

BEUTH HOCHSCHULE  
FÜR TECHNIK  
BERLIN

University of Applied Sciences

Fachbereich VI Informatik und Medien

**Masterarbeit**

von

Lino Fischer

zur Erlangung  
des akademischen Grades  
Master of Science (M.Sc.)

im Studiengang  
Medieninformatik

Thema:

**Konzeption und Implementierung einer Progressiven Web App für  
eine bestehende Projektmanagement Software**

Betreuer: Prof. Dr. Strzebkowski

Gutachter: Prof. Dr. Gers

Eingereicht: 01.10.2018

Beuth Hochschule für Technik Berlin

## **Abstract**

*Die vorliegende Arbeit beinhaltet die Konzeption und Implementierung einer Angularbasierten Progressiven Web App (PWA) für die Projektmanagement Plattform OpenProject. Es wird von der Anforderungsanalyse, über den Entwurf, bis hin zur Implementierung jeder Schritt der Entwicklung dokumentiert. Der ganzen Entwicklung liegt ein konkreter Anwendungsfall zugrunde. Die Anwendung soll später auf Baustellen eingesetzt werden, um Mängel, Tasks und Ideen schnell und unkompliziert in das bestehende Projektmanagement System einzupflegen. Die PWA soll in Bezug auf Usability einen Mehrwert gegenüber der bereits bestehenden Web App bieten.*

*Es wurden die wichtigsten allgemeinen Usability Standards für Webanwendungen aus verschiedenen Quellen zusammengetragen. Aufgrund dieser Ergebnisse wurden spezifische Richtlinien für formularbasierte Anwendungen entwickelt. Sowohl anhand der allgemeinen, als auch anhand der formularspezifischen Richtlinien wurden die entwickelte PWA und die bereits bestehenden Web App untersucht und miteinander verglichen. Es konnte gezeigt werden, dass für den zugrunde liegenden Anwendungsfall die PWA eine eindeutig bessere Usability bietet und dadurch dem Nutzer innerhalb des Anwendungsfalls eine performantere Arbeit mit der OpenProject Plattform ermöglicht.*

# Inhaltsverzeichnis

|  |           |
|--|-----------|
| <b>Inhaltsverzeichnis</b>                                  | <b>2</b>  |
| <b>Abbildungsverzeichnis</b>                               | <b>5</b>  |
| <b>1 Einleitung</b>  | <b>7</b>  |
| 1.1 Motivation   | 7         |
| 1.2 Problemstellung  | 8         |
| 1.3 Vorgehensweise   | 8         |
| <b>2 Applikationstypen - Einordnung PWA</b>                | <b>10</b> |
| 2.1 Native App   | 10        |
| 2.2 Web App  | 10        |
| 2.3 Hybride App  | 11        |
| 2.4 Progressive Web App                                    | 11        |
| 2.4.1 Definition   | 11        |
| 2.4.2 Charakteristik und Bedeutung                         | 12        |
| 2.5 Abgrenzung PWA - App Typen im Vergleich                | 14        |
| <b>3 Projektmanagement &amp; Mobile Enterprise</b>         | <b>17</b> |
| 3.1 Definition Projektmanagement                           | 17        |
| 3.2 Projektmanagement in der Baubranche                    | 17        |
| 3.3 Projektmanagement Software                             | 20        |
| 3.3.1 Definition   | 20        |
| 3.3.2 Webbasierte Projektmanagement Systeme (WPMS)         | 21        |
| 3.3.3 Projektmanagement Software in der Baubranche         | 21        |
| 3.4 Mobile enterprise                                      | 23        |
| <b>4 OpenProject. Collaborative Project Management</b>     | <b>24</b> |
| 4.1 Was ist OpenProject?                                   | 24        |
| 4.2 Funktionalität   | 25        |
| 4.2.1 Projects   | 25        |
| 4.2.2 Work packages  | 26        |
| <b>5 Anforderungsanalyse &amp; konzeptioneller Entwurf</b> | <b>28</b> |
| 5.1 Zielgruppe   | 28        |
| 5.2 Anwendungsfall   | 28        |
| 5.2.1 Einordnung   | 28        |
| 5.2.2 Beispielszenario                                     | 29        |
| 5.3 Entwurf der Progressiven Web App                       | 31        |
| 5.4 Qualitätszielbestimmungen                              | 34        |
| 5.4.1 Allgemeine nichtfunktionale Anforderungen            | 34        |
| 5.4.2 Anforderungen an eine Progressive Web App            | 35        |
| <b>6 Usability Richtlinien</b>                             | <b>36</b> |

|   |           |
|---|-----------|
| 6.1 Definition Usability                                  | 36        |
| 6.2 Die Bedeutung und Probleme von Usability Richtlinien  | 36        |
| 6.3 Der Mobile First Ansatz                               | 39        |
| 6.3.1 Definition  | 39        |
| 6.3.2 Allgemeine Unterschiede zwischen Mobile und Desktop | 39        |
| 6.4 Allgemeine Usability Prinzipien                       | 40        |
| 6.4.1 Gestaltpsychologische Prinzipien                    | 40        |
| 6.4.2 Allgemeine Heuristiken für User Interface Design    | 42        |
| 6.4.3 Die Prinzipien der Einfachheit (Simplicity)         | 43        |
| 6.4.4 Grundprinzipien der Dialoggestaltung                | 45        |
| 6.5 Formularspezifische Usability Standards               | 46        |
| 6.5.1 Navigationsstruktur & Seitenaufbau                  | 46        |
| 6.5.2 Input Felder  | 47        |
| 6.5.3 Buttons   | 49        |
| 6.6 Material Design Richtlinien                           | 50        |
| <b>7 Verwendete Technologien</b>                          | <b>54</b> |
| 7.1 Typescript  | 54        |
| 7.2 Angular 6   | 54        |
| 7.2.1 Components  | 54        |
| 7.2.2 Services  | 56        |
| 7.2.3 Modules   | 57        |
| 7.3 Service Worker  | 57        |
| 7.3.1 Definition  | 57        |
| 7.3.2 Einen Service Worker anmelden                       | 58        |
| 7.3.3 Die ngsw-config.json Datei                          | 58        |
| 7.4 Die Manifest.json Datei                               | 63        |
| 7.5 Angular Material                                      | 64        |
| 7.6 IndexedDB & Dexie.js                                  | 64        |
| <b>8 Implementierung</b>                                  | <b>66</b> |
| 8.1 Funktionale Umsetzung                                 | 66        |
| 8.1.1 Work package erstellen                              | 66        |
| 8.1.2 Liste an Work packages anzeigen                     | 68        |
| 8.1.3 Work package Details anzeigen                       | 69        |
| 8.1.4 Einstellungen                                       | 71        |
| 8.2 Technische Umsetzung                                  | 72        |
| 8.2.1 Caching der Anwendung                               | 72        |
| 8.2.2 Senden eines Work packages                          | 74        |
| 8.2.3 Senden von Work packages (Intervall)                | 76        |
| 8.2.4 Konzept der Datenbanken                             | 76        |
| 8.3 Qualitätssicherung                                    | 79        |
| 8.3.1 Allgemeine nichtfunktionale Anforderungen           | 79        |

|   |            |
|---|------------|
| 8.3.2 Anforderungen an die PWA                                    | 80         |
| <b>9 Vergleich OpenProject PWA &amp; OpenProject Webanwendung</b> | <b>82</b>  |
| 9.1 Allgemein   | 82         |
| 9.2 Work package erstellen  | 82         |
| 9.2.1 Allgemeine Richtlinien                                      | 82         |
| 9.2.2 Formular spezifische Richtlinien                            | 86         |
| 9.3 Work package Detailansicht                                    | 89         |
| 9.4 Work packages Listenansicht                                   | 92         |
| <b>10 Fazit &amp; Ausblick</b>                                    | <b>93</b>  |
| <b>Literaturverzeichnis</b>                                       | <b>96</b>  |
| <b>Anhang A Die PWA starten</b>                                   | <b>102</b> |
| <b>Anhang B Erklärung</b>   | <b>103</b> |

# Abbildungsverzeichnis

|  |    |
|--|----|
| Abbildung 1: Entwicklungsprozess der PWA.                                  | 9  |
| Abbildung 2: Eine PWA herunterladen.                                       | 11 |
| Abbildung 3: Native App, Web App, Hybride App (1).                         | 14 |
| Abbildung 4: Native App, Web App, Hybride App (2).                         | 15 |
| Abbildung 5: Native App, Web App, Hybride App (3).                         | 16 |
| Abbildung 6: Lebenszyklus eines Bauprojekts.                               | 19 |
| Abbildung 7: App zur mobilen Erfassung von Mängeln.                        | 22 |
| Abbildung 8: Desktop-Tool für eine Mängelmanagement Anwendung.             | 23 |
| Abbildung 9: Der Nutzen von mobiler Technologie in einem Geschäftsprozess. | 23 |
| Abbildung 10: OpenProject - Dashboard.                                     | 25 |
| Abbildung 11: OpenProject - Create New Project.                            | 26 |
| Abbildung 12: OpenProject - Create Work package.                           | 27 |
| Abbildung 13: OpenProject - Work package Liste.                            | 27 |
| Abbildung 14: OpenProject - Work package detail.                           | 28 |
| Abbildung 15: BPMN - Mängelmanagement.                                     | 30 |
| Abbildung 16: Use Case - PWA Produktfunktionen (PF).                       | 31 |
| Abbildung 17: Produktfunktion 1 - Ein Work package erstellen.              | 32 |
| Abbildung 18: Produktfunktion 2 - Work package Details anzeigen.           | 32 |
| Abbildung 19: Produktfunktion 3 - Liste von Work packages anzeigen.        | 33 |
| Abbildung 20: Produktfunktion 4 - Einstellungen bearbeiten.                | 34 |
| Abbildung 21: Angular Components.  | 55 |
| Abbildung 22: Angular Components - Event binding.                          | 56 |
| Abbildung 23: Angular Services - Dependency injection.                     | 56 |
| Abbildung 24: Service Worker Lifecycle.                                    | 58 |
| Abbildung 25: Service Worker - ngsw-config.json.                           | 59 |
| Abbildung 26: ngsw-config.json - AssetGroup interface.                     | 60 |

|   |    |
|---|----|
| Abbildung 27: ngsw-config.json - DataGroup interface.   | 61 |
| Abbildung 28: Dexie.js - Work package Datenbank.  | 65 |
| Abbildung 29: Work package Attribute.   | 77 |
| Abbildung 30: Create a new Work package - basic Input (links); advanced Input (rechts).   | 67 |
| Abbildung 31: Create a new Work package. Wenn der Nutzer keine Internetverbindung hat (links); Wenn Internetverbindung besteht (rechts).          | 68 |
| Abbildung 32: Work packages Listenansicht.  | 69 |
| Abbildung 33: Work packages Detailansicht. (links) Detailansicht bei vorhandenem Work package. (rechts) Ansicht wenn Work package gelöscht wurde. | 70 |
| Abbildung 34: Die Koordinaten eines Work packages in Google Maps.   | 71 |
| Abbildung 35: Settings.   | 72 |
| Abbildung 36: Die ngsw-config der PWA.  | 73 |
| Abbildung 37: Service Worker on Network Response  | 73 |
| Abbildung 38: Senden eines Work packages.   | 74 |
| Abbildung 39: CheckConnectionService.   | 75 |
| Abbildung 40: Intervall Observable.   | 76 |
| Abbildung 41: Flussdiagramm: Send Work package.   | 77 |
| Abbildung 42: Flussdiagramm: Send Work package Intervall.   | 78 |
| Abbildung 43: AllWorkpackages Datenbank.  | 79 |
| Abbildung 44: Lighthouse - PWA Analyse.   | 81 |
| Abbildung 45: OpenProject - Create Work package: ohne ausgewähltes Projekt (links); innerhalb eines Projekts (rechts).                            | 83 |
| Abbildung 46: Create a Work package: PWA (rechts), Web App (links).   | 85 |
| Abbildung 47: Navigation & Seitenstruktur: PWA (links), Web App (rechts).   | 86 |
| Abbildung 48: Input Felder: PWA (links), Web App (rechts).  | 86 |
| Abbildung 49: Buttons: PWA (links), Web App (rechts).   | 88 |
| Abbildung 50: Work package Detailansicht: PWA (rechts), Web App (links).  | 90 |
| Abbildung 51: Work package Detailansicht Vergleich: PWA (rechts), Web App (links).  | 91 |
| Abbildung 52: Work package Listenansicht: Web App (links), PWA (rechts).  | 92 |

# 1 Einleitung

## 1.1 Motivation

Der derzeitig gängige Weg um sich eine App auf das Smartphone zu laden ist der Weg über den Appstore. Durch Progressiv Web Apps (PWA) könnte dieser Schritt jedoch bald der Vergangenheit angehören und den App Store obsolet werden lassen.

Prinzipiell bietet die PWA dem Benutzer zweierlei Vorteile gegenüber der "klassischen App". (1) Zum einen kann der Nutzer die Anwendung ganz einfach als Web App (Kapitel 2.2) im Browser öffnen, und direkt von der Funktionalität der Anwendung profitieren, ohne vorher den App Store aufsuchen zu müssen. (2) Ist der Nutzer dann von der Funktionalität der Anwendung überzeugt, kann er sich diese direkt aus dem Browser herunterladen. Hierbei ist es überflüssig erst über einen Banner auf den App Store Download verlinkt zu werden. Das Herunterladen der App geschieht über den "Add To Homescreen"-Button innerhalb des mobilen Browsers. Ist die App einmal auf dem Homescreen installiert, so ist sie optisch nicht mehr von einer "klassischen" App zu unterscheiden.

Die durch Google bereits 2015 entwickelte Technologie der PWA gewann am 29. März 2018 noch mehr an Bedeutung, da Apple an diesem Tag die "Safari-Tore" für die neue Generation an Applikationen öffnete. Seit diesem Tag sind die zwei größten Smartphone Betriebssysteme (Android: 87,7% Marktanteil, Apple: 12,1% Marktanteil [Gartner 2017a]) in der Lage diese Technologie zu verwenden.

Das Marktforschungsinstitut Gartner Inc. gibt jedes Jahr einen Hype Circle heraus in dem neue Technologien bezüglich der generierten öffentlichen Aufmerksamkeit bewertet werden. In dem mobile-spezifischen *Gartner Hype Cycle for Mobile Applications and Development* (2017b) wird die Progressiv Web App als "on the rise", also als noch aufsteigende Technologie beschrieben. Das lässt erahnen, dass das volle Potential dieser Technologie noch nicht ausgeschöpft wurde. Es zeigt auch, dass diese vielversprechende Technologie wohl bisher in wenigen Wirtschaftszweigen zur Prozessoptimierung eingesetzt wird.

Hingegen ist das Endgerät für PWAs, das Smartphone, doch längst im beruflichen Alltag vertreten. So zeigte zum Beispiel Accenture bereits 2012, dass 20% der heruntergeladene Apps für berufliche Zwecke genutzt werden.

Auch das in der vorliegenden Arbeit behandelte Thema des Projektmanagements gewinnt in Unternehmen zunehmend an Bedeutung. Das ist zum Beispiel daran zu sehen, dass in immer mehr Unternehmen ein eigenes Projekt Management Offices (PMOs) vertreten ist. So stieg der Anteil an PMOs von 61% (2007) auf 71% (2017) (Project Management Institute, 2017). Dabei nutzt auch sozusagen jedes größere Unternehmen eine Form von Projektmanagement Software zur Unterstützung ihrer Projektarbeit. Diese Form der Projektmanagement Software ist oft online getrieben, kollaborative Plattform zur Kommunikation und Planung der Projekte. Diese online basierten Projektmanagement Tools sind seit mehreren Jahren stark im Trend. Mordor Intelligence (2018) sagt in einer Studie voraus, dass dieser Trend auch in den kommenden Jahren weiter anhalten wird. So liegt die prognostizierte durchschnittliche jährliche Wachstumsrate der online Projektmanagement Softwareindustrie zwischen 2018 und 2023 bei durchschnittlich 10.14% (Mordor Intelligence, 2018).



## 1.2 Problemstellung

Die vorliegende Arbeit behandelt die Konzeption und Implementierung einer progressiven Web Applikation. Die Anwendung wurde für das bestehende Backend der online basierten Projektmanagement Plattform *OpenProject*<sup>1</sup> entwickelt und konzentriert sich auf die Umsetzung eines mobile optimierten Frontends. Die PWA stellt verschiedenen Funktionen bereit und steht dabei in Kommunikation mit der bestehenden API des OpenProject Backends. Die Anwendung soll ein problemloses Nutzen der OpenProject Funktionalitäten ermöglichen und bei der Arbeit mit dem gesamten Web Tool unterstützen. Prinzipiell soll die Anwendung damit unternehmensinterne Geschäftsprozesse optimieren und dadurch die Performance des gesamten Unternehmens steigern.

Potentielle Nutzer der Anwendung sind vor allem Beschäftigte aus der Baubranche. Mit der App sollen Baumängel, generelle Taks oder Ideen unkompliziert als sogenannte Work packages in ein bestehendes Projekt der OpenProject Anwendung eingepflegt werden können. Weiterhin soll man sich listenartig alle über das Smartphone erstellten Work packages anzeigen lassen, um über den Projektablauf informiert zu bleiben. Es soll außerdem die Möglichkeit geben, sich jedes Work package in einem Detailsview mit all seinen erstellten Attributen anzeigen zu lassen.

Die Anwendung kommt hauptsächlich auf Baustellen zum Einsatz, wodurch eine durchgängige Internetverbindung nicht gewährleistet werden kann. Dadurch ergibt sich eine der Kernanforderungen an die Anwendung, die der Offlinefunktionalität. Man soll unabhängig von der aktuellen Internetverbindung in der Lage sein, Work packages zu erstellen.

Prinzipiell ist die Anwendung jedoch für jeden Benutzer der OpenProject Plattform sinnvoll, der die Funktionalität der Work packages in seinem Projekt nutzen möchte.

Die Anwendung wurde nach eigens entworfenen Usability Richtlinien entwickelt und gestaltet. Diese Richtlinien sind speziell auf die Probleme und Herausforderungen einer Formularbasierten mobilen Anwendungen zugeschnitten worden. Um den tatsächlichen Mehrwert der entwickelten PWA gegenüber der bestehenden Webanwendung zu ermitteln, wurde ein Vergleich beider Anwendungen anhand der festgelegten Usability Richtlinien durchgeführt.

## 1.3 Vorgehensweise

Die vorliegende Arbeit besteht zu einem Teil aus einer programmierten Anwendung und zum anderen Teil aus dieser schriftlichen Ausarbeitung.

Die Recherche der wissenschaftlichen Artikel und der verwendeten Sachbücher wurde über verschiedene Datenbanken vorgenommen:

- ACM Digital Library<sup>2</sup>

---

<sup>1</sup> <https://www.openproject.org/>

<sup>2</sup> <https://dl.acm.org/>

- Datenbank der Universitätsbibliothek Stuttgart<sup>3</sup>
- Google Scholar<sup>4</sup>.

Die wichtigsten Suchbegriffe für den Hauptteil der Arbeit waren: *Progressive Web App*, *Projekt Management Tool*, *Usability* und *Human Computer Interaction*. Literatur die nicht über Suchbegriffe in den genannten Datenbanken gefunden wurde, wurde oft durch Zitationen in anderen Artikel oder durch die Nutzung gängiger Suchmaschinen ermittelt.

Folgende Abbildung stellt den Ablauf bei der Entwicklung der PWA noch einmal genauer dar.

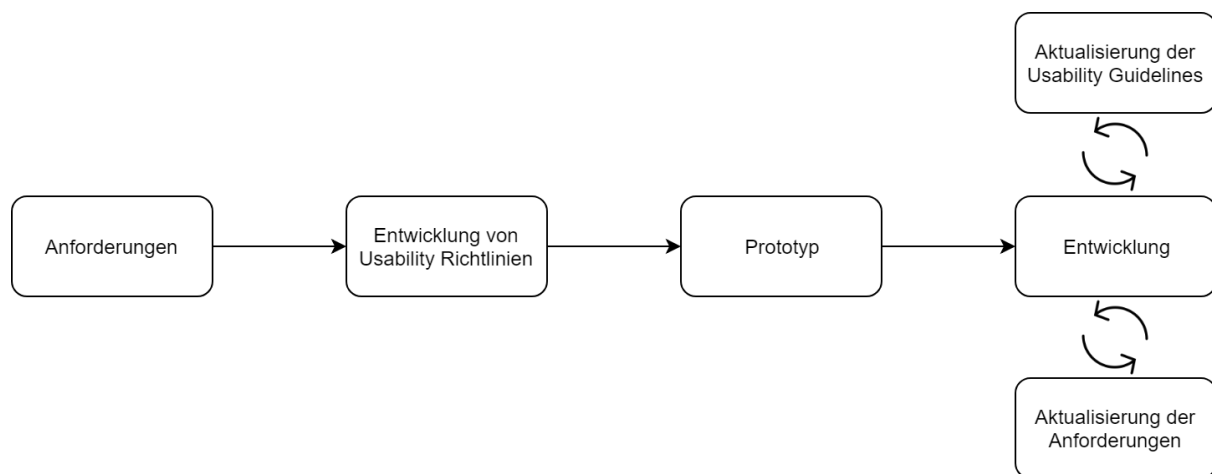


Abbildung 1: Entwicklungsprozess der PWA.

Zuerst wurden grundlegende Anforderungen für die Progressive Web App festgelegt. Auf dieser Grundlage wurden Usability Richtlinien für die PWA entwickelt (Kapitel 6). Danach wurde der Prototyp der Anwendung entwickelt. Nachdem dieser fertiggestellt war, begann die Entwicklung der App, welche durch regelmäßige Updates der Anforderungen begleitet wurde. Durch weitere Literaturrecherche ergab sich zudem immer wieder eine Aktualisierung der Usability Richtlinien, was wiederum Einfluss auf die entwickelte PWA hatte.

Nachdem die Anwendung fertig gestellt war, wurde der Vergleich zwischen der entwickelten PWA und der bestehenden OpenProject Webanwendung durchgeführt.

Die Arbeit beginnt in Kapitel 2 mit der Einordnung und Abgrenzung der PWA in Bezug auf verschiedene App Typen. Kapitel 3 gibt einen Überblick über Projektmanagement in der für den Anwendungsfall relevanten Baubranche und geht auf die verschiedenen Projektmanagement Softwaretypen sowie die Bedeutung von mobile Enterprise genauer ein. Kapitel 4 erklärt die wichtigsten Funktionen des zugrunde liegenden Tools von *OpenProject*. Kapitel 5 definiert sowohl die funktionalen als auch die nicht funktionalen Anforderungen die an die PWA gestellt werden. Kapitel 6 gibt die wichtigsten Usability Richtlinien aus der

<sup>3</sup> [https://stg.ibs-bw.de/aDISWeb/app?service=direct/0/Home/\\$DirectLink&sp=SOPAC02](https://stg.ibs-bw.de/aDISWeb/app?service=direct/0/Home/$DirectLink&sp=SOPAC02)

<sup>4</sup> <https://scholar.google.de/>

aktuellen Forschung wieder. In Kapitel 7 werden dann die für die Entwicklung der PWA notwendigen Technologien genauer erläutert. Kapitel 8 widmet sich ausführlich der tatsächlichen funktionalen und technischen Implementierung der Anwendung. In Kapitel 9 findet dann noch der Vergleich zwischen Webanwendung und PWA statt, bevor Kapitel 10 ein Fazit zieht und einen Ausblick in die Zukunft bietet.

## 2 Applikationstypen - Einordnung PWA

### 2.1 Native App

Die folgende Definition für eine native App wurde weitestgehend aus dem Whitepaper von IBM (2012) übernommen.

Im Grunde besteht eine native App aus Binärdateien, die aus dem Internet heruntergeladen, und anschließend lokal auf dem Gerät gespeichert werden können.

Der gängige Weg, um eine App zu downloaden, geschieht über den App Store (zum Beispiel der Google Play Store oder der App Store von Apple). Nach dem Installieren kann die App wie eine normale Funktion des Geräts geöffnet und genutzt werden. Die native App hat eine direkte Schnittstelle zu dem mobilen Betriebssystem ohne von irgendeinem Vermittler oder Container Gebrauch zu machen. Außerdem hat die App Zugriff auf die APIs die von dem Gerät aus bereitgestellt werden.

Eine native App wird je nach Betriebssystem des Endgeräts in verschiedenen Sprachen und in verschiedenen SDKs programmiert. So wird eine native App für Android in Java beziehungsweise Kotlin geschrieben und mit Android Studio entwickelt. Eine native iOS App hingegen wird in Swift beziehungsweise Objective-C und mit Hilfe der Entwicklungsumgebung Xcode umgesetzt.

### 2.2 Web App

Die folgende Definition für eine Webanwendung, kurz Web App, wurde weitestgehend aus dem Whitepaper von IBM (2012) übernommen.

Ein modernes mobiles Endgerät besitzt leistungsfähige Browser, die HTML 5, CSS und JavaScript unterstützen. Eine mobile Web App wird mit Hilfe dieser gängigen Webtechnologien umgesetzt.

Die Web App erkennt, wenn ein Smartphone darauf zugreift, und bietet einen Smartphone optimierten "touch-freundlichen" mobile View an.

Dadurch, dass die App im Browser läuft, wird eine Plattformunabhängigkeit gewährleistet und der Entwickler kann sich auf eine einzelne Codebasis konzentrieren. Durch fehlende APIs ist es Web Apps jedoch nicht immer möglich alle Device APIs anzusprechen (bspw. kann es problematisch sein, Gesten des Benutzers zu erkennen oder auf das Dateisystem zuzugreifen). Eine Web App funktioniert nur im Browser und ist damit immer von bestehender Internetverbindung abhängig.

## 2.3 Hybride App

Hybride mobile Apps basieren ebenfalls auf gängigen Web Technologien wie HTML, CSS und JavaScript. Ein Framework für die hybride Entwicklung (wie zum Beispiel Apache Cordova<sup>5</sup>) stellt dem Entwickler zwei wichtige Dinge bereit (Malavolta, Soru, & Terragni, 2017):

Erstens einen nativen Wrapper der den webbasierten Code beinhaltet und zweitens einen generische JavaScript API welche die Brücke zu der jeweiligen Plattform API schlägt (Malavolta et al., 2017). Weiterhin gibt es Frameworks, (wie zum Beispiel IONIC<sup>6</sup>) die nativ aussehende UI Elemente bereitstellen, um den *look & feel* einer nativen App herzustellen.

Die hybride App kann in Webtechnologien geschrieben, und später in einem Wrapper verpackt in den jeweiligen App Stores ausgeliefert werden. Es ist also eine Web App die verpackt in einem Container vorgibt, sie wäre eine native App. Meistens gibt es hier nur eine Codebasis für alle mobilen Betriebssysteme.

## 2.4 Progressive Web App

### 2.4.1 Definition

Um sich eine Progressive Web App zu installieren, öffnet der Benutzer zuerst den Standardbrowser seines mobilen Endgeräts. Bei iOS ist dies der Safari Browser, bei Android der Google Chrome Browser. Danach gibt der Nutzer die Adresse ein, von der er sich die PWA herunterladen möchte. Daraufhin kann sich der Nutzer, über den *Add to Homescreen* (Zum Home-Bildschirm) Menüpunkt des Browsers, die App herunterladen. Es ist nicht nötig erst den Appstore zu besuchen und dort die App zu downloaden, wie es bei nativen und hybrid entwickelten Apps der Fall ist. Abbildung 2 gibt noch einmal einen Überblick der durchzuführenden Schritte um sich eine PWA zu downloaden.

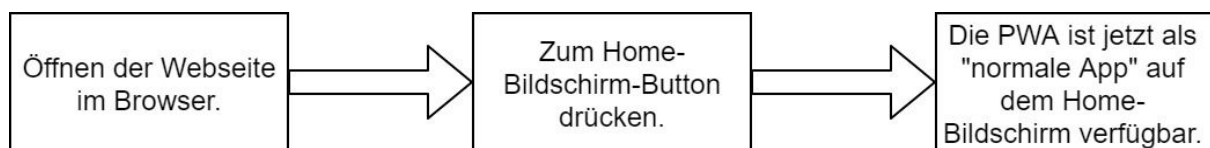


Abbildung 2: Eine PWA herunterladen.

Bei einer PWA, handelt es sich immer um eine webbasierte Applikation. Es ist eine Mischung aus einer Web App und einer hybriden App. Web App in dem Sinne, dass man die Anwendung theoretisch nur im Browser als gewöhnliche Web Anwendung nutzen kann. Downloadet man sich die App, fühlt sie sich jedoch eher wie eine hybride App an. Auch die Technologie hinter der PWA ist das gleiche HTML, CSS und Javascript wie bei einer hybriden App.

<sup>5</sup> <https://cordova.apache.org/>

<sup>6</sup> <https://ionicframework.com/>

Die PWA funktioniert also einerseits online im Browser, andererseits auch offline nachdem sie gedownloadet wurde. Offline funktioniert sie nun wie eine gängige Anwendung aus dem App Store. Die PWA bietet Features wie Push Notifications oder Background Synchronization, welche in der Browser basierten Web App nicht möglich wären.

Eine allgemeinere Definition der PWA bietet Samsung<sup>7</sup>:

Progressive Web Apps (PWAs) sind normale mobile und Desktop basierte Webanwendungen, die durch jeden Webbrowser aufrufbar sind. In Browsern mit neuen Webstandards (eingeschlossen Samsungs Internet for Android), können auch zusätzliche Funktionen wie offline Support oder Push Notifications angeboten werden.

## 2.4.2 Charakteristik und Bedeutung

Nach dem Release von progressiven Web Apps hat Google diese mit den drei grundlegenden Charakteristiken **Reliable, Fast und Engaging** beschrieben (Progressive Web Apps. A new way to deliver amazing user experiences on the web, n.d). Im Folgenden wird auf die Bedeutung dieser drei Eigenschaften genauer eingegangen.

**Reliable (zuverlässig):** Eine Progressive Web App soll sofort laden und Inhalte anzeigen unabhängig davon, ob das Smartphone aktuell mit dem Internet verbunden ist. Dank der offline Funktionalität existiert zu keiner Zeit der Fall, dass keine Inhalte angezeigt werden.

Die durchschnittliche Downtime einer Webseite betrug 2011 in Europa noch jährlich 14 Stunden (CA Technologies, 2011). Ist ein Service offline und somit nicht erreichbar, können Mitarbeiter nicht ihren geplanten Arbeiten nachgehen. Das schränkt die Produktivität des Unternehmens ein und führt damit indirekt zu weniger Gesamtumsatz. Das Unternehmen Gartner (2016) hat den wirtschaftlichen Schaden ausgerechnet der bei einem Netzerkausfall droht. Der durchschnittliche wirtschaftliche Schaden lag bei 5600\$ Dollar pro Minute.

Durch das offline Caching kann die PWA unabhängig von jeder Internetverbindung durchgängig genutzt werden, sodass hierbei kein finanzieller Schaden entstehen kann. Die Mitarbeiter sind auch ohne Internet in der Lage die App zu nutzen und ganz gewohnt ihrer Arbeit mit der App nachzugehen.

**Fast (Schnell):** Eine PWA soll schnell laden und langsame Animationen und schwerfälliges Scrollen soll vermieden werden.

Wie wichtig dem Nutzer das schnelle Laden einer mobilen Applikation ist, zeigt das Beispiel von Pinterest aus dem Jahr 2015:

Pinterest startet ein Experiment um herauszufinden wie sich verbesserte Performance auf das Nutzerverhalten auswirkt. Durch Bereitstellen vorgerenderter HTML Seiten und ohne das Nutzen interner template rendering engines, gelang ein Performance Anstieg der mobilen Landing Page um 60%. Dieser Performance Anstieg steigerte die Anzahl an

---

<sup>7</sup> <https://samsunginter.net/docs/progressive-web-apps>

Anmeldungen um 40%. Infolge dessen startete Pinterest ein großes restructuring und rewriting des gesamten Frontends.

Auch Google zeigte bereits 2009 den Einfluss von Performance auf das dauerhafte Nutzerverhalten durch die *Speed Matters* Studie (Google, 2009). Bei einer Gruppe von Google Nutzern wurde die Zeit von einer Suchanfrage bis zum Anzeigen der Suchergebnisse um 100 bis 400 ms, je nach Experiment, erhöht. Google konnte zeigen, dass dadurch die Suchanfragen pro Nutzer signifikant zurückgingen (-0,1% bis -0,6%). Auch über mehrere Wochen hinweg erholte sich die Anzahl der Suchanfragen nicht. Im Gegenteil, die Nutzer suchten sogar noch weniger als zuvor. Bei einem Delay von 200 ms ergab sich ein Rückgang an Suchanfragen von 0.22%. Nach vier Wochen lag der Rückgang bereits bei 0,44%. Selbst nachdem Google die Suchanfragen wieder in normaler Geschwindigkeit anzeigte, dauert es Wochen, bis Nutzer sich wieder ihrem ursprünglichen Suchverhalten annäherten.

Eine weitere Studie von *DoubleClick* im Auftrag von *Google* zeigt den “Need for Mobile Speed” (2016). *DoubleClick* konnte zeigen, dass 53% der mobile Nutzer, eine Seite verlassen, die länger als 3 Sekunden zu laden benötigt.

Die durchschnittliche Ladezeit einer mobilen Seite innerhalb eines 3G Netzwerkes beträgt 19 Sekunden (Google, 2016). Bei einem 4G Netzwerk 15,3 Sekunden (Google, 2018a).

Betrachtet man die beiden Fakten, dass Nutzer einerseits langsam ladende Webseiten eher verlassen und die durchschnittliche Ladezeit von mobilen Webseiten enorm hoch ist, wird klar, dass man beim Entwickeln von mobilen Anwendungen auf Offline Caching nicht verzichten sollte.

Dank der Offline Funktionalität bei PWAs, ist gewährleistet, dass zu jeder Zeit dem Nutzer Inhalte präsentiert werden können. Geht der Nutzer dann wieder online, kann der gecachte Inhalt aktualisiert werden und der Nutzer bekommt die aktuellen Daten der Anwendung angezeigt.

**Engaging:** Progressive Web Apps können installiert werden, ohne dass man sie über den App Store herunterladen muss. PWAs können im Vollbild dargestellt werden und vermitteln so den “look & feel” einer nativen App. Durch Push-Benachrichtigungen kann die Anwendung mit dem Nutzer in Verbindung treten auch wenn die Anwendung eigentlich geschlossen ist. Die Bedeutung von Push-Notifications bei Anwendungen zeigt eine Studie von Kahuna (2016). Hier konnte gezeigt werden, dass bei Nutzern die Push-Benachrichtigungen aktiviert haben, durchschnittlich eine doppelt so hohe Bindung zur App besteht, als bei Benutzern ohne aktivierte Push-Benachrichtigungen. Außerdem konnte die Nutzerquote der Nutzer mit aktivierten Push-Benachrichtigungen signifikant gesteigert werden. Die 30-Tage App Nutzerquote wurde um 125% erhöht, die 60 Tage Nutzerquote um 150% und die 90 Tage Nutzerquote um 180% (Kahuna, 2016).

Ähnlich zu den oben genannten, von Google aufgestellten Anforderungen, definiert Russel (2016) Anforderungen an eine Anwendung um diese als PWA bezeichnen zu können.

- Die App sollte sofort laden, unabhängig des Netzwerk Status. Damit ist nicht gemeint, dass die Anwendung komplett offline funktionieren muss, aber es sollte immer ein eigenes UI zur Verfügung stehen.
- Dem Nutzer sollte immer klar sein “woher er kommt”. Die Marke beziehungsweise Seite hinter der App sollte kein Mysterium sein.
- Die App soll ohne extra Browser funktionieren (z.B. ohne URL bar).

## 2.5 Abgrenzung PWA - App Typen im Vergleich

Abbildung 3 zeigt die drei App Typen. Links die native App mit direktem Zugriff auf die APIs des Geräts. In der Mitte die Web App die komplett im Browser läuft. Und rechts die Hybride App die in einem nativen Container läuft der wiederum Zugriff auf die API des Geräts hat.

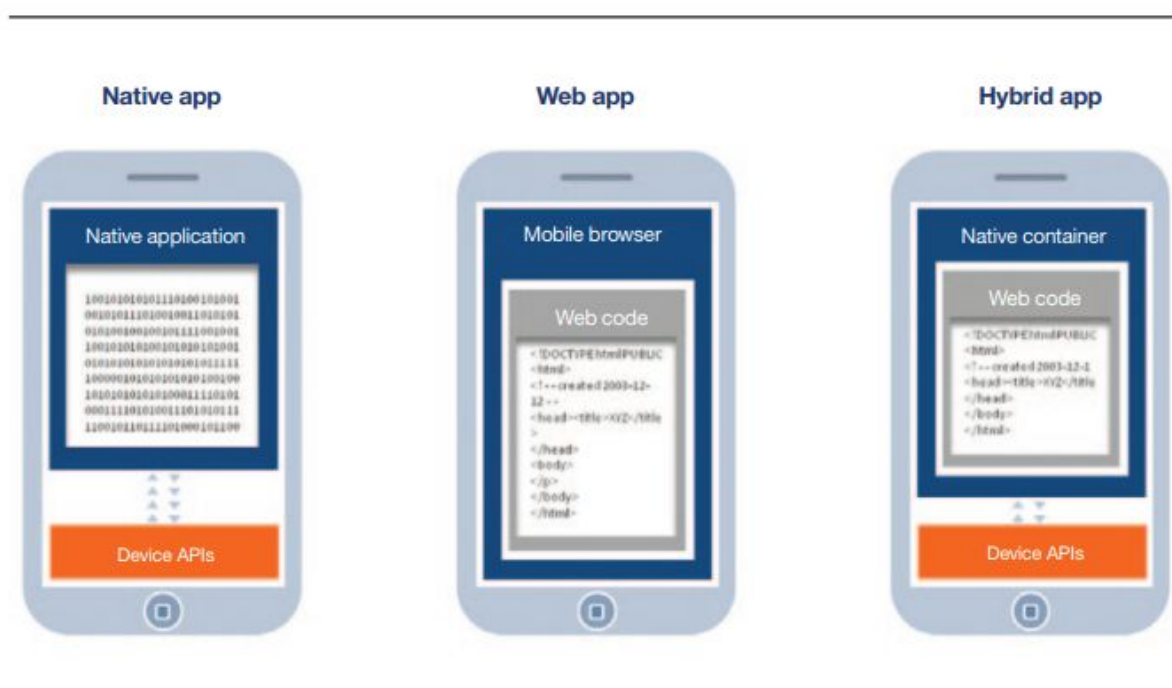


Abbildung 3: Native App, Web App, Hybride App (1) (IBM Corporation, 2012).

Alle drei (Web Apps werden ausgeklammert, da diese nicht als “klassische App” gelten) Apps haben für ihren jeweiligen Use Case ihre Daseinsberechtigung. Grob gesagt sollte man für hardwarenahe Programmierung, wie Spiele und grafisch aufwändige Anwendungen eher nativ programmieren. Will man aber eine “Single Code Base”, und möchte mit so wenig Aufwand wie möglich für alle Plattformen gleichzeitig programmieren sollte man eine PWA beziehungsweise eine Hybrid App entwickeln.

Detaillierte Vergleiche der verschiedenen App Typen wurden in der Vergangenheit bereits einige durchgeführt. Hierbei ist z. B. die Arbeit von Biørn-Hansen, Majchrzak & Grønli (2017) zu nennen. Sie haben ein und dieselbe App auf drei verschiedene Art und Weisen

implementiert. Die native beziehungsweise die interpretierte App wurde mit Hilfe von React Native<sup>8</sup> implementiert. React Nativ (RN) ist ein von Facebook entwickeltes JavaScript Framework. Dabei benutzt RN jedoch nicht die HTML5 UI Elemente, sondern nutzt die tatsächlichen iOS beziehungsweise Android spezifischen UI Elemente. JavaScript dient dabei nur als Beschreibung wie diese nativen Elemente zusammengesetzt werden. Die Hybride App wurde mit Hilfe des IONIC Frameworks entwickelt. IONIC wird mit Hilfe von Angular / JavaScript, HTML5 und CSS entwickelt. Anschließend wird der Code durch das Framework Cordova in einen Wrapper gepackt und läuft dann als Web Code in einem nativen Container. Die folgende Tabelle zeigt das Ergebnis anhand einiger getesteter Attribute.

|   | Hybrid   | Interpreted | PWA                      |
|---|----------|-------------|--------------------------|
| <b>Installationsgröße</b>   | 4.53MB   | 16.39MB     | 104KB                    |
| <b>Startzeit</b>  | 860ms    | 246ms       | 230ms                    |
| <b>Zeit vom drücken des App-Icons bis zum Rendern der Toolbar</b> | 9242.1ms | 862ms       | (a) 3152ms<br>(b) 1319ms |

Abbildung 4: Native App, Web App, Hybride App (2) (Biørn-Hansen et al. 2017).

Die Installationsgröße ist bei der PWA eindeutig am kleinsten (Abbildung 4), gefolgt von der interpretierten und der hybriden App. Bei der Startzeit der App ist die PWA auch die stärkste. Bezüglich des Renderns der Toolbar, sieht man, dass hier die App am besten performt, die auch die nativen UI Elemente verwendet.

Biørn-Hansen et al. (2017) haben auch einen generellen Vergleich der Features der App Typen aufgestellt. *Interpreted* ist hierbei die auf React Nativ basierende App, während *Native* eine tatsächlich native App ist.

---

<sup>8</sup> <https://facebook.github.io/react-native/>



| Feature                              | Interpreted | PWA       | Hybrid | Nativ |
|--------------------------------------|-------------|-----------|--------|-------|
| Installierbar                        | Ja          | Ja        | Ja     | Ja    |
| Offline-fähig                        | Ja          | Ja        | Ja     | Ja    |
| Testbar vor der Installation         | Nein        | Ja        | Nein   | Nein  |
| Verfügbarkeit auf dem App-Marktplatz | Ja          | Ja        | Ja     | Ja    |
| Push-Benachrichtigungen              | Ja          | Ja        | Ja     | Ja    |
| Cross-platform Verfügbarkeit         | Ja          | Limitiert | Ja     | Ja    |
| Hardware und Plattform API Zugriff   | Ja          | Limitiert | Ja     | Ja    |
| Hintergrundaktualisierung            | Ja          | Ja        | Ja     | Ja    |

Abbildung 5: Native App, Web App, Hybride App (3) (Biørn-Hansen, Majchrzak & Grønli, 2017).

Bei der PWA ist zu erwähnen, dass es eine limitierte Cross-plattform Verfügbarkeit gibt und der limitierte Hardware und Plattform API Zugang immer noch problematisch ist. Der API Zugang ist hierbei abhängig von HTML5 und dem Service Worker der Anwendung. Auf die Cross-plattform Funktionalität wird in späteren Kapiteln genauer eingegangen. Es ist zu sehen, dass man die PWA als einzige App vor dem Installieren bereits testen kann. Das liegt daran, dass man die Webseite zuerst durch den Browser öffnet, und sich dann entscheiden kann, ob man die Webseite als App downloaden möchte oder nicht.

Die Bedeutung, eine Webseite auch als PWA bereitzustellen, zeigt der Fall von AliExpress<sup>9</sup> (2016). AliExpress, das chinesische Pendant zu Amazon, entwickelte vor einiger Zeit eine neue PWA. Daraufhin stieg die Conversation Rate von neuen Nutzern um sagenhafte 104%. Auch die durchschnittliche Zeit, die Nutzer, über alle Browser hinweg, auf der Seite verweilen, konnte um 74% gesteigert werden. Weiterhin besuchten die Nutzer nun auch durchschnittlich doppelt so viele Seiten während einer Session. Somit zeigt die Studie, dass in diesem Fall eine PWA zu mehr Gewinn führte und den Nutzer prinzipiell länger auf der Seite bindet.

Eine ähnliche Erfolgsgeschichte kann Indiens größte e-commerce Plattform (Stand: 2015) Flipkart<sup>10</sup> im Jahr 2016 vorweisen. Flipkart entwickelten ebenfalls eine PWA für ihren Online-Shop. Die bemerkenswertesten Veränderungen waren folgende:

- Mobile Nutzer blieben durchschnittlich 3,5 Minuten auf der Webseite, vorher waren es ca. 70 Sekunden

<sup>9</sup> <https://aliexpress.com/>

<sup>10</sup> <https://www.flipkart.com/>

- Mobile Nutzer verbrachten 3 mal mehr Zeit auf der Webseite.
- Die re-engagement Rate stieg um 40%.
- Die Conversion-Rate bei Nutzern, die über den “Add to Homescreen” Button die Webseite besuchten, erhöhte sich um 70%.
- Es erfolgte eine 3-fach geringerer Datennutzung.

Auch hier kann man sagen, dass die PWA in der Gesamtheit zu mehr Verkäufen führte und den Nutzer länger auf der Seite hält.

Eine große Problematik der *klassischen App* ist der Umweg den Nutzer über den Appstore machen müssen, um sich die App herunterzuladen. Ist der Nutzer einmal im App Store verlässt er diesen oft ohne eine Conversion zu tätigen. Das zeigt eine 2018 erschienene Studie. Hierzu wurde das Verhalten von 10 Millionen Nutzer im App Store gemessen. Das Ergebnis zeigt, dass nur 26,4% der Nutzer die den App Store besuchen tatsächlich eine Conversion tätigen (Splitmetrics, 2018).

## 3 Projektmanagement & Mobile Enterprise

### 3.1 Definition Projektmanagement

Der Begriff des Projektmanagements besteht aus zwei unabhängig voneinander nutzbaren Begriffen, *Projekt* und *Management*.

Schneider & Volkmann (2013) definieren den Begriff **Projekt** wie folgt:

*„Ein Projekt ist ein Vorhaben mit definierten Zielen, definiertem Anfang und (durch Zielerreichung) definiertem Ende, mit den Merkmalen der Einmaligkeit, Komplexität und (mitunter auch) der Neuartigkeit. Ein Projekt wird in erster Linie durch die Elemente Kosten, Termine, Qualitäten und Quantitäten bestimmt. Ergänzend kommen Informationen, Verträge und Versicherungen hinzu sowie die alle Elemente übergreifende Organisation, Koordination und Dokumentation. Der Erfolg eines Projekts äußert sich in der Zielerreichung und Fortschrittskontrolle, im Management der Leistungen und der Verantwortung, Kosteneinhaltung, Termineinhaltung und Qualitätssicherung“* (p. 1).

Das Wort Management<sup>11</sup> bedeutet in diesem Kontext so etwas wie Verwaltung Organisation oder Betreuung.

### 3.2 Projektmanagement in der Baubranche

Bei Bauprojekten wird eine Unterscheidung zwischen dem Projekt und dem Objekt getroffen. Während der Architekt und Bauingenieur am konkreten Objekt arbeiten, arbeitet der

---

<sup>11</sup> <https://www.duden.de/rechtschreibung/Management#Bedeutung1/>

Prozessplaner / Projektmanager am Projekt. Ein Objekt definieren Schneider & Volkmann (2013) folgendermaßen:

*„Ein Objekt ist gekennzeichnet durch die Elemente Form, Funktion und Struktur. Dargestellt wird es in Zeichnungen, den nötigen Berechnungen (Mengen etc.) und Beschreibungen (Qualitäten) sowie dem sich anschließenden Erhalt/Unterhalt bzw. der Umnutzung des Werks (Produkts), die durch die Planung der Architekten und Ingenieure bestimmt werden. Die Erfüllung der geforderten und der vorausgesetzten Anforderungen an diese Elemente machen die Objektqualität aus. Ein Objekt ist somit die sichtbare, physisch realisierte Lösung der gestellten Anforderungen hinsichtlich der Nutzung eines Bauwerks sowie des Wegs dorthin in Form von Entwurfs-, Vertrags- und Realisierungsunterlagen“ (p.1).*

Jedes Projekt hat durch seine Einmaligkeit einen vorgegebenen Lebenszyklus.

Betrachtet man ein Bauprojekt von der Projektidee bis zur Beseitigung der baulichen Anlage besteht dieses aus folgenden 4 Phasen:

- Entwicklung,
- Realisierung,
- Nutzung,
- Verwertung

(Kochendörfer, Liebchen, & Viering, 2018, p. 5).

Schneider & Volkmann (2017) stellen den Projektablauf der Entwicklung und Realisierung in Abbildung 6 genauer dar (p. 2).

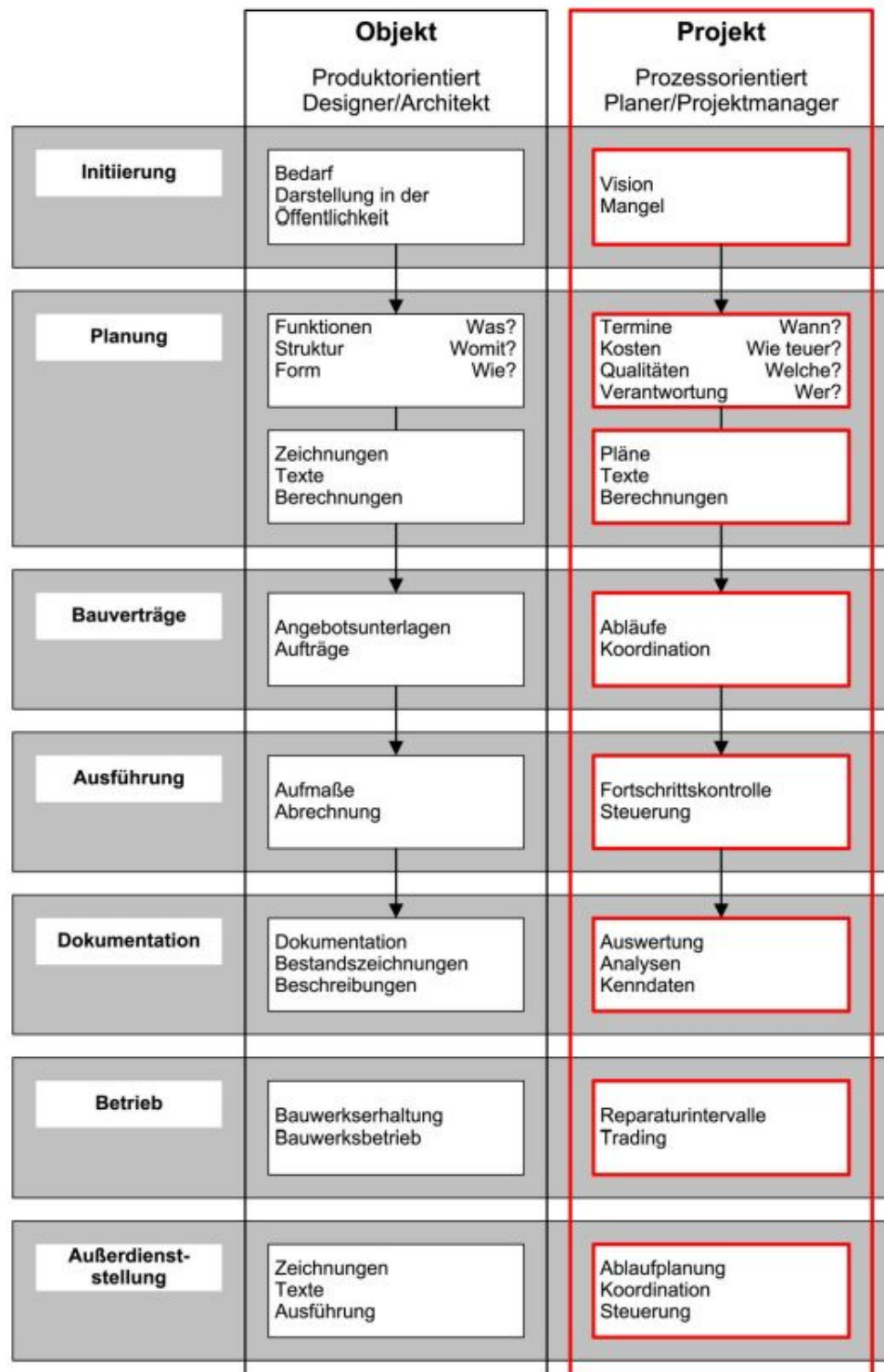


Abbildung 6: Lebenszyklus eines Bauprojekts (Schneider & Volkmann, 2017, p. 2).

Alles beginnt mit der Initiierung, der Vision eines Projekts. Daraufhin folgt die Planung mit Zeichnungen einerseits (durch den Architekten) und andererseits den Plänen des Projektmanagers. Danach werden Bauverträge abgewickelt und es kommt schließlich zu der

Durchführung des Projekts. Danach folgt die Dokumentation der Umsetzung. Ist die Dokumentation abgeschlossen, beginnt der Betrieb und schließlich die Außerdienststellung.

## 3.3 Projektmanagement Software

### 3.3.1 Definition

Projektmanagement Software ist ein Programm, welches den Projektablauf eines Unternehmens in gewissen Phasen unterstützen soll. Das kann von einfachen Aufgabenzuweisungen bis hin zu komplexen Budgetierungs-Funktionalitäten reichen.

Ahlemann (2003) definiert verschiedene Typen von Projektmanagement Systemen. Die folgenden Definitionen wurden von Ahlemann (2003) übernommen (p. 22).

#### **Plan-oriented Multi-Project Management Systems:**

Ein solches System bietet vor allem Funktionalitäten in den Bereichen der Planung und der Ressourcenplanung für Projekte. Es ist für komplexe Projekte geeignet, die detailliertes Planen benötigen.

#### **Process-oriented Multi-Project Management Systems:**

Im Gegensatz zu Plan orientierter Software liegt der Fokus hier auf Qualität und Prozessmanagement. Standardisiertes Projektmanagement mit gewährleisteter Qualitätskontrolle und Workflow Unterstützung stehen hier im Vordergrund.

#### **Resource-oriented Multi-Project Management Systems:**

Diese Art von Software fokussiert sich auf das Resource planning. Das System unterstützt bei der Vergabe von Ressourcen und der Zuweisung von Ressourcen.

#### **Enterprise Project Management Systems:**

Das System unterstützt den kompletten Produktlebenszyklus. Insbesondere beinhaltet es auch die Unterstützung für die Portfolioplanung, und das Controlling.

#### **Project Collaboration Platforms:**

Dieses System hat wenig multi-Projekt Support, bietet dafür jedoch eine webbasierte Zusammenarbeit. Zudem bietet es Funktionen wie Diskussionsmöglichkeiten in Threads, Dokumenten Management oder Shared Task.

### 3.3.2 Webbasierte Projektmanagement Systeme (WPMS)

Im Folgenden werden Webbasierte Projektmanagement Systems (WPMS) genauer betrachtet. Nach den Definitionen von Ahlemann (2003) handelt es sich hierbei um Project Collaboration Platforms.

Nitithamyong & Skibniewski (2004) sehen drei Möglichkeiten, eine WPMS einem Unternehmen bereitzustellen. Als erste Möglichkeit kann eine solche Software selbst Inhouse entworfen und entwickelt werden (Nitithamyong & Skibniewski, 2004). Die zweite Möglichkeit wäre, eine bereits bestehende Softwarelösung zu nehmen und diese Inhouse zu installieren und auf eigenen Servern zu hosten (Nitithamyong & Skibniewski, 2004). Die dritte Möglichkeit besteht darin, eine komplette Lösung bei einem Anbieter zu kaufen oder zu mieten (Nitithamyong & Skibniewski, 2004). Dabei hostet der Anbieter das komplette Tool, übernimmt die Wartung, und das Unternehmen zahlt bspw. pro Nutzer, der das System nutzt (Nitithamyong & Skibniewski, 2004). Der Anbieter solcher Software wird von Nitithamyong & Skibniewski (2004) als Project Management System-Application Service Provider (PM-ASP) bezeichnet.

Nitithamyong & Skibniewski (2004) nennen die folgenden Features, die PM-ASP bieten:

Dokumenten Management, Project Workflow, Project Directory, Central logs and revision control, Advanced searching, Conferencing and white-boarding, Online threaded discussion, Schedule and calendar, Project camera, File conversion, Printing service, Website customization, Offline access, Messaging outside the system, Wireless integration, Archiving of project information, Information service, Financial service und E-bidding and procurement.

Den konkreten Nutzen, den WPMS einem Unternehmen bringen, zeigten beispielsweise Yong, Yong & Caiyun (2012). Sie konnten deutlich machen, dass durch WPMS eine bessere Kostenkontrolle und eine bessere Projekt Koordination erreicht wurde.

### 3.3.3 Projektmanagement Software in der Baubranche

Wie in fast jedem Wirtschaftszweig werden auch in der Baubranche Geschäftsprozesse stark durch Software unterstützt. Die Anwendungen dienen der Kommunikation der beteiligten Personen innerhalb des Projekts. Sogenannte Projektkommunikationsmanagementsysteme (PKMS) bieten allen Beteiligten folgende projektunterstützende Funktionen (Kochendörfer et al., 2018, pp. 250-251).

- Austausch und Verteilung von Dokumenten (Zeichnung, Terminpläne, Protokolle und sonstige Schriftstücke) nach beliebig definierbaren Verteilerschlüsseln einschl. Protokollierung der Zeitpunkte von Einlieferung und Abholung aus den „Postfächern“
- Verwaltung und Archivierung der Dokumente mit definierbaren Zugriff für Projektbeteiligte
- Suchfunktion für Dokumente
- Versionierung von Dokumenten
- Auswertungswerkzeuge zum Vergleich von Dokumenten/ Versionen

- Unterstützen des Berichtswesens (Termin- und Kostensituation)
- Visualisierung von Bauwerksdaten (Bilder und Zeichnungen)
- Darstellung der Projektorganisation (Aufbau- und Ablauforganisation analog dem Projekt- und Organisationshandbuch) (Kochendörfer et al., 2018, pp. 250-251).

Laut Kochendörfer et al. (2018) bildet neben den oben genannten Anforderungen auch das Mängelmanagement einen Aufgabenbereich, der computergestützte Prozesse beinhaltet (p. 260). Hierzu gehören die Erfassung und die Dokumentation von Mängeln sowie deren Weiterleitung an die jeweiligen Bearbeiter und Beteiligten (Kochendörfer et al., 2018, p. 260). Weiterhin bietet ein solches unterstützendes System Informationen zum Stand der Abarbeitung und bietet die Möglichkeiten zur Terminsetzung (Kochendörfer et al., 2018, p. 260). Eine mögliche Anwendung zur Unterstützung im Mängelmanagement zeigt die App in Abbildung 7.



Abbildung 7: App zur mobilen Erfassung von Mängeln (Kochendörfer et al., 2018, p. 260).

Die Desktopansicht für eine solche Anwendung könnte folgendermaßen aussehen:

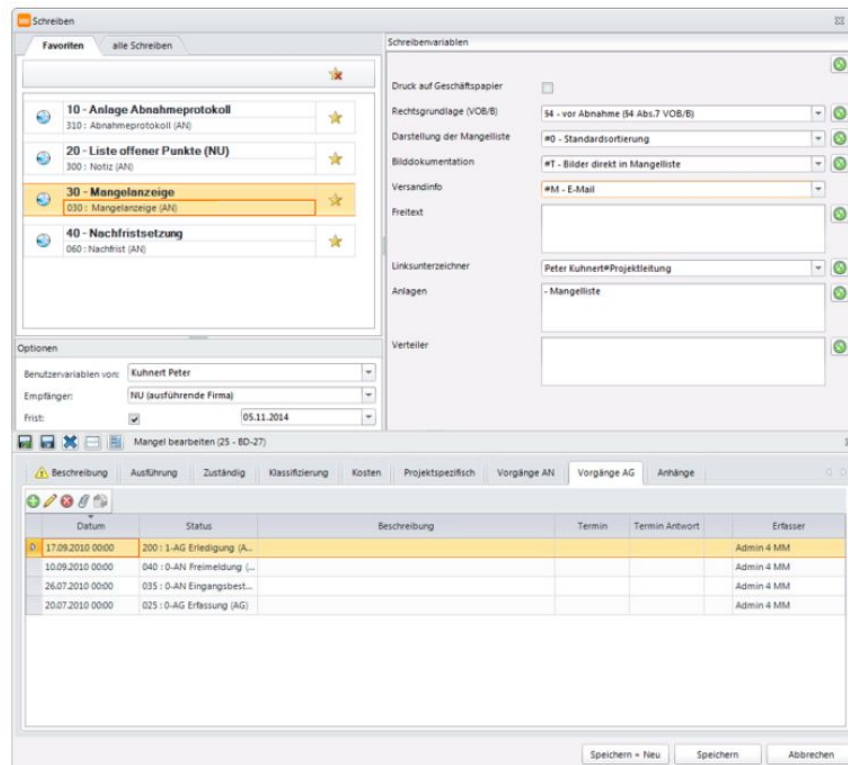


Abbildung 8: Desktop-Tool für eine Mängelmanagement Anwendung (Kochendörfer et al., 2018, p. 261).

### 3.4 Mobile enterprise

Unter mobile enterprise versteht man die Unterstützung von Geschäftsprozessen durch den Einsatz von mobilen Endgeräten. Das führt im besten Fall zu Performance Steigerung innerhalb des Prozesses (siehe Abbildung 9).

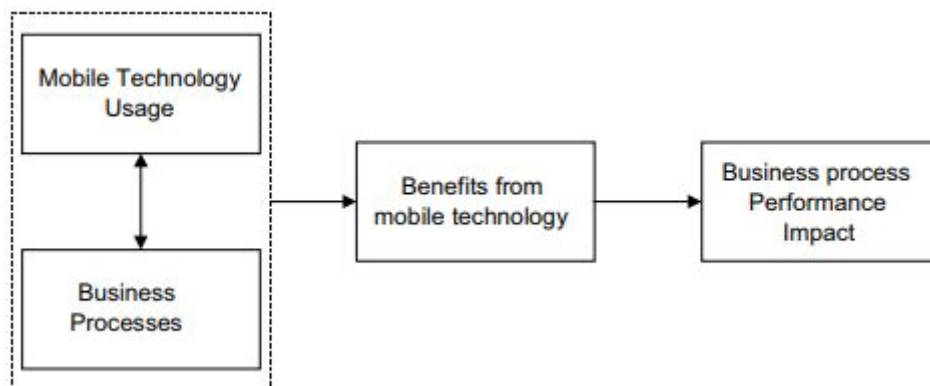


Abbildung 9: Der Nutzen von mobiler Technologie in einem Geschäftsprozess (van der Heijden & Valiente, 2002).



Mobile Enterprise kann in jeder Phase des Projektmanagements (Kapitel 3.2) stattfinden. Sofern es in den Businessprozess integrierbar ist und diesem messbaren Nutzen bringt, hat es einen positiven Einfluss auf die Performance des gesamten Projekts.

## 4 OpenProject. Collaborative Project Management

### 4.1 Was ist OpenProject?

OpenProject (OP) ist eine grundsätzlich Open Source verfügbare Projektmanagement Software. Die Anwendung ist webbasiert und für die Anwendung am Computer optimiert. Die Software deckt von Anfang bis Ende alle Projektmanagement Prozesse ab: Konzeption und Zielsetzung, Planung und Durchführung eines Projekts, Steuerung und Kontrolle sowie Abschluss und Dokumentation. Die folgende Prozessbereiche wurden von *Openproject.org* übernommen.

**Konzeption und Zielsetzung:** OpenProject bietet die Möglichkeit für einzelne Projekte verschiedenen Work packages zu erstellen. Ein Work package kann verschiedene Typen haben: Task, Milestone, Phase, Feature, Epic, user Story, Bug. So können Aufgaben an verschiedene Nutzer verteilt werden, Bugs gekennzeichnet, und Ideen festgehalten werden.

**Planung des Projekts:** Durch Gantt Charts können Meilensteine und Arbeitspakete visualisiert dargestellt werden.

**Durchführung des Projektes:** Es besteht die Möglichkeit, Arbeitspakete zu kommentieren und so anderen Nutzern zusätzliche Informationen zu übermitteln. Es gibt bei jedem Work package einen Activity Reiter, der anzeigt, wann und was verändert wurde.

**Steuerung und Kontrolle:** OpenProject bietet die Möglichkeit, Kosten und Zeit für einzelnen Work packages zu tracken.

**Projektabschluss:** Am Ende eines Projektes ist es möglich, sich noch einmal die Projektergebnisse zusammengefasst anzeigen zu lassen. Wie viel hat es gekostet, wie lange hat es gedauert, etc..

OpenProject unterstützt den Nutzer in jeder Phase eines Projektablaufs. Sowohl in der Planungsphase und während der Durchführung, als auch nach Abschluss des Projekts. OP bietet die Möglichkeit, mit mehreren Nutzern zeitgleich und unabhängig voneinander an einem Projekt webbasiert zu arbeiten. Nimmt man daher die Software Kategorien aus

Kapitel 3.3.1 zu Hilfe, so fällt OpenProject unter die Kategorie Project Collaboration Platforms.

OpenProject gibt es als OpenSource<sup>12</sup> Variante, die jedes Unternehmen kostenlos intern hosten und verwenden kann (Stand: 15.08.2018). Hierzu genügt es, sich ein Docker Image<sup>13</sup> herunterzuladen und dieses auf seinem Server zum Laufen bringen.

Es gibt OpenProject aber auch als Project Management System-Application Service Provider (siehe Kapitel 3.3.2). Hierbei wird das Hosting gegen eine monatliche Gebühr übernommen (Stand: 15.08.2018).

Im Folgenden Kapitel werden die wichtigsten Funktionalitäten von OpenProject dargestellt (Stand: 15.08.2018).

## 4.2 Funktionalität

### 4.2.1 Projects

Um die verschiedenen Funktionen von OpenProject zu nutzen, muss zunächst ein Projekt erstellt werden. Dafür kann vom Dashboard (Abbildung 10) aus über “+ Project” (linke Abbildungshälfte) ein neues Projekt angelegt werden. Alternativ geht dies auch über die Topnavigation (Abbildung 10, links oben). Dazu klick man auf “select project” und dann auf “+ Project”.

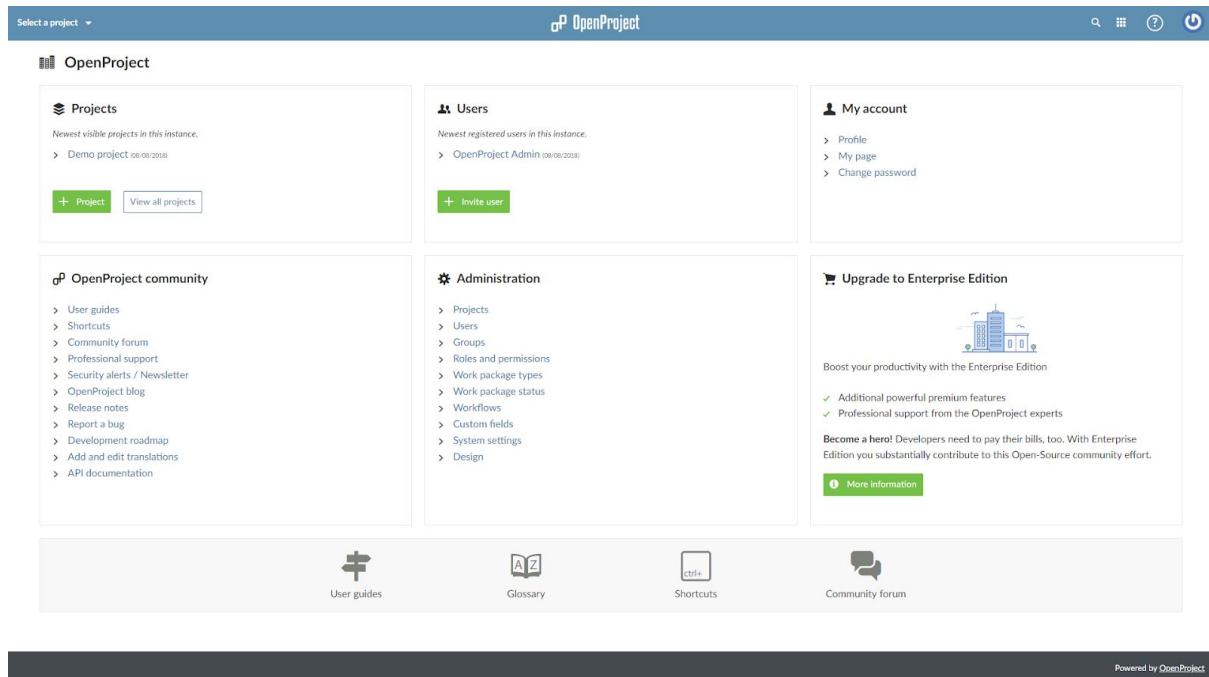


Abbildung 10: OpenProject - Dashboard.

<sup>12</sup> <https://github.com/opf/openproject>

<sup>13</sup> <https://docs.docker.com/get-started/>

Anschließend wird der Projekttitel eingegeben (Abbildung 11). Zusätzlich kann man noch Advanced settings wie *Description* oder *Project Type* angeben. Danach ist das Projekt erstellt und es können Work packages für das erstellte Projekt angelegt werden.

Nachdem das Projekt erstellt ist, besteht außerdem die Möglichkeit, Nutzer mit verschiedenen Rechten dem Projekt zuzuweisen.

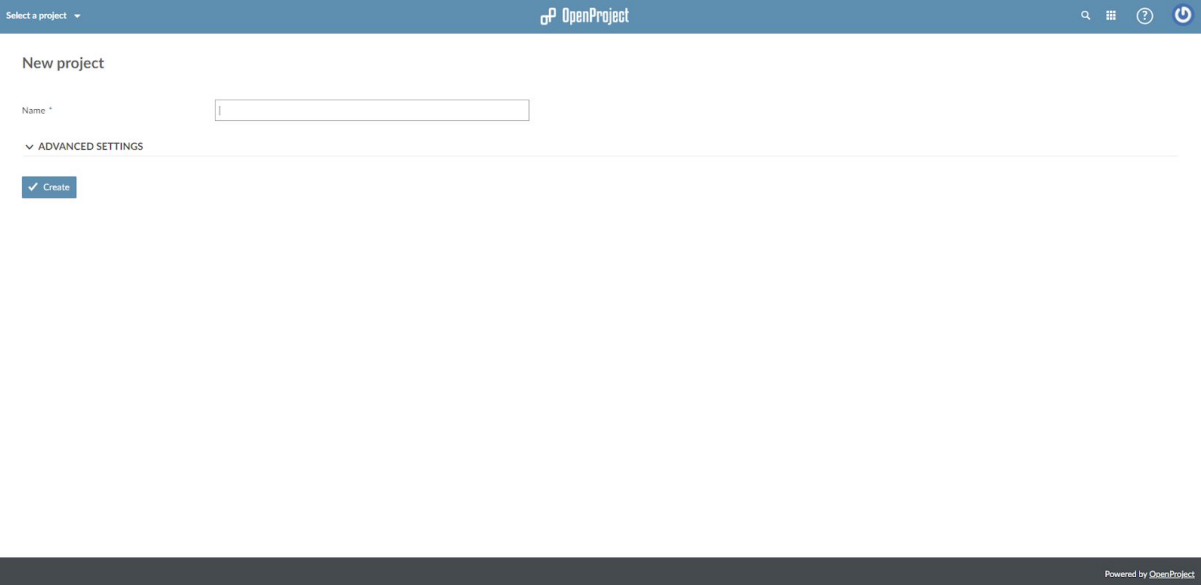
The screenshot shows the 'New project' form in the OpenProject application. At the top, there is a blue header bar with the text 'Select a project' on the left, the 'OpenProject' logo in the center, and search, grid, help, and power icons on the right. Below the header, the form is titled 'New project'. It features a 'Name' label followed by a text input field containing the letter 'I'. Below the input field is a section titled 'ADVANCED SETTINGS' with a downward arrow icon. At the bottom of the form is a blue button with a checkmark icon and the text 'Create'. The footer of the application is a dark grey bar with the text 'Powered by OpenProject' on the right.

Abbildung 11: OpenProject - Create New Project.

### 4.2.2 Work packages

Das wohl wichtigste Feature ist das Erstellen des bereits erwähnte Work packages.

Hierzu wird in der linken Navigationsleiste der Reiter Work packages ausgewählt. Anschließend wird über den grünen "+ create"-Button (Abbildung 10, oben rechts) das Work package angelegt.

Nach dem Klick auf den Button öffnet sich im rechten Teil der Anwendung ein Fenster zum Eingeben der Work package Attribute. Das können zum Beispiel Attribute bezüglich beteiligter Personen oder bezüglich der Kosten des Work packages sein.

**TestProjekt** OpenProject

**Work packages**

**New Task**

Enter subject here

DESCRIPTION

Attach and link files by dropping on this field, or pasting from the clipboard.

PEOPLE

Assignee: [dropdown] Responsible: [dropdown]

ESTIMATES AND TIME

Estimated time: [input] Remaining Hours: [input]

DETAILS

Category: [dropdown] Priority: Normal [dropdown]

Date: [input] Version: [dropdown]

Progress (%): [input]

COSTS

Budget: [dropdown]

Drop files here or click to add files

Save Cancel

Abbildung 12: OpenProject - Create Work package.

Es besteht die Option, sich alle Work packages eines Projekts listenartig anzeigen zu lassen (Abbildung 13). Hierfür ist es nur notwendig, in der Navigationsleiste (Abbildung 12, links) auf den Menüpunkt "Work packages" zu klicken.

Klickt man dann auf eines dieser Work packages, gelangt man zu dessen Detailansicht (Abbildung 14). In der Detailansicht ist es möglich, die Daten des Work packages zu bearbeiten und zu speichern.

**TestProjekt** OpenProject

**Work packages**

| ID | SUBJECT | TYPE      | STATUS | ASSIGNEE | UPDATED ON         |
|----|---------|-----------|--------|----------|--------------------|
| 10 | WP3     | Task      | New    | -        | 08/08/2018 2:18 PM |
| 11 | WP 4    | Epic      | New    | -        | 08/08/2018 2:19 PM |
| 8  | WP1     | Task      | New    | -        | 08/08/2018 2:18 PM |
| 9  | WP2     | Milestone | New    | -        | 08/08/2018 2:18 PM |

[1 - 4/4]

Abbildung 13: OpenProject - Work package Liste.

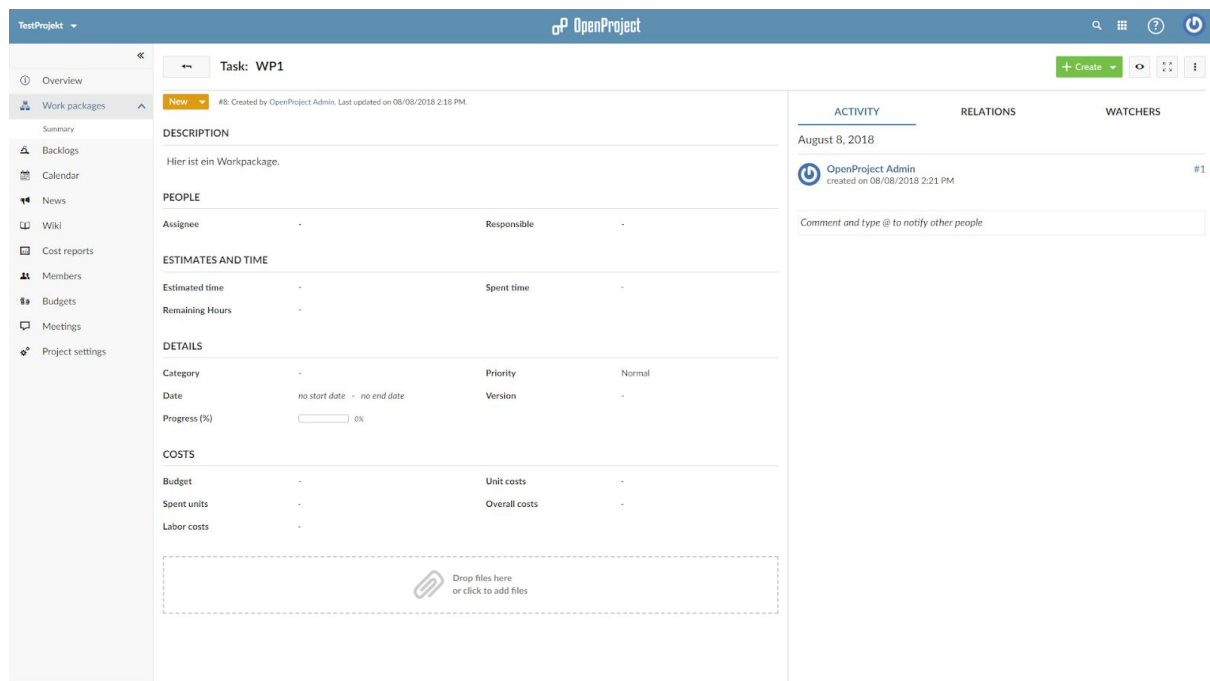


Abbildung 14: OpenProject - Work package detail.

## 5 Anforderungsanalyse & konzeptioneller Entwurf

### 5.1 Zielgruppe

Die Hauptzielgruppe der App sind Unternehmen aus der Baubranche. Hierbei kann die Anwendung vor allem von den projektverantwortlichen Bauleitern genutzt werden. Die Bauleiter können die Apps während des Projektablaufs im Mängelmanagement (siehe Kapitel 3.2) einsetzen.

Im Grunde kann jedoch jeder Nutzer von OpenProject auch die PWA verwenden. Für alle, die eine komprimierte Version von OpenProject mobil nutzen möchten kann die PWA relevant sein.

### 5.2 Anwendungsfall

#### 5.2.1 Einordnung

Die Anwendung wurde als Progressive Web App (Kapitel 2.4) für die bestehenden OpenProject (Kapitel 4) Webplattform entwickelt. Sie bietet verschiedene Features, die prinzipiell von jedem OpenProject Nutzer verwendet werden können.

Die Anwendung fällt in den Bereich des Mobile Enterprise (Kapitel 3.4) und dient zur Optimierung der Geschäftsprozesse und der damit verbundenen Effizienzsteigerung.

Speziell dient die Anwendung der Prozessoptimierung der *Ausführung* beziehungsweise der *Durchführung* (Kapitel 3.2, Abbildung 6) des Projekts.

Die App unterstützt Anwender bei der Fortschrittskontrolle und dem Mängelmanagement (Kapitel 3.3.3). In dieser Phase des Projekts können Mängel, Ideen oder Aufgaben einfach über die App in ein Projektmanagement Tool eingebunden werden. Das Projektmanagement Tool wiederum unterstützt das Projekt über den gesamten Lebenszyklus hinweg.

Die PWA kann für die Open Source Variante von OpenProject genutzt werden, kann aber später aber auch als Komplettservice innerhalb eines webbasierten Projektmanagement System (Kapitel 3.3.2) als Dienstleistung bereitgestellt werden.

Da die App international genutzt werden soll, wurde sie komplett in Englisch umgesetzt.

## 5.2.2 Beispielszenario

Die Anwendung könnte in folgendem Beispielszenario eingesetzt werden.

Das Bauunternehmen B&C ist dabei eine neue Fabrikanlage zu bauen und befindet sich mitten in der Ausführung des Projekts. Der Bauleiter läuft über die Baustelle und kontrolliert den aktuellen Stand, indem er sorgfältig das ganze Gelände untersucht. Ihm fällt auf, dass im Werkraum 1 die Wände nicht richtig verputzt wurden. Er öffnet daraufhin die App, wählt die Funktion "Neues Work package erstellen" und beginnt mit der Dokumentation des Mangels. Er trägt als *Title* "Verputzen des Werkraums 1" ein. Als Beschreibung gibt er genau an, was falsch gemacht wurde, und wie es behoben werden sollte. Weiterhin gibt er das Due Date an, d.h. das Datum, an dem das Problem behoben sein muss. Er macht mit der Smartphonekamera mehrere Bilder von den konkreten Mängeln und fügt diese dem Work package hinzu.

Der Bauleiter klickt auf "Work package erstellen", stellt nun aber fest, dass er auf dem Fabrikgelände kein mobiles Internet hat. Die App gibt ihm Feedback, dass das Work package erst gesendet werden kann, wenn Internetverbindung besteht. Der Bauleiter erkennt in der tabellarischen Ansicht, dass das gerade erstellte Work package bisher nur lokal gespeichert wurde. Andere Work packages die er in der Vergangenheit erstellte, liegen bereits auf dem Server und sind dementsprechend gekennzeichnet. Nachdem der Bauleiter, über die Baustelle verteilt, weitere Mängel in Form von Work packages dokumentiert hat, geht er schließlich zu seinem Baustellencontainer zurück und öffnet dort erneut die Anwendung. Dank des dort vorhandenen Wlans werden die Work packages nun gesendet. Die App teilt ihm mit, dass alle 5 Work packages erfolgreich gesendet wurden. In der tabellarischen Ansicht sieht der Bauleiter, dass alle Work packages inzwischen auf dem Server gespeichert sind. Um seine Angaben zu kontrollieren, klickt er auf das Work package "Verputzen des Werkraums 1" und gelangt so zur Detailansicht des Work packages. Er prüft alle Angaben und überprüft zudem, ob die gespeicherten Koordinaten korrekt sind. Sind die Koordinaten korrekt, könnte er diese nun beispielsweise für externe 3D-Modelle der Baustelle nutzen.

Der Bauleiter loggt sich über seinen Laptop bei der Web App von OpenProject ein und findet dort seine erstellten Work packages innerhalb des gewünschten Projekts.

Folgendes BPMN-Diagramm (Geschäftsprozessmodell und -notation) stellt einen möglichen, allgemeineren Prozess dar, bei dem die PWA im Bereich des Mängelmanagements zum Einsatz kommen könnte.

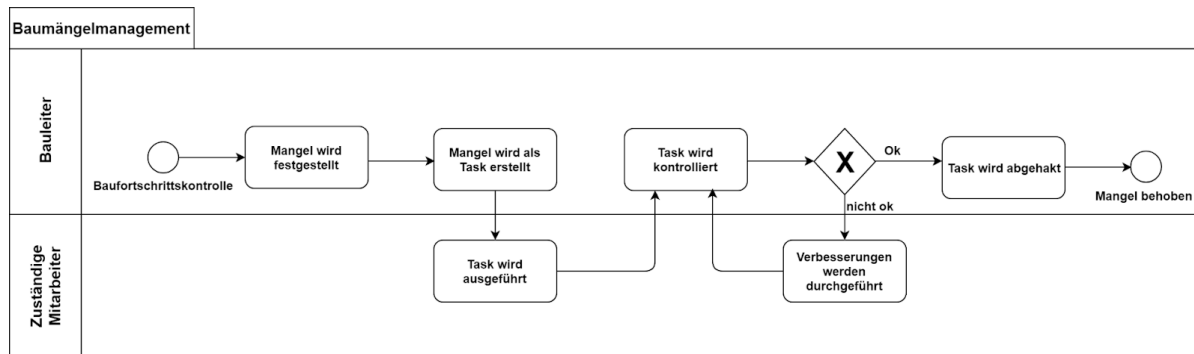


Abbildung 15: BPMN - Mängelmanagement.

Der in Abbildung 15 gezeigte Prozess findet während der Durchführung eines Bauprozesses (Kapitel 3.2) statt und befindet sich innerhalb des Teilprozesses des Mängelmanagements (Kapitel 3.3.3).

Der Prozess beginnt damit, dass der Bauleiter die routinemäßige, zeitlich determinierte Baufortschrittskontrolle angeht. Findet der Bauleiter einen Mangel auf der Baustelle, wird dieser mit Hilfe der PWA als Task in OpenProject erstellt. Zuständige Mitarbeiter führen diese Task dann aus, und markieren die Task daraufhin als bearbeitet. Entweder sie ändern direkt den Status der Task durch die OpenProject Webanwendung, oder sie geben dem Bauleiter über andere Wege Mitteilung, dass die Task bearbeitet wurde. Der Bauleiter kontrolliert nun die durchgeführten Arbeiten auf der Baustelle. Sind immer noch Mängel vorhanden, führen die zuständigen Mitarbeiter weitere Verbesserungen durch. Wurde jedoch alles zur Zufriedenheit des Bauleiters ausgeführt, ändert dieser den Status der Task innerhalb der OpenProject Webanwendung und der Mangel gilt als behoben.

## 5.3 Entwurf der Progressiven Web App

Das folgende Use Case Diagramm gibt einen Überblick der funktionalen Anforderungen, die an die PWA gestellt werden.

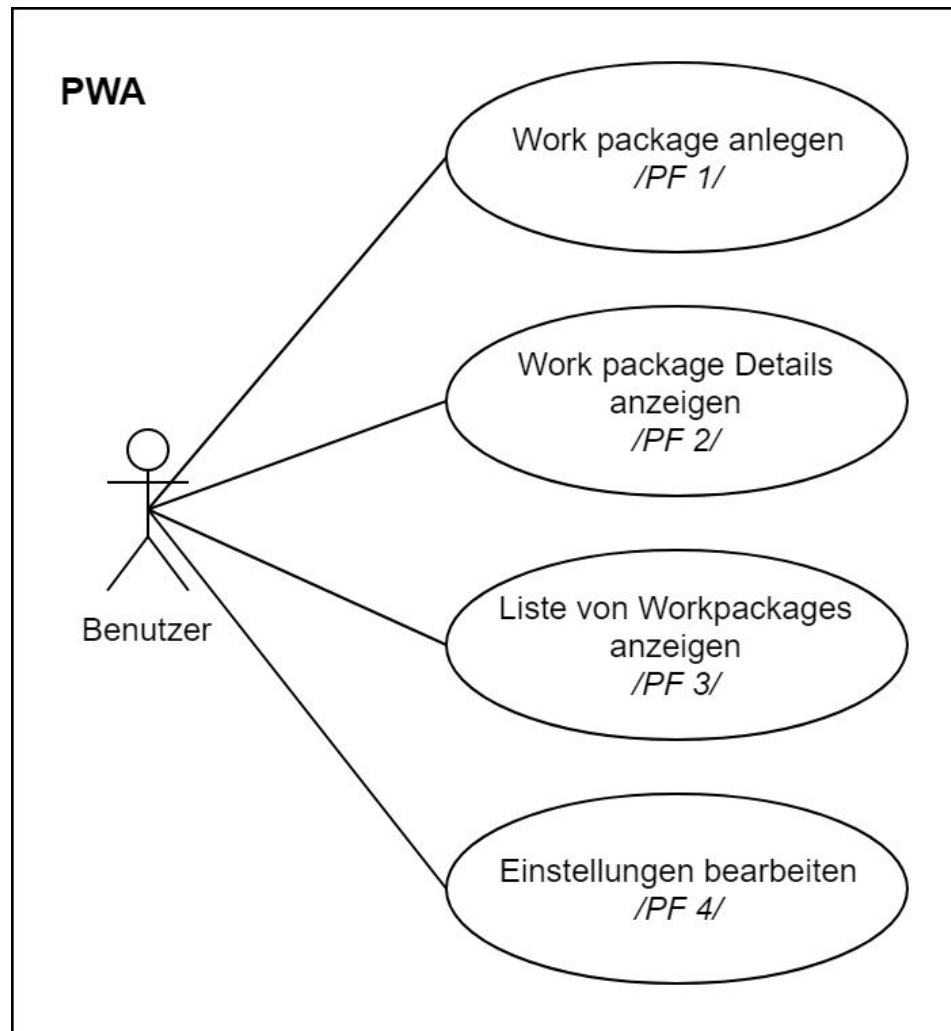


Abbildung 16: Use Case - PWA Produktfunktionen (PF).

Der Nutzer kann sich über die App ein Work package neu anlegen. Diese Funktionalität soll sich an der Funktionalität der webbasierten *OpenProject* Anwendung orientieren (Kapitel 4.2.2).

Danach kann er sich die Details des Work package anzeigen lassen. Auch diesen View gibt es in der *originalen* OpenProject Anwendung (Kapitel 4.2.2).

Ein weiterer View zeigt eine Liste aller bestehenden Work packages (Vergleich, Kapitel 4.2.2).

Die letzte zu nennende Funktion soll die Möglichkeit bieten, seine Einstellungen innerhalb der PWA zu ändern. Innerhalb der Einstellungen soll man beispielsweise das Projekt auswählen können, in dem man arbeiten möchte.



Nachfolgen die zum Use Case Diagramm gehörenden Tabellen.

|                                 |  |
|---------------------------------|--|
| <b>Ziel</b>                     | Ein Work package erstellen   |
| <b>Akteure</b>                  | Benutzer   |
| <b>Auslösendes Ereignis</b>     | Der Nutzer möchte eine Task, eine Idee oder eine Notiz anlegen.  |
| <b>Vorbedingung</b>             | <ol style="list-style-type: none"> <li>1. Der Nutzer muss in dem Projekt die Berechtigung zum Erstellen eines Work packages haben.</li> <li>2. Der Nutzer muss ein Projekt ausgewählt haben und seinen api key in den Einstellungen hinterlegt haben.</li> <li>3. Die Eingaben müssen den Formvalidierungsvorgaben entsprechen.</li> </ol> |
| <b>Nachbedingung Erfolg</b>     | Ein Feedback Pop-up gibt Meldung, dass das Work package gesendet werden konnte.  |
| <b>Nachbedingung Fehlschlag</b> | Ein Feedback Pop-up gibt Meldung, dass das Work package gespeichert wurde, aber erst gesendet werden kann, sobald eine Internetverbindung besteht.   |
| <b>Beschreibung</b>             | <ol style="list-style-type: none"> <li>1. Der Nutzer klickt auf den "+"-Button in der Topleiste.</li> <li>2. Der Nutzer füllt die Input Felder aus.</li> <li>3. Der Nutzer bestätigt das Erstellen über den "Create Work package"-Button..</li> </ol>  |

Abbildung 17: Produktfunktion 1 - Ein Work package erstellen.

|                             |   |
|-----------------------------|---|
| <b>Ziel</b>                 | Work package Details anzeigen   |
| <b>Akteure</b>              | Benutzer  |
| <b>Auslösendes Ereignis</b> | Der Nutzer möchte die Details eines bereits angelegten Work packages ansehen.   |
| <b>Vorbedingung</b>         | <ol style="list-style-type: none"> <li>1. Das Work package muss in der Vergangenheit von dem gleichen mobilen Endgerät erstellt worden sein.</li> <li>2. Der Nutzer muss mit dem Internet verbunden sein.</li> <li>3. Der Nutzer muss ein Projekt ausgewählt haben</li> </ol> |

|                                 |   |
|---------------------------------|---|
|                                 | und seinen api key in den Einstellungen hinterlegt haben.   |
| <b>Nachbedingung Erfolg</b>     | Der Nutzer sieht die Details des ausgewählten Work packages.  |
| <b>Nachbedingung Fehlschlag</b> | Besteht keine Internetverbindung, erhält der Nutzer Feedback über ein Pop-Up. Wurde das Work package gelöscht, erhält der Nutzer ebenfalls Information davon.   |
| <b>Beschreibung</b>             | <ol style="list-style-type: none"> <li>1. Der Nutzer klickt auf das gewünschte Work package.</li> <li>2. Der Nutzer sieht die Work package Details.</li> <li>3. Der Nutzer kann dem Work Package jetzt noch ein Attachment hinzufügen.</li> </ol> |

Abbildung 18: Produktfunktion 2 - Work package Details anzeigen.

|                                 |   |
|---------------------------------|---|
| <b>Ziel</b>                     | Liste von Work packages anzeigen  |
| <b>Akteure</b>                  | Benutzer  |
| <b>Auslösendes Ereignis</b>     | Der Nutzer möchte alle, jemals von dem mobilen Endgerät erstellten Work packages, als Liste sehen.  |
| <b>Vorbedingung</b>             | <ol style="list-style-type: none"> <li>1. Die Work packages müssen in der Vergangenheit von dem gleichen mobilen Endgerät erstellt worden sein.</li> <li>2. Der Nutzer muss ein Projekt ausgewählt haben und seinen api key in den Einstellungen hinterlegt haben.</li> </ol> |
| <b>Nachbedingung Erfolg</b>     | Der Nutzer sieht die Work packages in einer Tabelle   |
| <b>Nachbedingung Fehlschlag</b> | -   |
| <b>Beschreibung</b>             | <ol style="list-style-type: none"> <li>1. Der Nutzer öffnet die App oder klickt auf den Projektnamen/"Pfeil"-Button im Header.</li> <li>2. Der Nutzer sieht die Liste der Work packages.</li> </ol>   |

Abbildung 19: Produktfunktion 3 - Liste von Work packages anzeigen.

|                                 |  |
|---------------------------------|--|
| <b>Ziel</b>                     | Einstellungen bearbeiten   |
| <b>Akteure</b>                  | Benutzer   |
| <b>Auslösendes Ereignis</b>     | Der Nutzer möchte sein api key und/ oder das Projekt ändern.   |
| <b>Vorbedingung</b>             | -  |
| <b>Nachbedingung Erfolg</b>     | Der Nutzer sieht seine Einstellungen und kann diese bearbeiten.  |
| <b>Nachbedingung Fehlschlag</b> | Hat der Nutzer kein Internet, erhält er über ein Pop-up Feedback.  |
| <b>Beschreibung</b>             | <ol style="list-style-type: none"> <li>1. Der Nutzer klick auf das Einstellungs-Icon.</li> <li>2. Der Nutzer sieht seine Einstellungen.</li> <li>3. Der Nutzer kann jetzt Projekt und/ oder api key ändern.</li> <li>4. Zum Bestätigen klickt der Nutzer auf den "Update"-Button.</li> </ol> |

Abbildung 20: Produktfunktion 4 - Einstellungen bearbeiten.

## 5.4 Qualitätszielbestimmungen

Um einen gewissen Qualitätsstandard der Progressive Web App zu gewährleisten, wurden im Vorfeld einige nichtfunktionale Anforderungen definiert.

Zum einen wurde die PWA anhand allgemeiner, aus der Softwareentwicklung bekannter, nicht funktionaler Anforderungen entwickelt.

Zum andern wurde sich an der von Google (2018b) aufgestellten Checkliste für PWA Anforderungen orientiert.

### 5.4.1 Allgemeine nichtfunktionale Anforderungen

#### Sicherheit

Das System soll vor unautorisierten Zugriffen geschützt werden. Die Daten der Work packages sollen durch Sicherheitsmechanismen unberechtigten Benutzern nicht zugänglich sein.

#### Effizienz

Die Anwendung soll kurze Antwortzeiten liefern und dem Benutzer schnell Inhalte präsentieren.

## **Übertragbarkeit**

Die Anwendung wird plattformunabhängig als Web-Client verfügbar sein. Die Anwendung soll in den gängigen Standard Browsern mobiler Endgeräte ausführbar sein.

## **Änderbarkeit**

Die Anwendung soll durch sinnvolle Coding Strukturen die Möglichkeit bieten, ohne großen Aufwand, Änderungen an der Software durchzuführen.

## **Bedienbarkeit (Usability)**

Das System soll nach den in Kapitel 6 entsprechenden Richtlinien gestaltet werden. Die Anwendung soll intuitiv verstanden und bedient werden können

### **5.4.2 Anforderungen an eine Progressive Web App**

Die entwickelte Anwendung sollte den Richtlinien einer progressiven Web App entsprechen. Um die in Kapitel 2.4.2 genannten Anforderungen zu konkretisieren, wurde sich an die von Google (2018b) entwickelte Checkliste für eine progressive Web App orientiert. Die nachfolgenden Anforderungen sind nur die der *Baseline Progressive Web App Checklist* (Google, 2018b), auf die die PWA im späteren Verlauf geprüft wird.

- Die Webseite ist über HTTPS erreichbar.
- Die Seite ist responsive für Tablets und mobile Geräte.
- Alle App URLs sollen offline geladen werden können.
- Add to Home screen muss verfügbar sein.
- Schnelles erstmaliges Laden der Anwendung auch bei einem 3G Netz.
- Die Seite sollte in verschiedenen Browsern laufen (Chrome und Safari).
- Seitenübergänge sollten sich nicht so anfühlen, als ob sie das Netzwerk blockieren.
- Jede Seite hat eine eigene URL.

# 6 Usability Richtlinien

## 6.1 Definition Usability

Die International Organization for Standardization (ISO) definiert den Begriff der Usability folgendermaßen<sup>14</sup>:

*„The effectiveness, efficiency and satisfaction with which specified users achieve specified goals in particular environments.*

*effectiveness: the accuracy and completeness with which specified users can achieve specified goals in particular environments*

*efficiency: the resources expended in relation to the accuracy and completeness of goals achieved*

*satisfaction: the comfort and acceptability of the work system to its users and other people affected by its use”*

Übersetzen lässt sich dies folgendermaßen:

die Effektivität, Effizienz und Zufriedenheit in der ein bestimmter Nutzer bestimmte Ziele in einer speziellen Umgebung erreicht.

Effektivität bezeichnet hierbei die Genauigkeit und Vollständigkeit, mit der bestimmte Benutzer bestimmte Ziele in bestimmten Umgebungen erreichen können,

Effizienz bezeichnet die aufgewendeten Ressourcen in Bezug auf die Genauigkeit und Vollständigkeit der erreichten Ziele.

Zufriedenheit bezeichnet hier den Komfort und die Annehmbarkeit des work systems in Bezug auf den Nutzer und die Leute, die durch dessen Nutzung beeinflusst werden.

Betrachtet man diese Definition, sieht man, dass aus verbesserter Usability auch indirekt eine höhere Effizienz und Effektivität in dem gesamten Prozess entsteht.

## 6.2 Die Bedeutung und Probleme von Usability Richtlinien

Bereits in den 1980er Jahren begannen Forscher, sich mit dem Thema Usability auseinanderzusetzen und Usability Standards festzulegen (Reed et al., 1999). Der Grund, warum Forscher anfangen, sich mit diesem Thema zu beschäftigen, sehen Reed et al. (1999) in folgenden gesellschaftlichen Entwicklungen.

- Es gab eine schnelle Entwicklung von verschiedenen grafischen Benutzeroberflächen. Diese neuen GUIs fördern die Benutzerfreundlichkeit und erhöhen die Produktivität. Daraus folgte jedoch eine Unstimmigkeit und Inkompatibilität.

---

<sup>14</sup> <https://www.w3.org/2002/Talks/0104-usabilityprocess/slide3-0.html>

- Es wurden Standards für Hardware (Tastaturen, Bildwiederholfrquenzen, etc.) geschaffen, dabei wurden die Probleme in der Software jedoch übersehen.
- Es gab Bedenken hinsichtlich psychischer Belastung und Stress, die durch das Verwenden von Software entstehen.

Auch wenn dies in den 80er Jahren möglicherweise einige der Gründe waren, warum Guidelines entwickelt wurden, macht es den konkrete Nutzen von Guidelines noch nicht ganz klar. Reed et al. (1999) nennen für das Entwickeln solcher Standards folgende Vorteile:

- Erhöhte Produktivität
- Reduzierter mentaler und physischer Stress
- Geringere Schulungsaufgaben
- Verbesserte Interoperabilität zwischen Nutzer und Systemen
- Steigerung der gesamten Produktqualität und Ästhetik

Etwas konkreter sieht Henninger (2001) den Mehrwert von Usability Richtlinien darin, die Iterationen in dem "design-evaluate-redesign" Zyklus der HCI Entwicklung zu reduzieren, jedoch niemals komplett zu eliminieren.

Inzwischen gibt es einige solcher definierter Usability Guidelines, an denen man sich orientieren kann. Zu nennen wäre beispielsweise das komplexe "Research-Based Web Design & Usability Guidelines" (Leavitt & Shneiderman, 2006), in dem über 200 Richtlinien für gutes Design festgelegt wurden. Oder Hoehle, Aljafari & Venkatesh (2016) die beispielsweise die Microsoft Richtlinien nutzen um ein Konzept für mobile Usability Guidelines zu erstellen.

Prinzipiell ist ein striktes Befolgen von Anweisungen jedoch schwierig, da nicht jede Guideline auf jede Software anwendbar ist. Man muss immer den konkreten Anwendungsfall, die Nutzergruppe, das zu benutzende Gerät etc. beachten, um Use Case spezifische Usability zu garantieren. Ebenfalls spielt der kulturelle Hintergrund, d.h. das Land, in dem eine Anwendung veröffentlicht wird, eine Rolle. Beispielsweise sollten in amerikanischen Anwendungen Entfernungen in Feet und Inches angegeben werden, während es in Europa üblich ist, die Einheit Meter zu Nutzen (Marcus & Baumgartner, 2004). Gleiches gilt zum Beispiel auch für das Format, in dem Telefonnummern oder Adressen angegeben werden (Marcus & Baumgartner, 2004).

Gould & Lewis (1985) nehmen in ihrem "Designing for Usability: Key Principles and What Designers Think" sogar komplett Abstand von dem strikten Befolgen von Regeln. Sie gehen davon aus, dass Regeln alleine nicht ausreichen, um ein gutes Design für ein Computersystem zu erstellen. Es ist bestenfalls hilfreich für den ersten Prototyp einer Anwendung, danach sollte aber über empirisches Messen und iteratives Design das optimale Produkt erstellt werden (Gould & Lewis, 1985).

Ein grundsätzliche Problem an sehr konkreten Guidelines ist, dass sie oft plattformspezifisch sind und sich oft auf spezifische Elemente wie Dialogboxen und Screen Layouts beziehen

(Henninger, 2001). Andere Fragen, wie bspw. ein bestimmtes Widget genutzt werden soll, bleiben jedoch unbeantwortet (Henninger, 2001). Hingegen ist bei allgemeineren Designrichtlinien das Problem, dass diese oft sehr allgemeine Aussagen tätigen, wie: "Gib dem Nutzer Feedback für jede Aktion die er durchführt" (Henninger, 2001). Hierbei fehlt dem Designer, beziehungsweise dem Softwareentwickler, die Information, wie und wann er diese Richtlinie anwenden soll (Henninger, 2001).

Wie wichtig generell Usability jedoch für Anwendungen ist, konnte Nielsen 2008 zeigen. Nach einem Redesign einer Business Anwendung anhand von gewissen Usability Standards, stiegen die Key Performance Indikatoren um durchschnittlich 135%. Unter diese Indikatoren fallen Kennzahlen wie Conversion Rate, Besucherzahlen, User Performance (die Zeit, die Benutzer brauchen um entscheidende Aktionen durchzuführen) und Target Feature Usage (Nutzer, die auf einen Link klicken um zu wichtigen Informationen zu gelangen). Usability hat jedoch nicht nur Auswirkungen auf beispielsweise E-Commerce Plattformen, die direkt von dem Kunden bedient werden. Usability macht sich auch bei Anwendungen bemerkbar, die ausschließlich innerhalb eines Unternehmens von dessen Mitarbeitern genutzt werden.

Die wesentlichen Auswirkungen von Usability auf ausschließlich unternehmensintern genutzte Anwendungen sind folgende:

- Vermeidung von Kosten, die durch spätere Redesign-Maßnahmen entstehen,
- Verringerte Einarbeitungszeit der späteren Anwender,
- Erhöhte Produktivität der späteren Anwender und
- Weniger durch Benutzer verursachte Fehler (Bias & Mayhew, 2005, p.58).

Die durch unzureichende Usability entstehenden Kosten sind laut Nielsen (1997a) enorm. Schlechte Usability Methoden kosten die US Wirtschaft rund 30\$ Milliarden jährlich (Nielsen, 1997a). Konservativ geschätzt gab es 2001 einen Verlust in Höhe von 50\$ Milliarden, zurückzuführen auf schlechtes Intranet-Webdesign in Unternehmen (Nielsen, 1997a).

Im nächsten Kapitel werden verschiedenen Design- und Usability Guidelines herangezogen anhand derer die entwickelte Anwendung und die bestehende Webanwendung untersucht werden. Es wird darauf geachtet, nur anwendungsfallgerechte Usability Richtlinien zu verwenden. In Kapitel 6.4 werden allgemeine Prinzipien vorgestellt, die zwar gültig sind, aber noch keine konkreten Umsetzungsmöglichkeiten anbieten. In Kapitel 6.5 werden dann konkrete Guidelines für formularbasierte Anwendungen vorgestellt, die genaue Vorgaben machen, wie ein User Interface aussehen sollte.

## 6.3 Der Mobile First Ansatz

### 6.3.1 Definition

Entwickelt man eine Software mobile first, so entwickelt man nicht etwa erst eine Desktop optimierte Seite und versucht diese dann an ein mobiles Gerät anzupassen (Krug, 2013, p.147). Stattdessen wird beim Entwickeln darauf geachtet, eher eine reduzierte Variante der Software zu entwickeln die dafür aber perfekt auf das Smartphone und seine Größe abgestimmt ist (Krug, 2013, p.147).

### 6.3.2 Allgemeine Unterschiede zwischen Mobile und Desktop

Es wurde sich im Folgenden auf die zwei wesentlichen Unterschiede zwischen mobilen Geräten und Computern konzentriert. Die beiden Unterschiede haben enorme Auswirkungen auf diverse Usability Aspekte. Unterschiede wie Performance und Leistungsfähigkeit der Geräte wurden außer acht gelassen, da UI unabhängig davon gestaltet werden sollte.

#### **1. Größe**

Die Größe eines Smartphones beträgt zwischen 14 cm x 6,5 cm bis zu 15 cm x 7,2 cm. Ein Computer hingegen hat meist ein mindestens 5 mal größeres Display und dadurch viel mehr Platz für das Anzeigen von UI Elementen. Das führt zu veränderten Navigationsstrukturen, im Vergleich zu Desktop Anwendungen. Während man bei Desktop Anwendung meist die komplette Navigationsleiste am linken Bildschirmrand sieht, ist es auf mobilen Geräten oft nur sinnvoll, bestenfalls den Breadcrumb dauerhaft anzuzeigen. Das Menü lässt sich meist über einen Button in der Header Leiste ausklappen.

Auch die Größe der UI Elemente spielt eine wichtige Rolle. Im Gegensatz zu dem filigranen Nutzen der Maus, stehen einem Nutzer für das Smartphone nur seine durchschnittlich 16-20 mm breiten Fingerkuppen (Dandekar, Raju, & Srinivasan, 2003) zur Verfügung. Diese 16-20 mm entsprechen ungefähr 45 - 57 Pixeln und geben dadurch die Mindestgröße für "touchbare" Elemente vor.

#### **2. Bedienung**

Die Bedienung eines mobilen Geräts funktioniert über den Finger des Nutzers und nicht wie bei einem Computer über die klassische PC Maus. Ein mobiles Endgerät bietet in der Regel fünf verschiedene Touch Eingaben an. Es gibt den normalen Touch (vergleichbar mit dem linken Mausklick), den longpress Touch (vergleichbar mit dem Rechtsklick), den double Touch (vergleichbar mit dem Doppelklick), den Swipe und den Zoom. Man sollte darauf achten, all diese Möglichkeiten der Eingabe bei der Gestaltung eines UIs sinnvoll einzusetzen.



Die Positionierung der Elemente sollte auch an das Nutzerverhalten angepasst werden. So sollte immer versucht werden, die wichtigen Element im Zentrum des Display anzuzeigen, da dieses am liebsten von den Nutzern "getouchet" wird (Hoover, 2017).

## 6.4 Allgemeine Usability Prinzipien

### 6.4.1 Gestaltpsychologische Prinzipien

Bevor in den nächsten Kapiteln konkrete Guidelines für Usability vorgestellt werden, sollen hier erst einmal grundsätzlich psychologisch evaluierte Designvorgaben näher ausgeführt werden. Die Gestaltpsychologie, die bereits in den 1890er Jahren anfang sich zu entwickeln, hat heutzutage immer noch Gültigkeit. Es handelt sich hierbei um einen Bereich der Wahrnehmungspsychologie, und während sich viele Bereiche im menschlichen Leben verändern, so ändert sich der biologisch determinierte Vorgang der menschlichen Wahrnehmung letztendlich nicht.

Betrachten wir ein Objekt beginnt unser Gehirn direkt damit, das Objekt von seinem Hintergrund zu trennen und es so als eigenständige Einheit wahrzunehmen. (Mangold, 2007, p.100). Das Ganze ist ein natürlicher Prozess, der kaum unterdrückt werden kann (Mangold, 2007, p.101). Gleichzeitig nimmt man jedoch auch mehrere Objekte als eine zusammengehörige Einheit wahr. Das kann nach verschiedenen Kriterien erfolgen:

**Das Gesetz der Nähe:** Grundsätzlich nimmt man Elemente, die sich nahe beieinander befinden als zusammengehörige Gruppe wahr (Mangold, 2007, p.103). Beispielsweise werden die Buchstaben eines Wortes durch ihre Nähe (und Abgrenzung durch Leerzeichen) als zusammengehörig wahrgenommen, und ermöglichen so schnelles Lesen (Mangold, 2007, p.103).

**Das Gesetz der Ähnlichkeit:** Sich ähnelnde Objekte werden auch als Gruppe wahrgenommen (Mangold, 2007. S. 103). Ähnlichkeit kann dabei eine ähnliche Form, eine ähnliche Farbe oder eine gleichartige Bewegung sein (Mangold, 2007, p.103).

Weiterhin gilt, dass die Gruppierungstendenz in unserer Wahrnehmung umso stärker ist:

- je größer die Ähnlichkeit in einer Eigenschaft ist,
- je stärker der Kontrast in dieser gemeinsamen Eigenschaft zu anderen Objekten ist und
- je mehr gemeinsame Eigenschaften zusammengehörige Objekte aufweisen (Thesmann, 2016, p. 225).

Nützlich für Webanwendungen kann dieser Gestaltgrundsatz dann sein, wenn zum Beispiel Elemente aus gestalterischen Gründen nicht durch Nähe angeordnet werden können (Thesmann, 2016, p.226).

**Das Gesetz der guten Fortsetzung:** Zwei sich kreuzende Linien werden dennoch als zwei verschiedene Linien erkannt (Mangold, 2007, p.104). Eine stichpunktartig untereinander geschriebene Einkaufsliste wird ebenfalls als ein Objekt wahrgenommen (Mangold, 2007, p.104).

Es ist keine vollständige Konturlinie nötig, es genügt bereits wenn die zu gruppierenden Objekte auf einer gedachte Linie oder Kurve liegen. Das Gehirn ist in der Lage, eine gewisse Art der Fortsetzung von Elementen zu erkennen.

**Das Gesetz der Geschlossenheit:** Formen lassen sich leichter identifizieren wenn sie einen Umriss (eine Konturlinie) aufweisen oder durch eine Linie verbunden sind (Thesmann, 2016, p.228). Die vollständige Geschlossenheit ist nicht immer erforderlich. Erkennt unser Gehirn eine bekannte Form, und kann dies mit Hilfe erdachter Linien ergänzen, gilt das Gesetz der Geschlossenheit auch (Thesmann, 2016, p.229). Im Webdesign wird dieses Gesetz häufig in Form von Gliederung des Bildschirms mittels Farbflächen oder dem Nutzen von Konturlinien eingesetzt (Thesmann, 2016, p.229).

**Das Gesetz des gemeinsamen Schicksals:** Unser Bewusstsein fasst Figuren zusammen, die sich durch identische Bewegungen von ruhenden Objekten oder animierten Elementen, die sich anders bewegen, unterscheiden (Thesmann, 2016, p.232). Je ähnlicher das Verhalten, umso stärker die Zusammengehörigkeit (Thesmann, 2016, p.229). Die Ähnlichkeit des Verhaltens lässt sich neben Bewegung und Rotation, auch durch gemeinsames Erscheinen, Pulsieren oder Flimmern darstellen (Thesmann, 2016, p.229).

**Das Gesetz der Erfahrung:** Hierbei geht es um Elemente, die einem bereits aus der Vergangenheit bekannt sind (Thesmann, 2016, p.232). Wenn uns bereits ein Schema von einem Objekt bekannt ist, reicht es auch aus, nur Teile eines Objekts zu sehen, und es trotzdem als eben dieses Objekt zu erkennen (Thesmann, 2016, p.).

**Das Gesetz der Prägnanz:** Dieses Gesetz tritt in Kraft, wenn sich ein Element in seiner Darstellung wesentlich von den restlichen Elementen unterscheidet (Thesmann, 2016, p.233). Faktoren hierfür können Größe, Farbe, Form oder Hintergrundkontrast sein (Thesmann, 2016, p.233). Besteht ein Bild beispielsweise komplett aus Grautönen und nur ein Schriftzug, inklusive Logo ist weiß, fällt unser Blick direkt auf den Schriftzug und erkennt ihn als Einheit (Thesmann, 2016, p.233).

**Das Gesetz der Symmetrie:** Objekte die symmetrisch angeordnet sind, lassen sich leichter erkennen als Objekte, die ohne Struktur angeordnet sind (Thesmann, 2016, p.235). Hierbei versucht das Gehirn auf vertraute Formen zurückzugreifen (Thesmann, 2016, p.235). Objekte ohne erkennbare Struktur werden eher als Hintergrund interpretiert (Thesmann, 2016, p.235).

## 6.4.2 Allgemeine Heuristiken für User Interface Design

Eine gute Zusammenfassung allgemeiner Usability Prinzipien stellte Jacob Nielsen bereits 1995 auf. Die folgende Zehn-Punkte-Liste basiert auf den Prinzipien von Nielsen (1995), wurde aber mit Forschungsergebnissen und Meinungen anderer Forscher ergänzt.

### 1. Sichtbarkeit des System Status

Das System sollte dem Nutzer immer Informationen darüber geben, was gerade geschieht. Dieses Feedback sollte in angemessener Zeit erfolgen. Auch Apple definieren in ihren Guidelines (2018), dass, sobald ein Nutzer sein Smartphone dreht oder Gesten nutzt, direkt ein Feedback, für den Nutzer sichtbar, erfolgen soll.

### 2. Übereinstimmung zwischen System und der echten Welt

Das System sollte die Sprache des Nutzers "sprechen". Das gilt sowohl für textuelle Sprache, als auch für generelle Konzepte. Man sollte den Konventionen der realen Welt folgen. Der als Metapher bezeichneten Standard findet sich zum Beispiel auch in den Apple Guidelines (2018) wieder.

### 3. Benutzerkontrolle und Freiheit

Benutzer machen häufiger Fehler, weshalb es einen klar erkennbarer "emergency exit" geben sollte, der die aktuelle Aktion abbricht. Außerdem sollten die Aktionen *Rückgängig* und *Wiederholen* unterstützt werden.

Hierbei geht es auch um eine Art "Simplicity of Navigation". Der Nutzer soll die Möglichkeit haben, schnell zu navigieren ohne nutzlose Zwischenseiten durchqueren zu müssen.

### 4. Konsistenz und Standards

Benutzer einer Seite sollten sich nicht wundern wenn verschiedene Wörter, Situationen oder Aktionen das Gleiche bedeuten.

Eine Anwendung sollte in der Regel immer einem einheitlichen Standard folgen. Eine wichtige Richtlinie ist hierbei Jakob's Law (Nielsen, 2017). Jakob's Law besagt, dass Nutzer die meiste Zeit auf anderen Webseiten verbringen. Daraus folgt, dass man seine eigene Seite nach demselben Schema aufbauen sollte wie es andere Seiten bereits tun, weil Nutzer diese anderen Seiten ja bereits kennen und verstehen. Eine gängiger Standard ist es beispielsweise, sein Logo links oben auf der Webseite zu platzieren. Weicht man von diesem Standard ab, kann das zu enormen Problemen in der Navigation führen. So zeigte Whittenton (2016b), dass bei einem zentrierten Logo 24% der Nutzer nicht in der Lage waren, durch ein Klick auf das Logo zurück zur Startseite zu gelangen. Bei einem Logo oben links waren nur 4% nicht dazu in der Lage (Whittenton, 2016b).

### 5. Fehlervermeidung

Besser als gutes Error Handling ist eine Konzept, dass erst gar keine Fehler produziert. Entweder man sollte fehleranfällige Bedingungen entfernen, oder man sollte Benutzern die Möglichkeit geben, die ausgewählte Aktion extra zu bestätigen.

## **6. Erkennen statt Erinnern**

Ein Benutzer sollte sich so wenig wie möglich merken müssen. Daher sollten Objekte, Aktionen und Optionen dem Nutzer sichtbar präsentiert werden. Das gleiche gilt auch für Anweisung, wie das System zu nutzen ist. Steve Krug nimmt dieses Prinzip als Leitfaden für sein Buch *Don't make me think* (2013).

## **7. Flexibilität und Effizienz der Nutzung**

Es sollte sogenannte "Accelerators" (Beschleuniger) geben, die dem unerfahrenen Nutzer unsichtbar bleiben, dem Experten Nutzer jedoch die Interaktion erleichtern. So ist ein System für beide Nutzergruppen angenehm bedienbar.

## **8. Ästhetik und minimalistisches Design**

Dialoge sollten nicht mehr Informationen als nötig haben . Eine tiefergehende Beschreibung dieser Regel ist in Kapitel 6.4.3 (Simplicity) zu finden .

## **9. Dem Nutzer helfen, Fehler zu erkennen, zu diagnostizieren und wiederherzustellen**

Fehlermeldungen sollten in lesbarer Sprache geschrieben sein (kein Code), das Problem richtig beschreiben und konstruktive Lösungen anbieten.

## **10. Hilfe und Dokumentation**

Am sinnvollsten ist es, wenn ein System ohne Dokumentation genutzt werden kann. Oft ist es jedoch besser, solche Informationen anzubieten und diese dem Benutzer einfach zugänglich zu machen,

### **6.4.3 Die Prinzipien der Einfachheit (Simplicity)**

Das bereits in Kapitel 6.4.2 erwähnte Prinzip der Einfachheit spielt in der Gestaltung eine große Rolle. Die Prinzipien können einerseits auf die Gestaltung alltäglicher Gegenstände angewendet werden, finden aber andererseits auch im User Interface Design ihre Anwendung.

Die nachfolgenden 10 Prinzipien stammen aus John Maeda's *Simplicity. Die Zehn Gesetze der Einfachheit* (2006).

#### **1. Reduzieren**

Das Prinzip beschäftigt sich mit der Grundfrage, wie einfach etwas gestaltet sein kann, und wie kompliziert es gestaltet werden muss. Maeda stellt hier die "VVV"-Regel auf. Man sollte Dinge verkleinern, verstecken und verbinden.

Kleinere Gegenstände entwickeln bei Menschen eher Sympathie als große Gegenstände. Man sollte versuchen, die Komplexität zu verstecken, um so eine Illusion der Einfachheit zu erzeugen. Ein kleineres Objekt wirkt jedoch nur dann wertvoller als ein größeres Objekt, wenn es auch hochwertiger aussieht. Dazu sollte man versuchen, das Produkt mit einem Eindruck von Qualität zu verbinden.

## **2. Organisieren**

Bei diesem Prinzip geht es um das Ordnen und Gruppieren von Elementen im Kontext eines gemeinsamen großen Elements. Es gilt die Regel, dass Gruppieren prinzipiell gut ist, zu viele Gruppen jedoch schaden können.

## **3. Zeit**

Auch Zeitersparnis kann sich auf eine gewisse Art wie Einfachheit anfühlen. Es geht darum entweder Wartezeiten zu verkürzen, oder die Zeit, die man warten muss, in irgendeiner Form erträglicher zu machen. Es reicht auch schon aus, wenn die Zeit nur scheinbar reduziert wird, damit uns das Komplexere einfacher erscheint.

## **4. Lernen**

Haben wir einen gewissen Vorgang gelernt, erscheint uns dieser oft als weniger komplex als andere ähnlich komplexe Vorgänge. Das heißt man sollte entweder so gestalten, dass Benutzer den Vorgang/das Produkt schnell verstehen, oder auf intuitive Formen/Vorgänge zurückgreifen, die Benutzer direkt beim erstmaligen Verwenden antizipieren können.

## **5. Unterschiede**

Ohne den Kontrast der Komplexität, wüssten wir den Benefit von Einfachheit nicht zu schätzen. Erst im Zusammenspiel zwischen Komplexität und Einfachheit, fällt die Einfachheit dem Nutzer tatsächlich auf.

## **6. Kontext**

Dieses Prinzip stellt die Frage wie viel Richtungsvorgaben ein Mensch ertragen kann und wie viel Orientierungslosigkeit man sich leisten kann. Man möchte den Nutzer nicht mit Informationen überschwemmen, ihm aber dennoch ausreichend Informationen bieten, sodass er nicht in totaler Hilflosigkeit endet.

## **7. Gefühle**

Der Mensch ist ein emotionales Wesen. Viele Entscheidungen sind gefühlsbasiert, und wir neigen dazu, uns von unseren Gefühlen leiten zu lassen. Man sollte Objekte so gestalten, dass sie mit positiven Emotionen assoziiert werden. Hier gilt: mehr Emotionen sind besser als weniger Emotionen.

## 8. Vertrauen

Ein wichtige Eigenschaft, die ein System einem Benutzer vermitteln soll, ist Vertrauen. Hier entsteht wieder die schwierige Frage: Wieviel müssen wir über ein System wissen und wie viel weiß das System über uns.

## 9. Fehlschläge

Manche Dinge lassen sich nicht vereinfachen. Es gibt Systeme und Objekte, die sich nicht nach den bisher genannten Regeln vereinfachen lassen, da sie sonst ihre Bestimmung und Funktionalitäten verlieren.

## 10. Das eine Gesetz

“Das Offensichtliche entfernen und das Sinnvolle hinzufügen”. Betrachtet man Dinge aus der Entfernung, erscheinen diese oft weniger komplex, als wenn man ganz genau hinschaut. Außerdem gilt, dass Offenheit das Komplexe ebenfalls vereinfachen kann.

Eine weitere, hier aufgeführte Regel lautet: je mehr wir verbrauchen, desto weniger gewinnen wir. Dies wiederum kann auf alle möglichen Bereiche angewendet werden.

### 6.4.4 Grundprinzipien der Dialoggestaltung

Die Grundprinzipien der Dialoggestaltung wurden von ISO bereits 1996 aufgestellt. Sie geben an, welche Anforderungen an Systeme gestellt werden, die mit dem Menschen interagieren. Folgende Prinzipien existieren seit der Neuauflage der ISON 9241-11 Anforderungen 2006:

#### 1. Aufgabenangemessenheit

Ein System sollte so designed sein, dass es den Nutzer vollständig, korrekt und mit vertretbarem Aufwand bei seinen Aufgaben unterstützt. Die Aufgabe sollte vollständiger oder mindestens genauso erledigt werden wie zuvor. Der Aufwand, den die Person aufbringt, sollte weniger oder zumindest gleich sein wie der Aufwand, den der Nutzer ohne das System aufbringen würde.

#### 2. Selbstbeschreibungsfähigkeit

Dem Nutzer soll zu jeder Zeit bekannt sein, wo er sich in der Anwendung befindet, wie er dahin gekommen ist und wie er wieder zurückkommt. Konkret besteht die Selbstbeschreibungsfähigkeit aus vier Punkten:

- **Orientierung:** Auf jeder Seite sollten dem Nutzer Orientierungspunkte angeboten werden, so dass er jederzeit weiß wo er sich in der Seitenhierarchie befindet. Außerdem soll er dadurch erkennen können, wie weit er noch von seinem Ziel entfernt ist.

- **Antizipierbarkeit:** Der Nutzer sollte erkennen, wohin ihn die Navigationselemente in der Anwendung führen.
- **Feedback:** Der Nutzer sollte Feedback darüber bekommen, ob eine Aktion erfolgreich war oder nicht. Das führt zu Sicherheit bei Nutzern und stärkt dessen Vertrauensbildung zur Seite.
- **Hilfe:** Unerfahrene Nutzer benötigen oftmals Hilfe bei der Nutzung von Anwendungen. Hierfür können zusätzliche Informationen bereitgestellt werden.

### 3. Erwartungskonformität

Ein System gilt als konform, wenn es die Sprache und "Arbeitsgebräuche" der Nutzer während der Interaktion berücksichtigt. Konventionen und Standards helfen dabei enorm.

### 4. Fehlertoleranz

Das System sollte dem Nutzer gegenüber fehlertolerant sein. Das System sollte entweder falsche Eingabe erst gar nicht erlauben, oder den Nutzer konstruktiv darauf hinweisen, dass seine Eingabe falsch ist und ihn anweisen, wie er seine Eingabe korrigieren kann.

### 5. Steuerbarkeit

Ein System sollte sich sowohl in der vom Benutzer gewünschten Richtung, als auch in der vom Benutzer gewünschten Geschwindigkeit steuern lassen.

### 6. Lernförderlichkeit

Das System sollte den Benutzer beim Erlernen des Umgangs mit dem System unterstützen.

### 7. Individualisierbarkeit

Das System sollte sich in gewissem Maße den Fähigkeiten und Bedürfnissen des Nutzers anpassen.

## 6.5 Formularspezifische Usability Standards

### 6.5.1 Navigationsstruktur & Seitenaufbau

So gut wie jede Anwendung braucht eine Navigation um zwischen verschiedenen Ansichten hin und her zu wechseln. Meist handelt sich um eine menüartige Navigationsstruktur, die auf der linken oder oberen Bildschirmhälfte angeordnet ist. Folgende Punkte wurden als Usability Kriterien für Navigationsstrukturen festgelegt.

## 1. Positionierung

Wie Nielsen (1997b) in einer Studie zeigte, lesen tatsächlich nur 16% der Nutzer einer Webseite diese Wort für Wort durch. Immerhin 79% der Nutzer scannen eine Webseite wenn sie diese erstmals besuchen. Das macht es quasi unabdingbar auf bekannte Strukturen zurückzugreifen (Gesetz der Erfahrung, Kapitel 6.4.1) und den Nutzer nicht lange nach der Navigation suchen zu lassen.

## 2. Gestalt & Inhalt

Meist hat eine Webseite einen Header, in dem der Nutzer sieht, in welchem View er sich gerade befindet. Außerdem enthält der Header oft ein Logo des Unternehmens, wodurch man wieder zur Startseite gelangt, wenn man darauf klickt (Benutzerkontrolle und Freiheit, Kapitel 6.4.2; Selbstbeschreibungsfähigkeit: Feedback, Kapitel 6.4.4). Bei verschachtelten Strukturen macht eine Breadcrumb Sinn, um dem Nutzer immer Feedback zu geben, wo er sich gerade befindet (Sichtbarkeit des Systemstatus, Kapitel 6.4.2; Selbstbeschreibungsfähigkeit: Orientierung, Kapitel 6.4.4).

### 6.5.2 Input Felder

Nachfolgend werden aus verschiedenen Quellen Richtlinien für den richtigen Umgang mit Inputfeldern zusammengetragen. Bei den einzelnen Punkten wurde sich an *Website Forms Usability: Top 10 Recommendations* von Whitenton (2016a) orientiert.

#### 1. Anzahl

Das wohl wichtigste Prinzip lautet: je weniger Input Felder, desto besser. Man sollte Input Felder entfernen, die Informationen sammeln, wenn:

- diese Informationen auf anderem Weg (ohne direkten Input) gesammelt werden können,
- diese Informationen bequemer zu einem späteren Zeitpunkt gesammelt werden können
- oder diese Informationen einfach weggelassen werden können (Whitenton, 2016a).

Hier greift das Prinzip der Simplicity (Kapitel 6.4.3).

#### 2. Positionierung

Prinzipiell gilt bei dem Positionieren von Input Feldern, wie bei dem Positionieren aller Objekte, ein gewisser Grad an Symmetrie (Gesetz der Symmetrie, Kapitel 6.4.1). Weiterhin sollte man darauf achten, thematisch zusammengehörige Input Felder auch zusammen anzuordnen (Gesetz der Nähe, Kapitel 6.4.1).



### 3. Gestalt & Größe

Ein Input Feld sollte auch als solches erkannt werden. Dazu sollte es durch eine Linie umrandet sein, und sich so von anderen Objekten abgrenzen (Gesetz der Geschlossenheit, Kapitel 6.4.1). Dadurch kann es als eigenständiges Objekt wahrgenommen werden. Man sollte sich auch an gängigen Formen von Input Feldern orientieren um den Benutzer nicht zu verwirren (Erkennen statt Erinnern, Kapitel 6.4.2; Konsistenz und Standards, Kapitel 6.4.2; Erwartungskonformität, Kapitel 6.4.4).

Nicht für jeden Input macht das gleiche einzeilige, schlitzartige Input Feld Sinn. Erwartet man zum Beispiel eine Beschreibung die mindestens 300 Zeichen beinhaltet, so sollte man das Input Feld bereits auf die Größe von 300 Zeichen einstellen. Erwartet man jedoch nur einen Benutzernamen, so sollte das Inputfeld platzsparend und einzeilig sein. Dass die Größe einen signifikanten Einfluss auf das Nutzerverhalten hat, konnten Christian, Dillman, & Smyth (2007) zeigen. Nutzer sollte eine Jahreszahl (bestehend aus vier Ziffern) in ein Input Feld eingeben und den Monat (bestehende aus zwei Ziffern) in ein anderes Input Feld. Durch das Vergrößern des Jahres-Input Felds und das Verkleinern des Monat-Input Felds gelang es den Forschern, dass 63% der Nutzer das richtige Format eintrugen (MM YYYY). Bei zwei gleich großen Input Feldern trugen nur 55% der Nutzer das gewünschte Format ein. Ein Input Feld sollte auch immer Feedback in Form einer kleinen Animation geben, wenn der Nutzer darauf klickt (Kapitel 6.4.2, Kapitel 6.4.4).

### 4. Inhalt

Ein Input Feld sollte aus einem Label und dem dazugehörigen Input Feld bestehen.

Es sollte prinzipiell vermieden werden, einen Placeholder anstelle eines Labels zu verwenden. Sherwin (2014) bietet einige gute Argumente, warum auf ein eigenständiges Label nicht verzichtet werden sollte:

- Verschwindende Placeholder können Nutzer verwirren. Will ein Nutzer nach Eingabe des Feldes erneut wissen, um welche Eingabefeld es sich handelt, muss er erst alles löschen um erneut zu sehen, was die Bezeichnung des Inputs ist.
- Es gibt keine Möglichkeit, seine Eingabe vor dem Absenden zu überprüfen, da man nicht kontrollieren kann, ob die richtigen Eingaben in den richtigen Input Feldern stehen.
- Leere Felder stechen Nutzern eher ins Auge.
- Nutzt man eine Anwendung am PC, besteht außerdem die Gefahr, dass, wenn man via Tab die Input Felder wechselt, irritiert ist, weil man das Label nicht sieht. Außerdem kommt es gelegentlich durch Bugs (Browserseitig) dazu, dass man Placeholder manuell löschen muss, was zusätzlich als störend empfunden wird.

Ist ein Label optional, sollte es auch als solche gekennzeichnet sein. Alternativ kann man auch Pflichtfelder mit einem \* kennzeichnen (Konsistenz und Standards, Kapitel 6.4.2).

## **5. Das richtige Input Feld für den richtigen Input**

Gibt es nur eine bestimmte Anzahl an Auswahlmöglichkeiten, macht ein Dropdown Menü beziehungsweise Radio Button definitiv mehr Sinn als der klassische Input Schlitz mit Form Validation. Prinzipiell sollten auf einem Desktop Computer ab vier Auswahlmöglichkeiten, Radio Buttons anstelle eines Drop Down Menüs verwendet werden (Miller & Jarrett, 2001).

Eine Telefonnummer sollte beispielsweise auch nicht aus zwei Inputfeldern (Vorwahl und Nummer) bestehen um unnötiges hin und her Klicken zu vermeiden.

## **6. Label Positionierung**

Bei der Positionierung des Labels sollte man darauf achten, dass es nahe dem Input Feld angeordnet ist. Hier greift das in Kapitel 6.4.1 beschriebene Gesetz der Nähe. Liegen Input Feld und Label dicht beieinander erkennt der Nutzer so direkt die Zusammengehörigkeit der Elemente. Am besten sollte das Label links oberhalb des Input Feldes angeordnet werden. Seckler et al. (2014) konnten zeigen, dass durch eine solche Anordnung Nutzer deutlich schneller die Input Felder erfassen können. Nutzer brauchten weniger Fixationen mit dem Auge und weniger Sakkaden (spontanen Blickbewegungen, willkürliche Augenbewegungen) bevor sie das Formular das erste Mal ausfüllten (Seckler et al., 2014).

Zu dem gleichen Ergebnis kam bereits Penzo (2006). Er konnte zeigen, dass das Erfassen eines Labels oberhalb des Input Feldes am wenigsten kognitive Leistung benötigt.

## **7. Fehlermeldungen**

Prinzipiell sollten dem Nutzer "falsche" Eingaben nicht erlaubt werden bzw. es sollte dem Nutzer zumindest eine Fehlermeldung angezeigt werden, die konstruktives Feedback gibt, wie die Eingaben korrigiert werden kann (Fehlertoleranz, Kapitel 6.4.4).

Eine Error Message (Fehlermeldung) für ein Input Feld erscheint dann, wenn eine Eingabe nicht den Formvalidierungsvorgaben entspricht. Beispielsweise dann, wenn ein Pflichtfeld übersehen wird, oder eine Eingabe gegen ein bestimmtes Regex Pattern verstößt. Dank der Fehlermeldung erhält der Nutzer Feedback über den aktuellen Status seiner Formulareingabe (Sichtbarkeit des Systemstatus, Kapitel 6.4.2).

Diese Fehlermeldung sollte nach dem Prinzip der Nähe ebenfalls nahe dem Input Feld angeordnet werden, um die Zusammengehörigkeit zu verdeutlichen. Seckler et al. (2012) konnten zeigen, dass Nutzer die Error Message am liebsten rechts von dem Input Feld sehen möchten (Error Messages unter dem Input Feld ist auf Platz 2).

### **6.5.3 Buttons**

Zu jeder formulargetriebenen Anwendung gehört mindestens ein Button um die eingegebenen Daten zu verschicken.

### 1. Position

Der Submit Button, der die eingegebenen Daten einer Anwendung verschickt, sollte sich nahe dem Formular befinden (Gesetz der Nähe, Kapitel 6.4.1).

### 2. Gestalt & Größe

Solange noch nicht alle Eingaben korrekt sind, d.h. nicht den Formvalidierungsrichtlinien entsprechen, sollte der Button ausgegraut (disabled) sein.

### 3. Reset / Cancel Button vermeiden

Das Risiko, aus Versehen den Cancel Button an Stelle des Submit Buttons zu drücken, überwiegt dem Nutzen eines solchen Buttons (Whitenton, 2016a). Wird ein solcher Button dennoch gebraucht, sollte er doch deutlich geringere visuelle Präsenz als der Submit Button aufweisen (Whitenton, 2016a).

## 6.6 Material Design Richtlinien

Die Informationen aus diesem Kapitel stammen alle von der offiziellen Angular Material Webseite (Introduction, n.d.; Accessibility, n.d.).

Die von Google entwickelten Material Design Richtlinien sind laut eigenen Angaben eine Synthese aus gutem Design, kombiniert mit der Innovation der Technik und Wissenschaft. Ziel von Material ist es, eine gelungen User Experience unabhängig der Plattform, des Geräts oder der Input Methode zu bieten. Weiterhin sieht sich das Material Framework als flexible Grundlage für Nutzer, um auf Material aufbauend, eigene Ideen hinzuzufügen.

Material ist eine Metapher der Physischen Welt und deren Strukturen. Dazu gehört auch die Art und Weise, wie Strukturen Licht und Schatten widerspiegeln. Material wird von ursprünglichen Druck-Design-Methoden geleitet. Dabei geht es um die Typographie, Abstände, Farben, etc.. Material basiert auf dem Grundsatz, dass Bewegungen Aufmerksamkeit erzeugen. Deshalb gibt es in dem Design Framework subtile Rückmeldungen auf Aktionen und viele kohärente Übergänge. Grundprinzipien sind Klarheit, Robustheit, Haftung und Spezifizierung. Im folgenden werden hauptsächlich die mobile spezifischen Usability Richtlinien von Material wiedergegeben.

**Types of feedback:** Gibt dem Nutzer Feedback. Das Feedback soll dem Nutzer anzeigen, wo er sich in der Applikation gerade befindet. Außerdem soll der Nutzer Feedback bekommen, wenn er UI Elemente touched.

**Navigation:** Die Navigation sollte einen klaren Ablauf haben und mit minimalen Schritten auskommen. Die Navigation sollte leicht zu finden sein und häufig verwendete Aufgaben leicht erreichbar machen.

**Hierarchy:** Zum einfacheren Verstehen der Benutzeroberfläche sollten folgende Regeln beachtet werden:

- Gut sichtbare Elemente.
- Ausreichend Kontrast und Größe.
- Eine klare Hierarchie von Wichtigkeit.
- Verwandte Elemente einer ähnlichen Hierarchie sollten nebeneinander platziert werden.

**Focus order:** Es sollte eine gewisse Reihenfolge gewahrt werden. Wichtige Elemente sollten weiter oben stehen als unwichtige. Die Reihenfolge, in der Elemente wahrgenommen werden, hängt mit verschiedenen Faktoren zusammen:

- Die Art und Weise wie Elemente gruppiert sind.
- Wohin sich der Fokus bewegt, wenn gerade fokussierte Elemente verschwinden.

**Grouping:** Ähnliche Elemente sollten zusammen unter einer Überschrift gruppiert werden, die aussagt, um was für eine Gruppierung es sich handelt.

**Transition:** Es sollte ein geschmeidiges Fokussieren zwischen Bildschirm und Aufgabe möglich sein.

**Accessible color:** Farbe kann dazu beitragen, Stimmung und kritische Informationen zu vermitteln. Man sollte Primär-, Sekundär- und Akzentfarben für seine Anwendung auswählen. Es ist auch wichtig, zwischen den einzelnen Elementen einen Farbkontrast sicherzustellen um die Elemente sichtbar voneinander unterscheiden zu können.

### **Contrast ratios**

Es sollte einen gewissen Kontrast zwischen der Hauptfarbe und deren Hintergrund geben. Dieser Wert kann zwischen 1 und 21 liegen.

**Logos and decorative elements:** Logos müssen nicht den Kontrastverhältnisse entsprechen, sie sollten jedoch unterscheidbar sein, wenn sie eine wichtige Funktion repräsentieren.

**Other visual cues:** Der Kontrast sollte nicht nur durch verschiedene Farben geschaffen werden. Es sollte auch durch andere Design Elemente ein Kontrast erzeugt werden, um zum Beispiel farbenblinde Menschen nicht auszuschließen.

**Touch and pointer targets:** Ein touchbares Element hat immer einen größeren *touch-listener* als es seiner tatsächlichen Größe entspricht. So kann ein Element zum Beispiel 24 x 24 dp groß sein, hat aber einen touchbaren Bereich von 48 x 48 dp. Das ist auch die Größe, die ein touchbares Element mindestens haben sollte. Pointer Elemente sollten mindestens eine Größe von 44 x 44 dp besitzen.

**Touch target spacing:** Touch Elemente sollten in der Regel mindestens 8dp auseinander liegen, um "touch-Konflikte" zu vermeiden.

**Responsive Layouts:** Das Layout sollte flexibel sein und sich der Größe des Endgeräts anpassen.

**Grouping items:** Verwandte Objekte sollte räumlich beieinander angeordnet werden.

**Fonts:** Man sollte darauf achten, dass die Schriftgröße immer groß genug und lesbar für die Benutzer ist.

**Accessibility text:** Klare und hilfreiche Eingabehilfen sind eine der wichtigsten Methoden um UIs besser zugänglich zu machen.

**Visible and nonvisible text:** Sowohl sichtbarer, als auch nicht sichtbarer Text sollte aussagekräftig sein. Nicht sichtbare Texte sind zum Beispiel alternative Texte für Schaltflächen mit Symbolen.

**Be succinct:** Der Text der Barrierefreiheit sollte kurz und prägnant sein.

**Avoid stating control type or state:** Bildschirmleser können den Typ oder Status eines Steuerelements automatisch durch einen Ton oder durch den Namen des Steuerelements vor oder nach dem Text im Sinne der Barrierefreiheit bekannt geben.

**Developer note:** Jedes Element sollte eine eindeutige Rolle auf der Webseite haben. Das bedeutet, dass eine Schaltfläche als Schaltfläche und ein Kontrollkästchen als Kontrollkästchen wahrgenommen werden sollen.

**Indicate what an element does:** Man sollte Aktionsverben verwenden, um anzugeben, was ein Element oder eine Verknüpfung tut, und nicht wie ein Element aussieht. Dadurch können zum Beispiel auch blinde Menschen die Seite verstehen.

**Elements with state changes:** Bei einem Element, welches zwischen Werten beziehungsweise Stadien wechselt, sollte immer der Status angegeben werden, je nachdem wie es dem Nutzer aktuell präsentiert ist.

**Don't specify how to interact with a control:** Man sollte den Benutzern nicht beschreiben wie sie physisch mit einem Steuerelement interagieren sollen, da sie möglicherweise mit einer Tastatur oder einem anderen Gerät navigieren und nicht mit den Fingern oder einer Maus.

**Hint speech:** Für unklare Aktionen sollten Zusatzinformation bereitgestellt werden. Als Beispiel ist hier Android's "double-tap to select" zu nennen.

**Sound:** Man sollte dem Visuellen eine auditive Alternative geben und umgekehrt ebenso. Unnötige Sounds sollten jedoch vermieden werden, wie das automatische Abspielen von Musik beim Öffnen einer Webseite.

**Motion:** Material Design verwendet Bewegung, um den Fokus zwischen verschiedenen Ansichten zu lenken. Oberflächen werden zu Fokuspunkten und für den Benutzer unwichtige Elemente sollten entfernt werden.

**Timed controls:** Manche Steuerelemente in der App können so eingestellt werden, dass sie nach einer gewissen Zeit nicht mehr angezeigt werden (Bspw. das Ausblenden der Videoleiste).

**High-priority controls:** Das Verwenden von Timern für Steuerelementen mit hoher Priorität sollte vermieden werden.

**Implementing accessibility:** Eine App sollte der zugrundeliegenden Plattform angepasste Steuerelemente besitzen. Sie sollte so angepasst werden, dass die Standards der Barrierefreiheit garantiert werden.

**Help documentation:** Jedes Feature mit speziellen Zugangsmöglichkeiten sollte in der Hilfe Dokumentation enthalten sein.

**Testing and research:** Die Anwendung sollte von Anfang bis Ende durchgetestet werden.

## 7 Verwendete Technologien

### 7.1 Typescript

Die App wurde auf Basis von Microsofts Programmiersprache TypeScript<sup>15</sup> entwickelt. Bei TypeScript handelt es sich um eine Sprache, die hauptsächlich zum Entwickeln von Webanwendungen vorgesehen ist. Im Gegensatz zu JavaScript stellt TypeScript zusätzlich Typen, Klassen und Module zur Verfügung. Gängige Browser können TypeScript nicht interpretieren, weshalb es vor dem Deployen zu JavaScript kompiliert werden muss.

### 7.2 Angular 6

Die Grundarchitektur der Anwendung wird durch Angular 6<sup>16</sup> vorgegeben. Bei Angular handelt es sich um eine von Google vorangetriebene TypeScript basiertes Framework zur Entwicklung von Software. Hauptsächlich ist Angular zum Entwickeln von Webseiten gedacht, kann aber auch mit Hilfe von Frameworks wie zum Beispiel Ionic in der mobilen App Entwicklung eingesetzt werden. Seit Angular 5 gibt es auch einen Support für Service Worker, was den Weg für Angular basierte PWAs freigemacht hat. Im Folgenden werden die grundlegende Architektur und der Aufbau einer Angular Anwendung erklärt.

Die folgenden Unterkapitel stammen inhaltlich von der offiziellen Angular Webseite.

#### 7.2.1 Components

Eine Component<sup>17</sup> bezeichnet einen gewissen Teil einer Anwendung, des sogenannten Views. Man gliedert damit die Anwendung in viele kleine Teilstücke und jede dieser Teilstücke enthält seine eigene Struktur (HTML), seinen eigenen Style (SCSS) und seine eigene Logik (Typescript).

Die folgende Anwendung zeigt, wie eine Kapselung in Components erfolgen kann. Die beige markierten Elemente stellen jeweils eine Component dar. So werden zum Beispiel das Menü und der Header als eine eigene Components dargestellt. Der Image Upload Button und der Rest der Seite stellen ebenfalls wieder verschiedene Components dar.

---

<sup>15</sup> <http://www.typescriptlang.org/>

<sup>16</sup> <https://angular.io/>

<sup>17</sup> <https://angular.io/guide/architecture-components/>

The screenshot shows a mobile application interface with a light pink background. At the top, there is a header bar with a back arrow, a plus sign, and the text "OpenProject - PWA". Below the header, the main content area is titled "Create a new work package". It contains three input fields: "Title \*" (with an asterisk indicating it is required), "Description", and "Advanced data" (with a downward arrow indicating a dropdown menu). Below these fields, there is a button with a plus sign in a black circle and the text "Image Upload". At the bottom, there is a grey button with the text "Create Work package".

Abbildung 21: Angular Components.

Angular Components funktionieren mit dem Prinzip des sogenannten *two-way-data-binding* (Abbildung 22). Dadurch ist der Austausch von Daten zwischen Component und Template gewährleistet und es ist auf diese Weise möglich im View die Daten zu ändern, die dann über ein Event Binding (Beispiel: click event) die Daten in der Component anpassen.

Der umgekehrte Weg bietet die Möglichkeit über Property Binding in der Component automatisch den View im Template zu verändern. Auch bei Parent-Child Kommunikation spielt das two-way-binding eine wichtige Rolle.



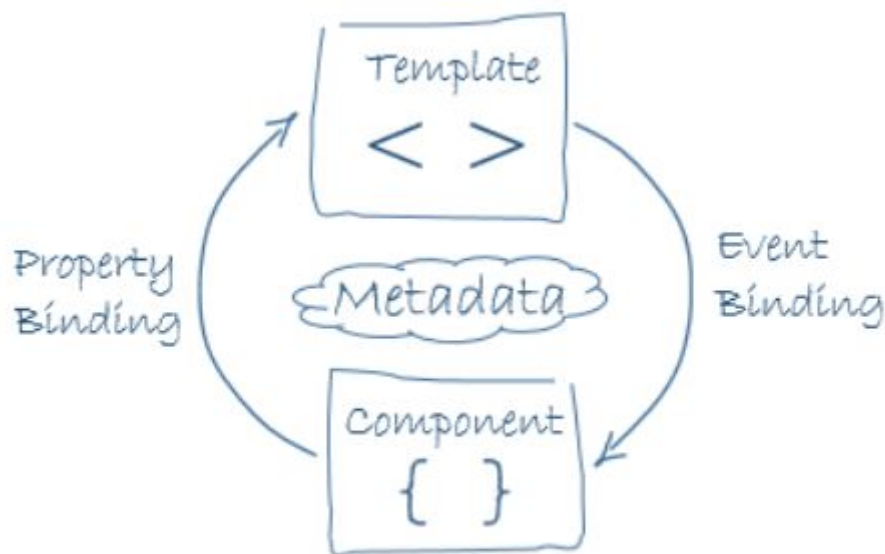


Abbildung 22: Angular Components - Event binding (Introduction to components, n.d.).

## 7.2.2 Services

Ein weiteres wichtiges architektonisches Merkmal von Angular sind Services<sup>18</sup>. Services sind Typescript-Klassen, die Logik bereitstellen und kein HTML und CSS besitzen. Eine Component kann sich diese Logik mittels Dependency injection zunutze machen (Abbildung 23).

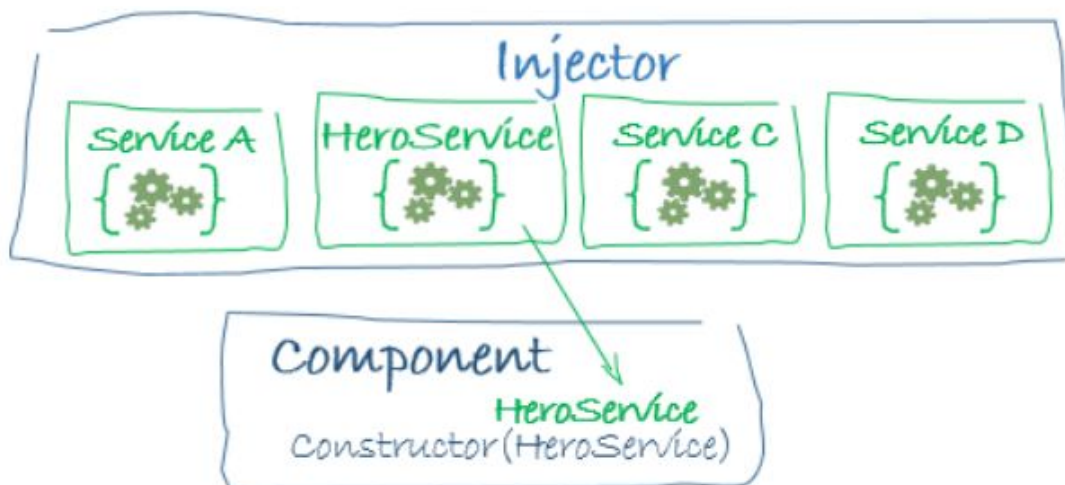


Abbildung 23: Angular Services - Dependency injection (Introduction to services and dependency injection, n.d.).

<sup>18</sup> <https://angular.io/guide/architecture-services/>

Im Idealfall dient eine Component nur dem Anzeigen von Inhalten und gibt die Logik an Services ab. Eine Component sollte zum Beispiel auf keinen Fall so etwas wie http Requests oder Vergleichbares initiieren, sondern statt dessen diese Funktionalität an Services übertragen.

### 7.2.3 Modules

Angular besitzt sein eigenes modulares System namens NgModules. Ein Modul<sup>19</sup> deklariert directives, pipes und components. Es importiert und exportiert Funktionalität an andere Module. Es stellt Services bereit, die dann in der Anwendung injected werden können (Introduction to modules, n.d.)

## 7.3 Service Worker

### 7.3.1 Definition

Die folgende Definition wurden von Matt Gaunt (2018) übernommen.

Der Service Worker (SW) ist ein Script, das während dem Surfen auf einer Webseite vom Browser im Hintergrund ausgeführt wird. Der Service Worker ist JavaScript basiert und kann den DOM nicht direkt ansprechen. Die Kommunikation mit der Seite erfolgt über *postMessages*.

Um einen Service Worker zu benutzen muss man diesen zuerst registrieren. Sobald er registriert ist, werden alle Seiten, die sich innerhalb seines Scopes befinden von dem Service Worker verwaltet. Sobald der Service Worker erfolgreich aktiviert wird, kann er zwei Zustände haben. *Terminated* bedeutet, der SW wurde beendet, um Speicher zu sparen. Bei *Fetch / Message* behandelt der SW Fetch bzw. Message Events, die bei einem Netzwerk Request oder bei Nachrichten der eigenen Seite auftreten. Der Lifecycle ist in Abbildung 24 noch einmal dargestellt.

---

<sup>19</sup> <https://angular.io/guide/architecture-modules/>

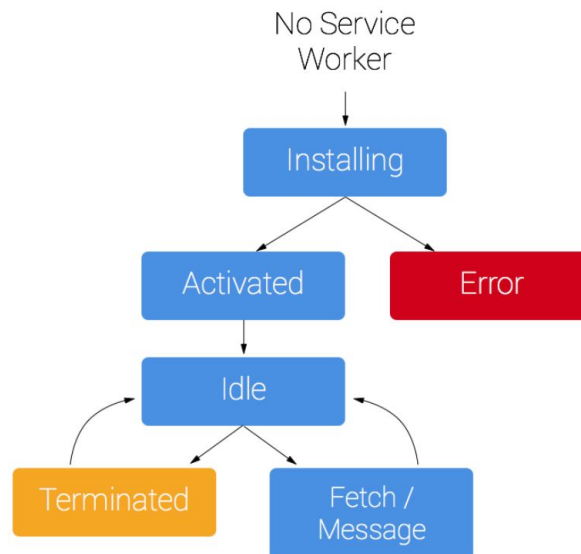


Abbildung 24: Service Worker Lifecycle (Gaunt, 2018).

### 7.3.2 Einen Service Worker anmelden

Mit Hilfe der Angular CLI kann man zu jedem bestehenden Projekt einen Service Worker hinzufügen. Dazu muss der Befehl:

**`ng add @angular/pwa --project *project-name*`**

ausgeführt werden. Dieser Befehl löst folgendes aus:

- Das Service Worker Paket wird zu dem Projekt hinzugefügt.
- Erlaubt den Service Worker Build in der CLI.
- Der Service Worker wird automatisch Importiert und im App Module registriert.
- Die index.html wird aktualisiert.
  - Beinhaltet einen Link, der die *manifest.json* (Kapitel 7.4) hinzufügt.
  - Fügt meta tags für *theme-color* hinzu.
- Es werden Icons installiert, die die PWA unterstützen.
- Die Service Worker Konfigurationsdatei wird generiert (siehe Kapitel 7.3.3).

### 7.3.3 Die ngsw-config.json Datei

Die ServiceWorker.js Datei wird erst nach einem erfolgreichen *ng build* zusammen mit den anderen JavaScript Dateien generiert. Wie die später generierte *ServiceWorker.js* Datei aussieht wird vorher in einer Konfigurationsdatei namens *ngsw-config.json*<sup>20</sup> festgelegt. Die Datei dient dazu, den Service Worker zu konfigurieren und ihm zu sagen wie er zu arbeiten hat. Die Datei wird im JSON Format hinterlegt und definiert wie der Service Worker das Caching behandeln soll. Die default *ngsw-config.json* sieht folgendermaßen aus:

<sup>20</sup> <https://angular.io/guide/service-worker-config>

```

{
  "index": "/index.html",
  "assetGroups": [{
    "name": "app",
    "installMode": "prefetch",
    "resources": {
      "files": [
        "/favicon.ico",
        "/index.html",
        "/*.css",
        "/*.js"
      ]
    }
  }, {
    "name": "assets",
    "installMode": "lazy",
    "updateMode": "prefetch",
    "resources": {
      "files": [
        "/assets/**"
      ]
    }
  }
]}

```

Abbildung 25: Service Worker - *ngsw-config.json*.

Im Folgenden werden die wichtigsten Konfigurationen der *ngsw-config.json* erklärt.

**index:** Definiert die Seite die der Anwendung als Index dient. Für gewöhnlich ist das die *index.html*.

**assetGroups:** Assets sind Ressourcen, die Teil der Appversion sind und auch zusammen mit der App aktualisiert werden. Sie können Ressourcen beinhalten, die von der Seite selbst bereitgestellt werden, aber auch Ressourcen, die von Drittanbietern (z. B. CDN oder anderen externen URL's) geladen werden. Da zum Zeitpunkt der Erstellung möglicherweise nicht alle URL's bekannt sind, können URL Muster verglichen werden. Asset Groups verwalten eine Menge an Ressourcen und geben an, wie mit diesen Ressourcen zur Programmlaufzeit umgegangen werden soll. Das heißt es wird festgelegt, wann die Ressourcen gefetched werden und was passiert, wenn zum Beispiel Änderungen festgestellt werden.

Das *AssetGroup* Interface sieht folgendermaßen aus:

```
interface AssetGroup {
  name: string;
  installMode?: 'prefetch' | 'lazy';
  updateMode?: 'prefetch' | 'lazy';
  resources: {
    files?: string[];
    /** @deprecated As of v6 `versionedFiles` and `files`
     * options have the same behavior. Use `files` instead.*/
    versionedFiles?: string[];
    urls?: string[];
  };
}
```

Abbildung 26: *ngsw-config.json* - *AssetGroup* interface

**name:** Der *name* identifiziert die Gruppe an Assets. Er ist obligatorisch.

**install mode:** Der *install mode* bestimmt wie die Ressourcen initial gecached werden sollen. Der *install mode* kann zwei verschiedene Werte haben, *prefetch* und *lazy*.

**prefetch:** Damit wird dem Service Worker gesagt, dass jede einzelne Ressource gefetched werden soll, während die aktuelle Version der App gecached wird. Das Ganze ist sehr bandbreitenintensiv, garantiert jedoch, dass die Inhalte immer verfügbar sind, auch wenn der Nutzer gerade offline ist.

**lazy:** Hiermit werden keine Ressourcen im Voraus gecached. Es werden nur Ressourcen gespeichert für die ein Request ansteht. Es handelt sich um ein On-Demand-Caching-Modus. Ressourcen die nie aufgerufen werden, werden auch nicht zwischengespeichert. Ein möglicher Anwendungsfall ist bei einer Anwendung die Bilder in verschiedenen Auflösungen zur Verfügung stellt. Hierbei könnte der Service Worker nur die passenden Bilder für die gegebene Auflösung speichern.

**updateMode:** Für alle Ressourcen die bereits gecached werden bestimmt *updateMode* wie diese auf neuere Versionen reagieren sollen.

**prefetch:** Der Service Worker downloadet und cached die Ressourcen.

**lazy:** Der Service Worker cached die Ressourcen von vornherein erst einmal nicht. Stattdessen werden

diese Ressourcen als `unrequested` deklariert. Somit wird die Ressource erst geupdated und gecached wenn sie noch einmal angefragt wird. Lazy ist nur erlaubt wenn der `installMode` ebenfalls lazy ist.

**resources:** Es gibt drei verschiedene Ressource Typen die gecached werden können.

**files:** Files sind Muster, die mit Dateien im Distributionsverzeichnis übereinstimmen. Das können einzelne Dateien oder glob-artige Muster sein, die einer Anzahl von Dateien entsprechen.

**urls:** Hierbei sind sowohl URLs als auch URL Muster enthalten. Die Ressourcen werden nicht direkt gefetched und haben keinen content hash. Statt dessen werden diese abhängig von ihren HTTP headern gecached. Das kann für CDNs wie Google Font Service nützlich sein.

**dataGroups:** Im Gegensatz zu Asset-Ressourcen werden Daten Requests nicht zusammen mit der App versioniert. Sie werden nach manuellen Richtlinien zwischengespeichert, die für Situationen wie API-Anforderungen und andere Datenabhängigkeiten sinnvoll sind.

Hier das *DataGroup* Interface:

```
export interface DataGroup {
  name: string;
  urls: string[];
  version?: number;
  cacheConfig: {
    maxSize: number;
    maxAge: string;
    timeout?: string;
    strategy?: 'freshness' | 'performance';
  };
}
```

Abbildung 27: *ngsw-config.json* - *DataGroup* interface.

**name:** Ebenso wie bei *assetGroup*, hat auch die *dataGroup* einen unique identifier.

**urls:** Eine Liste von URL Patterns. URLs, die auf dieses Pattern passen, werden nach den festgelegten Richtlinien gecached.

**version:** Gelegentlich sind APIs nicht rückwärts kompatibel. Eine neue Version der App könnte inkompatibel mit der alten API sein und diese könnte wiederum nicht kompatibel zu den existierenden gecachten Ressourcen der API sein. Die *Version* bietet den Mechanismus ,um zu erkennen ob die Ressourcen die gecached wurden in einem abwärtskompatiblen Weg geupdated worden sind. Die alten Caches aus früheren Versionen sollten dann gelöscht werden. *Version* ist ein Integer Feld und defaultmäßig 0.

**cacheConfig:** Hier wird definiert welcher Matching Request gecached wird.

**maxSize:** Beschreibt die maximale Anzahl an Einträgen oder Responses im Cache. MaxSize ist ein Pflichtfeld.

**MaxAge:** Es wird definiert, wie lange es einem response erlaubt ist in dem Cache gespeichert zu werden, bevor er invalid und evicted betrachtet wird. MaxAge ist ein String mit folgendenden suffixes:

- d: days
- h: hours
- m: minutes
- s: seconds
- u: milliseconds

Beispielsweise würde der string 3d12h Inhalte für dreieinhalb Tage speichern.

**timeout:** Der String definiert den Netzwerk Timeout. Es wird definiert wie lange der Service Worker auf einen Response warten soll, bevor er einen Gecachten Response zurückgibt.

**strategy:** Für Datenquellen kann der Service Worker eine von zwei Strategien anwenden.

**performance:** Standardmäßig für optimalen Response. Wenn Ressourcen im Cache existieren, werden diese verwendet. Nützlich für sich selten ändernde Ressources wie Avatar Bilder.

**freshness:** Bevorzugt das Abrufen von frischen Daten aus dem Netzwerk. Nur wenn das Netzwerk den Timeout überschreitet, fällt die Anforderung in den Cache zurück. Das bietet sich bei häufig ändernden Ressourcen wie zum Beispiel Kontoständen an.

## 7.4 Die Manifest.json Datei

Die Web App Manifest Datei stellt Informationen über die Anwendung im JSON Format (Manifest.json<sup>21</sup>) bereit. Diese Datei ist dafür verantwortlich, dass der Nutzer sich die App später auf dem Homescreen sinnvoll installieren kann. Innerhalb der Manifest Datei wird angegeben wie die App interpretiert werden soll.

**background\_color:** Hier wird angegeben welche Hintergrundfarbe die Web-Anwendung haben soll bis die CSS Dateien vom Browser geladen werden.

**description:** Gibt eine kurze Beschreibung darüber was die Web-Anwendung tut.

**dir:** Definiert die Textrichtung. Der Wert "ltr" bedeutet zum Beispiel eine Textrichtung von links nach rechts, während der Wert "lang" von rechts nach links bedeutet.

**display:** Hiermit wird der bevorzugte Anzeigemodus für die Webanwendung festgelegt. Bei der Einstellung *standalone* wird die App wie eine eigenständige Anwendung dargestellt. Das heißt die Browserleiste wird verborgen und Features wie z.B. einen neuen Tab öffnen oder in den privaten Modus zu wechseln stehen nicht zur Verfügung.

**icons:** Gibt ein Array von Bildobjekten an, die als Anwendungssymbole in verschiedenen Kontexten dienen. Beispielsweise können für eine Listenansicht aller Anwendungen und in der Ansicht eines Task-Switchers verschiedene Größen von Icons verwendet werden.

**lang:** Hier wird die primär Sprache der Anwendung festgelegt.

**name:** Stellt den Namen der Anwendung bereit. Dieser ist nach dem Öffnen der App und während des Ladens der Inhalte zu sehen.

**orientation:** Definiert die Standardausrichtung für alle Top-Level der Webanwendung Browsing-Kontexte.

---

<sup>21</sup> <https://developer.mozilla.org/en-US/docs/Web/Manifest/>



**prefer\_related\_applications:** Gibt einen Boolean zurück, der angibt ob verwandte Anwendungen (siehe nächsten Punkt) verfügbar sind.

**related\_applications:** Stellen ein Array aus nativen Applikationen dar. Diese Anwendungen sollen als Alternative zu der Webapplikationen dienen und können über den jeweiligen App Store des Geräts heruntergeladen werden.

**scope:** Definiert den Navigationsbereich der Anwendung.

**short\_name:** Zeigt den kurzen Namen der Anwendung.

**start\_url:** Gibt die URL an, die geöffnet wird wenn der Benutzer die App öffnet.

**theme\_color:** Definiert die Standardthemfarbe für die Anwendung.

## 7.5 Angular Material

Als Design Grundlage wurde Material<sup>22</sup> gewählt, ein von Google entwickeltes Design Schema. Inspiriert ist Material von der physischen Welt und seinen Texturen, zum Beispiel wie Licht reflektiert wird oder wie Schatten fallen. Die konkrete Umsetzung des Styles erfolgt durch Angular Material, ein speziell für Angular entwickeltes CSS Frameworks des Material Design Pattern.

In Kapitel 9 wird gezeigt in welchen Bereichen das Framework für eine gute Usability der Anwendung sorgt.

Zum Anordnen der einzelnen Elemente wurde nicht das Grid System von Material genutzt, sondern stattdessen wurde auf das *AngularFlexLayout*<sup>23</sup> zurückgegriffen.

## 7.6 IndexedDB & Dexie.js

Wie im nächsten Kapitel beschrieben, wird zum Cachen der Work packages die in jedem "Standardbrowser" integrierte IndexedDB verwendet. Die IndexedDB<sup>24</sup> ist eine asynchrone objektorientierte Datenbank deren API alle klassischen CRUD Operationen zur Verfügung stellt. Die Daten werden als Key-Value Pair gespeichert. Über den Key werden die CRUD Operationen ausgeführt um den Value (das gespeicherte Objekt) zu verändern.

---

<sup>22</sup> <https://material.io/design/introduction/>

<sup>23</sup> <https://github.com/angular/flex-layout>

<sup>24</sup> [https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB\\_API/](https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/)

Im Gegensatz zum synchronen LocalStorage<sup>25</sup> bietet die IndexedDB ein größeren Speicher, der für das cachen von Bildern benötigt wird. Synchrones speichern der Bilder würde außerdem enorme Performance Probleme mit sich bringen und nur schwer mit den vielen asynchronen HTTP Requests harmonisieren.

WebSQL<sup>26</sup> kam als Browser Datenbank nicht infrage, da diese Technologie seit 2010 nicht mehr weiterentwickelt wird.

Als Wrapper um die IndexedDB wurde die JavaScript Library Dexie.js<sup>27</sup> gewählt.

Dexie.js bietet eine übersichtliche und einfache Nutzung der IndexedDB. Es gibt eine Integration für Typescript was bei vielen Javascript basierten IndexedDb Wrappern leider fehlt. Die folgende Abbildung zeigt wie eine Datenbank mit Hilfe von Dexie.js erstellt werden kann.

```
import { WorkPackageModel } from
'../work-packages/work-package.model';
import Dexie from "dexie";

export class WorkPackageDb extends Dexie {
  workpackages: Dexie.Table<WorkPackageModel, string>;
  constructor() {
    super("Workpackages");
    this.version(1).stores({
      workpackages: 'id'
    });
  }
}
```

Abbildung 28: Dexie.js - Work package Datenbank.

Es wird Dexie und das Work package Model importiert. Es wird die Tabelle erstellt, bestehend aus eine Work package und einem String, der den key abbildet.

Im Konstruktor wird der Konstruktor der Parent Klasse aufgerufen, der das Erstellen der Datenbank abschließt.

<sup>25</sup> <https://developer.mozilla.org/de/docs/Web/API/Window/localStorage/>

<sup>26</sup> <https://www.w3.org/TR/webdatabase/>

<sup>27</sup> <http://dexie.org/>

# 8 Implementierung

## 8.1 Funktionale Umsetzung

### 8.1.1 Work package erstellen

Das Kernfeature der Anwendung ist das Erstellen eines Work packages. Über einen Klick auf das Plus-Symbol im Header der PWA gelangt man auf den View mit der Überschrift "Create a new Work package". Diese Ansicht beinhaltet mehrere Input Felder. Es gibt die immer sichtbaren "Basic Input Felder" und die ausklappbaren "Advanced Input Felder". Meist braucht der Anwender nur die Basic Felder, weshalb die Advanced Input Felder nur sichtbar sind, wenn man diese explizit ausklappt. Nachfolgend findet sich ein Übersicht aller Input Felder. In der ersten Spalte findet sich der Name des Input Feldes, in der Zweiten Spalte der Typ. Spalte 3 gibt den Datentyp des Feldes an und Spalte 4 gibt an, ob es eine Art der Form Validierung gibt. Nur wenn alle Form Validatoren *true* zurückgeben wird der "Create Work package" Button anklickbar.

| Name            | Input Feld Typ | Datentyp     | Form Validation   |
|-----------------|----------------|--------------|---|
| Title           | Text field     | string       | Pflichtfeld.  |
| Description     | Text field     | string       | -   |
| Image Upload    | Upload File    | img (base64) | nur png / jpeg / gif erlaubt.                                     |
| Estimated Time  | Text field     | number       | Es ist dem Nutzer nur erlaubt Zahlen einzugeben.                  |
| Remaining Hours | Text field     | number       | Es ist dem Nutzer nur erlaubt Zahlen einzugeben.                  |
| Start date      | Datepicker     | Date         | Das Start date muss zeitlich vor dem Due date liegen.             |
| Due date        | Datepicker     | Date         | Das Start date muss zeitlich vor dem Due date liegen.             |
| Progress        | Text field     | number       | Es ist dem Nutzer nur erlaubt Zahlen zwisch 1 und 100 einzugeben. |

Abbildung 29: Work package Attribute.

Abbildung 30 zeigt die "Basic Input" Felder (links). Klickt man dort auf den Button "Advanced Data" öffnen sich die "Advance Input" Felder (Abbildung 30, rechts).

←

+

Project: bauprojekt-c2

⚙

Create a new work package

Title \*

Wand streichen - Gebäude 3

Description


Die Rückwand von Gebäude 3 muss noch weiß gestrichen werden.

Advanced data

▼

+

Image Upload



Create Work package

←

+

Project: bauprojekt-c2

⚙

Create a new work package

Title \*

Wand streichen - Gebäude 3

Description

Die Rückwand von Gebäude 3 muss noch weiß gestrichen werden.

Advanced data

▲

Estimated Time (in hours)

20 h

Remaining Hours

10 h

Start date

9/2/2018

📅

Due date

9/12/2018

📅

Progress (1 - 100%)

50%

+

Image Upload

Create Work package

Abbildung 30: Create a new Work package - basic Input (links); advanced Input (rechts).

Will der Nutzer das Work package senden, ist aber zu diesem Zeitpunkt nicht mit dem Internet verbunden, erscheint eine Benachrichtigung, die den Nutzer darauf hinweist, dass das Work package gesendet wird sobald er wieder online ist (Abbildung 31, links). Abbildung 31 (rechts) zeigt den Fall, bei dem das Work package erfolgreich gesendet worden konnte.

67

← + Project: bauprojekt-c2 ⚙️

Create a new work package

Title \*  


---

Description  


---

Advanced data ▼

+

Image Upload

Create Work package

← + Project: bauprojekt-c2 ⚙️

Work packages

| Id  | Title                      | Location |
|-----|----------------------------|----------|
| 142 | t1                         | Server   |
| 143 | t2                         | Server   |
| 144 | Wand streichen - Gebäude 3 | Server   |

No Internet connection. Workpackage "Wand streichen - Gebäude 3" will be sent when you go back online

Workpackage "Wand streichen - Gebäude 3" was sent!

Abbildung 31: Create a new Work package. Wenn der Nutzer keine Internetverbindung hat (links); Wenn Internetverbindung besteht (rechts).

### 8.1.2 Liste an Work packages anzeigen

Eine tabellarische Übersicht aller vom Smartphone aus erstellten Work packages bildet zugleich die Startseite der Anwendung. Es gibt die Spalten *Id*, *Title* und *Location*. *Id* zeigt die Backendseitig vergebene ID des Work packages an. *Title* ist der Titel des Work packages. *Location* gibt den aktuellen Speicherort des Work package an. Konnte ein Work package aufgrund von fehlender Internetverbindung noch nicht gesendet werden, so wird es lokal gecacht und hat als Location *local* angezeigt. Konnte ein Work package gesendet werden, hat das Work package als Location *Server* in der Spalte stehen. Durch Anklicken eines solchen Work packages gelangt man in den Detailview des WPs, siehe Kapitel 6.

Abbildung 32 zeigt die Listenansicht aller Work packages.

| Project: bauprojekt-c2         |   |          |
|--------------------------------|---|----------|
| Work packages                  |   |          |
| Id                             | Title                                       | Location |
| 142                            | t1  | Server   |
| 143                            | t2  | Server   |
| 144                            | Wand streichen - Gebäude 3                  | Server   |
| 145                            | Fliesen legen - Bb                          | Server   |
| 146                            | Aufräumen Gebäude 4                         | Server   |
| 147                            | Fenster austauschen Hinterseite - Gebäude 3 | Server   |
| - Work packages not yet sent - |   |          |
| -                              | Wann verputzen - Raum B341 - Gebäude C      | Local    |

Abbildung 32: Work packages Listenansicht.

### 8.1.3 Work package Details anzeigen

Die Detailansicht stellt alle Input Felder aus Abbildung 29 in benutzerfreundlicher Form dar. Zusätzlich zu den genannten Feldern gibt es noch einen Google Maps Link in der Beschreibung. Der Link verweist auf die Koordinaten, an denen das Work package erstellt wurde. Wurden irgendwelche Daten des Work packages zwischenzeitlich über die Webanwendung von OpenProject verändert, so werden sie auch hier aktualisiert angezeigt, da die Daten bei jedem Aufruf neu angefragt werden. Wurde das Work package gelöscht erscheint eine Meldung *Work package doesn't exist anymore*. Abbildung 33 zeigt die Detailansicht eines Work packages. Links sieht man den *normalen View*, während die

Ansicht rechts dann angezeigt wird, wenn das Work package über die Webanwendung von OpenProject gelöscht wurde.

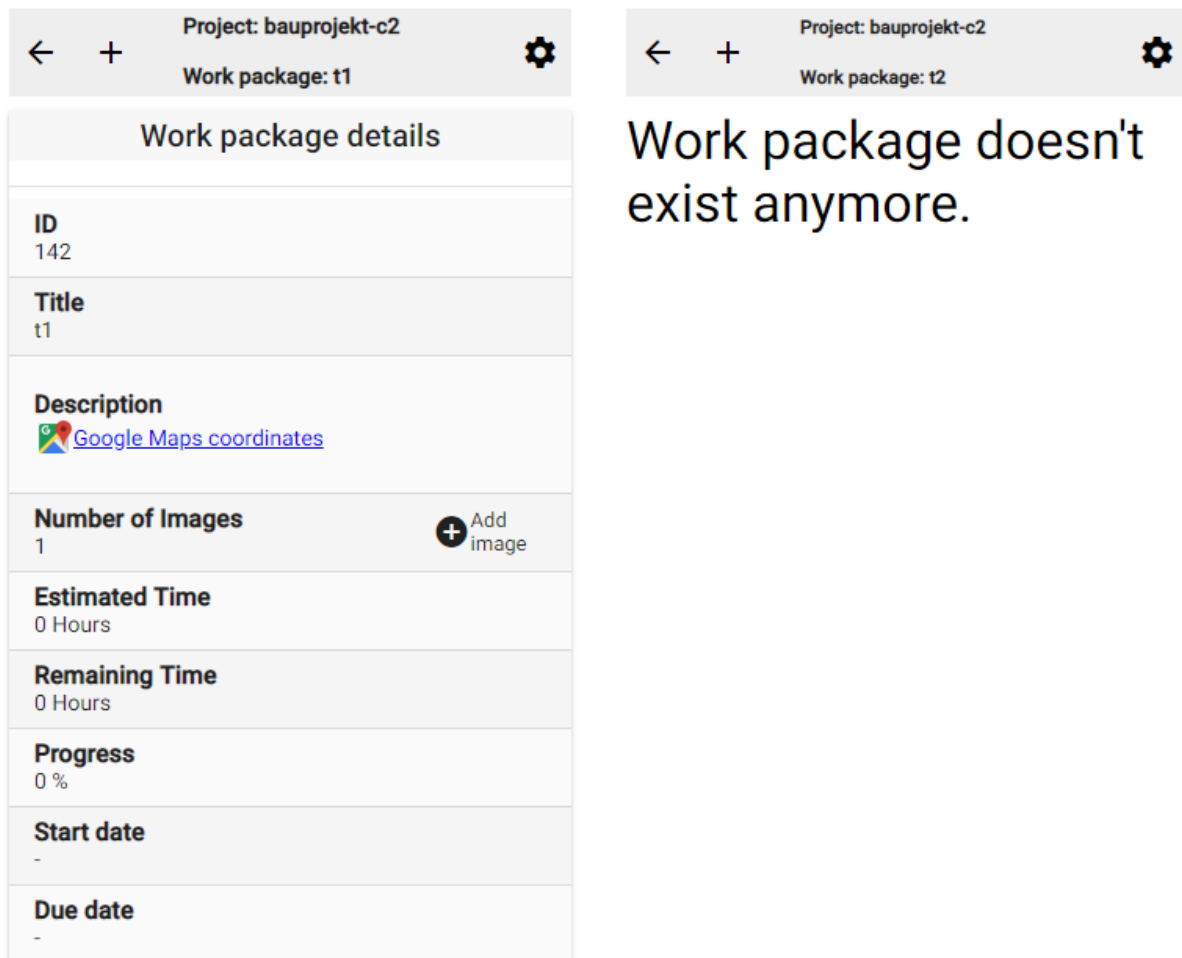


Abbildung 33: Work packages Detailansicht. (links) Detailansicht bei vorhandenem Work package. (rechts) Ansicht wenn Work package gelöscht wurde.

Klickt der Nutzer auf *Google Maps coordinates* gelangt er zu Google Maps, siehe Abbildung 34. Existiert kein Google Maps Link (zum Beispiel wenn der Smartphone Nutzer keinen Standortzugriff erlaubt), erscheint stattdessen in der Beschreibung, dass das Tracken der Geolocation nicht möglich war.

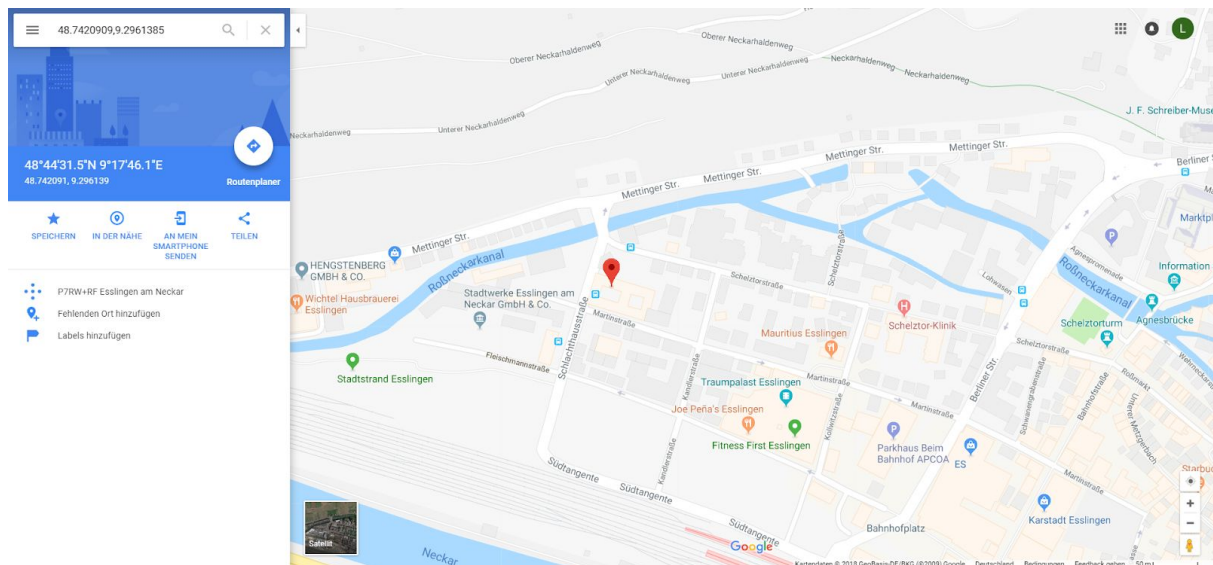


Abbildung 34: Die Koordinaten eines Work packages in Google Maps.

Die Abbildung 34 zeigt Google Maps mit einem Marker an der Stelle, an dem die Koordinaten des Work packages liegen. Die Koordinaten werden beim Erstellen des Work packages gespeichert. Sie sollen dem Bauleiter einen besseren geografischen Überblick der Mängelverteilung geben. Dadurch können Laufwege optimiert und logistische Operationen besser angepasst werden. Ein weiterer denkbarer Anwendungsfall wäre das Nutzen der Koordinaten für ein 3D Modell der Anwendung. Gäbe es eine Virtual Reality Anwendung, welche die Baustelle als Maßstabsgetreues 3D Modell darstellt, könnte man die Koordinaten dort eintragen. So könnte man einen einzigartigen visuellen Überblick über Mängel, Fortschritt und Geschehen auf der Baustelle bekommen.

### 8.1.4 Einstellungen

Die in den vorherigen Kapiteln beschriebenen Funktionen sollen natürlich nicht nur für ein bestimmtes Projekt, für einen bestimmten Nutzer zur Verfügung stehen. Deshalb gibt es die Möglichkeit verschiedene Einstellungen vorzunehmen. Zu den Einstellungen gelangt man im Header über einen Klick auf das "Einstellungen"-Icon. Abbildung 35 zeigt die Ansicht in den Einstellungen.



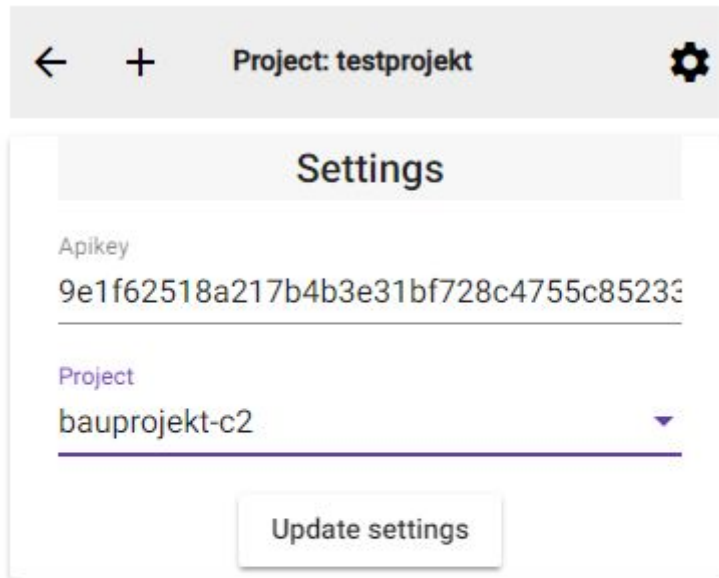


Abbildung 35: Settings.

Hier hat der Nutzer die Möglichkeit seinen Api Key ein zu tragen. Den Api Key kann sich jeder Nutzer über die Weboberfläche von OpenProject generieren lassen. Verliert man seinen Api Key, kann man sich einfach über die OpenProject Weboberfläche einen neuen generieren lassen. Der Api Key ist unique und wird für jeden HTTP Request benötigt. Er ist wichtig, um beim Erstellen eines Work packages automatisch den richtigen Autor/Ersteller des Work packages anzugeben.

Die zweite Möglichkeit, die die Einstellungen bieten, ist die Auswahl des gewünschten Projekts. Das Dropdown hierfür zeigt alle verfügbaren Projekte der OpenProject Plattform an. Hat man ein Projekt ausgewählt und über *update settings* seine Auswahl gespeichert, gelangt man zurück zur Startseite, der Listenansicht der Work packages. Die Liste besteht jetzt nur noch aus Work packages, die irgendwann einmal für das ausgewählte Projekt via PWA erstellt wurden.

## 8.2 Technische Umsetzung

### 8.2.1 Caching der Anwendung

Der Service Worker (Kapitel 7.3) sorgt für das Cachen der gesamten Anwendung.

In Angular 6 muss man der Anwendung lediglich einen Service Worker hinzufügen (Kapitel 7.3.2) und die *ngsw-config.json* Datei (Kapitel 7.3.3) an seine Bedürfnisse anpassen. Die *ngsw-config.json* der entwickelten Anwendung sieht folgendermaßen aus:

```

{
  "index": "/index.html",
  "assetGroups": [
    {
      "name": "app",
      "installMode": "prefetch",
      "resources": {
        "files": [
          "/favicon.ico",
          "/index.html",
          "/*.css",
          "/*.js"
        ],
        "urls": [
          "https://fonts.googleapis.com/**"
        ]
      }
    },
    {
      "name": "assets",
      "installMode": "lazy",
      "updateMode": "prefetch",
      "resources": {
        "files": [
          "/*.assets/**"
        ]
      }
    }
  ]
}

```

Abbildung 36: Die ngsw-config der PWA.

Man sieht, dass sämtliche *.js*, *.css*, die *index.html* und das favicon prefetch nach Installation der Anwendung gecached werden. Als URL werden die Google Fonts gecached, damit verwendete Icons und Schriften auch offline verfügbar sind. Ein Lazy Caching findet Performance bedingt nur bei den Assets statt.

Das Caching passiert nach folgendem Prinzip:

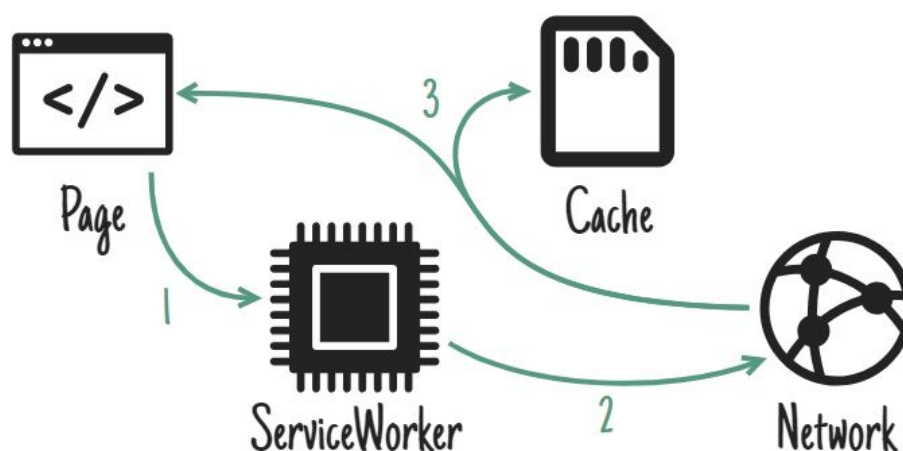


Abbildung 37: Service Worker on Network Response (Archibald, 2014)

Nach Aufruf der Webseite wird der Service Worker angesprochen. Dieser macht einen Request an das Netzwerk und überprüft, ob es eine neue Version der App gibt. Trifft dies zu

cached der Service Worker diese Version nach definierten Vorgaben (Kapitel 7.3.3) und zeigt sie dem Nutzer an. Besteht keine Internetverbindung werden die Daten aus dem Cache geladen und angezeigt.

## 8.2.2 Senden eines Work packages

Das folgende Sequenzdiagramm gibt eine Übersicht über die wichtigsten Services die angesprochen werden, sobald ein Work package erstellt wurde. Services die keinen Einfluss auf die direkten Daten haben wurden der Übersichtlichkeit halber nicht mit in das Diagramm aufgenommen. Hierzu zählt zum Beispiel der *SnackbarService*. Dieser sorgt für das Feedback, ob ein Work package gesendet werden konnte oder nicht.

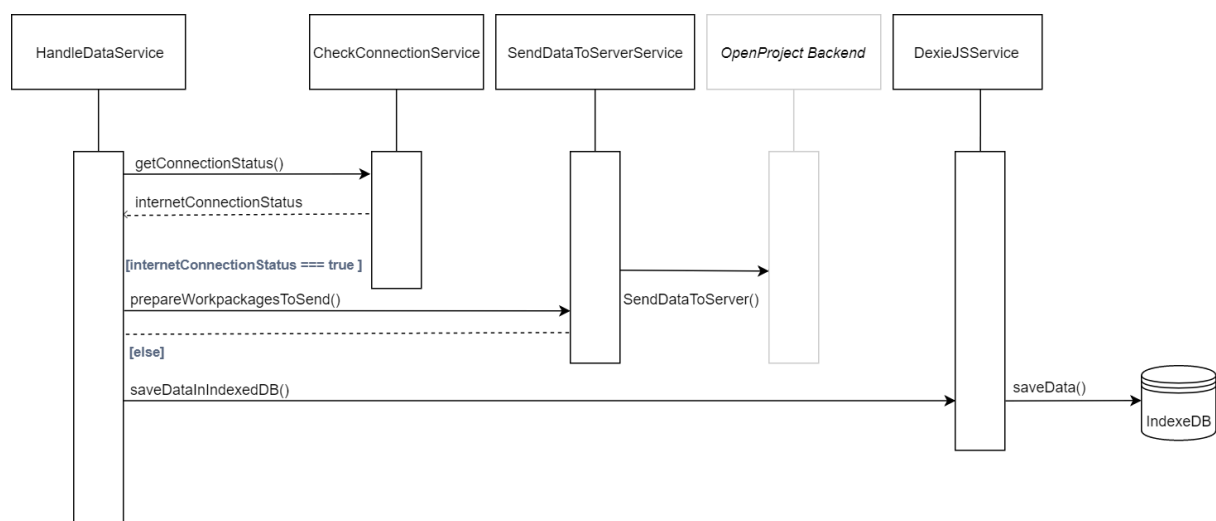


Abbildung 38: Senden eines Work packages.

Sobald ein Nutzer alle erforderlichen Eingaben für die Erstellung eines Work packages getätigt hat, die Form Validation *true* zurückgibt und schließlich den Button zum Bestätigen der Eingabe gedrückt hat, werden die in Abbildung 38 gezeigten Services aktiviert. Die *WorkpackageCreate* Komponente übermittelt alle Werte der Input Felder an den *HandleDataService*. Dieser wiederum fragt eine Variable des *CheckConnectionServices* ab.

Der *CheckConnectionsService* sieht folgendermaßen aus.

```
import { SendDataToServerService } from './send-data-to-server.service';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import 'rxjs/Rx';

@Injectable()
export class CheckConnectionService {

  online$: Observable<boolean>;
  internetConnectionStatus: boolean;

  constructor(public sendDataToServerService: SendDataToServerService) {

    this.online$ = Observable.merge(
      Observable.of(navigator.onLine),
      Observable.fromEvent(window, 'online').mapTo(true),
      Observable.fromEvent(window, 'offline').mapTo(false)
    )

    this.online$.subscribe(isOnline => {
      if (isOnline) {
        this.internetConnectionStatus = true;
      } else {
        this.internetConnectionStatus = false;
      }
    });
  }
}
```

Abbildung 39: *CheckConnectionService*.

Zuerst wird aus der *window.navigator.onLine*<sup>28</sup> Eigenschaft ein *Observable*<sup>29</sup> erstellt. Ein *Observable* ist ein Konstrukt der *rxjs*<sup>30</sup> Library.

Das erstellte Attribut gibt den Internet Status des Nutzers zurück. Das heißt, ist der Nutzer mit dem Internet verbunden wird *true* zurückgegeben, ist der Nutzer offline ist der Rückgabewert *false*.

Dieses *Observable* wird mit zwei weiteren *Observables* gemerged. Diese beiden *Observables* werden aus einem Event heraus erzeugt. Geht der Nutzer online wird das Event *online* gefeuert. Dadurch wird das *Observable online\$* auf *true* gemappt. Geht der Nutzer offline wird das Event *offline* getriggert und das *Observable online\$* wird auf *false* gemappt. Jetzt kann an das *Observable* subscribed werden und man erhält immer Auskunft über den derzeitigen Internet Status des Nutzers. Durch das subscriben wird die Variable *internetConnectionStatus* entweder auf *true* für online oder *false* für offline gestellt. Im Prinzip wird aktuell nur die Variable *internetConnectionStatus* benötigt, für künftige Features könnte das Nutzen des *online\$* *Observable* jedoch sinnvoll sein.

<sup>28</sup> <https://developer.mozilla.org/en-US/docs/Web/API/NavigatorOnLine/onLine>

<sup>29</sup> <https://angular.io/guide/observables/>

<sup>30</sup> <https://rxjs-dev.firebaseapp.com/>

Die Variable *internetConnectionStatus* wird von *HandleDataService* abgefragt.

Bekommt der *HandleDataService* *true* zurück, weiß dieser, dass die Anwendung online ist und das erstellte Work package gesendet werden kann. Daraufhin wird der *SendDataToServerService* aufgerufen, dem das zu sendende Work package übergeben wird. Dieser Service macht nun einen *POST* Request und sendet das Work package an die REST API von Open Project.

Bekommt der *HandleDataService* den Wert *false* zurück, weiß dieser, dass keine Internet Verbindung besteht und die Daten vorerst gecached werden müssen. Daraufhin wird die *saveDataInIndexedDB()* Methode aufgerufen. Diese wiederum spricht den *DexieJSService* an. Der *DexieJSService* wiederum speichert die Daten in der IndexedDB und ist generell für alle CRUD Operationen auf den Speicher verantwortlich.

### 8.2.3 Senden von Work packages (Intervall)

Kann ein Nutzer ein Work package nicht senden, da keine Internetverbindung besteht, wird dieses, wie in Abbildung 38 gezeigt, in der IndexedDB gespeichert. Jetzt soll das Work package gesendet werden, sobald der Nutzer wieder online ist. Hierfür wurde im *HandleDataService* ein Intervall Observable implementiert.

```
Observable.interval(1000).subscribe(x => {  
  if (this.checkConnectionService.internetConnectionStatus) {  
    this.sendDataToServerService.prepareWorkpackagesToSend();  
  }  
});
```

Abbildung 40: Intervall Observable.

Das Observable wird im Constructor der Klasse erstellt. Es prüft jede Sekunde ob sich der Internet Status der Anwendung geändert hat. Ist der Nutzer online (*internetConnectionStatus* === *true*) wird die *prepareWorkpackagesToSend()* Methode aufgerufen. Diese Methode sorgt dafür, dass die Work packages aus der *WorkpackagesToSend* Datenbank geladen werden und via *POST* Request an den Server geschickt werden. Nach erfolgreichem Senden werden die Work packages aus der Datenbank gelöscht und in die Datenbank *AllWorkpackages* eingefügt. Kapitel 8.2.4 gibt eine genauere Übersicht der verschiedenen Datenbanken.

### 8.2.4 Konzept der Datenbanken

Wie bereits in vorherigen Kapiteln beschrieben wurde zum Cachen der Work packages die IndexedDB verwendet. Als Abstraktionsschicht über der Datenbank wurde Dexie.js genutzt.

Die Anwendung beinhaltet zwei verschiedene Datenbanken innerhalb der IndexedDB. Die erste Datenbank wurde bereits in Kapitel 8.2.3 erwähnt. Diese beinhaltet alle Workpackages die noch nicht gesendet wurden. Die zweite Datenbank beinhaltet alle Work packages die von dem Endgerät jemals gesendet wurden. Abbildung 41 zeigt den Vorgang des Sendens eines Work packages anhand eines Flussdiagramms.

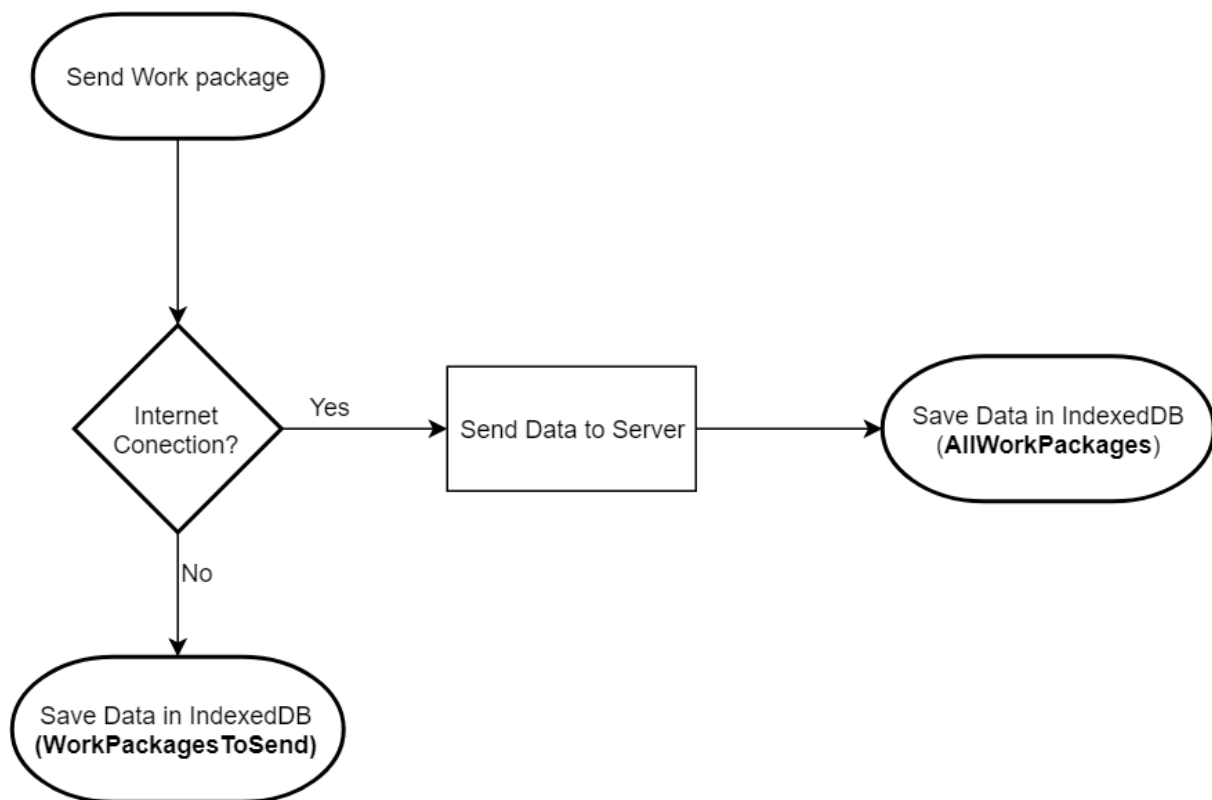


Abbildung 41: Flussdiagramm: Send Work package.

Nach dem Erstellen eines Work packages wird zuerst geprüft, ob der Nutzer mit dem Internet verbunden ist. Falls ja, werden die Daten direkt an den Server geschickt. Nach erfolgreichem Senden wird das Work package in der **AllWorkPackages** Datenbank gespeichert. Jeder Eintrag in der Datenbank ist ein Key-Value pair. Der Key ist ein String zur eindeutigen Identifizierung der einzelnen Values. Er wird zum Lesen und Löschen der Daten benötigt. Der Key wird von der OpenProject API zurückgegeben. Jedes Work package bekommt serverseitig eine eindeutige ID. Diese ID wird als Key für die einzelnen Work package Objekte benutzt.

Hat der Nutzer kein Internet wenn er ein Work package erstellt, wird eine andere Datenbank genutzt. In diesem Fall wird das Work package in der **WorkPackagesToSend** Datenbank gespeichert. Hierbei wird ein Key für jedes Work package erstellt. Dazu wird ein Timestamp generiert, der für jedes Work package unique ist. Dafür sorgt die `getTime()`<sup>31</sup> Methode.

<sup>31</sup> [https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global\\_Objects/Date/getTime](https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Date/getTime)

Das in Kapitel 8.2.3 beschriebene intervallbasierte Senden von Work packages und das damit verbundene Zusammenspiel der Datenbanken wird in folgendem Flussdiagramm genauer erläutert.

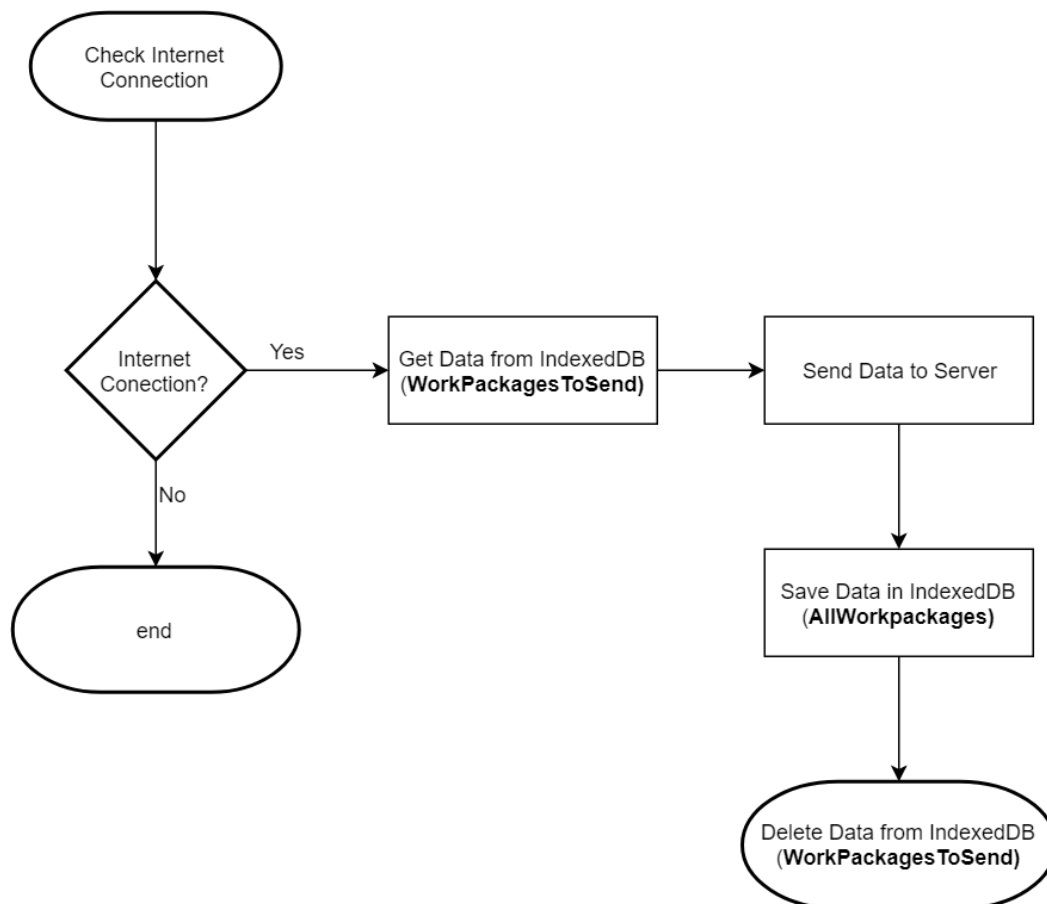


Abbildung 42: Flussdiagramm: Send Work package Intervall

Es wird jede Sekunde geprüft, ob eine Internetverbindung besteht. Besteht keine Verbindung, passiert nichts weiter, da dann auch keine Work packages gesendet werden können. Besteht jedoch eine Verbindung wird geprüft, ob in der IndexedDB (*WorkPackagesToSend*) Daten zum Senden vorhanden sind. Sind welche vorhanden, werden diese an das OpenProject Backend gesendet. Nach erfolgreichem Senden werden die Work packages in der *AllWorkpackage* Datenbank gespeichert. War das erfolgreich, werden die Daten aus der *WorkPackagesToSend* Datenbank gelöscht. Da die IndexedDb nur einen begrenzten Speicher hat, werden in der *AllWorkpackages* nicht mehr alle Attribute eines Work packages gespeichert.

Betrachtet man sich die *AllWorkpackages* Datenbank, könnte diese folgendermaßen aussehen:

| # | Key (Key path: "id") | Value  |
|---|----------------------|--|
| 0 | 14                   | ▼ {id: 14, title: "Fliesen legen - B7", sent: id: 14, sent: true, title: "Fliesen legen - B7"} |
| 1 | 15                   | ▶ {id: 15, title: "Fenster austauschen in Gebä"  |
| 2 | 16                   | ▶ {id: 16, title: "Abdeckung entfernen", sent:   |
| 3 | 18                   | ▶ {id: 18, title: "Wand streichen - Gebäude 3"   |

Abbildung 43: AllWorkpackages Datenbank.

Die Datenbank besteht aus zwei Spalten. Der Key ist gleichzeitig die ID des Work packages und wird von OpenProject Backend-seitig vergeben. Der Value ist ein JSON-Objekt mit mehreren Attributen:

**id:** Serverseitig generierte ID, eindeutig über alle Projekte hinweg.

**sent:** Ein Boolean, der aussagt ob das Work package erfolgreich an das Backend gesendet werden konnte.

**title:** Der Titel des Work packages.

Alle restlichen Attribute wie *Description*, *Attachments* oder *DueDate* werden nur noch serverseitig gespeichert. Grund dafür ist der endliche Speicher der IndexedDB. Sobald eine gewissen Anzahl an Work packages erstellt wurde, würde die Anzahl der Bilder den Speicher der IndexedDB überlasten.

Der Nutzer kann für jedes Projekt innerhalb OpenProject die PWA nutzen. Er muss in den Einstellungen nur das entsprechende Projekt auswählen. In den Einstellungen hat er auch die Möglichkeit seinen api key zu verändern. Sowohl api key, als auch das Projekt werden im LocalStorage gespeichert. Es handelt sich hierbei lediglich um Strings, die kaum Speicher brauchen, weshalb es nicht nötig ist, hierfür extra eine *Dexie*-IndexedDB Tabelle anzulegen. Die Informationen werden an vielen Stellen der Anwendung benötigt und Dank der synchronen Zugriffe auf den Localstorage wird ein Freezing der Anwendung vermieden.

## 8.3 Qualitätssicherung

### 8.3.1 Allgemeine nichtfunktionale Anforderungen

Im Folgenden wird erläutert, auf welche Art und Weise die in Kapitel 5.4.1 aufgestellten nichtfunktionalen Anforderungen umgesetzt wurden.

#### Sicherheit:

Für jeden HTTP-Request, der an die API von OpenProject gestellt wird, benötigt man einen Api Key. Dieser Api Key muss zu einem Benutzer gehören, der die Rechte für die



gewünschte Operation besitzt. Dieser Api Key kann innerhalb der OpenProject Web App generiert werden.

In der PWA muss ein gültiger Api Key in den Settings eingetragen sein, um die Funktionen der PWA zu nutzen. Das garantiert, dass nur berechtigte Nutzer Work packages anlegen oder bearbeiten können. Nutzer ohne Account der OpenProject Plattform haben somit keinen Zugriff auf die Funktionalitäten der App.

### **Effizienz:**

Diese Anforderung findet sich auch unter den konkreten PWA Anforderungen wieder und wird in Kapitel 8.3.3 behandelt.

### **Übertragbarkeit:**

Die entwickelte PWA basiert auf modernen und weit verbreiteten Web Technologien. Das garantiert eine Plattformunabhängigkeit der Anwendung. Durch die verwendeten Technologien (Kapitel 7) ist die entwickelte Anwendung sowohl als einfache Web App verfügbar, wie auch als "klassische" App, nachdem sie heruntergeladen wurde.

### **Änderbarkeit:**

Die PWA bietet eine Struktur, die es ermöglicht ohne größeren Aufwand die Anwendung um zusätzliche Funktionalitäten zu erweitern. Möchte man beispielsweise den verwendeten Datenbank Wrapper *Dexie.js* austauschen, muss man lediglich die CRUD Operationen in einer einzigen Service Klasse auf die API der neuen Library anpassen.

## **8.3.2 Anforderungen an die PWA**

Um die in Kapitel 5.4.2 definierten Anforderungen an eine PWA zu überprüfen, wurde das Tool *Lighthouse*<sup>32</sup> verwendet. Lighthouse ist eine von Google entwickelte *Google Chrome* Erweiterung, um die Performance, Accessibility und den "PWA-Faktor" einer Anwendung zu messen. Der "PWA-Faktor" orientiert sich an den in Kapitel 5.4.2 erläuterten Anforderungen. Nach einer Analyse mit Lighthouse entstand folgendes Ergebnis:

---

<sup>32</sup> <https://developers.google.com/web/tools/lighthouse/>

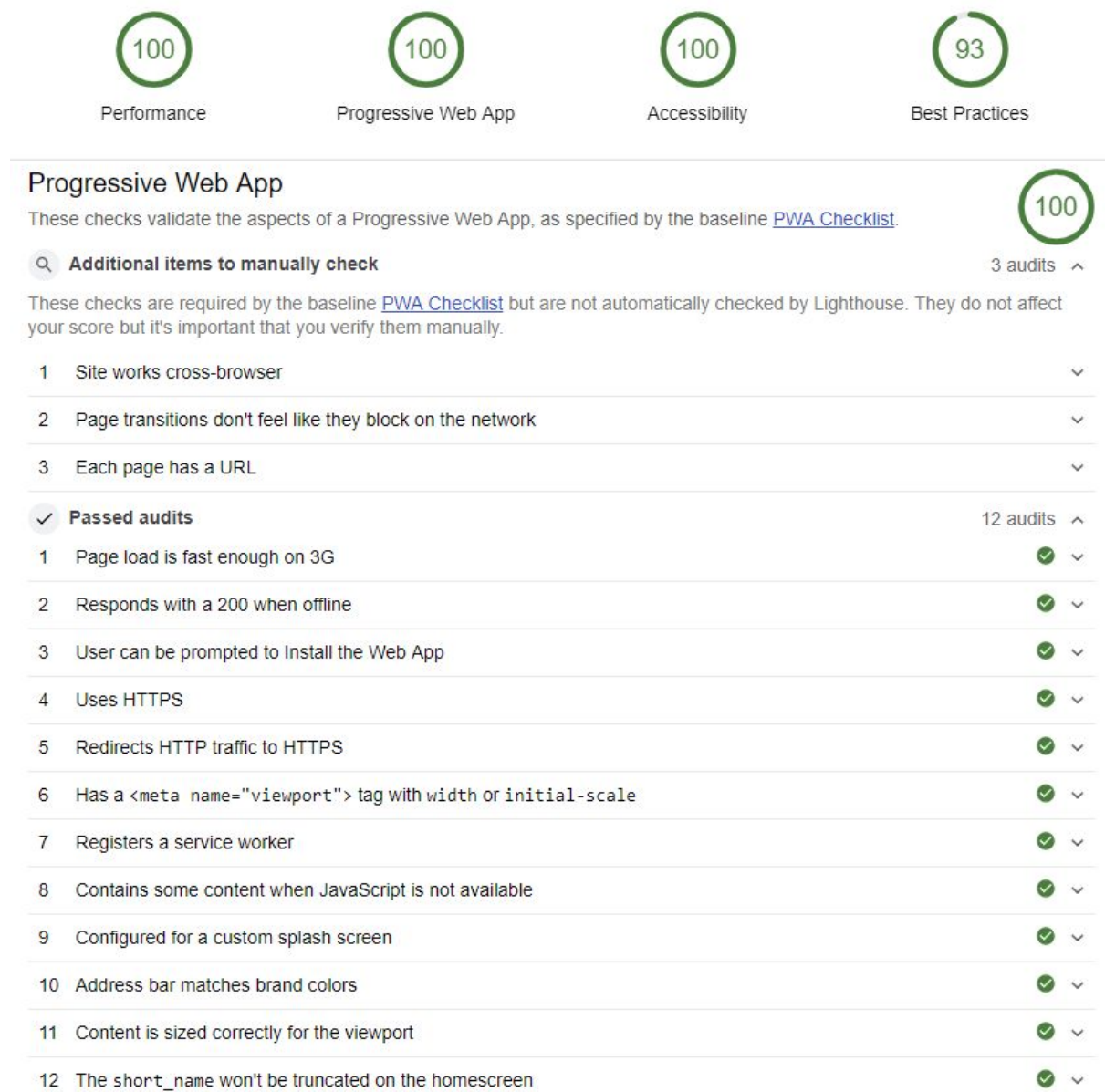


Abbildung 44: Lighthouse - PWA Analyse.

Wie man in Abbildung 44 sieht, wurden alle Kriterien für eine Progressive Web App erfüllt. Es werden noch drei Punkte genannt, die manuell getestet werden sollten.

**Site works cross-browser:** Die Cross-Browser Kompatibilität ist in der vorliegenden Anwendung nur für den aktuellen mobilen Safari Browser und den aktuellen mobilen Google Chrome Browser notwendig. Diese Kompatibilität wurde auf verschiedenen Endgeräten getestet.

**Page transitions don't feel like they block on the network:** Dank schneller Übergänge innerhalb der Anwendung und dank durchgehendem Feedback, in welchem Status sich die Anwendung befindet, wirkt es nicht, als ob die Anwendung das Netzwerk blockiere .

**Each page as URL:** Dank der single-page-Architektur von Angular ist für jede Ansicht eine eigene URL verfügbar.

## 9 Vergleich OpenProject PWA & OpenProject Webanwendung

### 9.1 Allgemein

Dieses Kapitel soll einen grundlegenden Vergleich zwischen der bereits bestehenden desktoptimierten Webanwendung von *Openproject.org* und der in Kapitel 8 beschriebenen, entwickelten PWA bieten.

Zum Vergleich werden beide Anwendungen in ihrer mobilen Ansicht betrachtet (Maße: 375 x 812, entspricht dem iPhone X) und auf konkrete Usability Eigenschaften untersucht. Hierfür werden die in Kapitel 6 entworfenen Usability Richtlinien verwendet. Als Anwendungsfall dient ein Beispiel aus der Praxis innerhalb der Baubranche (Vergleich Beispielszenario, Kapitel 5.2.1). Im Folgenden wird die bereits bestehende desktoptimierte Version von OpenProject als Web App und die entwickelte Progressive Web App als PWA bezeichnet.

Betrachten wir zuerst die konkrete Problemstellung bzw. Anforderung an die Anwendung anhand der Definition von Usability aus Kapitel 6.1:

Das Produkt ist in dem Fall die Progressive Web App.

Das Ziel ist die Erstellung und das Anzeigen von Work packages.

Die Effektivität bedeutet, dass um das Ziel zu erreichen, nur Maßnahmen ergriffen werden sollen, die dem Erreichen des Ziels nutzen. Konkret könnte das bedeuten, dass z.B. nur Eingabefelder angezeigt werden, die für die Erstellung eines Work packages wirklich notwendig sind.

Die Effizienz sagt aus, dass das Ziel mit möglichst wenig Aufwand und Ressourcen erreicht werden soll. Konkret könnte dies bedeuten, dass man die UI-Elemente so anordnet, dass der Benutzer schnellstmöglich ein Work package erstellen kann.

### 9.2 Work package erstellen

#### 9.2.1 Allgemeine Richtlinien

Das Erstellen eines Anwendungsfalls soll möglichst schnell und mit wenigen Klicks erfolgen.

Ist man auf dem Dashboard der Projektübersicht innerhalb der Web App, benötigt man zwei Klicks, um auf die gewünschte Ansicht zu kommen. Innerhalb der PWA ist es nur ein Klick.

Es gibt zwei verschiedene Arten in der Web App ein Work package zu erstellen. Einmal geschieht dies über *Project* (links oben in der Topleiste) → *+Create Task*. Die andere Möglichkeit funktioniert ohne die vorherige Auswahl eines Projekts. *Work packages* (rechts oben in der Topleiste) → *+Work package*.

The image displays two side-by-side screenshots of the OpenProject web application's 'New Task' form.

**Left Screenshot (Without Selected Project):**

- Header:** 'Select a project' dropdown menu.
- Buttons:** '+ Work package' (green), '+ Create Task' (blue).
- Form Fields:** 'New' dropdown menu, 'Please select' dropdown menu, 'Project \*' dropdown menu.
- DESCRIPTION:** Rich text editor with a toolbar (Paragraph, Bold, Italic, Link, Unlink, Bulleted List, Numbered List, Image, Quote, Table, Undo, Redo, Help, Embed).
- FILES:** 'Drop files here or click to add files' area.

**Right Screenshot (Within a Selected Project):**

- Header:** 'Demo project' dropdown menu.
- Buttons:** '+ Work package' (green), '+ Create Task' (blue).
- Form Fields:** 'New' dropdown menu, 'Please select' dropdown menu, 'Project \*' dropdown menu.
- DESCRIPTION:** Rich text editor with a toolbar (Paragraph, Bold, Italic, Link, Unlink, Bulleted List, Numbered List, Image, Quote, Table, Undo, Redo, Help, Embed).
- PEOPLE:** 'Assignee' dropdown menu, 'Accountable' dropdown menu.
- ESTIMATES AND TIME:** 'Estimated time' input field (0), 'Remaining Hours' input field (0).
- DETAILS:** 'Date' input field, 'Progress (%)' input field (0), 'Category' dropdown menu, 'Version' dropdown menu, 'Priority' dropdown menu (Normal).
- COSTS:** 'Budget' dropdown menu.

Abbildung 45: OpenProject - Create Work package: ohne ausgewähltes Projekt (links); innerhalb eines Projekts (rechts).

In dem folgenden Vergleich wird von der Web App nur die in Abbildung 45 rechts abgebildete Variante, ein Work package zu erstellen, untersucht. Prinzipiell ist hier jedoch zu erwähnen, dass es keine zwei komplett verschieden aussehende Ansichten mit derselben Funktionalität geben sollte.

In Abbildung 46 sind die beiden Ansichten des Work package Erstellens zu sehen. Links ist die Ansicht der Web App zu sehen und rechts die Ansicht der PWA.

Demo project

New Task

DESCRIPTION

Paragraph

B

I

<>

<>

PEOPLE

Assignee

Accountable

ESTIMATES AND TIME

Estimated time

0

Remaining Hours

0

DETAILS

Date

-

Progress (%)

0

Category

Version

Priority

Normal

COSTS

Budget

FILES

Drop files here  
or click to add files

✓ Save

✕ Cancel

+

Project: demo2-projekt

Create a new work package

Title \*

Description

Advanced data

+

Image Upload

Create Work package

Abbildung 46: Create a Work package: PWA (rechts), Web App (links).

## 9.2.2 Formular spezifische Richtlinien

Die nachfolgende Tabelle bezieht sich auf die in Kapitel 6.5 aufgestellten formularspezifischen Designrichtlinien und wendet diese auf die PWA und die Web App an. Es wird beschrieben wie, bzw. ob die Anwendungen die einzelnen Punkte erfüllen, und mit welchen Designrichtlinien das begründet werden kann.

### Navigation & Seitenstruktur:

| Progressive Web App  | Web App   |
|--|---|
| <b>Positionierung</b>  |   |
| Die Navigationsleiste befindet sich im Header der Anwendung und erfüllt damit die Erwartung der Nutzer.                                      | Die Navigationsleiste befindet sich im Header der Anwendung und erfüllt damit die Erwartung der Nutzer.                         |
| <b>Gestalt &amp; Inhalt</b>  |   |
| Man sieht im Header immer, in welchem Projekt man sich aktuell befindet. Durch einen Klick auf den Header gelangt man zurück zur Startseite. | Man sieht im Header immer, in welchem Projekt man sich aktuell befindet. Es gibt keinen "1-Click"-Escape innerhalb des Headers. |

Abbildung 47: Navigation & Seitenstruktur: PWA (links), Web App (rechts).

### Input Felder:

| Progressive Web App   | Web App   |
|---|---|
| <b>Anzahl</b>   |   |
| Die Anzahl der Input Felder wurde auf das Minimum begrenzt. Die wichtigsten Inputfelder, die aus den Anforderungen hervorgegangen sind (Title, Description, Bilder) sind durchgängig zu sehen. Weniger oft benötigte Input Felder werden dank eines "Expand Panel" versteckt, bis diese tatsächlich gebraucht werden.<br><br>Keine der gesammelten Informationen kann über einen anderen Weg gesammelt werden. Benötigt man Input Felder, die nicht in der PWA vorhanden sind, kann man diese Attribute später über den Webview | Für den zugrunde liegenden Anwendungsfall bildet die Web App zu viele nicht benötigte Input Felder ab. Input Felder wie <i>Category</i> oder <i>Priority</i> werden für den vorliegenden Anwendungsfall nicht benötigt. |

|  |   |
|--|---|
| <p>einpflegen. Das Attribut des Work package "Type" wurde bewusst weggelassen, da alle WPs vom Type Task sein sollen.</p>  |   |
| <b>Positionierung</b>  |   |
| <p>Dank der Verwendung von <i>AngularFlexLayout</i> und <i>Angular Material</i> ist eine symmetrische Anordnung unkompliziert möglich. Es wird auch eine räumlich Trennung von <i>basic Input</i> Felder und <i>advanced Input</i> Feldern vorgenommen und somit das Gesetz der Nähe gewahrt.</p>  | <p>Die Input Felder sind thematisch zugehörig angeordnet. Die Abstände sind jedoch etwas zu groß und verschwenden dadurch Platz.</p>  |
| <b>Gestalt &amp; Größe</b>   |   |
| <p><i>Angular Material</i> bietet eindeutige, als solche erkennbare Input Felder. Das <i>Description</i> Feld ist deutlich größer, als die restlichen Felder. Das soll den Nutzer ermutigen, eine ausführliche Beschreibung des Work packages einzugeben. Die Input Felder bieten durch kleine Animationen Feedback und signalisieren dadurch, dass sie angeklickt wurden.</p> | <p>Die Größe des Input Feldes <i>Description</i> ist zu groß und bietet für einen mobile View zu viele Formatierungsmöglichkeiten. Das führt zum Verlust von Platz, der für anderen Inhalte genutzt werden könnte. Die Input Felder bieten Feedback, nachdem man sie anklickt. Leider ist dieses Feedback jedoch inkonsistent. Beispielsweise wird der Rahmen bei dem <i>Description</i> Input Feld blau, während er bei dem <i>title</i> Input Feld schwarz wird.</p> <p>Das Input Feld des <i>Progress (%)</i> ist zu groß, da es hier eine maximale Inputgröße von 3 Ziffern (100) geben kann.</p> |
| <b>Inhalt</b>  |   |
| <p>Alle Input Felder sind standardmäßig mit einem extra Label ausgestattet. Das Pflichtfeld <i>title</i> ist als solches deutlich gekennzeichnet.</p>  | <p>Für <i>title</i> Input fehlt das Label, was zu Verwirrungen führen kann. Auch der File Input ist nicht mobile optimiert, "Drop files here" ist über mobile Geräte nicht möglich. Ansonsten hat jedes Input Feld ein Label, was positiv hervorzuheben ist. Es ist vor dem Klick auf den Submit Button leider nicht erkennbar, dass das <i>title</i> Feld ein Pflichtfeld ist.</p>   |
| <b>Das richtige Input Feld für den richtigen Input</b>   |   |



|   |   |
|---|---|
| Bei fast allen Input Feldern hat der klassische <i>Input Schlitz</i> Sinn gemacht. Nur der Datepicker macht bei der Auswahl eines Datums mehr Sinn, als eine freie Eingabe durch den Nutzer.  | Hier wurden die richtigen Input Felder für die zugrunde liegenden Daten gewählt. Nur der Datepicker erscheint leider nicht mittig und ist nicht zu einhundert Prozent smartphonegerecht. Auch ist das <i>Date</i> Input Feld etwas zu klein, um seinen Inhalte komplett anzuzeigen. |
| <b>Label Positionierung</b>   |   |
| Die Labels sind dank <i>Material</i> schon immer direkt oberhalb des Inputs angeordnet.   | Prinzipiell sind die Labels immer links des Input Feldes angeordnet ist, was gestalterisch noch akzeptabel ist.   |
| <b>Fehlermeldungen</b>  |   |
| Dank der implementierten Form Validation (Kapitel 8.1.1) werden dem Nutzer, wenn sinnvoll, nur bestimmte Eingaben erlaubt, oder ihm wird Mitteilung gegeben, wie er seinen Input zu verändern hat, damit dieser akzeptiert wird. Die Fehlermeldung wird immer nahe dem Input Feld angeordnet. | Es gibt eine aussagekräftige Fehlermeldung, die anzeigt, dass der <i>title</i> ein Pflichtfeld ist. Es gibt jedoch keine Formvalidation, die besagt, dass das <i>start date</i> vor dem <i>due date</i> liegen muss.  |

Abbildung 48: Input Felder: PWA (links), Web App (rechts).

#### Buttons:

| Progressive Web App   | Web App  |
|---|--|
| <b>Position</b>   |  |
| Der Submit Button befindet sich nahe bei den Input Felder.                                | Der Submit Button befindet sich unterhalb des Formulars. Er könnte jedoch, um Platz zu sparen, etwas näher zu den Input Feldern angeordnet werden. |
| <b>Gestalt &amp; Größe</b>  |  |
| Der Button ist ausgegraut, solange die Form Validation (Kapitel 8.1.1) nicht korrekt ist. | Der Button hat leider immer die gleiche Form/Farbe unabhängig vom Status der Form Validation.  |
| <b>Reset / Cancel Button vermeiden</b>  |  |

|   |  |
|---|--|
| Es existiert kein Reset/Cancel Button. Will der Nutzer den Vorgang wirklich abbrechen, kann er diese über den “Zurück”-Pfeil oder einen Klick auf den Header tun. | Es existiert ein Cancel Button nahe dem Submit Button. |
|---|--|

Abbildung 49: Buttons: PWA (links), Web App (rechts).

## 9.3 Work package Detailansicht

In diesem Kapitel wird die Work package Detailansicht der entwickelte PWA mit der Work package Detailansicht der bestehenden Web App verglichen.

Abbildung 50 zeigt die beiden Ansichten nebeneinander.

demo2-Projekt

←

↻

⋮

Task: Fenster reparieren

#59: Created by p wa. Last updated on 09/04/2018 2:52 PM.

New

DESCRIPTION

Das Fenster im Aufenthaltsraum 3 ist kaputt. Es lässt sich nicht mehr kippen.

<https://www.google.com/maps/search/48.7420924,9.296165>

PEOPLE

|             |   |
|-------------|---|
| Assignee    | - |
| Accountable | - |

ESTIMATES AND TIME

|                 |         |
|-----------------|---------|
| Estimated time  | 2 hours |
| Spent time      | -       |
| Remaining Hours | 2 hours |

DETAILS

|              |               |               |
|--------------|---------------|---------------|
| Date         | no start date | - no end date |
| Progress (%) | <div>0%</div> |               |
| Category     | -             |               |
| Version      | -             |               |
| Priority     | Normal        |               |

COSTS

|               |   |
|---------------|---|
| Overall costs | - |
| Labor costs   | - |
| Unit costs    | - |
| Spent units   | - |
| Budget        | - |

FILES

📎

Drop files here or click to add files

ACTIVITY

RELATIONS

WP

August 30, 2018

🔊

p wa

created on 08/30/2018 10:56 AM

September 3, 2018

🔊

p wa

updated on 09/03/2018 1:50 PM

- Subject changed from t2 to fenster in 3.Stock reparieren

September 4, 2018

🔊

p wa

updated on 09/04/2018 2:52 PM

- Subject changed from fenster in 3.Stock reparieren

Project: demo2-projekt

Work package: Fenster reparieren

⚙️

Work package details

ID

59

Title

Fenster reparieren

Description

Das Fenster im Aufenthaltsraum 3 ist kaputt. Es läs...  
[Google Maps coordinates](#)

Number of Images

0

+ Add image

Estimated Time

2 Hours

Remaining Time

2 Hours

Progress

0 %

Start date

-

Due date

-

Abbildung 50: Work package Detailansicht: PWA (rechts), Web App (links).

Da diese Ansicht nicht formularbasiert ist, macht die Anwendung der in Kapitel 6.5 erarbeiteten Usability Richtlinien wenig Sinn. Statt dessen wird im Folgenden ein Vergleich anhand der allgemeineren Usability Kriterien (Kapitel 6.4) durchgeführt.

| Progressive Web App   | Web App   |
|---|---|
| <b>Gestaltpsychologische Prinzipien</b>   |   |
| <p>Label und Text wurden nahe beieinander angeordnet. Eine Trennlinie und eine farbliche Abgrenzung dienen dazu, die einzelnen Labels und Texte als zusammengehörig wahrzunehmen. Eine grundsätzliche symmetrische Anordnung existiert.</p>   | <p>Prinzipiell existiert eine gewisse Symmetrie bei der Anordnung der Elemente. Auch Label und Input Feld sind zusammen angeordnet. Hier ist leider das Label nicht unbedingt als solches zu erkennen. Hier fehlt die Umrandung (Gesetz der Geschlossenheit). Thematisch sind die Elemente nach dem Gesetz der Nähe angeordnet.</p> |
| <b>Sichtbarkeit des Systemstatus / Selbstbeschreibungsfähigkeit</b>   |   |
| <p>Der Header bietet einen Überblick über das Work package und das Projekt in dem man sich gerade befindet. Der Spinner gibt Feedback, dass die Anwendung gerade lädt, bzw. der Request an das Backend etwas länger dauert. Wenn kein Internet vorhanden ist und die Detailseite nicht geladen werden kann, gibt ein Pop-Up darüber Auskunft.</p> | <p>Der Header gibt hier kein Feedback wo man sich in der Anwendung gerade befindet, dafür aber eine gut erkennbare Überschrift.</p>   |
| <b>Aufgabenangemessenheit, Ästhetik und minimalistisches Design &amp; Simplicity</b>  |   |
| <p>Die Anwendung zeigt alle für den Anwendungsfall notwendigen Daten an. Unwichtige Daten werden verborgen. Eine gewisse Form von Ästhetik ist durch das verwendete Design Framework Material gegeben.</p>  | <p>Es werden auch nicht benötigte Attribute angezeigt. Auf dem Iphone X ist die Ansicht nur bis <i>Details</i> sichtbar bevor man anfangen muss zu scrollen. Für die mobile Ansicht sind zu große Abstände gewählt worden.</p>  |
| <b>Erwartungskonformität, Konsistenz und Standards &amp; Erkennen statt Erinnern</b>  |   |
| <p>Durch das Nutzen eines weit verbreiteten Frameworks ist ein Standard und eine Konsistenz von Design gegeben.</p>   | <p>Die Anwendung folgt einem Styleguide und wirkt dadurch konsistent.</p>   |

**Fehlertoleranz & dem Nutzer helfen, Fehler zu erkennen, zu diagnostizieren und wiederherzustellen**

Besteht keine Internetverbindung, wird das dem Nutzer mitgeteilt.

-

Abbildung 51: Work package Vergleich Detailansicht: PWA (rechts), Web App (links).

## 9.4 Work packages Listenansicht

In Abbildung 52 ist die Gegenüberstellung der Web App Listenansicht und der PWA Listenansicht zu sehen.

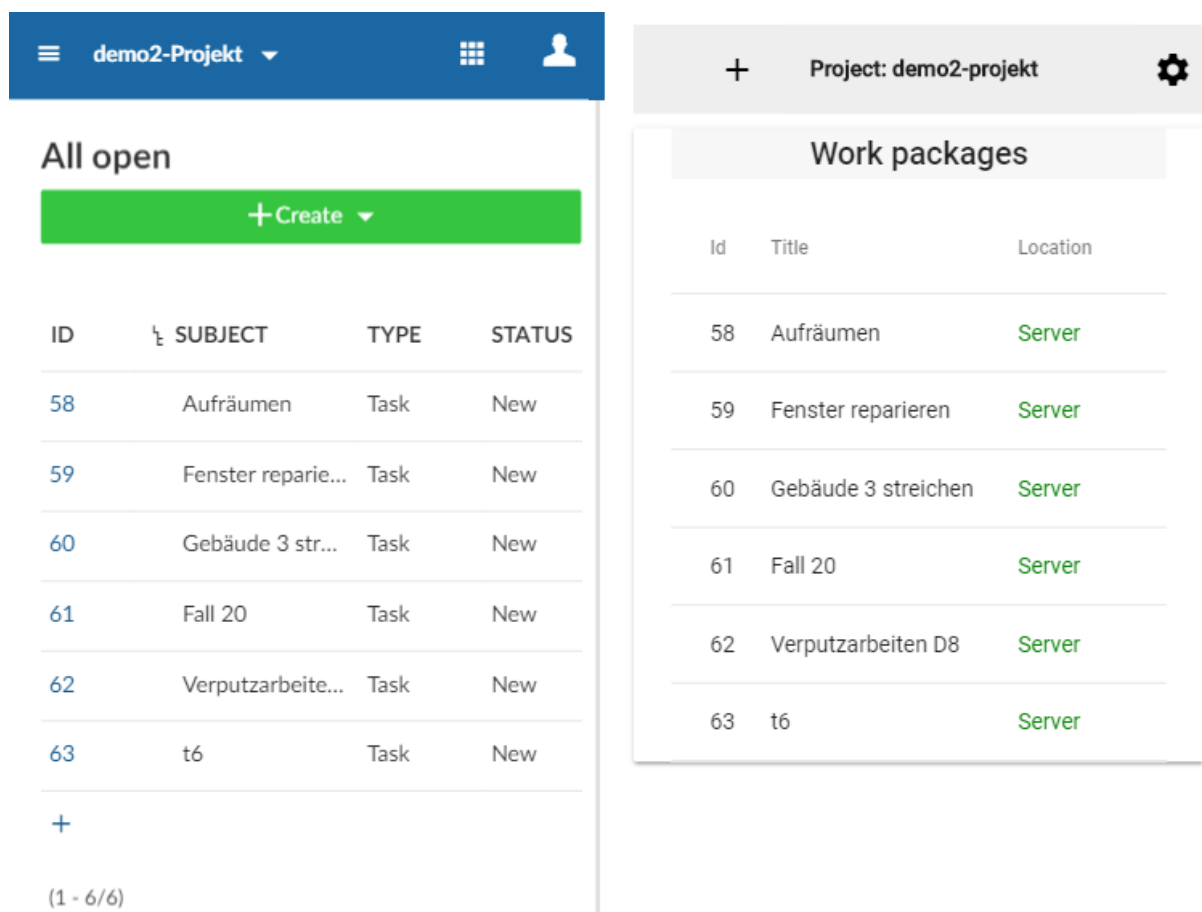


Abbildung 52: Work package Listenansicht: Web App (links), PWA (rechts).

Man sieht bei beiden Ansichten den Header, der dem Nutzer auch in dieser Ansicht Feedback gibt, wo er sich gerade in der Anwendung befindet. Der +Create-Button nimmt etwas zu viel Platz innerhalb der Web App ein, während in der PWA der +-Button in den Header ausgelagert wurde. Auch die Anzahl der angezeigten Attribute ist in der Web App

nicht auf den Anwendungsfall optimiert. In der PWA ist es durch das Anzeigen der wichtigsten Eigenschaften möglich, auch längere Titel in Gänze anzuzeigen.

## 10 Fazit & Ausblick

In der vorliegenden Arbeit wurden das Potential aufgezeigt, das eine Progressive Web App für bestimmte Prozessoptimierungen bietet. Es wurde anhand eines konkreten Anwendungsfalls eine PWA entwickelt, und diese mit der bestehenden Web App von *OpenProject* anhand eigens erstellter Usability Richtlinien untersucht.

Am Anfang der Arbeit wurden die Unterschiede zwischen verschiedenen App Typen (Native Apps, Web Apps, Hybriden Apps und PWAs) dargestellt und eine eindeutige Definition des Begriffs der PWA festgelegt. Daraufhin wurde das Projektmanagement innerhalb der Baubranche mit all seinen Prozessen und Abläufen erläutert. Danach wurde eine Übersicht verschiedener Projektmanagement Softwaretypen erstellt, und ein möglicher Einsatz solcher Software in der Baubranche untersucht. Hierbei wurde vor allem der Teilprozess des Mängelmanagements genauer betrachtet, da die PWA für diese Phase innerhalb eines Bauprojekts entwickelt wurde. Im Anschluss wurde der Begriff des Mobile Enterprise definiert, da die entwickelte Anwendung in diesen Bereich von Geschäftsprozessoptimierung fällt.

Anschließend wurde die Projektmanagement Plattform *OpenProject* ausführlich mit ihren Funktionalitäten vorgestellt. Die entwickelte PWA orientiert sich an den bereits vorhandenen Funktionalitäten der *OpenProject* Web App. Bevor die Implementierung der Anwendung erläutert wurde, erfolgte eine Anforderungsanalyse und einen Softwareentwurf. Hier wird die Zielgruppe und der Anwendungsfall der PWA genauer beschrieben. Nutzer der App sind vor allem Bauleiter, die mit Hilfe der mobilen App Mängel, Ideen oder Task in bestehende Projekte der *OpenProject* Plattform einpflegen können.

Dieser Entwurf wurde anhand von festgelegten Usability Richtlinien implementiert, um Nutzern einen Mehrwert zur aktuellen Anwendung zu garantieren. Hierbei handelt es sich sowohl um allgemeine User Interface Heuristiken bzw. gestaltpsychologische Prinzipien, als auch um spezielle Anforderungen, die nur an formularbasierte Anwendungen gestellt werden.

Die Implementierung wurde sowohl von technischer Seite, als auch von funktionaler Seite ausführlich beleuchtet. Zudem wurden auch die Qualitätszielbestimmungen, welche die Anwendung erfüllen soll, untersucht. Hierbei handelt es sich um allgemein bekannte, nicht funktionale Anforderungen der Softwareentwicklung, wie auch speziell auf eine PWA zugeschnittene Anforderungen.

Anhand der entwickelten Usability Richtlinien wurden die PWA und die bereits bestehende Web App von *OpenProject* untersucht und verglichen. Der Vergleich basierte auf dem bereits erwähnten Anwendungsfall aus der Baubranche. Es wurden hier ein Vergleich auf Grundlage der allgemeinen Usability Prinzipien, als auch ein Vergleich anhand der formularspezifischen Richtlinien erstellt.

In der Arbeit wurden gleich mehrere Aspekte untersucht. Zuerst wurden die Vorteile einer Progressiven Web App im Vergleich zu anderen App Typen erarbeitet. Es wurden die Möglichkeiten und der Nutzen einer PWA veranschaulicht dargestellt. Es wurde belegt, welche Bedeutung die Merkmale der PWA (z.B. Geschwindigkeit) auf potentielle Nutzergruppen haben.

Einen Schwerpunkt der Arbeit bildete das Zusammenfassen von Usability Richtlinien aus diversen Quellen der Forschung. Hierfür wurden die wichtigsten allgemeinen Richtlinien für User Interface Design zusammengetragen. Anhand dieser Zusammenfassung wurden formalspezifische Richtlinien zur Qualitätssicherung der PWA erarbeitet. Hierbei wurden Designregeln für Form Felder, Buttons und die Navigationsstruktur erstellt.

Sowohl die allgemeinen Richtlinien, als auch die spezifischen Formular-Richtlinien wurden genutzt, um einen Vergleich zwischen der aktuellen mobilen Ansicht der *OpenProject* Plattform und der entwickelten PWA durchzuführen. Ziel dieses Vergleichs war es, herauszufinden ob die PWA eine bessere Usability bezüglich des zugrunde liegenden Anwendungsfalls bietet, als es der bisherige mobile View der Web App tut. Die Annahmen waren, dass die Anwendung, die mehr Usability Richtlinien erfüllt, den Nutzer schneller, effektiver und effizienter arbeiten lässt. Das wiederum wirkt sich auf ein Performance Anstieg des gesamten Prozesses aus. Das Ergebnis zeigt eindeutig, dass sowohl die allgemeinen, als auch die spezifischen Richtlinien innerhalb der PWA deutlich mehr Anwendung finden, als es bei der bestehenden Web App der Fall ist. So hat der bestehende View beispielsweise für den Anwendungsfall irrelevante Inputfelder oder verwehrt dem Nutzer durch zu große Abstände einen kompakten Überblick der Work package Details.

Es konnte hierbei gezeigt werden, dass in allen drei entwickelten Funktionalitäten (Work package erstellen, Work package Liste anzeigen, Work package Details anzeigen) die PWA die besser Usability bietet.

Betrachtet man die noch junge Geschichte der PWA, so lässt das erahnen, dass noch nicht alle Möglichkeiten und Anwendungsgebiete erforscht wurden. In den nächsten Jahren wird sich zeigen, ob sich die PWA allgemein in der Gesellschaft und der Wirtschaft durchsetzen wird. Die Anwendung hat aufgezeigt, dass eine PWA durchaus zur Prozessoptimierung in gewissen Wirtschaftszweigen eingesetzt werden kann. Vor allem bei formularbasierten Apps sehe ich großes Potential. Formularbasierte Apps und Apps, die keine aufwendigen clientseitigen Berechnungen durchführen müssen (z.B. 3D-Grafiken) sind prinzipiell gut dafür geeignet, mit Webtechnologie entwickelt zu werden. Der entscheidende Vorteil ist hierbei die Single-Code-base.

Für Anwendungen, die später sowohl am Computer im Browser, als auch von unterwegs mobil genutzt werden sollen ist eine PWA ebenfalls empfehlenswert. Man kann so mit einer Single-Code-base eine Desktop Web App, eine Mobile Web App und eine Progressive Web App umsetzen. Durch die offline Funktionalität wirkt die PWA zwar wie eine native App, bietet aber trotzdem die Möglichkeit des BrowservIEWS und den Vorteil der Single-Code-Base.

Die Arbeit bietet einen grundlegenden Überblick über das Thema der Progressiven Web Apps. Es wird gezeigt, wofür eine derartige Technologie verwendet werden kann, und wie dadurch Geschäftsprozesse optimiert werden können. Die entwickelte PWA ist auf einem Stand, in dem sie in der Praxis eingesetzt werden kann. Dennoch gibt es einige Features, die in zukünftigen Arbeiten ergänzt werden könnten. Hier ist zum Beispiel das Bearbeiten eines Work packages zu nennen. In zukünftigen Arbeiten könnte man dieses sicherlich nützliche Feature einbauen, um so von mobilen Endgeräten Work packages bearbeiten zu können.

Auch könnte man die gesamte Funktionalität noch ausbauen, um noch mehr Teilprozesse eines Projekts unterstützen zu können. Es wäre möglich für jeden Prozess innerhalb eines Projektes die dazugehörigen Features von *OpenProject* in die PWA zu integrieren.

Ein weiterer interessanter Ansatz, die Arbeit fortzuführen, wäre der technische Vergleich der Web App und der PWA. Hierbei könnte man prüfen, ob die Performance der App signifikante Unterschiede zur bereits bestehenden Web App bietet und das wiederum Auswirkungen auf die gesamte User Experience hat.

Auch wäre es denkbar, die Usability Richtlinien beispielsweise bezüglich tabellenspezifischer Richtlinien zu erweitern und anhand derer einen Vergleich beider Anwendungen durchzuführen.



# Literaturverzeichnis

- Accenture (2012). Mobile Web Watch 2012. Retrieved from [https://www.accenture.com/t20151123T001914\\_\\_w\\_\\_/\\_ch-de/\\_acnmedia/Accenture/Conversion-Assets/DotCom/Documents/Local/de-ch/PDF/Accenture-Mobile-Web-Watch-Internet-Usage-Survey-2012.pdf](https://www.accenture.com/t20151123T001914__w__/_ch-de/_acnmedia/Accenture/Conversion-Assets/DotCom/Documents/Local/de-ch/PDF/Accenture-Mobile-Web-Watch-Internet-Usage-Survey-2012.pdf) (accessed 30.08.2018).
- Accessibility (n.d.). Retrieved from <https://material.io/design/usability/accessibility.html> (accessed 31.08.2018).
- Ahlemann (2003). *Comparative Market Analysis of Project Management Systems* (p. 22). Osnabrück: Universität Osnabrück FB Wirtschaftswissenschaften.
- AliExpress (2016). AliExpress increases conversion rate for new users by 104% with new Progressive Web App. Retrieved from <https://developers.google.com/web/showcase/2016/pdfs/aliexpress.pdf/> (accessed 17.08.2018).
- Archibald, J. (2014). The offline cookbook. Retrieved from <https://jakearchibald.com/2014/offline-cookbook/> (accessed 22.08.20218).
- Bias, R. G. & Mayhew, D., J. (2005). *Cost-Justifying Usability. An Update for the Internet Age.* (p.59). San Francisco, USA: Morgan Kaufmann.
- Biørn-Hansen, A., Majchrzak, T. & Grønli, T. (2017). *Progressive Web Apps: The Possible Web-native Unifier for Mobile Development.* In Proceedings of the 13th International Conference on Web Information Systems and Technologies (pp. 344-351), Porto, Portugal: SciTePress.
- Christian, L. M., Dillman, D., A., Smyth, J., D. (2007). Helping Respondents Get It Right the First Time: The Influence of Words, Symbols, and Graphics in Web Surveys. *Public Opinion Quarterly*, 71(1), 113–125.
- Dandekar, K., Raju, B. I., Srinivasan, M. A. (2003). 3-D Finite-Element Models of Human and Monkey Fingertips to Investigate the Mechanics of Tactile Sense. *Journal of Biomechanical Engineering*, 125(5), 682-691.
- Flipkart (2017). Flipkart triples time-on-site with Progressive Web App. Retrieved from <https://developers.google.com/web/showcase/2016/flipkart/> (accessed 17.08.2018).

- Gartner (2017a), Gartner Says Demand for 4G Smartphones in Emerging Markets Spurred Growth in Second Quarter of 2017. Retrieved from <https://www.gartner.com/en/newsroom/press-releases/2017-08-22-gartner-says-demand-for-4g-smartphones-in-emerging-markets-spurred-growth-in-second-quarter-of-2017/> (accessed 17.08.2018).
- Gartner (2017b), Hype Cycle for Mobile Applications and Development. Retrieved from <https://www.gartner.com/doc/3762279/hype-cycle-mobile-applications-development/>
- Gartner (2011). The Avoidable Cost of Downtime. Retrieved from <https://luminet.co.uk/wp-content/uploads/2014/10/avoidable-cost-of-downtime-part-1.pdf> (accessed 17.08.2018).
- Gartner (2014). The Cost of Downtime. Retrieved from <https://blogs.gartner.com/andrew-lerner/2014/07/16/the-cost-of-downtime/> (accessed 10.07.2018).
- Gaunt, M. (2018). Service Workers: an Introduction. Retrieved from <https://developers.google.com/web/fundamentals/primers/service-workers/> (accessed 10.05.2018).
- Google (2009). Speed Matters. Retrieved from <https://research.googleblog.com/2009/06/speed-matters.html> (accessed 29.06.2018).
- Google (2016). The need for mobile speed: How mobile latency impacts publisher revenue. Retrieved from <https://www.thinkwithgoogle.com/intl/en-154/insights-inspiration/research-data/need-mobile-speed-how-mobile-latency-impacts-publisher-revenue/>
- Google (2018a). When it comes to mobile, it's time to stop making excuses. <https://www.thinkwithgoogle.com/intl/en-ca/advertising-channels/mobile/mobile-shopping-ecosystem> (accessed 19.06.2018).
- Google (2018b). Progressive Web App Checklist. Retrieved from <https://developers.google.com/web/progressive-web-apps/checklist/> (accessed 29.08.2018)
- Google (2014). Simple is better - Making your web forms easy to use pays off. Retrieved from <https://ai.googleblog.com/2014/07/simple-is-better-making-your-web-forms.html> (accessed 20.08.2018)
- Gould, J.D. & Lewis, C. (1985). Designing for Usability: What Designers Think.

*Communications of the ACM CACM Homepage archive*, 28(3), 300-311.doi: 10.1145/3166.3170

van der Heijden, H., & Valiente, P. (2002). *The Value of Mobility for Business Process Performance: Evidence from Sweden and the Netherlands*. ECIS 2002 Proceedings (pp. 1144-1153), Gdańsk, Poland: Wydawnictwo Uniwersytetu Gdanskiego.

Henninger (2001). *An Organizational Learning Method for Applying Usability Guidelines and Patterns*. IFIP International Conference on Engineering for Human-Computer Interaction, (141-155), Toronto, Canada: Springer.

Hoehle, H., Aljafari, R., & Venkatesh, V. (2015). Leveraging Microsoft's mobile usability guidelines: Conceptualizing and developing scales for mobile application usability. *International Journal of Human-Computer Studies*, 89, 35-53. doi: 10.1016/j.ijhcs.2016.02.001

Hoover, S. (2017). Design for Fingers, Touch, and People, Part 1. Retrieved from <https://www.uxmatters.com/mt/archives/2017/03/design-for-fingers-touch-and-people-part-1.php#comments> (accessed 21.08.2018)

Human Interface Guidelines (n.d.). Retrieved from <https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/> (accessed 20.08.2018).

IBM Software (2012). Native, web or hybrid mobile-app development. Retrieved from [ftp://ftp.software.ibm.com/software/in/events/softwareuniverse/resources/Native\\_web\\_or\\_hybrid\\_mobile-app\\_development.pdf](ftp://ftp.software.ibm.com/software/in/events/softwareuniverse/resources/Native_web_or_hybrid_mobile-app_development.pdf) (accessed 31.08.2018).

Introduction (n.d.). Retrieved from <https://material.io/design/> (accessed 31.08.2018).

Introduction to components (n.d.). Retrieved from <https://angular.io/guide/architecture-components> (accessed 30.08.2018)

Introduction to services and dependency injection (n.d.). Retrieved from <https://angular.io/guide/architecture-services> (accessed 30.08.2018).

Introduction to modules (n.d.). Retrieved from <https://angular.io/guide/architecture-modules> (accessed 30.08.2018)

Kahuna (2016). The Kahuna Mobile Marketing Index. Retrieved from [http://go.kahuna.com/rs/052-HXZ-275/images/Kahuna\\_Marketing\\_Index\\_Q1\\_2016.pdf](http://go.kahuna.com/rs/052-HXZ-275/images/Kahuna_Marketing_Index_Q1_2016.pdf) (accessed 17.08.2018).

- Kochendörfer, B., Liebchen, J.H. & Viering, M.G. (2018). *Bau-Projekt-Management. Grundlagen und Vorgehensweisen*. (pp. 5, 250-251, 260-261). Berlin, Deutschland: Springer Vieweg.
- Krug, S. (2013). *Don't make me think*. (p. 147). Frechen, Deutschland: mitp.
- Leavitt, M. & Shneiderman, B. (2006). *Research-Based Web Design & Usability Guidelines*. Washington, DC: U.S. Government Printing Office.
- Marcus & Baumgartner (2004). Mapping user-interface design components vs. culture dimensions in corporate websites. *Visible Language Journal*, MIT Press, 38 (1), pp. 1–65.
- Maeda, J. (2006). *SIMPLICITY. Die Zehn Gesetze der Einfachheit*. Heidelberg, Deutschland: Elsevier.
- Malavolta, I., S., Soru, T., & Terragni, V. (2015). *Hybrid Mobile Apps in the Google Play Store: An Exploratory Investigation*. MOBILESoft '15 Proceedings of the Second ACM International Conference on Mobile Software Engineering (pp. 56-59), Florence, Italien: IEEE Press.
- Mangold, R. (2007). *Informationspsychologie: Wahrnehmen und Gestalten in der Medienwelt*. (pp. 100-101, 103-105). Wiesbaden, Deutschland: Springer VS.
- Miller, S., Jarrett, C. (2001). Should I use a drop-down? Four steps for choosing form elements on the web. Retrieved from <http://www.formsthatwork.com/files/Articles/dropdown.pdf> (accessed 20.08.2018)
- Mordor Intelligence (2018). Online Project Management Software Market (End use - Small and Medium-Sized Enterprises (SMEs), Large Enterprises, Government) -Global Industry Analysis, Size, Share, Growth, Trends, and Forecast 2018-2026. Retrieved from <https://www.mordorintelligence.com/industry-reports/project-management-software-systems-market/> (accessed 17.08.2018).
- Nielsen, J (1997a). Discount Usability for the Web. Retrieved from <https://www.nngroup.com/articles/web-discount-usability/> (accessed (21.08.2018)
- Nielsen, J. (1997b). How Users Read on the Web. Retrieved from

- <https://www.nngroup.com/articles/how-users-read-on-the-web/> (accessed 21.08.2018)
- Nielsen, J. (2017). Jakob's Law of Internet User Experience. Retrieved from <https://www.nngroup.com/videos/jakobs-law-internet-ux/> (accessed 21.08.2018)
- Nielsen, J. (2008). Usability ROI Declining, But Still Strong. Retrieved from <https://www.nngroup.com/articles/usability-roi-declining-but-still-strong/> (accessed 10.09.2018).
- Nielsen, J. (1995). 10 Usability Heuristics for User Interface Design. Retrieved from <https://www.nngroup.com/articles/ten-usability-heuristics/> (accessed 20.08.2018).
- Nitithamyong, P., Skibniewski, M.J. (2004). Web-based construction project management systems: how to make them successful? *Automation in Construction*, 13(4) , 491 – 506. doi: 10.1016/j.autcon.2004.02.003
- Penzo, M. (2006). Label Placement in Forms. Retrieved from <https://www.uxmatters.com/mt/archives/2006/07/label-placement-in-forms.php/> (accessed 20.08.2018)
- Pinterest (2017). Driving user growth with performance improvements. Retrieved from [https://medium.com/@Pinterest\\_Engineering/driving-user-growth-with-performance-improvements-cfc50dafadd7/](https://medium.com/@Pinterest_Engineering/driving-user-growth-with-performance-improvements-cfc50dafadd7/) (accessed 29.06.2018).
- Progressive Web Apps A new way to deliver amazing user experiences on the web. (n.d.) Retrieved from <https://developers.google.com/web/progressive-web-apps/> (accessed 17.05.2018).
- Project Management Institute (2017). Success Rates Rise Transforming the high cost of low performance. Retrieved from <https://www.pmi.org/-/media/pmi/documents/public/pdf/learning/thought-leadership/pulse/pulse-of-the-profession-2017.pdf> (accessed 17.08.2018).
- Reed, P., Holdaway, K., Isensee, S., Buie, I., Fox, J., Williams, J., & Lund, A. (1999). User interface guidelines and standards: progress, issues, and prospects. *Interacting with Computers*, 12(2), 119–142. doi: 10.1016/S0953-5438(99)00008-9
- Russel, A. (2016). What, Exactly, Makes Something A Progressive Web App? Retrieved from <https://infrequently.org/2016/09/what-exactly-makes-something-a-progressive-web-app/> (accessed 17.08.2018).

- Schneider, W., Volkmann, W. (2013). *Prozessorientiertes Bauprojektmanagement. Kurzanleitung Heft 1* (pp. 1-3). Berlin: Springer Vieweg.
- Seckler, M., Tuch, A. N., Opwis, K., Bargas-Avila, J. A. (2012). User-friendly locations of error messages in web forms: Put them on the right side of the erroneous input field. *Interacting with Computers, Volume 24*, 107–118. doi: 10.1016/j.intcom.2012.03.002
- Seckler, M., Heinz, S., Javier, A., Bargas-Avila, B., Opwis, K., Tuch, A. N. (2014). *Designing Usable Web Forms – Empirical Evaluation of Web Form Improvement Guidelines*. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (pp. 1275-1284). doi: 10.1145/2556288.2557265
- Sherwin, K. (2014). Placeholders in Form Fields Are Harmful Retrieved from <https://www.nngroup.com/articles/form-design-placeholders/> (accessed 20.08.2018).
- Splitmetrics (2018). What's a Good App Store Page Conversion Rate? We Asked 10M Users. Retrieved from <https://splitmetrics.com/blog/whats-a-good-app-store-page-conversion-rate/> (accessed 17.08.2018).
- Thesmann, S. (2016). *Interface Design. Usability, User Experience und Accessibility im Web gestalten*. (pp. 225-235). Springer Vieweg.
- Whitenton, K. (2016a). Website Forms Usability: Top 10 Recommendations. Retrieved from <https://www.nngroup.com/articles/web-form-design/> (accessed 20.08.2018)
- Whitenton, K. (2016b). Centered Logos Hurt Website Navigation. Retrieved from <https://www.nngroup.com/articles/centered-logos/> (accessed 21.08.2018)
- Yong, L., Chunlai, C., & Caiyun (2012). Study on a Web-based Project Integrated Management Information System in A/E/C Industry. *Journal of Software, ISSN 1796-217X* Volume 7, 1713-1720. Retrieved from <https://pdfs.semanticscholar.org/8253/483e14b9cf26aad6a6f0f0b85f4757ebdd8.pdf>

# Anhang A Die PWA starten

Im folgenden werden kurz die beiden Möglichkeiten erklärt, die es gibt, um die Anwendung zu nutzen. Die OpenProject Instanz mit der die PWA interagiert, ist unter dem Link **<https://pwa.openproject.com/>** zu finden. Die Admin Anmeldedaten sind folgende:

Benutzernamen: Linofischer@outlook.com

Passwort: adminadminPWA

Prinzipiell gibt es zwei Möglichkeiten die Anwendung zu testen und zu benutzen.

## 1. Gehostete Variante Nutzen

Der aktuelle Stand der PWA ist auf der Domain **<https://neu.w11k.de/>** bis auf weiteres gehostet. Hier kann die PWA genutzt, getestet und heruntergeladen werden.

## 2. Das Angular Projekt lokal starten

Der Quellcode der Anwendung befindet sich auf dem beigelegten USB-Stick. Die Anwendung kann in einer beliebigen sdk geöffnet werden. Um die Anwendung lokal zu starten muss *node.js* auf dem Rechner installiert sein. Um CORS-Konflikte zu vermeiden muss die Anwendung über einen proxy gestartet werden. Hierzu genügt der Angular-Befehl:

***ng serve -o --proxy-config proxy.config.json***

Der Befehl startet die Anwendung und öffnet sie in dem Standardbrowser. Gibt es hier irgendwelche Dependency Probleme, sollten diese durch den node-Befehl:

***npm install***

behoben werden.

Sollte unerwarteterweise bei eine der beiden Methoden einen 401 unauthorized Fehler bei Nutzung der PWA auftreten, kann dieser durch Löschen des LocalStorage und neuladen der Seite behoben werden.

# Anhang B Erklärung

## Erklärung

Hiermit versichere ich an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne die Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

Stuttgart, den 01. Oktober 2018