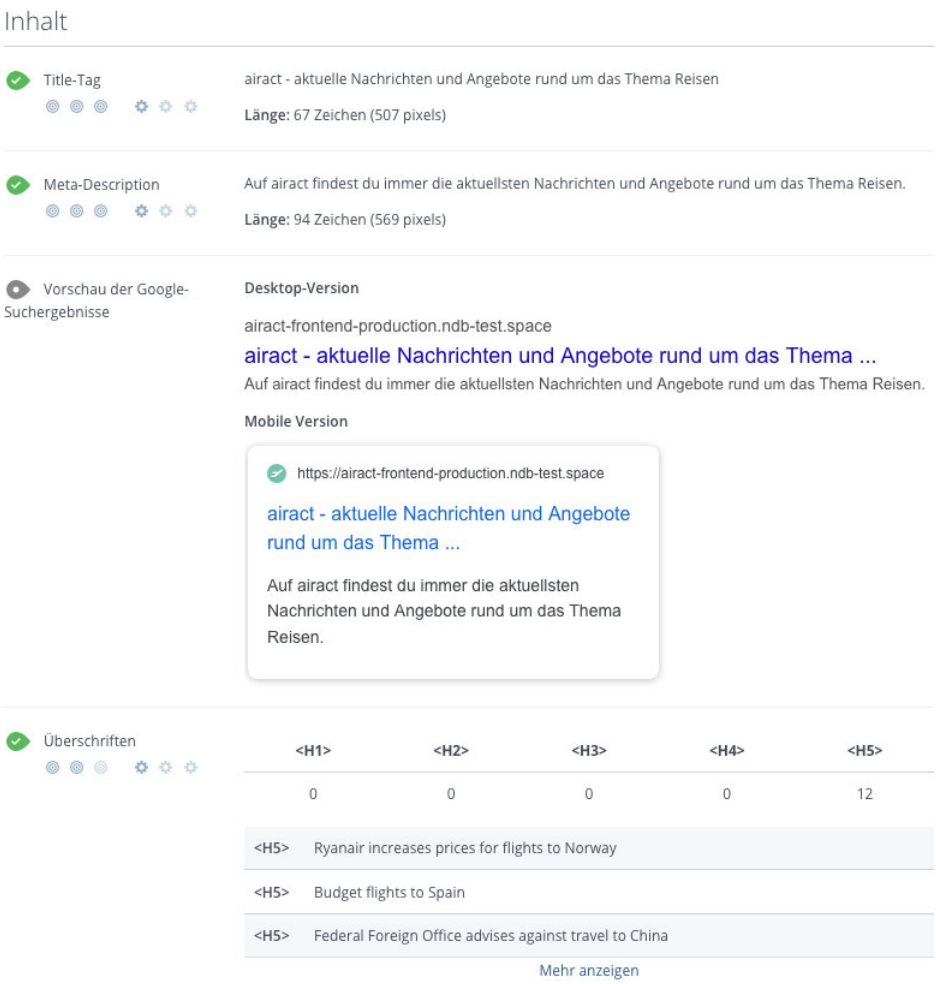

Abbildung A.22: WooRank-Audit: Inhalt (1/2)
(Quelle: erstellt mit WooRank)

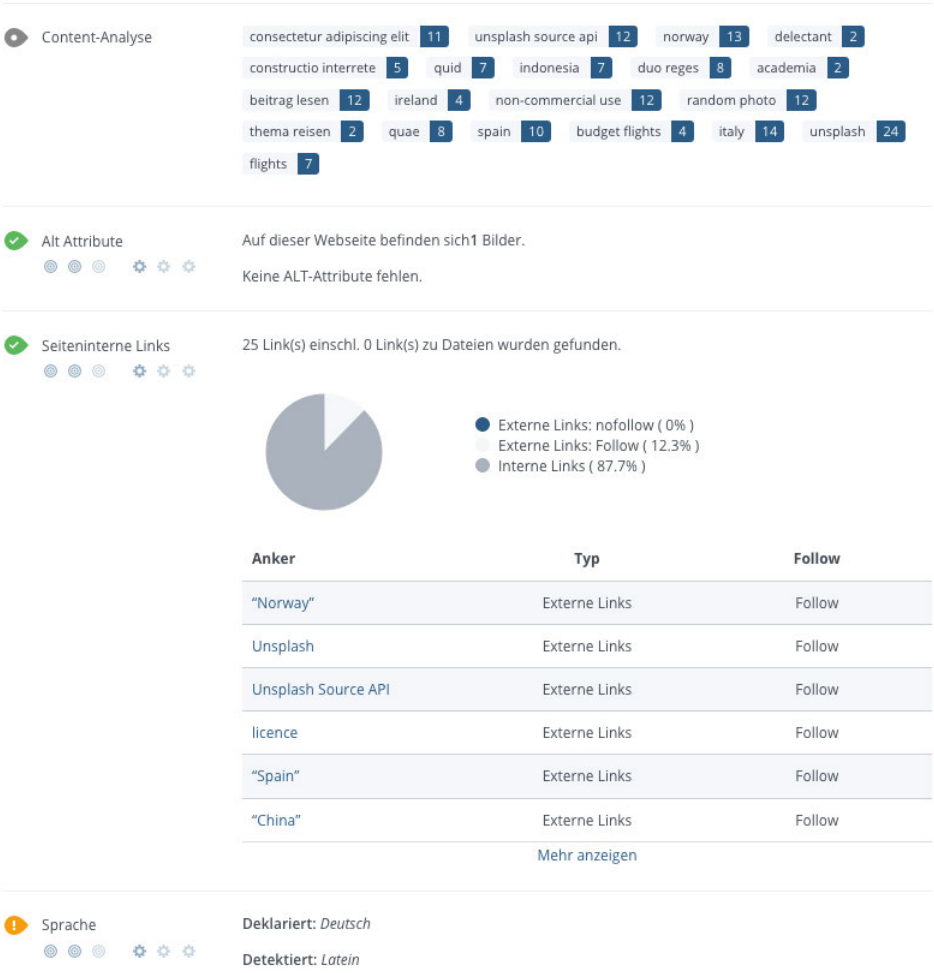

Abbildung A.23: WooRank-Audit: Inhalt (2/2)
(Quelle: erstellt mit WooRank)

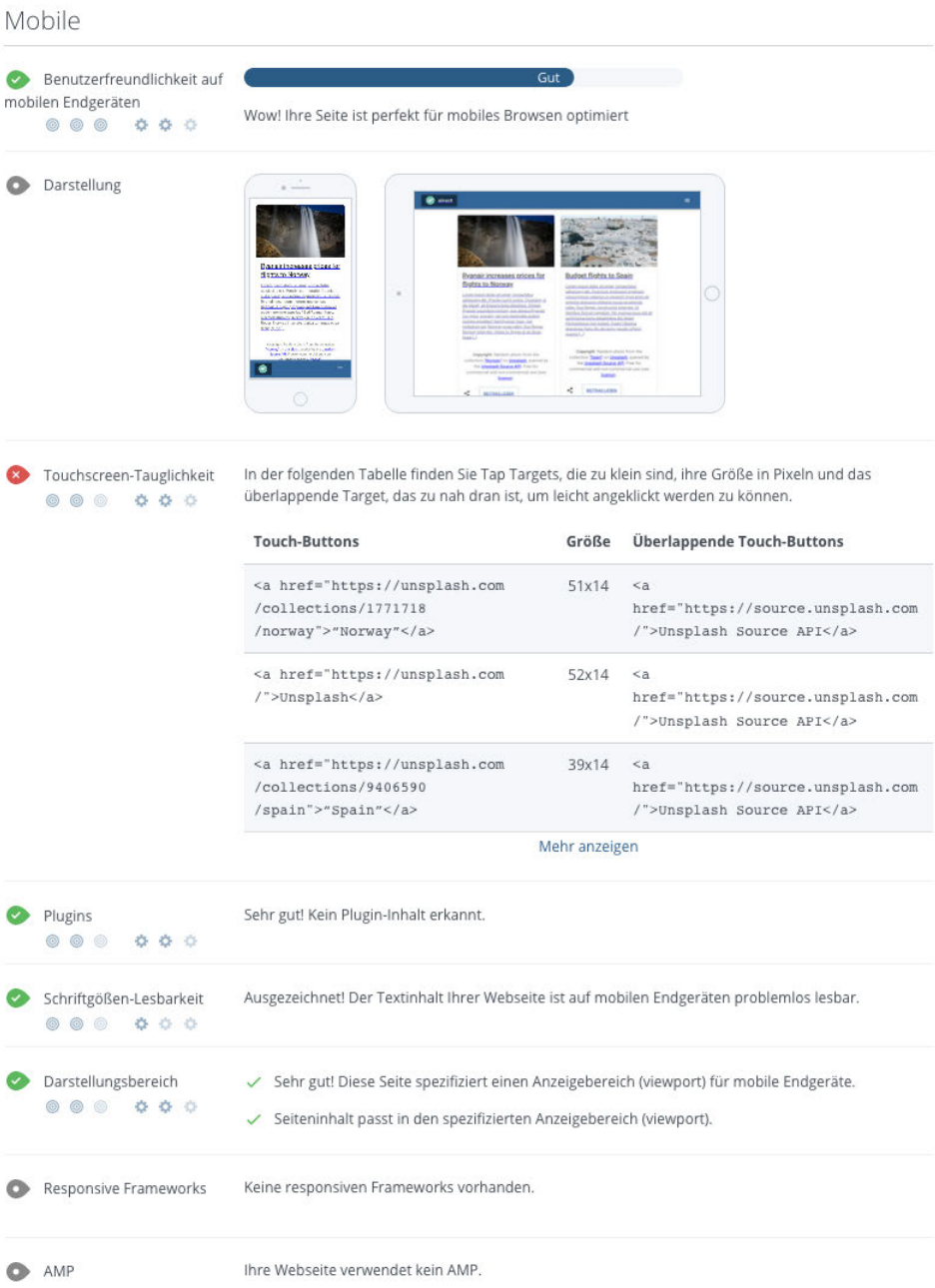

Abbildung A.24: WooRank-Audit: Mobile
(Quelle: erstellt mit WooRank)

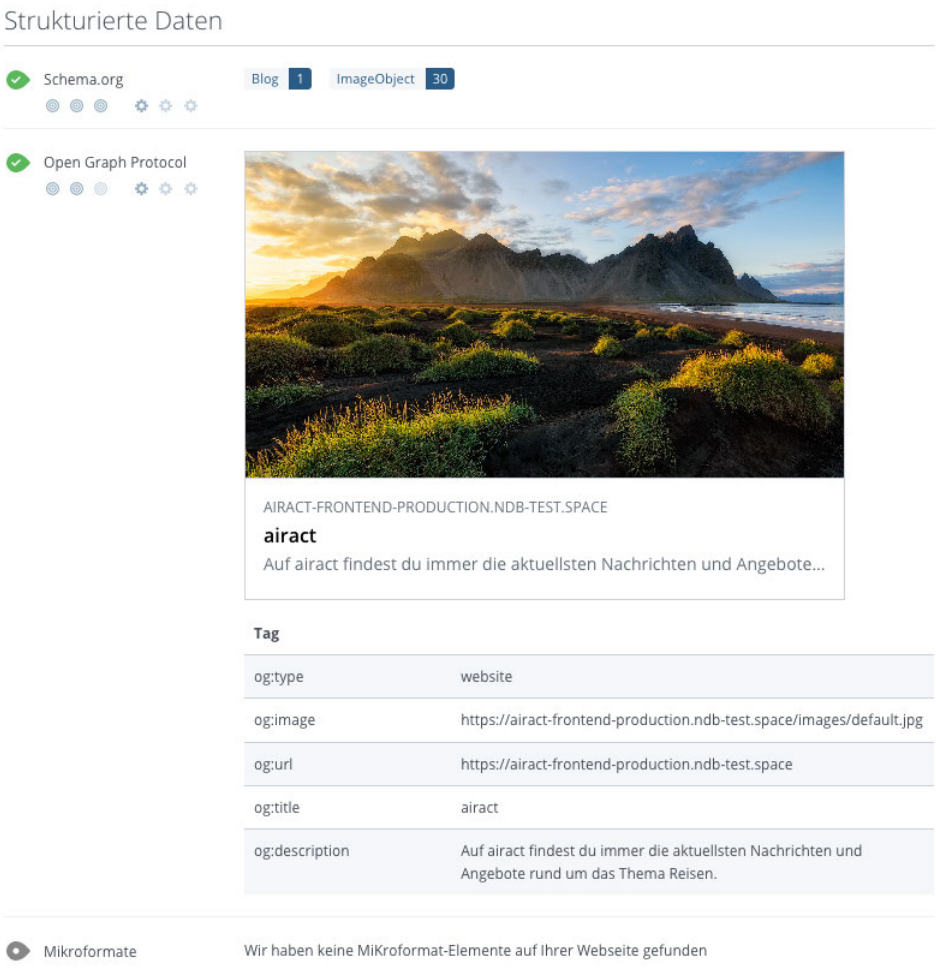

Abbildung A.25: WooRank-Audit: Strukturierte Daten
(Quelle: erstellt mit WooRank)

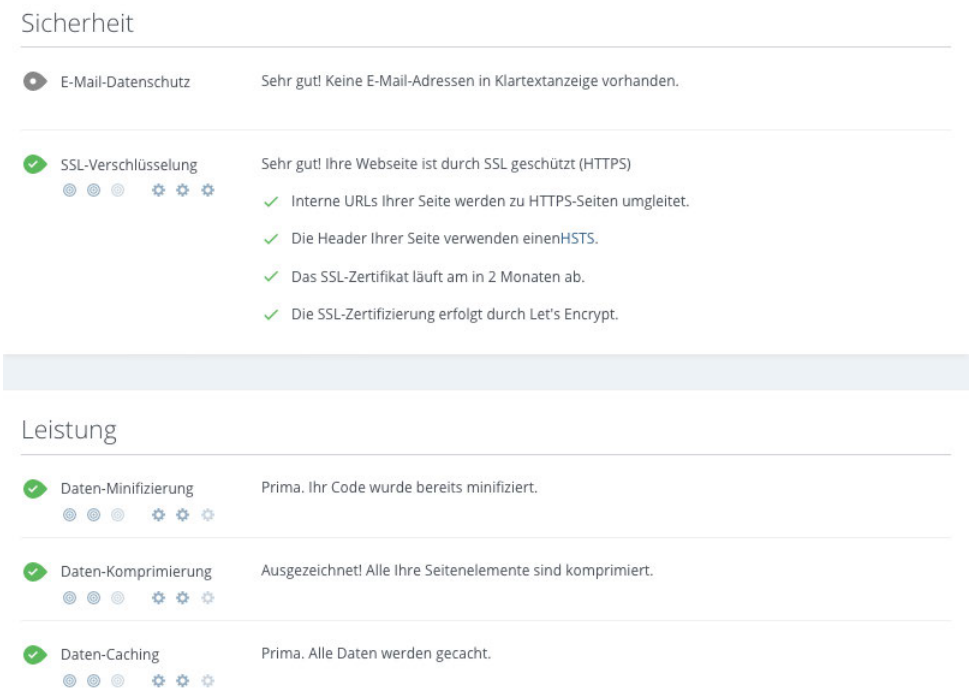

Abbildung A.26: WooRank-Audit: Sicherheit und Leistung (Quelle: erstellt mit WooRank)

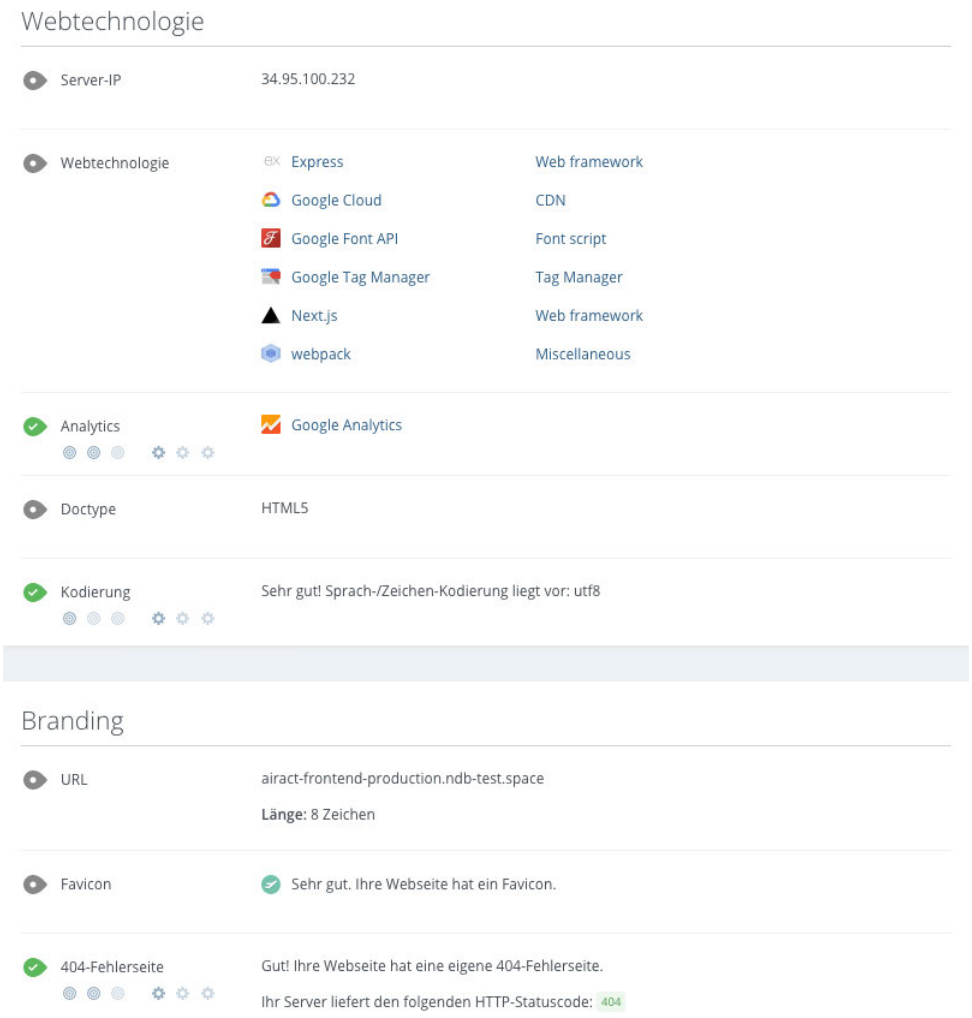

Abbildung A.27: WooRank-Audit: Webtechnologie und Branding (Quelle: erstellt mit WooRank)

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „– bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] – ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Konzeption und prototypische Umsetzung einer Progressive-Web-App unter Verwendung von WordPress als Headless Content-Management-System

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	[redacted]
Ort	Datum	Unterschrift im Original