DHBW
Duale Hochschule
Baden-Württemberg

Gegenüberstellung der Entwicklung nativer Mobilanwendungen und Progressive Web Apps (PWAs)

Studienarbeit (T3101)

für den Studiengang
Informatik

an der
Dualen Hochschule
Baden-Württemberg
Stuttgart

von
Stefan Goldschmidt
Oliver Rudzinski

Abgabedatum	08. Juni 2020
Matrikelnummer (Goldschmidt)	9760520
Matrikelnummer (Rudzinski)	5481330
Kurs	TINF17A
Hochschulbetreuer	Arne Heimeshoff
Studiengangsleitung	Prof. Dr. Dirk Reichardt
	Prof. Dr. Carmen Winter

Erklärung

Wir versichern hiermit, dass wir unsere Studienarbeit mit dem Thema:

Gegenüberstellung der Entwicklung
nativer Mobilanwendungen und
Progressive Web Apps (PWAs)

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben.

_____, _____		_____
Ort	Datum	Unterschrift **Stefan Goldschmidt**
_____, _____		_____
Ort	Datum	Unterschrift **Oliver Rudzinski**

Zusammenfassung

Das ist die Zusammenfassung auf Deutsch.

Abstract

This is the abstract in English.

Inhaltsverzeichnis

Abkürzungsverzeichnis

CLI	Command Line Interface
CRUD	Create, Read, Update und Delete
PWA	Progressive Web App
UI	User Interface

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

1 Einleitung

1.1 Marktüberblick

Marktanteile / Nutzung von Mobilgeräten

Für das Jahr 2020 wird erwartet, dass 45% der Weltbevölkerung ein Smartphone nutzt. [1] [2] Sowohl Unternehmen, die ihre Produkte überwiegend offline vertreiben, als auch die führenden IT-Unternehmen wissen um diesen Trend, denn App-Nutzer sind (potenzielle) Kunden. Auch die Medienbranche verdient mit App-Nutzern Geld, da mit jedem Nutzer ihrer Website oder App die Werbeeinnahmen steigen. Newsportale wie Focus Online, BILD, Welt.de oder Spiegel Online gehören 2019 zu den verbreitetsten mobilen Webseiten [3]. Diese Webseiten sind bereits für Mobilegeräte optimiert. Für Entwickler liegt es nahe, die Webseite in einer Mobilanwendung einzubetten, anstatt alle Features in einer nativen Anwendung neu zu entwickeln. Der Nutzer kann diese Anwendungen über einen App Marktplatz beziehen. Mit Bannern und Links versuchen Unternehmen auf ihre Mobilanwendungen aufmerksam zu machen. Mit dem Konzept der Progressive Web App (PWA) könnte dieser Schritt bald obsolet werden, wenn Nutzer nicht mehr eine plattformabhängige App, sondern vielmehr die Webseite selbst als "Anwendung" installieren. Bezieht man die vergleichsweise hohen Entwicklungskosten mobiler Anwendungen in diese Rechnung mit ein, besteht eine große Chance, dass die Erweiterung der bestehenden Webseite zur PWA schneller, wartungärmer und insgesamt günstiger ist.

Die Progressive Web App verspricht nicht nur einen einfachen Container um eine Website, sondern auch Offline-Funktionalität, vergleichsweise einfache Entwicklung mit JavaScript und Unabhängigkeit der Plattform. Damit löst sie viele bekannte Probleme einfacher Container Apps um eine Webseite. Nicht selten wird dort die Geduld der Nutzer herausgefordert, da alle Inhalte (Scripte, Stylesheets, Schriftarten, HTML etc.) ständig über eine möglicherweise langsame Datenverbindung heruntergeladen werden müssen.

Der Hohe Marktanteil (60% in 2019) des mobilen Browsers Google Chrome (welcher die Installation von PWAs voll unterstützt [4, S. 8]) und Apples Safari für iOS (20% in

2019), welcher die Unterstützung der PWA stetig erweitert, inspirieren diese Arbeit, die Möglichkeiten der PWA im Vergleich zur nativen App ausführlich zu vergleichen. [5]

1.2 Motivation dieser Arbeit

Forschungsfrage: kann PWA native App langfristig ablösen

Mobilegeräte Leistungsstärker Möglichkeiten plattformübergreifend zu entwickeln

Nutzer Conversion kann langfristig gesteigert werden –> Profit!
https://developers.google.com/web/progressive-web-apps [6]

1.3 Kapitelübersicht

In diesem Kapitel (1) wurden die aktuellen Marktenwicklungen kurz mit Zahlen benannt und die Motivation dieser Arbeit dargelegt. Das folgende Kapitel (2 Grundlagen) legt vorwiegend technischen Grundlagen für die spätere Implementierung einer Anwendung als PWA und nativer App. Dabei wird auf die verwendeten Technologien und Frameworks eingegangen und speziell die PWA auf technischer Ebene erklärt.

Kapitel 3 (Architektur) erläutert die Architektur der entwickelten Anwendung: eine Todoliste. In diesem Kapitel werden detaillierte Spezifikationen beschrieben, die den Entwicklungsprozess erst vergleichbar machen. Außerdem wird auf architekturbezogene Entscheidungen der Plattformen eingegangen, beispielsweise die Gründe für die Wahl der Frameworks der PWA.

Um den Entwicklungsprozess vergleichen zu können, wird in Kapitel 4 (Wissenschaftliches Framework) der Vergleichsprozess beschrieben und Kriterien mit ihrer Gewichtung aufgestellt und erläutert. Anschließend werden im zweigeteilten Kapitel 5 (Implementierung) die Implementierungsprozesse der PWA und der nativen App detailliert dokumentiert.

Nach dem Sammeln von Erfahrungen bei den Implementierungen werden in Kapitel 6 (Evaluation) beide Technologien mithilfe der in Kapitel **??** erstellten Kriterien verglichen und evaluiert.

Abschließend gibt Kapitel 7 (Reflexion) ein Urteil über den Erfolg dieser Arbeit und gibt einen Ausblick auf die Zukunft der PWA.

2 Grundlagen

2.1 Progressive Web App

Der Software Entwickler und Author Majid Hajian charakterisiert PWAs mit 8 Eigenschaften. Die wichtigsten dieser Charakteristika werden im Folgenden zusammenfassend erläutert:

Installierbarkeit Die Anwendung muss installierbar sein und wie eine native App vom Startbildschirm gestartet werden können.

Ähnlichkeit mit nativer App Die PWA soll, wie eine native App, auf die Hardware des Mobilgeräts zugreifen können (beispielsweise die Nutzung Bluetooth-Chips). Außerdem unterscheidet sich das User Interface der PWA nicht zur nativen App.

Offline-Verwendung Die PWA soll unabhängig von Netzwerkverbindung sein. Sie ist nach dem öffline-first design"konzipiert. Die Google-Chrome Dokumentation für Cloud-Entwickler beschreibt Offline First Apps als Webanwendung, deren Dateien (JavaScript, HTML, CSS etc.) bereits heruntergeladen sind. Daten werden temporär über eine Browser-Schnittstelle gespeichert und bei Bedarf synchronisiert. Außerdem kann die Anwendung auf eine unterbrochene Netzwerkverbindungen reagieren [7]. Die PWA ist demnach eine Webanwendung, die sowohl online, als auch offline nutzbar ist.

Mobiloptimiert Die PWA ist für die (meist leistungsschwache) Mobilhardware konzipiert und funktioniert hierauf ohne Performanceprobleme. Hajian legt besonders auf das schnelle Laden beim Start der Anwendung wert.

Informierung des Nutzers Wie native Apps, kann die PWA den Nutzer über Push-Nachrichten informieren oder zu Interaktion auffordern.

[8, S. 1f.]

Diese Charakteristiken decken sich mit der Beschreibung durch die Entwickler-Dokumentation der PWA von Google. Im Vergleich zu Hijian ist diese etwas spezifischer und erwähnt

beispielsweise die Kontrolle des Anwendungscaches durch einen JavaScript Service Worker, um die Abhängigkeit von einer Netzwerkverbindung aufzuheben. [6]

2.1.1 Installation einer PWA

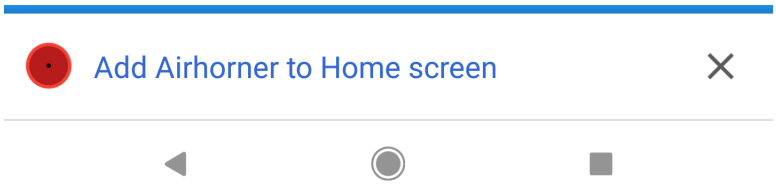

Abbildung 2.1: Browserdialog zur Installation einer PWA [9]

Die Aufforderung zur Installation einer PWA kann entweder über den Browser (siehe Abbildung 2.1) erfolgen oder über ein Element der Website, dass ein Event erzeugt, wie beispielsweise ein Button oder ein Dialog.

Für Mobilgeräte generiert der Browser dann eine WebAPK, welche auf dem Gerät installiert wird. Auf Desktopgeräte startet die PWA in einem verschlankten Browserfenster ohne Suchleiste und Bedienelemente. [10]

2.1.2 Manifest Datei

Um die PWA auf einem Gerät installieren zu können, muss es eine web-app-manifest Datei zur Verfügung gestellt werden. Diese ist ein JSON file, welche als Konfigurationsdatei der installierten Anwendung dient. [11]

```
{
  "short_name": "Maps",
  "name": "Google Maps",
  "icons": [
    {
      "src": "/images/icons-192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
    {
      "src": "/images/icons-512.png",
      "type": "image/png",
      "sizes": "512x512"
    }
  ],
  "start_url": "/maps/?source=pwa",
  "background_color": "#3367D6",
  "display": "standalone",
  "scope": "/maps/",
  "theme_color": "#3367D6"
}
```

Quellcode-Ausschnitt 1: Manifestdatei einer PWA

Abbildung 1 zeigt den Inhalt einer Manifest-Datei. Neben diversen Icons werden auch Name, Farbschema und Anzeigeeinstellungen festgelegt. Die Manifest-Datei wird im HTML der Webanwendung eingebunden, siehe Abbildung 2. Es ist die Einfachheit dieses Prozesses hervorzuheben: Das Hinzufügen einer (wenige Zeilen langer) JSON-Datei macht die gesamte Webanwendung für den Nutzer installierbar. Es wird kein App-Store, nutzerverwalteter Dateidownload oder Installer benötigt.

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

Quellcode-Ausschnitt 2: Einbinden der Manifestdatei

2.1.3 Unterstütze Plattformen von PWA

Apples Mobilbetriebssystem iOS unterstützt einige Features der PWA noch nicht. Die Versionen zeigen aber eine stetig voranschreitende Integration der , erweitert den Funktionsumfang in den neusten Versionen von iOS

Die Nutzung von PWAs ist nicht ausschließlich auf Smartphones begrenzt. Wie normale Desktopprogramme können Desktop PWAs in einem eigenen Fenster gestartet

werden. Der normale Nutzer erkennt auch hier den Unterschied zwischen einer nativen Desktopanwendung und einer Desktop PWA nicht. Stark vereinfacht beschrieben, sind Desktop PWAs Browserfenster ohne Tabs und Adressleiste. Durch die Nutzung von Service Workern, welche die Webanwendung cachen, sind auch Desktop PWAs nicht zwangläufig an eine Netzwerkverbindung gebunden.

Grundsätzlich können Desktop PWAs auf jedem Betriebssystem installiert werden, auf dem Google Chrome (Version größer 73) installiert werden kann: Windows, Mac, Linux und Chrome OS. [12]

2.1.4 Service Worker

Damit die PWA trotz fehlender Netzwerkverbindung funktioniert wird ein besonderer Mechanismus benötigt: der Service Worker. Mit ihm können Abhängigkeiten der App lokal gecached werden, so dass die Anwendung auch bei schlechter oder gar fehlender Netzwerkverbindung funktioniert. [4, S. 7]

Ein Service Worker ist ein von der UI seperiertat laufendes Hintergrundscript der Webanwendung, siehe Abbildung 2.2. Er wird genutzt um Bilder, Scripte, Styles oder ganze Seiten zu cachen. Bei bestehender Netzwerkverbindung führt er nötige Synchronisierungen durch. Nicht zuletzt ist er auch für das senden von Push-Notifications zuständig. [4, S. 24]

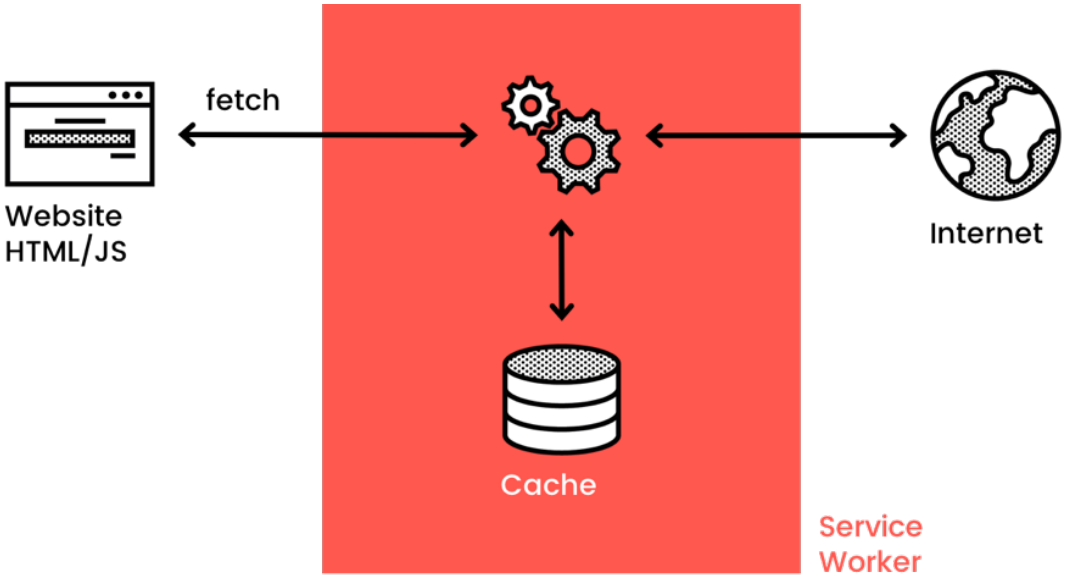

Abbildung 2.2: Konzept des Service Workers [13]

Alle verbreiteten Desktopbrowser wie Chrome, Firefox, Opera, Edge und mitterweile auch Safari unsterstützen das Service Worker Konzept. Der Mobile Chromebrowser

unter Android unterstützt Service Worker bereits voll, während Safari unter iOS noch an diesem Feature arbeitet. [4, S. 9]

2.1.5 Node.js

Node.js ist eine open-source JavaScript Laufzeitumgebung für die Entwicklung skalierbarer Webanwendungen [14]. Selbst baut Node.js auf der V8-Engine auf, einer Laufzeitumgebung, die auch von Google Chrome genutzt wird [15, S. 1]. Wegen zeitsparenden Features, wie automatischem Typecasting oder der Tatsache, dass Node.js alle Daten als Objekt behandelt, erfreut sich Node.js großer Beliebtheit [16, S. 12]. Die Kombination mit dem Package Manager npm ermöglicht die einfache Installation und Nutzung von Modulen, um die Funktionalität der Plattform zu erweitern. [15, S. 9].

2.1.6 Angular

Angular ist ein open-source Type-Script basiertes Framework zur Entwicklung von Webanwendungen, welches Node.js nutzt.

Mit über Achttausend Mitwirkenden Entwicklern (Angular CLI) beziehungsweise über Siebentausend Mitwirkender (Angular Framework) belegt das Angular Command Line Interface und das Angular Framework die Plätze 4 und 6 der größten Projekte auf Github. [17]

Das Framework arbeitet auf Basis von Komponenten. Ein Eingabefeld, Seite oder eine Liste werden in Angular als solche Komponenten seperat betrachtet. Auch in der Dateistruktur werden Komponenten stark getrennt. Jede Komponente besitzt beispielsweise ein eigenes CSS (oder SCSS) und HTML-File. Eine Komponente für eine Seite kann so auch eine oder sogar mehrere Listenkomponenten einbinden. Durch die Wiederverwendung von Code-Fragmenten in Komponenten wird der Programmcode sehr übersichtlich und strukturiert.

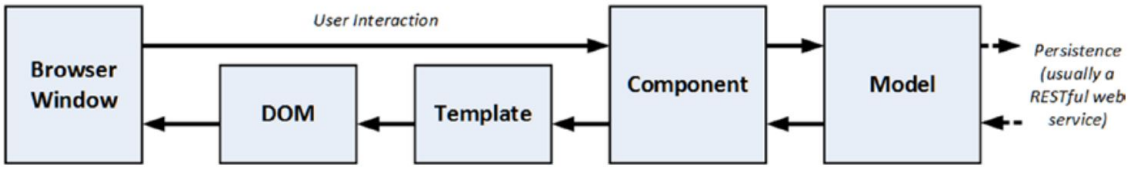

Abbildung 2.3: MVC Konzept von Angular [18, S. 35, Abbildung 3-4]

Eine Angular Anwendung ist in drei Einheiten gegliedert:

Model
Enthält Logik für die Verwaltung von Daten, beispielsweise das Erstellen, Speichern oder Modifizieren. Dies kann über die Kommunikation mit einem Webserve via REST-API erfolgen. Das Model enhält keine Logik, um mit dem Nutzer zu interagieren.

Component
Enthält Logik für das Aktualisieren der Daten im Model aufgrund Nutzerinteraktion.

Template
Enthält Logik und Markup, um dem Nutzer Daten anzeigen zu können.

[18]

2.2 native Apps

2.2.1 iOS

2.2.2 Android

Entwicklung

Native Android Anwendungen werden in der Regel in Java entwickelt. Durch die Nutzung der zahlreichen APIs wird aus einem Java Programm eine native Android App. [19, S. 1] Mittlerweile wird die teilweise veraltete Java-Syntax graduell durch die modernere Programmiersprache Kotlin abgelöst. [20]

Meist werden Android Apps mithilfe der Entwicklungsumgebung Android Studio entwickelt, welches auf der IntelliJ IDE von JetBrains aufbaut, aber von Google weiterentwickelt wird. Die IDE bietet Entwicklern unter anderem einen visuellen Layout Editor und eine Vielzahl von Android Emulatoren zum Testen der Apps auf verschiedenen Android Versionen und unterschiedlicher Hardware. Dafür wird jedoch performante Hardware zum Entwickeln benötigt: 8 Gigabyte Arbeitsspeicher oder mehr ist die Empfehlung der Herausgeber. [21]

Vertrieb

Die meisten Apps beziehen Nutzer über den Google Play Store, einem Onlineshop für kostenlose und kostenpflichtige Android Anwendungen. Updates werden ebenfalls über den Play Store installiert. Alternativ kann ein Nutzer eine App in Form einer `.apk`-Datei installieren. Dieser Weg bleibt jedoch aufgrund der Umständlichkeit und unbekannten Sicherheitsprüfung der App weitestgehend ungenutzt.

2.2.3 Grundlegendes

2.2.4 Installation einer nativen App

Android

3 Architektur

Die hier zu entwickelnde Anwendung dient zum Anlegen und Verwalten von Aufgaben der Nutzer*innen. Somit löst sie die sogenannte, analoge *To-Do-Liste* ab. Dabei hat die Anwendung (und somit auch die Studienarbeit) keinen Anspruch auf Innovationsdarbierung. Die Begründung der dieses speziellen Entwicklungsbeispiels liegt darin, dass eine To-Do-Listen-Anwendung ein großes Spektrum von Funktionen abbilden kann. Dieses Spektrum reicht von grundlegenden Funktionen (bspw. dem bloßen Anlegen von Aufgaben) bis zu komplexeren Inhalten (bspw. automatischen Push-Notifications über unerledigte oder überfällige Aufgaben). Diese design- und architekturbedingenden Entscheidungen werden im Folgenden definiert und näher beschrieben.

Die Beschreibung der Architektur einer zu entwickelnden Applikation ist eine maßgebende Disziplin im Software-Engineering-Prozess. Dieser Prozess geschieht im Regelfall vor Beginn der Implementierung und ermöglicht, bezogen auf den Umfang dieser Studienarbeit, die Vergleichbarkeit der Applikation hinsichtlich der relevanten Entwicklungsplattformen. Der Umfang sämtlicher Software-Engineering-Prozesse wird grundsätzlich in großen Entwicklerteams praktiziert. Diese gehen der Entwicklung meist komplexer und skalierbarer Anwendungen nach. Im Vergleich dazu ist die hier zu entwickelnde Mobilanwendung lediglich Mittel zum Zweck für die Beantwortung der Forschungsfrage. Das Entwicklungsteam der To-Do-Anwendung besteht aus zwei Personen, welche sich im Rahmen dieser Arbeit autark mit unterschiedlichen Entwicklungsplattformen beschäftigen.

Dies sorgt dafür, dass sich lediglich eine abgespeckte Form des Software-Engineering auf dieses Projekt anwenden lässt. Konkret bedeutet dies, dass zum Einen keine spezifischen Aussagen über den Software-Prozess bzw. über das Entwicklungsmodell (Wasserfall-Modell, iteratives Modell, etc.) gemacht werden. Dies würde sich hinsichtlich des verhältnismäßig geringen Entwicklungs- und Wartungsaufwands der App kontraproduktiv auf die Zielorientierung auswirken. Umso wichtiger ist die Definition funktionaler sowie nicht-funktionaler Anforderungen, welche daraufhin näher erläutert und spezifiziert werden. Auch die Interaktion zwischen Nutzer*in und Anwendung muss für eine vergleichende Entwicklung definiert werden. Die zeitlichen

und komponentenabhängigen Abläufe innerhalb der App gehören ebenfalls zu den Bestandteilen der Architektur. Letztere beiden Punkte werden im Rahmen des sog. *Systems Modelling* definiert. Darüber hinaus werden auch optische Aspekte und Verhaltensweisen des UI abgegrenzt.

3.1 Anforderungsdefinition

Die Funktionalität der Anwendung wird zunächst über die Anforderungsdefinition näher beschrieben. Diese kann auf allgemeine Funktionsweisen der App (nicht-funktionale Anforderungen), sowie auf spezifische, technische Charakteristika abgegrenzter Bereiche der Anwendung (funktionale Anforderungen). Grundsätzlich gilt es, nicht-funktionale Anforderungen im Laufe der Anforderungsdefinition in meist mehrere funktionale Anforderungen zu überführen, da diese qualifizierbarer sowie quantifizierbarer Natur sind und somit ebenfalls für eine bessere Vergleichbarkeit der zu entstehenden Anwendungen beitragen könnten.

3.1.1 Nicht-Funktionale Anforderungen

Die folgenden nicht-funktionalen Anforderungen beziehen sich auf Teile der Anwendung, welche jedoch abstrakter Natur sind, weswegen sie zunächst zu nicht-funktionalen Anforderungen gezählt werden müssen. Aufgrund der Formulierung werden diese Anforderungen auch als *Nutzeranforderungen* (engl. *User Requirements*) bezeichnet und stehen den spezifischeren, technisch versierteren *Systemanforderungen* (engl. *System Requirements*) gegenüber.

Anlegen, Auflisten, Bearbeiten und Löschen von Aufgaben Die Anwendung ermöglicht es, Aufgaben hinzuzufügen. Die hinzugefügten Aufgaben werden aufgelistet. Bei Bedarf soll der Inhalt der Aufgabe nachträglich abgeändert werden können. Ebenfalls ist es möglich, die Aufgabe aus der Ansicht innerhalb der Anwendung zu entfernen.

Aufgaben bestehen nach Neustart der Anwendung bei Wird die Applikation (gewollt und ungewollt) neu gestartet, bildet sie nach Neustart dieselben Aufgaben und Einstellungen wie zuvor ab.

Priorisierung der Aufgaben möglich Bei Bedarf ist es möglich, einer bestimmten Aufgabe einen gesonderten Stellenwert zuzuweisen.

Benachrichtigungen über nicht-erledigte und überfällige Aufgaben Der*die Nutzer*in wird unabhängig vom Status der Applikation oder des Smartphones (d.h. geöffnet, im Hintergrund oder geschlossen bzw. gesperrt oder entsperrt) über nicht-erledigte und überfällige Aufgaben benachrichtigt.

Neben dieser Art der nicht-funktionalen Anforderungen koexisitieren jene, welche zwar ebenfalls abstrakt und allgemein gehalten sind, jedoch keinen Bedarf resp. keine Möglichkeit zur weiteren Spezifizierung an dieser Stelle des Prozesses aufweisen.

Bereitstellung der Anwendung für mehrere Plattformen Die Anwendung ist nicht nur auf einer Plattform verfügbar, sondern kann auf Geräten unterschiedlicher Betriebssysteme installiert und verwendet werden.

Aussehen und Verhalten sind deckungsgleich Unabhängig davon, welche Plattform genutzt wird, ist die Interaktion zwischen dem*der Nutzer*in und der Anwendung annähernd identisch. Davon ausgenommen sind Aspekte, welche auf der entsprechenden Plattform nicht oder nur mit unverhältnismäßigem Aufwand erreicht werden können.

zeiteffizienter Entwicklungsprozess Um auch einen entwicklungstechnischen Vergleich ziehen zu können, soll die Anwendung in einer dem Projekt angemessenen Zeit vollständig entwickelt werden können.

Die Problematik nicht-funktionaler Anforderungen im Bezug auf realistisch zu betrachtende Entwicklungsprojekte kann hier interpretiert werden. Vor allem bei eher unerfahrenen Entwicklern (zu welchen sich das Entwicklerteam dieses Projektes zu zählen erlaubt) sind bestimmte Tendenzen unklar. Dazu gehören bspw. das Bewusstsein über die Realisierbarkeit bestimmter Komponenten sowie die zeitliche Aufwandseinschätzung. Diese Störfaktoren werden im Laufe der Arbeit versucht, entkräftet zu werden und sind in die Evaluation der Forschungsfrage kritisch einzubeziehen.

3.1.2 Funktionale Anforderungen

Nichtsdestotrotz ist die Spezifizierung der in Sektion 3.1.1 eingangs definierten nichtfuntionalen Anforderungen noch ausstehend. Zur Unterstützung der Lesbarkeit werden diese in Reihenfolge der nicht-funktionalen Anforderungen abgehandelt.

Bereitstellung klassischer CRUD-Operationen Sog. *Create, Read, Update und Delete (CRUD)*-Operationen greifen auf das Datenmodell der Anwendung zu. Diese erlauben die Manipulation der Daten auf Basis der gewünschten Operation. Diese Operationen sind unabhängig voneinander zu definieren und sinnvoll in

den Verwendungsprozess der App einzubauen. Man spricht hier auch von sog. CRUD-*Endpoints*, welche vereinfacht als statische Funktionen beschrieben werden können.

Listendarstellung Die Aufgaben sollen grundsätzlich in einer sortierten Liste dargestellt werden. Die Liste besteht aus individuellen Elementen, welche jeweils eine Aufgabe darstellen. Jene CRUD-Operationen, welche speziell auf eine bestimmte Aufgabe angewandt werden sollen, finden ihre Aktivierung ebenfalls über ihre entsprechenden Elemente.

Bereitstellung eines Persistent Services Bei Ausführen der zuvor definierten CRUD-Operationen werden die Daten nicht nur in den flüchtigen Arbeitsspeicher des Smartphones geschrieben, sondern zugleich auch auf einen der Applikation zugewiesenen Festspeicher. Diese idealisierte Datenbank gleicht dem Datenmodell für die CRUD-Operationen und wird somit bei jeder Ausführung dieser aktualisiert bzw. beansprucht.

Definition einer Hierarchie für Aufgaben Eine hierarchische Struktur der Daten soll ermöglichen, bestimmte Aufgaben seitens der Anwendung anders zu behandeln als andere. Durch das Setzen eines sog. *Flags* können die Aufgaben entsprechend der Hierarchiestruktur bestimmte Zustände übergeben bekommen, konkret eine hervorgehobene optische Darstellung innerhalb der UI sowie die Präsentation an Anfang der Liste (Eingriff in die Sortierung der Aufgaben)

Streng genommen sind funktionale Anforderungen sehr granulär zu definieren. Da es sich hier jedoch um eine wissenschaftliche Arbeit handelt, und nicht um eine Entwicklerdokumentation, wird auf kleinste Genauigkeiten verzichtet. Viel eher soll die nächste Sektion die genaue, weiterhin plattformunabhängige Umsetzung dieser Anforderungen erläutern, welche als Maßgabe für die spätere Entwicklung der Applikation dienen soll.

3.2 Funktionen

Die Anwendung unterstützt klassische CRUD CRUD
offline cache

Bezeichnung	HEX-Code	Beispiel
Allgemeines		
Hintergrund	#F2F2F2	
Schriftfarbe	#8C8C8C	
Bedienelemente		
Hintergrund für inaktive Bedienelemente	#CECECE	
Checkbox (checked) Hintergrund	#1A66FF	
Schriftfarbe der Checkbox (checked)	#FFFFFF	

Tabelle 3.1: Farbtabelle

3.3 Oberfläche (UI)

3.3.1 Elemente

-Todo Listen eintrag -Flag important -checkbox -edit Methode -delete -Input Methode

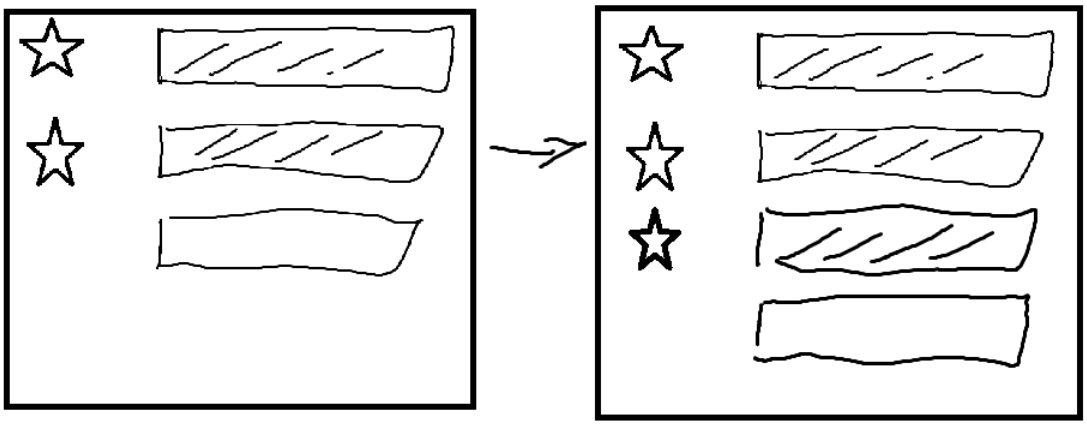

Abbildung 3.1: Verhalten beim Einfügen

3.3.2 Größen

3.3.3 Farben

Um die Anpassungsmöglichkeiten von nativen Apps und PWA zu evaluieren, werden in Tabelle 3.1 Farben für einzelne Elemente des User Interface festgelegt.

3.3.4 Aussehen/Design

3.4 Komponenten

- **App Container**
  Das unterste UI-Element der Anwendung wird im Folgenden als App Container bezeichnet, da es aufgrund der verschiedenen Technologien keine einheitliche Bezeichnung dafür gibt. Der App Container ist bei der PWA das `body` Element des HTML und bei der nativen iOS App ein `View` Element.

  Besonderes: Ist der Inhalt des App Containers größer, als der Bildschirm, wird in vertikale Richtung gescrollt. Es gibt kein horizontales Scrolling, stattdessen müssen sich alle anderen Elemente automatisch anpassen.

- **Todoliste**
  Der Kern der Anwendung ist eine Liste, die einzelne Todos enthält. Neue Todos werden stets unten an die Liste angefügt.

  Besonderes: Es gibt genau eine Todoliste. Sie kann nicht gelöscht werden und enthält mindestens ein (leeres) Todo.

- **Todo (Listeneintrag)**
  Ein Todo ist ein Listenelement der Todoliste. Dieses besteht aus einem editierbaren Textfeld. Seitlich des Textfelds befinden sich Buttons zum Entfernen des Eintrags und zum Markieren des Eintrags als "wichtig".

  Das Todo hat links neben dem Eingabefeld eine Checkbox. Bis zum "check"durch den Nutzer ist die Checkbox nicht markiert.

  Besonderes: Wenn der Inhalt eines bestehenden Todos gelöscht wird, bleibt es bestehen und wird nicht automatisch entfernt.

- **Eingabemöglichkeit**
  Damit der Nutzer neue Todos eintragen kann, wird ein Eingabefeld benötigt. Das Eingabefeld ist idealerweise in die Todoliste integriert. Für den Nutzer soll stets ein Eingabefeld unter der Todoliste sichtbar sein. Da bereits die Todoliste aus einzelnen Eingabefeldern besteht, muss diese während der Eingabe eines neuen Elements nur um ein neues leeres Listenelement erweitert werden.

  Besonderes: Es werden keine leeren Todos eingefügt. Ein Todo muss mindestens ein Zeichen enthalten.

3.5 Designentscheidungen PWA

Seit 2014 bis heute ist JavaScript die häufigste Sprache auf GitHub. Das stärker typisierte TypeScript gehört zu den am schnellsten wachsenden Sprachen [17]. Um diese Arbeit zukunftsgewandt zu evaluieren, werden deshalb das bereits beschriebene Angular-Framework verwendet, dass TypeScript (transpilierter zu JavaScript) und die Node.js Laufzeitumgebung nutzt. Nicht zuletzt ist auch der Große Beitrag von Google (als Mobilgerätehersteller und Betriebssystementwickler von Android) zum Angular-Projekt ein Grund, die PWA in Form einer Angular Anwendung zu realisieren.

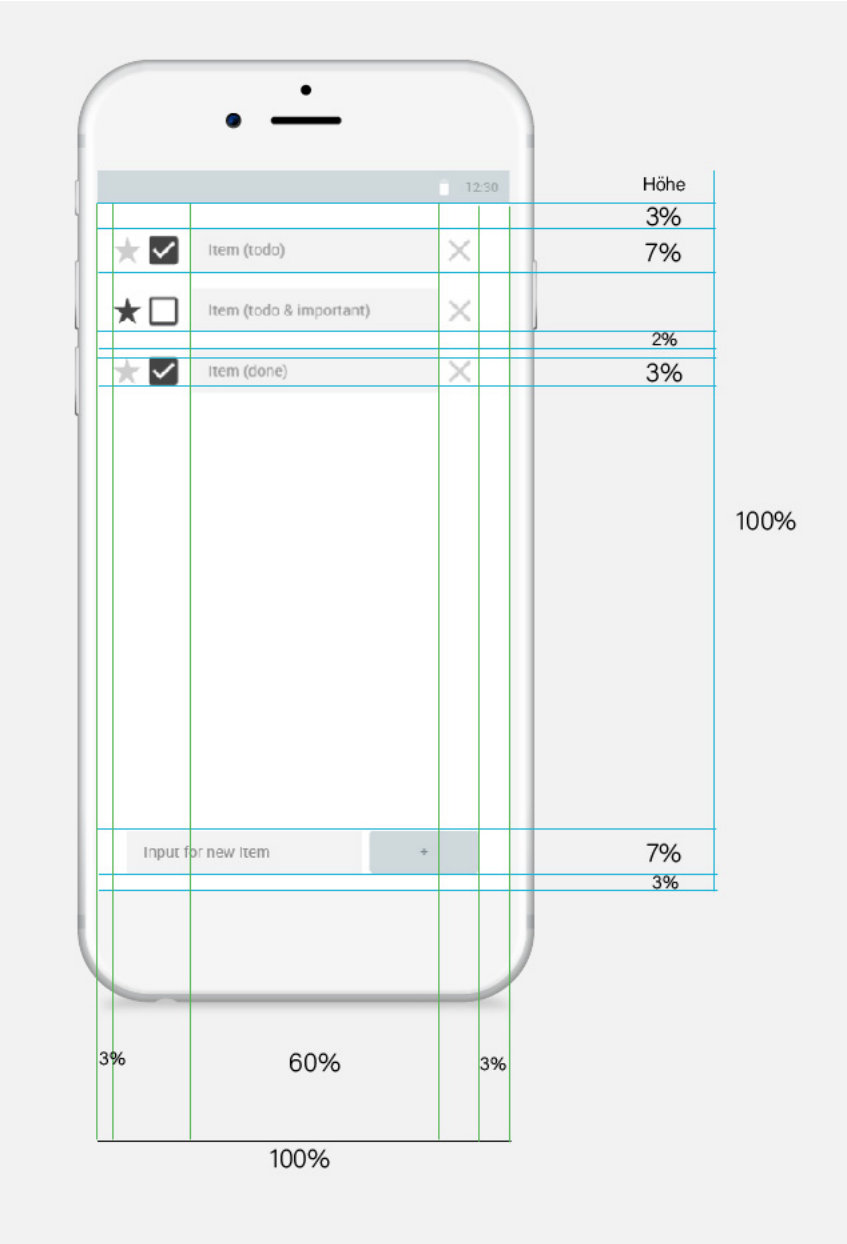

Abbildung 3.2: Größen

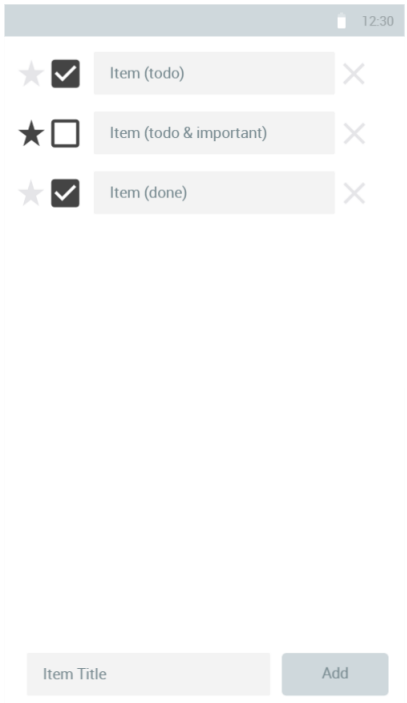

Abbildung 3.3: Wireframe

4 Wissenschaftliches Framework

Erläuterung:

- **Plattformabhängigkeit**
  Wie viele Plattformen beziehungsweise Betriebssysteme werden unterstützt? Ist die Nutzung auf Smartphone begrenzt oder kann die Anwendung auch auf einem Tablet oder Desktop installiert werden?

- **Dateizugriff**
  Können Daten einfach zwischengespeichert und wieder ausgelesen werden?

- **Installation**
  Wie kompliziert ist die Installation der Anwendung für den Nutzer?
  Wie kompliziert ist die zur Verfügungstellung durch die Entwickler?

- **Speicherbedarf**
  Der Speicherplatz auf einem Smartphone ist deutlich kleiner, als bei einem Desktop-Computer. Wie hoch ist der Speicherbedarf für eine Anwendung?

- **Aktualisierbarkeit**
  Kann der Nutzer Updates verhindern? Muss der Nutzer Updates aktiv zustimmen? Wie kompliziert ist es für Entwickler Updates auszurollen?

Kriterium		- -	-	neutral	+	++	Gesamtanteil
Plattformabhängikeit	Android	0	1	2	3	4	15%
	Desktop	0	1	2	3	4	15%
Dateizugriff		0	1	2	3	4	15%
Installation		0	1	2	3	4	15%
Speicherbedarf		0	1	2	3	4	15%
Aktualisierbarkeit		0	1	2	3	4	15%
Berechtigungen		0	1	2	3	4	15%
Summe		0	1	2	3	4	100

Tabelle 4.1: Punktekatalog für Kriterien

5 Implementierung

5.1 Implementierung nativ

5.2 Implementierung PWA

5.2.1 Einrichten der Projektumgebung

5.2.2 Angular

https://www.sitepoint.com/angular-2-tutorial/

5.2.3 Hinzufügen der Manifestdatei

https://medium.com/poka-techblog/turn-your-angular-app-into-a-pwa-in-4-easy-steps-543510a9b626

Abbildung 5.1: Größen

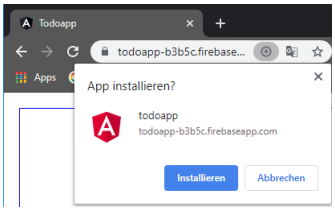

Abbildung 5.2: Größen

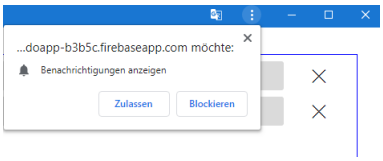

Abbildung 5.3: Größen

6 Evaluation

7 Reflexion

Literatur

[1] *Number of smartphone users worldwide 2014-2020*. Adresse: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/.

[2] *Development of the world population until 2050*. Adresse: https://www.statista.com/statistics/262875/development-of-the-world-population/.

[3] *Mobile websites by net reach in germany 2019*. Adresse: https://www.statista.com/statistics/425372/mobile-websites-by-net-reach-germany/.

[4] D. Sheppard, *Beginning progressive web app development - creating a native app experienc*. Apress, 2017. DOI: 10.1007/978-1-4842-3090-9.

[5] *Mobile browser market share worldwide 2012-2019*. Adresse: https://www.statista.com/statistics/263517/market-share-held-by-mobile-internet-browsers-worldwide/.

[6] Google, *Progressive Web Apps | Google Developers*. Adresse: https://developers.google.com/web/progressive-web-apps (besucht am 12. 10. 2019).

[7] ——, *Offline First - Google Chrome*. Adresse: https://developer.chrome.com/apps/offline%7B%5C_%7Dapps (besucht am 12. 10. 2019).

[8] M. Hajian, *Progressive Web Apps with Angular*. Apress, 2019. DOI: 10.1007/978-1-4842-4448-7.

[9] P. LePage, *The mini-info bar*. Adresse: https://developers.google.com/web/updates/images/2018/06/a2hs-infobar-cropped.png.

[10] ——, *Add to home screen | web fundamentals | google developers*. Adresse: https://developers.google.com/web/fundamentals/app-install-banners.

[11] M. Gaunt und P. Kinlan, *The Web App Manifest | Web Fundamentals | Google Developers*. Adresse: https://developers.google.com/web/fundamentals/web-app-manifest?hl=en (besucht am 12. 10. 2019).

[12] P. LePage, *Progressive web apps on desktop | google developers*. Adresse: https://developers.google.com/web/progressive-web-apps/desktop.

[13] C. Liebel, *Funktionsweise eines service workers*, Juni 2017. Adresse: https://www.heise.de/developer/imgs/06/2/2/1/9/1/4/9/ServiceWorker-8a0968f1b295f1ff.png.

[14] N. Foundation, *About*. Adresse: https://nodejs.org/en/about/.

[15] C. Gackenheimer, *Node A Problem-Solution Approach*. Springer, 2013. DOI: 10.10007/978-1-4302-6059-2.

[16] A. Mardan, *Practical Node.js Building Real-World Scalable Web Apps*. Apress, 2018.

[17] *Projects*, 2019. Adresse: https://octoverse.github.com/projects.

[18] A. Freeman, *Pro Angular 2*. Springer Verlag, 2017. DOI: 10.1007/978-1-4842-2307-9.

[19] J. Friesen, *Learn Java for Android Development*. Apress, 2014.

[20] *Kotlin and android : Android developers*. Adresse: https://developer.android.com/kotlin.

[21] *Android studio and sdk tools*. Adresse: https://developer.android.com/studio.