BACHELORARBEIT

Progressive Web Apps: Eine eingehende Analyse der Kerntechnologien und prototypische Integration in ein modernes Web-Framework

Fakultät Design, Medien und Information

Department Medientechnik

Tom Stehling

Progressive Web Apps: Eine eingehende Analyse der Kerntechnologien und prototypische Integration in ein modernes Web-Framework

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Medientechnik
am Department Medientechnik
der Fakultät Design, Medien und Information
der Hochschule für Angewandte Wissenschaften Hamburg
vorgelegt am 3. Mai 2024
Tom Stehling

Betreuende Prüferin: Prof. Dr. Larissa Putzar
Zweitprüfer: Thorsten Wagener

Zusammenfassung

Diese Arbeit untersucht die Technologie der Progressive Web App (PWA) mit dem Ziel, ihre Funktionsweise und ihre Eignung für mobile Applikationen zu prüfen. Zunächst werden die Kernelemente von PWAs eingehend erklärt und analysiert. Anhand einer minimalen Anwendung werden diese theoretischen Konzepte praktisch umgesetzt und durch die Verwendung von Analysetools validiert. Die Ergebnisse bieten dem Leser ein fundiertes Verständnis für die Funktionsweise der Technologie.

Im nächsten Schritt wird die Implementierung der PWA-Technologie in ein modernes Web-Framework betrachtet, um deren Eignung für die Entwicklung mobiler Anwendungen zu demonstrieren. Die Umsetzung erfolgt anhand eines Prototyps für das fiktive Stadtfest "Waterplane", dessen zentraler Bestandteil ein interaktiver Zeitplan ist, der von einem Content-Management-System verwaltet wird.

Die Evaluation des Prototyps zeigt, dass PWA eine ernstzunehmende Alternative zu herkömmlichen App-Technologien darstellt. Die Vorteile der Technologie, wie zum Beispiel die Indexierung durch Suchmaschinen konnten mit Hilfe von Analysetools verifiziert werden. Die Akzeptanz von Nutzern für die Technologie wurde durch einen Benutzertest bestätigt.

Abstract

This paper examines the technology of Progressive Web App (PWA) with the aim of investigating its functionality and suitability for mobile applications. Initially, the core elements of PWAs are thoroughly explained and analyzed. These theoretical concepts are then practically implemented using a minimal application and

validated through the use of analysis tools. The results provide the reader with a solid understanding of the technology's operation.

Next, the implementation of PWA technology in a modern web framework is considered to demonstrate its suitability for the development of mobile applications. Implementation is done through a prototype for the fictional city festival "Waterplane", with its central component being an interactive schedule managed by a content management system.

The evaluation of the prototype shows that PWA represents a viable alternative to conventional app technologies. The benefits of the technology, such as search engine indexing, was verified using analysis tools. User acceptance of the technology was confirmed through user testing..

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

1.1 Motivation

Das Zeitalter der Digitalisierung ist davon geprägt, die Schnittstelle zwischen Mensch und Maschine stetig zu optimieren. Die nahtlose Integration von Technologie in unseren Alltag wurde insbesondere durch die Verbreitung von Smartphones ermöglicht, die uns die Möglichkeit geben, diese Schnittstelle immer bei uns zu haben. Die Verwendung unterschiedlichster Endgeräte macht die Bereitstellung von Anwendungen jedoch aufwändig. Die Anwendungen, kurz Apps (im Englischen: Applications) sind in ihrer ursprünglichen, „nativen“ Form plattformspezifisch und können daher nur auf dem Betriebssystem ausgeführt werden, für das sie entwickelt wurden.

Aus dieser Problematik heraus sind weitere Entwicklungsansätze entstanden. Browserbasierte Webanwendungen sind plattformunabhängig und müssen nicht vorab installiert werden. Im Vergleich zu nativen Apps sind diese Anwendungen jedoch weniger performant, in ihrem Zugriff auf Gerätefunktionen beschränkt und ihre Funktion ist netzwerkabhängig.

Im Jahr 2015 setzten Alex Russel und Francis Berryman in einem Blogbeitrag den Grundstein für eine neuartige Web Technologie, die sie als Progressive Web App bezeichneten. Progressive Web App (kurz PWA) ist eine erweiterte Web App, die schnell, zuverlässig und unabhängig von der Netzwerkverfügbarkeit funktioniert. Die PWA kann auf dem Gerät installiert werden, offline verfügbar sein und im

Hintergrund Push-Benachrichtigungen empfangen. Sie hat weiterhin die Vorteile einer Web App, ist gleichzeitig jedoch mit Funktionen ausgestattet, die nativen Anwendungen vorenthalten sind.

Progressive Web Apps bieten zusätzliche Möglichkeiten, die in herkömmlichen Apps nicht vorhanden sind. Sie ermöglichen beispielsweise die Indexierung durch Suchmaschinen, sind unabhängig von App Stores und führen Updates automatisch durch. Zudem sind sie plattformübergreifend und können somit von einer Vielzahl von Geräten genutzt werden. Progressive Web Apps haben ein ansprechendes Vollbildlayout, schnelle Ladezeiten, ästhetisches Design und sind offline Verfügbar.

1.2 Zielsetzung

Ziel der Arbeit ist es, die Technologie der Progressive Web App zu untersuchen. Dazu sollen zunächst die Kernelemente eingehend erklärt und deren Funktionsweise analysiert werden. Um die Analyse zu stützen, wird eine minimale Anwendung herangezogen. An deren Beispiel können die zuvor erklärten Funktionen unter Zuhilfenahme von Analysetools praxisnah nachvollzogen werden. Dabei sollen die theoretischen Sachverhalte validiert werden, um dem Leser ein gutes Verständnis dahingehend zu geben, wie die Technologie im Kern funktioniert.

Im nächsten Schritt wird die Technologie in ein modernes Web-Framework implementiert, um ihre Eignung für die Entwicklung mobiler Anwendungen zu demonstrieren. Dabei wird ein Prototyp für das fiktive Stadtfest "Waterplane" erstellt, eine interaktive Zeitplan-App, die von einem Content-Management-System verwaltet wird. Zur Bewertung der Eignung werden zunächst Auditing-Werkzeuge verwendet, um eine objektive Analyse des Prototyps zu ermöglichen. Anschließend wird ein Benutzertest durchgeführt, um die Nutzerwahrnehmung der PWA im Vergleich zu herkömmlichen Apps zu erfassen und zu bewerten. Dieser

Test zielt darauf ab, festzustellen, ob die PWA dieselbe Akzeptanz und Benutzerzufriedenheit erreichen kann wie native Anwendungen. Durch eine umfassende Bewertung der Nutzererfahrung und der Reaktionen auf die Anwendung können potenzielle Stärken und Schwächen der PWA identifiziert werden, was entscheidend ist, um ihre praktische Anwendbarkeit und ihren Erfolg zu bestimmen.

2 Grundlagen

In diesem Abschnitt werden grundlegende theoretische Konzepte vermittelt, welche für nachfolgende Kapitel essenziell sind. Es wird eine Einführung in Content-Management-Systeme (CMS) gegeben, wobei der Unterschied zwischen klassischen CMS und Headless CMS erläutert wird. Anschließend werden verschiedene App Technologien betrachtet und der Begriff der Cross-Platform App differenziert. Zum Abschluss werden die betrachteten App Technologien mit der fortschrittlichen Progressive Web App verglichen.

2.1 Content Management Systeme

CMS sind Softwarepakete, welche die Erstellung von Webseiten vereinfachen. Die Idee, Personen ohne technischen Hintergrund die Möglichkeit zu geben, Inhalte im Web zu veröffentlichen, wurde 2003 mit der Gründung von WordPress Realität (Cabot, 2018). CMS dienen dazu, digitale Inhalte effizient zu erstellen, zu organisieren, zu bearbeiten, zu veröffentlichen und zu verwalten. Traditionelle CMS-Plattformen, wie WordPress, sind integrierte Systeme, bei denen Backend (Content-Management) und Frontend (Content-Präsentation) eng miteinander verbunden sind. Die monolithische Struktur ermöglicht eine einfache Verwaltung und die Vorschau von Inhalten im Kontext der Endnutzeransicht, und kann deshalb leicht ohne tiefergehendes technisches Verständnis, verwendet werden. Der Vorteil der Benutzerfreundlichkeit geht dabei zu Lasten der Flexibilität, da Backend und Frontend fest miteinander verbunden sind (Kitsios, 2015).

Headless CMS hingegen entkoppelt das Content-Management von der Content-Auslieferung, indem es nur das Backend bereitstellt und Application Programming Interfaces (APIs) für die Bereitstellung von Inhalten an verschiedene Frontend-Systeme wie Websites oder Mobile Apps verwendet (Cabot, 2018).

2.2 Relevante App-Technologien

Die Welt der mobilen Anwendungen hat in den letzten Jahren einen beispiellosen Aufschwung erlebt (vgl. Abbildung Anhang 1). Im Jahr 2023 wurden in Deutschland über zwei Milliarden Apps heruntergeladen (Bitkom e.V., 2023). Hinter dieser Erfolgsgeschichte stehen verschiedene App-Technologien, die es ermöglichen, die kleinen Programme auf Smartphones, Tablets, Armbanduhren und Spielkonsolen zum Leben zu erwecken. In diesem Kapitel werden die wichtigsten dieser Technologien erläutert und Vor- und Nachteile verschiedener Ansätze verglichen.

2.2.1 Web Apps

Bei der Einführung des iPhones im Jahr 2007 standen Entwicklern ausschließlich Web Apps zur Verfügung, um Anwendungen für das Gerät zu erstellen.

„The full Safari engine is inside of iPhone. And so, you can write amazing Web 2.0 and Ajax apps that look exactly and behave exactly like apps on the iPhone.“ - Steve Jobs (Steve Jobs at MacWorld 2007)

Web Apps bilden einen zentralen Bestandteil der digitalen Landschaft und bieten Nutzern die Möglichkeit, über einen Webbrowser auf unterschiedlichste Anwendungen zuzugreifen. Im Gegensatz zu nativen Apps, die speziell für bestimmte Plattformen entwickelt werden, sind Web Apps plattformunabhängig und können

auf verschiedenen Geräten mit Internetzugang genutzt werden (Raj & Tolety, 2012). Die Entwicklung von Web Apps erfolgt in der Regel unter Verwendung von Webtechnologien wie HTML, CSS und JavaScript, wodurch sie leicht zugänglich und ohne vorherige Installation nutzbar sind (Arinir, 2023, p. 22f.). Durch ihre Flexibilität und Verfügbarkeit sind Web Apps zu einer essenziellen Komponente im Bereich der Anwendungsentwicklung geworden, die eine breite Palette von Funktionen und Dienstleistungen abdeckt (Raj & Tolety, 2012).

2.2.2 Native Apps

Native Apps sind speziell für ein bestimmtes Betriebssystem, wie zum Beispiel iOS oder Android optimiert und vollständig in der nativen Programmiersprache der Plattform geschrieben. Dadurch sind sie in der Lage, direkt auf native APIs zuzugreifen und so alle Funktionen und Ressourcen des jeweiligen Betriebssystems effizient zu nutzen, was zu einer hohen Leistung und nahtlosen Integration in die Geräteumgebung führt (Rohan Anand Choudhary, 2020).

Native Apps werden durch die jeweiligen App-Stores vertrieben. Plattformspezifische App-Stores wie der Apple App Store oder Google Play bieten eine breite Reichweite und ermöglichen es, Anwendungen einem globalen Publikum zugänglich zu machen (Nunkesser, 2023, p. 274f.). Aktualisierungen werden ebenfalls über den jeweiligen App Store abgewickelt, indem Benutzer automatisch oder manuell die neuesten Versionen herunterladen und installieren können (Rohan Anand Choudhary, 2020).

Der wesentliche Nachteil liegt in der Notwendigkeit, separate Codebasen für verschiedene Plattformen zu pflegen, was zu höheren Entwicklungskosten und längeren Entwicklungszeiten führen kann. Außerdem besteht eine Abhängigkeit zu den App Store Regulierungen. So verlangt Apple beispielsweise zwischen 15% und

30% Kommission für *digital goods and services*, welche dann beim Kauf einer App oder auch bei in-App-Käufen anfällt (Baggott, 2023).

2.2.3 Hybride Apps

Entwickler begannen schon früh, verschiedene Ansätze für die Entwicklung mobiler Anwendungen zu kombinieren und bezeichneten diese als "Hybride Apps". Sie werden mit Webtechnologien entwickelt und mit Hilfe von Frameworks wie Cordova[1] in eine native Hülle gepackt (Arinir, 2023, p. 25f.). Hybride Apps stellen eine Art Zwischenform von nativen Apps und Web Apps dar (Que, 2016).

Im Spektrum der Hybriden Apps haben sich über die Jahre verschiedene Ansätze entwickelt, welche oftmals fälschlicherweise nicht differenziert oder falsch erklärt werden (Nunkesser, 2023, p. 69). Im Folgenden soll deshalb knapp auf die Unterschiede eingegangen werden.

Im Ursprung ist die Hybrid-App im Wesentlichen eine kleine Webseite, die in Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) und JavaScript geschrieben ist (Arinir, 2023, p. 26). Sie unterscheidet sich von normalen Websites darin, dass sie nur in einer Browser-Shell läuft und Zugriff auf die native Plattformebene hat. Um wie eine native App zu funktionieren, ist sie auf einen nativen Wrapper, wie Cordova, angewiesen (Arinir, 2023, p. 25f). In Abbildung 1 ist dieser Wrapper als “Bridge” zwischen den Nativen APIs (rechts) und den Webtechnologien (links) zu erkennen. Daher wird die Hybrid-App wie eine Website entworfen und programmiert, jedoch anschließend mit den Fähigkeiten ausgestattet, auf native Plattformfunktionalitäten zuzugreifen (Prashant Ghimire, 2017).

[1] https://cordova.apache.org/

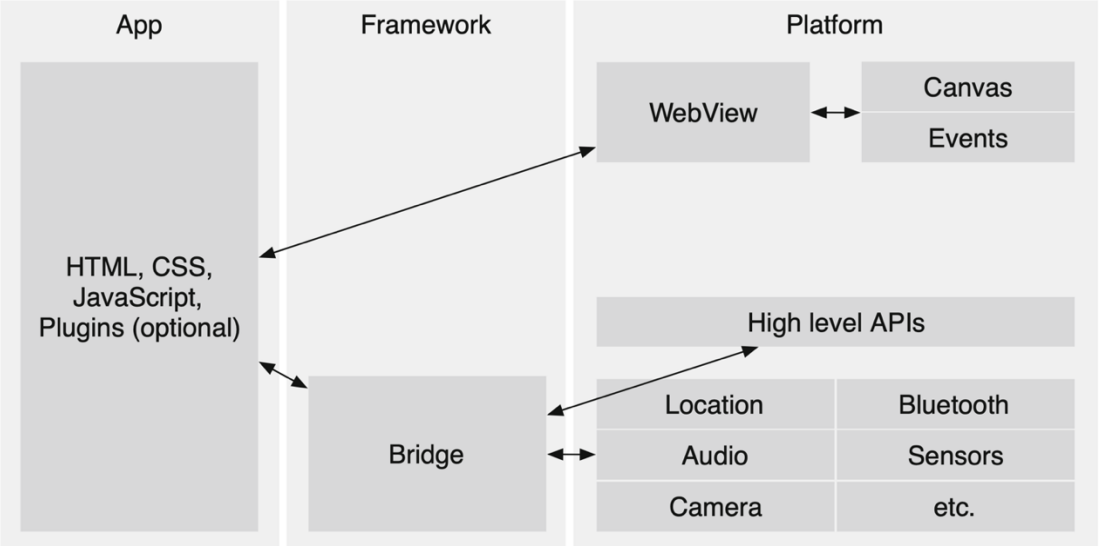


Abbildung 1: Aufbau von Hybrid Web Apps
Quelle: R. Nunkesser, 2023, S. 73

Cross-Platform-Apps repräsentieren die moderne Version der Hybriden App und sind eine wegweisende Technologie in der mobilen App-Entwicklung, die es ermöglicht, Anwendungen auf verschiedenen Betriebssystemen und Plattformen mit einer gemeinsamen Codebasis auszuführen. Im Gegensatz zu nativen Apps, die spezifisch für ein bestimmtes Betriebssystem entwickelt werden, und ursprünglichen hybriden Apps, die eine Kombination aus webbasiertem und nativem Ansatz darstellen, verfolgen Cross-Platform-Apps einen einheitlichen Code-Ansatz, der auf verschiedenen Plattformen wiederverwendet werden kann (Arinir, 2023, p. 25f). Die Verwendung von Cross-Platform-Technologien, wie beispielsweise React Native, Flutter oder Xamarin, ermöglicht es Entwicklern, die Entwicklungszeit erheblich zu reduzieren und damit die Kosten, im Vergleich zur Erstellung separater nativer Apps für jede Platform, zu senken. Allerdings geht dies auf Kosten der Performanz, was besonders bei höheren Grafikleistungen spürbar ist (Ebone, 2018).

Cross Platform ist ein Überbegriff, welcher verschiedene Framework-Strukturen umfasst. Hier kann zwischen Hybrid Bridged, System Language und Foreign Language Ansätzen unterschieden werden (Nunkesser, 2023, p. 74f.). Hybrid Bridged Apps nutzen neben einer Bridge zu den nativen APIs auch eine Bridge zu nativen UI-Elementen (vgl. Abbildung 2). Dies hat zur Folge, dass das User Interface, anders als bei zuvor beschriebenen Hybrid Web Apps, nativ, und keine Web-View mehr ist. Daraus ergibt sich eine native User Experience, welche direkter und flüssiger ist. Ein Beispielframework ist React Native, welches von Facebook entwickelt wurde, und nach Stack Exchange Daten das am weitesten verbreitete Cross Platform Framework ist (Abbildung Anhang 10).

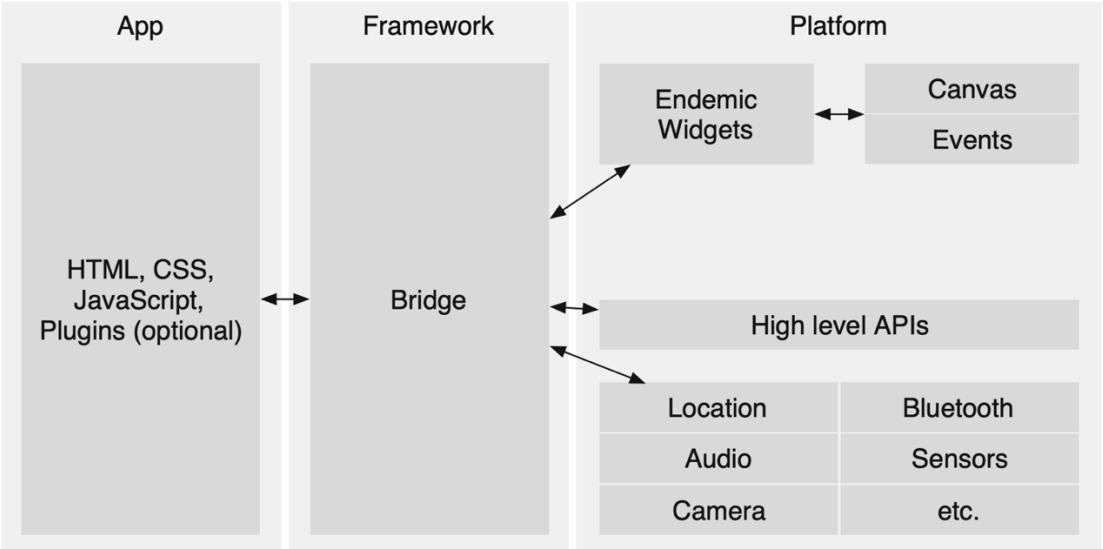


Abbildung 2: Hybrid Bridged App
Quelle: R. Nunkesser, 2023, S.74

Bei System Language Apps ist die UI gänzlich vom System abgekoppelt. Dieser Ansatz kommt vor allem bei Spiele Engines wie Unity zur Anwendung, ist für klassische App Entwicklung jedoch weniger bedeutend und wird deshalb in diesem Kontext nicht näher betrachtet.

Foreign Language Apps nutzen plattformfremde Sprachen, welche durch verschiedene Methoden an die jeweilige Plattform angekoppelt werden können. Ein bekanntes Beispiel ist Flutter, welches die Entwicklung von Apps in der Programmiersprache Dart ermöglicht. Um native APIs zu erreichen, stellt das Framework sogenannte Platform Channels bereit. Für das User Interface werden hier zwar auch keine nativen UI-Elemente verwendet, die Umsetzung ist jedoch weitaus ausgefeilter als bei herkömmlichen Hybriden Apps, bei welchen die User Interface (UI) Elemente auf Webtechnologien beruhen (Nunkesser, 2023, p. 75f.).

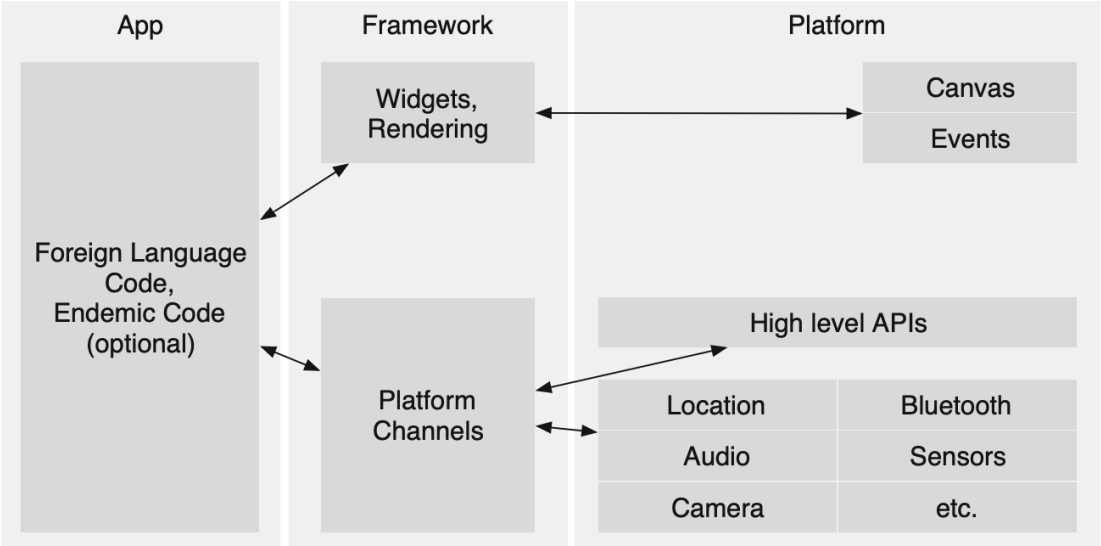


Abbildung 3 Foreign Language App
Quelle: R. Nunkesser, 2023, S.77

Cross Platform Apps sind auf verschiedenste Technologien aufgebaut, haben jedoch alle gemeinsam, dass es nur eine Code Basis benötigt, um die App auf allen Plattformen zur Verfügung zu stellen (Rohan Anand Choudhary, 2020). Trotz ausgefeilter Ansätze gibt es weiterhin Herausforderungen bei der Entwicklung von

Cross Platform Apps. Die Performanz kann im Vergleich zu nativen Apps beeinträchtigt sein, insbesondere bei ressourcenintensiven Anwendungen. Zudem müssen Entwickler sicherstellen, dass ihre Apps auf den verschiedenen Plattformen gleichermaßen funktionieren und die spezifischen Designrichtlinien und Standards erfüllen. Diese werden insbesondere bei Apple äußerst restriktiv gehandhabt (Apple Inc, 2023).

2.2.4 Progressive Web Apps

Obgleich die Konzeption der Web App bereits im Jahre 2007 als wegweisende Innovation präsentiert wurde (Nunkesser, 2023, p. 69), bedurfte es einer Zeitspanne von acht Jahren, bis im Jahr 2015 der Google-Mitarbeiter Alex Russell die Progressive Web App einführte und somit der Technologie neuen Aufschwung verlieh. Russel spricht von einem Hybridisierungsprozess, bei dem Web- und App-Erlebnisse verschmelzen, und identifiziert diesen als Herausforderung, die nicht mit den grundlegenden Prinzipien des Webs in Einklang zu bringen ist (Russel, 2015).

If you can't link to it, it isn't part of the web - Alex Russel (Russel, 2015)

Progressive Web App wird, anders als zuvor beschriebene App-Arten, nicht über die zugrunde liegenden technischen Aspekte definiert, sondern ist vielmehr ein Set an Eigenschaften, welche die Webapp haben muss, um eine PWA zu sein. Russel definiert das Set an Eigenschaften, welche die Grundbausteine für die Progressive Web App, kurz PWA sind, folgendermaßen (Russel, 2015):

- **Responsiv:** Eine PWA ist auf verschiedenen Endgeräten gleichermaßen übersichtlich und benutzerfreundlich.

- **Progressiv**: Eine PWA basiert auf dem Prinzip der progressiven Verbesserung und ist daher unabhängig vom Browser des Benutzers.

- **Netzwerkunabhängig**: Eine PWA ist auch bei schlechter Netzwerkverbindung oder offline Szenarien funktionsfähig.

- **App-ähnlich**: Navigation und Interaktion der PWA orientieren sich an nativen Apps.

- **Immer aktuell**: Eine PWA ist immer auf der neuesten Version.

- **Sicher**: Eine PWA wird über TLS/HTTPS bereitgestellt.

- **Auffindbar**: Eine PWA kann durch Suchmaschinen indexiert werden.

- **Bindend**: Eine PWA stellen nutzerbindende Möglichkeiten zur Verfügung, wie beispielsweise Push-Notifications

- **Installierbar**: Eine PWA kann auf dem Startbildschirm installiert werden, sodass Benutzer Apps schnell wiederfinden können.

- **Verlinkbar**: Eine PWA ist über eine URL verlinkbar.

2.2.5 Vergleich mit klassischen Apps

Progressive Web Apps (PWAs) markieren eine bedeutende Weiterentwicklung im Bereich der Anwendungsentwicklung, indem sie die Vorteile von Web Apps mit erweiterten Funktionalitäten verbinden, die traditionell mit nativen Apps in Verbindung gebracht werden (Steiner, 2018). PWAs zeichnen sich insbesondere durch ihre Plattformunabhängigkeit aus, indem sie alle Plattformen mit einem installierten Browser unterstützen und einen reibungslosen Übergang für bestehende

Websites ermöglichen. PWAs können außerhalb von App-Stores vertrieben werden. Trotz ihrer beschränkten Zugriffsmöglichkeiten auf Gerätefunktionen und potenziellen Leistungskompromissen, insbesondere im Vergleich zu nativen Apps, stellen PWAs eine überzeugende Lösung für bestimmte Anwendungen dar (Sayali Sunil Tandel, 2018). In Tabelle 1 ist ein Vergleich der App Technologien zu sehen.

Tabelle 1: Vergleich unterschiedlicher Arten von Apps
Quelle: Eigene Darstellung

	Native App	Hybrid App	Cross-Platform App	PWA
Entwicklungskosten	Höher	Niedriger	Niedriger	Niedriger
Entwicklungsdauer	Länger	Kürzer	Kürzer	Kürzer
Performanz	Höchste	Mittel	Höher	Mittel
Plattformunabhängigkeit	Nein	Ja	Ja	Ja
Zugriff auf Gerätefunktionen	Volle Funktionalität	Eingeschränkt durch Plugins	Eingeschränkt durch Frameworks	Begrenzter Zugriff, aber wachsend
Benutzererfahrung	Nativ aussehend und reagierend	Könnte variieren	Könnte variieren	App-ähnliche Erfahrung
Offline-Verfügbarkeit	Ja	Ja	Ja	Ja
App-Store-Präsenz	Ja, plattformspezifisch	Ja	Ja	Optional
Aktualisierungen	Durch App-Store-Updates	Durch App-Store-Updates	Durch App-Store-Updates	Automatische Aktualisierungen dank Service Worker

3 Analyse der Kerntechnologien

Ziel dieses Kapitels ist es, die Technologie hinter der Progressive Web App zu untersuchen. Dazu werden die Kernelemente von PWA zunächst vorgestellt und theoretisch erklärt. Um die theoretischen Erklärungen zu untermauern, soll außerdem die Funktionsweise eingehend analysiert werden. Um die Analyse durchzuführen, wird eine Web App mit minimalem Funktionsumfang aufgesetzt, an deren Beispiel die zuvor erklärten Funktionen, unter Zuhilfenahme von Analysetools, verifiziert werden können. Dabei sollen die theoretischen Sachverhalte praxisnah nachvollzogen werden, um dem Leser ein gutes Verständnis dahingehend zu geben, wie die Technologie im Kern funktioniert. Dieses Verständnis bildet die Grundlage für nachfolgende Kapitel, ab Abschnitt 4 (Anforderungsanalyse und Spezifikation), wenn die Technologie in ein modernes Web Framework implementiert wird.

Hinweis

Um die in diesem Kapitel vermittelten Inhalte noch besser nachvollziehen zu können, hat der Leser/die Leserin die Möglichkeit, mit Hilfe der unten genannten Beispielapplikation auf GitHub die einzelnen Schritte aktiv und selbstständig durchzuführen. --> https://github.com/tomstehling/BasicPWA

3.1 Einführung

Progressive Web Apps stellen eine Weiterentwicklung herkömmlicher Webanwendungen dar, indem sie zusätzliche Funktionen und Eigenschaften bieten, die eine verbesserte Benutzererfahrung ermöglichen. Im Wesentlichen sind PWAs jedoch immer noch Webanwendungen, die über herkömmliche Webtechnologien wie HTML, CSS und JavaScript entwickelt werden. Um eine Webanwendung in eine PWA zu verwandeln, sind bestimmte Technologien und Konzepte erforderlich. Dazu gehören das App Manifest, das grundlegende Informationen wie den Anwendungsnamen und die Symbole bereitstellt, Service Worker, die eine Offline-Funktionalität und eine verbesserte Leistung ermöglichen, sowie HTTPS, um die Sicherheit der Datenübertragung zu gewährleisten. In diesem Kapitel sollen die Kerntechnologien näher betrachtet und insbesondere die Verhaltensweise des Service Worker praxisnah erklärt werden. Dazu wird die App „BasicPWA“ (vgl. Abbildung Anhang 2) herangezogen, welche aus einem einzelnen index.html File (vgl. Abbildung Anhang 3) besteht. Dieses stellt die Überschrift „BasicPWA“ dar und registriert einen Service Worker. Im folgenden Abschnitt soll zunächst das App Manifest erklärt und der App hinzugefügt werden.

3.2 Das App Manifest

Um eine App auf dem Homescreen installierbar zu machen, benötigt das Zielbetriebssystem einige appspezifische Informationen, wie beispielsweise das Icon der App, welches später auf dem Homescreen zu sehen ist, oder auch den Namen, welcher unterhalb des Icons angezeigt wird. Diese appspezifischen Informationen werden durch eine JSON-Datei bereitgestellt, welche im Head der Hypertext Markup Language (HTML) Datei jeder Seite der App verlinkt wird (vgl. Code Listing 1)

```
<head>
  <link rel="manifest" href="manifest.json" />
</head>
```

Code Listing 1: Verlinkung des app manifest im html
Quelle: Eigene Darstellung

Das App Manifest besteht aus einem JSON Objekt, dessen Top-Level Schlüssel als Members bezeichnet werden. Für Progressive Web Apps sind die minimalen Anforderungen folgende Members:

Tabelle 2: Minimal Anforderungen des App Manifest
Quelle: Eigene Darstellung

Bezeichnung	Typ	Beschreibung
name	string	Repräsentiert den Namen der Webapp und ist beispielsweise für das Label unter dem Homescreen Icon relevant.
icons	array	Enthält Objekte, welche Bilddateien repräsentieren. Icons werden in verschiedenen Größen bereitgestellt, um verschieden dimensionierte Endgeräte zu unterstützen.
start_url	string	Repräsentiert die start Uniform Resource Locator (URL) der Webapp, welche geladen werden soll, wenn die App im standalone Modus gestartet wird.
display und/oder display_override	string	Gibt den Display Modus des Browsers an. In den meisten Fällen wird hier *fullscreen* oder *standalone* angegeben, um der PWA einen nativen Look ohne Navigationselemente des Browsers zu geben. Das display member gibt das höchste Glied einer Fallback Kette an, an der bei Kompatibilitätsproblemen heruntergegangen wird. Die Reihenfolge der Fallbacks kann mit dem display_override member spezifiziert werden.

Um die JSON-Datei zu erstellen, kann entweder einer der vielen online Generatoren[2] verwendet werden, das App Manifest kann jedoch auch einfach von Hand geschrieben werden. Der größte Vorteil der online Generatoren ist dabei die Erstellung verschiedengroßer Icons, die man sonst entweder manuell oder auch mit einer entsprechenden Helfersoftware, wie beispielsweise das *npm package pwa-asset-generator*[3] erstellen würde. Im App Manifest werden diese dann referenziert (vgl. Abbildung Anhang 4).

Begriffserklärung: Client

Während der Begriff Client im Kontext von client-server Architektur das Endgerät oder die Software (Webbrowser), die Anfragen an den Server schickt, beschreibt, so steht der Begriff Client im Kontext von Service Worker für die Instanz der Webpage, die vom Service Worker kontrolliert wird.

[2] Bsp. https://app-manifest.firebaseapp.com/
[3] https://www.npmjs.com/package/pwa-asset-generator

3.3 Der Service Worker Lifecycle

Der Service Worker Lifecycle durchläuft drei Zustände (vgl. Tabelle 3: Zustände des Service Worker).

Tabelle 3: Zustände des Service Worker
Quelle: Eigene Darstellung

Zustand (Nr)	Zustand (Name)	Beschreibung
1.	download	Registriert die Webpage einen *service worker,* so wird dieser sofort heruntergeladen.
2.	install	Ist der *service worker* heruntergeladen, versucht der Browser diesen zu installieren
3.	activate	Ist der service *worker* installiert, kann dieser aktiviert werden. Sobald der service worker aktiviert ist, hat dieser die Kontrolle über den Client.

Um die Vorgänge der App nachvollziehen zu können wir das Analysetool Chrome DevTools[4] verwendet. Chrome DevTools ist ein Set von Entwicklertools, welche direkt in den Chrome Browser integriert sind.

In Chrome DevTools kann der Service Worker Lifecycle nachvollzogen werden. Zunächst wird der Download des Service Worker bei Ausführung des index File, hier als localhost zu erkennen, getriggert. Daraufhin wird der Service Worker, hier sw.js, angefragt und heruntergeladen (vgl. Abbildung 4).

[4] https://developer.chrome.com/docs/devtools

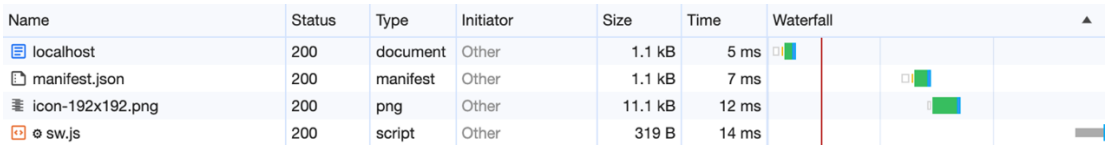


Abbildung 4: Netzwerk
Quelle: Screenshot Chrome DevTools

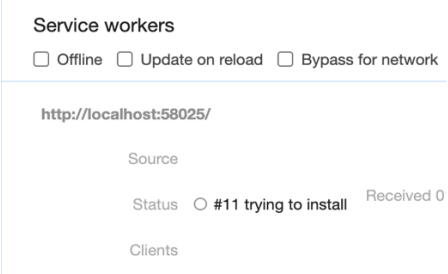


Abbildung 5: Service Worker Status
Quelle: Screenshot Chrome DevTools

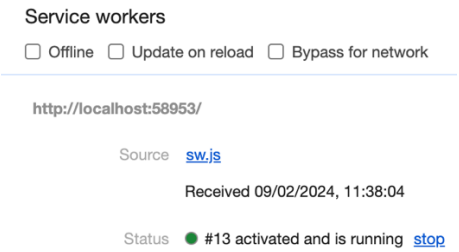


Abbildung 6: Service Worker activated
Quelle: Screenshot Chrome DevTools

Im Application Tab der DevTools kann beobachtet werden, dass der Browser versucht, den Service Worker zu installieren (Abbildung 5). War die Installation erfolgreich, aktiviert sich der Service Worker automatisch und übernimmt die Kontrolle über den Client (Abbildung 6).

Ist der Service Worker installiert, kann dieser auf install, activate und fetch Events reagieren. Loggt man eine Nachricht zur Bestätigung des install und activate Events, wird die asynchrone Funktionsweise des Service Worker deutlich. Die chronologisch aufgeführten logs zeigen, dass die Seite bereits vollständig geladen hat, der Service Worker jedoch asynchron im Hintergrund arbeitet, auf die Events wartet und entsprechende logs erstellt (vgl. Abbildung 7).

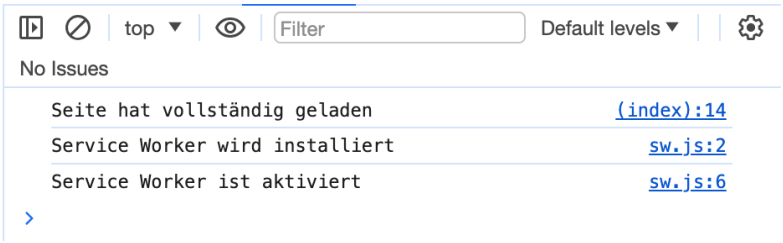


Abbildung 7: Konsole nach erstmaligem Aufruf der App
Quelle: Screenshot Chrome DevTools

Logs der App sind als „index“, die des Service Worker als „sw.js“ zu erkennen.

Wird die App neu geladen, bleiben die Nachrichten des Service Worker aus. Hier wird deutlich, dass der Lifecycle des Service Worker unabhängig von der Webpage ist. Die entsprechenden logs des Service Worker bleiben aus, da dieser weiterhin im Hintergrund aktiv bleibt, selbst wenn der Client geschlossen wird.

Steht ein neuer Service Worker zur Verfügung, so lädt der Browser diesen sofort herunter. Der Service Worker aktiviert sich jedoch standardmäßig nicht sofort, sondern wartet, bis alle Clients, welche vom bisherigen Service Worker kontrolliert werden, geschlossen wurden (vgl. Abbildung 8). Schließt man den Tab und navigiert daraufhin erneut zur Webapp, so ist der neue Service Worker nun aktiv (vgl. Abbildung 9). Um die Service Worker besser unterscheiden zu können, zählt Chrome, wie oft ein Service Worker update durchgeführt wurde. In Abbildung 9 ist gerade Version #41 aktiv.

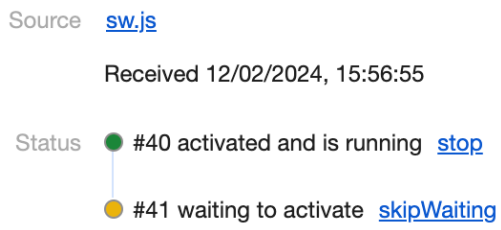


Abbildung 8: Service Worker Status (neuer Service Worker verfügbar) Quelle: Screenshot Chrome DevTools

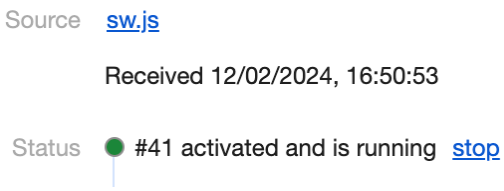


Abbildung 9: Service Worker Status (neuer Service Worker aktiv) Quelle: Screenshot ChromeDev Tools

3.4 Cache Storage API

Eindes der größten Probleme des Web ist die Notwendigkeit einer dauerhaften Internetverbindung. Egal wie ausgefeilt eine Webapp sein mag, bei schlechter Internetverbindung ist die User Experience unbefriedigend. Service Worker können dieses Problem beheben, indem sie benötigte Daten im Cache speichern, und diese auch in offline Szenarien bereitstellen können (Mozilla Foundation, 2023).

Um Inhalte speichern zu können stehen dem Service Worker verschiedene Möglichkeiten zur Verfügung. Ressourcen, um Netzwerkanfragen bedienen zu können, wie beispielsweise HTML-Seiten, Stylesheets oder Bilder, können mit Hilfe der Cache Storage API gespeichert werden. Das CacheStorage Interface erlaubt es einen Cache Speicher zu verwalten, welcher mehrere Request/Response Paare enthalten kann. Das Interface verfügt darüber hinaus über einen matching Algorithmus, um gecachte Einträge effizient finden zu können. Dies erlaubt es dem Service Worker, Ressourcen abzuspeichern, und diese dem Client zu einem späteren Zeitpunkt bereitstellen zu können. Die Cache Storage API steht nicht nur Service Workern zur Verfügung, sondern kann auch aus einem Window Kontext erreicht werden. Als Beispiel wird ein neuer Cache mit Namen *cacheStorageTest* angelegt.

Dort gespeichert wird ein Request/Response Paar, wobei der fiktive Pfad *RequestURL* als Schlüssel, und die Payload *Dieser String ist im Cache gespeichert!* verwendet wird

Im Application Tab, unter Cache Storage findet man den eben angelegten Cache *cacheStorageTest,* welcher einen Eintrag, *RequestURL* enthält (vgl. Abbildung 10).

Abbildung 10: Cache Storage Panel in ChromeDevTools
Quelle: Screenshot Chrome DevTools

Das gespeicherte Response Objekt. Dieses enthält Metadaten, welche im linken *Headers* Tab angesehen werden können. Wählt man den *Preview* Tab so findet man den gespeicherten String (vgl. Abbildung 11).

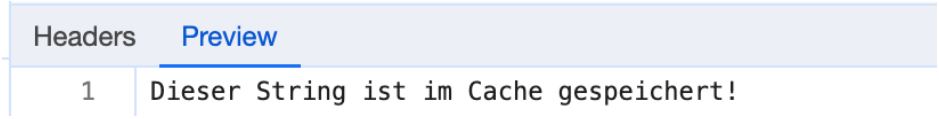


Abbildung 11: Cache Storage: Preview des Strings
Quelle: Screenshot Chrome DevTools

3.5 Offline Fallback

Doch die Speicherung allein hilft der Userexperience nicht weiter. Im Nachfolgenden soll deshalb der Service Worker den zuvor gespeicherten Eintrag als offline Fallback verwenden. Hierzu muss der Service Worker Fetch Events abfangen und je nach Netzwerkverfügbarkeit die zugehörige Response oder den offline Fallback bereitstellen. Mit Hilfe der DevTools können verschiedene Netzwerkgeschwindigkeiten simuliert werden. Wählt man zunächst „offline "und lädt die Seite zum ersten Mal, erscheint, wie zu erwarten Chrome 404. Wird die Internetverbindung aktiviert und die Seite erneut geladen, wird der Service Worker aktiv und das

offline Fallback gecacht.. Sowohl der aktive Service Worker, als auch das gecachte offline Fallback kann im Application Tab nachvollzogen werden (vgl. Abbildung 12).

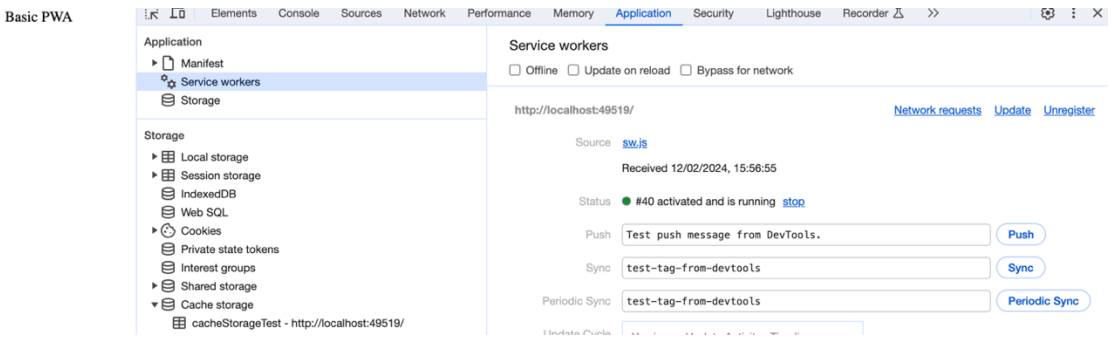


Abbildung 12: Links: Basic PWA page, rechts: aktive Service Worker
Quelle: Screenshot Chrome DevTools

Setzt man die Netzwerkeinstellung wieder auf „offline“ und lädt die Seite erneut, so kann die Arbeit des Service Worker im Netzwerk beobachtet werden: Der Client fragt das index.html an, der Service Worker fängt die Anfrage ab und stellt daraufhin sofort selbst genau dieselbe Anfrage. In Abbildung 13 sind beide Anfragen zu finden. Unten in rot ist die fehlgeschlagene Anfrage des Service Worker zu sehen. Dies ist zu erwarten, da keine Netzwerkverbindung besteht. Die ursprüngliche Anfrage ist jedoch mit Status 200 erfolgreich. In Abbildung 13 kann im *Size* Tab erkannt werden, dass die Response nicht aus dem Netzwerk kommt, sondern vom Service Worker.

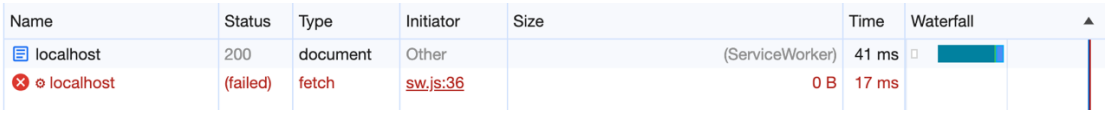


Abbildung 13: Netzwerk
Quelle: Screenshot Chrome DevTools

Der Client bekommt vom Service Worker nun das offline Fallback zugespielt, ohne von der fehlgeschlagenen Anfrage beeinträchtigt zu sein (vgl. Abbildung 14). Die Möglichkeit, Usern bei fehlender Netzwerkverbindung maßgeschneiderte Optionen bieten zu können, kann die User Experience signifikant verbessern.

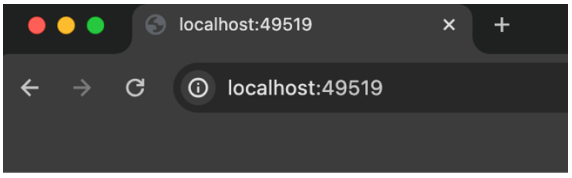


Abbildung 14: Basic PWA
Quelle: Screenshot Chrome DevTools

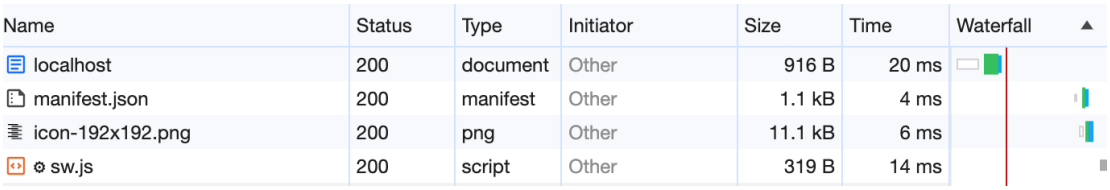
3.6 Runtime Caching

Die Möglichkeit Request/Response Paare speichern zu können, ermöglichen es dem Service Worker nicht, nur offline Fallbacks, sondern auch die angefragten Inhalte offline zur Verfügung stellen zu können. Je nach Use Case kann der Service Worker dabei auf unterschiedliche Art und Weise vorgehen, wobei entweder das Netzwerk oder der Cache priorisiert wird. Eine Strategie, bei der zunächst am Netzwerk angefragt wird, und nur bei Fehlschlag im Cache wird als *Network first*[5] Strategie bezeichnet. Wird der Cache priorisiert, handelt es sich um eine Cache first Strategie. Sollen dynamische Inhalte gecacht werden, so ist eine Cache first Strategie unbrauchbar, da der Service Worker immer auf die älteste Version zurückgreift und diese nicht aktualisiert. Bei einer Network first Strategie hingegen, wird immer die aktuelle Version angezeigt, es sei denn, der Client ist offline.

[5] https://developer.chrome.com/docs/workbox/caching-strategies-overview

Um den Mechanismus noch besser zu greifen, soll dieser am Beispiel einer Network first Strategie untersucht werden.

In Abbildung 15 sind zunächst die Netzwerkanfragen beim ersten Laden der Seite zu sehen. Der Browser schickt Anfragen für alle notwendigen Ressourcen und lädt diese erfolgreich (Status 200).



Name	Status	Type	Initiator	Size	Time	Waterfall
localhost	200	document	Other	916 B	20 ms	
manifest.json	200	manifest	Other	1.1 kB	4 ms	
icon-192x192.png	200	png	Other	11.1 kB	6 ms	
sw.js	200	script	Other	319 B	14 ms	

Abbildung 15: Netzwerk bei erstem Laden
Quelle: Screenshot Chrome DevTools

Der Service Worker wird heruntergeladen und sofort aktiviert. Betrachtet man den Storage (vgl. Abbildung 16), so ist hier zunächst erwartungsgemäß nur der Service Worker (853Byte) aufgeführt.

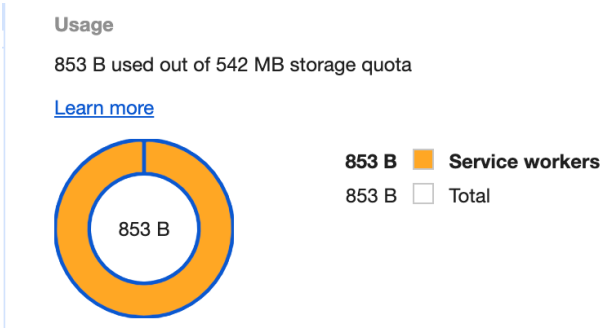


Abbildung 16: Storage von Basic PWA nach erstem Laden
Quelle: Screenshot Chrome DevTools

Wird die Seite neu geladen, sind neben den erwarteten Requests, die auch schon beim ersten Laden zu sehen war, weitere hinzugekommen (vgl. Abbildung 17). Der *initiator* Tab zeigt, dass nicht nur der Client Anfragen geschickt hat, sondern dass nun der Service Worker selbst Ressourcen anfragen kann.

Name	Status	Type	Initiator	Size	Time	Waterfall
localhost	200	document	Other	(ServiceWorker)	68 …	
⚙ localhost	200	fetch	sw.js:15	113 B	28 …	
manifest.json	200	manifest	Other	(ServiceWorker)	7 ms	
{:} ⚙ manifest.json	200	fetch	sw.js:15	113 B	3 ms	
icon-192x192.png	200	png	Other	(ServiceWorker)	8 ms	
⚙ icon-192x192....	200	fetch	sw.js:15	113 B	5 ms	

Abbildung 17: Netzwerk bei erneutem Laden
Quelle: Screenshot Chrome DevTools

Betrachtet man den *size* tab, fällt auf, dass der Client die angefragten Ressourcen vom Service Worker geliefert bekommt. Der Ablauf der Caching Strategie ist wie folgt: Der Client fragt eine Ressource an, der Service Worker fängt die Anfrage ab, fragt selbst die entsprechenden Ressourcen an und serviert diese dann dem Client. Zuvor speichert der Service Worker noch die Response im Cache. Eine visuelle Abbildung der Reihenfolge ist im Anhang zu finden (vgl. Abbildung Anhang 6).

Im Application Tab (vgl. Abbildung 18) ist zu sehen, dass der Service Worker einen Cache Storage angelegt hat, in dem die Anfragen (14,4 kB) abgespeichert sind.

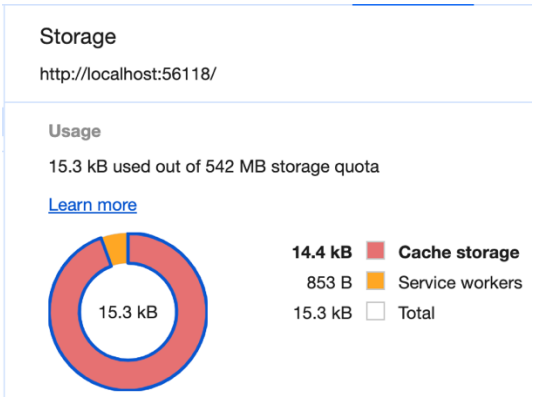


Abbildung 18: Storage nach erneutem Laden
Quelle: Screenshot Chrome DevTools

Lenkt man seinen Blick auf den Cache Storage (vgl. Abbildung 19), kann der angelegte Cache näher betrachtet werden. Hier findet man einen Eintrag, „testCache“, welcher vom Service Worker angelegt wurde. Der Cache enthält Key-Value Paare, wobei als Schlüssel genau die drei Request Pfade, welche von der App beim Laden angefragt werden, zu finden sind.

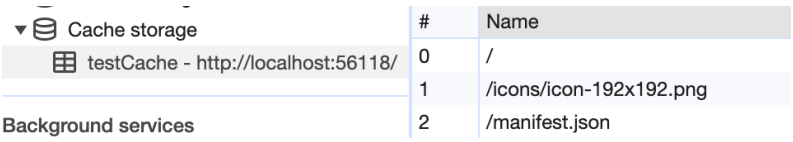


Abbildung 19: Cache Storage
Quelle: Screenshot Chrome DevTools

Betrachtet man den Value des ersten Schlüssels, so findet man tatsächlich das entsprechende Response Objekt, wie es vom Server geliefert wird. Als Payload findet man so das index.html der Webapp (vgl. Abbildung Anhang 7). Der Service Worker ist fähig, die vollständige Response des Servers abzuspeichern und diese

über einen Schlüssel, der normalerweise aus der Request URL besteht, zu identifizieren. Wird nun eine offline Situation simuliert und die Seite neu geladen, so lädt diese sofort, da der Service Worker dem Client die zuvor gecachte Response serviert.

Betrachtet man den Netzwerkverkehr, kann nachvollzogen werden, wie dies geschehen ist. Auf den ersten Blick fällt zunächst auf, dass einige Anfragen (rot markiert) nicht erfolgreich waren (vgl. Abbildung 20). Hier gilt es jedoch zu beachten, dass nicht nur der Client, sondern auch der Service Worker Anfragen schickt. Zieht man den *Initiator* Tab heran, so kann man feststellen, dass tatsächlich alle Anfragen des Client (*Other*), erfolgreich waren.

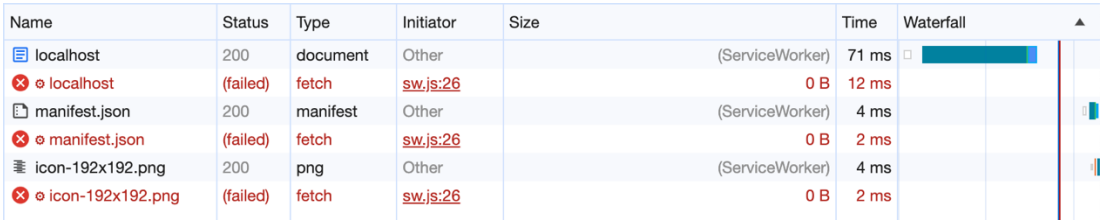


Abbildung 20: Netzwerk nach Offline Reload
Quelle: Screenshot Chrome DevTools

Die Antworten, die der Service Worker dem Client zuspielt, nachdem eigene Anfragen ins Netzwerk fehlgeschlagen sind, wurden zuvor vom Service Worker im Cache abgespeichert. Unter Heranziehen der Request URL konnte der Service Worker die passende Antwort identifizieren und dem Client bereitstellen. Hier kann am Beispiel des Request für das manifest.json file nachvollzogen werden (vgl. Abbildung 21), wie der Service Worker die passende Response ermittelt. Der Service Worker ermittelt über die Request URL des Client *http://localhost:51254/manifest.json* , und den Namen der Elemente im Cache, ein Match an Speicherstelle 2 (vgl. Abbildung 22). Wichtig zu erwähnen ist hier, dass im Cache die Bezeichnung immer relativ zur root der Webapp vorliegt. Der Pfad */manifest.json* ist daher gleichbedeutend mit */rootURL/manifest.json.*

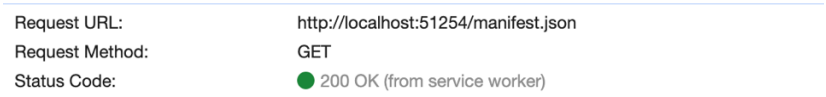


Abbildung 21: Example Request Header
Quelle: Screenshot Chrome DevTools

#	Name	Respon…	Content…	Content…	Time Ca…
0	/	basic	text/htm…	946	14/02/2…
1	/icons/icon-192x192.png	basic	image/png	10,830	14/02/2…
2	/manifest.json	basic	applicati…	398	14/02/2…

Abbildung 22: Cache Storage Items
Quelle: Screenshot Chrome DevTools

3.7 Precaching

Das vorangegangene Beispiel zeigt, wie der Service Worker Netzwerkanfragen abfangen, zwischenspeichern und bereitstellen kann. Um die Konzepte vollkommen verstehen zu können, welche nun folgen, muss noch einmal vor Augen geführt werden, welche Schritte der Service Worker durchläuft, um die Webapp schließlich offline zur Verfügung stellen zu können. Lädt die Webapp zum ersten Mal, so lädt der Browser den Service Worker herunter und installiert diesen. Nun muss die Seite erneut geladen werden, sodass der Service Worker die entsprechenden Requests abfangen, und die zugehörigen Responses speichern kann. Erst jetzt ist die App offline fähig. Besucht man die Webapp nur einmal, und versucht im Anschluss daran die App unter offline Bedingungen erneut aufzurufen, wird das nicht funktionieren. Der entscheidende Schritt ist das Laden der Webapp mit aktivem Service Worker. Da beim erstmaligen Besuchen der App, der Service Worker erst heruntergeladen und aktiviert wird, kann dieser noch nicht arbeiten. Erst beim erneuten Laden der Ressourcen kann der Service Worker diese verarbeiten.

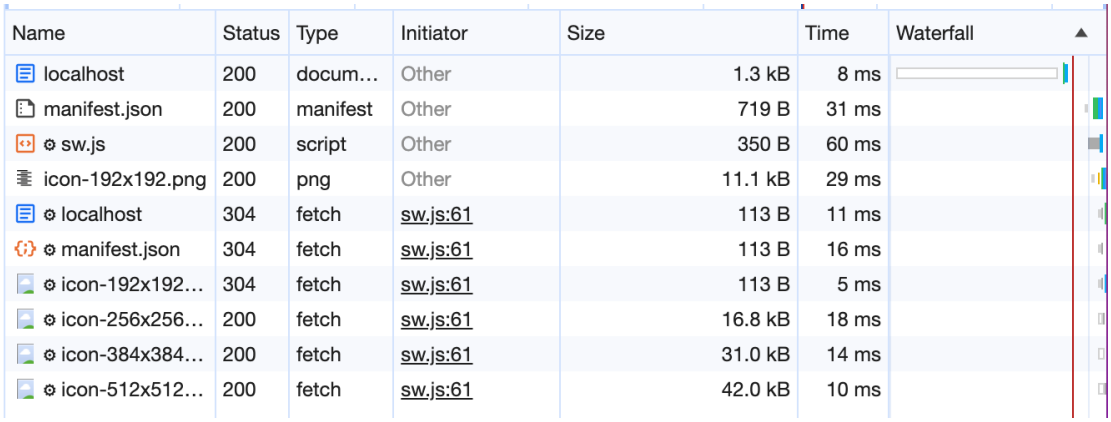
Das Konzept hinter dieser Vorgehensweise wird als Runtime Caching bezeichnet. Der Service Worker speichert Ressourcen, indem er Requests des Client abfängt. Bezogen auf die Beispiel App ist dies jedoch keine optimale Strategie, da die App nach einmaligem Besuchen noch keine offline Fähigkeiten hat. Als erste Demonstration der Fähigkeiten des Service Worker wurde ein Offline Fallback implementiert. Der Vorteil dieser Strategie war es, dass sobald der Service Worker aktiv war, was unmittelbar nach erstmaligem Besuchen der Webapp der Fall ist, das Offline Fallback funktioniert. Der Grund hierfür ist, dass die Response schon im Voraus bekannt war und daher schon vorab im Cache gespeichert werden konnte. Man könnte an dieser Stelle argumentieren, dass schlussendlich ja schon alle Ressourcen und deren requestURLs im Voraus bekannt sind. So sollte es theoretisch möglich sein den Cache, den der Service Worker bisher mit Hilfe der Runtime Strategie erstellt, manuell anzulegen. Ein aktiver Service Worker könnte dann diese Ressourcen dem Client zur Verfügung stellen, wenn keine Netzwerkverbindung besteht. Diese Überlegung wäre im Rahmen dieser, doch sehr minimalistischen, Webapp, theoretisch eine Lösung. Praktisch wäre die Herangehensweise, jede einzelne Ressource manuell in den Cache einzupflegen, in diesem Fall mit erheblichem Zeitaufwand verbunden, und im Falle einer leicht komplexeren Anwendung schon nicht mehr umzusetzen. Denkt man den Gedanken jedoch einen Schritt weiter, so kommt man auf ein Konzept, welches tatsächlich Anwendung findet, und als Precaching bekannt ist (Google LLC, 2024). Um die notwendigen Responses zu cachen, reicht es, wenn lediglich der Pfad bekannt ist. Mit Hilfe des Pfades kann der Service Worker dann die entsprechenden Ressourcen vom Server herunterladen und cachen. Die notwendige Information, die URL zu den Ressourcen, wird beim Runtime Caching vom Client geliefert, wenn dieser eine Anfrage sendet. Diese Information kann einfach vorab in das Service Worker-Skript geschrieben werden (vgl. Code Listing 2). Der Service Worker kann dann, sobald er aktiv ist, die entsprechenden Ressourcen anfragen und im Cache speichern.

```
//sw.js
const CACHE_NAME = "precaching-v1";

const urlsToCache = [
  "/",
  "/manifest.json",
  "/icons/icon-192x192.png",
  "/icons/icon-256x256.png",
  "/icons/icon-384x384.png",
  "/icons/icon-512x512.png",
];
```

Code Listing 2: precache manifest
Quelle: Eigene Darstellung

Lädt man die Seite zum ersten Mal, so kann im Netzwerktab (vgl. Abbildung 23) nachvollzogen werden, dass der Service Worker die festgelegten Ressourcen precached. Die Anfragen sind im Diagramm chronologisch von oben nach unten sortiert. Die Anfragen des Client, welche im *Initiator* Tab mit *Other* gekennzeichnet sind, erfolgen sinngemäß zuerst, wobei unter anderem der Service Worker heruntergeladen, und anschließend automatisch installiert wird. Sobald dieser aktiv ist, werden alle Ressourcen aus der Precaching-Liste angefragt, heruntergeladen und abgespeichert. Die Webapp ist nun nach einmaligem Besuch voll offline fähig.



Name	Status	Type	Initiator	Size	Time	Waterfall
localhost	200	docum…	Other	1.3 kB	8 ms	
manifest.json	200	manifest	Other	719 B	31 ms	
⚙ sw.js	200	script	Other	350 B	60 ms	
icon-192x192.png	200	png	Other	11.1 kB	29 ms	
⚙ localhost	304	fetch	sw.js:61	113 B	11 ms	
⚙ manifest.json	304	fetch	sw.js:61	113 B	16 ms	
⚙ icon-192x192…	304	fetch	sw.js:61	113 B	5 ms	
⚙ icon-256x256…	200	fetch	sw.js:61	16.8 kB	18 ms	
⚙ icon-384x384…	200	fetch	sw.js:61	31.0 kB	14 ms	
⚙ icon-512x512…	200	fetch	sw.js:61	42.0 kB	10 ms	

Abbildung 23: Netzwerk bei erstem Laden
Quelle: Screenshot Chrome DevTools

Die Analyse der Kerntechnologien, insbesondere des Service Worker, bildet das Fundament für die Entwicklung einer erfolgreichen Progressive Web App. Durch die Erklärung und Analyse des Service Worker in einem minimalen Beispielprojekt wurde veranschaulicht, wie Web Apps offline verfügbar gemacht werden können. Das in diesem Kapitel vermittelte Wissen bildet die Basis für den weiteren Fortlauf der Arbeit.

Im nächsten Schritt soll die Implementierung von Progressive Web App in ein modernes Webframework durchgeführt werden. Die Technologie wird verwendet, um eine komplexe Anwendung zu entwickeln, die in einer realen Umgebung zum Einsatz kommen könnte. Um diesen Entwicklungsprozess zu dokumentieren, wird der Software Engineering Prozess angewendet, der in den folgenden Kapiteln detailliert beschrieben wird.

Entwickelt wird eine Anwendung für das fiktive Waterplane Festival. Innerhalb der App soll der Zeitplan des Festivals interaktiv dargestellt werden.

4 Anforderungsanalyse und Spezifikation

Im Kontext der Entwicklung von Softwaresystemen ist eine gründliche Anforderungsanalyse von entscheidender Bedeutung. Sie bildet den Ausgangspunkt für die gesamte Entwicklung und legt den Grundstein für den Erfolg des Projekts. Die präzise Erfassung der Anforderungen ermöglicht es, die Bedürfnisse der Benutzer zu verstehen und letztendlich `umsetzen zu können. Diese Analyse unterteilt die Anforderungen in funktionale und nichtfunktionale Kategorien, wobei funktionale Anforderungen die Dienste und Reaktionen des Systems beschreiben und nichtfunktionale Anforderungen die Beschränkungen und Standards festlegen (Sommerville, 2018, p. 124).

4.1 Funktionale Anforderungen

In diesem Kapitel sollen zunächst die funktionalen Anforderungen der Waterplane App definiert werden. Dazu werden User Stories geschrieben, und mit Akzeptanzkriterien versehen. Die Akzeptanzkriterien werden später verwendet, um zu prüfen, ob die Anforderung erfüllt wurde oder nicht (Sommerville, 2018, p. 281).

Epic EP1: Als Nutzer möchte ich den Timetable anschauen.

User Story EP1 US1: Als Nutzer möchte ich eine Auflistung (chronologisch sortiert) aller Events sehen.

Akzeptanzkriterien:

1. Die Events werden chronologisch sortiert angezeigt.
2. Jedes Event zeigt den Titel, die Uhrzeit und den Veranstaltungsort an.

User Story EP1 US2: Als Nutzer möchte ich den Timetable nach Tagen filtern.

Akzeptanzkriterien:

1. Der Nutzer kann den Timetable nach verschiedenen Tagen filtern.
2. Die Events werden entsprechend dem ausgewählten Tag aktualisiert.

User Story EP1 US3: Als Nutzer möchte ich den Timetable nach Bühnen filtern.

Akzeptanzkriterien:

1. Der Nutzer kann den Timetable nach verschiedenen Bühnen filtern.
2. Die Events werden entsprechend der ausgewählten Bühne aktualisiert

Story EP2: Als Nutzer möchte ich die Sprache wählen können.

Akzeptanzkriterien:

1. Der Nutzer kann die Sprache der App aus einer Liste von unterstützten Sprachen auswählen.
2. Die App-Inhalte werden entsprechend der ausgewählten Sprache aktualisiert.

Story EP3: Als Nutzer möchte ich Events favorisieren können.

Akzeptanzkriterien:

1. Der Nutzer kann Events als Favoriten markieren.
2. Der Timetable ist filterbar, sodass nur die favorisierten Events angezeigt werden.

Story EP4: Als Nutzer möchte ich die App auf meinem Homescreen installieren.

Akzeptanzkriterien:

1. Der Nutzer erhält eine Option, die App auf seinem Homescreen zu installieren.
2. Die App wird als eigenständige Anwendung mit einem eigenen Symbol auf dem Homescreen installiert.

4.2 Nicht funktionale Anforderungen

Im Gegensatz zu den funktionalen Anforderungen beschreiben nicht-funktionale Anforderungen keine spezifischen Dienste des Systems, sondern beschreiben vielmehr dessen Eigenschaften und Einschränkungen. Beispiele für solche Eigenschaften sind Sicherheit, Performanz oder Zuverlässigkeit (Sommerville, 2018, p. 126)

Bei der Konzeption einer Timetable-App für ein Festival sind zwei nicht funktionale Anforderungen von zentraler Bedeutung: die Suchmaschinenindexierung und die Kosten für die Entwicklung. Die Suchmaschinenindexierung gewährleistet die Auffindbarkeit der App-Inhalte durch gängige Suchmaschinen wie Google oder Bing, wodurch die Zugänglichkeit für potenzielle Festivalbesucher erhöht wird und somit die Teilnahme am Fest positiv beeinflusst werden kann. Außerdem sollten die Entwicklungskosten im Rahmen gehalten werden, da das zur Verfügung stehende Budget begrenzt ist, welches eine Stadt für eine Festival-App aufbringen kann.

Zu den weiteren nicht funktionalen Anforderungen zählen:

Netzwerkunabhängig: Die App sollte auch offline funktionieren, da das Netzwerk bei einem solchen Event überlastet sein könnte.

Sicher: Die Verbindung soll über eine gesicherte Verbindung (HTTPS) erfolgen.

Aktuell: Die App soll automatisch neue Versionen herunterladen und installieren App ähnliches Verhalten: Die App soll aussehen und sich anfühlen, wie eine „App Store“ App, um Nutzern eine intuitive und reibungslose Nutzung zu ermöglichen.

4.3 Spezifikation

„Die Anforderungsspezifikation ist der Prozess, bei dem die Benutzer- und Systemanforderungen […] niedergeschrieben werden.“ (Sommerville, 2018, p. 141)

In diesem Kapitel sollen die User Stories aus Abschnitt 4.1 näher spezifiziert werden. Dazu sollen neben textuellen Beschreibungen auch Wireframes verwendet werden. Die Verwendung von Wireframes ermöglicht es, die Anforderungen visuell zu veranschaulichen und die Benutzeroberfläche der Anwendung zu konkretisieren.

4.3.1 Epic EP1

User Story EP1 US1:

„Als Nutzer möchte ich eine Auflistung (chronologisch sortiert) aller Events sehen.“,

Der Timetable des Festivals besteht aus Events, welche an einem bestimmten Tag, um eine bestimmte Uhrzeit und auf einer bestimmten Bühne stattfinden. Der Timetable soll alle Events des Festivals beinhalten und die Events nach der Uhrzeit sortiert darstellen. Dabei kann es sein, dass es mehrere Events zur selben Uhrzeit gibt, da mehrere Bühnen gleichzeitig bespielt werden. In dem Fall ist eine konsistente Sortierung nach Bühnen wünschenswert, sodass beispielsweise Bühne A immer vor Bühne B aufgeführt ist.

User Story EP1 US2

„Als Nutzer möchte ich den Timetable nach Tagen filtern.“

Weiter soll der Timetable nach Tagen gefiltert sein, sodass beispielsweise immer nur Events von Tag X oder nur die Events von Tag Y angezeigt werden.

User Story EP1 US3:

Als Nutzer möchte ich den Timetable nach Bühnen filtern.

Zusätzlich zur Filterung nach Tagen soll der Timetable nach Bühnen gefiltert werden können. Dabei soll es möglich sein, nach beliebig vielen Bühnen zu filtern. Es sollte gut ersichtlich sein, nach welchen Bühnen der Timetable aktuell gefiltert ist.

Der Timetable sollte unter einer ausdrucksstarken URL, wie beispielsweise <BASE_URL / TIMETABLE>zu finden sein, um leicht von Suchmaschinen gefunden zu werden.

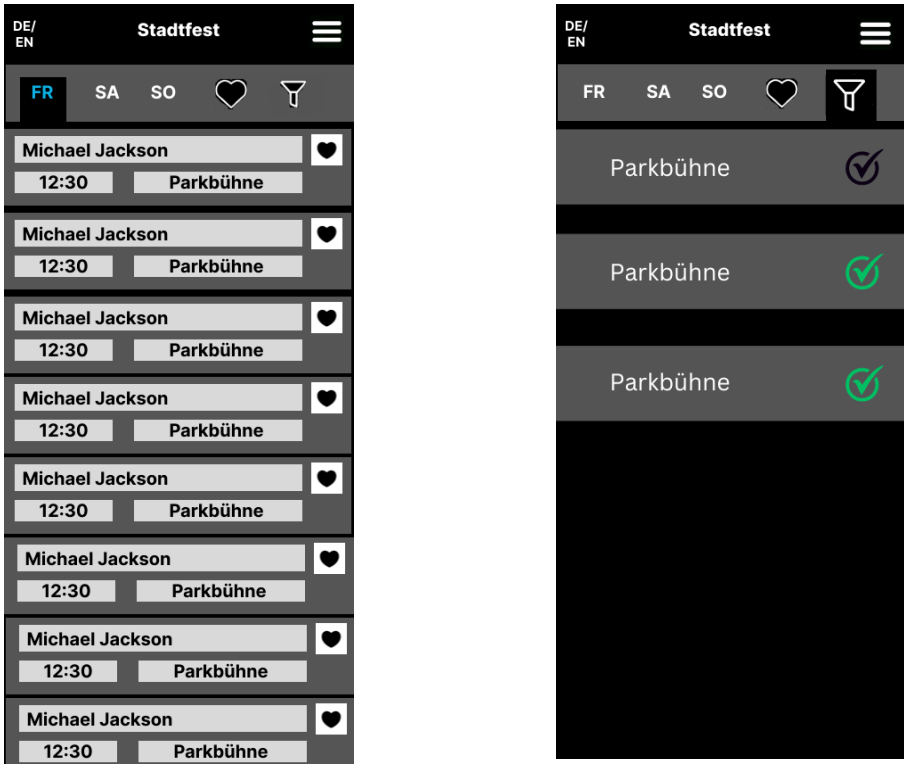


Abbildung 24: Wireframe zu Epic EP1: Timetable (links) und Filtermenü (rechts)
Quelle: Eigene Darstellung

4.3.2 Story EP2

„Als Nutzer möchte ich die Sprache wählen können."

Das Festival findet in einer deutschen Stadt statt, es wird aber auch mit internationalen Besuchern gerechnet. Daher sollte die App sowohl auf deutsch als auch auf englisch verfügbar sein. Die Sprachwahl soll auf der Startseite <BASEURL>, und unmittelbar nach Besuch der App, stattfinden.

4.3.3 Story EP3

„Als Nutzer möchte ich Events favorisieren können."

Jedes Event des Timetable sollte eine Option zur Favorisierung haben. Außerdem solle ersichtlich sein, ob ein Event bereits favorisiert wurde, oder nicht. Es soll die Option geben, dass nur favorisieren Events angezeigt werden.

Abbildung 25: Wireframe zu EP3
Quelle: Eigene Darstellung

4.3.4 Story EP4

„Als Nutzer möchte ich die App auf meinem Homescreen installieren."

Nutzer sollten dahin bewegt werden, die App auf ihrem Homescreen zu installieren. Es sollte entweder einen automatischen Mechanismus geben, die App nach Erlaubnis des Nutzers zu installieren, oder eine verständliche Beschreibung, wie der Nutzer die App manuell installieren kann. Ist die App bereits installiert, soll der Hinweis nicht mehr erscheinen.

4.4 Aktuelle Fähigkeiten von PWA

Nachdem die funktionalen und nicht funktionalen Anforderungen für das System festgelegt wurden, stellt sich die Frage, ob alle Anforderungen mit der Technologie Progressive Web App erfüllt werden können. Im folgenden Abschnitt sollen deshalb die Fähigkeiten, welche PWA hat, aufgeführt werden. Dabei werden nicht nur die Anforderungen, welche für die „Waterplane“ App benötigt werden, abgehandelt. Vielmehr wird ein breiteres Bild gezeichnet, um dem Leser ein Gefühl dafür zu geben, für welche Art von Anwendungen PWA geeignet ist, und für welche nicht.

Progressive Web Apps sind zwar Plattform unabhängig, ihre Funktionalität hängt jedoch von der Kompatibilität der jeweiligen Browser Engine ab. Während Progressive Web Apps auf Android sehr gut unterstützt werden, verlaufen Fortschritte von Apple bei der Optimierung des Supports für PWAs bisher vergleichsweise langsam (Tim A. Majchrzak, 2018). Betrachtet man die Fähigkeiten von PWAs, so sind diese, Plattform übergreifend betrachtet, nur so gut, wie sie Unterstützung bei Apple finden. Im Nachfolgenden soll deshalb die Unterstützung von PWA auf dem iPhone Betriebssystem iOS betrachtet werden.

Anmerkung

Während der Verfassung dieser Arbeit gab es massive Bewegungen im Bereich PWA Kompatibilität auf Apples Seite. Dies hing mit einer neuen Regelung der Europäischen Union, dem Digital Markets Act, zusammen, welcher eine Gleichbehandlung von App Anbietern vorsieht. Nachdem Apple zunächst PWAs von iOS völlig entfernen wollte, werden diese nun doch weiter unterstützt (Speckner, 2024). Leider sind die technischen Möglichkeiten von PWA auf iOS dadurch jedoch seit dem letzten iOS Update (17.4) stark eingeschränkt.

Zu den Fähigkeiten von PWA zählen (Steiner, 2018):

Standalone

PWAs können als Standalone Apps vom Homescreen aus gestartet, und im Vollbild Modus, also ohne Browser Navigationselemente, dargestellt werden. Das Erscheinungsbild von PWA ist deshalb im Grunde nicht von anderen Apps zu unterscheiden.

In der EU wird diese Möglichkeit mit dem neuesten iOS (17.4) seit März 2024 nicht mehr unterstützt. Hier sind PWA zwar immer noch vom Homescreen aus aufrufbar, jedoch handelt es sich lediglich um einen Weblink, welcher dann im Browser dargestellt wird. Das Erscheinungsbild gleicht dann einer herkömmlichen Website.

Push Notifications

Mit dem iOS update 16.4 hat Apple 2023 erstmals Push Notifications möglich gemacht und damit ein lange gewünschtes Feature integriert. Push Notifications können auch dann verschickt werden, wenn die App geschlossen ist. Das macht sie zu einem wichtigen Instrument der Nutzerbindung, da Nutzer mit Hilfe von Push Notifications immer wieder zur Verwendung der App bewegt werden können (Kumar, 2015). Diese Möglichkeit von PWA hat die Technologie massiv gestärkt und in Sachen Nutzerbindung an den nativen Ansatz aufrücken lassen. Seit den Änderungen aufgrund des DMA in iOS 17.4 (März 2024) ist die Funktion in der EU jedoch nicht mehr verfügbar, da PWAs hier keine Standalone Apps mehr sind.

Speicher

PWAs können auf iOS zunächst 50MB speichern. Durch die Nutzung von IndexedDB kann dieser Speicher jedoch (fast) beliebig erweitert werden (Love, 2021). Während für Webapps die Daten nach einer Woche gelöscht werden, und eine persistente Datenhaltung deshalb nicht möglich ist, können Standalone Apps, Daten quasi für unbegrenzte Zeit speichern und haben damit ähnliche Speichermöglichkeiten wie native Apps

Automatische Installation

Safari stellt, anders als Chrome auf Android, keine automatischen Installations-Prompts zur Verfügung. Nutzer müssen PWAs manuell zum Homescreen hinzufügen (Firtman, 2023).

USB, Bluetooth, NFC und Sensoren

PWAs haben auf iOS bislang keinen Zugriff auf Bluetooth, USB oder NFC (Firtman, 2023).

Suchmaschinenindexierung

PWAs können, anders als native Apps, von Suchmaschinen indexiert, und so leicht gefunden werden. Dies ist auf eine hybride Form des Renderns moderner JavaScript Frameworks zurückzuführen, wobei neben dem modularen, JavaScript kontrolliertem DOM, auch fertiges HTML bereitgestellt wird, welches von Suchmaschinen interpretiert werden kann (Russel, 2015).

Es gilt zu beachten, dass Einschränkungen, die auf den DMA zurückzuführen sind, nur in der Europäischen Union gelten. Möchte man die PWA jedoch auch im europäischen Markt verfügbar machen, so sind die Möglichkeiten von PWA eingeschränkt. Diese Einschränkungen für PWAs könnten jedoch mit der Zeit, und unter Druck, aufgehoben werden. Sollte Apple die entsprechenden Schnittstellen für Homescreen Apps bereitstellen, sodass andere Browser Anbieter eigene Browser Engines entwickeln können, so könnten PWAs in Zukunft (wieder) besser werden und eventuell sogar Android ähnliche Unterstützung erreichen.

Diese Arbeit orientiert sich an der Situation vor den Änderungen des DMA, da die Arbeit schon weit fortgeschritten war, als die Änderungen publik wurden. Demnach könnte eine PWA alle fachlichen Anforderungen erfüllen.

5 Architektur und Technologien

„Eine Softwarearchitektur ist eine Beschreibung, wie ein Softwaresystem strukturiert ist. Systemeigenschaften wie Leistung, Informationssicherheit und Verfügbarkeit werden von der verwendeten Architektur beeinflusst“ (Sommerville, 2018, p. 219). Die Softwarearchitektur beschreibt, wie das System organisiert ist, und legt damit den Rahmen für dessen Verhalten fest. Daher ist sie ein zentraler Faktor für die Funktionalität der Software. Die Auswahl der eingesetzten Technologien und des Frameworks ist entscheidend, ob und wie gut die nichtfunktionalen Anforderungen von dem System erfüllt werden können

Im Kern dieses Kapitels geht es zum einen um die Auswahl geeigneter Frameworks und Bibliotheken, zum anderen wird die Architektur des Service Worker erarbeitet.

5.1 Technologie Auswahl

In diesem Abschnitt sollen drei der beliebtesten Frameworks für Web Apps erkundet werden: React[6], Angular[7] und Vue[8] (Statista, 2023)

Das Fundament für Web Apps, die in ihrer Bedienung und Erscheinung nativen Apps, in nichts nachstehen, war die Entwicklung von modernen JavaScript

[6] https://react.dev/
[7] https://angular.io
[8] https://vuejs.org/

Frameworks (Delcev, 2018). Durch die Nutzung von virtuellem DOM und reaktiven Systemen verbesserten React und Vue erheblich die Rendering-Effizienz und -Leistung (Diniz-Junior, 2022). In den frühen Tagen der Webentwicklung basierte die Erstellung von Websites stark auf manuellem HTML-, CSS- und JavaScript-Coding. Es wurden serverseitige Rendertechniken genutzt, komplette Webseiten wurden dynamisch auf dem Server generiert, was für jede Interaktion vollständige Neuladungen erforderte, (Karwan Jacksi, 2019). Moderne JavaScript Frameworks führten stattdessen eine komponentenbasierte Architektur ein, die es ermöglicht, wiederverwendbare und modulare UI-Komponenten zu erstellen, welche einzeln aktualisiert werden können. Clientseitiges Rendering, bei welchem UI-Elemente vom JavaScript im Browser gerendert werden, führt zu reaktionsfähigeren Benutzererfahrungen. Diese waren bis dahin nur von nativen Apps bekannt. Die Frameworks erlauben es, Web Apps zu erstellen, welche, in Kombination mit progressiver Verbesserung durch Service Worker, mit nativen Apps konkurrieren können (Kitsios, 2015). Im Folgenden sollen die drei bekanntesten dieser Frameworks vorgestellt werden.

5.1.1 React, Angular und Vue

React, welches von Facebook entwickelt wurde, ist das am meisten genutzte Framework in der Landschaft der Webanwendungen (Stack Exchange Inc , 2024). Es hat eine komponentenbasierte Architektur und wird durch eine Vielzahl von Libraries wie Create React App und Next.js, unterstützt. Create React App[9] ermöglicht es nahezu konfigurationslos eine PWA zu erstellen, indem es einen Ser-

[9] https://create-react-app.dev/docs/making-a-progressive-web-app/

vice Worker registriert, ein App Manifest verlinkt und Optionen für Pre- und Runtime Caching bereithält (Facebook, 2022). React wird primär in JavaScript entwickelt, unterstützt jedoch auch TypeScript (Meta Open Source, 2024).

Angular, entwickelt von Google, ist neben React das am zweitstärksten verbreitete Framework für Webanwendungen (Stack Exchange Inc , 2024). Mit seiner komponentenbasierten Architektur und der umfangreichen Unterstützung durch Tools wie Angular CLI[10] und Angular-Universal bietet Angular effiziente Lösungen für die Erstellung von PWAs (Google LLC, 2024). Angular CLI ermöglicht eine schnelle Einrichtung von PWA-Projekten durch die Generierung von Projektstrukturen, ähnlich wie create react app. Durch die Integration von Angular Universal wird die Möglichkeit des serverseitigen Renderns eröffnet, was zu einer verbesserten Leistung und Suchmaschinenoptimierung der PWAs führt (Google LLC, 2024). Die Entwicklung erfolgt ausschließlich in TypeScript. Angular ist dabei nicht mit AngularJS zu verwechseln, welches das Vorgängerframework ist, heutzutage jedoch keine bedeutende Rolle mehr spielt (vgl. Abbildung Anhang 10) (Stack Exchange Inc , 2024).

Vue, entwickelt von Evan You, ist anders als React und Angular kein Produkt eines großen Softwarekonzerns, sondern wird hauptsächlich von der Community gestützt (Shah, 2024). Mit seiner reaktiven und komponentenbasierten Architektur bietet Vue.js eine flexible und benutzerfreundliche Lösung für die Entwicklung moderner Webanwendungen. Vue bietet eine Vielzahl von Tools, darunter Pinia und Vue Router, die State Management und Routing des Projekts vereinfachen (Evan You, o.J), (Morote, o.J). Darüber hinaus bietet Vue.js eine umfangreiche Bibliothek von UI-Komponenten sowie Unterstützung für TypeScript, was zu einer verbesserten Codequalität und Wartbarkeit führt (Shah, 2024).

[10] https://angular.io/cli

5.1.2 Vergleich der Frameworks

React und Vue sind zunächst mit einem minimalen Set an Funktionalität ausgestattet, welches durch Libraries erweitert werden kann. Vor allem bei Vue sind diese hauptsächlich von der Community gestützt. Angular dagegen stellt ein weitaus größeres Set an Funktionalität bereit, weshalb hier oftmals keine weiteren Bibliotheken eingebunden werden müssen (Sevim, 2023). Angular macht darüber hinaus die strengsten Vorgaben zur Code Strukturierung und folgt dem Model View Controller Pattern. Bei React und Vue muss ein größerer Fokus auf die Beachtung von Architektur und Entwicklungspattern gelegt werden, da vor allem bei React Funktionalität und Repräsentation durch die JSX-Syntax verschwimmen (Jelica Cincović, 2020). Vue ist schnell zu lernen und hat eine abgegrenzte Komponentenstruktur. Darüber hinaus ist es auch für kleinere Projekte gut geeignet, da das Projekt durch den begrenzten Funktionsumfang weniger Speicher benötigt. Die lebhafte Community stellt ein diversifiziertes Ökosystem mit verständlichen Dokumentationen bereit (Jelica Cincović, 2020).

Die Integration von PWA in ein modernes Framework soll hier am Beispiel von Vue gezeigt werden. Vue eignet sich hervorragend für die Entwicklung eines kleineren Projekts, wie es im Zuge dieser Arbeit umgesetzt werden soll.

5.1.3 Nuxt

Die Navigation innerhalb einer SPA unterscheidet sich von herkömmlichen Websites darin, dass nicht zwischen verschiedenen HTML-Seiten, welche durch einzigartige URLs identifiziert werden, hin und her navigiert wird, sondern nur so-

genannte Views innerhalb einer einzigen HTML gewechselt werden. Für Suchmaschinen ist die Indexierung dieser Views zunächst nicht möglich. Nuxt[11] ist ein Framework, welches auf der Grundlage von Vue aufgebaut ist, und unter anderem Server Side Rendering (SSR) von SPAs möglich macht. Nuxt nutzt dabei Node.js um serverseitig statisches HTML zu generieren, welches durch Suchmaschinen erfasst werden kann. Die App bleibt jedoch im Kern JavaScript gesteuert, da die HTML-Elemente durch einen Hydration Prozess reaktiv gemacht werden, sobald der Browser diese geladen hat (Clements, 2024). Darüber hinaus stellt Nuxt ein Plugin bereit, um jede View, einen zugeschnittenen Meta Tag, anzuheften. Meta Tags enthalten Zusammenfassungen, Zeichencodierungen und andere wichtige Informationen, welche Suchmaschinen dabei helfen, den Inhalt der Seite korrekt zu interpretieren (Nuxt , 2024).

5.1.4 VitePWA

Vite[12] ist ein build Tool, welches von Nuxt standartmäßig verwendet wird. Es beinhaltet eine Reihe von Features, wie beispielsweise Hot Module Replacement (HMR), um eine responsive Entwicklungsumgebung bereitzustellen. Darüber hinaus fungiert Vite als Bundler für die Production (You & Contributors, 2024). Wie bereits im Abschnitt erklärt, wird für Precaching eine Liste der Dateipfade benötigt, um diese direkt nach Aktivierung des Service Worker anfragen zu können. Der build Prozess ist deshalb der Moment, wo ein PWA Plugin einhakt, um die Dateipfade der build Dateien abzufangen, um diese, als Precache Manifest, in die Service Worker Datei hinein zu injizieren. VitePWA[13] ist ein Modul, welches Zero Config PWA Integration für eine Reihe von Frameworks bereitstellt, darunter

[11] https://nuxt.com/
[12] https://vitejs.dev/
[13] https://vite-pwa-org.netlify.app/

React, Svelte und Nuxt. Das Modul stellt Komponenten zur einfachen Implementierung eines App Manifest bereit und registriert einen Service Worker (You & Contributors, 2024).

5.1.5 Workbox

Steht man vor der Aufgabe, einen Service Worker für eine komplexe Applikation zu entwerfen, so begegnen Entwickler immer wieder denselben Herausforderungen (Google LLC, 2024). Damit das Rad nicht immer wieder neu erfunden werden muss, kann eine Bibliothek viel Zeit ersparen: Workbox.

Workbox ist eine Zusammenstellung von JavaScript Bibliotheken, welche von Google unterhalten wird und die Erstellung eines Service Worker vereinfacht. Workbox enthält eine Vielzahl vorgefertigter Runtime-Caching-Strategien, die auf unterschiedliche Netzwerkgeschwindigkeiten und zu cachende Ressourcen abgestimmt sind. Diese Strategien sind leicht implementierbar und ermöglichen eine schnelle und unkomplizierte Integration. Workbox integriert nahtlos mit bewährten build Tools, was eine einfache Einbindung in den build Prozess ermöglicht, und von PWA-Modulen, wie VitePWA, verwendet wird, um Precaching zu realisieren. Darüber hinaus stellt es Background Synchronisation und offline Support zur Verfügung. Workbox ist sehr weit verbreitet und verfügt über eine gute und ausführliche Dokumentation[14] (Google LLC, 2024).

[14] https://developer.chrome.com/docs/workbox

5.2 Application Shell Architecture

Mit Workbox haben Entwickler ein umfassendes Werkzeug an der Hand, um Caching Strategien zu implementieren. Spätestens jetzt stellt sich jedoch die Frage, welche Ressourcen überhaupt gecacht werden sollen. Bisher war die Herausforderung, Inhalte offline verfügbar zu machen, technischer Natur. Was in diesem Abschnitt jedoch beleuchtet werden soll, ist eine übergreifende Strategie, nach welcher Ressourcen, beziehungsweise Teile der App, gecached werden sollen. Hilfreich ist hierbei die Überlegung, in welche Teile die meisten Apps gegliedert werden können. Zumindest für SPAs gilt, dass es meist ein Set an Navigationselementen gibt, die häufig in einem Header oder Footer untergebracht sind. PWAs, welche Standalone bedienbar bleiben sollen, können sich nicht auf die Navigationselemente des Browsers stützen und sind deshalb mit eigenen Navigationselementen ausgestattet. Die Application Shell beschreibt das minimale HTML, CSS und JavaScript, welches das Grundgerüst der UI darstellt (Osmani, 2015). Osmani vergleicht die App Shell mit dem Teil des Codes einer nativen App, welcher auf dem App Store veröffentlicht wird. Bei Installation der App wird die UI lokal gespeichert, während die Inhalte häufig dynamisch durch APIs herangezogen werden. Für den Entwurf einer PWA nach dem Application Shell Model gilt derselbe Grundsatz: Die UI ist lokal gespeichert (gecached), während die Inhalte dynamisch heruntergeladen und dargestellt werden. In Abbildung 26 kann das Application Shell Model am Beispiel der Google Maps Web App nachvollzogen werden. Der Header mit Navigationselementen ist dabei die Application Shell, während die Karte dynamisch geladen und angezeigt wird.

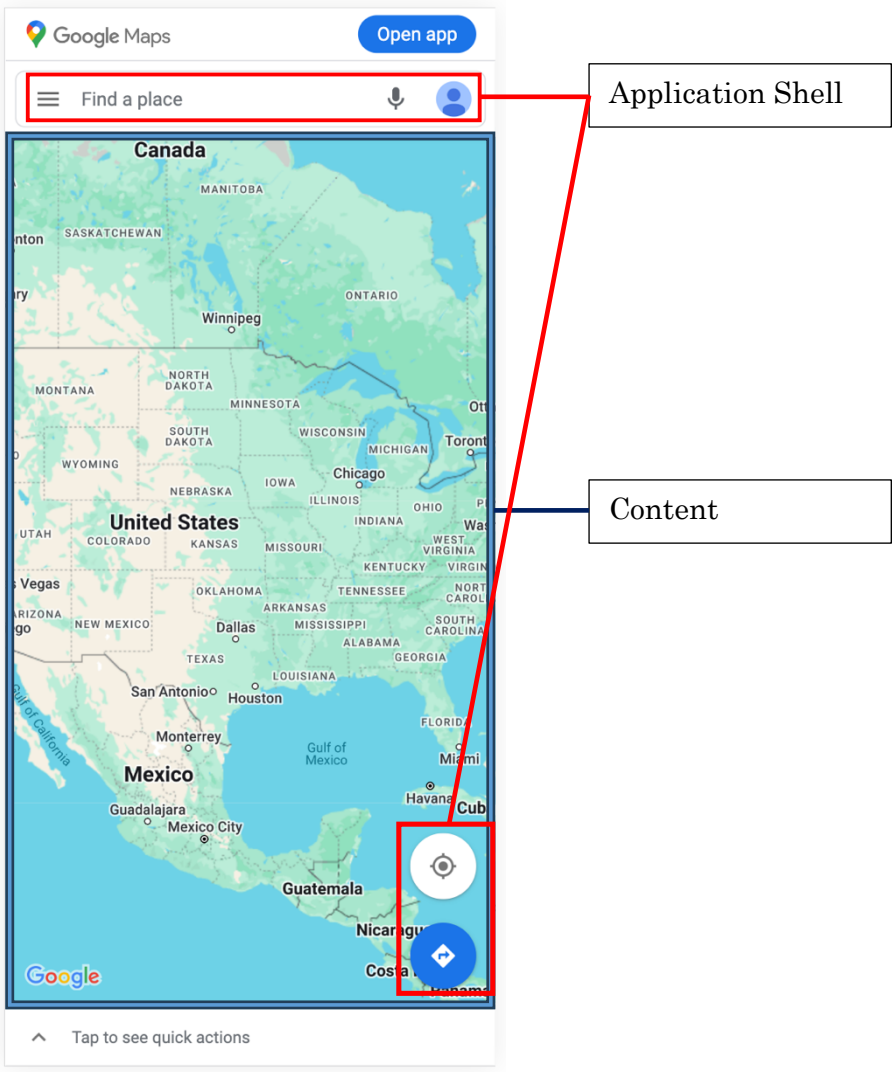


Abbildung 26: Applicaton Shell Model
Quelle: Eigene Darstellung

Das Application Shell Model eignet sich dann, wenn dynamische Inhalte innerhalb eines statischen Containers dargestellt werden sollen. Für die „Waterplane“ ist dies der Fall, da die dynamischen Inhalte des Timetable innerhalb einer statischen Hülle, welche Navigations- und Filterelemente enthält, dargestellt werden sollen.

5.3 Datenbank

Um Inhalte dynamisch zu laden und anzeigen zu können, wird eine Datenbank benötigt. Für die Waterplane App, bei der zum einen die zu speichernden Daten nicht besonders komplex sind, und zum anderen die Pflege der Daten im besten Fall von einer internen Kraft, welche keinen technischen Hintergrund besitzt, erfolgen kann, bieten sich Content Management Systeme an. Während bei klassischen CMS, wie Wordpress, das Frontend an die Datenhaltung gekoppelt ist, bieten Headless CMS die Möglichkeit, Daten über APIs zu beziehen, und diese in ein beliebiges Frontend einzubetten. Im Bereich Headless CMS gibt es inzwischen eine Vielzahl an Optionen. Laut Brannon (Brannon, 2024) gehören zu den bekanntesten:

- Wordpress Headless
- Storyblok
- Contentful

Wordpress Headless ist die Einführung von Representational State Transfer (REST) API in Wordpress 4.7, welches die Möglichkeit gibt, Wordpress Inhalte Frontend unabhängig zu beziehen (Erdmann, 2023). Wordpress ist ein open source Projekt und daher grundsätzlich kostenlos, die Einrichtung des Systems ist jedoch mit einem gewissen Aufwand verbunden und es können Kosten für Domain und Host anfallen (McCollin, 2023).

Storyblok ist ein Headless CMS Service, welcher die Erstellung, Verwaltung und Distribution von Content mit Hilfe einer herausragenden Bedienoberfläche, REST und GraphQL APIs, sowie Software Development Kits (SDKs) für eine einfache Integration mit bekannten Web Frameworks bietet. Storyblok erlaubt einen kostenlosen Account mit einem Space, einem Nutzer und 250 Gigabyte monatlichem Durchlauf (Storyblok GmbH , 2024).

Contentful ist laut Brannon (Brannon, 2024) inzwischen das am weitesten verbreitete Headless CMS auf dem Markt. Contentful ist ähnlich wie Storyblok ein Content as a Service Anbieter, welcher seine Ursprünge in Berlin hat, und einen kostenlosen Account für Privatpersonen zur Verfügung stellt. Contentful bietet eine schnell zu lernende Benutzeroberfläche zur Erstellung eines Content Models und zur Verwaltung der Daten, GraphQL und REST APIs zur Distribution, sowie SDKs für eine unkomplizierte Integration in moderne Web Frameworks (Contentful, 2024). Contentful eignet sich durch seinen großzügigen Free Tier, sowie der intuitiven Benutzeroberfläche hervorragend für ein Projekt wie die Waterplane App und soll deshalb in diesem Projekt genutzt werden.

6 Implementierung

6.1 Versionsverwaltung

Für die Versionsverwaltung wurde das Versionsverwaltungstool Git[15] verwendet. Innerhalb von Git wurde das Git Flow Branching Model angewandt. Dabei werden Branches mit verschiedenen Funktionen vorgegeben und deren Zusammenhänge definiert (Demkovych, 2022).

Der Vorteil des Models ist, dass der Master Branch zu jeder Zeit in einem Release fähigen Zustand ist. Für die Entwickung wird zunächst ein Development Branch vom Master Branch abgezweigt, von welchem wiederum Feature Branches abzweigen. Diese wurden in diesem Projekt nach dem Schema feature/featureBezeichnung benannt. Feature Branches werden dann zunächst in den Development Branch gemerged. Hier können Bugs gefunden und ausgebessert werden. Bevor der stabile Development Branch final wieder in den Masterbranch gemerged werden kann, wird zunächst ein release Branch vom Development Branch abgezweigt. Hier werden ausschließlich Bugs ausgebessert. Der release Branch wird anschließend in den Master Branch gemerged (Demkovych, 2022).

[15] https://git-scm.com/book/de/v2/Erste-Schritte-Was-ist-Versionsverwaltung%3F

Bugs im Master Branch, welche durch merging Konflikte entstehen, können in einem, vom Master Branch abzweigenden Hotfix Branch verbessert werden, welcher anschließend wieder in den Master Branch gemerged wird (Demkovych, 2022). Das gesamte Modell kann in Abbildung Anhang 11 nachvollzogen werden.

Da die Waterplane App während des Entwicklungszeitraumes nicht kontinuierlich integriert wurde, wurde der Einfachheit halber auf Release Branches verzichtet.

6.2 Statische Codeanalyse

Um den Code Stil einheitlich zu halten sind Tools zur statischen Codeanalyse hilfreich. Hierbei handelt es sich um eine Reihe automatisierter Prüfungen, denen der Quellcode unterzogen wird. Im Zuge dieses Projekts wurden drei Tools zur statischen Codeanalyse verwendet: Volar, Prettier[16] und ESLint[17].

Volar ist der offizielle Syntaxsupport für Vue3. Vue Komponenten bestehen aus Blocks von HTML, JavaScript oder TypeScript und CSS. Darüber hinaus nutzt Vue3 eigenen Syntax. Volar unterstützt einen effizienten Workflow, indem es auch Vue spezifischen Code vervollständigen kann. Darüber hinaus hat das Tool weitere nützliche Features, welche vor allem bei der Nutzung von TypeScript Zeit ersparen können. Dazu gehört unter anderem die direkte Navigation zu Type Definitions.

Prettier ist ein Tool zur Code Formatierung, welches den Code bei Speicherung formatiert. Prettier kann mit Hilfe eines `"prettier"` Schlüssels in der package.json Datei, einer „`.prettierrc`“ JSON Datei oder in den VS Code Einstellungen konfiguriert werden. Prettier erhält die Kontinuität des Codestils, was zu

[16] https://prettier.io/
[17] https://eslint.org/

einer effizienten Wahrnehmung des Codes führt. Vor allem bei der Mitwirkung mehrerer Entwickler ist ein einheitlicher Codestil besonders wichtig.

ESLint ist ein weiteres Tool zur statischen Codeanalyse. ESLint nutzt node.js und kann einfach mit dem Node Package Manager (npm) installiert werden. Wie Volar und Prettier lässt sich ESLint auch über eine offizielle VSCode Extention verwalten. ESLint analysiert dabei den Code nach zuvor festgelegten Parametern. So kann beispielsweise festgelegt werden, ob Strings in einfachen oder doppelten Anführungszeichen stehen sollen oder ungenutzte Variablen existieren. Die Konfiguration findet in einer Textdatei im root Directory statt. Wie bei Prettier wird der Code bei Speicherung automatisch verbessert.

6.3 Service Worker Design

Das Design des Service Worker folgt dem Application Shell Model, wobei die Application Shell direkt nach Aktivierung des Service Worker gecacht wird, um bei nachfolgenden Besuchen sofort verfügbar zu sein. Der Content soll durch eine Runtime Caching Strategie gecacht werden, um auch dann verfügbar zu sein, wenn die Netzwerkverbindung schlecht oder überhaupt nicht vorhanden ist.

Um diese Anforderungen umzusetzen, wird ebenfalls die Bibliothek Workbox verwendet. Dateien, welche dem Precache Manifest hinzugefügt werden sollen, können in der nuxt.config Datei unter dem Workbox Schlüssel globPatterns, mit Hilfe eines regulären Ausdrucks festgelegt werden. Der default: ["**\/*.{js,css,html}"], zieht dabei alle JavaScript, CSS und HTML-Dateien des Directories und genesteten Directories ein. Da die gesamte App Shell gecacht werden soll, sollen, neben dem index.html, alle JavaScript und Stylesheets, die nach build verfügbar sind, in das Precache Manifest injiziert werden. Die globPatterns Option kann deshalb bei

default belassen werden. Innerhalb des Service Worker wird die Workbox Funktion precacheAndRoute verwendet, die einen Array von Objekten mit einem Pfad und einem Revision Hash erwartet. Dieser Array wird auch als precache Manifest bezeichnet und wird während des build Prozesses von dem verwendeten Workbox Modul workbox-build erstellt. Das Precache Manifest wird zunächst durch einen speziellen Platzhalter, „self.__WB_MANIFEST“, gekennzeichnet. Workbox injiziert dann das während des build Prozesses erstellte Precache Manifest in den Platzhalter. Da die HTML Dateien dynamisch auf dem Server generiert werden, können diese nicht während des build Prozesses abgegriffen werden. Sie werden deshalb noch nachträglich dem Precache Manifest hinzugefügt. Das vollständige Precache Manifest wird dann der Funktion recacheAndRoute übergeben (vgl. Code Listing 3). Diese legt einen entsprechenden Cache an und implementiert Logik, um Anfragen nach gecachten Ressourcen mit einer cache first Strategie zu bedienen. Dabei gilt es noch anzumerken, dass Workbox die Dateien aktualisiert, wenn eine neue Version verfügbar ist. Durch den Revision Hash kann Workbox verschiedene Versionen von Dateien unterscheiden.

Für den Content der Waterplane App, welcher von der Contentful API kommt, wird mit einer Stale While Revalidate Strategie gearbeitet (vgl. Code Listing 3). Bei dieser wird zunächst die angefragte Ressource aus dem Cache bedient und parallel eine Netzwerk-Anfrage gestellt, mit deren Antwort der Cache für die nächste Anfrage der Ressource aktualisiert wird.

```
import { precacheAndRoute} from "workbox-precaching";
import { registerRoute} from "workbox-routing";
import { ExpirationPlugin } from "workbox-expiration";
import { StaleWhileRevalidate } from "workbox-strategies";
import { CacheableResponsePlugin } from "workbox-cacheable-response";

//make sw take control immediately and all clients
self.addEventListener("install", () => self.skipWaiting());
self.addEventListener("activate", () => self.clients.claim());

//precache
precacheAndRoute([
  ...self.__WB_MANIFEST,
  { revision: "", url: "/" },
  { revision: "", url: "/?standalone=true#/" },
  { revision: "", url: "/timetable" },
]);


//runtime cache contentful requests
registerRoute(
  ({ request }) =>
    request.url && request.url.startsWith("https://cdn.contentful.com"),
  new StaleWhileRevalidate({
    cacheName: "contentful",
    plugins: [
      new CacheableResponsePlugin({ statuses: [200] }),
      new ExpirationPlugin({
        maxEntries: 50,
        maxAgeSeconds: 60 * 60 * 24 * 30,
      }),
    ],
  })
);
```

Code Listing 3: Service Worker
Quelle: Eigene Darstellung

7 Test und Veröffentlichung

„Tests sollen zeigen, dass ein Programm, das tut, was es tun soll, sowie Programmfehler vor der Benutzung aufdecken.“ (Sommerville, 2018, p. 256) Sie sind dabei Teil der übergeordneten Softwarevalidierung und Softwareverifikation, wobei die Verifikation die Erfüllung der funktionalen und nicht-funktionalen Anforderungen beurteilt, und die Validierung die allgemeine Brauchbarkeit der Software für ihren Verwendungszweck bewertet.

Bei größeren Softwareprojekten ist es sinnvoll, eine automatisierte Testumgebung zu erstellen, wobei Tests beispielsweise nach jedem Commit automatisch durchgeführt werden. Der Vorteil besteht hier darin, dass die Software in sehr vielen Iterationen getestet wird, und Fehler dadurch genauer bestimmten (neu hinzugekommenen) Codeabschnitten zugeordnet werden können (Amazon Web Services, 2024). Dieses Vorgehen wird auch als Continuous Integration (CI) bezeichnet. Es fand jedoch in diesem Entwicklungsprozess keine Anwendung, da der zeitliche Aufwand den Umfang dieser Arbeit überschritten hätte.

Stattdessen wurde neben einzelnen Tests für isolierte Komponenten hauptsächlich auf Programminspektionen gesetzt. Laut Sommerville (Sommerville, 2018, p. 259) sind Programminspektionen effizienter als Softwaretests und können bis zu 90% aller Fehler innerhalb eines Softwaresystems finden.

Die Inspektion einer Anwendung scheint zunächst sehr einfach, handelt es sich jedoch um eine Progressive Web App, sind einige Dinge nennenswert. Durch das Heranziehen von Ressourcen aus dem Cache, kann es dazu kommen, dass Änderungen im Code, nicht in der visuellen Darstellung der App reflektiert werden.

Dies liegt daran, dass die Version, welche im Code Editor zu sehen ist, vom Browser ignoriert wird, da dieser noch eine alte Version der App im Cache findet. Bevor die App inspiziert wird, muss deshalb zunächst sichergestellt werden, dass keine Rückstände älterer Versionen der App mehr im Cache zu finden sind. Hilfreich ist hierbei die Verwendung von privaten Fenstern, da diese keine Rückstände hinterlassen, und der Browser deshalb auf jeden Fall die aktuelle Version anfragen muss.

Aus Sicherheitsgründen gilt für PWAs, dass eine https Verbindung zwischen Client und Server bestehen muss. Localhost ist hierbei eine Ausnahme und kann für das Testen von PWAs auf der lokalen Maschine verwendet werden. Möchte man die PWA jedoch auf einem Smartphone testen, kann sogenanntes port forwarding[18] in Google Chrome verwendet werden. Localhost wird hier an einen anderen Port gebrückt und kann dann von einem Smartphone erreicht werden. Leider lassen lediglich Android Geräte PWA Fähigkeiten über einen gebrückten Localhost zu.

Um eine PWA auf iOS zu testen kann ein Tunnel Service wie ngrok[19] verwendet werden. Hier wird der lokale Server (localhost) mit Hilfe eines Tunnels, welcher über https aufgebaut wird, dem Internet geöffnet. Um die PWA zu testen, kann der lokale Server über den https Tunnel aufgerufen werden.

7.1 Offline Test

Um die Offline Verfügbarkeit der App zu testen wurden die Google DevTools verwendet. Die Standalone App muss einmal vom Homescreen gestartet werden, um offline zu funktionieren, da der Service Worker nicht durch alleiniges Hinzufügen

[18] https://chromeos.dev/en/web-environment/port-forwarding
[19] https://ngrok.com/

heruntergeladen und aktiviert wird. Die App ist darüber hinaus voll offline fähig. Die nichtfunktionale Anforderung der Netzwerkunabhängikeit ist daher gegeben.

7.2 Auditing

Um die Fähigkeiten der App objektiv bewerten zu können, wurden zwei Auditing Tools herangezogen. Um eine umfassende Bewertung der App zu erhalten, wurde das Tool Lighthouse genutzt. Da für die Waterplane App die Auffindbarkeit von großer Bedeutung ist, wurde außerdem das Auditing Tool WooRank verwendet, um eine tiefergehende Analyse der Suchmaschinenoptimierung zu erhalten. Die Ergebnisse der Audits sollen im Folgenden vorgestellt werden.

7.2.1 Lighthouse

Lighthouse ist ein Open Source Auditing Werkzeug, welches von Google entwickelt und bereitgestellt wird. Lighthouse[20] kann mit Hilfe des Node Package Manager (NPM) installiert oder direkt in den ChromeDevTools ausgeführt werden und beliebig Webseiten testen. Ermittelt wird dabei unter anderem Performanz, Accessibility, Best Practices, SEO und PWA Score (Google LLC, 2024).

Im Lighthouse Test erfüllt die Waterplane App alle Voraussetzungen einer PWA und erreicht top Wertungen in den Bereichen Accessibility, Best Practices und SEO (vgl. Abbildung 27).

[20] https://developer.chrome.com/docs/lighthouse/overview/

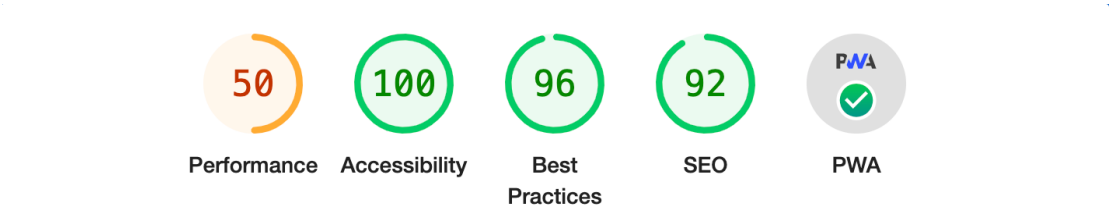


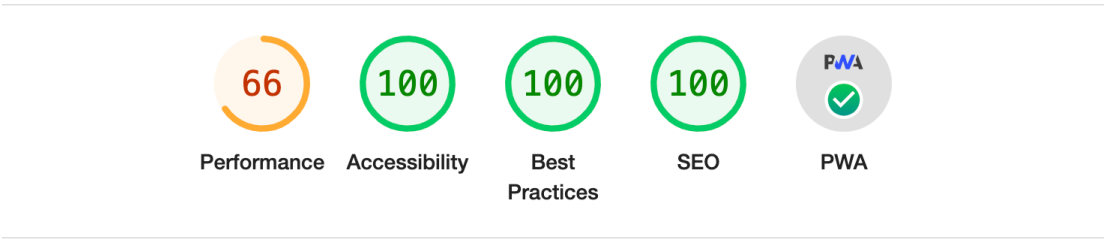
Abbildung 27: Waterplane Lighthouse Audit
Quelle: Screenshot Google Lighthouse

Bei der Performanz erreicht die App leider nur einen mangelhaften Wert. Der Performanz Score bildet vor allem die Geschwindigkeit des Ladens der App ab, und nicht die Responsivität.

Das Lighthouse Audit zeigt, dass viele der nichtfunktionalen Anforderungen umgesetzt werden konnten. So ist die App sicher, da die Verbindung über https erfolgt. Außerdem ist die App immer aktuell, da sie mit einem Service Worker ausgestattet ist, welcher immer die aktuellste Version der App herunterlädt, ohne dass Aktionen des Nutzers notwendig sind.

Ein Problem, welches bei Betrachtung des Netzwerkes aufgefallen ist, ist die Größe des entry Files, welches mit 1,6 MByte bei weitem zu groß scheint. Nach einiger Suche ist aufgefallen, dass dies an der verwendeten UI Library NaiveUI liegen könnte. Nachdem das Plugin deaktiviert wurde, konnte eine deutliche Verkleinerung des entry Files festgestellt werden. Das Plugin war zunächst global installiert, was einen unnötigen Overhead verursachte. Nachdem nur die verwendeten Komponenten der Library on Demand importiert wurden, konnte der Performanz Score etwas verbessert werden.

Weitere Maßnahmen zur Performanzoptimiertung werden im Kapitel 9.2 diskutiert



7.2.2 WooRank

Da Suchmaschinenoptimierung besonders wichtig für den Use Case des Waterplane Prototyps ist, wurde beim Testing besonderes Augenmerk auf die Auffindbarkeit der App gelegt. Um die Suchmaschinenoptimierung spezifischer untersuchen zu können, wurde ein WooRank Audit durchgeführt. WooRank ist ein digitaler Servicedienstleister, welcher Suchmaschinen- und Marketing relevante Metriken untersucht (Harrington, 2018). Die gesamte Analyse kann im Anhang (vgl. Abbildung Anhang 12 bis Abbildung Anhang 25) betrachtet werden.

Die Bewertung durch WooRank zeigt, dass die App Suchmaschinenoptimiert ist. Die nichtfunktionale Anforderung der Auffindbarkeit der App ist durch das Audit bestätigt. Die Auffindbarkeit durch Suchmaschinen ist ein Vorteil, welcher schon in vorangegangenen Kapiteln beschrieben wurde. Dass dieser Vorteil auch tatsächlich Bestand hat, konnte durch die WooRank Analyse verifiziert werden.

Neben der Suchmaschinenoptimierung bewertet WooRank den Prototypen auch in Marketing relevanten Metriken positiv, so ist die App beispielsweise mobile optimiert (vgl. Abbildung Anhang 19).

7.3 Hosting

Für das Hosting der App wurde der Serverless Ansatz gewählt, da dieser weniger operativen Aufwand bedeutet, für kleine Projekte kostenlos, und automatisch skalierbar ist. Dabei übernimmt ein Cloud Dienstleister die gesamte Serverkonfiguration, wobei der Server aus kleinen Funktionen besteht, welche durch Events, wie beispielsweise ein http Request, ausgelöst werden. Dadurch können Rechenressourcen optimal genutzt, verteilt und skaliert werden (Rajan, 2018).

Vercel ist ein Cloud Dienstleister, der eine direkte Integration mit Github hat, wobei Änderungen am Repository unmittelbar einen build Prozess auslösen, und die Anwendung sofort gehostet werden kann. Neben Vercel gibt es weitere bekannte Cloud Dienstleister, wie Netlify oder Heroku, jedoch wurde Vercel schon in der Vergangenheit verwendet und wird deshalb in diesem Projekt wieder herangezogen.

Um ein GitHub Repository zu hosten sind nur wenige Schritte und keinerlei Konfiguration notwendig. Vercel erlaubt es, einen GitHub Account zu verknüpfen und direkt auf Repositories zuzugreifen, welche dann automatisiert gebuildet und gehostet werden. Die Waterplane App ist unter dem Link https://waterplane.vercel.app/ öffentlich verfügbar.

7.4 Benutzertest

Laut (Sommerville, 2018, p. 256) sind „Benutzer- oder Kundentests […] eine Phase im Testprozess, bei der Benutzer oder Kunden die Systemtests beratend begleiten und Eingaben bereitstellen."

Ein Benutzertest, welcher zum Ende der Entwicklung durchgeführt wird und dazu dient, festzustellen, ob das entwickelte System den Anforderungen entspricht, wird Abnahmetest genannt (Sommerville, 2018, p. 280)

Abnahmetests leiten sich von den Akzeptanzkriterien (vgl. Abschnitt 4.1) ab. Dabei sollten sowohl die funktionalen als auch die nicht funktionalen Anforderungen berücksichtigt werden (Sommerville, 2018, p. 281).

In diesem Test soll die nichtfunktionale Anforderung der App Ähnlichkeit verifiziert werden. Da es sich dabei vor allem um die subjektive Wahrnehmung des Benutzers handelt, ist ein Benutzertest die beste Möglichkeit, um herauszufinden, wie gut diese Anforderung umgesetzt werden konnte. Um die App Ähnlichkeit zu testen, wurden folgende Parameter identifiziert:

- Benutzerfreundlichkeit
- Look and Feel
- wahrgenommene Performanz

Im Zuge des Benutzertests sollen auch die funktionalen Anforderungen getestet werden. Folgende User Stories sind hier relevant (vgl. Tabelle 4):

Tabelle 4: Testrelevante Funtionale Anforderungen
Quelle: Eigene Darstellung

Name	*Beschreibung*	*Ref.*
User Story EP1 US1	Chronologische Auflistung der Events	4.3.1
User Story EP1 US2	Filter nach Tagen	4.3.1
User Story EP1 US3	Filtern nach Bühnen	4.3.1
Story EP2	Sprache wählen	4.3.2
Story EP3	Events favorisieren	4.3.3
Story EP4	App auf Homescreen installieren	4.3.4

Um die Parameter der App Ähnlichkeit zu testen, sowie die funktionalen Anforderungen aus Tabelle 4 zu berücksichtigen, wurden folgende Hypothesen aufgestellt:

- Nutzer können die App intuitiv zu ihrem Homescreen hinzufügen
- Nutzer können problemlos durch die App navigieren und eine gestellte Aufgabe lösen
- Nutzer erleben die App wie eine „herkömmliche App“

Um die Tests vergleichbar zu machen, werden Gerätetyp, Betriebssystem und Netzwerkverbindung kontrolliert.

Für die Testergebnisse wurden folgende Metriken festgehalten:

Objektiv: Messung der Zeit, die für die Durchführung einer zuvor festgelegten Aufgabe benötigt wird.

Subjektiv: Feedback in Form eines Ratings zur Nutzererfahrung

Anschließend sollten die Probanden ihre subjektiven Erfahrungen einordnen.

Um den Test durchzuführen, wurde eine Aufgabe konzipiert. Die Aufgabe ist im Anhang, Abbildung Anhang 26, zu finden. Um die Tests so gleich wie möglich abzuhalten, wurde diese den Testpersonen ohne weitere Erklärungen vorgelegt.

Die Ergebnisse des Tests sind in (vgl. Tabelle 5) zu finden. Die vollständigen Namen sind aus Datenschutzgründen nur den Prüfern zugänglich.

Name	Alter	Intuitive Bedienung	Ansprechendes Design	Gewohntes User Feeling	Filtermenü wurde gefunden und verwendet	App konnte problemlos zum Homescreen hinzugefügt werden
Silas	15	sehr	sehr	sehr	ja	ja
Tina	55	sehr	sehr	sehr	ja	ja
Mona	23	sehr	sehr	sehr	ja	ja
Jannik	21	sehr	sehr	sehr	ja	ja
Josh	22	sehr	sehr	sehr	ja	ja
Jule	26	sehr	sehr	sehr	ja	ja
Oliver	38	sehr	sehr	sehr	ja	ja

Tabelle 5: Ergebnisse Benutzertest
Quelle: Eigene Darstellung

8 Evaluation

Die Kerntechnologien von PWA wurden eingehend analysiert. Die Analyse hat dabei besonders die Funktion des Service Worker hervorgehoben, da dieser als Schlüsselelement der Technologie fungiert. Hierbei wurden alle wichtigen Eigenschaften, wie der Lebenszyklus und die asynchrone Funktionsweise theoretisch beleuchtet und anschließend mit Hilfe von Analysetools an einer minimalen Beispiel Applikation verifiziert. Weiter wurde besonders auf die Möglichkeit, Inhalte offline verfügbar zu machen, eingegangen. Das Prinzip von Precaching ist hier besonders wichtig, da PWAs dadurch heruntergeladen werden können und somit die Natur konventioneller Apps haben. Um dynamische Inhalte offline bereitstellen zu können wurde Runtime Caching vorgestellt und eingehend analysiert.

Um die praktischen Möglichkeiten von Progressive Web Apps zu erforschen, wurde PWA in ein modernes Web Framework implementiert. Es wurde eine Prototyp App für das fiktive Waterplane Festival umgesetzt.

Tabelle 6 zeigt, inwiefern die fachlichen Anforderungen umgesetzt werden konnten.

Tabelle 6: Evaluation der fachlichen Anforderungen
Quelle: Eigene Darstellung

Funktionale Anforderungen

Name	*Beschreibung*	*Ergebnis (verifiziert durch)*
User Story EP1 US1(4.3.1)	Chronologische Auflistung der Events	Ja (Inspektion, Benutzertest)
User Story EP1 US2 (4.3.1)	Filter nach Tagen	Ja (Inspektion, Benutzertest)
User Story EP1 US3 (4.3.1)	Filtern nach Bühnen	Ja (Inspektion, Benutzertest)
Story EP2 (4.3.2)	Sprache wählen	Ja (Inspektion, Benutzertest)
Story EP3 (4.3.3)	Events favorisieren	Ja (Inspektion, Benutzertest)
Story EP4 (4.3.4)	App auf Homescreen installieren	Ja (Inspektion, Benutzertest, Analysetools)

Nichtfunktionale Anforderungen

Beschreibung (ausführlich in Abschnitt (4.2)	*Ergebnis (verifiziert durch)*
App-Kosten minimieren	-
Auffindbarkeit durch Suchmaschinen	Ja (Analysetools)
Netzwerkunabhängigkeit	Ja (Inspektion, Analysetools)
HTTPS Verbindung	Ja (Analysetools)
Aktuell	Ja (Analysetools)
App ähnlich	Ja (Benutzertest)

Die Caching Strategie der App folgt dem App Shell Model, wobei die App Shell durch Precaching heruntergeladen und offline verfügbar gemacht wird. Der Inhalt der App wird mit Hilfe einer Runtime Caching Strategie gecacht, was dazu führt, dass bei offline Situationen die letzte heruntergeladene Version der Daten angezeigt wird und die Daten bei Netzwerkverbindung aktualisiert werden.

Für optimale Auffindbarkeit durch Suchmaschinen wurde Server Side Rendering implementiert, sowie die Best Practices von Google Lighthouse SEO und WooRank befolgt. Die Verwendung des Frameworks Nuxt erlaubt es, trotz serverseitigen Renderns, die UI clientseitig von JavaScript zu kontrollieren, was zu einer ansprechenden Nutzererfahrung führt. Dies wurde ebenfalls durch den Benutzertest verifiziert.

Das responsive Design wurde getestet und durch ein objektives Rating unter Verwendung von WooRank bestätigt. Die Designsprache ist an nativen Apps orientiert, was zu einer gewohnten Darstellung führt. Die App ist mit einem Installationsprompt versehen und ermutigt den Nutzer, die App zu installieren und im Standalone Modus zu verwenden. Dadurch verbessert sich die Nutzererfahrung weiter, da die App auf diese Weise im Vollbild Modus angezeigt wird (vgl. Abschnitt 4.4). Auch das responsive Design wurde durch den Benutzertest getestet, mit dem Ergebnis, dass das Design als „sehr ansprechend“, und das allgemeine Erscheinungsbild als „App ähnlich“ wahrgenommen wurde.

9 Fazit und Ausblick

In diesem Kapitel sollen die Ergebnisse der Arbeit zunächst im Abschnitt (9.1) final zusammengefasst werden. Anschließend sollen in Abschnitt (9.2) aktuelle Entwicklungen und mögliche Weiterentwicklungen besprochen werden.

9.1 Fazit

Zu Beginn dieser Arbeit wurden die wichtigsten Kerntechnologien von Progressive Web Apps (PWAs) eingehend untersucht und analysiert. Besonderes Augenmerk lag dabei auf allen Aspekten des Service Worker, da dieser als Schlüsselelement fungiert. Insbesondere der Lebenszyklus, die asynchrone Funktionsweise, sowie Caching Strategien wurden hier eingehend erklärt, analysiert und an einer minimalen Beispiel Applikation verifiziert.

Weiter wurde die Technologie in ein modernes Web Framework implementiert. Die Umsetzung der „Waterplane“ App hat gezeigt, dass Progressive Web App eine echte Alternative zu herkömmlichen Apps darstellt. Durch den Standalone Modus weist das Erscheinungsbild der App keine Unterschiede zum nativen Ansatz oder anderen Arten von Apps auf. Durch die JavaScript kontrollierte Benutzeroberfläche ist die App responsiv und bietet eine gute Benutzererfahrung. Dies wurde durch den Benutzertest verifiziert. Die Testpersonen haben die Nutzererfahrung durchweg als „gewohnt“ beschrieben, was zeigt, dass PWA ernsthafte Konkurrenz zu anderen App Technologien ist. Die App ist vom Desktop gleichermaßen ver-

wendbar wie von Mobilen Endgeräten. Außerdem sorgt die Suchmaschinenoptimierung für eine gute Auffindbarkeit der App durch ebensolche. Caching Strategien, welche durch den Service Worker umgesetzt werden, machen die App netzwerkunabhängig.

Die Inhalte der App werden durch das CMS Contentful bereitgestellt, was eine Benutzerfreundliche Datenverwaltung erlaubt, die selbst eine Person ohne technischen Hintergrund durchführen kann. Die Integration des CMS war problemlos über dessen REST API möglich.

Die Verwendung von Nuxt hat eine schnelle Entwicklung unterstützt, da viele Grundbausteine, wie beispielsweise das Routing bereits fertig integriert sind. Um das User Interface zu entwickeln, wurde die Bibliothek NaiveUI verwendet. Die Bibliothek stellt vorgefertigte UI Elemente zur Verfügung, die es erlauben in kurzer Zeit ein ansprechendes User Interface zu erstellen. Die Frontend Architektur der App ist an anderen Apps angelehnt und dafür konzipiert, erweiterbar zu sein. So wurde beispielsweise ein erweiterbares Menü implementiert und die Möglichkeit einen Footer anzusetzen eingeräumt.

Die Bereitstellung der App über Vercel erlaubt einen schnellen Bereitstellungsprozess. Updates werden automatisch ausgeliefert, ohne dass Nutzer weiteres unternehmen müssen. Die App kann problemlos weiterentwickelt und kontinuierlich bereitgestellt werden, was für eine etwaige Weiterentwicklung positiv ist.

9.2 Ausblick

In diesem Kapitel soll ein Ausblick über mögliche Verbesserungen des Prototyps sowie aktuelle Entwicklungen gegeben werden.

Eine Herausforderung, der sich die Waterplane App gegenübersieht, ist die noch nicht optimale Performanz, wie im Lighthouse Audit deutlich wird. Dies liegt vor

allem an den großen JavaScript Dateien, welche zu Beginn geladen werden. Die Performanz der App könnte durch sogenanntes Code Splitting weiter verbessert werden. Dabei werden Module dynamisch geladen, was zu einer Verringerung des JavaScript Codes führt, der bei Besuch der App geladen werden muss. Die Möglichkeit, asynchrone, dynamischer Importe in JavaScript umzusetzen, besteht seit 2020 (Turcotte, 2023). Eine weitere Möglichkeit, die Datenmenge zu verringern, sind Kompressionstools wie Brotli[21]. Dabei wird der JavaScript Code komprimiert, bevor dieser an den Client geschickt wird, wobei erhebliche Kompressionsraten erreichbar sind (Syed, 2018). Brotli wird außerdem von Vercel unterstützt, was die Integration erleichtern würde.

Ein Feature, das in dieser Arbeit nicht implementiert wurde, sind Push-Benachrichtigungen. PWAs können im Hintergrund Benachrichtigungen empfangen und die Nutzer zur Nutzung der App motivieren. Für die Waterplane App wären zeitabhängige Benachrichtigungen für bevorstehende, favorisierte Events eine sinnvolle Ergänzung. Allerdings wäre dies in Europa nur auf Android-Geräten möglich.

Die Beschränkung auf Android-Geräte in Europa resultiert aus den Auswirkungen des Digital Markets Act, einem Gesetz der Europäischen Union. Dieses Gesetz hat weitreichende Konsequenzen für die Unterstützung von PWAs, insbesondere auf iOS-Geräten. Die Ungewissheit über die Unterstützung von PWA auf iOS macht eine Verwendung der Technologie im Moment fragwürdig. Zu diesem Zeitpunkt ist es ungewiss, wie die Zukunft von Progressive Web Apps aussehen wird.

.

[21] https://github.com/google/brotli

‘

Literaturverzeichnis

Amazon Web Services, I. (2024). Retrieved 04 2024, from https://aws.amazon.com/devops/continuous-integration/#:~:text=Continuous%20integration%20is%20a%20DevOps,builds%20and%20tests%20are%20run.

Apple Inc. (2023). Retrieved 03 2024, from https://developer.apple.com/design/human-interface-guidelines/designing-for-ios#Best-practices

AppMySite. (2024). Retrieved 02 2024, from https://www.appmysite.com/blog/cms-market-share-top-trends-and-usage-statistics/

Arinir, D. (2023). *Mobile Computing.*

Baggott, A. (2023). Retrieved 02 2024, from https://appleinsider.com/articles/23/01/08/the-cost-of-doing-business-apples-app-store-fees-explained#:~:text=App%20Store%20fees,t%20apply%20to%20free%20apps.

Bitkom e.V. (2023). Retrieved 02 2024, from https://www.bitkom.org/Presse/Presseinformation/Deutscher-App-Markt-stabilisiert-uebertrifft-Vor-Corona-Niveau

Bosch, J. (2000). *Design and Use of Software Architectures.* Harlow, Addison-Wesley.

Brannon, J. (2024). Retrieved 04 2024, from https://coalitiontechnologies.com/blog/best-headless-cms-platforms-in-2024

BSS Commerce . (2022). Retrieved 03 2024, from https://bsscommerce.com/blog/best-pwa-framework-angular-vue-react/

Cabot, J. (2018). WordPress: A Content Management System to Democratize Publishing. *IEEE Software, 35*, 89-92.

Clements, R. (2024). Retrieved 04 2024, from https://ryanclements.dev/posts/fixing-nuxt-hydration-mismatches-in-the-real-world

Contentful. (2024). Retrieved 04 2024, from https://www.contentful.com/developers/docs/references/

Costa, R. (2024). Retrieved 04 2024, from https://www.justinmind.com/blog/slider-design-web/

Delcev, S. a. (2018). Modern JavaScript frameworks: A Survey Study. *2018 Zooming Innovation in Consumer Technologies Conference (ZINC)*, 106-109.

Demkovych, I. (2022). Retrieved 04 2024, from https://geniusee.com/single-blog/everything-you-need-to-know-about-git-flow-branch-model

Diniz-Junior, R. N.-J. (2022). Evaluating the performance of web rendering technologies based on JavaScript: Angular, React, and Vue. *2022 XVLIII Latin American Computer Conference (CLEI)*, 1-9.

Ebone, A. a. (2018). *A Performance Evaluation of Cross-Platform Mobile Application Development Approaches.*

Erdmann, E. (2023). Retrieved 04 2024, from https://raidboxes.io/en/blog/webdesign-development/headless-cms-wordpress/

Evan You, E. S. (o.J). Retrieved 04 2024, from https://router.vuejs.org/

Facebook, I. (2022). Retrieved 04 2024, from https://create-react-app.dev/docs/making-a-progressive-web-app/

Figma, I. (2024). Retrieved 03 2024, from https://www.figma.com/community/file/1035203688168086460/material-3-design-kit

Firtman, M. (2023). Retrieved 03 2024, from https://firt.dev/notes/pwa-ios/

Google LLC. (2024). Retrieved 04 2024, from https://angular.io/guide/service-worker-getting-started

Google LLC. (2024). Retrieved 03 2024, from https://developer.chrome.com/docs/workbox/modules/workbox-precaching

Google LLC. (2024). Retrieved 04 2024, from https://developer.chrome.com/docs/workbox/what-is-workbox

Google LLC. (2024). Retrieved 04 2024, from https://developer.chrome.com/docs/lighthouse/overview

Harrington, C. (2018). Retrieved 04 2024, from https://www.woorank.com/en/blog/what-is-woorank

Jelica Cincović, M. P. (2020). Comparison: Angular vs. React vs. Vue. Which framework is the best choice? 250-255.

Karwan Jacksi, S. M. (2019). Development History Of The World Wide Web. *INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH*.

Kitsios, S. A. (2015). *48th Hawaii International Conference on System Sciences.*

Kumar, A. a. (2015). Push notification as a business enhancement technique for e-commerce. *2015 Third International Conference on Image Information Processing (ICIIP)*, 450-454.

Lisboa, M. (2024). Retrieved 03 2024, from https://michaellisboa.com/blog/prompt-ios

Love, C. (2021). Retrieved 03 2024, from https://love2dev.com/blog/what-is-the-service-worker-cache-storage-limit/

McCollin, R. (2023, Januar). Retrieved 04 2024, from https://kinsta.com/blog/wordpress-rest-api/#:~:text=WordPress%20REST%20API-,What%20is%20the%20WordPress%20REST%20API%3F,create%20interactive%20websites%20and%20apps.

Meta Open Source. (2024). Retrieved 04 2024, from https://react.dev/learn/typescript

Morote, E. S. (o.J). Retrieved 04 2024, from https://pinia.vuejs.org/

Mozilla Foundation. (2023). Retrieved 02 2024, from https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers(9.2.24)

Nunkesser, R. (2023). *App-Entwicklung für Mobile und Desktop.* Berlin, Heidelberg.: Springer Vieweg.

Nuxt . (2024). Retrieved 04 2024, from https://nuxt.com/docs/getting-started/seo-meta

Ollila, R. a. (2022). Modern Web Frameworks: A Comparison of Rendering Performance. *Journal of Web Engineering*, 789-813.

Osmani, A. (2015). Retrieved 04 2024, from https://medium.com/google-developers/instant-loading-web-apps-with-an-application-shell-architecture-7c0c2f10c73

Pinto, C. M. (2018). *From Native to Cross-platform Hybrid Development.*

Prashant Ghimire, D. T. (2017). *HYBRID APP APPROACH: COULD IT MARK THE END OF NATIVE APP DOMINATION?* IISIT.org.

Que, P. a. (2016). *A Comprehensive Comparison between Hybrid and Native App Paradigms.*

Raj, C. R., & Tolety, S. B. (2012). A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. *2012 Annual IEEE India Conference (INDICON) IEEE*, 625–629.

Rajan, R. A. (2018). *Serverless Architecture - A Revolution in Cloud Computing.*

Rohan Anand Choudhary, T. M. (2020). *A Comparative Analysis of Native and Hybrid Mobile Application Development .* Juni Khyat. Retrieved from http://www.junikhyatjournal.in/no_10_may_20/9.pdf

Russel, A. (2015). Retrieved 02 2024, from https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/

Sayali Sunil Tandel, A. J. (2018). Impact of Progressive Web Apps on Web App Development. *InternatiInternational Journal of Innovative Research in Science, Engineering and Technology*, 9439-9444.

Sevim, M. (2023). Retrieved 04 2024, from https://medium.com/commencis/angular-vs-react-vs-vue-which-is-the-better-1b6c88b3825c

Shah, V. (2024). Retrieved 04 2024, from https://www.tatvasoft.com/blog/angular-vs-react-vs-vue/

Sommerville, I. (2018). *Software Engineering.*

Speckner, C. (2024, März). Retrieved 03 2024, from https://blog.mayflower.de/17231-ios-17-4-pwa.html

Stack Exchange Inc . (2024). Retrieved 04 2024, from https://insights.stackoverflow.com/trends?tags=reactjs%2Cvue.js%2Cangular%2Csvelte%2Cangularjs%2Cvuejs3

Statista. (2023). Retrieved 03 2024, from https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/

Steiner, T. (2018). What is in a Web View: An Analysis of Progressive Web App Features When the Means of Web Access is not a Web Browser. 789–796.

Storyblok GmbH . (2024). Retrieved 04 2024, from https://www.storyblok.com/pricing.

Syed, Z. a. (2018). Compression Algorithms:Brotli, Gzip and Zopfli Perspective. *IJST*, 1–4.

Tim A. Majchrzak, A. B.-H.-M. (2018). *Progressive Web Apps: the Definite Approach to Cross-Platform Development?*

Turcotte, A. a. (2023). Increasing the Responsiveness of Web Applications by Introducing Lazy Loading. *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 459-470.

You, E., & Contributors, V. (2024). Retrieved 03 2024, from https://vitejs.dev/guide/

Anhang

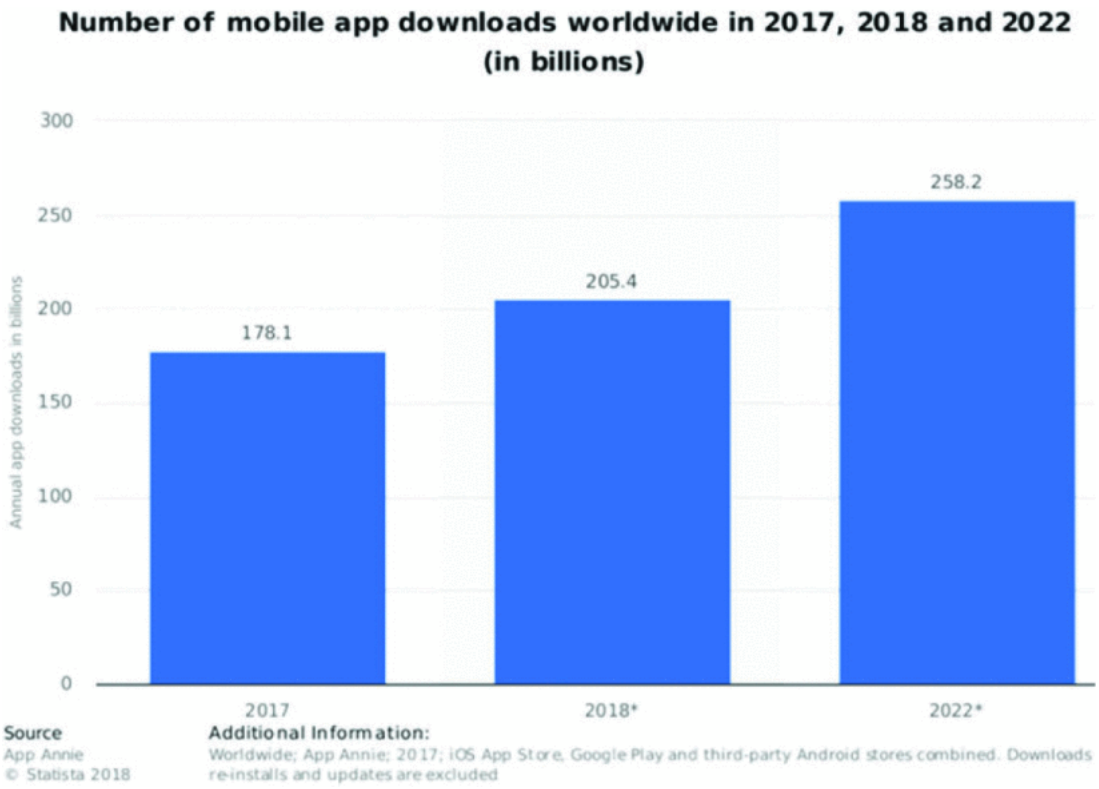


Abbildung Anhang 1: App Downloads weltweit
Quelle: (Pinto, 2018)

```
1
2    └── Basic PWA
3        └── Basic PWA.code-workspace
4        └── index.html
5        └── sw.js
6
```

Abbildung Anhang 2: Folder Structure „BasicPWA“
Quelle: Screenshot VSCode

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    Basic PWA
    <script>
      if ("serviceWorker" in navigator) {
        navigator.serviceWorker.register("/sw.js");
      }

      document.addEventListener("DOMContentLoaded", function () {
        caches.open("cacheStorageTest").then(function (cache) {
          cache.put(
            "RequestURL",
            new Response(
              "<html><body>Dieser String ist im Cache gespeichert!</body></html>",
              { headers: { "Content-Type": "text/html" } }
            )
          );
        });
      });

      // log when page has completely loaded
      window.onload = function () {
        console.log("Seite hat vollständig geladen");
      };
    </script>
  </body>
</html>
```

Abbildung Anhang 3: Codelisting index.html von BasicPWA
Quelle: Eigene Darstellung

```
{
  "name": "Basic PWA",
  "start_url": "/",
  "icons": [
    {
      "src": "icons/icon-192x192.png",
      "sizes": "192x192"
    },
    {
      "src": "icons/icon-256x256.png",
      "sizes": "256x256"
    },
    {
      "src": "icons/icon-384x384.png",
      "sizes": "384x384"
    },
    {
      "src": "icons/icon-512x512.png",
      "sizes": "512x512"
    }
  ],
  "display": "fullscreen"
}
```

Abbildung Anhang 4: Code Listing App Manifest
Quelle: Eigene Darstellung

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <link rel="manifest" href="manifest.json" />
  </head>
  <body>
    Basic PWA
  </body>
</html>
```

Abbildung Anhang 5: index.html
Quelle: Screenshot VSCode

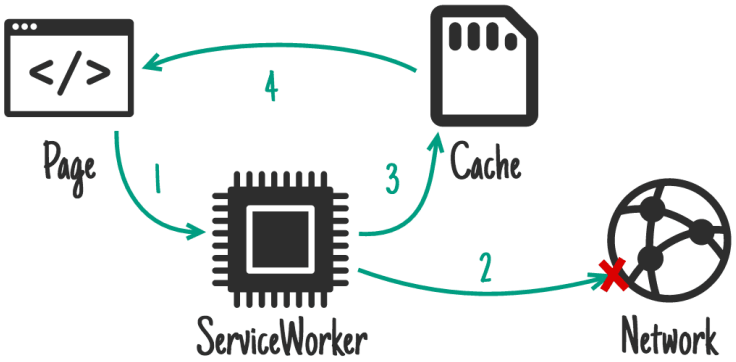


Abbildung Anhang 6: Network First Caching Strategy
Quelle: https://developer.chrome.com/docs/workbox/modules/workbox-strategies

Headers Preview

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <link rel="manifest" href="manifest.json" />
  </head>
```

Abbildung Anhang 7: Cache Storage Element Preview
Quelle: Screenshot Chrome DevTools

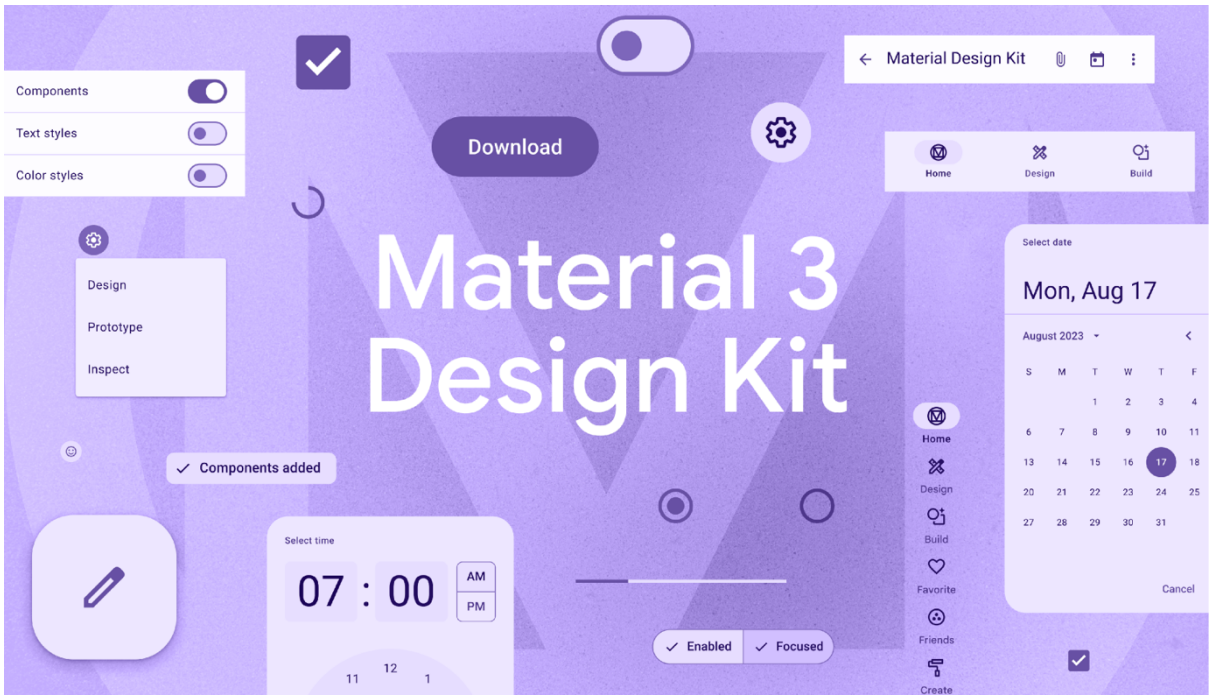


Abbildung Anhang 8: Material3 Design Kit
Quelle: (Figma, 2024)

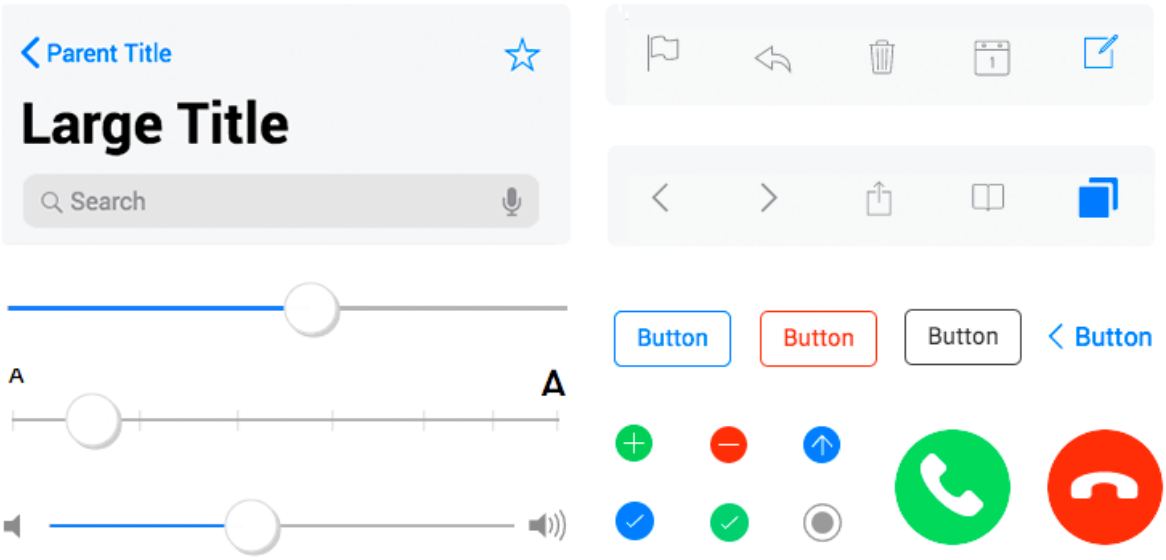


Abbildung Anhang 9: iOS UI Kit
Quelle: (Costa, 2024)

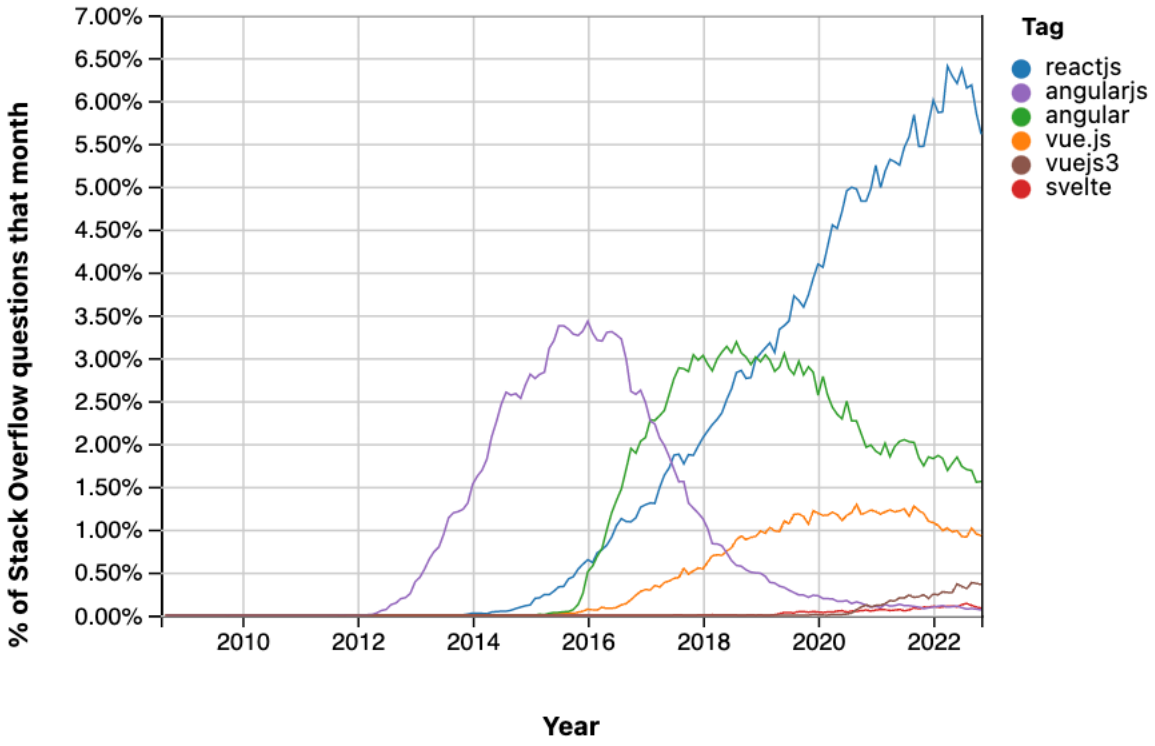


Abbildung Anhang 10: Frontendtechnologien auf Stack Exchange
Quelle: (Stack Exchange Inc , 2024)

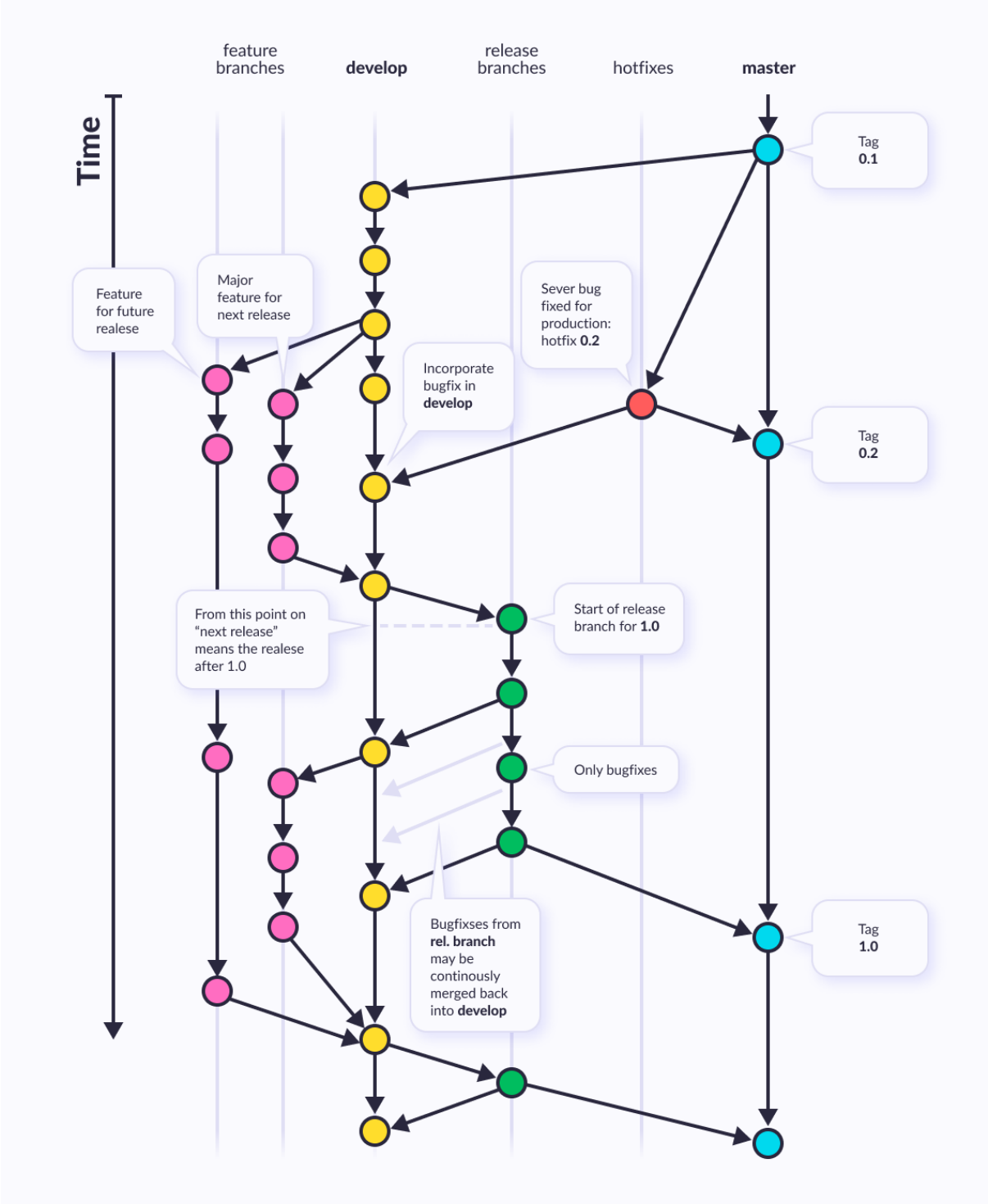


Abbildung Anhang 11: Git Flow Model
Qualle: https://nvie.com/posts/a-successful-git-branching-model/

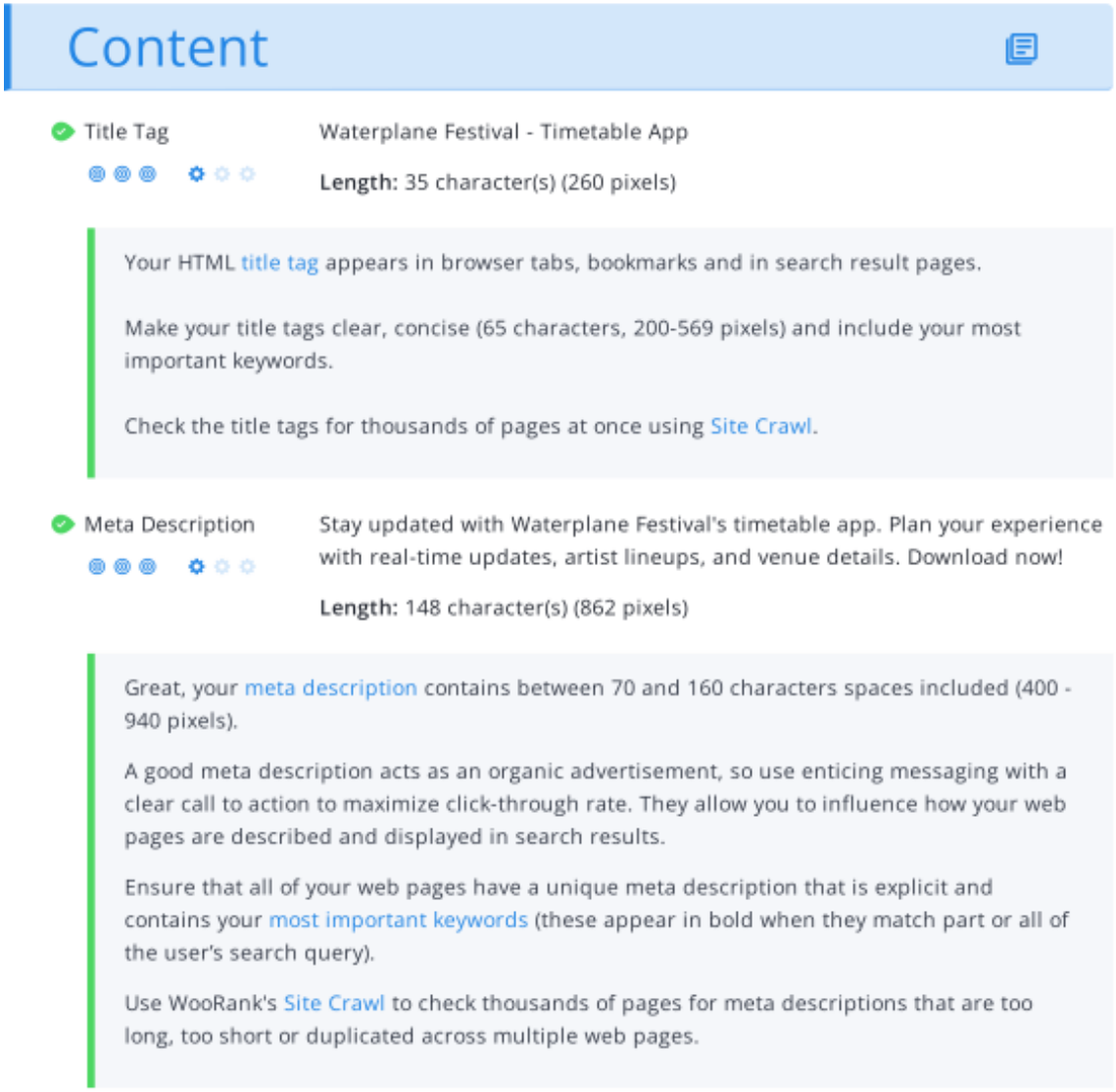


Abbildung Anhang 12: WooRank Audit (1/14)
Quelle: WooRank

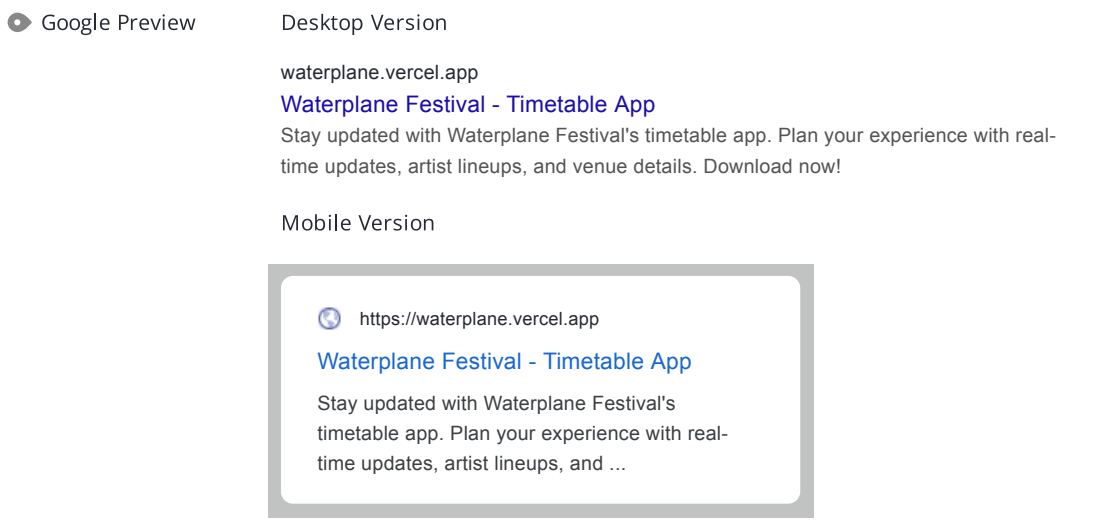


Abbildung Anhang 13:WooRank Audit (2/14)
Quelle: WooRank

Content Analysis Timetable App 2 Waterplane Festival 2

> This data represents the words and phrases that your page appears to be optimized around. We use what's called "natural language processing" (NLP), which is a form of artificial intelligence that allows computers to read human language, to do this analysis.
>
> The numbers next to each word or phrase represents how often we detected them and their variants on the page.
>
> Are these the keywords you want to target for your page? If so, great! Track your site's rankings in Google search results using WooRank's Keyword Tool.
>
> If these keywords aren't relevant to your page, consider updating your content to optimize it for your target keywords.

Alt Attribute No images with missing alt attributes were found.

> Alternative text allows you to add a description to an image. Since search engine crawlers cannot see images, they rely on alternative text attributes to determine relevance to a search query. Alternative text also helps makes an image more likely to appear in a Google image search and is used by screen readers to provide context for visually impaired users.
>
> It looks like most or all of your images have alternative text. Check the images on your website to make sure accurate and relevant alternative text is specified for each image on the page.
>
> Try to keep your alternative text to a simple, one-sentence description of what's in the image.

Abbildung Anhang 14: WooRank Audit (3/14)
Quelle: WooRank

In-Page Links — We found a total of 0 link(s) including 0 link(s) to files

Links pass value from one page to another. This value is called 'link juice'.

A page's link juice is split between all the links on that page so lots of unnecessary links on a page will dilute the value attributed to each link. There's no exact number of links to include on a page but best practice is to keep it under 200.

Using the Nofollow attribute in your links prevents some link juice, but these links are still taken into account when calculating the value that is passed through each link, so using lots of NoFollow links can still dilute PageRank.

Check your site's internal linking using Site Crawl.

Language — **Declared:** *English*

Detected: *English*

Great, your declared language matches the language detected on the page.

Make sure your declared language is the same as the language detected by Google.

Tips for multilingual websites:

- Define the language of the content in each page's HTML code.
- Specify the language code in the URL as well (e.g., "mywebsite.com/fr/mycontent.html").
- Use hreflang tags to specify language and country for Google, and the "content-language" meta tag to specify the language and country for Bing.

Abbildung Anhang 15: WooRank Audit (4/14)
Quelle: WooRank

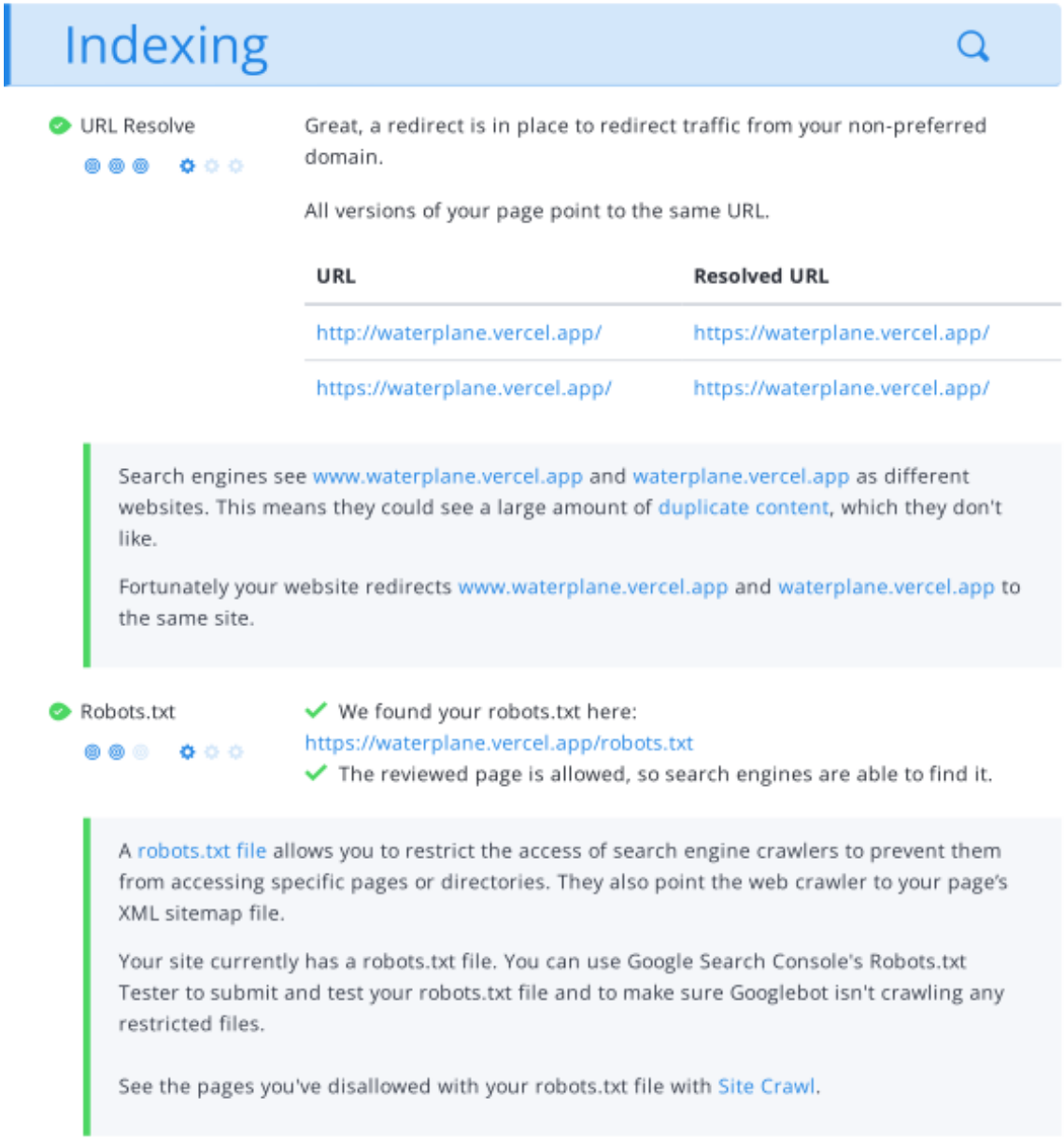


Abbildung Anhang 16: WooRank Audit (5/14)
Quelle: WooRank

Canonical Tags — We didn't find any canonical URLs on your page. It is, however, strongly recommended to add a self-referencing canonical tag when possible.

A canonical tag, also called "rel canonical" is an HTML tag that tells search engines that the enclosed URL is the original, definitive version of the page. Practically speaking, it tells Google which page you want to appear in search results. Search engines see different URLs as different pages, even if they are serving the same purpose. Having multiple versions of a page might cause you to suffer issues with duplicate content.

Implementing a canonical tag can be done through the <link> tag in the <head> or through HTTP headers.

To learn more about best practices and how to implement your canonicals correctly, check out our complete canonical tag guide. Also, head to WooRank's Site Crawl to discover any duplicate content issues on your site.

Robots Tags — We did not find any robots meta tags in your website.

The robots meta tag gives you control over the content search engines are able to index and display to users. Discover all of the ways you can use robots meta tags here.
Although not necessary to have on each page, it is highly recommended to include robots tags so you can control how your content is delivered.

You can implement your robots tag either as an HTML tag in the head element of your page, or by using an X-Robots-Tag in the HTTP Header response.
You can use both approaches to specify instructions to one particular search engine, or any other search engine visiting and crawling your site.

Index and Follow — This page is set to 'index' and 'follow'

Noindex and nofollow are HTML meta tags that guide search engine crawlers. A 'noindex' tag excludes a page from search results, while 'nofollow' prevents crawlers from following links on a page, halting the transfer of authority to linked pages. These tags help manage visibility and avoid outdated or irrelevant content from being served to potential visitors who might get little value out of it.

Abbildung Anhang 17: WooRank Audit (6/14)
Quelle: WooRank

Hreflang Tags — No hreflang tags were found on this page

> The hreflang tag is an HTML tag that tells search engines which languages and (optionally) countries a page's content is relevant for. Hreflang tags also tell search engines where to find the relevant content in alternate languages.
>
> If your website targets users all around the world, using hreflang tags will help make sure the right content is being served to the right users.
>
> The value of the hreflang attribute identifies the language (in ISO 639-1 format) and optionally a region in ISO 3166-1 Alpha 2 format of an alternate URL.
>
> Use WooRank's Site Crawl to perform a thorough check on hreflang validity across a website.

Broken links — No broken links were found on this web page

> Broken links send users to non-existent web pages. They hurt a site's usability and reputation, which impacts SEO.
>
> Fortunately your page doesn't contain any broken links.
>
> Be proactive in checking your pages to make sure they don't have any broken links.
>
> See the HTTP status for potentially thousands of links using Site Crawl.

Underscores in the URLs — Great, you are not using underscores (these_are_underscores) in your URLs.

> Great, you aren't using underscores (these_are_underscores) in your URLs.
>
> Google sees hyphens as word separators while underscores aren't recognized. So the search engine sees www.example.com/green_dress as www.example.com/greendress. The bots will have a hard time determining this URL's relevance to a keyword.

Abbildung Anhang 18: WooRank Audit (7/14)
Quelle: WooRank

Abbildung Anhang 19: WooRank Audit (8/14)
Quelle: WooRank

Font Size Legibility — Perfect, this web page's text is legible on mobile devices.

> At least 60% of your page's font size is 12 pixels or greater.

Mobile Viewport
- Great, a configured viewport is present.
- The content fits within the specified viewport size.

> Great, the viewport is well configured.
>
> Keep in mind that since the width (in CSS pixels) of the viewport may vary, your page content should not solely rely on a particular viewport width to render well. Consider these additional tips:
> - Avoid setting large absolute CSS widths for page elements.
> - If necessary, CSS media queries can be used to apply different styling depending on screen size.
> - Ideally, serve responsively-sized images.

Mobile Frameworks — No mobile frameworks have been detected.

> Mobile or responsive frameworks are an important part of website optimization as they assist developers in creating applications which are applicable to multiple devices.

AMP — We didn't find AMP on your page.

> AMP is an open-source library that provides a straightforward way to create web pages that are compelling, smooth, and load near instantaneously for users.
>
> Check your AMP markup with the AMP validator.

Abbildung Anhang 20: WooRank Audit (9/14)
Quelle: WooRank

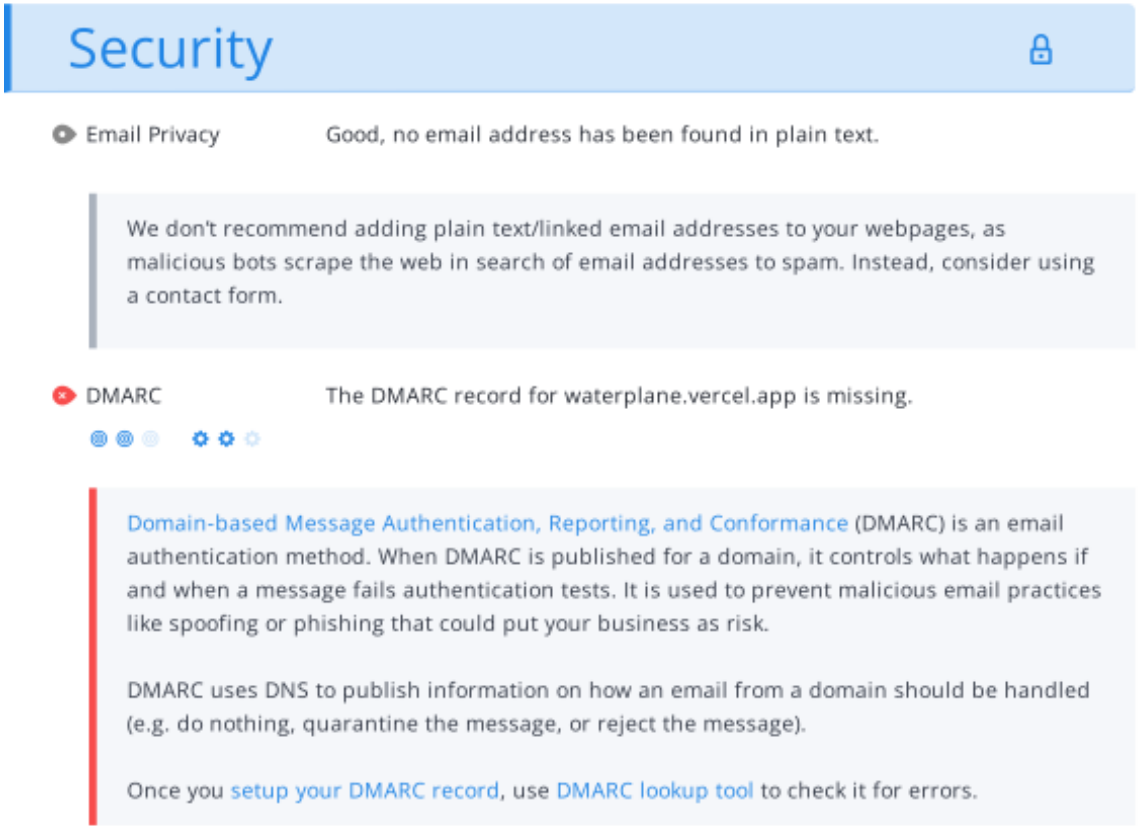


Abbildung Anhang 21: WooRank Audit (10/14)
Quelle: WooRank

SSL Secure

Great, your website is SSL secured (HTTPS).

- Your website's URLs redirect to HTTPS pages.
- Your website is configured with HSTS.
- The SSL certificate expires in 3 months.
- The certificate issuer is Let's Encrypt.

Modern websites tend to be SSL secured (HTTPS) as it provides an extra security layer while logging in to your Web Service. In 2014, Google announced that an HTTPS (vs HTTP) website would receive an extra boost in their ranking.

While switching to HTTPS, make sure your site remains optimized and see to it that your website will still run quickly. Follow these best practices for a smooth transition:

- Use a serious issuer to purchase your SSL certificate
- Redirect all of your HTTP pages to the HTTPS version of your website
- Use HTTP Strict Transport Security (HSTS) in your headers
- Renew your SSL certificate every year, before it expires
- Make sure that all of your content (CSS, etc.) is linked to HTTPS
- Update your XML sitemap to ensure the URLs include HTTPS and update the robots.txt file to reference this version
- Register the HTTPS website in Google & Bing Search Console/Webmaster Tools

Mixed Content

We didn't find any mixed content on this web page.

Great, your website is secure and does not contain mixed content types.

Mixed content occurs when a URL is loaded over a secure HTTPS protocol, but other resources on the page (such as images, videos, stylesheets, scripts, etc.) are loaded over an insecure HTTP connection.

Modern browsers may block this content, or may display warnings to the user that this page contains insecure resources which causes them not to view your page.

Check out these techniques to prevent mixed content on your site.

Abbildung Anhang 22: WooRank Audit (11/14)
Quelle: WooRank

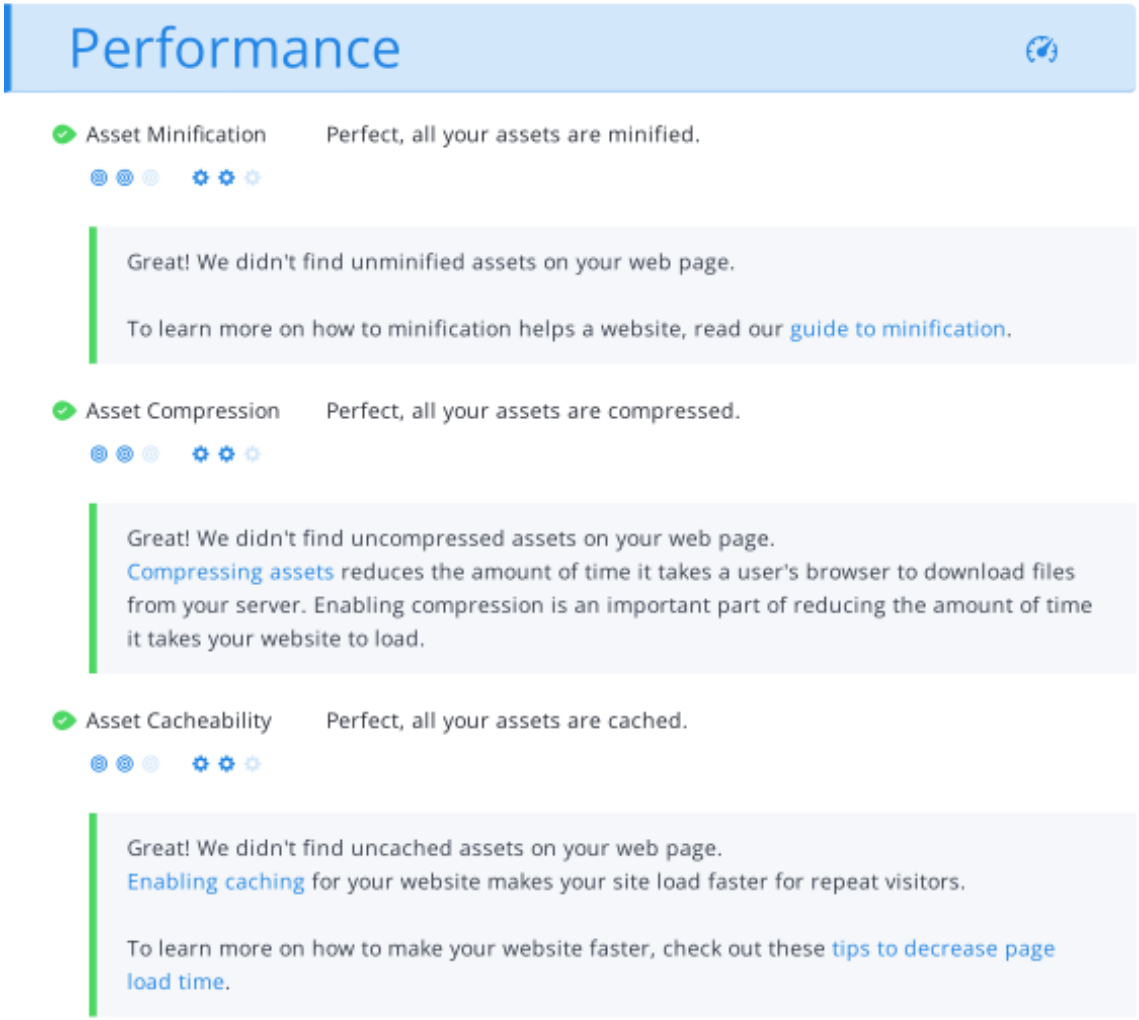


Abbildung Anhang 23: WooRank Audit (12/14)
Quelle: WooRank

Image Optimization

Great! This page passed all 5 checks

- All images are efficiently encoded
- All images are using next-gen formatting
- All images are properly sized
- All offscreen images have been deferred
- All images have explicit 'width' and 'height' dimensions

This page appears to have passed all 5 checks for image optimization.

Optimizing your images can yield the greatest performance improvements and impress your website visitors. Lighthouse helps to identify issues that could frustrate users when the page loads, like images without dimensions. By not specifying an image's width and height, you could be causing a layout shift on your page.

Website visitors will benefit from properly sized images and efficiently encoded images. If there are any images that load offscreen or are hidden in the page, it would be better to defer those images to speed up your load time.

Finally, it's important to serve next-gen image formats, which have better compression and quality than their older JPEG and PNG counterparts. The images will load faster, consume less data and provide an overall better user experience.

Layout Shift Elements

The table below shows the DOM elements that contribute the most to the CLS of the page.
Your total CLS score is 0.374.

Element	CLS contribution
DE EN *<div class="n-layout n-layout--static-positioned" style="--n-bezier:cubic-bezier(.4, 0, .2, 1);--n-color:rgb(16, 16, 20);--n-t...*	0.374

Cumulative Layout Shift (CLS) is used to measure the user-friendliness of a page. Specifically, it measures how much the content on a page moves around as the page loads.

CLS is measured any time an element that's visible in the viewport changes its position on the screen between two frames as the page loads. So, if a button moves from the left to the right, or if it moves two or three lines down, a CLS is recorded. This is bad user experience, as your visitors can get frustrated when they try to click something and it moves positions.

Each shifted element on your page contributes to your CLS score. To learn more about how CLS is calculated and some of the common causes, check out our Cumulative Layout Shift guide.

Abbildung Anhang 24: WooRank Audit (13/14)
Quelle: WooRank

Abbildung Anhang 25: WooRank Audit (14/14)
Quelle: WooRank

Waterplane
Festival App
Probandentest

Name _____
Alter __

Aufgabe

- Nimm dein Smartphone, öffne den Browser und navigiere zu https://waterplane.vercel.app
- Wähle deine Sprache und folge der Anleitung, um die App auf deinem Homescreen zu installieren.
- Öffne die App auf deinem Homescreen und finde den Act, welcher Samstagabend um 18:00 auf der Stadtparkbühne spielt. Nutze dazu am besten die Filteroption (oben rechts)
- Füge den Act zu deiner Favoritenliste hinzu.

	sehr	mittel	weniger
Die Bediehnung ist intuitiv			
Das Design ist ansprechend			
Die App hat sich gewohnt angefühlt			

	ja	nein
Ich habe das Filtermenü gefunden und verwendet		
Ich habe das Share Menü sofort gefunden und konnte die App schnell zum Homescreen hinzufügen		

Abbildung Anhang 26: Benutzertest Aufgabe
Quelle: Eigene Darstellung

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit mit dem Titel:

_____

selbständig und nur mit den angegebenen Hilfsmitteln verfasst habe. Alle Passagen, die ich wörtlich aus der Literatur oder aus anderen Quellen wie z. B. Internetseiten übernommen habe, habe ich deutlich als Zitat mit Angabe der Quelle kenntlich gemacht.

3.5.2024

Datum

Unterschrift