St. Pölten University of Applied Sciences

/ medien & digitale technologien

/fh/// st. pölten

Prototypische Umsetzung und Evaluation einer interaktiven Plattform für Menschen mit arthrotischen Beschwerden

Diplomarbeit

Ausgeführt zum Zweck der Erlangung des akademischen Grades
Dipl.-Ing. für technisch-wissenschaftliche Berufe

am Masterstudiengang Digitale Medientechnologien an der Fachhochschule St. Pölten, **Masterklasse Grafikdesign**

von:
Daniel Winter, BSc
dm151540

Betreuer/in und Erstbegutachter/in: Dipl.-Ing. Gernot Rottermanner, BSc
Zweitbegutachter/in: FH-Prof. Dipl.-Ing. Dr. Peter Judmaier

Wien, 18.10.2017

Ehrenwörtliche Erklärung

Ich versichere, dass

- ich diese Arbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfe bedient habe.

- ich dieses Thema bisher weder im Inland noch im Ausland einem Begutachter/einer Begutachterin zur Beurteilung oder in irgendeiner Form als Prüfungsarbeit vorgelegt habe.

Diese Arbeit stimmt mit der vom Begutachter bzw. der Begutachterin beurteilten Arbeit überein.

....

Ort, Datum Unterschrift

Kurzfassung

Die Anzahl der Menschen mit arthrotischen Beschwerden hat in den letzten Jahren deutlich zugenommen und wird durch die zunehmende demografische Alterung der Bevölkerung auch künftig steigen. Um den Folgen und Auswirkungen der Erkrankung entgegenzuwirken, sind im Laufe der Zeit diverse Webseiten und Apps zum Thema Arthrose erstellt worden, welche umfangreiche Informationen über Behandlung, Strategien und Techniken bereitstellen.

Zahlreiche Beispiele aus der Forschung in diesem Themenbereich zeigen jedoch, dass die angebotenen Anwendungen selten in Zusammenarbeit mit der betroffenen Zielgruppe erarbeitet werden. Deren Bedürfnisse und Anliegen finden bei der Erstellung eine geringfügige Berücksichtigung, was zu einer verminderten Nachfrage bei der Zielgruppe führt.

Die vorliegende Arbeit untersucht zunächst die Problematiken derzeitiger Arthrose Plattformen für webbasierte mobile Anwendungen wie Apps und Webseiten. Basierend auf den Untersuchungen und den erhobenen Anforderungen der Zielgruppe aus Fokusgruppen wurde ein interaktiver Prototyp entwickelt und anschließend mit UserInnen getestet. Aufgrund der gewonnenen Erkenntnisse der Umsetzung der Plattform für Menschen mit arthrotischen Beschwerden, werden Empfehlungen abgegeben, welche EntwicklerInnen bei der erfolgreichen Realisierung eines Prototyps im Gesundheitsbereich als Hilfestellung dienen sollen. Es wird daher u.a. empfohlen, die potentiellen körperlichen Einschränkungen der Zielgruppe zu bedenken und bestehende Richtlinien der Barrierefreiheit im Web, bei der Entwicklung eines Prototyps zu berücksichtigen.

Abstract

The number of people with Osteoarthritis (OA) has grown substantially in the last few years and will continue to rise as a result of the demographic aging of the population. In order to counteract the consequences and effects of this disease, various websites and apps have been developed about this issue, providing extensive information about treatment, strategies and techniques.

Numerous research examples on this topic show that the offered applications are rarely developed in collaboration with the affected target group. Their needs and concerns are hardly taken into account during the development process, resulting in a low demand of the target group.

This thesis firstly examines the problems of current OA platforms for web-based mobile applications such as apps and websites. On the basis of the investigations and the gathered requirements of the target group from focus groups an interactive prototype was developed and subsequently tested with users. Based on the findings of the implementation of the platform for people with OA, recommendations were made, helping web developers in the successful realization of a prototype in the healthcare sector. Therefore, among other things, it is recommended, to consider potential physical limitations of the target group and existing web accessibility guidelines, in the development of a prototype.

Inhaltsverzeichnis

1 Einleitung

„Although there is extensive information on treatments and facts about the disease, there is a lack of information about strategies and techniques to shift ones viewpoint about the impact of OA.“ (Umapathy et al., 2015)

1.1 Problemstellung

Eine im Jahr 2014 im Auftrag des Bundesministeriums für Gesundheit durchgeführte österreichische Gesundheitsbefragung der Statistik Austria hat ergeben, dass die Anzahl der Menschen mit arthrotischen Beschwerden in den letzten Jahren deutlich angestiegen ist und dies in den kommenden Jahren auch weiterhin anhalten wird. Laut dieser Statistik sind bei den 60- bis 74-jährigen Frauen bereits jede dritte und bei den gleichaltrigen Männern beinahe jeder fünfte betroffen. Eine noch höhere Verbreitung herrscht bei Personen ab 75 Jahren, hier sind 44% der Frauen und 22% der Männer betroffen (Klimont & Baldaszti, 2015).

Die zunehmende Zahl an älteren Frauen und Männern in der Bevölkerung bedeutet, dass es auch potenziell mehr Menschen mit chronischen Erkrankungen wie z.B. Arthrose geben wird. Bei Arthrose (engl. Osteoarthritis / OA) handelt es sich um eine degenerative Gelenkserkrankung, welche durch wiederkehrende Fehl- und/oder Überlastung zu Knorpelabbau in den Gelenken führt. Arthrose hat für Betroffene nicht nur physische Einschränkungen zur Folge, sondern oftmals auch psychische und soziale Auswirkungen (Pearson, Walsh, Carter, Koskela, & Hurley, 2016).

Die zunehmende Anzahl an webbasierten mobilen Anwendungen, wie Apps oder Webseiten zum Thema Arthrose, macht es für Menschen mit arthrotischen Beschwerden zunehmend schwieriger, die passenden Informationen herauszufiltern. Jede Anwendung bietet eine Vielfalt an Funktionen sowie unterschiedliche Möglichkeiten der Nutzung an. Auch das Alter und die Mediennutzung spielen hierbei eine wichtige Rolle. Die Zielgruppe stellen „Digital Immigrants“ dar, welche mit dem Gebrauch und der Nutzung von Onlineanwendungen nicht so sehr vertraut sind, wie jüngere Generationen, bei denen die Mediennutzung im Alltag nicht mehr wegzudenken ist.

Laut Pearson et al. (2016) nutzen ältere Menschen das Internet vermehrt als Quelle, um sich über Themen rund um Gesundheitsthemen zu informieren, haben aber Bedenken, was die Qualität der Informationen und den sicheren Umgang mit persönlichen Daten betrifft. Ein weiteres Problem von vielen Arthrose Plattformen ist, dass diese selten mit Feedback der EndnutzerInnen entwickelt werden. Die Bedürfnisse der Zielgruppe werden bei der Entwicklung wenig bis kaum berücksichtigt.

Aus diesen Gründen hat die vorliegende Diplomarbeit das Ziel, die genannten Probleme und Herausforderungen näher zu behandeln und aufzuarbeiten.

Diese Arbeit wurde vom Ludwig-Boltzmann-Cluster „Arthritis und Rehabilitation“ (http://crbr.lbg.ac.at) gefördert. In Zusammenarbeit mit der Diplomarbeit von Shadja El Aeraky (2017), im Rahmen derer der User Centered Design-Prozess der interaktiven Arthrose-Plattform behandelt wird, erfolgte die prototypische Umsetzung einer interaktiven Plattform für Personen mit arthrotischen Beschwerden.

1.2 Forschungsfragen

Folgende Forschungsfragen ergeben sich aus der Problemstellung, die in der Arbeit untersucht und beantwortet werden sollen:

- Welche technischen Anforderungen ergeben sich aus den Anforderungen der Zielgruppe hinsichtlich einer interaktiven Kommunikationsplattform?
- Welche Technologien und Frameworks eignen sich am Besten für die prototypische Umsetzung einer interaktiven Plattform (basierend auf den Anforderungen der Zielgruppe) und gewährleisten, bei Bedarf, eine einfache Erweiterung?
- Welche gültigen Accessibility Guidelines sind für die Zielgruppe relevant und wie lassen sich diese am Besten umsetzen?

1.3 Struktur der Arbeit

Grundsätzlich kann diese Arbeit in vier Hauptbereiche gegliedert werden. Im ersten Bereich (Kapitel 2) werden zunächst die aktuellen Anforderungen, Bedürfnisse sowie Probleme derzeitiger Arthrose Anwendungen aufgezeigt. Zudem werden vier Plattformen genauer untersucht und gegenübergestellt. Diese wurden auch der Zielgruppe bei der ersten Befragung (El Aeraky, 2017, S. 56ff) vorgestellt und anschließend evaluiert.

Kapitel 3 umfasst die theoretischen und technologischen Grundlagen, welche die Basis für den empirischen Teil schaffen sollen. Hier werden vor allem Termini und Technologien aus der Welt der webbasierten mobilen Anwendungen erläutert, die im Zuge der Umsetzung des Prototyps verwendet wurden. Gefolgt von Kapitel 4, in welchem die für die Entwicklung der Arthrose Plattform möglichen Technologien anhand von vorab festgelegten Auswahlkriterien gegenübergestellt und bewertet wurden. Dieser Vergleich soll zeigen, welche Technologie(n) bei der Entwicklung der Plattform am Besten heranzuziehen sind.

In Kapitel 5 folgt eine detaillierte Beschreibung der prototypischen Umsetzung der interaktiven Arthrose Plattform, die in Zusammenarbeit mit Shadja El Aeraky konzipiert wurde. Neben der Darstellung allgemeiner technischer Vorkehrungen, werden auch Teile der Funktionalitäten erläutert, welche durch die erhobenenen Bedürfnisse der Zielgruppe entstanden. Abgeschlossen wird dieser Abschnitt mit den Herausforderungen und Problemen, die während der Entwicklung des Prototyps entstanden.

Das Kapitel 6 beinhaltet ein Fazit und beantwortet die in Kapitel 1.2 aufgestellten Forschungsfragen. Zudem werden die gewonnenen Erkenntnisse als Lessons Learned zusammengefasst.

Im abschließenden Kapitel 7 werden Ansätze und Ideen gelistet, wie sich die Plattform zukünftig weiterentwickeln könnte.

1.4 Methodik und Motivation

Um die in Kapitel 1.2 aufgestellten Forschungsfragen zu beantworten wurden folgende wissenschaftliche Methoden verwendet:

Erforschung des „State of the Art“

Der erste Schritt der Arbeit stellte die ausführliche Recherche der Probleme derzeitiger Arthrose Plattformen dar. Hinterfragt wurde, welche Plattformen es bisher im Bereich der Arthrose gibt und welche unterstützenden Funktionen und Features diese den Betroffenen anbieten. Die Literaturrecherche soll einen Überblick möglicher Ansätze bieten, wo es im Bereich der Arthrose Plattformen noch Verbesserungspotenzial gibt.

Proof of Concept

Aufgrund der Ergebnisse zweier Fokusgruppen zur Erhebung der Bedürfnisse der Zielgruppe und zur Evaluierung eines Konzepts (El Aeraky, 2017, S. 71ff), ist der finale Schritt die prototypische Entwicklung einer interaktiven Arthrose Plattform. Zu diesem Zweck wurden vorab Kriterien festgelegt, die dabei helfen sollen, Technologien auszuwählen, die bei der Entwicklung des Prototyps nützlich sein

sollen. Das Ziel des Prototyps soll sein, diesen in einem abschließenden Nutzungstest von der Zielgruppe evaluieren zu lassen und dessen zukünftiges Potenzial zu erfragen.

Motivation

Diese Diplomarbeit soll vor allem WebentwicklerInnen eine Orientierung bieten, welche vor dem Problem der Entwicklung eines interaktiven Prototyps stehen. Mehrere hilfreiche Kriterien zur Technologiebeurteilung und -findung sowie theoretische Grundlagen sind in dieser Arbeit gelistet. Diese können nicht nur bei der Evaluierung für Webanwendungen im Gesundheitsbereich als Hilfestellung dienen, sondern auch über diesen Bereich hinaus eingesetzt werden.

Ziel dieser Arbeit ist, die Bedürfnisse und Anforderungen der Zielpersonen an eine mögliche Arthrose Plattform herauszufinden, sodass die entwickelte Plattform als Informations- und Wissensquelle regelmäßig genutzt wird.

2 Arthrose Plattformen

Dieses Kapitel umfasst den derzeitigen Stand an Arthrose Plattformen, welche sowohl am Desktop als auch als App verwendet werden können. In Kapitel 2.1 werden die akutellen Anforderungen, Bedürfnisse und Probleme derzeitiger Arthrose Anwendungen aufgezeigt. Darauf folgt in Kapitel 2.2 ein Vergleich vorausgewählter Plattformen, welche der Zielgruppe zur Evaluierung vorgestellt wurden (El Aeraky, 2017, S. 56ff).

2.1 Herausforderungen aktueller Plattformen

Das Internet ist als Informationsquelle bei Themen rund um die Gesundheit nicht mehr wegzudenken, daher gibt es auch im Bereich Arthrose unzählige Anwendungen, die der Zielgruppe helfen sollen, sich über ihre Krankheit zu informieren. Das spiegelt sich auch im App Bereich wieder. Nach Spielen und Hilfsprogrammen (wie z.B. Navigation oder Übersetzung), sind mobile Gesundheits-Apps (mHealth), die App Kategorie mit dem drittgrößten Wachstum, wodurch sich auch das Gesundheitswesen-Modell nachhaltig ändert. (Morera, Díez, Garcia-Zapirain, López-Coronado, & Arambarri, 2016, S. 1). Trotz der steigenden Anzahl an Gesundheits-Apps und damit auch dem steigenden Angebot an Arthrose Plattformen, enthalten diese oftmals Probleme, die nicht außer Acht gelassen werden dürfen.

Eines der Hauptprobleme von vielen Arthrose Plattformen ist, dass diese Programme selten mit Feedback der EndnutzerInnen entwickelt werden. Die Bedürfnisse der Zielgruppe werden bei der Entwicklung wenig bis kaum berücksichtigt, was dazu führt, dass das Endprodukt oftmals als unerwünscht, unbrauchbar oder zu komplex betrachtet wird (Pearson et al., 2016).

Außerdem fehlt es den Anwendungen an Überzeugungstechniken, um die Zielgruppe ausreichend bei der „digitalen“ Verwaltung ihrer Krankheit zu engagieren. Da häufig keine regelmäßige Durchführung von Übungen und anderen Bewegungsprogrammen stattfindet, müssen Ideen gefunden werden, wie man die Zielgruppe dazu animieren kann, laufend etwas für ihre Gesundheit zu tun. Zusätzlich würden Plattformen davon profitieren, wenn man soziale Unterstützung einbindet, beispielsweise in Form von Social-Media-Kanälen oder Foren. Auch ein Belohnungs- oder Lobsystem, bei dem man z.B. für regelmäßig durchgeführte Übungen Punkte sammeln kann, würde bei der Zielgruppe die Motivation zur häufigeren Nutzung einer Anwendung steigern (Geuens et al., 2016).

Ältere Menschen nutzen das Internet als Quelle, um sich über ihre Gesundheit zu informieren. Jedoch zeigen sich bei dieser Gruppe Bedenken hinsichtlich der Nutzung und vor allem der Qualität der Informationen. Daher ist es essenziell, dass man BenutzerInnen eine glaubwürde Plattform anbietet, welche Unterstützung und Feedback bereitstellt (Pearson et al., 2016).

Außerdem wäre die Zielgruppe daran interessiert, zugeschnittene und personalisierte Informationen und Ratschläge zu bekommen. Dies kann in den meisten Fällen jedoch nur über eine Registrierung und anschließendes Einloggen erfolgen, was allerdings häufig als Hindernis wahrgenommen wird. Zum einen hat die Zielgruppe Bedenken, sich an den Usernamen und das Passwort zu erinneren, mehr jedoch, dass die Personalisierung Zeit beansprucht und eventuell technische Anforderungen bereithält und somit eine zusätzliche Aufwandsbelastung entsteht (Pearson et al., 2016).

Obendrein ist es wichtig, dass die Plattform leicht und intuitiv zu bedienen ist, damit die benötigten Informationen möglichst schnell und einfach gefunden werden können. Informationen sollten so aufbereitet sein, damit diese von der Zielgruppe leicht verstanden werden und überdies frei von Fachsprache und Akronymen sind (Pearson et al., 2016).

Eine der schwierigsten Herausforderungen bleibt jedoch, eine Veränderung der Haltung von PatientInnen gegenüber der Krankheit herbeizuführen. Obwohl es zahlreiche und umfassende Informationen über Behandlungen und Therapien zu arthrotischen Beschwerden gibt, fehlt es an Strategien und Techniken, um bei der Zielgruppe eine Wahrnehmungsänderung bezüglich der Auswirkungen und Ursachen der Erkrankung und somit eine Lebensstilanpassung herbeizuführen (Umapathy et al., 2015).

Fazit

Zusammenfassend kann gesagt werden, dass die Zielgruppe durchaus gewillt und interessiert daran ist, mobile Gesundheit-Apps zum Selbstmanagement ihrer Krankheit einzusetzen, es aber noch einige Bedenken hinsichtlich der regelmäßigen Nutzung und der Integration in den Alltag gibt.

2.2 Plattformvergleich

Für die Erstevaluation relevanter Plattformen wurde eine Recherche bestehender Arthrose Plattformen durchgeführt, um diese von den Testpersonen bei der ersten Fokusgruppe und den Interviews Anfang Mai 2017 bewerten zu lassen (El Aeraky, 2017, S. 56ff). Dabei war es wichtig, dass diese Plattformen ein möglichst breites Spektrum an unterschiedlichen Features und Funktionen beinhalten, um diese der Zielgruppe vorzustellen und Feedback darüber einzuholen.

Es wurden sowohl Webseiten und Apps im deutschsprachigen Raum, als auch Seiten auf Englisch gesichtet. Aufgefallen ist, dass es bei einem Großteil der englischsprachigen Plattformen die Möglichkeit der Personalisierung gegeben hat. UserInnen konnten sich hier ein Konto erstellen und nach teilweiser ausführlicher Anamese wurde, basierend auf den erhobenen Daten, ein Profil mit dazupassenden Übungen und/oder Trainingsplänen erstellt. Die nachfolgenden vier Plattformen bieten ein breites Spektrum an Funktionen, die den Testpersonen anhand eines Handouts zur Bewertung vorgelegt wurden.

Um die jeweils genutzten Technologien bzw. Frameworks der Webseiten herauszufinden, wurde die Browser Erweiterung „Wappalyzer“ [1] genutzt. Dieses Plug-In (Hilfsprogramm) identifiziert sowohl Content Management Systeme (CMS), Webtechnologien, Server Software und Analysetools. Zudem wurde eine ausführliche Quellcode Analyse gemacht, wenn das Hilfsprogramm keine Technologien identifizieren konnte.

2.2.1 Arthrose behandeln

Abbildung 1. Screenshot der Webseite http://arthrose.behandeln.at, Stand: April 2017

Die erste getestete Plattform Arthrose behandlen (Abbildung 1) bietet einen allgemeinen Überblick über die vorkommenden Arten von Arthrose, deren

[1] Informationen und Download unter https://wappalyzer.com/

Ursachen, Symtome und Behandlungsmöglichkeiten. Zu jedem betroffenen Gelenk gibt es außerdem die drei wirksamsten Übungen als leicht verständlich aufbereitete Bilder zu sehen. Zusätzlich findert man hier auch Informationen zur Ernährung, Selbstpflege sowie Tipps für die täglichen Aktivitäten im Haushalt oder in der Freizeit.

Bei der Analyse der Webseite ist aufgefallen, dass diese sich nicht an die unterschiedlichen Auflösungen der Endgeräte anpasst, jedoch eine mobile Version für Smartphones angeboten wird. Auch wurde, bis auf die JavaScript Bibliothek jQuery, komplett auf unterstützende Webtechnologien bei der Umsetzung verzichtet. Der Aufbau der Seite wurde mit HTML (Hypertext Markup Language), CSS (Cascading Style Sheets) und JavaScript realisiert, was die Wartbarkeit der Seite erhöhen dürfte. Allgemein kann gesagt werden, dass arthrose.behandeln.at eine Informationsplattform ist, über die man sich schnell und einfach über die verschiedenen Arten von Arthrose erkundigen kann.

2.2.2 Deutsches Arthrose Forum

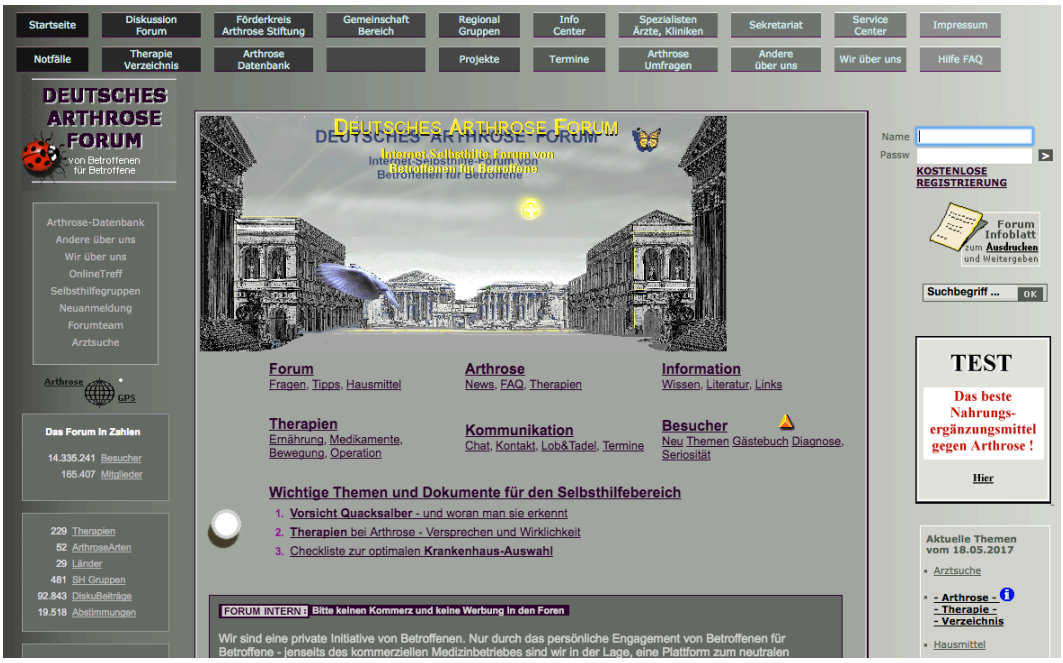

Abbildung 2. Screenshot der Webseite https://www.deutsches-arthrose-forum.de, Stand: April 2017

Das Deutsche Arthrose Forum (Abbildung 2) ist ein umfangreiches Informationscenter mit Foren, Chats, Therapien und Selbsthilfegruppen. Bei dieser Plattform steht der gegenseitige Erfahrungs- und Wissensaustausch im Fokus. Zudem bietet die Plattform eine umfassende Datenbank an, die Informationen zu Arthroseformen sowie medizinische Informationsquellen zu Ärzten, Kliniken und Physiotherapeuten bietet. Viele dieser Informationen verweisen auf externe

Datenquellen. Weiterhin besteht für NutzerInnen die Möglichkeit, sich zu registrieren um somit auf zusätzliche Funktionen zugreifen zu können.

Diese Plattform ist nicht responsive und kann nur auf größeren Bildschirmen komplett betrachtet werden. Das Analyse Plug-In hat keine unterstützenden Webtechnologien gefunden. Bei der Recherche des Quellcodes wurde ersichtlich, dass diese Plattform mit einem zentralen `<iframe>` arbeitet, über welchen der Inhalt je nach Aufruf des Links ausgetauscht wird. Die Umsetzung erfolgt mit einfachen HTML, CSS und JavaScript. Diese Plattform kann nicht als „moderne Anwendung“ bezeichnet werden, da hier veraltete Technologien (z.B. iframe oder Tabellen für Layout) sowie überholte Stilelemente verwendet werden und keine neuen Frameworks zur Unterstützung genutzt werden.

2.2.3 MyJointPain

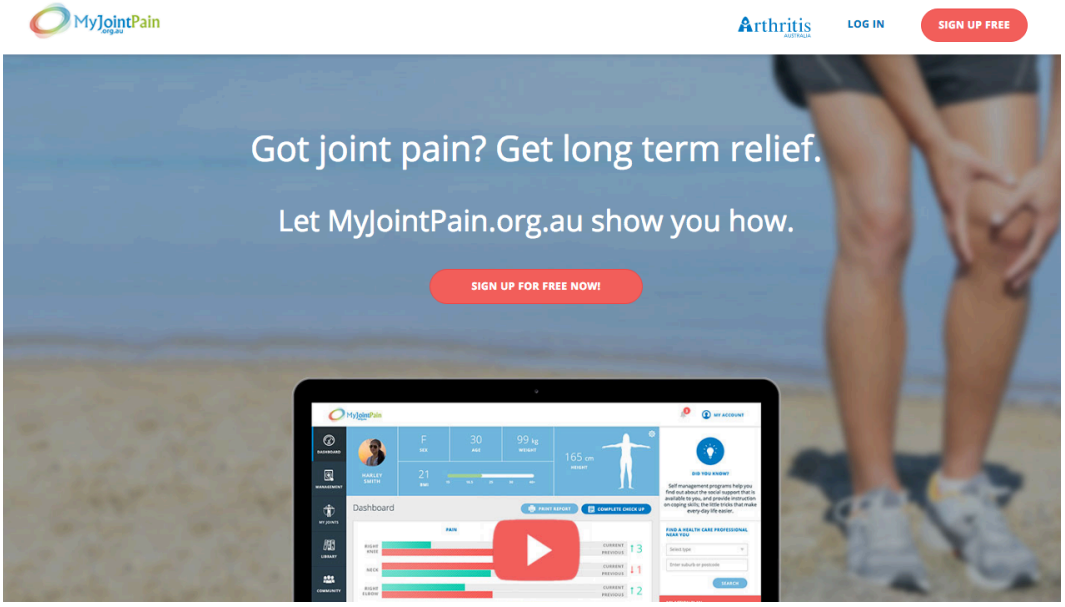

Abbildung 3. Screenshot der Webseite https://www.myjointpain.org.au, Stand: April 2017

Die australische Webseite MyJointPain (Abbildung 3) kann in einen öffentlichen und einen privaten Bereich eingeteilt werden. Im öffentlich zugänglichen Teil der Seite findet man ein profundes Wissen über die Krankheit, Behandlungen, Videos mit Übungen und Informationsblätter zum Download oder Ausdrucken. Bevor man in den geschützten Bereich kommt, werden zuerst über ein ausführliches Screening Daten über den/die NutzerIn gesammelt. Hier wird sehr präzise der aktuelle Gesundheitszustand sowie die betroffenen Gelenke erfragt und anschließend ausgewertet. Der/die BenutzerIn hat im persönlichen Login-Bereich Zugriff auf die Profilverwaltung mit diversen zusätzlichen Funktionen. Über Check-Ups (Untersuchungen) kann in regelmäßigen Abständen der Gesundheitszustand der betroffenen Gelenke neu erhoben und mit der letzten Erhebung verglichen

werden. Ein weiteres besonderes Merkmal ist das Selbstmanagement von Trainings- und Übungsplänen. Hier kann die/der BenutzerIn sich nach eigenen Bedürfnissen und Wünschen einen Aktionsplan für die nächsten Wochen und Monate zusammenstellen und diesen je nach Bedarf ändern. Zusätzlich gibt es eine Rubrik mit dem Titel „Wussten Sie schon“, in welcher täglich wechselnde Tipps rund um das Thema Arthrose veröffentlicht werden, damit UserInnen animiert werden die Webseite stetig aufzusuchen.

MyJointPain kann sowohl auf mobilen Endgeräten als auch auf Desktop Computer verwendet werden. Für die technische Realisierung werden diverse aktuelle Webtechnologien verwendet. Für das Backend das PHP Framework Laravel sowie im Frontend u.a. die JavaScript Frameworks AngularJS, jQuery und als CSS Framework Bootstrap.

Anmerkung: Für die Erstevaluation wurden alle vorgestellten Funktionen und Features für die Testpersonen ins Deutsche übersetzt, sodass eine eventuelle Beeinflussung aufgrund möglicher Sprachbarrieren der Testpersonen auszuschließen ist.

2.2.4 Arthrose Tagebuch

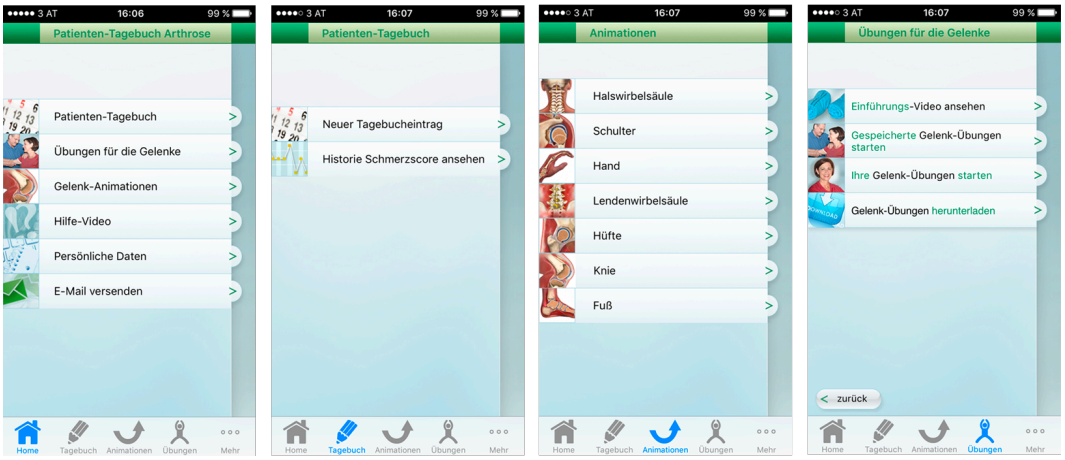

Abbildung 4. Screenshots der Applikation „Arthrose Tagebuch“, Stand: April 2017

Die letzte getestete Plattform (Abbildung 4) ist eine Applikation die nur auf mobilen Endgeräten verwendet werden kann. Nach erfolgreichem Download der App aus dem jeweiligen Store muss kein eigener Account erstellt werden, um die App zu nutzen. Es besteht die Möglichkeit, über einen Fragenbogen persönliche Daten über die/den BenutzerIn zu erheben, diese werden aber nicht weiterverarbeitet, sondern können je nach Bedarf an den behandelnden Arzt geleitet werden. Im Vordergrund dieser App steht das PatientenInnen Tagebuch. Dieses erlaubt UserInnen durch tägliche Tagebucheinträge den eigenen Schmerzverlauf zu kontrollieren und dadurch Rückschlüsse auf den Gesundheitszustand zu ziehen. Ein weiteres Merkmal sind die angebotenen Übungen und Animatioen für Gelenke.

Je nach Bedarf können sich UserInnen Videos für betroffenene Gelenke auf das mobile Endgerät laden. Diese Videos haben den Vorteil, dass man sie jederzeit abrufen kann und die Übungen teilweise sogar unabhängig vom jeweiligen Standort ausüben kann.

Die Anwendung ist sowohl im Apple's App Store als auch im Google's Play Store erhältlich. Es konnten aber keine Rückschlüsse gezogen werden, welche eingesetzten Technologien oder Services die App nutzt oder aus welchen Komponenten sich diese zusammensetzt.

2.2.5 Gegenüberstellung und Zusammenfassung

Abschließend folgt ein tabellarischer Vergleich der ersten drei vorgestellten Plattformen anhand von vorab festgelegten Kriterien. Da es sich beim Arthrose Tagebuch um eine App handelt, wurde diese nicht in die Gegenüberstellung mit aufgenommen. Sollte eine genaue Bestimmung eines Kriteriums nicht möglich gewesen sein, wurde an dieser Position ein Minuszeichen eingesetzt. Ob eine zusätzliche mobile Version der Plattform abrufbar ist, wurde anhand eines iPhone5 und dem Safari Browser getestet. Bei der Auflistung der Technologien und Frameworks werden nur jene aufgelistet, die über die Standardsprachen HTML, CSS und JavaScript (JS) hinaus eingesetzt wurden (siehe Tabelle 1).

Tabelle 1. Vergleich der vorgestellten Arthrose Plattformen

	Arthrose behandeln	***Deutsches Arthrose Forum***	***MyJointPain***
Sprache	*Deutsch*	*Deutsch*	*Englisch*
Responsive	*Nein*	*Nein*	*Ja*
Extra Version für mobile Endgeräte	*Ja*	*Nein*	*Nein (da responsive)*
Anpassung möglich	*Nein*	*Nein*	*Ja*
Technologien / Frameworks	***JS**: jQuery*	*-*	***JS**: jQuery, Angular, und MomentJS;* ***PHP**: Laravel;* ***CSS**: Bootstrap*
Funktionen / Features*	*- Ernährungstipps* *- Übungen* *- Aktionspläne*	*- Themenchats* *- Foren* *- Selbsthilfegruppen*	*- Expertensuche* *- Regelmäßige Check-Ups* *- Tipp des Tages*
Social Media Share oder Follow Möglichkeit	*Ja (Facebook, Google Plus, Twitter)*	*Nein*	*Ja (Facebook, Twitter, YouTube)*

* Alle drei Plattformen bieten eine umfangreiche Wissensbibliothek an, um sich über die verschiedenen Arthrosearten zu informieren, daher wurde hier nur eine kleine Auswahl an Features aufgelistet, welche die jeweiligen Plattformen zusätzlich bereitstellen.

Zusammenfassung der Vor- und Nachteile

Die erste vorgestellte Plattform „Arthrose behandeln“ stellt die Informationen zur jeweiligen Arthroseart kompakt und übersichtlich dar. Die Seite ist einfach zu bedienen und man findet relativ schnell die gesuchten Informationen. Nachteilig muss erwähnt werden, dass diese Seite eher wie eine Werbeplattform wirkt und dadurch ein leicht unseriöser Eindruck entsteht, was auch von den Testpersonen bei der Vorstellung der Plattformen angesprochen wurde (El Aeraky, 2017, S. 56ff).

Das „Deutsche Arthrose Forum“ bietet Betroffenen mit Foren und Chats die Möglichkeit, aktiv Fragen zu stellen und Erfahrungen mit anderen zu teilen. Als weiteren Vorteil können die unzähligen Selbsthilfegruppen gesehen werden, zu denen man beitreten kann. Nicht von der Hand zu weisen ist jedoch, dass diese

Plattform nicht „State of the Art“ ist. Die Seite passt sich nicht den Auflösungen verschiedener Endgeräte an und verwendet keine aktuellen Technologien. Zudem erweckt das Layout durch die Grau-in-Grau Farbkombination einen sehr bedrückenden Eindruck und wirkt durch die vielen unterschiedlichen Elemente überladen und unaufgeräumt.

Der Vorteil des Portals „MyJointPain“ liegt in der angebotenen Personalisierung, welche die Betroffenen nach einer ausführlichen Voruntersuchung und Registrierung nutzen können. Zusätzlich stehen Videos mit Übungen und Datenblätter zum Download und Ausdrucken zur Verfügung. Nachteilig ist, dass die Seite eher für australische Arthrose-Betroffene gedacht und optimiert ist und man gute Englischkenntnisse braucht, sonst können die Informationen fehlinterpretiert werden.

Die App „Arthrose Tagebuch“ bietet Betroffenen eine Tagebuchfunktion an, in der die Schmerzen für ein Gelenk eingetragen und verglichen werden können. Zudem weist die App eine gute Auswahl an Videos für Gelenksübungen auf. Als Nachteil kann gesehen werden, dass die App nur auf mobilen Endgeräten zur Verfügung steht und keine zusätzlichen allgemeinen Informationen abgerufen werden können.

3 Theoretische Grundlagen

Dieses Kapitel stellt die theoretischen und technologischen Grundlagen dar. In Kapitel 3.1 werden zunächst die möglichen Entwicklungsansätze für webbasierte mobile Anwendungen vorgestellt. Im danach Folgenden Kapitel 3.2 geht es um verschiedene Themen der „mobilen Webentwicklung“. Es werden hier nicht nur die bei der prototypischen Entwicklung der Plattform zum Einsatz kommenden möglichen Technologien oder Begrifflichkeiten erläutert, sondern auch alternative Methoden vorgestellt.

3.1 Webbasierte mobile Anwendungen

Mobilen SoftwareentwicklernInnen steht eine Vielzahl an unterschiedlichen Entwicklungsansätzen für mobile Anwendungen, wie Webseiten oder Apps zur Verfügung. Dabei kann es sich entweder um native mobile Apps für eine spezifische Plattform handeln, oder eine mit Webtechnologien erstellte webbasierte mobile App. Die webbasierten mobilen Apps können wiederrum unterteilt werden in Mobile Web Apps, Hybride Mobile Apps und das relativ neue Konzept der Progressive Web Apps (Malavolta, 2016). Aufgrund der bevorzugten Desktopnutzung der Zielgruppe (El Aeraky, 2017, S. 82), wird im folgenden Abschnitt dieser Entwicklungsansatz für die Arthrose Plattform genauer erläutert. Als mögliche alternative Ansätze werden zudem Hybride Mobile Apps und Progressive Web App kurz vorgestellt.

Webbasierte mobile Apps sind Anwendungen, die mit den neuesten Standard Webtechnologien HTML5, CSS3 und JavaScript entwickelt werden. Dieses gemeinsam ausgerichtete Technologiepaket lässt sich in die o.g. drei Arten aufteilen (Malavolta, 2016):

3.1.1 Mobile Web Apps

Mobile Web Apps können mittels eindeutig identifizierbarer URL (Uniform Resource Locator) über den Browser des Betriebssystems aufgerufen und dessen Inhalt bezogen werden (Malavolta, 2016). Durch CSS Media Queries, mit deren Hilfe die Breite des jeweiligen Endgeräts des/der Users/Userin abgefragt werden kann, können mobile Web Apps auf viele unterschiedliche Endgeräte angepasst werden und für Browser auf allen Plattformen gleichermaßen für UserInnen zugänglich gemacht werden (Heitkötter, Hanschke, & Majchrzak, 2012, p. 4).

Diese Art der Distribution bietet EntwicklerInnen die Möglichkeit der schnellen App Entwicklung, leichten Wartbarkeit einer gemeinsamen Quellcodebasis und der vollen Übertragbarkeit einer Anwendung. Obwohl viele Browser die neuesten HTML5 Standards interpretieren können und mithilfe von unterstützenden APIs (Application Programming Interface) auf einige native Funktionen zugreifen können, haben diese nicht uneingeschränkten Zugriff auf alle gerätespezifischen Hardware Features. Weiter ist es nicht möglich, mobile Web Apps via App Stores zu verteilen (Malavolta, 2016). Außerdem unterscheiden sich Web Apps in der Erscheinung und im Verhalten zu nativen Apps. Diese können nicht auf die nativen User Experience Elemente zugreifen und werden daher zumindest teilweise von UserInnen immer als gewöhnliche Webseiten wahrgenommen (Heitkötter et al., 2012, S. 4).

3.1.2 Hybride Mobile Apps

Um dem Mangel an Zugriff auf gerätespezifische Hardware Features von mobilen Web Apps entgegenzuwirken, bei der Entwicklung aber trotzdem Webtechnologien wie HTML, CSS und JavaScript zu verwenden, sind hybride Ansätze in den letzten Jahren entwickelt worden (Heitkötter et al., 2012, S. 4). Auf Webtechnologien basierte hybride Apps vereinen die besten Komponenten von nativen und Web Apps und können auf allen mobilen Plattformen verteilt werden. Zur Erstellung hybrider Apps benötigen EntwicklerInnen ein hybrides Entwicklungsframework, welches einen „native wrapper“ bereitstellt, in dem der Web Source Code enthalten ist und eine JavaScript API, welche sich um alle Service Anfragen des webbasierten Codes an die jeweiligen nativen plattformabhängigen API kümmern (Malavolta, 2016).

Anhand dieses „native wrappers“ kann der geschriebene Source Code nicht nur auf allen mobilen Plattformen verteilt werden, sondern auch für mobile Web Apps, die im Browser laufen, wiederverwendet werden. Die Vorteile sind daher, dass nur eine einzige Source Code Basis für alle Plattformen gepflegt und adaptiert werden muss und zusätzlich bleibt die User Experience über mehrere Plattformen hinweg gleich. Auf der anderen Seite jedoch kann die verwendete JavaScript API nur auf einen Teil der zur Verfügung gestellten plattformabhängigen APIs zugreifen. Außerdem entsteht durch jeden JavaScript API Aufruf ein zusätzlicher Performance Overhead, der sich nachteilig auf die User Experience auswirken kann (Malavolta, 2016).

3.1.3 Progressive Web Apps (PWA)

EntwicklerInnen von webbasierten mobilen Apps müssen im Gegensatz zu nativen Apps Einschränkungen in der Entwicklung in Kauf nehmen (Steyer, 2017):

- Längere Ladezeit webbasierter mobiler Anwendungen
- Eingeschränkter Zugriff auf Plattform Services wie z.B. Benachrichtigungen
- Keine Datensynchronisation im Hintergrund oder
- Teilweise fehlende Offlinefähigkeit

Um diesen und anderen Nachteilen entgegenzuwirken und die Vorteile von Browseranwendungen zu nutzen, wurden PWA entwickelt. Diese stehen für moderne Webanwendungen, welche in gängigen Browsern laufen und in neuesten Versionen zusätzliche Features anbieten (Steyer, 2017). Damit eine mobile Web App als PWA betrachtet werden kann, muss diese folgende drei Bedingungen enthalten (Malavolta, 2016):

- Ist über HTTPS erreichbar
- Enthält ein Web App Manifest in dem die Metadaten deklariert sind
- Nutzt Service Workers, um auf Plattform APIs zuzugreifen

Eine PWA wird zunächst ähnlich wie eine mobile Web App über einen Webbrowser via HTTPS aufgerufen. Wird die PWA innerhalb kurzer Zeit mehrmals aufgerufen, können UserInnen entscheiden, ob diese zum Homescreen als App hinzugefügt wird. Die Daten dafür werden im Web App Manifest hinterlegt (Malavolta, 2016).

Web App Manifest

Ein Web App Manifest ist eine JSON (JavaScript Object Notation) Datei und kann wie folgt aufgebaut sein (Farrugia, 2016) (siehe Listing 1):

Listing 1. Beispiel PWA Web App Manifest, angepasst von (Farrugia, 2016)

```
//manifest.json
{
   "name": "Manifest Example",
   "short_name": "Manifest",
   "description": "Example for a manifest",
   "icons": [
      ....
    ],
  "start_url": "index.php",
  "orientation": "landscape",
  "display": "standalone",
  „theme_color“: "#00f",
  "background_color": "#f00"
```

```
}

// include the script in the <head> section
<head>
  ...
  <link rel="manifest" href="manifest.json">
</head>
```

Mit der Eigenschaft `name` legt man zunächst den Namen der App fest, der beim Aufruf, „App zum Homebildschrim hinzufügen“, angezeigt wird. Der `short_name` ist der Kurzname, der unterhalb des Icons auf dem Homescreen angezeigt wird. Zusätzlich bietet die `description` eine allgemeine Beschreibung der Webanwendung. Im `icons` Array werden die Icons als Objekte für die unterschiedlichen Bildschirmgrößen der Endgeräte definiert. Die `start_url` gibt den Startpunkt der Anwendung an. Bei Bedarf kann hier eine alternative Startseite festgelegt werden. Mit `orientation` kann man die voreingestellte Ausrichtung beim Öffnen der App angeben, zur Auswahl stehen `portrait` oder `landscape`. Die Eigenschaft `display` definiert den Modus wie die App dargestellt wird, `standalone` z.B. blendet alle browserähnlichen Darstellungen aus und zeigt nur die Statusbar oben an. Über `theme_color` und `background_color`, kann das Erscheinungsbild der Anwendung angepasst werden. Hier können sowohl Farbnamen als auch hexadezimale Farbwerte verwendet werden. Damit dieses Manifest interpretiert werden kann, muss es abschließend noch in das `<head>` Tag der HTML Datei eingebunden werden (Farrugia, 2016).

Jedoch ist der aktuelle Browsersupport des Web App Manifestes derzeit noch nicht besonders hoch (siehe Abbildung 5).

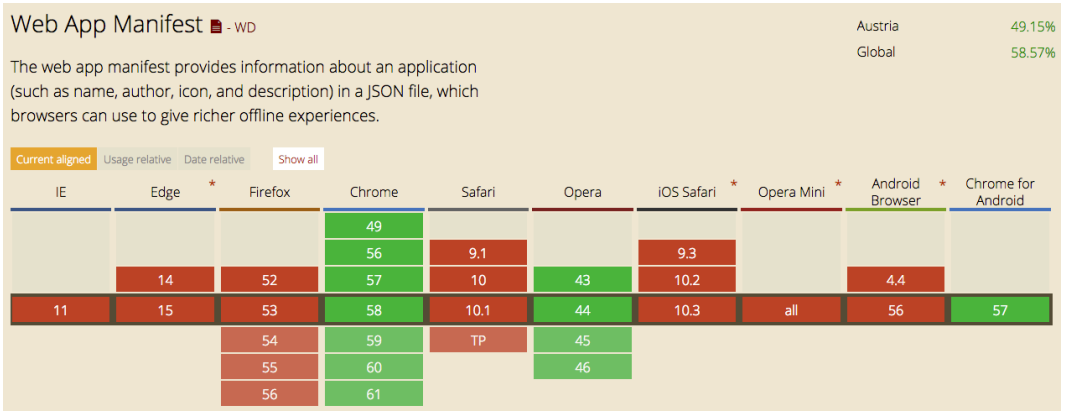

Abbildung 5. Screenshot derzeitige Unterstützung Web App Manifest, Stand: 02.06.2017; http://caniuse.com/#search=web%20app%20manifest

Derzeit unterstützen nur die Browser Chrome und Opera sowie Chrome für Android das Web App Manifest, was lediglich eine Unterstützung von knapp 60% weltweit bedeutet. Laut Steyer (2017) gibt es allerdings für Browser, die kein Manifest interpretieren können, die Möglichkeit über Metatags im `<head>` Bereich der Webseite, diese Einstellungen vorzunehmen.

Service Worker

Die eigentliche Kern von PWA sind die Service Worker (SW):

„Dabei handelt es sich um Hintergrundprozesse, die losgelöst von der eigentlichen App ablaufen, und Offlineszenarien, Datensynchronisation sowie Push Notifications unterstützen“ (Steyer, 2017).

Der SW läuft als eine Art Vermittler zwischen Webseite und dem Server. Fordert der Browser z.B. eine Ressource vom Server an, gelangt diese zuerst zum SW und erst danach wird sie über das Internet an den Browser zurückgesendet. Dadurch, dass der SW vollen Zugriff zum Cache des Browsers hat, kann dieser sämtliche Ressourcen dort ablegen und offline zur Verfügung stellen, sodass es keine Rolle spielt, wie gut die Internetverbinung der UserInnen ist (Bachert, 2016).

SW können HTTPS-Anfragen abfangen und dynamisch entscheiden, woher die Ressource geladen wird: aus dem Cache oder über das Netzwerk. Je nach gewählter Caching-Strategie, kann aus sieben unterschiedlichen Strategien gewählt werden. So können die Ressourcen entweder kompett aus dem Cache oder Netzwerk geladen werden oder eine Mischung davon. (Steyer, 2017).

3.2 Mobile Webentwicklung

3.2.1 Frameworks und Libraries

Bei der prototypischen Umsetzung der Arthrose Plattform und bei der Bewertung der Entwicklungstechnologien werden unterschiedliche Frameworks und Libraries eingesetzt. Diese beiden Fachbegriffe werden oft gleichbedeutend für ein und dieselbe Sprache verwendet, müssen jedoch unterschieden werden. Die folgende Unterscheidung gilt hauptsächlich für JavaScript Frameworks und Libraries, da im Zusammenhang mit Stylesheets lediglich CSS Frameworks genannt werden.

Framework

Bereits seit 2007 werden in der Webentwicklung Frameworks zur Erstellung von Webanwendungen verwendet. Jeff Croft definiert Frameworks als:

„a set of tools, libraries, conventions, and best practices that attempt to abstract routine tasks into generic modules that can be reused.“ (Croft, 2007)

Croft charakterisiert Frameworks als eine Sammlung von Werkzeugen, Bibliotheken und Methoden, die dabei helfen sollen, regelmäßig wiederkehrende Aufgaben in Module zu kapseln, welche wiederverwendet werden können. EntwicklerInnen sind somit in der Lage für verschiedene Projekte auf ein Grundgerüst an bereitgestellten Funktionen zurückgreifen zu können und sich

gezielt auf spezielle Anfordungen des Projekts zu konzentrieren, anstatt jedes Projekt von Grund auf neu zu definieren (Croft, 2007).

Es ist nichts Ungewöhnliches, dass einige WebentwicklerInnen sich in den letzten Jahren ihre eigenen Basisdateien als Framework zurechtgelegt haben, um ein neues Projekt zu starten. Dies lässt eventuell auch auf die große Anzahl an kostenlos angebotenen Frameworks schließen, aus denen heutzutage gewählt werden kann. Je nach Projekt muss abgewägt werden, ob und welches Framework zum Einsatz kommen soll. Grundsätzlich kann gesagt werden, dass Frameworks nach einer gewissen Einarbeitungszeit, meist zeitsparend sind und einheitliche Coderichtlinien in größeren Entwicklungsteams vorschreiben. Bei falscher Nutzung kann sich jedoch der Code nachteilig aufblähen und letztendlich zu einem erhöhten Verwaltungsaufwand führen (Grochtdreis, 2012, S. 112ff.)

Library

Der Unterschied zwischen einem Framework und einer Library kann am besten durch die nachfolgende Abbildung 6 visualisiert werden:

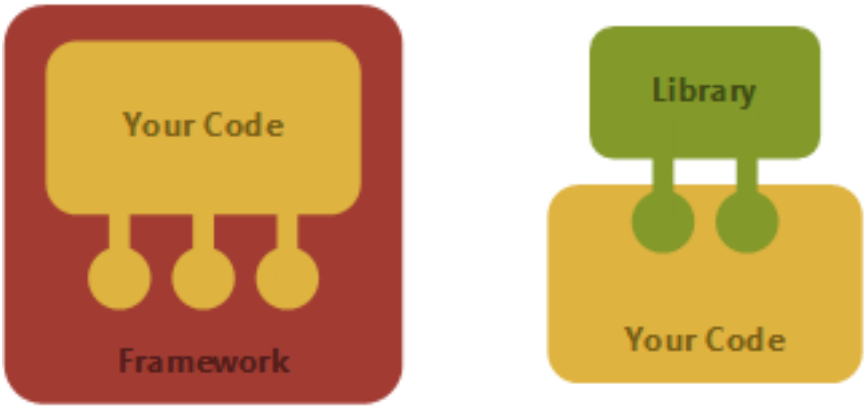

Abbildung 6. Unterschied zwischen Framework und Library, (Petricek, 2015)

Das Framework stellt ein funktionierendes Grundgerüst dar, in dem der selbst geschriebene Code der Anwendung platziert werden kann und beliebig neue Funktionen und Komponenten hinzugefügt oder entfernt werden können. Bei der Library hingegen wird der von WebentwicklerInnen selbst geschriebene Code mit Hilfe einer Sammlung hilfreicher Komponenten, welche die Library zur Verfügung stellt, erweitert. Buckler (2017) benutzt zum besseren Verständnis einen Vergleich mit einem Auto: ein Framework liefert ein funktionierndes Fahrgestell inklusive Karosserie und Motor, an dem man selbst basteln kann, solange das Auto funktionsfähig bleibt. Eine Library hingegen, stellt Autobauteile zur Verfügung, aus denen man selbst ein fahrtüchtiges Fahrzeug konstruieren muss (Buckler, 2017).

Fazit

Zusammenfassend kann also gesagt werden, dass eine Unterscheidung zwischen Frameworks und Libraries nur in den seltensten Fällen eindeutig ist, denn auch ein Framework könnte eine Library enthalten und eine Library kann auch framework-ähnliche Methoden implementieren (Buckler, 2017).

3.2.2 CSS Methodologien

Bei der mobilen Webentwicklung gehört das Schreiben von CSS zum Stylen des Inhaltes zu den Standardaufgaben von Frontend EntwicklerInnen. Dadurch, dass Mobile Web Apps im Laufe der letzten Jahre immer größer und komplexer geworden sind, hat auch die Anzahl an CSS Codezeilen in vielen Projekten deutlich zugenommen. Infolgedessen wird es somit wesentlich schwiergier, den Code zu verwalten und den redundanten, sich überschreibenden Code zu vermeiden oder ausfindig zu machen. Da oftmals auch kein einheitlicher Styleguide zum Schreiben von CSS vorliegt, leidet die Performance macher Webseiten unter der übermässigen Anzahl an CSS Zeilen (Rishabh, 2015).

Diese Problematik führte dazu, dass in den letzten Jahren unterschiedliche Ansätze entwickelt wurden, um das Schreiben und Gliedern von CSS für EntwicklerInnen zu verbessern und zu erleichtern (Rishabh, 2015). Im folgenden Abschnitt werden drei dieser Verfahren detailierter erläutert und mit Codebeispielen ergänzt.

OOCSS

Das Akronym OOCSS steht für „Object Oriented CSS“ (Objektorientiertes CSS) und wurde 2009 von Nicole Sullivan vorgestellt. Das Ziel Sullivans war es, CSS übersichtlicher zu machen, indem man objektorientiere Prinzipen von Programmiersprachen wie Java oder Ruby übernimmt und dadruch die Wiederverwendbarkeit und Wartbarkeit von CSS Code erhöht. OOCSS wird in zwei grundlegende Prinzipien eingeteilt (Arsenault, 2017):

Trennung von Struktur und Aussehen (Separation of structrue/skin)

Das erste Prinzip handelt von der klaren Trennung zwischen Strukturaspekten die für UserInnen „unsichtbar“ sind wie z.B. Breiten- und Höhenangaben, Positionsangaben und Abstände, etc. und visuelle Aspekten von Elementen wie z.B. Farben, Schriften oder Hintergründe. Um ein Objekt zu positionieren und es zu stylen, bedarf es nach diesem Prinzip mindestens zwei verschiedener CSS Klassen (Arsenault, 2017).

Trennung von Rahmenelementen und Inhalten (Separation of container/content)

Das zweite Prinzip sorgt für die Trennung von Inhaltstypen wie Buttons, Bildern oder Texten und deren Rahmenelementen (z.B. `<header>`, `<footer>`) in denen diese Inhaltstypen platziert werden. Visuelle Aspekte von Elementen sollten, wenn möglich, nicht auf die Rahmenelemente angewandt werden, sondern auf die Inhaltstypen selbst (Arsenault, 2017).

Falsch angewendet, kann dieser Ansatz schnell dazu führen, dass ein Element sehr viele Klassen benötigt und damit der Code eine übermäßig große Anzahl an Klassen enthält (auch „Classitis“[2] genannt). Frontend WebentwicklerInnen müssen daher eine Lösung finden, wie sie mit möglichst wenigen Klassen zur besseren Lesbarkeit und Wartbarbeit von CSS beitragen können (Rishabh, 2015).

SMACSS

SMACSS steht für „Scalable and Modular Architecture for CSS“. Das Grundprinzip hinter SMACSS ist die Aufteilung von CSS Regeln in Kategorien. Dies soll bewirken, dass wiederkehrende Muster entstehen. SMACSS lässt sich daher in fünf Kategorien gliedern (Snook, 2012, S. 4):

- Base
- Layout
- Module
- State
- Theme

Der Zweck der Klassifikation ist, Wiederholung innerhalb des Codes zu vermeiden und somit die Wartbarbeit zu erleichtern. Für jede der genannten Kategorien gelten bestimmte Richtlinien (Snook, 2012).

Basisregeln (Base)

Basisregeln beinhalten Standards für jene Elemente, die in der gesamten Anwendung das gleiche Aussehen haben sollen. Diese enthalten keine Klassen oder ID-Selektoren, Pseudo-Klassen hingegen sind erlaubt (Snook, 2012, S. 8) (siehe Listing 2).

Listing 2. SMACSS Basisregeln (Snook, 2012, S.8)

```
html, body { margin: 0; padding: 0; }
input[type=text] { border: 1px solid #333; }
a { color: #ddd; } a:hover { color: #ccc; text-decoration: underline;}
```

[2] http://www.steveworkman.com/html5-2/standards/2009/classitis-the-new-css-disease/

Layoutregeln (Layout)

Layoutregeln umfassen größere Komponenten auf der Seite, wie den Kopf- oder Fußbereich oder den Hauptinhalt. Diese Bereiche werden oftmals mit ID-Selektoren angesprochen, da sie nur einmal auf der Seite vorkommen. Größere Komponenten, die jedoch öfter auf der Seite vorkommen, sollten mit angemessenen Klassenselektoren angesprochen werden. Diesen Klassen können zur besseren Differenzierung das Prefix `l-` oder `layout-` vorangestellt werden (Snook, 2012, S. 10) (siehe Listing 3).

Listing 3. SMACSS Layoutregeln (Snook, 2012, S.10)

```
#header, #footer { width: 100%; margin: 0 auto; }
.l-article { width: 33%; border-right: 1px solid #333; margin: 10px; }
```

Modulregeln (Module)

Module sind eigenständige Komponenten im Layout und sollten so entworfen werden, dass sie für sich eigenständig sind und flexibel in unterschiedlichen Regionen der Seite platziert werden können, ohne, dass dadurch das Layout beeinträchtigt wird. Es ist darauf zu achten, dass keine ID- oder Element Selektoren (`<span>`) verwendet werden und ausschließlich Klassenamen zum Einsatz kommen. Das nachfolgende Listing 4 zeigt CSS Selektoren für eine eigenständige Hinweis (`alert`) Komponente (Snook, 2012, S. 19).

Listing 4. SMACSS Modulregeln (eigener Code)

```
.alert { padding: 10px; border: 1px solid red}
.alert-heading { font-size: 1.5rem; color: red; }
.alert-desc { margin-top: 1rem; font-size: .875rem; }
```

Statusregeln (State)

Statusregeln erweitern oder überschreiben den Zustand eines Elements (Snook, 2012, S. 25). Man benützt z.B. Klassen wie `.is-active` oder `.is-hidden`, um den visuellen Zustand eines Modules oder eines Layoutelements an seinen aktuellen Zustand anzupassen und auf UserInneneingaben zu reagieren. Diese Klassen überschreiben somit den Zustand ihres Basismodules (Rishabh, 2015).

Themenregeln (Theme)

Themenregeln können als eine Art „Skin“ (Aussehen) betrachtet werden, in denen das Look & Feel der Seite geändert werden kann, indem man Layoutelemente oder Module überschreibt. Man legt eine unsichtbare „Ebene“ über das eigentliche Layout um die Seite für UserInnen zu Personalisieren oder an bestimmte Anlässe (z.B. Weihnachtsaktion) anzupassen (Rishabh, 2015).

Die Idee hinter der CSS Methodologie SMACSS ist, die Anwendung in wiederverwendbare Komponenten aufzuteilen und ID- sowie Element Selektoren weitesgehend zu vermeiden und hauptsächlich Klassen zu verwenden. Es ist jedoch erlaubt, Selektoren wie `.element h2` oder Kind-Selektoren wie `.element > a` zu verwenden, wenn man sicherstellen kann, dass alle Elemente in der Klasse gleich aussehen werden oder nur auf diesem Level existieren (Rishabh, 2015).

BEM

Die CSS Methodologie BEM steht für „Block Element Modifier“ und hat einige Gemeinsamkeiten mit dem SMACSS Ansatz. BEM stellt Namenkonventionen für HTML und CSS zur Verfügung, um die gesamte Webanwendung in Blöcke und deren Elemente einzuteilen (Rishabh, 2015). Das nachfolgende Listing 5 verdeutlicht dieses Prinzip anschaulich:

Listing 5. BEM Beispiel (eigener Code)

```
<div class="tip">
  <div class="tip__symbol">
    <svg class="tip__symbol-item tip__symbol-item--xs">...</svg>
  </div>
  <div class="tip__box">...</div>
</div>
```

Block

Blöcke sind wiederkehrende Komponenten auf einer Webseite. Im vorangehenden Beispiel (Listing 5) ist die Klasse `tipp` ein Block, der in beliebigen Bereichen der Seite zum Einsatz kommen kann. Unabhängig von seinem Elternelement soll der Block wahllos in der Anwendung platziert werden können, ohne das sich dadurch das Layout ändert. Einfache weitere Blockbeispiele wären z.B. Buttons, Listen oder Formulare (Rishabh, 2015).

Element

Innerhalb eines Blockes können wiederum unzählige Elemente platziert werden. Diese „Kindelemente“ haben eine definierte Namenskonvention und werden durch zwei Unterstriche nach dem Block angegeben (`block__element`). Der Block `tipp` im Listing 5 enthält zwei Kindelemente (`tip__symbol` und `tip__box`) welche ein Symbol und den eigentlichen Inhalt des Blockes bereitstellen (Rishabh, 2015).

Modifier

Modifier verändern das visuelle Erscheinungsbild eines Blocks oder Elements und können mit Themenregeln von SMACSS vergleichen werden. Diese zusätzlichen Klassen werden durch zwei Bindestriche nach dem Block oder Element gekennzeichnet. Im Listing 5 bekommt das Element eine Modifier Klasse `tip__symbol-item--xs`, was bewirken soll, dass das Element in seiner Darstellung verkleinert angezeigt wird (Rishabh, 2015).

Michalak (2015) behauptet jedoch, dass der BEM Ansatz schnell zu langen Klassennamen führen kann. Kommen zusätzlich zu dieser Klasse auch noch Modifier hinzu, kann das ganze relativ schnell unübersichtlich werden. Die Folge sind oftmals aufgeblähte und lange Klassenbezeichnungen, sowohl im CSS als auch im HTML Code, wodurch die Lesbarbeit und Übersichtlichkeit abnimmt. Berner (2016) hingegen argumentiert, dass der Code durch die Vielzahl an Klassen und Modifier lesbarer wird und man dadurch jederzeit genau weiß, was der Block oder das Element machen soll. In dem Artikel „Battling BEM CSS“ listet Berner zehn häufige Probleme mit dem BEM Ansatz auf und gibt Ratschläge wie man diese vermeiden kann[3].

3.2.3 Task Runners und Building Tools

Im mobilen Web Development Workflow sind Task Runners und Building Tools ein essenzieller Bestandteil bei der Entwicklung einer Webseite oder Applikation geworden. Task Runners und Building Tools sind JavaScript basierende Werkzeuge, welche durch die Automatisierung von unzähligen Entwicklungsaufgaben einen Mehrwert bei der Erstellung von Webanwendungen schaffen. Durch die Verwendung eines dieser Automatisierungswerkzeuge im Frontend Workflow können u.a. CSS oder JavaScript Dateien verknüpft und komprimiert werden oder auch Preprozessoren wie LESS oder SASS (Syntactically Awesome Style Sheets) eingesetzt werden, um CSS Codes zu kompilieren (Simpson, 2016). Nachfolgend werden die drei derzeitig gängisten Werkzeuge kurz vorgestellt und abschließend deren Vor- und Nachteile tabellarisch veranschaulicht. Alle hier vorgestellten Werkzeuge und deren Plug-Ins können mittels des Node Package Managers[4] (npm) installiert werden. Da auch bei der prototypischen Umsetzung der Arthrose Plattform ein Task Runner zum Einsatz kommen soll, werden die nachfolgenden Beispiele anhand der CSS Kompilierung mit Hilfe des Preprozessors SASS verdeutlicht.

Grunt

Grunt wurde im März 2012 von Ben Alman veröffentlicht. Ursprünglich gedacht als aufgabenbasiertes Command Line Build Tool für JavaScript Projekte, bezeichnet sich Grunt mittlerweile selbst als „The JavaScript Task Runner“. Durch den Anstieg an JavaScript basierten Web Applikationen hat Grunt in der Open Source Welt mittlerweilte eine große Beliebtheit erreicht (Pillora, 2014, S. 7). Der Einstieg bei Grunt erfolgt über das `Gruntfile.js`. Über diese im Root-Verzeichnis gelegene JavaScript Datei werden die verschiedenen Plug-Ins konfiguiert, welche

[3] https://www.smashingmagazine.com/2016/06/battling-bem-extended-edition-common-problems-and-how-to-avoid-them/

[4] Infomationen zur Installation von npm: https://www.npmjs.com/get-npm

unterschiedliche Aufgaben erledigen (Simpson, 2016). Das folgende Listing 6 zeigt den examplarischen Aufbau eines Gruntfiles.

Listing 6. Aufbau eines Gruntfile.js (Angepasst von Pillora, 2014, S. 8)

```
module.exports = function (grunt) {
  grunt.loadNpmTasks('grunt-contrib-sass');
  grunt.loadNpmTasks('grunt-contrib-watch');

  grunt.initConfig({
    // Tasks
    sass: {
      dist: {
        dest: 'css', ext: '.css'
      }
    },
    watch: {
      css: {
        files: '**/*.scss', tasks: ['sass']
      }
    }
  });

  // Register Grunt tasks
  grunt.registerTask('default', ['sass']);
};
```

Zuerst werden in den Zeilen 2 und 3 mittels der Funktion `loadNpmTasks` die jeweiligen Plug-Ins geladen, welche in der, im gleichen Verzeichnis liegenden Datei `package.json`, als Abhängigkeiten definiert sind. Anschließend werden die Aufgaben definiert. Dem sass-Task können jeweils unterschiedliche Parameter übergeben werden, wie beispielsweise legt `dest` (destination) den Zielordner fest, in dem die komprimierten CSS Dateien gespeichert werden sollen und der Parameter `ext` (extension) legt die Dateiendung fest. Daraufhin können dem `watch` Plug-In mehrere Tasks übergeben werden, die beobachtet werden sollen. In diesem Beispiel gibt es nur den Task `sass` zu überwachen. Wenn in einer Datei mit dem Suffix `.scss` eine Änderung durchgeführt werden sollte, wird der sass-Task ausgeführt. Gibt man nun in die Kommando Zeile den Befehl `grunt watch` ein, wird die in der letzten Zeile definierte Funktion `registerTask` abgewickelt und ein Überwachungsprozess gestartet (Pillora, 2014, S. 8).

Gulp

Gulp erschien kurze Zeit nach Grunt und im Gegensatz zum im vorherigen Abschnitt gezeigten `gruntfile.js`, welches als Konfigurationsdatei gesehen werden kann, ist das `gulpfile.js` als JavaScript Code zu verstehen. Gulp folgt dem Prinzip „Code über Konfiguration“. Damit erreicht Gulp zwar eine höhere Kompaktheit und Flexibilität, zu oft sehr langen und unübersichtlichen Konfigurationsdateien wie bei Grunt, aber EntwicklerInnen müssen sich zuerst mit

der Node.js Streaming API vertraut machen (Simpson, 2016). Diese Streaming API nutzt das Konzept der „Streams“ (Ströme) und „Pipes“ (Kanäle). Dabei schickt Gulp die jeweiligen Aufgaben durch Pipes, welche nacheinander abgearbeitet werden und somit den Arbeitsablauf beschleunigen (Sauer, n.d.). Im Folgenden examplarischen Listing 7 wird dieses Stream-Pipe Konzept genauer deutlich.

Listing 7. Aufbau eines Gulpfile.js (eigener Code)

```
var gulp = require('gulp'),
   sass = require('gulp-sass'),
   prefix = require('gulp-autoprefixer'),
   notify = require("gulp-notify"),
   cleanCSS = require('gulp-clean-css'),
   cssFiles = 'assets/sass/**/*.scss';

gulp.task('sass', function () {
  return gulp.src(cssFiles)
      .pipe(prefix(...))
      .pipe(cleanCSS({...}))
      .pipe(notify("Success in creating SASS file"))
      .pipe(gulp.dest('public/assets/css'));
});

gulp.task('watch', function() {
  gulp.watch(cssFiles, ['sass']);
});
```

Ähnlich wie bei Grunt werden zuerst die benötigen Plug-Ins aus den Abhängigkeiten (Dependencies) der `package.json` Datei gelesen und in Variablen gespeichert. Danach wird der sass-Task erstellt. In der Callback Funktion (`return`) dieser Aufgabe wird die `gulp.src` API benutzt, um alle Dateien mit der Endung `.scss` zu erfassen und diesen Strom der Reihe nach durch diverse Kanäle zu schicken (Simpson, 2016). Zuerst werden mit dem `prefix()`-Kanal je nach Angabe der Browserversionen alle CSS Eigenschaften mit browserspezifischen Präfixen versehen. Mit `cleanCSS()` wird das generierte CSS ohne Leerzeichen zusammenhängend in eine Datei geschrieben, um die Größe der Datei zu reduzieren. Sobald dies passiert ist, wird mit `notify()` eine Meldung ausgegeben damit der/die EntwicklerIn weiß, dass der Prozess erfolgreich war und in den angegeben Ordner in `gulp.dest()` abgespeichert.

Abschließend wird noch ein watch-Task erzeugt. Der Funktion `gulp.watch` werden zwei Parameter übergeben. Als Erstes wird der Dateipfad angegeben, der auf Änderungen hin überwacht werden soll. Wird eine Änderung in dem angegebenen Pfad gemacht, wird zweitens der sass-Task ausgeführt und die definierte `.pipe` Reihefolge durchgegangen.

Ein Gulp Workflow besteht somit immer aus drei Schritten: Zuerst alle Dateien eines bestimmten Types finden und einlesen, diesen erzeugten Strom mittels

Plug-Ins nach den jeweiligen Anforderungen bearbeiten und schlussendlich an der gewünschten Position ausgeben (Simpson, 2016).

Webpack

Webpack ist im Vergleich zu Grunt und Gulp kein Task Runner, sondern ein „Module Bundler“. Webpack analysiert die Struktur des Projektes, sucht dort nach Assets wie JavaScript- oder CSS Dateien und Bildern, um diese dann zu bündeln und für den Browser aufzubereiten (Antonio, 2015).

Abbildung 7 zeigt den unterschiedlichen möglichen Workflow eines Task Runners im Vergleich zu Webpack.

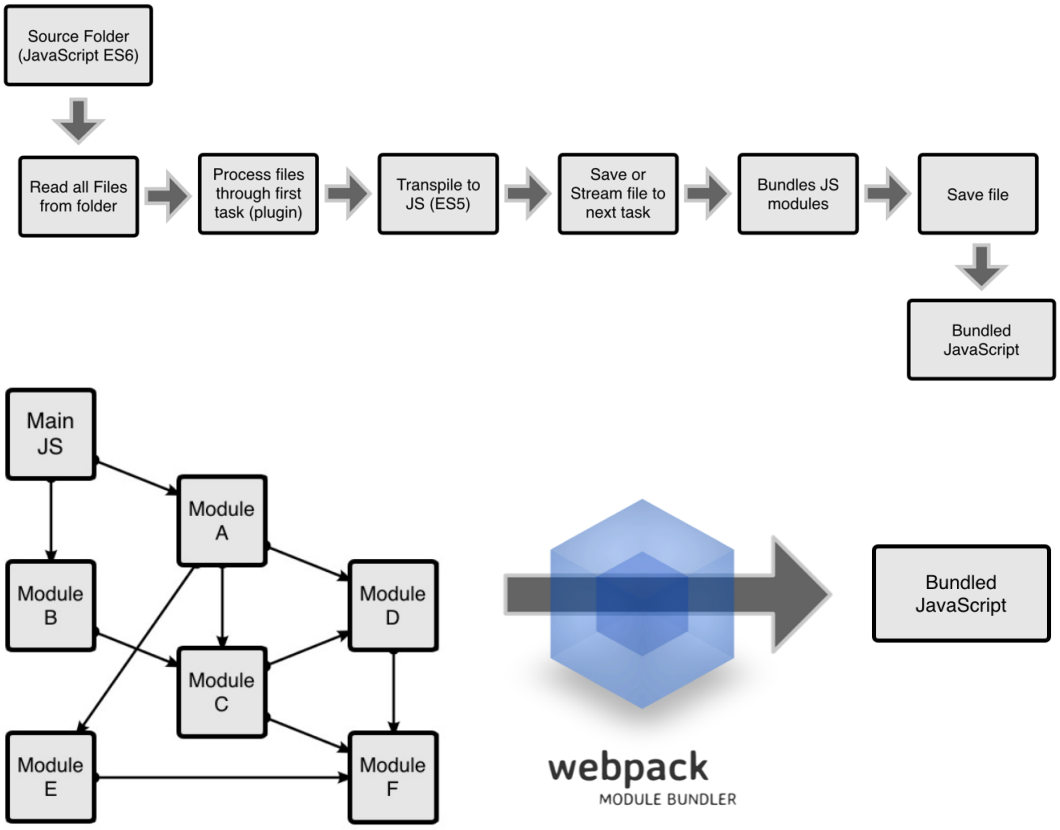

Abbildung 7. Beispielhafter Workflow Vergleich zwischen einem Task Runner (oben) und Webpack (unten), (Antonio, 2015)

Task Runner suchen in einem definierten Ordner oder Pfad nach Dateien welche der angegebenen Konfiguration entsprechen und lesen diese ein. Danach durchlaufen die Dateien unterschiedliche Prozesse oder Aufgaben durch die gewählten Plug-Ins. Diese werden transformiert oder kompiliert und abschließend gebündelt an einem definierten Pfad wieder ausgegeben. Webpack hingegen analysiert das Projekt als geschlossene Einheit. Ausgehend von einer Einstiegsdatei schaut sich Webpack alle Abhängigkeiten eines Projektes an, verarbeitet diese mit Hilfe von „Loadern“ und bündelt sie in das gewünschte

Endformat. Dieser Ansatz macht Webpack schneller und geradliniger und eröffnet somit neue Ansätze für das Bündeln von verschiedenen Dateiarten. (Antonio, 2015).

Auch hier wird an einem Beispiel die CSS Kompilierung mit Hilfe des Preprozessors SASS verdeutlicht (siehe Listing 8).

Listing 8. Aufbau einer Webpack Konfiguration Datei (eigener Code)

```
var ExtractTextPlugin = require("extract-text-webpack-plugin");

module.exports = {
  entry: ['./sass/app.scss'],
  output: {
    path: __dirname + '/public',
    filename: 'dist/bundle.js'
  },
    module: {
      rules: [{
          resource: /\.(sass|scss)$/,
          use: ExtractTextPlugin.extract(['css-loader', 'sass-
loader'])
      }]
    },
    plugins: [
      new ExtractTextPlugin({
        filename: '/css/app.css',
      }),
    ],
  };
```

Die Konfigurationsdatei von Webpack muss `webpack.config.js` heißen und als Mindestanforderung muss mindestens einen Einstiegs- (`entry`) sowie einen Ausgangspunkt (`output`) enthalten. Der Einstiegspunkt ist hier die root-SASS Datei, in welcher alle `.scss` Dateien zusammenlaufen. Anschließend wird in den `rules` Array mit `resource` eine Bedingung angegeben, dass nur Dateien mit der Endung `.sass` oder `.scss` verwendet werden dürfen, welche mit den im Parameter `use` angegebenen Plug-Ins geladen werden. Diese Plug-Ins werden in der `package.json` Datei als Abhängigkeiten angegeben und als Variablen in der `webpack.config.js` definiert (Zeile 1). Zuletzt wird im `plugins` Array der Pfad und der Name der kompilierten CSS Dateien angegeben (Simpson, 2016).

Zusammenfassung/Überblick

In der nachfolgenden Tabelle 2 (siehe Seite 35) werden als Zusammenfassung die Vor- und Nachteile der in den vorherigen Abschnitten genannten Task Runners und Module Bundler übersichtlich dargestellt (Simpson, 2016, Sauer, n.d., Antonio, 2015 und Pillora, 2014:

Tabelle 2. Vor- und Nachteile von Task Runners/Module Bundler im Überblick

	Grunt	***Gulp***	***Webpack***
+	*- Große Community* *- Hohe Flexibilität* *- Ähnlich große Anzahl an Plug-Ins wie Gulp* *- Stabil und ausgereift* *- Keine Programmierkenntnisse notwendig*	*- Schneller als Grunt durch schnelle Node.js Streams* *- Kompakter, flexibler und übersichtlicher als Grunt* *- Schnellere und einfachere Programmierung möglich* *- Abarbeitung mehrere Aufgaben gleichzeitig (asynchron)* *- Ein Plug-In für eine Aufgabe* *- Große Anzahl an Plug-Ins*	*- Kann komplexe, strukturierte Daten verarbeiten* *- Geeignet für EntwicklerInnen mit geringen JS Kenntnissen* *- schneller und einfacher als Task Runners*
-	*- Oftmals entstehen lange und unübersichtliche Konfigurationsdateien*	*- Höheres technisches Wissen wird benötigt (über Node.js Streaming API und JavaScript)*	*- unübersichtliche Dokumentation* *- Komplizierter / schwerer Workflow für allgemeine Aufgaben wie CSS Komplierung*

Auch wenn Webpack schneller als mancher Task Runner ist, so ist Webpack noch relativ neu und bietet eine etwas unübersichtliche Dokumentation. Nicht nur deshalb muss man bei Problemen eventuell etwas länger nach Lösungen suchen. Grunt bietet viele Vorteile und ist vor allem für EntwicklerInnen mit wenig bis gar keinen Programmierkenntnissen geeignet. Jedoch können die Konfigurationsdateien bei mittleren bis größeren Projekten rasch unübersichtlich werden. Obwohl zwar beim Task Runner Gulp ein höhers technisches Know-How erforderlich ist, überwiegen dessen Vorteile gegenüber den anderen beiden Tools. Die kompakte, einfache und übersichtliche Syntax sowie Darstellung und Aufbereitung der Codes ist ein wesentlicher Bestandteil bei der Erstellung einer mobilen Webentwicklung und daher auch bei der prototypischen Umsetzung der Arthrose Plattform zu bevorzugen.

3.2.4 Cache Manifest

Damit einzelne Seiten der Arthrose Plattform nicht bei jedem Aufruf neu geladen werden müssen, können statische Seiten, deren Inhalte sich nur sehr selten ändern, im Cache Manifest des Browsers angegeben werden. Die dort aufgelisteten Seiten werden vom Browser lokal speichert und ausgegeben, wenn

der Benutzer offline ist und die Plattform trotzdem verwenden will. Beim erstmaligen Aufruf der Webseite werden die im Cache Manifest angegebenen Dateien in den Cache des Browsers geladen und zu einem späteren Zeitpunkt wieder von dort bezogen, sollte der/die UserIn keine aktive Internetverbindung haben (Novy, 2015, S. 41). Das nachfolgende Listing 9 zeigt den examplarischen Aufbau einer Cache Manifest Datei und das Einbetten in den HTML Code.

Listing 9. Beispiel für ein Cache Manifest

```
// Aufbau Cache Manifest
CACHE MANIFEST
# v1 – 2017-07-11
CACHE:
/index.html
/css/app.css
/js/app.js
# Absolute Pfadangabe
https://www.beispiel.com/images/logo.svg
NETWORK:
*
FALLBACK:
/.html /offline.html

//Einbinden in HTML Code
<html lang="de" manifest="manifest.appcache">
```

Das Cache Manifest kann in drei Bereiche unterteilt werden: Cache, Network und Fallback. Im ersten Abschnitt `CACHE` werden die Dateien festgelegt, welche nach dem ersten Heruntergeladen im Cache gespeichert werden sollen und somit dem/der UserIn offline zur Verfügung stehen. Die Pfadangabe zur Datei, kann sowohl relativ als auch absolut sein. Der Abschnitt `NETWORK` legt fest, was mit den restlichen Dateien passieren soll. Indem man einen "*" setzt, ist für das Beziehen aller übrigen Dateien eine Verbindung zum Internet erforderlich. Im letzten optionalen Bereich `FALLBACK`, werden Fallback Seiten angegeben für den Fall das eine Ressource nicht gefunden werden kann. Wird eine Datei mit dem Suffix `.html` nicht gefunden, wird stattdessen die Datei `offline.html` als Fallback ausgegeben. Abschließend wird dann mit dem Attribut „manifest" im `<html>` Tag auf die Cache Manifest Datei referenziert (Bidelman, 2010).

Bei der Verwendung des Cache Manifestes sollte jedoch darauf geachtet werden, inwieweit diese Schnittstelle in Zukunkt unterstützt wird. Laut HTML Living Standard[5] wird die Unterstützung dieser Funktion irgendwann eingestellt werden. Auch wenn der Prozess länger dauern sollte, wird empfohlen schon jetzt auf das Service Worker Manifest zurückzugreifen (siehe Kapitel 3.1.3).

[5] https://html.spec.whatwg.org/multipage/offline.html#offline

3.2.5 Promises

Um die Personalisierung der Plattform zu ermöglichen, wurde mit Hilfe des Frameworks AngularJS ein „Service“ (siehe Kapitel 5.2.1) erstellt. Services nutzen das relativ neue Konzept von Promises. In JavaScript verkörpern Promises das Ergebnis einer asynchronen Operation. Promises stellen eine `then`-Methode zur Verfügung, die aufgerufen wird, wenn die Operation ausgeführt wurde und abgeschlossen ist. Daher befindet sich ein Promise immer in einem von drei Zuständen (Novy, 2015, S. 31):

- **Pending**: Das Promise wird einen der beiden Zustände (fulfilled oder rejected) annehmen
- **Fulfilled**: Das Promise wurde erfolgreich mit einem Wert aufgelöst
- **Rejected**: Das Promise wurde mit einem Fehler aufgelöst

Das AngularJS Framework bietet einen eigenen Promise-Service $q[6] an, um Promises zu erzeugen. Der Einsatz dieses bereits enthaltenen Service stellt unterschiedliche Methoden zur Verfügung. Eine davon ist die `deferred` API. Mit der Methode `defer()` kann somit ein deferred-Objekt erzeugt werden (Lerner, 2013, S. 232) (siehe Listing 10).

Listing 10. Erzeugen eines deferred-Objektes (Lerner, 2013, S. 232)

```
var deferred = $q.defer();
```

Das deferred-Objekt bietet drei Methoden und eine einzelne Promise Eigenschaft an, welche EntwicklerInnen nutzen können, um mit Promises zu interagieren (Lerner, 2013, S. 232):

- **Resolve**(Wert): Auflösung des Promise mit einem Wert
- **Reject**(Grund): Ablehnung des Promise mit einem Grund
- **Notify**(Wert): Rückgabe des Zustands des Promise

Ein detailiertes Beispiel, wie Promises eingesetzt werden können, wird in Kapitel 5.2.1.2, bei der Umsetzung der Personalisierung der Arthrose Plattform, nochmals im Detail erläutert.

[6] AngularJS Promises mit $q: https://docs.angularjs.org/api/ng/service/$q

3.3 Lokale Personalisierung durch browserseitige Datenspeicherung

Da sich die Zielgruppe der Arthrose Plattform nur ungern an bestehenden Systemen registrieren bzw. einloggen will sowie keine persönliche Daten von sich preisgeben möchte (El Aeraky, 2017), muss eine alternative Art der Datenspeicherung verwendet werden, um die Plattform an die jeweiligen Bedürfnisse anzupassen. Die Möglichkeiten, Daten lokal im Browser zu speichern und zu verwalten, haben sich in den letzten Jahren stark weiterentwickelt. Durch diese Konzepte können webbasierte mobile Apps schneller geladen, offline verfügbar und die User Experience merklich verbessert werden. Statische Daten und Einstellungen können im Browser der UserInnen gespeichert werden, jedoch sollten sensible und empfindliche Daten immer unzugänglich für Dritte auf Servern gespeichert werden (Laine, 2012, S. 1).

Die älteste, aber stabilste Art, Daten lokal zu speichern sind Cookies. Diese wurden 1994 in einer Beta Version des damaligen Browsers Netscape eingeführt. Cookies stellten lange Zeit die einzige Möglichkeit dar, Daten im Client abzuspeichern (Camden, 2015, S. 3). Jedoch können Cookies die Anforderungen von modernen webbasierten mobilen Apps kaum bis gar nicht mehr erfüllen. Cookies müssen bei jeder HTTP (Hypertext Transfer Protocol) Anfrage an den Server und dessen Antwort mitgeschickt werden, was dazu führt, dass viele unnötige Daten (Overhead) produziert werden. Außerdem ist die Speichergröße von 4 Kilobyte pro Cookie unzureichend für komplexe webbasierte mobile Anwendungen und darüber hinaus kann ein Cookie nicht gleichzeitig über mehrere Browserfenster hinweg verwendet werden (Laine, 2012, S. 3).

Infolgedessen hat das World Wide Web Consortium (W3C) drei verschiedene APIs veröffentlicht, um Daten dauerhaft im Browser der UserInnen speichern und manipulieren zu können (Laine, 2012, S. 4):

1. Web Storage API
2. Indexed DB
3. Web SQL (veraltet)

Allerdings wird die zuletzt angegebene Web SQL Spezifikation mittlerweile nicht mehr weiter vom W3C unterstützt und gewartet. Das W3C rät vom Einsatz dieser API auf ihrer Webseite ausdrücklich ab (siehe Abbildung 8).

Abbildung 8. Screenshot des W3C Hinweis zur Verwendung von Web SQL, Stand: 07.06.2017; https://www.w3.org/TR/webdatabase/

Laut Camden (2015, S. 63) wäre die Web SQL API jedoch vor allem für WebentwicklerInnen sehr interessant gewesen, da Web SQL eine gewisse Ähnlichkeit zu der Datenbankabfragespache SQL (Structured Query Language) aufweist und dadurch die Möglichkeit bietet, auf eine relationale Datenbank innerhalb des jeweiligen Browser zuzugreifen. Vom Einsatz der Web SQL API, um Daten lokal im Browser des/der Users/Userin zu speichern, ist nach derzeitigem Stand abzuraten.

3.3.1 Web Storage API

Im Internet wird der Begriff Local Storage oft gleichbedeutend mit dem Terminus Web Storage verwendet. Jedoch beinhaltet Web Storage zwei sehr ähnliche Speichervarianten: Local Storage und Session Storage. Beide benutzen zwar die gleiche API, Local Storage speichert die Daten jedoch dauerhaft, wohingegen die Daten bei Session Storage beim Schließen des Browsers unwiderruflich verloren gehen. Nachdem zur Datensicherung zumeist die dauerhafte Version eingesetzt wird, verwenden die meisten WebentwicklerInnen (und reden auch über) Local Storage (Camden, 2015, S. 13).

Mit der Web Storage API ist es möglich einfache Schlüssel-Wert Paare (Key-Value) abzuspeichern und diese bei Bedarf auch wieder abzufragen. Im Gegensatz zu Cookies besteht hier nicht die Möglichkeit, Key-Value Paare über verschiedene Domains oder Subdomains hinweg verfügbar zu machen. Das bedeutet, dass sich Keys mit dem gleichen Namen auf unterschiedlichen Webseiten nicht überschreiben oder beeinflussen können. Der Speicherplatz für Web Storage bewegt sich im Bereich von fünf bis maximal zehn Megabyte. Üblicherweise sollte dies für kleine Mengen an Daten vollkommen ausreichend sein, sollten die Grenzen jedoch trotzdem erreicht werden, werfen die meisten Browser eine Fehlermeldung aus (Camden, 2015, S. 13ff).

Die nachfolgenden beiden Listings 11 und 12 beziehen sich ausschließlich auf die dauerhafte Local Storage Variante.

Listing 11. Beispielaufrufe Web Storage

```
// save key-value
localStorage.setItem('age', 65);

// retrieve key-value and assign to variable age
var age = localStorage.getItem('age');

// remove key-value
localStorage.removeItem('age');

// clear all key-values from storage
localStorage.clear();
```

Die Web Storage API ist sehr einfach aufgebaut und besteht lediglich aus vier Methoden: `setItem()`, `getItem()`, `removeItem()` und `clear()`. Listing 11 auf Seite 39 zeigt, wie `setItem()` dem Schlüssel „age“ der Wert „65“ zuweist. Mit `getItem()` holt man den Wert des Schlüssels „age“ aus dem lokalen Speicher und weist es der Variablen zu. Die Funktion `removeItem()` entfernt das Schlüssel/Wert Paar aus dem Objekt. Bei Bedarf können mit der Methode `clear()` alle Schlüssel/Wert Paare aus dem Storage wieder gelöscht werden (Camden, 2015, S. 14).

Da Web Storage nur einfache Zeichenketten abspeichern kann, müssen komplexere Daten zuerst verschlüsselt und geparst werden, bevor sie angezeigt werden können (Camden, 2015, S. 14) (siehe Listing 12).

Listing 12. Speichern und auslesen von komplexen Daten mit Web Storage

```
// create array
var kindsOfOa = ['finger', 'hip', 'knee', 'shoulder'];

// store and und stringify/encode array
localStorage.setItem('kindsOfOa', JSON.stringify(kindsOfOa));

// parse/decode array
var storedOaKinds = JSON.parse(localStorage.getItem("kindsOfOa"));

// retrieve first value of array
var getSpecialOaKinds = storedOaKinds[0];
```

In Listing 12 wird zuerst ein Array mit dem Namen „kindsOfOa“ erstellt. Dieses Array wird dann mittels `JSON.stringify` in ein JSON Objekt verwandelt und im Web Storage mit dem Schlüssel `kindsOfOa` gespeichert. Anschließend wird mittels der Methode `JSON.parse` das Array ausgelesen und in die Variable `storedOaKinds` übergeben. Danach kann je nach Verwendungszweck mittels einer for-Schleife über die einzelnen Werte des Arrays iteriert und dessen Werte ausgegeben oder gezielt mit Hilfe des Indexes (`storedOAKinds[0]`) auf einen Wert zugegriffen werden.

3.3.2 Indexed Database API

Indexed Database API oder auch IndexedDB kann als kombinierter und verbesserter Mix aus Web Storage und Web SQL gesehen werden (Laine, 2012, S. 8). Im Gegensatz zu Web Storage kann IndexedDB wesentlich größere Datenmengen verwalten und speichern. Bei IndexedDB handelt es sich aber nicht um eine relationale Datenbank wie Web SQL, sondern die Daten werden als JavaScript Objekte gespeichert (Novy, 2015, S. 37).

Beim IndexedDB Konzept werden Daten in einer Datenbank gespeichert. Diese Datenbank stellt die höchste Ebene dar. Es ist theoretisch möglich, unzählige Datenbanken zu erstellen, aber generell sollte für ein Projekt nicht mehr als eine

Datenbank angelegt werden. Darin werden dann die sogenannten „Object Stores“ gespeichert, welche die eigentlichen Daten beinhalten. Object Stores sind vergleichbar mit Tabellen aus relationalen Datenbanken. Je nach Umfang der Anwendung und der gespeicherten Daten können diese in eine oder mehrere Object Stores gespeichert werden. Die Daten in Object Stores haben keine feste Datenstruktur, an die sich halten müssen. IndexedDB ist sehr variabel, was das Speichern von Daten betrifft (Camden, 2015, p. 28).

Ein weiterer wichtiger Bestandteil von IndexedDB sind die „Indizes“. Indizes ist eine mögliche Art, Daten aus den Object Stores zu holen. In machen Fällen möchte man anstatt aller Daten eines Object Stores nur die Daten einer bestimmten Eigenschaft abfragen. Legt man jedoch eine Tabellenspalte/Eigenschaft als Index fest, ist es im späteren Verlauf der Anwendung leichter an die Daten der Eigenschaft zu gelangen (Camden, 2015, S. 28).

In den nachfolgenden drei Listings 13-15 wird der Workflow des IndexedDB Konzeptes, vom Anlegen einer Datenbank bis hin zum Auslesen der Werte, im Detail erklärt.

Erstellen und öffnen einer Datenbank

Listing 13. Erstellen und öffnen einer Datenbank mit IndexedDB

```
// create a database
var openRequest = indexedDB.open("arthrose", 1);

// called function, if database version is 1
openRequest.onupgradeneeded = function(e) {
   var thisDB = e.target.result;
   if(!thisDB.objectStoreNames.contains("users")) {
      var peopleOS = thisDB.createObjectStore("users ",
      { keyPath: "id", autoIncrement: true });
   }
}
// called function if database is loaded
openRequest.onsuccess = function(e) {
   db = e.target.result;
   // call functions insert, add, ...
}
// called function if error occurs
openRequest.onerror = function(e) {
   console.dir(e);
}];
```

Zu Beginn wird mittels der Methode `indexedDB.open` eine neue Datenbank angelegt. Diese Datenbank kann als Top-Level Container der Anwendung angesehen werden, in der sämtliche Daten gespeichert werden. Beim Anlegen der Datenbank muss sowohl ein Name als auch eine Versionsnummer angegeben werden. Die Versionsnummer ist grundsätzlich frei wählbar, sollte jedoch bei 1

beginnen. Der Versionsnummer kommt dabei eine wichtige Rolle zu: Sollte an der Struktur der Datenbank (Object Stores oder Indexes) eine Änderung vorgenommen werden, muss die Versionsnummer geändert werden, ansonsten werden die Änderungen nicht übernommen (Camden, 2015, S. 29).

Da IndexedDB ausschließlich asynchron funktioniert, muss auf die Events gehört werden, die nach dem Öffnen der Datenbank ausgelöst werden: `onupgradeneeded()`, `onsuccess()`, `onerror()` und `onblocked()`. Die Methode `onupgradeneeded()` wird nur dann ausgeführt, wenn die Datenbank das erzeugt wird, oder sich die Versionsnummer aufgrund einer Schemamodifikation ändert. Innerhalb von `onupgradeneeded` wird die Methode `objectStoreNames` aufgerufen, die überprüft, ob der Object Store bereits existiert. Wenn nicht, wird ein neuer Object Store mit `createObjectStore` erstellt. Zusätzlich zum Object Store Namen, können weitere Parameter übergeben werden. In einem Object Store sollte jeder Datensatz eindeutig über einen Primärschlüssel identifizierbar sein. Über die Eigenschaft `keyPath` wird „id“ als Primärschlüssel festgelegt und bei jedem neuen Eintrag in der Datenbank um eins hochgezählt (`autoIncrement: true`). Über den festgelegten Primarschlüssel kann im späteren Verlauf der Anwendung dann gesucht werden (Camden, 2015, S. 34).

Wenn die Datenbank fertig geladen wurde und es keine Fehlermeldung gibt, wird die Methode `onsuccess()` ausgeführt. Dort können dann z.B weitere Methoden aufgerufen werden, um Daten zu speichern oder zu aktualisieren. Mit `onerror()` können Fehlermeldungen abgefangen werden, `onblocked()` hingegen wird verwendet, wenn die Datenbank nicht verfügbar ist oder benutzt werden kann (Camden, 2015, S. 29ff).

Datensätzen einfügen in Object Stores

Listing 14. Datensätze einfügen in Object Stores mit IndexedDB

```
// define transaction
var transaction = db.transaction(["users"]," readwrite ");

// request to object store
var store = transaction.objectStore("users");

// determine data
var user = { name: "testperson1", email: "test@testmail.de" };

// save entry
var request = store.add(user);
```

Alle Vorgänge, bei denen Daten gelesen, geschrieben, aktualisiert oder gelöscht werden, werden via Transaktionen ausgeführt. Diese Transaktionen sind vergleichbar mit dem Transaktionskonzept aus der Datenbanksprache SQL. Wenn bei der Transaktion etwas schieflaufen sollte, werden alle Änderungen rückgängig gemacht. Dadurch wird eine zusätzliche Sicherheitsstufe eingeführt um die

Datenintegrität zu gewährleisten. In Zeile 2 des Listings 14 (siehe Seite 42) wird zuerst eine Transaktion mit dem Namen `users` erstellt, welche sowohl das Lesen als auch das Schreiben auf die Datensätze ermöglicht (`readwrite`). Danach wird via `transaction.objectStore` nach dem erstellten Object Store gefragt. In der letzten Zeile wird dann der zuvor definierte Datensatz mittels `objectStore.add()` hinzugefügt (Camden, 2015, S. 42).

Datensätzen aktualisieren

Das Aktualisieren der Daten funktioniert auf ähnliche Weise wie das Speichern der Daten: die Methode `store.add(user)` muss durch die Methode `store.put(updateUser)` ersetzt werden. Ein Problem das auftreten kann, wenn man `store.put()` verwendet ist, dass diese Methode auch neue Einträge hinzufügen kann. Wenn also ein Datensatz aktualisiert werden muss, ist es notwendig, vorab zu überprüfen, ob dieser Datensatz bereits existiert, oder nicht (Novy, 2015, S. 39).

Datensätzen abrufen

Listing 15. Datensätze abrufen mit IndexedDB

```
// define transaction
var transaction = db.transaction(["users"]," readonly ");

// request to object store
var store = transaction.objectStore("users");

// retrieve entry
var request = store.get();

// request the result
request.onsuccess = function() {
   var user = request.result();
}
```

Auch das Abfragen von Daten in Listing 15 ähnelt sehr dem Speichern und dem Aktualisieren, ist aber wesentlich komplexer als z.B. mit der Web Storage API. Nachdem eine Transaktion definiert wurde, welche nur schreiben kann (`readonly`) und eine Verbindung zum Object Store erstellt wurde, kann via `store.get()` auf die Werte des Datensatzes zugegriffen werden. Wenn man auf andere Indizes zugreifen will, müssen diese vorab definiert werden. Das Ergebnis des Aufrufes kann anschließend über das Event `onsuccess` ausgegeben werden (Novy, 2015, S. 39).

3.3.3 Datenschutz / Sicherheit und Einschränkungen

Beim Abspeichern von Daten im Browser des/der Users/Userin sollten gewisse Bedenken hinsichtlich Datenschutz und Sicherheit in Betracht gezogen werden. Als größte Privatsphärenbedrohung kann das Verfolgen des UserInnen-

Verhaltens gesehen werden. Drittanbieter können ihr browserseitiges Speicherverfahren dafür nutzen, um UserInnen über mehrere Sitzungen hinweg zu verfolgen, um ein Profil zu erstellen und gezielte Werbung zu schalten. Aber auch die Möglichkeit sensible private Daten wie z.B. gesundheitsbezogene Daten im Browser zu speichern, kann das UserInnen-Verhalten beeinträchtigen (Laine, 2012, S. 10).

Alle beschriebenen browserseitigen Speichermethoden verwenden das Origin-Prinzip („same-origin policy“), was bedeutet, dass verschiedene Webseiten (Origin) nicht auf Daten anderer Webseiten zugreifen können. Nichtsdestotrotz sollte den dort abgespeicherten Daten, nicht ohne Überprüfung vertraut werden, denn über browserseitige Werkzeuge (z.B. Chrome’s Developer Tools) können diese Daten von böswilligen NutzerInnen manipuliert werden (Laine, 2012, S. 10).

Abgesehen von den technischen Möglichkeiten, welche lokale Datenspeicherungen bieten, gibt es gewisse Einschränkungen bei der Verwendung vorgestellter Speichervarianten, welche die NutzerInnen selbst betreffen. Heutzutage ist es üblich, dass eine Person mehrere unterschiedliche Endgeräte benutzt (Smartphone, Tablet, Laptop, Computer, etc.). Ruft eine Person eine Webseite, welche Daten lokal im Browser speichert, auf, müssen vorgenommene Einstellungen beim ersten Aufruf der Seite auf einem anderen Gerät neu durchgeführt werden, da es bisher noch keine Möglichkeit gibt, die Daten zu übertragen. Auch die Situation, dass in einem Haushalt mehrere Personen, den gleichen Computer sowie den gleichen Browser verwenden, ist nicht unüblich. Für diese Szenarien bieten die vorgestellten Speichervarianten noch keinen Mechanismus an, wie man zwischen unterschiedlichen BenutzerInnen differenzieren kann (Janik & Kiebzak, 2014, S. 10).

3.3.4 Zusammenfassung / Browsersupport

In Tabelle 3 werden die Vor- und Nachteile der verschiedenen lokalen Speichermöglichkeiten kurz zusammengefasst (Buckler, 2013 und Aderinokun, 2016) und in Abbildung 9 (siehe Seite 46) der Browsersupport von Web Storage und IndexedDB gegenübergestellt.

Tabelle 3. Vor- und Nachteile von lokalen Speicherverfahren

	Cookie	***Web Storage***	***IndexedDB***
+	*- Wird in allen modernen Browsern unterstützt* *- Kann zur Kommunikation mit dem Server verwendet werden* *- Kann ein Ablaufdatum enthalten*	*- Wird in fast allen modernen Browsern unterstützt (siehe Abbildung 9)* *- Zwischen dauerhafter (local) und ablaufbarer (session) Speicherart wählbar* *- Bietet ein intuitives Interface zum Datenspeichern an* *- Sicherer als Cookies* *- Größerer Speicherplatz (~5MB)* *- Event-Model das Daten über verschiedene Tabs und Fenster hinweg synchronisiert*	*- Kann komplexe, strukturierte Daten verarbeiten* *- SQL ähnlicher Aufbau mit Datenbanken und Object Stores (Tabellen)* *- Sicherer als Cookies* *- Speicherplatz kann bis zur Hälfte der Festplatte umfassen* *- Mehr Kontrolle zum Interagieren mit den Daten*
-	*- Nur Strings speicherbar* *- Beschränkter Speicherplatz (~4KB)* *- Zusätzlicher Speicherplatz im Dokument* *- Kann geblockt und gelöscht werden* *- Seit der Einführung von Web Storage wird vom Einsatz abgeraten*	*- Nur Strings speicherbar (Serialisierung kann notwendig sein)* *- Bei großen Datenmengen evtl. schlechte Performance* *- Keine Transaktionen, Indexes oder Suchfunktionen vorhanden*	*- Komplexe und größere API als Web Storage* *- Momentan noch geringerer Support als Web Storage (siehe Abbildung 9)* *- Daten sind unstrukturiert und es kann zu Integritätsproblemen kommen*

Browser Support

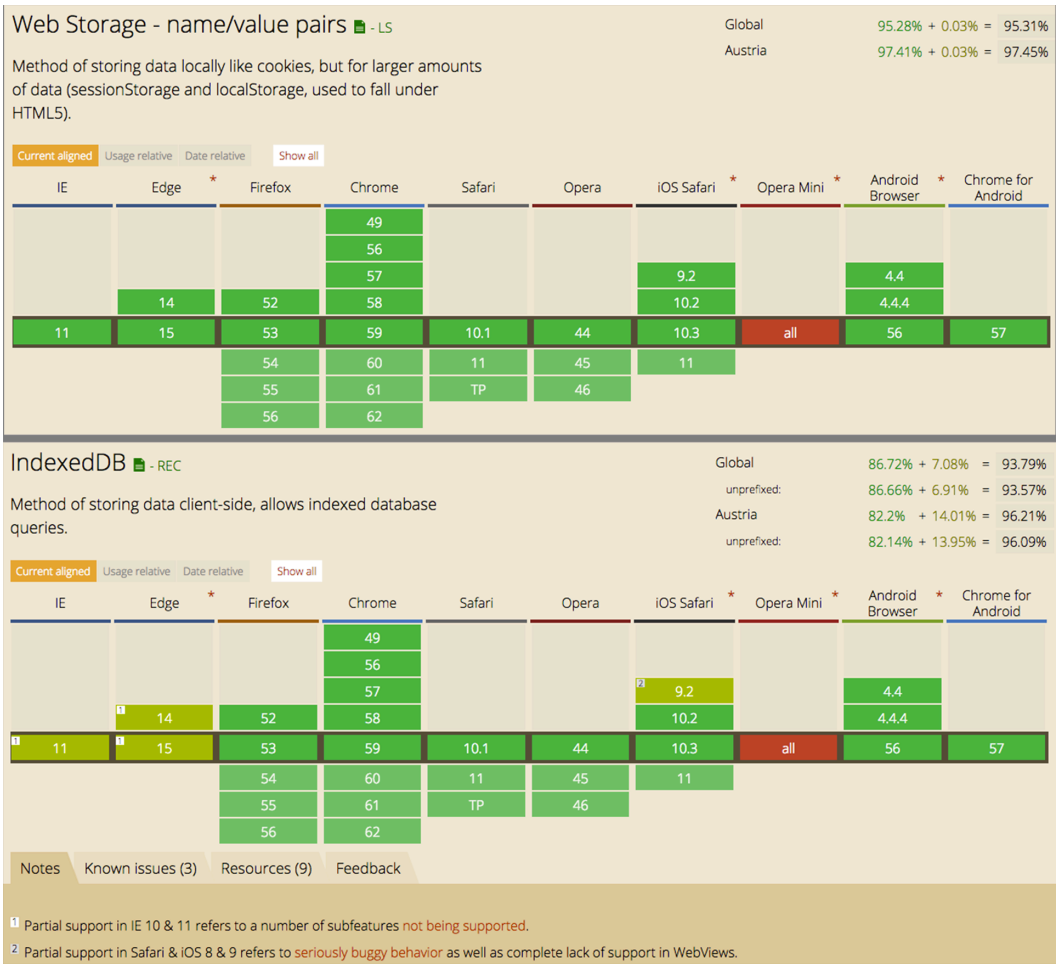

Abbildung 9. Screenshot derzeitige Unterstützung Web Storage (http://caniuse.com/#search=Web%20Storage) und IndexedDB (http://caniuse.com/#search=IndexedDB), Stand: 09.06.2017;

Der globale Browsersupport von Web Storage liegt bei ca. 95%, österreichweit sogar bei 97,5%. Web Storage kann für moderne Webanwendungen also bedenkenlos eingesetzt werden. Mit fast 94% (österreichweit ca. 96%) liegt der Support von IndexedDB nur sehr knapp dahinter. Die letzte Version des Internet Explorers und der neue Edge Browser von Mircosoft unterstützen IndexedDB nur teilweise. Auch beim Einsatz in älteren Versionen von Apple's Betriebssystem iOS 8 & 9 kann es teilweise zu Probleme kommen. Der Einsatz von IndexedDB muss je nach Anforderung an die zu entwickelnde Anwendung vorab geklärt werden.

3.4 Barrierefreiheit im Web

Barrierefreiheit (engl. Accessibility) und Benutzerfreundlichkeit (Usability) spielen bei der Erstellung von Webanwendungen eine wichtige Rolle. Während bei der Benutzerfreundlichkeit, der Nutzen eines Produktes, einer bestimmten Zielgruppe in einem bestimmten Kontext auf eine gewisse Zeit beschränkt ist, ist die Barrierefreiheit als eine Art Erweiterung der Usability zu sehen:

„Barrierefreiheit ist die Gebrauchstauglichkeit eines Produkts, eines Service, einer Umgebung oder eines Gebäudes für Menschen mit größtmöglicher Variation von Fähigkeiten“ (Zimmermann, 2005).

Grundsätzlich gibt es zwischen Benutzerfreundlichkeit und Barrierefreiheit einige Überschneidungen. Jedoch sorgt Usability im Wesentlichen dafür, dass eine Anwendung benutzerfreundlich, effizient und effektiv bedient werden kann, während Barrierefreiheit noch einen Schritt weiter geht, indem es eine Anwendung für alle möglichen Benutzer bedienbar macht, mit oder ohne körperliche Beeinträchtigung sowie ohne Zeiteinschränkung (Zimmermann, 2005).

Aufgrund der physischen Einschränkungen der Zielgruppe, soll auch die geplante Plattform möglichst barrierefrei gestaltet und umgesetzt werden, damit diese z.B. bei Bedarf rein mit Tastatur bedienbar ist. Daher werden nachfolgend die Grundlagen für barrierefreie Webseiten im Web genauer erläutert.

3.4.1 WCAG Richtlinien

Die W3C Gruppe, welche sich u.a. auch für die Spezifikationen von HTML, CSS etc. verantwortlich zeigt, hat auch die Richtlinien für Barrierefreiheit im Internet entworfen. Diese von der Web Accessibiltiy Initiative (WAI) entwickelten Web Content Accessibility Guidelines (WCAG) dienen heute als Standard für die Erstellung barrierefreier Webanwendungen (“WebAIM: Web Content Accessibility Guidelines,” 2013).

Die primär veröffentlichte Version der WCAG (1999) war der erste große Schritt das Internet auch für Menschen mit Einschränkungen zugänglich zu machen. Die finalisierte Version 1.0 enthielt 14 Richtlinien und zahlreiche Kontrollpunkte, mit denen die Barrierefreiheit einer Webseite ermittelt werden konnte. Diese Version konzentrierte sich stark auf die Techniken zur Erreichbarkeit von Barrierefreiheit, vor allem in Bezug auf HTML. Da sich Webtechnologien und die Technik der verwendeten Geräte für Menschen mit Einschränkungen weiterentwickelten, wurde es zunehmend schwieriger, mit den gegebenen Richtlinien die Barrierefreiheit der Anwendungen zu messen. Aufgrund dessen wurden die Richtlinien erneuert und erweitert (“WebAIM: Web Content Accessibility Guidelines,” 2013).

WCAG 2.0 Standard

Im Unterschied zur Version 1.0 sind die aktuellen Standards nicht mehr technikzentriert, stattdessen sind diese stärker auf unterschiedliche Prinzipien fokussiert. So sind die Richtlinien unabhängig vom Technologiewandel und zusätzlich so konzipiert, dass eine zuverlässige Konformität der Barrierefreiheit ermittelt werden kann. Um das zu erreichen, sind die Richtlinien so strukturiert, dass diese nur einen sehr geringen Interpretationsspielraum zulassen, um wirkliche Barrierefreiheit festzustellen ("WebAIM: Web Content Accessibility Guidelines," 2013).

Die WCAG 2.0 sind in vier Hauptprinzipen aufgeteilt, welche zusammen das Akronym POUR bilden (Horton & Quesenbery, 2014, S. 6):

Perceivable (P) / Wahrnehmbar

Alle User Interface (UI) Komponenten und Informationen müssen für BenutzerInnen in einer verständlichen Art und Weise präsentiert werden, damit diese sie auch sehen und hören können.

Operable (O) / Navigierbar oder bedienbar

Die Navigation sowie alle UI Komponenten müssen so entworfen werden, dass BenutzerInnen mit diesen interagieren können und diese von technischen Hilfsmitteln wie z.B. einem Screenreader (Software zur Ausgabe der Bildschirmanzeige) unterstützt werden.

Understandable (U) / Nachvollziehbar

Informationen und Interaktionen mit dem UI müssen konsistent und klar kommuniziert werden, sodass der Inhalt für UserInnen verständlich ist.

Robust (R) / Stabil

Der Inhalt muss so geschrieben sein, dass dieser zuverlässig von einer Vielzahl an Technologien inklusive assistiver Technologie interpretiert werden kann (z.B. Browser, Media Players, Plug-Ins, etc.) (Horton et al, 2014, S. 6).

Diese vier Prinzipien enthalten 12 Richtlinien. Die Erfolgskriterien (engl. „success criteria") für jede Richtlinie sind in drei Konformitätslevel eingeteilt (A, AA, AAA). Diese Konformitätslevel geben Auskunft darüber, in wie weit die geprüften Elemente das Erfolgskriterium erfüllen (Groves, 2013):

Level A

Erfolgskriterien die diesem Level entsprechen, haben einen hohen Einfluss auf einen Großteil der UserInnen und können daher als Grundvoraussetzung gesehen werden. Diese Kriterien haben zudem nur einen sehr geringen Einfluss auf das Erscheinungsbild einer Website und die Umsetzung dieser Anforderungen ist in der Regel leicht zu erreichen.

Level AA

Auch diese Erfolgskriterien haben einen hohen Einfluss auf UserInnen, aber im Gegensatz zu Level A sind hier manchmal nur spezifische NutzerInnen davon betroffen. Diese Erfolgskriterien sind so wichtig, dass die Einhaltung ebenjener sogar Änderungen an dem Erscheinungsbild einer Website fordern kann.

Level AAA

Jene Kriterien zielen oftmals auf Verbesserungen einer bestimmten UserInnen Gruppe ab. Die Umsetzung zur Einhaltung dieser Kriterien kann je nach Webseite schwierig oder sogar teuer werden. Zielgruppenabhängig sollte vorab kalkuliert werden, ob es rentabel ist, dieses Level zu erreichen (Groves, 2013).

Anhand des nachfolgenden Tabelle 4 soll der Aufbau der WCAG Richtlinien genauer erklärt werden.

Tabelle 4. Auszug aus den WCAG Standard von 2009; *("Richtlinien für barrierefreie Webinhalte (WCAG) 2.0 (Web Content Accessibility Guidelines (WCAG) 2.0)," 2009)*

Richtlinie 2.4 Navigierbar: Stellen Sie Mittel zur Verfügung, um Benutzer dabei zu unterstützen zu navigieren, Inhalte zu finden und zu bestimmen, wo sie sich befinden	
***2.4.1** Blöcke umgehen*	*Es gibt einen Mechanismus, um Inhaltsblöcke zu umgehen, die auf verschiedenen Webseiten wiederholt werden. (Stufe A)*
...	*...*
***2.4.8** Position*	*Es gibt Informationen zu der Position des Benutzers innerhalb eines Satzes von Webseiten. (Stufe AAA)*

Tabelle 4 zeigt eine Richtline aus dem zweiten Prinzip „navigierbar". Unter dieser Richtlinie werden alle Punkte gelistet, welche BenutzerInnen dabei helfen sollen, durch die Seite zu navigieren, sowie Inhalte leichter auffindbar machen, zu finden oder zu erkennen, wo sich der/die NutzerIn gerade befindet. Das erste Erfolgskriterium „Blöcke umgehen" dieser Richtline, fordert von der Webseite, dass es einen Mechanismus geben muss, um Inhaltsblöcke zu übergehen. Menschen die z.B. zum Navigieren einer Webseite auf eine Tastatur oder Screen Reader angewiesen sind, soll die Möglichkeit gegeben werden, das Menü mittels Sprungmarke zu übergehen und sind somit in der Lage, direkt zum Inhalt der Seite zu kommen. Die Richtlinie 2.4.1 ist der Stufe A zugeordnet und in der Regel leicht ohne große Aufwände realisierbar.

Als zweites Beispiel in dieser Richtlinie findet sich in der Tabelle (siehe Seite 49) das Kriterium 2.4.8 „Position“. Damit ist gemeint, dass es für den/die BenutzerIn jederzeit nachvollziehbar sein muss, wo er/sie sich gerade auf einer Webseite befindet. Dies wird zumeist mit einem Breadcrumb oder einer Sitemap auf der Seite realisiert. Die Realisierung und der damit verbundene Aufwand zur Einhaltung dieser Richtlinie ist wesentlich umfangreicher und wird daher der höchsten Stufe AAA zugeordnet.

3.4.2 WCAG 2.1

Mitte April 2017 hat das W3C einen neuen ersten Entwurf der WCAG Richtlinien (Version 2.1)[7] veröffentlicht. Dieser Entwurf ist die erste große Aktualisierung der aktuell gültigen Version 2.0 von 2009. Folgerichtig, betrachtet die Aktualisierung die neuen und unterschiedlichen Möglichkeiten wie UserInnen die vorhandenen Technologien nutzen. Die erste WCAG 2.1 Version ist eine Erweiterung der bestehenden Richtlinien, welche 28 neue Erfolgskriterien bereithält, die den Schwerpunkt auf folgende drei Bereiche legen (“WCAG 2.1 — It’s here!,” 2017):

- UserInnen mit geringem Sehvermögen
- UserInnen mit Wahrnehmungs- oder Lerneinschränkungen
- Barrierefreiheit auf mobilen Geräten

Da es sich derzeit mit WCAG 2.1 um einen vorläufigen Entwurf handelt, der sich im Laufe der nächsten Monate noch mehrmals ändern kann, werden für die Erstellung der Arthrose Plattform die aktuell gültigen WCAG 2.0 Richtlinien herangezogen.

3.4.3 WAI-ARIA

Die von der WAI entwickelte „Accessible Rich Internet Applications“ (ARIA) Spezifikation kann als Erweiterung des HTML5 Standards gesehen werden. (Max, 2014). Diese Erweiterung definiert eine Möglichkeit, den Inhalt von komplexen Webanwendungen auch für UserInnen mit Einschränkungen zugänglich zu machen. WAI-ARIA ermöglicht es, durch ein Set an zusätzlichen Eigenschaften, welche in den HTML Code eingefügt werden, den BenutzerInnen eine Interaktion mit dynamischen Features oder Funktionen anzubieten (Wilson, 2015). Diese zusätzlichen Attribute können in drei Hauptkategorien eingeteilt werden (“WAI-ARIA basics,” 2017):

[7] https://www.w3.org/TR/WCAG21/

Landmark roles (Rollen)

Rollen beschreiben ein HTML UI Element sowie deren Zweck. Manche dieser Rollen duplizieren den semantischen Wert eines HTML Elements, wohingegen andere verschiedene Seitenstrukturen beschreiben, welche oft in UI's vorkommen ("WAI-ARIA basics," 2017). Das Listing 16 verdeutlicht den möglichen Einsatz von Rollen.

Listing 16. Einsatz von Rollen in HTML (Max, 2014)

```
<nav>
   <ul role="tablist">
      <li><a href="#home" role="tab">Home</a></li>
      <li><a href="#about" role="tab">About</a></li>
   </ul>
</nav>

// content
<div id="tabContent>
   <div id="home" role="tabpanel">Content Home</div>
   <div id="about" role="tabpanel">Content About</div>
</div>
```

Mit den ARIA Rollen ist es möglich, zumeist bedeutungslosen HTML Elementen für assistive Technologien zu kennzeichnen. In diesem Beispiel werden einer ungeordneten Liste `<ul>` die Rolle „tablist", dem Listenelement `<li>` die Rolle „tab" und dem Container (`<div>`), der den Inhalt enthält, die Rolle „tabpanel" zugeordnet. Damit werden den unterschiedlichen HTML Elementen ihre Rolle im Dokument zugewiesen. Diese Rollen überschreiben den üblichen Element Typ, womit z.B. die ungeordnete Liste nun zu einer Tabliste wird (Max, 2014).

Properties (Eigenschaften) und States (Zustände)

Eigenschaften und Zustände bieten den UserInnen zusätzliche Informationen darüber, wie sie mit den speziellen UI Elementen interagieren können. UserInnen, welche Tastatur oder Screen Reader verwenden, werden somit die Beziehungen und Zustände der Anwendung aufgezeigt. Am erweiterten Listing 17 Beispiel werden diese Zusammenhänge deutlich (Max, 2014).

Listing 17. Einsatz von Properties und States in HTML (Max, 2014)

```
<nav>
   <ul role="tablist">
      <li class="active">
         <a id="tab1" href="#home" role="tab" aria-controls="home">Home
</a>
      </li>
      ...
</nav>

// Inhalt
<div id="tabContent>
      <div id="home" role="tabpanel" aria-labelledby="tab1">
```

```
        Content Tab 1
      </div>
      <div id="about" role="tabpanel" aria-labelledby="tab1"
        aria-hidden="true">
        Content Tab 2
      </div>
</div>
```

Die Verbindung zwischen der Tabliste und dem jeweiligen Tabinhalt wurde bisher mit JavaScript gelöst. Jedoch besteht somit keine dazugehörende Beziehung zueinander. Das `aria-controls` Attribute verbindet das Kontrollelement „home“ mit der Region das es kontrolliert (Verbindung in orange). Die Eigenschaft `aria-labelledby` funktioniert ähnlich dem HTML Tag `<label>` und wird verwendet, um das Element anzugeben, das als Kennzeichnung (Label) für das Element agiert (Verbindung in grün). `aria-controls` und `aria-labelledby` sind Beispiel für ARIA Eigenschaften und sind mit dem jeweiligen `id`-Attribute miteinander verbunden (Max, 2014).

Da nur ein Tab ausgewählt sein kann und somit nur der Inhalt eines Panels sichtbar ist, müssen die anderen Panels währenddessen ausgeblendet werden. Um dies zu erreichen, kann man den `aria-hidden` Zustand verwenden. Da es sich bei `aria-hidden` um ein boolsches Attribute handelt, akzeptiert `aria-hidden` nur zwei Werte: `true` oder `false`. „True“ bedeuet, dass das jeweilige Panel sichtbar ist und durch „false“ hingegen, wird das Panel ausgeblendet (Max, 2014).

Einsatz und Regeln

Beim Einsatz von ARIA Elementen in HTML Code sollte man grundsätzlich folgende Punkte beachten (Luhur, 2016):

Wenn möglich semantische HTML Elemente benutzen

Semantisch bedeutet, dass alle HTML Tags, Klassen und Identifikatoren „entsprechend ihrer Bedeutung eingesetzt und nicht entsprechend ihrer visuellen Präsentation beschrieben werden“ (Hoffmann, 2012, S. 49). Folgender Vergleich zeigt den Einsatz von HTML Elementen vor und nach der Einführung von HTML5 (Luhur, 2016) (siehe Listing 18).

Listing 18. HTML Elemente vor und nach der Einführung von HTML5

```
// navigation before HTML5
<div class="navigation" role="navigation"> ... </div>

// navigation after introduction of HTML5
<nav> ... </nav>

// button before HTML5
<div class="button" role="button"> ... </div>
```

```
// button after introduction of HTML5
<button> // Content </button>
```

Bevor HTML5 eingeführt wurde, war es gebräuchlich, Elemente mit Klassen, Identifikatoren oder Rollen zu beschreiben (siehe Zeile 2 und 8 in Listing 18). Anstelle des Tags verwenden WebentwicklerInnen heutzutage semantische Elemente wie z.B. <nav> oder <button> (Zeile 5 und 10). Die ARIA Attribute role="navigation" und role="button" werden nicht länger benötigt, da „nav", „button" und diverse andere HTML5 Tags bereits implizierte ARIA Anweisungen enthalten. Unter folgender Liste[8] kann überprüft werden, ob für ein HTML Element eine ARIA Eigenschaft notwendig ist (Luhur, 2016).

Ein HTML Element kann nur eine Rolle haben

Ein HTML Element kann nicht mehr als eine ARIA Rolle enthalten. Aus semantischen Gründen kann und darf ein Element nicht zwei verschiedene Rollen gleichzeitig haben. Daher sollte die Rolle gewählt werden, welche die Funktion des Elementes am besten beschreibt (Luhur, 2016).

Keine Änderung der nativen Semantik

Desweitern sollte einem semantischen HTML Element keine Rolle zugeordnet werden, welcher der implizierten Semantik widerspricht (siehe Listing 19).

Listing 19. Falsch/Richtig Beispiel für den Einsatz einer ARIA Rolle (Luhur, 2016)

```
// WRONG
<footer role="navigation">
  //Content
</footer>

// RIGHT
<footer>
  <nav> //Content </nav>
</footer>
```

Mit dem Hinzufügen des ARIA Attributs role="navigation" überschreibt die Eigenschaft die native Semantik des Elements <footer>. Assistive Technologien würden den Footer nun als Navigation interpretieren. Stattdessen wäre es sinnvoller und semantisch korrekt, die HTML Struktur ineinander zu verschachteln in dem man im Footer ein Navigationselement <nav> einfügt (Luhur, 2016).

[8] https://www.w3.org/TR/html-aria/#docconformance

4 Technologiebewertung

Bei der Konzeptüberprüfung Mitte Juni 2017 im AKH Wien wurden der Zielgruppe unterschiedliche Features und ein ausgearbeitetes Konzept für eine zukünftige Arthrose Plattform vorgestellt. Diese Features wurden von der Zielgruppe anschließend mit Hilfe eines Fragebogens bewertet und gereiht (El Aeraky, 2017, S. 79ff). Aufgrund der Anforderungen der Zielgruppe an die Arthrose Plattform wurden Nutzungsszenarien erstellt und daraus technische Anforderungen, welche die zukünftige Plattform erfüllen sollte, abgeleitet. Diese sind in Kapitel 4.1 ersichtlich. Im darauffolgenden Kapitel 4.2 werden die, für die Entwicklung der Plattform möglichen, Frameworks und Libraries (Unterscheidung und Definition in Kapitel 3.2.1) anhand von vorab festgelegten Auswahlkriterien gegenübergestellt und bewertet. Diese Gegenüberstellung soll aufzeigen, warum welche Frameworks und Libraries für die Umsetzung der Plattform herangezogen wurden.

4.1 Use Cases und Definition der technischen Anforderungen

Die nachfolgenden Use Cases sowie deren technischen Anforderungen (TA) wurden von den Bedürfnissen der Zielgruppe abgeleitet (El Aeraky, 2017, S. 59ff).

Use Case 1: **„UserInnen wollen die Anwendung bevorzugt am Desktop verwenden."**

TA: Die Anwendung muss besonders gut auf großen Bildschirmen verwendbar sein. Zusätzlich wird die Anwendung auch für mobile Endgeräten/Viewports angepasst damit die Plattform auch für kommende Generationen, welche die Plattform eventuell auch mobil nutzen, verwendbar ist.

Use Case 2: „UserInnen wollen wichtige Informationen der Arthrose Plattform ausdrucken."

TA: Die Inhalte der Anwendung sollen für den Ausdruck via CSS Stylesheet optimiert sein und somit einen Zusatznutzen für UserInnen anbieten.

Use Case 3: **„UserInnen wollen eine effiziente Nutzung der Plattform, damit sie schnell an ihre gewünschten Informationen gelangen."**

TA: Die Anwendung muss schnell geladen werden können, damit die Wartezeit der UserInnen geringgehalten wird. Die bei Bedarf eingesetzten Technologien/Frameworks sollten eine geringe Größe haben und bei Bedarf mit

Funktionen erweitert werden können. Auch sollten die Inhalte je nach Bedarf geladen und somit eine Verkürzung der Ladezeit erreicht werden.

Use Case 4: „**UserInnen verwenden bei der Nutzung der Plattform vorwiegend ältere Browser(versionen)".**

TA: Beim Einsatz von neuen Technologien/Frameworks ist darauf zu achten, dass diese abwärtskompatibel sind und von allen Browsern unterstützt werden, oder ob ggf. Polyfills9 eingesetzt werden müssen, um die Funktionalität zu gewährleisten.

Use Case 5: **„Aufgrund der körperlichen Einschränkungen der Zielgruppe in den Fingern bzw. Händen sollte die Anwendung bei Bedarf auch ohne Maus oder mit technischen Hilfsmitteln wie Screenreader bedienbar/verwendbar sein".**

TA: Zur besseren Unterstützung der UserInnen sollten möglichst alle Funktionen der Anwendung mit Tastatur verwendet werden können, um für eine Entlastung der betroffenen Gelenke zu sorgen. Ausgenommen von dieser Funktion sind eingesetzte externe Dienste von Drittanbietern.

Use Case 6: **„Aufgrund der Angabe der PLZ, des Wohnortes und der gesuchten Fachrichtung (Einrichtung, ExpertIn) werden UserInnen die in der Umgebung befindlichen ExpertInnen oder Einrichtungen angezeigt".**

TA: Die Anwendung sollte auf den Google Places Service zugreifen, um Einrichtungen und ExpertInnen in der Umgebung der UserInnen anzuzeigen und zusätzliche Daten wie z.B. Adresse oder Telefonnummer zur Verfügung zu stellen.

Use Case 7: **„Um Inhalte wie Übungen/Videos für UserInnen anzuzeigen, werden Inhalte von Drittanbietern bezogen. Diese Videos können bei Bedarf sortiert und bildschirmfüllend angezeigt werden".**

TA: Es sollte die Möglichkeit bestehen, Übungen/Videos über eine API von einem Drittanbieter wie z.B. Vimeo oder YouTube zu laden, den Inhalt auf der Plattform zu integrieren und wichtige Videos zu merken. Zudem sollen die Videos nach Auswahl in einer „Lightbox" geöffnet und der Hintergrund abgedunkelt werden.

Use Case 8: **„UserInnen wollen sich nur sehr ungerne in bestehende Systeme einloggen und nur wenige Daten preisgeben."**

TA: Die Anwendung sollte die Möglichkeit der lokalen Personalisierung bieten. Der/die UserIn kann hierbei die Anwendung bei Bedarf an seine/ihre Wünsche anpassen, das bedeutet, dass Neuigkeiten, Übungen und Videos je nach Einstellung angezeigt werden.

---

9 Einführung in Polyfills: https://www.sitepoint.com/html5-cross-browser-polyfills/

Use Case 9: **„Einige Inhalte der Plattform müssen über Volltextsuche erreichbar sein“.**

TA: Die betreffenden Seiten sollten eine Suchfunktion anbieten, mit der die jeweiligen Inhalte durchsucht werden können.

Use Case 10: **„UserInnen wollen die Plattform auch offline, ohne Internetverbindung nutzen“.**

TA: Es sollten entsprechend Technologien/Frameworks verwendet werden, die die Offlinefähigkeit der Anwendung gewährleisten.

Use Case 11: **„UserInnen werden Veranstaltungen je nach gewähltem Bundesland und Zeitraum angezeigt“.**

TA: Ein Setzen von Filtern für das Bundesland oder das Datum sollte die Selektion von Veranstaltungen und die Anpassung auf die Wünsche der UserInnen ermöglichen.

Use Case 12: **„Eine Seitennavigation (oder Seitenmenü) erleichtert UserInnen das Navigieren durch lange Inhaltsseiten“.**

TA: Durch das Setzen von Sprungmarken sollen UserInnen zum jeweiligen gewünschten Bereich springen und durch das Platzieren einer fixierten Seitennavigation zu anderen Themenbereichen navigieren können. Zusätzlich soll durch einen „Nach oben“ Button wieder zum Anfang der Seite gesprungen werden können.

Des Weiteren wurden folgene technischen Anforderungen im Bereich der Barrierefreiheit festgelegt, welche auf Grundlage der in Kapitel 4.4.1 beschriebenen WCAG 2.0 Richtlinien erarbeitet wurden:

- Alle Bilder und Icons sollen durch ein Attribut beschrieben werden (bei Bildern: ALT-Attribut); Ausnahmen sind deklarative Bilder und Icons, welche nur der visuellen Formatierung dienen.
- Pflichtfelder in Formularen sollten ein „required“ Merkmal enthalten, damit UserInnen sofort auf mögliche Fehlerquellen hingewiesen werden.
- Um UserInnen das Lesen zu erleichtern, sollte der Farbkontrast von Text und Hintergrund ein hohes Verhältnis haben.
- Texte sollen ohne assistierende Technik um bis 200 Prozent vergrößert werden können, ohne dass dabei Inhalt oder Funktionalität verloren gehen.
- UserInnen sollten redundante Blöcke (z.B. Navigation) überspringen können, um direkt zum Inhaltsbereich der Seite oder zur Personalisierung zu gelangen.

- Eine Sitemap soll UserInnen eine komplette Übersicht über die logische Struktur der Anwendung bieten und als zusätzliche Methode der Navigation der Plattform dienen.
- UserInnen, die sich mit der Tastatur durch die Anwendung bewegen, sollten anhand des Tastaturfokus wissen, auf welchem Element sie sich gerade befinden. Daher ist es wichtig, dass sich dieser prägnant von den anderen Elementen hervorhebt.
- UserInnen sollten innerhalb einer Webseite jederzeit wissen, wo sie sich gerade befinden. Durch einen Breadcrumb finden UserInnen ihre aktuelle Position auf der Webseite wieder.

4.2 Auswahl der Entwicklungstechnologien

Bei der Entwicklung von mobilen Web Apps stehen WebentwicklerInnen unzählige Frameworks und Libraries zur Verfügung. Diese geben EntwicklerInnen ein Grundgerüst vor und sollen dabei helfen, dass man Anwendungen nicht von Grund auf neu programmieren muss. Die große Auswahl an Frameworks und Libraries macht es für EntwicklerInnen immer schwieriger, die richtige Wahl für ein Projekt zu treffen.

Um für die Erstellung des Prototyps der Arthrose Plattform die unterstützenden CSS und JavaScript Technologien zu finden, wurden vor der Realisierung der Plattform Kriterien festgelegt, wonach die ausgewählten Frameworks und Libraries bewertet wurden. Dabei ist es wichtig anzumerken, dass sich diese Kriterien auf die Entwicklung eines Prototyps beziehen und sich von einer finalen Umsetzung unterscheiden können. Im Folgenden Kapitel 4.2.1 werden die Kriterien und deren Überprüfung genauer erläutert.

4.2.1 Festlegung und Überprüfung der Auswahlkriterien

Die Kriterien für CSS/JavaScript Frameworks und Libraries wurden nach ausgiebiger Recherche selbst vom Autor dieser Arbeit festgelegt. Die Faktoren zur Beurteilung sind für beide Sprachen überwiegend gleich und unterscheiden sich nur in einzelnen Punkten. Bevor jedoch auf die Kritieren und deren Überprüfung genauer eingegangen wird, folgt in den zwei Tabellen 5 und 6 (Seite 58) die Erklärung von Erfüllung und Gewichtung eines Kriteriums.

Tabelle 5. Legende für die Erfüllung eines Kriteriums

Sehr gut erfüllt	*1*
Gut erfüllt	*2*
Ausreichend erfüllt	*3*
Kaum erfüllt	*4*

Wurde das Kriterium von dem Framework oder der Library komplett oder fast annähernd erfüllt, erhielt dieses Kriterium den Wert 1 („sehr gut erfüllt“). Gab es bei der Erfüllung einige Defizite wurde das Kriterium mit einer 2 („gut erfüllt“) bewertet. Wurde das Kriterium gerade noch angemessen erfüllt, erhielt es den Wert 3 („ausreichend erfüllt“). Konnte das Kriterum jedoch kaum oder eventuell sogar gar nicht eingehalten werden, wurde es mit dem schlechtesten Wert 4 („kaum erfüllt) evaluiert.

Tabelle 6. Legende für die Wichtigkeit (Gewichtung) eines Kriteriums

Sehr wichtig	*3*
Mäßig wichtig	*2*
Kaum wichtig	*1*

Auch wurden die unterschiedlichen Kriterien nach Bedeutsamkeit gewichtet. Wurde das Kriterium z.B. mit dem Wert 3 ausgezeichnet, bedeutet dies, dass das Kriterium „sehr wichtig“ ist und ihm ein hoher Einfluss bei der Wahl der Technologie zukommt. Der Wert 2 wurde vergeben, wenn das Kriterium „mäßig wichtig“ und somit nicht von absoluter Relevanz ist. Erhielt das Kriterum den Wert 1 („kaum wichtig“), spielt es bei der Entscheidung eine untergeordnete Rolle.

Da es sich bei der Arthrose Plattform um einen reinen Prototyp handelt, der lediglich zu Testzwecken entwickelt wurde, kann die Gewichtung verschiedener Kriterien, im Gegensatz zu einer echten Anwendung, stark differieren. Das Kriterium „High Performance und Speed“, welches für JavaScript Sprachen herangezogen wurde, wurde mit der Gewichtung 1 beurteilt. Die Geschwindigkeit und Ladezeit einer Plattform hat bei einem Test eine untergeordnete Funktion, hingegen kommt diesem Kriterium bei einer echten Anwendung eine entscheidende Rolle hinzu. Anwendungen, dessen Ladezeiten die Geduld von UserInnen strapazieren, haben keine großen Chancen, sich dauerhaft auf dem Markt zu halten.

Kriterien mit Gewichtung und deren Überprüfung

Die folgende Auflistung der Kriterien dient der Übersicht, wie diese überprüft wurden. Zunächst werden die gemeinsamen Kriterien gelistet, gefolgt von den

Kriterien für CSS und JavaScript Frameworks/Libraries. Die Gewichtung des Kriteriums steht dahinter in Klammer.

Schnelles Prototyping und Coding (3)

Es sollte hinterfragt werden, ob der Einsatz eines Frameworks oder einer Library WebentwicklerInnen bei der schnellen und effizienten Umsetzung eines Prototyps hilft. Ebenso stellt sich die Frage, ob die Technologie eine einheitliche Syntax, hohe Flexibilität, leichte Anpassbarkeit sowie Unterstützung moderner Webstandards und hohen Browsersupport anbietet. Erfüllt ein Framework/Library einen Großteil dieser Kriterien, kann sich ein Einsatz positiv auf die Entwicklungszeit und das Prototyping auswirken. Um die Frameworks und Libraries gegenüberzustellen, wurde der Vergleich von Buckler (2017) und der Artikel über die „Top 10 Front-End Frameworks 2016“[10] herangezogen.

Angemessene Lernkurve (2)

Es macht wenig Sinn, zuerst ein Framework zu lernen, bevor man nicht die Sprache hinter dem Framework versteht. Es wäre z.B. nicht zielgerichtet, zuerst AngularJS zu lernen, ohne sich vorab intensiv mit JavaScript auseinandergesetzt zu haben. Oft ist die Zeit für die Entwicklung eines Projektes stark begrenzt. Will man daher ein neues Framework oder Library verwenden, ist eine angemessene Lernkurve wichtig, damit WebentwicklerInnen nicht von einem möglichen Einsatz abgehalten werden. Ein Vergleich von Lernkurven kann unter folgenden Links nachgeschalgen werden[11].

Kleiner Datenkern / keine Codeaufblähung (2)

Von allen verglichenen Technologien wurde eine komprimierte Version heruntergeladen, um die Größe der reduzierten Versionen in Kilobyte gegenüberzustellen (siehe Anhang A). Zusätzlich wurde darauf geachtet, dass durch Verwendung der Technologie der Code nicht übermäßig aufgebläht und verschachelt wird.

Community und Support (2)

Um die Community und den Support messen zu können, wurden auf der Plattform stackoverflow.com die Beiträge zu den jeweiligen Frameworks/Libraries gezählt und diese miteinander verglichen (siehe Anhang A). Stackoverflow bietet EntwicklerInnen die Möglichkeit, Probleme und Fragen zu unterschiedlichen Themen zu stellen. Für den Vergleich wurde in der Suche „CSS

[10] https://www.keycdn.com/blog/front-end-frameworks

[11] Lernkurven Vergleiche: https://academind.com/articles/javascript/angular-vs-reactjs-vs-vuejs/#comparing-ease-of-learning und https://hackernoon.com/reactjs-vs-angular-comparison-which-is-better-805c0b8091b1

<Framework/Library>“ sowie „JS <Framework/Library>“ eingegeben. Je mehr Treffer die Anfrage ergab, desto größer ist die Community dahinter und desto größer ist die Wahrscheinlichkeit, dass ein aufgetretenes Problem schon einmal behandelt wurde. Findet man zudem viele Tutorials und Videos in den Sprachen, ist dies ein weiteres Anzeichen dafür, dass eine Technologie eine hohe Onlinecommunity und Support hat.

Gute und ausführliche Dokumentation (2)

Zum Vergleich dieses Kriteriums wurden die verschiedenen Dokumentationen der Sprachen auf der jeweiligen Plattform betrachtet. Es wurde hinterfragt, ob die die Dokumentation verständlich und übersichtlich aufgebaut ist und ob Codebeispiele den möglichen Nutzen und Einsatz aufzeigen. Da es keine eindeutig messbaren Faktoren zur Bestimmung einer guten und ausführlichen Dokumentation gibt, ist zu beachten, dass die Ergebnisse dieses Kriteriums subjektiv sind.

Einsatz von Preprozessoren (3) – CSS

Es sollte hinterfragt werden, ob das CSS Framework seinen Sourcecode mit einem oder mehreren Preprozessoren anbietet und ob dadurch die Vorteile von Preprozessoren wie LESS oder SASS verwendet werden können, um den Workflow mit CSS zu beschleunigen. Eine Übersicht, welches CSS Framework welchen Preprozessor verwendet und für welche Browser das Framework optimiert ist, kann Anhand dieses Vergleichs nachgeschlagen werden[12].

Nutzung zusätzlicher JavaScript Funktionen (1) – CSS

Um kleine Interaktionen wie z.B. Tooltips oder Pop-Ups auszuführen, bietet das CSS Framework zusätzlich JavaScript Funktionen an, die bei Bedarf eingebunden werden können.

High Performance und Speed (1) – JS

Krause (2016) vergleicht in seinem Blog die Performance einiger beliebter JavaScript Frameworks/Libraries. Dazu hat Krause ein eigenes Skript geschrieben, das die Zeit vom Start des Klickevents bis zum Ende des Ladevorgangs misst. Dabei wird der Test zwölf Mal wiederholt und der Durchschnitt daraus berechnet. Die Ergebnisse des Vergleichs[13] wurden für die Bewertung dieses Kriteriums herangezogen.

[12] CSS Framework Vergleich: http://usablica.github.io/front-end-frameworks/compare.html

[13] JS Performance Vergleich: http://www.stefankrause.net/wp/?p=431

Zukunftsfähigkeit und Lebensdauer (2) – JS

Zum Vergleich dieses Kriteriums wurde das Erscheinungsdatum des Frameworks/Library herangezogen (siehe Anhang A). Wichtig ist die Überlegung ob eine Sprache schon länger existiert oder ob die neueste Version erst vor kurzem veröffentlicht wurde. Je länger eine Sprache im produktiven Einsatz ist, desto besser wurde das Kriterium erfüllt. Aber auch die Zukunftsfähigkeit spielt eine wichtige Rolle bei der Wahl der Sprache. Hier ist besonders das Framework AngularJS zu erwähnen. Dieses hat unter den Sprachen zwar die zweitlängste Lebensdauer, jedoch könnte es sein, dass dieses Framework, mit der Veröffentlichung von Angular 2 (aktuell Angular 4), nicht mehr aktiv weiterentwickelt und irgendwann die Unterstützung eingestellt wird. Es kann also vorkommen, dass für aufgetretene Probleme keine Lösungen mehr angeboten werden.

Zusätzliche Build Tools oder Kenntnisse notwendig (3) – JS

Es wurde überprüft, ob der Einsatz des JavaScript Frameworks/Library an zusätzliche Build-Tools und Kenntnisse gekoppelt ist oder EntwicklerInnen ohne großen Aufwand sofort mit der Umsetzung des Projektes beginnen können. Bei diesem Kriterium wurde darauf geachtet ob zusätzlich zum Download der Sprache noch weitere Tools oder Kenntnisse benötigt werden. Um z.B. Angular 4 zu verwenden ist es nötig, auch Know-How in TypeScript[14] zu haben, aber auch beim Einsatz von ReactJS sind zusätzliche Plug-Ins notwendig, um den vollen Umfang der Funktionalitäten nutzen zu können.

4.2.2 Ergebnisse

In Anhang A dieser Arbeit wurde die Excel Tabelle mit der genauen Gegenüberstellung der CSS und JavaScript Frameworks/Libraries angefügt. Hier können einzelne Werte wie z.B. die Größe der reduzierten Versionen in Kilobyte oder die Anzahl der Beiträge auf der Plattform Stackoverflow nachvollzogen werden. Für die Berechnung des Endwertes wurde der Erfüllungswert eines Kriteriums mit dem Wert der Gewichtung multipliziert und für alle Kriterien addiert. Das Framework/Library mit dem geringsten Wert schneidet beim Vergleich am besten ab (siehe Tabellen 7 und 8; Seiten 62 und 63).

[14] Dokumentation und Download von TypeScript: https://www.typescriptlang.org/

Tabelle 7. Vergleich der CSS Frameworks

Kriterien	***Foundation***	***Skeleton***	***Bootstrap***	***Semantic UI***	***Materialize***
Prototyping	*1*	*3*	*1*	*1*	*2*
Lernkurve	*2*	*2*	*2*	*2*	*2*
Datenkern	*2*	*1*	*3*	*4*	*3*
Community / Support	*2*	*3*	*1*	*3*	*2*
Dokumentation	*2*	*3*	*1*	*1*	*1*
Preprozessoren	*2*	*4*	*2*	*2*	*2*
Nutzung JS Funktionen	*1*	*4*	*1*	*1*	*1*
Ergebnis	***28***	***44***	***25***	***31***	***30***

Der Vergleich der fünf ausgewählten CSS Frameworks zeigt, dass Bootstrap mit 25 erreichten Punkten das beste Ergebnis erzielt. Auf Platz 2 liegt Foundation. Knapp dahinter, punktemäßig fast gleichauf, folgen Materialize, welches auf Google's Material Design[15] basiert und Semantic UI. Klar abgeschlagen an letzter Position liegt das Skeleton Framework. Für die Umsetzung des Designs wurde daher für die Arthrose Plattform das CSS Framework Bootstrap verwendet.

[15] Informationen zu Google's Material Design: https://material.io/

Tabelle 8. Vergleich der JavaScript Frameworks/Libraries

Kriterien	***jQuery****	***ReactJS****	***AngularJS*****	***Angular 4*****	***VueJS 2*****
Prototyping	*1*	*3*	*1*	*3*	*1*
Lernkurve	*1*	*2*	*2*	*3*	*2*
Performance	*3*	*1*	*2*	*1*	*1*
Datenkern	*1*	*2*	*2*	*4*	*2*
Community / Support	*1*	*3*	*1*	*2*	*4*
Dokumentation	*1*	*1*	*1*	*2*	*2*
Lebensdauer	*1*	*1*	*3*	*2*	*2*
Build-Tools notwendig	*1*	*3*	*2*	*3*	*1*
Ergebnis	***20***	***38***	***30***	***47***	***33***

* Library, ** Framework

Bei der Gegenüberstellung der fünf ausgewählten JavaScript Frameworks/Libraries erzielt jQuery mit 20 erreichten Punkten das beste Ergebnis. Mit etwas Abstand folgen AngularJS und VueJS auf den Plätzen 2 und 3. ReactJS erreicht mit 38 Punkten Platz 4. Auf dem letzten Platz findet sich die neueste Version von Angular. Für die Interaktionselemente auf der Arthrose Plattform wurde daher die jQuery Library und das AngularJS Framework herangezogen.

5 Umsetzung des webbasierten Prototyps

Dieses Kapitel befasst sich mit der webbasierten prototypischen Umsetzung der interaktiven Plattform für Menschen mit arthrotischen Beschwerden mit Hilfe aktueller Webtechnologien HTML5, CSS3 sowie JavaScript. Das fertige Screendesign in Abbildung 10, welches von der Master-Studentin Shadja El Aeraky in Zusammenarbeit mit dieser Arbeit konzipiert wurde, diente als Grundlage für die Entwicklung des nachfolgend vorgestellten interaktiven Prototyps.

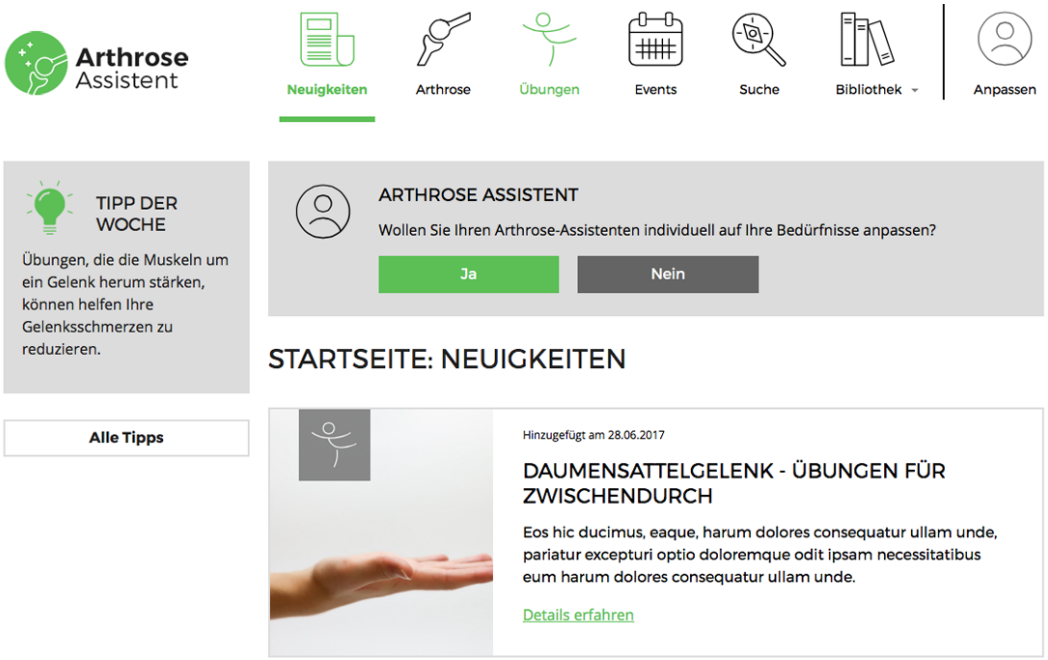

Abbildung 10. Screendesign der Startseite der Plattform (El Aeraky, 2017, S. 88)

5.1 Allgemeines

5.1.1 Hosting

Die Arthrose Plattform wurde auf der kostenlosen Hostingplattform Heroku[16] veröffentlicht und kann unter folgender URL abgerufen werden:

https://arthrose-assistent.herokuapp.com

Heroku bietet die Möglichkeit, eine Anwendung in einer ausgewählten Programmiersprache zu erstellen und diese dort zu verwalten. Damit eine Anwendung auf Heroku gehostet werden kann, muss zuerst ein Account erstellt werden. Um ein neues Projekt anzulegen, ist es wichtig, sich über die Konsole mit dem Befehl `heroku login` mit Benutzernamen und Passwort anzumelden. Anschließend kann über den Befehl `heroku create <name>` eine neue leere Anwendung erstellt werden und Heroku stellt eine Domain sowie ein GIT Repository mit dem angegebenen Namen zur Verfügung. Zum Testing und zur Entwicklung einer Anwendung mit Heroku werden lediglich drei Dateien benötigt:

Datei 1: Package.json

Damit Heroku die Anwendung auf der erstellten Domain veröffentlichen kann, muss der Anwendung noch eine Skriptsprache zugeordnet werden (zur Auswahl stehen u.a. NodeJS, Java, PHP, etc.). Die Arthrose Plattform basiert auf dem JavaScript-Framework NodeJS[17]. Daher muss zunächst eine `package.json` Datei erstellt werden (siehe Listing 20).

Listing 20. Konfigurationsdatei package.json (eigener Code)

```
{
  "name": "arthrose-assistent",
  "version": "0.2.6",
   "main": "index.js",
  "scripts": {
    "start": "node index.js"
  }
"dependencies": {
    "express": "^4.15.4"
  }
}
```

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

[17] Informationen zu NodeJS unter https://nodejs.org/en/

Die Erstellung der Datei kann entweder über eine Konsole mit dem Befehl `npm init` erfolgen oder manuell. Der Name und die Version sind essentielle Bestandteile der Datei und bilden zusammen einen eindeutigen Identifier. Der `name` in der package.json Datei muss mit dem vergebenen Namen bei der Erstellung der App identisch sein. Als Einstiegspunkt für die Anwendung wird anschließend die `index.js` Datei festgelegt. Damit die App gestartet werden kann und um Routen zu definieren, wird das Web-Framework Express.js[18] benötigt. Express.js ist ein schnelles und flexibles Framework, welches ein Set von Features bereitstellt, um Web- sowie mobile Anwendungen zu bauen. Um Express.js zu nutzen, muss der Befehl `npm install` ausgeführt werden und das Framework wird als Abhängigkeit installiert.

Datei 2: Procfile (optional)

Das Procfile ist eine Textdatei, in der festgelegt wird, mit welchem Befehl die App gestartet werden soll. Diese Datei enthält lediglich die Zeile: `web: node index.js`. Wird kein Procfile definiert, greift die Anwendung auf den, in Listing 20 (siehe Seite 65) definierten, Startbefehl zurück.

Datei 3: .env (optional)

Die Environment-Datei (`.env`) ist ähnlich wie das Profile optional. Hier kann der Port der Anwendung, auf der die App lokal laufen soll, angegeben werden. Diese Datei ist vor allem dann wichtig, wenn gleichzeitig mehrere Node-Anwendungen über Heroku lokal getestet werden und ein eigener Port für die jeweilige Anwendung definiert werden muss. Dafür muss allerdings in die Environment-Datei (`.env`) im Root-Verzeichnis ein anderer Port als der Standardport 5000 eingetragen werden z.B. `PORT=7000`.

Um die Anwendung letztendlich lokal mit Heroku zu testen und zu entwickeln, muss der Befehl `heroku local web` aufzurufen werden. Anschließend steht die Anwendung unter der Domain http://localhost:5000 zur Verfügung.

Da die Plattform für einen Nutzungstest online zur Verfügung stehen musste, gab es vor der prototypischen Umsetzung auch die Überlegung, die Anwendung auf dem eigenen Webhosting Service SHEEP (Student Hosting Environment for Extended/external Projects) der FH St. Pölten zu hosten. Dieser steht StudentInnen der Fachhochschule kostenlos zur Verfügung und läuft mit der Skriptsprache PHP. Das ausschlaggebende Kriterium für das Hosting auf Heroku war jedoch, dass Heroku beim Hochladen von Änderungen in das Repository mittels `git push` die geänderten/neuen Dateien automatisch auf den Server hochlädt. Um das gleiche beim FH eigenen Hosting zu erreichen, hätten die Dateien entweder immer manuell über einen FTP (File Transfer Protocol) Client

[18] Informationen zu Express.js unter http://expressjs.com/de

hochgeladen werden müssen oder es wäre die Einrichtung eines `git hooks` Service[19] erforderlich gewesen, um die Daten zu aktualisieren. Aufgrund der Zeitersparnis und Einfachheit des Heroku Services, wurde das Hosting auf Heroku bevorzugt.

5.1.2 Architektur

Die Arthrose Plattform basiert auf dem JavaScript-Framework NodeJS in Kombination mit dem Web-Framework Express.js. Um in NodeJS jedoch HTML-Seiten erstellen zu können, bedarf es einer der unzähligen Javascript-Templating-Engines. Hierfür wurde EJS (Effective JavaScript templating)[20] verwendet, da hier der HTML Code einfach um eine JavaScript ähnliche Syntax erweitert werden kann.

Da es für NodeJS keine Best-Practice gibt, wie eine Systemarchitektur am Besten auszusehen hat oder Vorgaben woran man sich halten sollte, wurde nachfolgende Ordnerstruktur für die Web-Anwendung gewählt (siehe Abbildung 11).

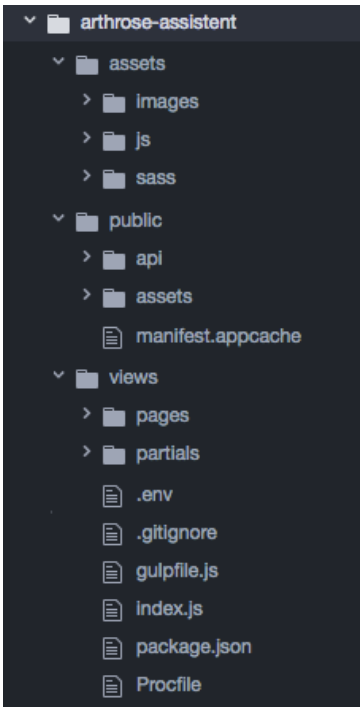

Abbildung 11. Systemarchitektur der Plattform (eigene Abbildung)

Der assets-Ordner enthält alle Source-Dateien, welche mittels Task Runner noch für die Anwendung aufbearbeitet werden müssen. Mittels dem Task Runner Gulp (Details siehe Kapitel 5.1.5), der die jeweiligen Unterordner überwacht, werden

[19] GIT Hooks Dokumentation: https://git-scm.com/docs/githooks

[20] EJS (Effective JavaScript templating): http://ejs.co/

z.B. die Bilder auf ein bestimmtes Größenformat angepasst oder die SASS Dateien in für Browser verständliches CSS kompiliert. Diese werden automatisch im Ordner public/assets abgelegt und können somit in die Anwendung eingebunden werden. Der public-Ordner enthält somit alle statischen Dateien wie z.B. Logo, Favicon-Icon oder CSS- und JavaScript Dateien. Im views-Ordner liegen alle EJS Dateien, die für den strukturellen Aufbau der Anwendung wichtig sind. Diese sind wiederum eingeteilt in die jeweiligen Seiten (pages) und Blöcke (partials). Die verschiedenen Blöcke werden mittels `include`-Befehl in die einzelnen Seiten eingebunden. Nähere Informationen zum Einbinden von Dateien via EJS wird im Kapitel 5.1.4 erläutert.

5.1.3 Einstieg in die Anwendung

Listing 21 zeigt Auszüge aus dem Einstiegspunkt der Arthrose Plattform in der Datei index.js. Hier werden u.a. die Templating Sprache, globale Variablen und die Routen der Anwendung definiert.

Listing 21. Basisdatei der Anwendung: index.js (eigener Code)

```
var express = require('express'),
    app = express();

app.use(express.static(__dirname + '/public'));

// views is directory for all template files
app.set('views', __dirname + '/views');
app.set('view engine', 'ejs');

//global variables
app.locals.siteTitle = "Arthrose Assistent";

//routes
app.get('/', function(request, response) {
  response.render('pages/index, {
    'pageTitle': "Startseite", 'pageID': "home"
  });
});
...
//default: if requested page was not found
app.get('*', function(request, response) {
   response.render('pages/index, {
    'pageTitle': "Fehlerseite", 'pageID': "error"
  });
});
```

Zunächst wird in den ersten beiden Zeilen, das Web-Framework Express.js mittels `require` Methode angefordert und für die Anwendung festgelegt. Damit kann auf verschiedene Methoden, die Express zur Verfügung stellt, zugegriffen werden. Eine davon ist die `use()` Methode, mit der das Verzeichnis festgelegt wird, in dem sich alle statischen Dateien wie CSS, JavaScript oder Bilder befinden.

Anschließend legt man über die Methode `set()` den Ordner der Template Dateien (`views`) und die Templatesprache (`view engine`) selbst fest. Gefolgt von der Eigenschaft `locals`, mit der es möglich ist, globale Variablen zu definieren, welche über das gesamte Projekt hinweg benutzt werden können. Hierfür eigenen sich vor allem Werte, die sich während der Laufzeit der Anwendung nicht ändern werden, wie z.B. der Seitentitel.

Im nächsten Schritt werden via `get()` Methoden die unterschiedlichen Routen der Anwendung definiert. Der erste Parameter der Methode legt die Pfad-Komponente der angeforderten URL fest. Das ' `/` ' steht hier für die Startseite der Plattform, wohingegen mit ' `*` ' eine Fallback Seite aufgerufen wird, falls eine Seite nicht gefunden werden kann, ein Link nicht korrekt angegeben ist oder der/die UserIn eine falsche URL-Eingabe getätigt hat. Als zweites Argument wird eine Funktion mit zwei Übergabeparametern (`request, response`) überreicht. Beim Aufruf der Startseite, liefert diese Funktion eine „`response`“ zurück, welche die `index` Datei im Ordner `pages` aufruft und mittels `render` darstellt. Zusätzlich werden der Antwort zwei Parameter übergeben (`pageTitle, pageID`). Anhand der `pageID` kann die jeweilige Seite in der Anwendung eindeutig identifiziert und bei Bedarf angesprochen werden. So ist es möglich, gezielt Inhalte für bestimmte Seiten ein- und/oder auszublenden.

Alle Methoden, Eigenschaften und Events, die Express.js für EntwicklerInnen bereithält, können in der ausführlichen Dokumentation nachgelesen werden[21].

5.1.4 Templating Sprache EJS

Dieses Kapitel der Arbeit soll keine Dokumentation der Templating Sprache EJS darstellen, sondern anhand eines Praxiseinsatzes zeigen, welche Aufgaben EJS in der Anwendung übernimmt. Mithilfe von EJS kann HTML durch den reinen Einsatz von JavaScript generiert werden. Dabei verwendet EJS bestimmte Tags um auf JavaScript Variablen zuzugreifen bzw. diese auszugeben (Listing 22).

Listing 22. Einsatz von EJS in der Anwendung (eigener Code)

```
//request pageID
<% if(pageID == "home"){ %>
   <%- include ../pages/news %>
<% }
else if (pageID == "arthrose") { %>
   <%- include ../pages/arthrose %>
<% }
...
else { %>
   <%- include ../pages/404-error %>
```

[21] Dokumentation Express (Version 4.x): http://expressjs.com/en/4x/api.html

```
<% } %>

//display pageID
<body id="<%= pageID %>" ... >
```

Wie in Listing 21 (siehe Seite 68) dargestellt, wird bei jedem Seitenaufruf der Seitentitel (`pageTitle`) und der Seiten-Identifier (`pageID`) übergeben. Diese Funktionalität wird nun in Listing 22 genutzt, um mit EJS die jeweiligen Templates seitenabhängig zu laden und anzuzeigen. In der `if-else` Abfrage werden im `<main>` HTML Element der Anwendung je nach `pageID` die Inhalte dargestellt oder ausgetauscht. Wird eine nicht verfügbare Seite oder eine ungültiger Seiten-Identifier aufgerufen, wird die Fehlerseite (`404-error.ejs`) angezeigt. Durch die Platzierung der `pageID` Variable im `<body>` Element der Seite wird angegeben, welche Inhalte geladen werden sollen. Zusätzlich kann der Identifier genutzt werden, um z.B. mittels CSS oder JavaScript Elemente der Seite zu manipulieren.

5.1.5 Assets optimieren

Webanwendungen werden immer umfangreicher und die enthaltenen Funktionalitäten nehmen immer mehr zu. Um lange Ladezeiten für den User zu vermeiden, sollte der Umfang der Anwendung daher so klein wie möglich gehalten werden. Zur Verringerung der Ladezeit der Arthrose Plattform, wurde für die Optimierung der Assets (CSS- und JavaScript Dateien sowie Bilder) ein Task Runner verwendet. Wie im tabellarischen Vergleich in Kapitel 3.2.3 ersichtlich ist, überwiegen die Vorteile von Gulp gegenüber den anderen beiden vorgestellten Tools Grunt und Webpack, daher wurde Gulp zur Optimierung der verschiedenen Assets eingesetzt (siehe Listings 23-25).

Listing 23. Optimierung der CSS Dateien mit Gulp (eigener Code)

```
var gulp = require('gulp'), sass = require('gulp-sass'), ...;
//optimize CSS
gulp.task('sass', function () {
  gulp.src(cssFiles)
    .pipe(sass.sync().on('error', sass.logError))
    .pipe(prefix('last 1 version', '> 3%'))
    .pipe(cleanCSS())
    .pipe(gulp.dest('public/assets/css'));
});
```

Ähnlich wie in Listing 7 (siehe Seite 32) müssen zunächst die unterschiedlichen Erweiterungen (Plug-Ins) als Variablen definiert werden, um später darauf zugreifen zu können. Essenziell für die Komprimierung sind nun die beiden Plug-Ins `prefix()` und `cleanCSS()`. Dem `prefix` Plug-In werden zwei Optionen übergeben. Mit der Angabe `last 1 version` werden zunächst die aktuelle und die letzte Version des Browsers angesprochen. Kennt die jeweilige Browserversion eine CSS Eigenschaft nicht, wird die Eigenschaft nochmals

aufgelistet, aber zu Beginn mit einem browserspezifischen CSS-Präfixe[22] versehen. Jedoch wird dieser gerade beschriebene Vorgang nur für jene Browser ausgeführt, deren Marktanteile über 3% liegen[23]. Übergibt man der `prefix()` Funktion keine Angaben, werden die Standardwerte des Plug-Ins verwendet, was wiederum dazu führen kann, dass die komprimierte Datei größer wird. Die Funktion `cleanCSS()` sorgt anschließend dafür, dass alle Leerzeichen und Umbrüche im CSS entfernt werden und erzeugt so eine unlesbare Aneinanderreihung von Eigenschaften.

Listing 24. Optimierung der JavaScript Dateien mit Gulp (eigener Code)

```
gulp.task('js', function () {
   return gulp.src(jsFiles)
      .pipe(concat('app.js'))
      .pipe(minify({
         ext: { min:'.js' }, noSource: true
      }))
      .pipe(gulp.dest('public/assets/js'))
});
```

Das Plug-In `concat()` sorgt dafür, dass alle JavaScript Dateien zu einer Datei mit dem Namen `app.js` zusammengefügt werden. Ähnlich wie `cleanCSS()` bei der Optimierung von CSS, kümmert sich `minify()` um das unlesbar machen der finalen JavaScript Datei. Dies ist auch eine Methode, um es externen EntwicklerInnen zu erschweren, die Datei zu lesen und so eine ungewollte Verbreiterung zu verhindern. Abschließend wird im letzten Kanal (`gulp.dest`) die erzeugte Datei in den angegebenen Pfad gespeichert.

Listing 25. Optimierung der Bilder mit Gulp (eigener Code)

```
gulp.task('images', function () {
  return gulp.src(imgFiles)
      .pipe(imageResize({
         width : 1280, height: 800, upscale : false
      }))
      .pipe(imagemin())
      .pipe(gulp.dest('public/assets/images/'));
});
```

Bei der Optimierung der Bilder kommen zwei Plug-Ins zum Einsatz. Mit Hilfe von `imageResize()` können die Bilder auf eine vordefinierte Größe skaliert werden. Die Angaben der Optionen Breite (`width`) und Höhe (`height`) sorgen dafür, dass je nach Hoch- oder Querformat des Bilders, dieses nie unter den angegeben Wert verkleinert wird. Läuft ein querformatiges Bild durch den Kanal (pipe) so ist es danach immer mindestens 1280 Pixel breit (Hochformat: 800 Pixel). Sollte ein Bild jedoch kleiner als 1280 Pixel sein, wird es nicht auf den angegebenen Wert

[22] Die Marktanteile der Browser werden von der Seite http://caniuse.com bezogen

[23] Eine Liste aller Brower-Präfixe kann hier nachgelesen werden: https://www.mediaevent.de/css/browser-praefix.html

vergrößert, dafür sorgt die Option `upscale: false`. Das Plug-In `imagmin()` komprimiert das Bild letztendlich noch und spart dadurch nochmals zusätzliche Bytes ein. Welche Einsparungen durch Gulp erzielt werden konnten, zeigt der Vorher-Nachher Vergleich in Tabelle 9.

Tabelle 9. Vergleich Assets vor und nach Optimierung

	CSS	***JavaScript***	***Bilder***
Vorher	*224 KB*	*38 KB*	*10,7 MB*
Nachher	*112 KB*	*21 KB*	*1,9 MB*
Reduzierung	*ca. 50%*	*ca. 45%*	*ca. 82%*

In dem Vorher-Nachher Vergleich der Assets kann man gut erkennen, welche Einsparungen erreicht wurden. Am meisten Speicherplatz und somit auch Ladezeit konnte bei den Bildern verringert werden. Dies liegt auch daran, dass diese vorab ohne jegliche Komprimierung in der Originalgröße in die vorgesehenen Ordner geladen wurden. Bei den CSS Dateien konnte die Hälfte der ursprünglichen Größe erreicht werden. Die Einsparungen bei den JavaScript Dateien ist etwas weniger wie bei CSS, die Gesamtgröße von JavaScript ist jedoch im Vergleich zu den anderen beiden Assetsgruppen eher gering.

5.1.6 Versionierung

Um die Vorteile der Versionierung auszunutzen, damit ggf. zu einem späteren Zeitpunkt auf ältere Dateiversionen und Datenbestände zurückgegriffen werden kann, wurde die Anwendung mittels des Versionierungssystems GIT[24] in einem privaten Repository unter https://bitbucket.org/daniel_winter_/arthrose-assistent abgelegt. Da es sich bei der Anwendung rein um einen Testprototypen handelt, wurde entschieden, das Repository nicht öffentlich zugänglich zu machen.

5.2 Technische Umsetzung

5.2.1 Lokale Datenspeicherung

Um die Plattform individuell an die Bedürfnisse anzupassen, ist es möglich, ein eigenes Profil zu hinterlegen. Daher kann man zum einen Gelenksregionen auswählen, um die Inhalte verschiedener Seiten auf die betroffenen Gelenke zu beschränken. Zum anderen können einzelne Themengebiete festgelegt werden, um auf der Startseite nur Neuigkeiten anzuzeigen, welche den/die UserIn

[24] GIT: https://git-scm.com/

interessieren. Die getätigten Anpassungen können jederzeit geändert werden, um bei Bedarf wieder alle Inhalte darzustellen.

Die Ergebnisse der zweiten Fokusgruppe Mitte Juni 2017 haben gezeigt, dass es bei einigen Testpersonen mehrere Betroffene im Haushalt gibt, die Arthrose haben. Da in diesen Haushalten oftmals nur ein Computer oder Laptop zur Verfügung steht sowie auch der gleiche Browser benutzt wird, sollte es zusätzlich möglich sein, mehrere Profile anzulegen und zwischen diesen zu wählen. Für weitere Informationen kann die Arbeit von Shadja El Aeraky (2017, S. 75) herangezogen werden.

Um die Personalisierung der Arthrose Plattform zu erreichen, wurden die erhobenen Daten des/der Users/Userin im Browser gespeichert. Es wurde sowohl die dauerhafte Webstorage Methode Local Storage als auch IndexedDB verwendet. Local Storage kommt für die Informationsbox auf der Startseite zum Einsatz, hier wird abgefragt, ob der/die UserIn die Plattform personalisieren will. Der restliche Teil der Anpassung wird jedoch ausschließlich mit IndexedDB realisiert.

5.2.1.1 Local Storage

Der Informationshinweis auf der Startseite der Anwendung wurde ausschließlich mit Local Stoarge und der JavaScript Library jQuery realisiert. Zur besseren Darstellung wird die Umsetzung auf zwei Listings (26, 27) aufgeteilt.

Listing 26. Info-Box Realisierung Startseite Teil I (eigener Code)

```
var customizeData = JSON.parse(localStorage.getItem('customizeData'));
if (customizeData == null || customizeData.length == 0) {
  customizeData = { active: true };
  localStorage.setItem('customizeData',JSON.stringify(customizeData));
}
else if (customizeData.active == false)  {
  $(".alert-box").remove();
}
```

Zu Beginn wird eine Variable `customizeData` deklariert und mit `localStorage.getItem` einem Wert aus der lokalen Datenbank initialisiert. Sollte der Wert jedoch undefiniert oder leer sein, wird der vorab deklarierten Variable ein Objekt zugewiesen und mit `setItem` in die Datenbank geschrieben. Das Setzen der Eigenschaft `active: true` im Objekt bedeutet, dass die Info-Box auf der Startseite sichtbar ist. Ändert sich der boolsche Wert der Eigenschaft `active` jedoch auf `false`, wird die Info-Box im `else if` Abschnitt der Abfrage mittels jQuery Selektor `remove()` entfernt.

Listing 27. Info-Box Realisierung Startseite Teil II (eigener Code)

```
// user/in want to customize
$("#customizeBtnTrue").on("click", function() {
  $(".jsCustomizeDropdown").slideToggle('slow');
```

```
  customizeData = { allowed: true, active: false };
  setCustomizeData();
});

// user/in DONT want to customize
$("#customizeBtnFalse").on("click", function() {
  customizeData = { allowed: false, active: false };
  setCustomizeData();
});

function setCustomizeData() {
  $(".alert-box").remove();
  localStorage.setItem('customizeData',JSON.stringify(customizeData));
}
```

Sollte der/die UserIn eine Personalisierung wünschen und auf den Button mit der ID `customizeBtnTrue` klicken, wird das Personalisierungs-Dropdown via `slideToggle()` eingeblendet. Im Objekt `customizeData` wird die Eigenschaft `active` auf `false` gesetzt und zusätzlich eine neue Eigenschaft zugewiesen. Diese zeigt an, ob die Personalisierung erlaubt wurde oder nicht und kann eventuell für spätere Abfragen wiederverwendet werden. Auch wäre es vorstellbar, diese Informationen an eine serverseitige Datenbank weiterzuleiten, um herauszufinden, ob die Info-Box generell genutzt wird oder ob UserInnen den Menüpunkt „Anpassen“ zur Personalisierung bevorzugen.

Abschließend wird noch die Funktion `setCustomizeData()` ausgeführt, in welcher die Info-Box ausgeblendet wird und die Daten gespeichert werden. Wünscht der/die UserIn allerdings keine Anpassung, folgen die gleichen Schritte wie gerade beschrieben, mit dem Unterschied, dass die Eigenschaft `allowed` auf `false` gesetzt wird und das Personalisierungs-Dropdown nicht geöffnet wird.

5.2.1.2 IndexedDB

Die Möglichkeit, mehrere Profile in einem Browser anlegen und verwalten zu können, bedarf mehr als die von localStorage gebotenen Funktionen `getItem()`, um Daten aus der Datenbank zu holen und diese mit `setItem()` wieder zu speichern. Es muss zusätzlich die Möglichkeit geben, die Werte eines angelegten Profils zu aktualisieren, diese „aktiv“ zu setzen und alle anderen angelegten Profile wiederrum auf „inaktiv“ zu stellen.

Das von Google entwickelte AngularJS[25] Framework bietet für solche Szenarien die Möglichkeit der „Services“ an. Ein Service stellt ein JavaScript Objekt zur Verfügung, das wiederum über „Dependency Injection“ in weitere Komponenten eingebunden werden kann, wie z.B. in Controller oder andere Services (Potthof, 2014).

[25] Eine ausführliche Dokumentation ist unter https://docs.angularjs.org/guide verfügbar

Erstellen eines Service

Zum Erzeugen eines Services bietet Angular drei Methoden an: `service()`, `factory()`, und `provider()`. Potthof (2014) empfiehlt für Funktionen die in Services ausgelagert werden sollen die Methode `factory()`, da dessen return-Werte Objekte sind, welche wiederum einfach in andere Komponenten einfügt werden können. Die Listings 28 und 29 zeigen Auszüge aus dem erstellten Service und Listing 30, dessen Einbindung in den Controller (siehe Seite 78), auf.

Listing 28. Erstellen eines Services (eigener Code)

```
app.factory('iDBS', ['$q', '$window', function($q, $window) {
   var indexedDB = $window.indexedDB || $window.mozIndexedDB || ... ;
   ...
   var open = function () {
      var deferred = $q.defer();
      var request = indexedDB.open("users", 1);
      request.onupgradeneeded = function (e) {
         db = e.target.result;
         e.target.transaction.onerror = indexedDB.onerror;
         if (db.objectStoreNames.contains("people")) {
            db.deleteObjectStore("people");
         }
         var store = db.createObjectStore("people", {keyPath:"name"});
         store.createIndex("name", "name", {unique: true });
      };
      request.onsuccess = function (e) {
         db = e.target.result;
         deferred.resolve();
       };
      request.onerror = function () { deferred.reject(); };
      return deferred.promise;
  };
}]);
```

In der ersten Zeile des Listings wird der Name des Service festgelegt (`iDBS`) sowie der in Kapitel 3.2.6 beschriebene Promise-Service `$q` und das `$windows` Objekt injiziert. Zu Beginn des Service, wird die Variable `indexedDB` deklariert. Hier ist es essenziell, dass alle möglichen Browser Prefixes angegeben werden, da ansonsten die Daten nicht in allen Browser gespeichert werden, vor allem für ältere und mobile Browser ist diese Angabe vonnöten.

Anschließend wird die erste Funktion im Service deklariert (`open`). Diese Funktion dient dazu, die lokale indexedDB-Datenbank im Browser einzurichten. Dafür wird zunächst ein deferred-Objekt sowie eine Datenbank mit dem Namen `users` erzeugt. Die Funktion `onupgradeneeded()` erzeugt einen Object Store (Tabelle) mit dem Namen `people` inklusive des Identifiers (keyPath) `name`. Der festgelegte Primarschlüssel muss eindeutig sein und sollte nicht mehrfach vergeben werden können. Da es in einem Haushalt mehrere mit Arthrose betroffene Personen, welche die Plattform nutzen, geben kann, sollte der Name als Identifier jedoch

ausreichen. Der letzte Schritt beim Einrichten der Datenbank ist es, den Object Store mit einem Index (`createIndex`) zu versehen, nach dem gesucht werden kann.

Hat das Einrichten der Datenbank funktioniert, muss in der Funktion `onsuccess()` noch die Methode `deferred.resolve()` des deferred-Objekts aufgerufen werden, um den Promise aufzulösen. Optional kann bei einem Fehler (`onerror`) wiederum das gleiche mit der Methode `reject()` gemacht werden. Abschließend wird das deferred-Objekt als Promise (`deferred.promise`) zurückgegeben. Wie dieses Promise aufgelöst wird, zeigt Listing 30.

Anlegen von UserInnen

Das Service enthält noch vier weitere Funktionen neben dem Erstellen der Datenbank: das Hinzufügen und Aktualisieren von UserInnen, das Laden aller UserInnen sowie eines/einer einzelnen Users/Userin. Im Folgenden Listing 29 wird das Anlegen eines/einer Users/Userin erklärt, da hier das Zusammenspiel von IndexedDB und Promises gut verdeutlicht wird. Alle anderen Methoden funktionieren größtenteils sehr ähnlich und werden nicht separat erläutert.

Listing 29. addUser() Funktion im Service (eigener Code)

```
var addUser = function (name) {
   var deferred = $q.defer();
   if (db === null) {
      deferred.reject("DB not exist");
   } else {
      var trans = db.transaction(["people"], "readwrite");
      var store = trans.objectStore("people");
      //create user object
      var interests = [];
      $(".jsAddSingleUser input:checkbox").each(function () {
         interests.push( {
            id: $(this).attr("id"),
            value: $(this).prop('checked') });
         });
         lastIndex++;
         var user = {
            "id": lastIndex, "name": name,
            created: moment().format('LLL'),
            interests: interests
         };
         var request = store.add(user);
         ...
};
```

Wie schon beim Anlegen der Datenbank, wird auch beim Anlegen eines/einer neuen Users/Userin zuerst ein deferred-Objekt erzeugt. Anschließend wird in einer `if/else` Bedingung, abgefragt, ob die IndexedDB Datenbank existiert, falls nicht, wird dem deferred-Objekt die Methode `reject()` mit einer Nachricht übergeben,

dass die Datenbank nicht existiert. Im `else` Bereich geschieht dann der wesentliche Teil: das Hinzufügen eines/einer Users/Userin. Zunächst wird eine Transaktion gestartet, mit der sowohl in die Datenbank geschrieben als auch davon gelesen werden kann. Die erstellte Transaktion fragt dann nach dem Objekt-Store `people` in dem der/die neue UserIn gespeichert werden soll.

Im nächster Schritt werden die Daten für das UserInnen-Objekt zusammengetragen. Zunächst wird das Array `interests[]` deklariert. Um das Array zu befüllen, wird die jQuery Funktion `each()` verwendet. Diese geht alle Checkboxen in der Klasse `jsAddSingleUser` durch und fügt dem Array mit der Methode `push` für jede Checkbox ein neues Objekt mit zwei Eigenschaften hinzu. Der Schlüssel der ersten Eigenschaft ist `id` und greift auf den Wert des Attributes `#id` der Checkbox zu. In der zweiten Eigenschaft `value` wird gespeichert, ob der Wert der Checkbox geklickt wurde (`true`) oder nicht (`false`).

Im Anschluss wird die zuvor global deklarierte Variable `lastIndex` um einen Wert nach oben gezählt, damit für die Erstellung des UserInnen-Objektes ein zusätzlicher Identifier (`id`) zur Verfügung steht. Der Name des/der Users/Userin wird zu Beginn der Funktion als Übergabeparameter definiert und später in den Wert der Eigenschaft `name` übergeben. Zusätzlich wird im Objekt noch der Zeitpunkt der Erstellung gespeichert (`created`) und mit der JavaScript Bibliothek MomentJS[26] in ein lesbares Format gebracht. Auch wäre es denkbar, hier den Wert rein als Timestamp mit der bereitgestellten JavaScript Funktion `new Date()` zu hinterlegen. Als letzter Wert des UserInnen-Objekts wird der zuvor generierte interests-Array hinzugefügt.

Um den/die UserIn letztendlich in der Datenbank zu speichern, wird von der zuvor initialisierten Variable `store`, die `add()` Funktion aufgerufen und das erzeugte UserInnen-Objekt übergeben und in die Datenbank geschrieben. Abschließend werden in dieser Funktion ähnlich wie in Listing 28. die Funktionen `onsuccess()` und `onerror()` spezifiziert und die jeweiligen `deferred` Methoden aufgerufen. Da die Vorgehensweise bei allen Funktionen sehr ähnlich ist, wird der Code in diesem Listing nicht ein weiteres Mal aufgeführt.

[26] Dokumentation und Details zu MomentJS: https://momentjs.com/

Seiten-Controller

Der Zugriff auf die Funktionen des erstellten Service erfolgen nun über den Seiten-Controller (siehe Listing 30).

Listing 30. Auszug aus AngularJS Seiten-Controller und Einbindung/Aufruf in HTML (eigener Code)

```
//insert controller into HTML
<body ... data-ng-controller="pageController as aa">

// call function via HTML Button
<button class="... " data-ng-click="aa.addUser()">Speichern</button>

//declaring controller
app.controller('pageController', ['iDBS', $window, function (iDBS,
...) {
   ...
   aa.addUser = function () {
      iDBS.addUser(aa.name).then(function () {
         aa.messageAdd = "Benutzer '" + aa.name + "' erfolgreich
angelegt!";
         $timeout(function() { aa.messageAdd = false; }, 5000);

         var customizeData = { allowed: false, active: false };
         localStorage.setItem('customizeData',
JSON.stringify(customizeData));
         $(".alert-box").remove();
         $('#addSingleUser')[0].reset();

         aa.listUsers();
      }, function (err) {
          $window.alert(err);
      });
   };
   ...
}]);
```

Da die Daten des/der angelegten Users/Userin jederzeit zur Verfügung stehen müssen, wird der Seiten-Controller global in das `body` Element der Seite eingefügt. AngularJS stellt dafür das `ng-controller` Attribut zur Verfügung. Dem Controller wird ein Aliasname zugewiesen, mit dessen Hilfe dieser im weiteren Verlauf angesprochen werden kann. Die Abkürzung `aa` steht für „arthrose-assistent“. Des Weiteren ist es wichtig, alle AngularJS Attribute welche mit `ng` beginnen, zusätzlich zu Beginn noch mit dem Wert `data` zu versehen, damit bei der Validierung der Plattform mit Hilfe eines HTML5 Validators[27] keine Fehlermeldungen diesbezüglich angezeigt werden.

[27] https://validator.w3.org/

Die in Listing 29 (siehe Seite 76) erklärte Service-Funktion `addUser()` wird nun mit Hilfe von AngularJS und dem Attriubut `data-ng-click` an einen Button gebunden (Zeile 5). Wird dieser Button geklickt, wird im `pageController` die gleichnamige Funktion `addUser()` aufgerufen und zudem der Username (`aa.name`) übergeben. Der Service `iDBS` kann im Seiten-Controller genutzt werden, da dieser mittels Dependency Injection zur Verfügung gestellt wird (Zeile 8). Der Aufruf des Promise erfolgt mittels `.then()` Methode. Zu Beginn der Methode wird die Variable `messageAdd` erzeugt, welche per JavaScript Funktion `$timeout` für fünf Sekunden eingeblendet wird, nachdem der/die BenutzerIn angelegt wurde. Sollte der/die BenutzerIn die Personalisierung nicht über die Info-Box auf der Startseite aufgerufen haben, sondern über das Navigationsmenü, wird diese im LocalStorage wie in Listing 27 beschrieben, auf inaktiv gesetzt und ausgeblendet.

Danach werden alle Daten in den Checkboxen und das Eingabefeld des Formulars via jQuery `reset()` Funktion geleert, um die Eingabe eines/einer weiteren Users/Userin zu erleichtern. Zuletzt wird noch die Funktion `listUsers()` aufgerufen, die dafür sorgt, dass alle neu angelegten UserInnen direkt neben dem Button „Person hinzufügen“ auflistet werden (siehe Abbildung 12).

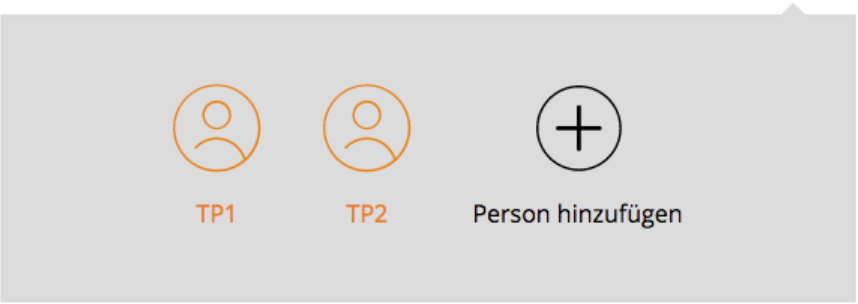

Abbildung 12. Anzeige zweier angelegter UserInnen (eigene Abbildung)

5.2.2 Einbindung externer Dienste

Um spezielle Features des Prototyps der Arthrose Plattform von der Zielgruppe testen zu lassen, war es notwendig, auf die Dienste externer Anbieter zurückzugreifen. Via API des Anbieters können die Daten dieser Dienste abgerufen werden. Zum einen wurde die Google Standortsuche (Places API) genutzt, um nach ExpertInnen und Einrichtungen (Krankenhäuser, Apotheken, etc.) zu suchen und zum anderen wurde die API des Videoportals YouTube verwendet, um Übungen zu den jeweiligen Arthrose Arten ansehen zu können.

5.2.2.1 Google Places API für Expertensuche

Bevor man die Places API nutzen kann, muss zuerst ein Google Account erstellt und anschließend ein API-Key für Google Places generiert werden. Hat man sich einmal bei Google angemeldet, kann man diverse API’s und Dienste für die

jeweiligen Projekte aktivieren lassen. Dieser Schlüssel, bestehend aus Zahlen und Buchstaben, muss bei jedem Aufruf der API übergeben werden und dient zur Identifizierung. Um im abschließenden Nutzungstest nach ExpertInnen und Einrichtungen suchen zu können, wurde im Prototypen der Arthrose Plattform die kostenlose frei zugängliche Google Places API[28] verwendet, welche auf der Google Maps API[29] aufbaut und zusätzliche Informationen zum Ort zur Verfügung stellt. Zur besseren Übersicht werden die Auszüge über den Einsatz dieser API in mehrere Listings (31-33) aufgeteilt.

Listing 31. HTML Code Auszug Google Places (eigener Code)

```
//search
<input type="text" id="plz" class="form-control input-search"
placeholder="PLZ">
<input type="text" id="place" class="form-control input-search"
placeholder="Ort">
<select class="form-control" id="expert">
   <option value="Apotheken">Apotheken</option>
   <option value="Krankenhäuser">Krankenhäuser</option>
</select>
...
<button type="submit" class="btn btn-block btn-primary"
onclick="initMap()" disabled="disabled">Suchen</button>
//show map and results
<div id="map"></div>

<div id="results" class="expert-search__results">
   <div class="row row-eq-height"></div>
</div>
```

Die Suche enthält zu Beginn zwei `<input>` Felder sowie eine `<select>` Auswahlbox. Die beiden Eingabefelder sind Pflichtangaben und müssen die Postleitzahl (`plz`) und den Ort (`place`) enthalten. Außerdem kann mittels der select-Box ein/e Experte/Expertin oder eine Einrichtung ausgewählt werden. Nur wenn beide Eingabefelder ausgefüllt wurden, wird die Eigenschaft `disabled="disabled"` des Suchen Buttons deaktiviert und der/die UserIn kann den Button klicken und die Funktion `initMap()` wird ausgeführt (siehe Listing 32). Danach folgt die Anzeige der gefundenen Ergebnisse als Marker auf der Karte, welche mit dem Identifier `map` angesprochen wird. Gefolgt von den gefundenen Ergebnissen, die zweispaltig in Textform dargestellt werden (`results`).

Listing 32. Initialisierung der Google Maps Karte (eigener Code)

```
function initMap() {
   ...
   $('#results .row').empty();
   var geocoder = new google.maps.Geocoder,
```

[28] Details zu Google Places: https://developers.google.com/places/javascript/?hl=de

[29] Übersicht Google Maps: https://developers.google.com/maps/

```
      plz = $('#plz').val(),
      place = $('#place').val(),
      expertValue = $('#expert').val();
  if (plz === '') { plz = "1010"; } else {plz = plz; }
  ...
  var address = +plz+" "+place;
  geocoder.geocode({ 'address': address}, function(results, status) {
     if (status == google.maps.GeocoderStatus.OK) {
        var lat = results[0].geometry.location.lat();
        var lng = results[0].geometry.location.lng();
     }
     var coordinates = {lat: lat, lng: lng};
     map = new google.maps.Map(document.getElementById('map'), {
       center: coordinates, zoom: 14, scrollwheel: false
     });
     var request = {
       location: coordinates, radius: 1000, query: expertValue
     };
     var service = new google.maps.places.PlacesService(map);
     service.textSearch(request, callback);
  });
}
```

Als Erstes werden in der `initMap()` Funktion die gefundenen Ergebnisse der letzten Suchanfrage via jQuery `empty()` gelöscht. Danach werden einige Variablen definiert. Zunächst wird von der Google Maps Klasse `Geocoder` eine neue Instanz des Objektes erstellt. Im nächsten Schritt werden für die Postleitzahl, den Ort und den/die Experten/Expertin oder Einrichtung die Werte aus den Eingabefeldern via jQuery `val()` Funktion gelesen und in die jeweilige Variable gespeichert. Damit die Karte beim ersten Laden nicht leer ist, werden als Fallback Parameter die Postleitzahl und der Ort mit „1010 Wien“ vorausgefüllt. Infolgedessen werden die beiden Variablen zu einer Adresse zusammengefasst und anschließend dem zuvor erstellten Geocoder Objekt in der Funktion `geocode({'address': address})` übergeben. Daraufhin werden von der angegebenen Adresse die Latitude und Longitude Koordinaten ermittelt und in der Variablen `coordinates` gespeichert.

Im Anschluss wird eine neue Instanz des Objektes Map erstellt und an den in Listing 31 (siehe Seite 80) angegebenen Identifier `map` übergeben. Als Parameter werden die zuvor erstellten Koordinaten (`coordinates`) und der Zoomfaktor (`zoom`) überreicht sowie das Zoomen deaktiviert (`scrollwheel: false`). Sollte der/die UserIn über die Karte mit der Maus scrollen, wird die Karte dadurch nicht automatisch rein- oder rauszoomt. Abschließend wird ein Objekt erzeugt (`request`), welches die Koordinaten, den Radius der Suche sowie den Wert der Suchanfrage (Einrichtung/Experten) enthält. Dieses `request` Objekt wird dann an die Methode `textSearch` des zuvor erstellten `PlacesService` Objektes übergeben.

Listing 33. Initialisierung der Places API (eigener Code)

```
service.getDetails(place, function(place, status) {
  if (status === google.maps.places.PlacesServiceStatus.OK) {
    var marker = new google.maps.Marker({
      map: map, position: place.geometry.location, icon: image
    });
    var website = place.website;
    if(website === undefined) {website = "Keine Webseite angegeben";}
    else {
      website = "<a href="+place.website+" ... >"+place.website+"</a>
    }
    ...
    $('#results .row').append("<div class='col-md-6'>
     <h2>"+ place.name +"</h2>...
     <p>Adresse: "+ place.formatted_address +"<br></p>...</div>"
    );
    google.maps.event.addListener(marker, 'click', function() {
     infowindow.setContent("<div class='d-flex flex-row'><div><img
src="+place.photos[0].getUrl({'maxWidth': 120, 'maxHeight':
120})+"></div>...</div></div>");
     infowindow.open(map, this);
    });
  }
});
```

Das in Listing 32 (siehe Seiten 80/81) erstellte `PlacesService` Objekt wird nun dazu benutzt, um weitere Informationen vom zurückgegebenen Ort abzufragen. Dafür wird die Methode `getDetails()`[30] verwendet. Hier ist es wichtig, den Ort (`place`) mit in der Funktion zu übergeben, um auf die Details des Ortes zugreifen zu können. Nachdem der Statuscode des Service mit `OK` zurückgeliefert wurde, wird zunächst eine neue Instanz des Marker Objektes erzeugt. Dem Marker Objekt werden drei Parameter übergeben: die Karte (`map`), die `geometry.location` welche den Längen- und Breitengrad des Ortes beinhaltet, sowie das `icon` des Markers, das auf der Karte angezeigt wird. Hier wurde eigens ein angepasstes Icon entwickelt, damit die Suche im CI (Corporate Identity) der Arthrose Plattform ist.

Anschließend wird die Webseite des zurückgelieferten Objektes in einer Variablen gespeichert. Wenn eine Webseite angegeben ist, wird für die Webseite ein Link erzeugt, welche in einem neuen Tab geöffnet werden kann. Sollte die Webseite jedoch keinen Wert enthalten, wird ein kurzer Hinweis ausgegeben, dass „Keine Webseite angegeben“ ist. Im nächsten Schritt werden die Angaben im `place` Objekt dazu genutzt, um unterhalb der Karte die Suchergebnisse anzuzeigen. Dazu wird die jQuery Funktion `append()` verwendet. Mit dieser Funktion werden

[30] Details und Dokumentation: https://developers.google.com/maps/documentation/javascript/places?hl=de#place_details

an die Spalten des Identifiers `#results` der Name, die Adresse die Webseite, die Telefonnummer sowie die Öffnungszeiten angehängt und ausgegeben. Zuletzt wird noch ein Klick Event auf den erzeugten Marker gesetzt. Dafür werden in das `InfoWindow` Objekt mit Hilfe der Funktion `setContent()` die Daten des jeweiligen Ortes platziert. Im Unterschied zu den Suchergebnissen unter der Karte, wird beim Pop-Up zusätzlich ein kleines Vorschaubild (`place.photos[0].getUrl`) angezeigt, die Öffnungszeiten werden jedoch weggelassen.

Herausforderungen

Bei der Umsetzung dieses Features und Implementation der Google Places API sind zwei Herausforderungen aufgetreten. Das erste Problem fand sich bei Realisierung der Suche. Die Google Places API Web Service beinhaltet nur eine begrenzte Anzahl an geladenen Karten pro Tag[31]. Diese ist anfänglich auf 1.000 Stück pro Tag beschränkt. Bei Testzwecken während der Umsetzung kam es einmal vor, dass diese Anzahl erreicht wurde und keine Ergebnisse mehr geladen werden konnten. Die Beschränkung kann jedoch kostenlos auf bis zu 150.000 Anfragen pro Tag erhöht werden, dafür muss jedoch die Identität via Kreditkarte verifiziert werden. Sollte auch dieses Kontingent erschöpft sein, muss eine Lizenz erworden werden ("Nutzungsbeschränkungen und Abrechnung | Google Places API Web Service," 2017). Sollte die Plattform realisiert werden, ist eventuell mit einer höheren Anzahl an Anfragen pro Tag zu rechnen, daher sollte vorab über eine Alternative nachgedacht oder das kostenpflichtige „Premium" Modell der Places API in Betracht gezogen werden.

Ein weiteres Problem stellte die unterschiedliche Anzahl an gefundenen Ergebnissen dar. Die zurückgelieferte Anzahl an Marker, welche auf der Karte platziert werden können, sind auf 20 Stück beschränkt. Trotz ausführlicher Recherche konnte keine Lösung gefunden werden, diese Anzahl zu erhöhen. Die zurückgelieferten Daten für die Darstellung der Ergebnisse unter der Karte reichen nicht komplett aus, da nur der Name und die Adresse des Ortes zur Verfügung stehen. Um mehr Informationen des Ortes zu erhalten, wurde auf die bereitgestellte `getDetails()` Funktion zugegriffen. Diese liefert jedoch ausschließlich jene Ergebnisse, welche vollständig alle Daten enthalten. Daher kommt es fast ausnahmslos zu der Situation, dass die Markeranzahl mit den Ergebnissen der Suche stark variieren. Dies könnte letztendlich zur Verwirrung des/der Users/Userin führen. Man sollte hier eine einheitliche Lösung des Problems anstreben und die Anzahl der Ergebnisse in beiden Bereichen auf die gleiche Anzahl festlegen.

[31] Nutzungsdetails: https://developers.google.com/places/web-service/usage?hl=de

5.2.2.2 YouTube API für Übungen

Ähnlich wie bei der Google Places API im vorherigen Kapitel beschrieben, muss auch für die YouTube API ein Key für die Anwendung generiert werden. Zum Testen der YouTube API stellt Google zudem einen API Explorer[32] für EntwicklerInnen zur Verfügung. Nachdem der API-Key generiert wurde, kann eine Suchanfrage an die API erzeugt werden. Das nachfolgende Listing 34 zeigt eine mögliche Standardsuchanfrage.

Listing 34. YouTube Standardsuchanfrage (eigener Code)

```
https://www.googleapis.com/youtube/v3/search?key=AIza...&part=snippet&
q=Arthrose
```

Die Suchanfrage an die API benötigt nur zwei Pflichtparameter. Den erzeugten Schlüssel (`key`) und den Parameter `part`. Je nach Angabe des `part` Wertes können EntwicklerInnen auf unterschiedliche Werte des Videos zugreifen und diese ausgeben lassen. Der letzte Parameter `q` ist optional und gibt an, nach welchem Begriff gesucht werden soll.

Bei der Recherche relevanter Plattformen wurde u.a. die App „Arthrose Tagebuch“ untersucht (siehe Kapitel 2.2.4). Diese App stellt ihre Videos zusätzlich auf einem öffentlich zugänglichen YouTube Channel bereit[33], welcher unterschiedliche Videos mit Schwerpunkten zu verschiedenen Arthrosearten (Finger/Hand, Knie, etc.) anbietet, die für den abschließenden Nutzertest verwendet wurden. Das Listing 35 verdeutlicht, wie die Videos dieses Channels in die Arthrose Plattform eingebettet wurden.

Listing 35. Einbindung YouTube Videos via AngularJS Controller (eigener Code)

```
app.controller('exerciseController', ['$scope', '$http',
function($scope, $http) {
   var url = https://www.googleapis.com/youtube/v3/search?,
   key = "AIzaSyAGm...", order = "title", maxResults = 6,
   channelId = "UCjuMPQudaiZ3v5R99j4sWTg", part = "snippet";
  //finger hand osteoarthritis
  $http
  .get(url, { params: {
          key: key, order: order, maxResults: maxResults,
          channelId: channelId, q: "Hand", part: part
      }
   })
   .then(function (response) {
         var data = response.data;
         $scope.dataHand = data.items;
   }, function errorCallback(response) { console.log("NOT loaded");
  }); ....
}]);
```

[32] YouTube API Explorer: https://developers.google.com/apis-explorer/#p/youtube/v3/

[33] YouTube Channel für Übungen: https://www.youtube.com/user/zeelarthrose/videos

Zu Beginn des `exercise` Controllers werden zwei Abhängigkeiten (`$scope`, `$http`) eingebunden. Scopes sind ein Kernelement des AngularJS Frameworks. Das `$scope` Objekt enthält die Geschäftslogik der Anwendung, außerdem können Methoden und Eigenschaften hier definiert werden (Lerner, 2013, S. 21). AngularJS stellt zudem einige eingebaute Services zur Verfügung. Zu diesen gehört der `$http` Service. Dieser bietet einen einfachen Zugang zum XMLHttpRequest Objekt (XHR) des Browsers an. Anstatt mit dem XHR Objekt des Browser direkt zu interagieren, kann die $http API von AngularJS verwendet werden (Lerner, 2013, S. 157).

Nachdem die Abhängigkeiten definiert wurden, werden die einzelnen Bestandteile der YouTube URL als Variablen definiert und initialisiert. Für alle Arthrosearten wird nun via `$http` Service eine Anfrage an das XHR Objekt des Browsers gesendet. Dieser Service bietet eine `get()` Funktion an, an welche die Werte der Variablen mittels Parameter (`params`) übergeben werde. Der einzige Parameter, der sich bei der jeweiligen Anfrage ändert, ist die Arthroseart (hier: `q: „Hand“`). Die zurückgelieferte Antwort (`response`) wird dann in der `then()` Funktion weiterverarbeitet. Das `response` Objekt enthält neben dem angeforderten Daten-Objekt (`data{...}`) u. a. auch Informationen zum Status und anderen Parametern, welche zusätzlich abgefragt werden können. In diesem `data` Objekt befindet sich wiederum ein `items` Array (`data.items`), welches an das `$scope` Objekt (`$scope.dataHand`) übergeben wird und dessen Daten somit im Controller zur Verfügung stehen und darauf zugegriffen werden kann.

Herausforderungen

Es war nicht möglich, auf weitere Informationen des Videos zuzugreifen und diese auf der Übersichtseite darstellen zu lassen. Um dies zu erreichen, wäre es nötig gewesen, den Wert des Parameters `part` auf `contentDetails` zu setzen, um weitere Daten wie z.B. die Videolänge zu erhalten. Jedoch kann diese Anfrageart immer nur für ein Video verwendet werden, was für den Zweck der Suche nicht ausreichend gewesen wäre.

Ein weiteres Problem war die Reihenfolge der Videos, welche mit der Eigenschaft `order: title` nach dem Titel des Videos sortiert werden sollten. Da die Titel der Videos des externen Anbieters nicht konsequent vergeben wurden, konnte keine angemessene Reihenfolge der Videos erreicht werden, was auch die Testpersonen im abschließenden Nutzungstest bemängelten (El Aeraky, 2017).

5.2.3 Wahrung von Barrierefreiheit

Aufgrund der möglichen Einschränkungen der Testpersonen an Fingern bzw. Händen, wurden in Kapitel 3.1 technische Anforderungen im Bereich der Barrierefreiheit festgelegt, damit die Plattform auch ohne Maus oder mit technischen Hilfsmitteln bedienbar ist. Eine sehr wichtige Bedeutung kommt z.B.

der Anforderung nach einer Möglichkeit zu, Texte ohne assistierende Technik um bis zu 200% zu vergrößern, ohne dass dabei die Funktionalität der Anwendung verloren geht. Dies wurde auch im abschließenden Nutzungstest nochmals deutlich, in dem zwei von sechs Testpersonen angegeben haben, dass ihnen die Standardschriftgröße zu klein ist und diese nach einer Vergrößerungsmöglichkeit gefragt haben (El Aeraky, 2017, S. 110). Mit dem Tastaturkürzel „Strg und +“ (Mac: „Cmd und +“) kann die Schriftart beliebig vergrößert werden. Abbildung 13 zeigt die Arthrose Plattform mit einer 200% Text-Vergrößerung im Google Chrome Browser.

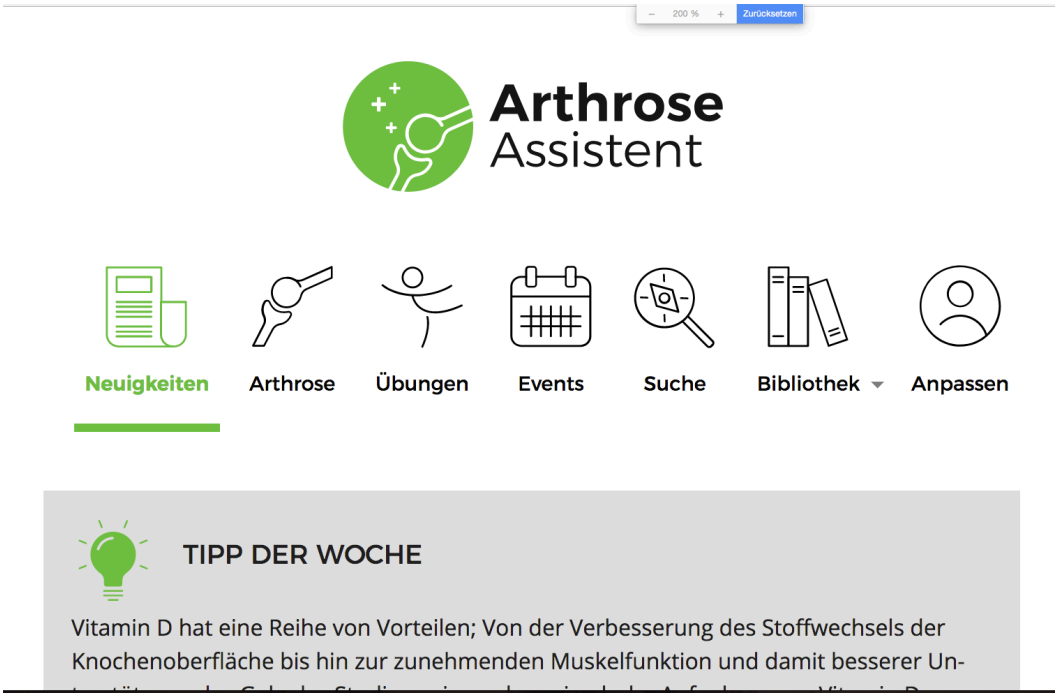

Abbildung 13. Screenshot der Vergrößerung auf 200% (eigene Abbildung)

Die korrekte und noch lesbare Darstellung der Plattform bei 200% Vergrößerung, wird durch die in den Stylesheets gesetzten Breakpoints gewährleistet. Die Anwendung passt sich, ähnlich wie bei kleinen Bildschirmauflösungen, an die Auflösung des Endgerätes an. Durch das Vergrößern der Schriftart springt die Arthrose Plattform je nach Bildschirmauflösung des Monitors oder Laptops in ein anderes „Layout“ und erzwingt dadurch eine andere Darstellung für die vergrößerten Elemente.

Ein weiterer wesentlicher Punkt bei der barrierefreien Realisierung der Plattform war, dass redundante Blöcke bei der Nutzung der Anwendung ohne Maus von UserInnen übergangen werden können. Ist der/die UserIn auf die Nutzung der Tastatur angewiesen, können zwei Sprungmarken genutzt werden, um schneller durch die Anwendung zu navigieren (siehe Abbildung 14, Seite 87).

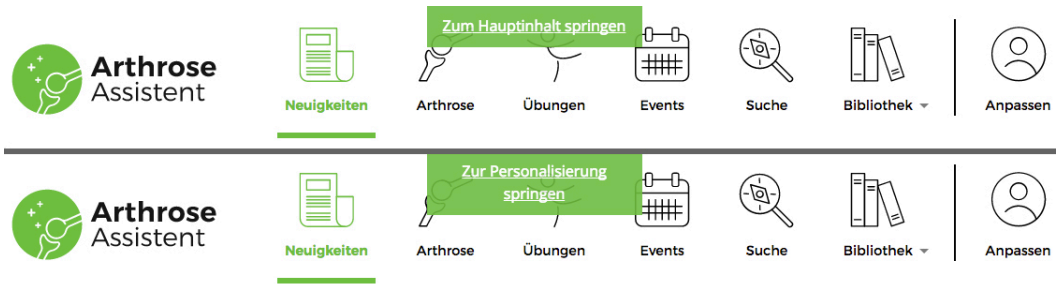

Abbildung 14. Screenshot der möglichen Sprungmarken (eigene Abbildung)

Sprungmarke eins „Zum Hauptinhalt springen“, übergeht die Punkte der Navigation und springt direkt zum HTML Element `<main>`, wo sich der Hauptinhalt der Anwendung befindet. Die zweite Sprungmarke „Zur Personalisierung springen“, ermöglicht es auch, die Navigationselemente zu umgehen und setzt den Fokus direkt auf das „Anpassen“ Symbol am Ende der Navigation. Um die Anwendung weiter für UserInnen, die auf alternative Ein- und Ausgabehilfen angewiesen sind, anzupassen, wäre es möglich, dass sich beim Sprung auf das „Anpassen“ Symbol, auch gleich das Personalisieren Pop-Up öffnet. Der Einsatz und Nutzen dieser Funktion müsste jedoch mit der Zielgruppe abgeklärt werden.

Herausforderungen

Bei der Wahrung der technischen Anforderungen und der damit verbundenen Einhaltung des Konformitätslevels AA sind gleichwohl auch ein paar Herausforderungen/Probleme aufgetreten.

Farbkontrast

Die WCAG Richtlinie „1.4.3 Contrast (Minimum)“ fordert für die visuelle Darstellung von Schriftfarbe zu Hintergrundfarbe ein Kontrastverhältnis von mindestens 4,5:1. Um den Farbkontrast von Webseiten zu testen, stehen unzählige kostenlose online Kontrast Checker zur Verfügung[34]. Abbildung 15 zeigt den durchgeführten Kontrastcheck der Arthrose Plattform (siehe Seite 88).

[34] http://contrastchecker.com

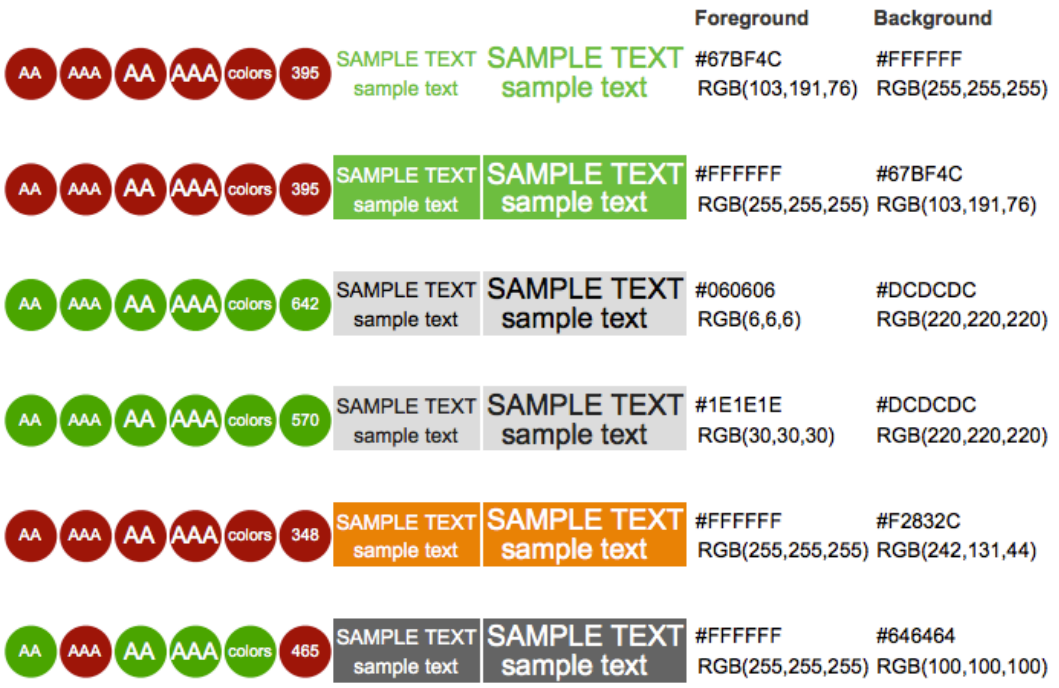

Abbildung 15. Screenshot Farbkontrast Checker (Quelle: http://contrastchecker.com/)

Bei der Primärfarbe Grün und der Sekundärfarbe Orange der Arthrose Plattform, konnte der vorgeschriebene Kontrastwert nicht eingehalten werden. Alle anderen Farbkontraste der Plattform erfüllen immerhin das Konformitätslevels AA. Zumindest bei der Navigation kann das zu geringe Kontrastproblem etwas abgemildert werden, da die grüne Schrift zusätzlich mit einem Symbol darüber und einen dicken Balken darunter die Schrift unterstützt.

Youtube Videos enthalten keine Untertitel

Die vom externen Anbieter „Arthrose Tagebuch“ eingebundenen YouTube Videos, welche für die Darstellung der Übungen der jeweiligen Arthrose Art verwendet wurden, enthalten keine integrierten Untertitel. Die WCAG Richtlinie „1.2.2 Captions (Prerecorded)“ fordert jedoch den Einsatz von Untertitel in aufgezeichneten Video-Inhalten. Dies ist vor allem für die Zielgruppe der Arthrose Plattform wichtig, da sie oftmals zusätzlich zu körperlichen Beeinträchtigungen an Finger bzw. Hand auch Schwierigkeiten beim Hören haben. Daher sollten bei einer möglichen Realisierung der Anwendung, alternative Videos in Betracht gezogen werden.

Zur weiteren Überprüfung der Barrierefreiheit können unterschiedliche automatisierte Evaluationswerkzeuge herangezogen werden. Um die Barrierefreiheit des Prototyps zu bewerten, wurde die Seite von zwei „Accessibility

Checkern“ (WAVE[35] und AChecker[36]) auf Fehler hin untersucht. Deren Vorschläge wurden auf Machbarkeit und Relevanz überprüft und in den Prototypen eingearbeitet.

5.2.4 Browserkompatibilität

Bei der Konzeptüberprüfung Mitte Juni 2017 wurden die TeilnehmerInnen in einem Vorabfragebogen u.a. dazu befragt, welche Browser die Testpersonen verwenden, um Webseiten im Internet abzurufen (El Aeraky, 2017, S. 74). Aufgrund dieser Ergebnisse wurden acht Browser und deren Versionen definiert, für welche die Arthrose Plattform optimiert sein sollte (Stand August 2017):

- Microsoft Internet Explorer (IE): letzte Version 11
- Edge: aktuelle Version 15
- Firefox: aktuelle Version 55
- Google Chrome: aktuelle Version 61
- Apple Safari: aktuelle Version 10.1
- iOS Safari: aktuelle Version 10.3
- Chrome für iOS: aktuelle Version 60
- Chrome für Android: aktuelle Version 59

Zur Überprüfung der Browserkompatibilität wurde jeweils die Homepage der Anwendung sowie die Unterseite „Arthrose“ nach einer durchgeführten Personalisierung mit den definierten Browsern untersucht. In der Tabelle 10 (siehe Seite 90) werden die Probleme der einzelnen Browser gelistet und anschließend teilweise mit Bildern verdeutlicht. Hat ein Browser die Überprüfung ohne Probleme absolviert, wurde dies mit einem Minuszeichen in der Tabelle versehen.

[35] WAVE: http://wave.webaim.org

[36] AChecker: https://achecker.ca/checker

Tabelle 10. Vergleich der Browserkompatibilität

Browser	***Probleme***
Microsoft IE	*Unproportional dargestellte Bilder* *Symbole auf der Startseite werden nicht angezeigt* *Trotz Personalisierung werden alle Arthrosearten in der Auswahlbox dargestellt*
Microsoft Edge	*Unproportional dargestellte Bilder*
Firefox	-
Google Chrome	-
Apple Safari	*Das „Nach oben“ Scrollen Icon wird nicht angezeigt.* *Trotz Personalisierung werden alle Arthrosearten in der Auswahlbox dargestellt.*
iOS Safari	*Trotz Personalisierung werden alle Arthrosearten in der Auswahlbox dargestellt.*
Chrome für iOS	*Trotz Personalisierung werden alle Arthrosearten in der Auswahlbox dargestellt.*
Chrome für Android	-

Tabelle 10 zeigt, dass zwei Hauptprobleme bei der Überprüfung der verschiedenen Browser aufgetreten sind. Die Einrichtung des Profils und die damit verbundene Personalisierung hat zwar bei allen Browsern funktioniert (siehe Abbildung 16 rechts), will der/die UserIn bei den Arthrosearten jedoch in der Auswahlbox zwischen den eingestellten Arten wechseln, werden in der Auswahlbox trotzdem alle Arthrosearten angezeigt (siehe Abbildung 16 links).

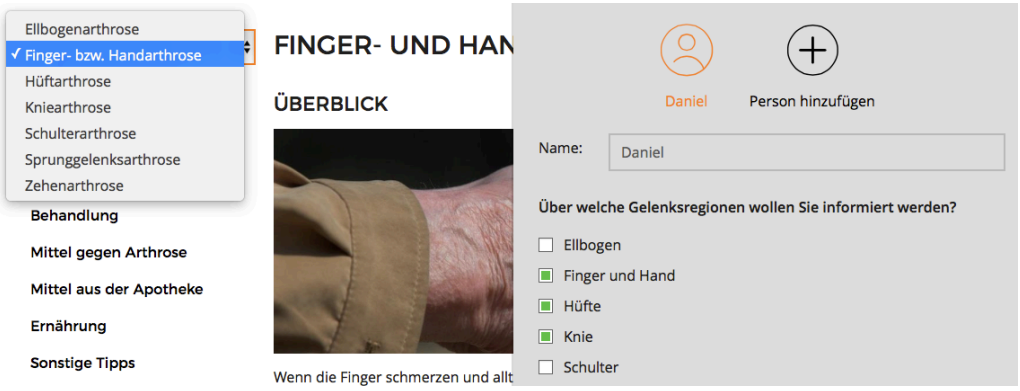

Abbildung 16. Screenshot Auswahl Arthrosearten; Apple Safari (eigene Abbildung)

Das zweite Problem trat ausschließlich bei den beiden Browsern Mircosoft IE und Edge auf. Um die dargestellten Bilder im vorgegebenen Platz zu positionieren, wurde die CSS Eigenschaft `object-fit: cover` verwendet. Mit dieser Methode wird spezifiziert, wie sich das Bild innerhalb der Box verhalten soll. Mit dem Wert `cover` soll das Seitenverhältnis des Bildes aufrechterhalten bleiben und so positioniert werden, dass es zu keinen Verzerrungen kommt. Diese Eigenschaft kennen die beiden oben angeführten Browser jedoch nicht[37]. Daher muss für dieses Problem ein Workaround gefunden werden. Eine Lösung wäre z.B. die Bibliothek Modernizr[38] zu verwenden. Diese erkennt, welche Funktionen die Browser unterstützen und bei welchen man gegebenenfalls mit Workarounds nachbessern muss.

Zudem bietet die Seite http://www.browser-update.org/ ein nützliches Features an, um BesucherInnen einer Seite auf eine eventuell nicht einwandfreie Nutzung der Webseite hinzuweisen. Bei welchen Browsern und Versionen diese Meldung angezeigt werden soll, kann je nach Anforderungen eingestellt werden. Der nach der Konfiguration generierte JavaScript Code muss am Ende der Seite eingebunden werden und UserInnen wird nach Aufrufen der Webseite am obenen Rand eine unaufdringliche Benachrichtigung angezeigt (siehe Abbildung 17).

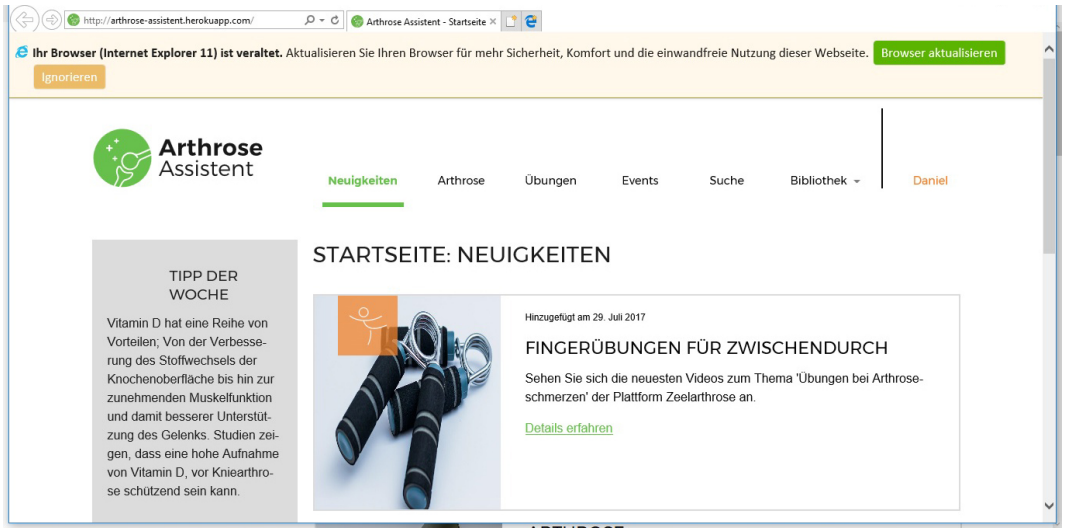

Abbildung 17. Hinweis auf veralteten Browser im Internet Explorer Version 11 (eigene Abbildung)

5.2.5 Offline-Fähigkeit

In Rahmen der Konzeptüberprüfung stellte sich weiterhin heraus, dass es drei von sieben Testpersonen als vorteilhaft empfinden würden, wenn sie die Arthrose

[37] Die Unterstützung von object-fit kann auf folgender Seite nachgelesen werden: http://caniuse.com/#search=object-fit

[38] Features Erkennung mit Hilfe von Modernizr: https://modernizr.com/

Plattform auch ohne Internetverbindung nutzen könnten (El Aeraky, 2017, S. 82). Diese nicht zu unterschätzende Anzahl, lässt darauf schließen, dass es eventuell öfter zu Situationen im Alltag der Zielgruppe kommt (z.B. in öffentlichen Verkehrsmitteln), in denen diese zwar gerne Informationen beziehen würden, aber keine Internetverbindung vorhanden ist, um ebendiese zu konsumieren. Um den Bedürfnissen der Zielpersonen gerecht zu werden, wurde eine Cache Manifest Datei erstellt und zu Beginn der Anwendung eingebunden (Listing 36).

Listing 36. Cache Manifest der Arthrose Plattform (eigener Code)

```
CACHE:
# rev 03

#sites
/arthrose
/fachbegriffe
..
#css
/assets/css/app.css
#images
/favicon.ico
/assets/images/logo.svg
...
#js
/assets/js/app.js
/assets/js/default.js
...
# Seiten, die nur online genutzt werden koennen
NETWORK:
*
FALLBACK:
/assets/images/ /assets/images/default.png
/ /offline;
```

Im ersten Abschnitt `CACHE` werden alle Dateien bzw. Seiten festgelegt, welche trotz fehlender Internetverbindung zur Verfügung stehen sollen. Dieser Abschnitt wurde aufgeteilt in die Seiten der Anwendung (`#sites`), die CSS (`#css`) und JavaScript Dateien (`#js`) sowie die Bilder (`#images`) die offline bereitgestellt werden. Bei den Seiten wurden hauptsächlich jene gelistet, deren Inhalte sich wenig bis gar nicht ändern. Es würde z.B. wenig Sinn machen, die Liste der Veranstaltungen offline abzuspeichern, da sich der Datenbestand regelmäßig ändern kann. Auch sollen die wichtigsten CSS und JavaScript Dateien sowie Bilder offline gespeichert werden, damit die gewohnte Struktur und Darstellung der Anwendung beibehalten wird.

Durch das Setzen von "*" im Abschnitt `NETWORK`, ist für das Beziehen der restlichen Dateien eine Verbindung zum Internet erforderlich. Im letzten Bereich `FALLBACK` werden alternative Dateien angegeben, für den Fall, dass eine Ressource offline nicht verfügbar ist. Wird ein Bild im Ordnerpfad `/assets/images` nicht gefunden,

wird stattdessen ein alternatives Ersatzbild, wie in Abbildung 18 (links) zu sehen ist, dargestellt. Fordert der/die UserIn eine Seite an, die nicht im Cache der Anwendung geladen ist ("`/`"), wird der/die UserIn auf die extra für Offlinezwecke eingerichtete Standardseite (`/offline`) weitergeleitet und ein Hinweis angezeigt, dass momentan nicht alle Seiten und Funktionen der Anwendung zur Verfügung stehen (siehe Abbildung 18, rechts).

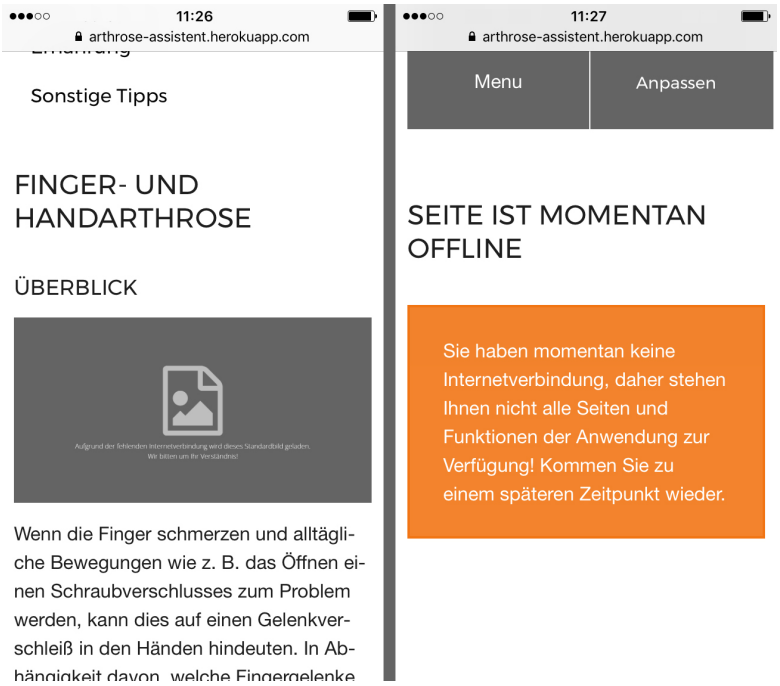

Abbildung 18. Fallback-Varianten für Bilder (links) und nicht gefundene Seiten (rechts) (eigene Abbildung)

Wird eine Cache Manifest Datei verwendet, um Inhalte offline abrufbar zu machen, kann es durchaus vorkommen, dass sich der Inhalt einer oder mehrerer Dateien ändert, der Name der jeweiligen Datei jedoch gleichbleibt. Durch das Caching wird infolgedessen trotzdem die alte Datei dargestellt. Dies kann verhindert werden, indem man die zu Beginn gesetzte Revisionsnummer (`#rev 03`) bei jeder Änderung um einen Wert erhöht (siehe Listing 36, Seite 92) (Hoffmann, 2012, S. 244). Wie bereits in Kapitel 3.2.4 erwähnt, müssen WebentwicklerInnen jedoch darauf achten, inwieweit diese Schnittstelle in Zukunft unterstützt wird oder ob alternativ bereits das Service Worker Manifest eingesetzt werden kann (siehe Kapitel 3.1.3).

5.2.6 Inhalte ausdrucken

Relevante Informationen der Plattform auszudrucken, hat sich im Rahmen der Konzeptüberprüfung als eine weitere, für die Testpersonen wichtige, Funktion herausgestellt. Fünf von sieben Testpersonen würden es als vorteilhaft empfinden, wenn man die Inhalte für den Druck optimieren könnte, damit man diese zu einem

späteren Zeitpunkt jederzeit griffbereit hat (El Aeraky, 2017, S. 83). Um den Bedürfnissen der Testpersonen zu entsprechen, wurden die druckbaren Inhalte via CSS optimiert (siehe Listing 37).

Listing 37. Auszüge aus der CSS Print-Datei (eigener Code)

```
@media print {
   @page {
      size: A4; margin: 0.5cm; line-height: $spacer*1.25;
   }
   aside, header, footer, img, ... {
      display: none;
  }
  main::before {
      display: block;
      content: "Hinweis: Sie sollten unsere Webseite regelmäßig
besuchen, da wir die Inhalte kontinuierlich aktualisieren und somit
manche Informationen zu einem späteren Zeitpunkt veraltet sein
könnten!";
      margin-bottom: $spacer;
      border: 2px solid $brand-primary;
      padding: $spacer/2;
   }
   .arthrose__topics {
      page-break-after: always;
   }
   [href^="http://"]::after {
       display: block;
       content: "Link: ("attr(href)")";
       position: absolute;
       font-size: .8rem;
    }
   ...
};
```

Anhand des Media Queries `@media print{}` zu Beginn wird dem Browser mitgeteilt, dass alle Eigenschaften innerhalb dieser Klammer nur dann ausgeführt werden, wenn die Inhalte ausgedruckt werden sollen. Um eine optimal für den Druck formatierte Ausgabemöglichkeit zu bieten, wurde von der W3C Gruppe eine eigens dafür definierte Spezifikation für gedruckte Medien („CSS Paged Media Module“) erstellt[39]. Diese Spezifikation enthält u.a. auch das Seitenmodel („page model“). Mittels `@page` kann auf die verschiedenen Aspekte des Seitenmodels wie z.B. die Abmessungen oder die Ausrichtung zugegriffen werden und diese mit Hilfe der Eigenschaft `size` gesetzt werden. Das Format des Ausdrucks kann somit beliebig angepasst werden. Als Layout wird hier DIN A4 gesetzt und auf allen Seiten ein Rand von einem halben Zentimeter gewählt. Im nächsten Schritt werden alle Elemente der Seite gelistet, die nicht ausgedruckt werden sollen. So ist es z.B.

[39] W3C Spezifikation

unnötig, die Navigation, Bilder oder den Footer jedes Mal auszudrucken. Durch das Ausblenden dieser Elemente kann Druckfarbe eingesparrt werden.

Um UserInnen auf die kontinuierlich aktualisierenden Inhalte hinzuweisen, wird zu Beginn des Ausdrucks ein Hinweis angezeigt, dass die Inhalte ggf. veraltet sein könnten und der/die UserIn, die Seite regelmäßig aufsuchen soll. Dieser Hinweis wird mit Hilfe der Eigenschaft `content`, am Anfang der DIN A4 Seite im Pseudoelement `:before` des HTML Elements `<main>` platziert (siehe Abbildung 19).

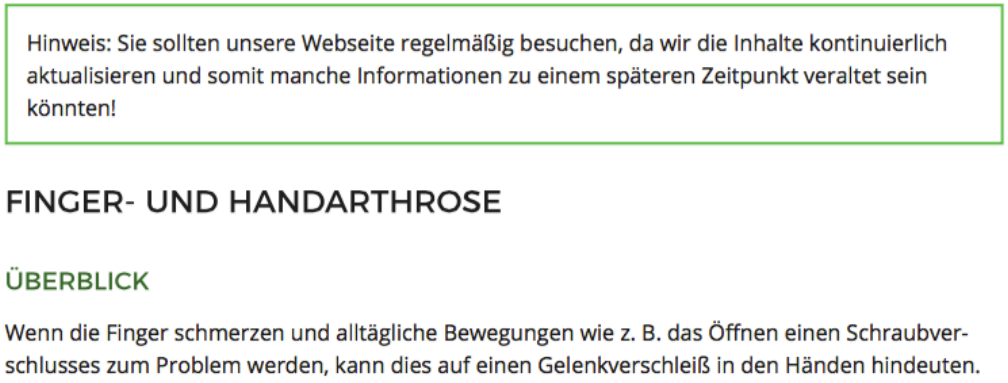

Abbildung 19. Hinweis zu Beginn eines Ausdrucks (eigene Abbildung)

Die nächste CSS Regel betrifft nur den Bereich der Arthrosearten. Der Inhalt jedes Menüpunktes der Subnavigation ist mit der Klasse `arthrose__topics` ausgezeichnet. Mit Hilfe der Eigenschaft `page-break-after: always` wird nach jedem Bereich ein Seitenumbruch im Ausdruck erzwungen. Dies soll die Strukur verbessern und die Lesbarkeit für die UserInnen erhöhen. Die letzte angeführte CSS Regel `href^="http://"]::after{}` spricht alle externen Links an, die mit http beginnen. Dadurch wird der externe Link via `href` Attribut als Verlinkung in die Eigenschaft `content` geschrieben und nach dem eigentlichen Linktext dargestellt. Die Anzeige der externen Links soll es UserInnen ermöglichen, jederzeit Details zu den Verlinkungen nachzusehen.

5.3 Herausforderungen und Probleme

Während der Realisierung des Prototyps kam es in unterschiedlichen Phasen der Entwicklung zu Herausforderungen und Problemen, worauf im nachfolgenden Abschnitt genauer eingegangen wird.

LocalStorage vs. IndexedDB

Vor der Umsetzung der Personalisierung war unklar, welche lokale Speichermöglichkeit für die Umsetzung herangezogen werden soll. Da von Seiten des Autors bisher keine Erfahrung mit beiden Speicherverfahren gesammelt werden konnten, wurde zu Testzwecken die geplante Funktionalität zunächst mit Local Storage und anschließend mit IndexedDB umgesetzt. Bei der Umsetzung mit Local Storage zusammen mit der Library jQuery konnten zwar schnell sichtbare Erfolge erzielt werden, mit zunehmender Anzahl an Funktionen wurde die Realisierung jedoch rasch unübersichtlich und komplex.

Auch die anfängliche Implementation mit IndexedDB und jQuery konnte die Anforderungen nicht erfüllen. So war es z.B. nicht möglich die hinterlegten Daten von UserInnen dauerhaft über alle Seiten der Anwendung hinweg abzufragen und darzustellen, um gezielt Inhalte ein- oder auszublenden. Erst die Erstellung eines Service mit AngularJS stellte eine adequate Lösung für den abschließenden Nutzungstest dar.

Es wird daher empfohlen, vor dem Einsatz eines der beiden Speicherverfahren, den Umfang und den Einsatz der Technologie abzuklären. Die Web Storage Variante Local Storage ist eher geeignet, um kleine Datenmengen zu speichern. Jedoch ist Indexed DB, wie in Kapitel 3.3.4 beschrieben, besser dafür geeignet, komplexe, strukturierte Daten zu verarbeiten, was bei der Umsetzung des Prototyps vorteilhaft war.

IndexedDB

Zwei von sieben Testpersonen gaben bei der Konzeptüberprüfung an, dass es mehrere Personen im Haushalt mit Arthrose gibt und in diesen Haushalten oftmals nur ein Computer oder Laptop zur Verfügung steht sowie auch der gleiche Browser benutzt wird. Daher sollte es möglich sein, mehrere Profile anzulegen und zwischen diesen zu wählen (El Aeraky, 2017, S. 75). Das Anlegen und Aktualisieren von UserInnen konnte mit Hilfe von IndexedDB und AngularJS umgesetzt werden, jedoch konnte keine Lösung gefunden werden, wie man zwischen zwei oder mehreren Profilen wählen konnte. Um dieses Problem zu beheben, wurden zwei Lösungsansätze ausprobiert, welche beide allerdings keinen Erfolg hatten:

- Beim Anlegen neuer UserInnen, werden die bisherigen UserInnen mit Hilfe einer Variablen auf „inaktiv“ gesetzt und der/die gerade neu angelegte UserIn auf „aktiv“. Es gibt jedoch keine Möglichkeit, mehrere Datensätze gleichzeitig zu ändern.
- Auch wäre es denkbar gewesen, einen zusätzlichen Button im Profil anzubieten, mit dem man ein Profil „aktiv“ setzen kann. Man hätte dann z.B. den Zeitpunkt des „Aktivsetzens“ in dem Profil als Timestamp speichern können und mit dem Timestamp der anderen Profile vergleichen können, aber auch hier gab es das gleiche Problem, wie im ersten Lösungsansatz beschrieben, dass nicht mehrere Datensätze gleichzeitig geändert werden konnten.

Markierte Inhalte abspeichern

Um die Anwendung zusätzlich zum Profil weiter individuell anzupassen, war es laut Screendesign von Shadja El Aeraky (2017) angedacht, einzelne Inhalte, wie Übungen, Events oder Tipps, zu markieren und bei Bedarf nur diese anzuzeigen (siehe Abbildung 20).

Abbildung 20. Markieren Funktion, Beispiel Übungen (eigene Abbildung)

Abbildung 20 zeigt jeweils eine markierte Übung (grüne Stecknadel) und eine unmarkierte Übung (hellgraue Stecknadel). Nach dem Klick auf den Button „Markiert“ links in der Abbildung, sollen nur die mit einer grünen Stecknadel markierten Übungen angezeigt werden.

Leider konnte für diese Anforderung bis zum abschließenden Nutzungstest keine zufriedenstellende Lösung gefunden werden, sodass diese Funktion zwar vorhanden, aber nicht voll funktionsfähig ist.

6 Fazit

Diese Arbeit untersucht die gegenwärtigen Probleme aktueller Arthrose Plattformen. In einem Vergleich werden diese gegenübergestellt und deren Vor- und Nachteile zusammengefasst. Außerdem wurde ermittelt, welche technischen Anforderungen die Arthrose Plattform aufgrund der Bedürfnisse der Zielgruppe erfüllen muss und welche Technologien und Frameworks bei der Erstellung der Plattform hilfreich sind.

Zu Beginn der Diplomarbeit wurden folgende Forschungsfragen gestellt:

- Welche technischen Anforderungen ergeben sich aus den Anforderungen der Zielgruppe hinsichtlich einer interaktiven Kommunikationsplattform?
- Welche Technologien und Frameworks eignen sich am Besten für die prototypische Umsetzung einer interaktiven Plattform (basierend auf den Anforderungen der Zielgruppe) und gewährleisten, bei Bedarf, eine einfache Erweiterung?
- Welche gültigen Accessibility Guidelines sind für die Zielgruppe relevant und wie lassen sich diese am Besten umsetzen?

Auf Basis der zu Beginn der Diplomarbeit durchgeführten Erhebungen konnten die Anforderungen der Zielgruppe an eine interaktive Plattform verifiziert werden. Infolge dieser Bedürfnisse wurden Nutzungsszenarien und daraus technische Anforderungen an die Arthrose Plattform abgeleitet. Diese ergaben unter anderem, dass die Anwendung besonders gut auf größeren Bildschirmen verwendbar sein muss und wichtige Informationen schnell erreichbar und für den Druck optimiert sein müssen. Zudem sollte die Anwendung für ältere Browser(versionen) optimiert sein und körperliche Einschränkungen der Zielgruppe bedacht werden, damit diese auch bei Bedarf mit technischen Hilfsmittel oder rein mit Tastatur verwendbar ist.

Um herauszufinden, welche CSS und JavaScript Frameworks/Libraries sich am Besten für die prototypische Umsetzung einer interaktiven Plattform auf Basis der Anforderungen der Zielgruppe eignen, ist es wichtig, vorab Kriterien für die Auswahl aufzustellen. Es wurden daher elf Auswahlkriterien definiert. Fünf Kriterien (Prototyping/Coding, Lernkurve, Codeaufblähung, Community/Support und Dokumentation) gelten für beide Gruppen, zwei Kriterien (Einsatz von Preprozessoren und Nutzung von JavaScript Funktionen) nur für CSS Frameworks und drei Kriterien (Performance, Zukunftsfähigkeit/Lebensdauer sowie notwendige Kenntnisse bzw. Build Tools) ausschließlich für JavaScript Frameworks/Libraries. Diesen Kriterien wurde zunächst eine Gewichtung von sehr

wichtig (Wert 3) bis kaum wichtig (Wert 1) zugewiesen. Es muss unterschieden werden, ob es sich bei der Umsetzung um einen Prototyp oder um ein marktreifes Produkt handelt, denn je nach Einsatz wird dem Kriterium eine andere Bedeutung zugeordnet.

Anschließend wurden die Kriterien überprüft und mit einem Erfüllungswert von 1 bis 4 versehen. Für die Berechnung des Endwertes wurde der Erfüllungswert eines Kriteriums mit dem Wert der Gewichtung multipliziert und für alle Kriterien addiert. Beim Vergleich der fünf ausgewählten CSS Frameworks konnte Bootstrap das beste Ergebnis erzielen. Dies liegt daran, dass Bootstrap eine gute und umfangreiche Dokumentation mit vielen Beispielen bereithält, die Anzahl der Hilfestellungen bei Problemen groß sowie ein schnelles Prototyping und Coding durch inkludierte Komponenten (Navigation, Buttons, etc.) möglich ist. Außerdem können die Komponenten mit Hilfe von JavaScript einfach erweitert und mit Interaktivität versehen werden.

Bei der Gegenüberstellung der fünf ausgewählten JavaScript Frameworks/Libraries ging jQuery als Sieger hervor, gefolgt von AngularJS. jQuery hat von den ausgewählten Frameworks/Libraries die längste Lebensdauer (Veröffentlichung ca. Mitte 2006, siehe Anhang A) und kann damit auf eine große Community und Support zurückgreifen. Die Dokumentation ist ausführlich und bietet Codebeispiele und Hilfestellungen bei Problemen. Es sind keine zusätzlichen Build-Tools notwendig, um mit dem Entwickeln zu beginnen. Zudem ist der Library Code klein und die Lernkurve steil, wenn man bereits gute JavaScript Kenntnisse besitzt.

Für das Styling der prototypischen Umsetzung wurde daher Bootstrap verwendet und für die Interaktionselemente auf der Plattform wurden die jQuery Library und das AngularJS Framework herangezogen.

Basis für die Untersuchung der Barrierefreiheit waren die WCAG 2.0 Richtlinien. Aufgrund dieser wurden zusätzlich technische Anforderungen im Bereich der Barrierefreiheit erarbeitet, die für die Plattform relevant waren (siehe Kapitel 4.1). Mit Hilfe der oben genannten Frameworks wurden diese umgesetzt und von automatisierten Evaluationswerkzeugen überprüft (siehe Kapitel 5.2.3). Diese kostenlosen Werkzeuge können jedoch nicht alle Barrieren finden, daher sollte man nicht auf die manuelle Überprüfung der Barrierefreiheit verzichten.

Lessons Learned

Abschließend können die Erkenntnisse der prototypischen Umsetzung in folgende allgemeine Empfehlungen zusammengefasst werden:

1. Bei der technischen Umsetzung einer Online-Plattform müssen die Anforderungen und Bedürfnisse der Zielgruppe berücksichtigt sowie deren Feedback eingearbeitet und laufend evaluiert werden.

2. Für das Styling eines interaktiven Prototyps eignet sich besonders gut das CSS Framework Bootstrap, da Bootstrap gut dokumentiert ist, zahlreiche Codesnippets das schnelle Prototyping fördern und die angebotenen Komponenten einfach mit Interaktivität erweitert werden können.
3. Um weitere Interaktionselemente eines Prototyps umzusetzen, bietet sich die JavaScript Library jQuery an. Hier sind vor allem die steile Lernkurve, der kleine Datenkern, die lange Lebensdauer sowie die detaillierte Dokumentation von Vorteil.
4. Clientseitige Speicherschnittstellen stellen eine gute Alternative zu serverseitigen Datenbanken dar, wenn nur wenige unpersönliche Daten von UserInnen gespeichert werden müssen, um eine Personalisierung zu erreichen. Zudem entfällt der oft unangenehme Prozess der Registrierung.
5. Potentielle körperliche Einschränkungen der Zielgruppe sollten bedacht werden und somit bestehende Richtlinien zur Barrierefreiheit im Web bei der Programmierung eingehalten werden. Zur Einhaltung können Hilfsprogramme herangezogen werden, welche die manuelle Überprüfung unterstützen.
6. Der Zugriff auf Daten externer Dienste unterliegt geringfügigen Einschränkungen, welche vorab bei der Umsetzung einer marktreifen Anwendung berücksichtigt werden sollten. So ist z.B. bei Google Places, die Anzahl der geladenen Karten auf 1.000 Anfragen pro Tag begrenzt. Diese Anzahl kann eventuell ein Problem verursachen und mit Hilfe der Angabe einer Kreditkarte gelöst werden. Will man hingegen mit der YouTube API mehrere Videos darstellen und zusätzlich Details zu einem Video abrufen, müssen zwei Aufrufe an die API gemacht werden, da die Standardsuche nur wenig Informationen über das Video liefert.

7 Ausblick

Die positiven Ergebnisse der finalen Konzeptüberprüfung haben gezeigt, dass sich nahezu alle Testpersonen vorstellen könnten, die in Kapitel 5 ausführlich beschriebene Arthrose Plattform in Zukunft zu verwenden und es somit einen Bedarf gibt, die Anwendung über diese Diplomarbeit hinaus weiter zu verbessern (El Aeraky, 2017, S. 220, 225). Die nachfolgenden Punkte sollen daher Ansätze liefern, wie sich die Plattform mittel- oder langfristig entwickeln könnte.

Wartung und Pflege des Inhalts

Die wichtigste Frage lautet, wie die Plattform in Zukunft gewartet werden soll und wie neue Inhalte aufgenommen werden können. Da die Plattform sehr viele unterschiedliche Informationen bereitstellt, bietet sich zur Verwaltung und Pflege der Seite ein klassisches CMS (Content Management System) an, in dem Inhalte erstellt, freigeschalten oder aktualisiert werden können. Da unzählige kostenlose CMS Systeme zur Verfügung stehen, müssen vorab Kriterien festgelegt werden, welche das CMS erfüllen muss, um für eine mögliche Verwendung in Frage zu kommen.

Die Wartung der Seite sowie die gleichzeitige Erstellung von hochwertigen Inhalten bedarf einer Zusammenarbeit unterschiedlicher Organisationen (z.B. Agentur, medizinische Fachkräfte, etc.). Es müssen Möglichkeiten geschaffen werden, wie man zusätzliche externe Inhalte einpflegen kann. Daher ist es denkbar, dass externe Personen neue Inhalte zu Themen wie Literatur, Videos, Veranstaltungen, etc. über ein Antragsformular auf der Arthrose Plattform einreichen können (siehe Abbildung 21, Seite 102).

Abbildung 21. Beispielscreen für eine mögliche Einreichfunktion für Inhalte (eigene Abbildung)

In diesem möglichen Formular wird zunächst die Art der Einreichung ausgewählt. Je nach Auswahl werden weitere Angaben benötigt, um das Formular abzusenden. Sind alle Pflichtfelder ausgefüllt worden und wurde das Formular abgesendet, ist es denkbar, dass ein/e MitarbeiterIn der Plattform eine E-Mail mit den Details erhält. Zusätzlich wurde in der Datenbank des Systems ein Eintrag erzeugt, der jedoch noch freigeschalten werden muss. Wurden die Daten von dem/der MitarbeiterIn erfolgreich kontrolliert, kann der Datensatz im CMS je nach Prüfung abgelehnt oder freigegeben werden.

Web Push Notifications

Notifications werden seit geraumer Zeit erfolgreich auf mobilen Endgeräte eingesetzt, um UserInnen über Neuigkeiten jeglicher Art zu informieren. Das W3C arbeitet gerade an einem Entwurf für eine Push API, um Push Nachrichten zu senden, auch wenn die Webseite, für welche die Push Nachricht aktiviert wurde,

gerade nicht geöffnet ist[40]. Die momente Browserunterstützung der Push API ist noch relativ gering[41].

Nichtdestotrotz wäre es denkbar, diesen Service zukünftig zu nutzen, um UserInnen über Neuigkeiten zu informieren. Es wäre daher vorstellbar, bei der Personalisierung zusätzlich abzufragen, ob der/die UserIn regelmäßig auf Neuigkeiten via Push Nachrichten hingewiesen werden will. Nebenbei hätte dies den Effekt, dass der/die UserIn die Plattform kontinuierlich besucht und so zu einem aktiven Konsumierenden wird.

Möglicher Einsatz von tragbaren Computersystemen

Bei der ersten Erhebung Anfang Mai wurden die Testpersonen im Rahmen von Interviews befragt, ob es aus ihrer Sicht Bedenken gegenüber tragbaren Computersystemen wie z.B. Smartwatches oder Smartphones gibt, die Gesundheitsdaten aufzeichnen und auswerten.

Die Testpersonen haben keinerlei Bedenken gegenüber tragbaren Computersystemen, wollen aber z.B. nicht an die Medikamenteneinnahme erinnert werden (El Aeraky, 2017, S. 59). Folgende alternative Use Cases, welche nicht der Zielgruppe vorgestellt wurden, wären daher für tragbare Systeme denkbar:

Use Case 1: „Person A stellt im persönlichen Profil auf der Arthrose Plattform ein, dass diese regelmäßig um 12.30 Uhr eine Benachrichtigung mit einer kurzen Übung zum Thema Fingerarthrose auf dem Smartphone erhalten möchte. Diese Übung kann bequem während der Mittagspause durchgeführt werden“.

Use Case 2: „Person B richtet sich eine Benachrichtigung für das Smartphone auf der Arthrose Plattform ein, die sie bekommt, wenn eine neue Veranstaltung im Bundesland Wien veröffentlicht worden ist. Diese kann anschließend in den Kalender des Smartphones übernommen werden“.

Das Ziel der Plattform könnte sein, dass, wie in Use Case 1 beschrieben, die Zielgruppe durch regelmäßige Benachrichtigungen an die Plattform gebunden wird und diese so nicht in die Versuchung kommt, mit den regelmäßigen Übungen aufzuhören, was ein zentrales Problem beim Umgang mit der Erkrankung ist (El Aeraky, 2017, S. 59).

[40] W3C Push API Spezifikation: https://www.w3.org/TR/push-api/

[41] Die Unterstützung kann auf folgender Seite nachgelesen werden: http://caniuse.com/#search=Push%20API

Einbindung sozialer Netzwerke

Ein weiterer möglicher Ansatz die Plattform zu erweitern, wäre die Einbindung sozialer Netzwerke, wie Facebook oder Twitter. Zwar haben bei der ersten Befragung vier von zehn Testpersonen angegeben, dass sie Facebook nicht nutzen und auch kein Interesse daran haben das Netzwerk zu verwenden (El Aeraky, 2017, S. 59), dennoch könnte es eine Möglichkeit geben, die Inhalte verschiedener Seiten auf sozialen Netzwerken zu teilen um diese einer breiten Masse zugänglich zu machen. Auch wäre es vorstellbar, eine eigene Facebook Seite oder ein Profil bei Twitter zu erstellen, um dort, ähnlich wie auf der Startseite der Arthrose Plattform, Neuigkeiten zu posten und so NutzerInnen auf die Seite aufmerksam zu machen.

Literaturverzeichnis

Aderinokun, I. (2016, December 13). An Overview of Client-Side Storage [Blog]. Retrieved June 9, 2017, from https://bitsofco.de

Antonio, C. de S. (2015). Webpack. In *Pro React - Build complex front-end applications in a composable way with React* (1st ed., p. 29). Apress.

Arsenault, C. (2017, June 8). OOCSS - The Future of Writing CSS. Retrieved July 30, 2017, from https://www.keycdn.com/blog/oocss/

Bachert, L. (2016, 06). Ein Einblick in Progressive Web Apps | techscouting through the java news [Blog]. Retrieved June 2, 2017, from https://blog.oio.de/2016/06/10/ein-einblick-in-progressive-web-apps/

Berner, D. (2016, June 1). Battling BEM CSS: 10 Common Problems And How To Avoid Them. Retrieved July 30, 2017, from https://www.smashingmagazine.com/2016/06/battling-bem-extended-edition-common-problems-and-how-to-avoid-them/

Bidelman, E. (2010, June 18). Leitfaden für die ersten Schritte bei der Verwendung des Anwendungscaches - HTML5 Rocks. Retrieved July 12, 2017, from https://www.html5rocks.com/de/tutorials/appcache/beginner/

Buckler, C. (2013, October 9). HTML5 Browser Storage: the Past, Present and Future — SitePoint [Blog]. Retrieved June 9, 2017, from https://www.sitepoint.com/html5-browser-storage-past-present-future/

Buckler, C. (2017, May 29). Best JavaScript Frameworks, Libraries and Tools to use in 2017. Retrieved August 22, 2017, from https://www.sitepoint.com/top-javascript-frameworks-libraries-tools-use/

Camden, R. (2015). *Client-Side Data Storage*. Retrieved from http://shop.oreilly.com/product/0636920043676.do

Croft, J. (2007, December). Frameworks for Designers. Retrieved August 22, 2017, from http://alistapart.com/article/frameworksfordesigners

El Aeraky, S. (2017). *Konzeptionelle Umsetzung und Evaluation einer interaktiven Plattform für Menschen mit arthrotischen Beschwerden*. FH St. Pölten, St. Pölten.

Farrugia, K. (2016, August 11). A Beginner's Guide To Progressive Web Apps. Retrieved June 2, 2017, from https://www.smashingmagazine.com/2016/08/a-beginners-guide-to-progressive-web-apps/

Geuens, J., Swinnen, T. W., Westhovens, R., de Vlam, K., Geurts, L., & Vanden

Abeele, V. (2016). A Review of Persuasive Principles in Mobile Apps for Chronic Arthritis Patients: Opportunities for Improvement. *JMIR mHealth and uHealth*, *4*(4). https://doi.org/10.2196/mhealth.6286

Groves, K. (2013, May 20). Understanding WCAG Level. Retrieved June 14, 2017, from http://www.karlgroves.com/2013/05/20/understanding-wcag-level/

Heitkötter, H., Hanschke, S., & Majchrzak, T. A. (2012). Evaluating Cross-Platform Development Approaches for Mobile Applications. In *Web Information Systems and Technologies* (pp. 120–138). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-36608-6_8

Hoffmann, M. (2012). *Modernes Webdesign: Gestaltungsprinzipien, Webstandards, Praxis* (Auflage: 1). Bonn: Galileo Design.

Horton, S., & Quesenbery, W. (2014). *A Web for Everyone - Designing Accessible User Experiences*. Rosenfeld Media. Retrieved from http://rosenfeldmedia.com/books/a-web-for-everyone/

Janik, A., & Kiebzak, S. (2014). Client-side Storage : Offline Availability of Data. *Journal of Automation Mobile Robotics and Intelligent Systems*, *Vol. 8*, *No. 4*. Retrieved from http://yadda.icm.edu.pl/baztech/element/bwmeta1.element.baztech-6f807a1c-fe76-4671-ab0f-f37c95650abb

Klimont, J., & Baldaszti, E. (2015). *Österreichische Gesundheitsbefragung 2014. Hauptergebnisse des Austrian Health Interview Survey (ATHIS) und methodische Dokumentation* (p. 245). Wien: Statistik Austria.

Krause, S. (2016, January). JS web frameworks benchmark – Round 1 [Blog]. Retrieved September 7, 2017, from http://www.stefankrause.net/wp/?p=191

Laine, M. (2012). Client-Side Storage in Web Applications. Retrieved from http://media.tkk.fi/webservices/personnel/markku_laine/client-side_storage_in_web_applications.pdf

Lerner, A. (2013). *ng-book - The Complete Book on AngularJS* (1st ed.). Fullstack io.

Luhur, G. (2016, February 11). How to Use ARIA Effectively with HTML5. Retrieved June 16, 2017, from https://www.sitepoint.com/how-to-use-aria-effectively-with-html5/

Malavolta, I. (2016). Beyond Native Apps: Web Technologies to the Rescue! (Keynote). In *Proceedings of the 1st International Workshop on Mobile Development* (pp. 1–2). New York, NY, USA: ACM. https://doi.org/10.1145/3001854.3001863

Max, S. (2014, July 21). An Introduction to WAI-ARIA [Blog]. Retrieved June 14, 2017, from https://www.sitepoint.com/introduction-wai-aria/

Michalak, P. (2015, February). How to choose the right CSS methodology. Retrieved July 30, 2017, from http://labs.bench.co/blog/2015/2/19/how-to-choose-the-right-css-methodology

Morera, E. P., Díez, I. de la T., Garcia-Zapirain, B., López-Coronado, M., & Arambarri, J. (2016). Security Recommendations for mHealth Apps: Elaboration of a Developer's Guide. *Journal of Medical Systems*, *40*(6), 152. https://doi.org/10.1007/s10916-016-0513-6

Novy, F. (2015, May). *Offline-Capable Web-Based Rich Internet Applications: making REST-based Web Apps ready for offline use*. FH St. Pölten, St. Pölten.

Nutzungsbeschränkungen und Abrechnung | Google Places API Web Service. (2017, June). Retrieved September 6, 2017, from https://developers.google.com/places/web-service/usage?hl=de

Pearson, J., Walsh, N., Carter, D., Koskela, S., & Hurley, M. (2016). Developing a Web-Based Version of An Exercise-Based Rehabilitation Program for People With Chronic Knee and Hip Pain: A Mixed Methods Study. *JMIR Research Protocols*, *5*(2). https://doi.org/10.2196/resprot.5446

Petricek, T. (2015, March 3). Library patterns: Why frameworks are evil. Retrieved August 22, 2017, from http://tomasp.net/blog/2015/library-frameworks/

Pillora, J. (2014). *Getting Started with Grunt: The JavaScript Task Runner* (1st ed.). Packt Publishing. Retrieved from http://ebook-dl.com/book/608

Potthof, T. (2014, December). Service, Factory und Provider verstehen - AngularJS.DE. Retrieved August 31, 2017, from https://angularjs.de/artikel/service-factory-provider/

Richtlinien für barrierefreie Webinhalte (WCAG) 2.0 (Web Content Accessibility Guidelines (WCAG) 2.0). (2009, October). Retrieved June 14, 2017, from https://www.w3.org/Translations/WCAG20-de/#keyboard-operation

Rishabh, A. (2015, June 24). An Overview of OOCSS, BEM, SMACSS (CSS Methodologies/Practices) with References. Retrieved July 28, 2017, from http://codetheory.in/an-overview-of-oocss-bem-smacss/

Sauer, M. (n.d.). Durchstarten mit Gulp.js – Websites optimieren, Arbeitsabläufe automatisieren. Retrieved July 26, 2017, from http://magazin.phlow.de/webdesign/gulp/

Simpson, A. (2016, June 22). How To Harness The Machines: Being Productive With Task Runners. Retrieved July 25, 2017, from

https://www.smashingmagazine.com/2016/06/harness-machines-productive-task-runners/

Snook, J. (2012). *Scalable and Modular Architecture for CSS*. n/a.

Steyer, M. (2017). *Progressive Web-Apps: Offlinefähige Web-Anwendungen mit nativen Qualitäten*. entwickler.Press.

Umapathy, H., Bennell, K., Dickson, C., Dobson, F., Fransen, M., Jones, G., & Hunter, D. J. (2015). The Web-Based Osteoarthritis Management Resource My Joint Pain Improves Quality of Care: A Quasi-Experimental Study. *Journal of Medical Internet Research*, *17*(7). https://doi.org/10.2196/jmir.4376

WAI-ARIA basics. (2017). Retrieved June 14, 2017, from https://developer.mozilla.org/en-US/docs/Learn/Accessibility/WAI-ARIA_basics

WCAG 2.1 — It's here! (2017, March 21). Retrieved June 16, 2017, from https://medium.com/@intopia/wcag-2-1-its-here-70abeca88b2f

WebAIM: Web Content Accessibility Guidelines. (2013). Retrieved June 13, 2017, from http://webaim.org/standards/wcag/

Wilson, B. (2015, February 4). Accessible Web Development Using W3C's WAI-ARIA [Blog]. Retrieved June 14, 2017, from http://usabilitygeek.com/accessible-web-development-using-w3c-wai-aria/

Zimmermann, G. (2005, September). Usability und Barrierefreiheit - Gemeinsam sind wir stark? Retrieved June 13, 2017, from http://www.accesstechnologiesgroup.com/pubs/Zimmermann2005-Usability-Barrierefreiheit/

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Anhang

A. CSS/Javascript Technologie Vergleich

Kriterien gemeinsam	Gewicht	CSS	JS	Foundat	Skeleto	Bootstra	Semant	Materia	jQuery	ReactJS	AngularJS	Angular 4	VueJS 2
Schnelles Prototyping und Coding	3	x	x	1	3	1	1	2	1	3	1	3	1
Angemessene Lernkurve	2	x	x	2	1	2	2	2	1	2	2	3	2
High Performance und Speed	1		x						3	1	2	1	1
Kleiner Datenkern/keine Codeaufblähung	2	x	x	2	1	3	4	3	1	2	2	4	2
Community/Support	2	x	x	2	3	1	3	2	1	3	1	2	4
Gute ausführliche Dokumentation	3	x	x	2	3	1	1	1	1	1	1	2	2
Zukunftsfähigkeit/Lebensdauer	2		x						1	1	3	2	2
Einsatz von Preprozessoren	3	x		2	4	2	2	2					
Nutzung möglicher JS Funktionen	1	x		1	4	1	1	1					
Zusätzliche Build-Tools notwendig	3		x						1	3	2	3	1
Gesamt				**28**	**44**	**25**	**31**	**30**	**20**	**38**	**30**	**47**	**33**
Framework/Library									L	L	F	F	F
Stackoverflow (Stand: 06.09.2017)				4188	784	10956	650	2743	172895	858	14423	2408	59
Größe in KB (Minified Version)				72	6	151	548	131	87	153	167	636	79
Github Stars				25.820	14.297	112.496	35.922	27.555					
Github				foundat	Skeleto	bootstra	Semant	materialize					
Preprozessoren				SASS	-	v4, SASS,LESS		SASS					
Zusätzliche Plug-Ins notwendig									-	Babel, JSX	-		TypeScript
Versionsnummer				6.4.2	-	4	2.2.10	0.99	3.2.1	15.6.1	1.6.4	-	2.3.2
Erscheinungdatum									8/2006	3/2013	10/2010	9/2016	9/2016

Kriterien	Gewichtung
Kriterium sehr gut erfüllt	1
Kriterium gut erfüllt	2
Kriterium ausreichend erfüllt	3
Kriterium kaum erfüllt	4

Kriterium sehr wichtig	3
Kriterium mäßig wichtig	2
Kriterium kaum wichtig	1

L = Library
F = Framework