Konzeption und Entwicklung von adaptiven Web Components

Christoph Kleber

16. Dezember 2016

Kurzfassung

In dieser Arbeit geht es um Web Components...

Abstract

This thesis is about Web Components...

Listing Verzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

API Advanced Programming Interface

DOM Document Object Model

HTML Hypertext Markup Language

URL Uniform Resource Locator

HTML5 Hypertext Markup Language Version 5

CSS Cascading Style Sheets

W3C World Wide Web Consortium

REST Representational State Transfer

HTTP Hypertext Transfer Protocol

JSON JavaScript Object Notation

WCAG 2.0 Web Content Accessibility Guidelines 2.0

AWC Framework Adaptive Web Components Framework

Inhaltsverzeichnis

1 Adaptivität

Der Begriff Adaptivität wird in vielen verschiedenen Kontexten genutzt. Die Bedeutung ist somit mehrdeutig und abhängig vom Umfeld. Beispielsweise wird in der Psychotherapie unter Adaptivität im therapeutischen Vorgehen eine „Grundhaltung [beschrieben], welche die Bereitschaft impliziert, unter stetiger Reflexion der Prozesse von Übertragung und Gegenübertragung flexibel auf die jeweils aktuellen Bedürfnisse des Patienten einzugehen“[1, S. 45]. Im Kontext von Datenbanken und der „Adaptivität an unterschiedliche Anforderungen“ wird von Verfahren gesprochen, die „die Anpassungsfähigkeit an unterschiedliche Anforderungen und damit auch an verschiedene Einsatzumgebungen [erhöhen]“[2, S. 112]. In der Softwareentwicklung kann Adaptivität folgendermaßen beschrieben werden: „Interaktive Softwaresysteme werden von Benutzern mit unterschiedlichsten Zielen, Interessen, Fähigkeiten, Erfahrenheitsgraden und Präferenzen verwendet. Um einem möglichst breitem Personenkreis zugänglich zu sein, bieten viele derzeit erhältliche Programme bereits die Möglichkeit, daß Benutzer (oder Systemadministratoren) in bestimmtem Ausmaß eine Anpassung des Programms an die jeweiligen individuellen Präferenzen vornehmen können“[3, S. 1].
In den verschiedenen Auslegungen des Wortes ist ein Muster zu erkennen. Adaptivität kann definiert werden als die Fähigkeit eines Objekts, zum Beispiel eine Person oder ein System, sich an seine Umgebung anzupassen. Diese Anpassung basiert auf bestimmten Einflüssen, so kann sich eine Datenbank an äußere Einflüsse, wie beispielsweise ihre Einsatzumgebung, oder ein Softwaresystem an die Vorlieben seines Nutzers anpassen. Das kann auch automatisch geschehen. So gleicht sich beispielsweise der Inhalt der Seite „Facebook“ aufgrund eines Algorithmus an den einzelnen Nutzer an, ohne dass dieser bestimmte Einstellungen vornehmen muss[4]. Die Möglichkeit zur Adaptivität kann jedoch auch dem Nutzer bereitgestellt werden. Ist das der Fall, kann der Nutzer beispielsweise die Benutzeroberfläche

nach seinen eigenen Vorlieben einrichten. So kann zum Beispiel die Benutzeroberfläche der Entwicklungsumgebung „JetBrains PhpStorm“ anhand persönlicher Präferenzen aufgebaut werden und Farbstile und Kontraste verändert werden.
Hier stellt sich jedoch die Frage, wie im Zusammenhang von Internetanwendungen eine Adaptivität bereitgestellt werden kann und insbesondere, an welche Aspekte sie sich adaptieren soll. Hierfür muss die Umgebung, der Nutzer und auch dessen Arbeitsaufgaben erforscht und definiert werden.

1.1 Nutzungskontext

Die Betrachtung des Nutzungskontext bietet diese Möglichkeit. Der Nutzungskontext wird nach der DIN EN ISO 9241-210 von den Benutzermerkmalen, Arbeitsaufgaben und der organisatorischen, technischen und physischen Umgebung bestimmt. Im Folgenden wird ein Überblick dieser Beschreibung gegeben[5, S.15 ff.].

Nutzer und sonstige Interessengruppen Zu Beginn sollten die Nutzergruppen und weitere Interessengruppen identifiziert und deren wesentliche Ziele und Einschränkungen beschrieben werden.

Merkmale der Nutzer oder Nutzergruppen Diese Merkmale können „Kenntnisse, Fertigkeiten, Erfahrung, Ausbildung, Übung, physische Merkmale, Gewohnheiten, Vorlieben und Fähigkeiten einschließen“[5, S.16]. Sie beschreiben also insgesamt den Nutzer, um diesen besser einordnen und sich besser an diesen anpassen zu können.

Ziele und Arbeitsaufgaben der Nutzer Auf der einen Seite werden die Ziele der Nutzer beschrieben, auf der anderen Seite die Gesamtziele des Systems. Danach müssen die Arbeitsaufgaben betrachtet und nach ihren Merkmalen untersucht werden, beispielsweise wie oft eine Aufgabe ausgeführt werden soll.

Umgebung(en) des Systems Die Umgebung lässt sich in die technische Umgebung, also die der Computerkomponenten und Anwendungen, die physikalische Umgebung, also Aspekte wie beispielsweise Beleuchtung und soziale und kulturelle Umgebung aufteilen. Zur kulturellen Umgebung zählen beispielsweise die Arbeitsweise und Einstellungen der Umgebung des Systems.

Insgesamt lässt sich somit sagen, dass die Adaptivität an den Nutzungskontext ausgerichtet werden kann. Verschiedene Merkmale des Nutzungskontext haben Einfluss darauf, wie die Adaptivität, passend zur Situation, erfolgen sollte.

2 Web Components

Web Components sind eine World Wide Web Consortium (W3C) Spezifikation. Diese soll es ermöglichen, eigenständige und wiederverwertbare Komponenten für Web Anwendungen zu erstellen[6, S. 1]. Sie setzen sich zusammen aus den vier Technologien *Custom Elements*, *Hypertext Markup Language (HTML) Imports*, *Templates* und *Shadow Document Object Model (DOM)*. Das Nutzen dieser Technologie soll Applikationen im Web leichter wiederverwertbar, wartbar, unabhängiger und kapselbar machen[6, S.2].

2.1 Geschichte der Web Components

Web Components wurden vom W3C das erste Mal im Jahr 2012 als ein *Working Draft*, also Arbeitsentwurf, erwähnt. Hier wurde es auch *Component model for the web* genannt und bestand aus den vier Technologien *Templates*, *Decorators*, *Custom Elements* und *Shadow DOM*. In der derzeitigen Version der *Web Components* wird die *Decorators* Technologie nicht mehr verwendet. Die *HTML Import* Technologie wurde jedoch zu den *Web Components* ergänzt[7]. Da die *Web Components* aus verschiedenen Technologien zusammengesetzt sind, wird in dem nächsten Abschnitt auf die Geschichte der einzelnen Technologien eingegangen um einen Überblick zu verschaffen. Hierbei wird insbesondere auf die erste Unterstützung der Technologien in den verschiedenen Browsern und die heutige Browserkompatibilität eingegangen.

2.1.1 Custom Elements

Custom Elements liegen in der Version v0 und in der Version v1 vor. Die Version v0 wurde das erste Mal im Jahr 2014 von den Browsern *Chrome* in der Version 33 und

Opera in der Version 20 unterstützt. In den mobilen Varianten dieser Browser wird sie seit 2016 von *Opera for Android* in der Version 37 und von *Chrome for Android* in der Version 53 verwendet[8]. Im *Android* Browser ist diese Version schon seit 2014 in Gebrauch, in der *Android* Version 4.4.4. Im *Samsung Internet* wird sie seit 2016 in der Version 4 genutzt[9]. Die Version v0 wird von der Version v1 abgelöst, hier ergeben sich einige Änderungen in der Syntax der Advanced Programming Interface (API)[10]. Derzeit wird sie nicht per Standardeinstellung von anderen Browsern unterstützt[11], im Gegensatz zur Version v1. *Chrome* und *Opera* unterstützen diese. Das erste mal wurde sie im Jahr 2016 in der *Chrome* Version 54 und in der *Opera* Version 41 genutzt[12].

2.1.2 HTML Imports

HTML Imports wurden in den Browsern *Chrome* und *Opera* zuerst 2014 verwendet. Die *Imports* wurden als Erstes in der *Chrome* Version 36 und in der *Opera* Version 23 genutzt. Gegenwärtig wird die Technologie von den zuvor erwähnten Browsern auch in den mobilen Browser-Varianten unterstützt, in den Versionen 53 des *Chromium* und 37 des *Opera for Android*[13]. Der *Android* Browser hat die *HTML Imports* seit 2016 in Gebrauch, in der Version 53. Der Browser des *Android* Betriebssystems ist ab dem Jahr 2016 in der Version 53 kongruent mit dem Chromium Browser. *Samsung Internet* unterstützt die *Imports* seit 2016 in der Version 4[14].

2.1.3 Decorators

Decorators erscheinen nur in Dokumenten und Artikeln, sie wurden nie von Browsern implementiert. So wird im Jahr 2012 in einem *Working Draft* des W3C von einem Beispiel gesprochen: „Here is an example of how decorators could be used to implement a simple variant of the details element“[7]. Dies zeigt, dass an dieser Stelle noch keine Implementierung der Technologie vorliegt. Auch in einem Arbeitsentwurf des W3C vom Jahr 2013 wird davon gesprochen, dass „Decorators, unlike other parts of Web Components, do not have a specification yet“[15]. Auf einer aktuellen Übersichtsseite des Konsortiums wird die *Decorators* Technologie nicht mehr im Zusammenhang mit *Web Components* erwähnt[16].

2.1.4 Templates

Templates werden schon über einen längeren Zeitraum in den verschiedenen Browsern unterstützt. Zuallererst wurden sie im *Chrome* im Jahr 2013 verwendet, in der Version 26. Im selben Jahr wurden sie vom *Firefox* Browser in der Version 22 und vom *Opera* Browser in der Version 15 erstmals verwendet[17][18]. Im Jahr 2015 wurden sie dann vom *Edge* Browser unterstützt, in der Version 13[19]. Auf den Browsern des *Macintosh* Betriebssystems wurden *Templates* zuerst 2014 verwendet, in der *Safari* Version 7.1 und der *Safari & Chrome for iOS* Version 8[14]. In den meisten mobilen Varianten der Browser werden *Templates* seit 2016 bereitgestellt. So erschien diese Funktion in dem *Opera for Android* Browser in der Version 37, in *Chrome for Android* in 53, in *Firefox for Android* in 49 und im *Samsung Internet* Browser in der Version 4. Der Standard Android Browser unterstützt *Templates* jedoch schon seit 2013, in der Version 4.4[14].

2.1.5 Shadow DOM

Der *Shadow DOM* existiert in der Version v0 und in der Version v1. Die Version v0 wird seit 2014 vom *Chrome* in der Version 35 und *Opera* Browser in der Version 21 unterstützt. Mit der Safari Version 10 kann der *Shadom DOM* seit 2016 genutzt werden[20]. Die mobilen Varianten der Browser gebrauchen das *Shadow DOM* seit 2016, *Opera* in der Version 37 und *Chrome* in der Version 53, somit auch der *Android* Browser[21]. Die Version v0 wird von der Version v1 abgelöst, welche verschiedene Neuerungen in der Syntax, aber auch in dem Gebrauch von bestimmten Funktionen aufweist. So kann beispielsweise in der v0 ein *shadow root* immer nur als „open“ definiert werden, in der v1 kann er auch als „closed“ *shadow root* erstellt werden[22]. Die Version v1 wird noch nicht in so großem Ausmaß wie die Version v0 unterstützt. Vollständig wird sie nur vom *Chrome* 53 und *Opera* 40 Browser genutzt, jeweils seit 2016. In den mobilen Versionen wird sie nur von dem *Chromium* und somit auch auf *Android* in der Version 53 verwendet[23].

2.2 Technik der Web Components

Im Folgenden werden die technischen Grundlagen der *Web Components* und insbesondere deren Syntax erforscht und durch Beispiele erläutert.

2.2.1 Custom Elements

Das *Custom Element* ist eine API, welches das Bilden eigener, voll funktionstüchtiger DOM Elemente ermöglicht[24]. Die API beschreibt in diesem Zusammenhang eine Schnittstelle, die einem anderen Programm ein Werkzeug zur Verfügung stellt, um sich an das eigene Softwaresystem anbinden zu können[25]. Somit ermöglicht eine API einen Austausch von Informationen zwischen verschiedenen Programmen oder Systemen. Die *Custom Element* API ermöglicht den Nutzern die Auszeichnungsprache HTML zu erweitern[26]. Es können bestehende HTML Elemente erweitert, oder neue hinzugefügt werden. Jedes neue oder erweiterte Element wird unter einem *Tag* Namen registriert. Dies ermöglicht eine Kapselung des erstellen Programmiercodes in Elemente. In Listing 2.1 ist ein JavaScript Programmcode dargestellt, der ein leeres *Custom Element* definiert und unter dem Namen „new-custom-element“ registriert wird. Für *Custom Elements* sind mehrere *Callbacks* ver-

```
1 class NewCustomElement extends HTMLElement {
2         constructor() {
3                 super();
4         }
5 }
6 customElements.define('new-custom-element', NewCustomElement);
```

Listing 2.1: Custom Element JavaScript

fügbar. *Callback* Funktionen beschreiben hier Funktionen, die bei bestimmten Ereignissen des Lebenskreislaufs der Applikation von außerhalb des *Custom Elements* aufgerufen werden. Im Folgenden werden diese Funktionen aufgelistet[26].

connectedCallback() Diese Funktion wird aufgerufen, wenn das *Custom Element* an den DOM angehängt wird.

disconnectedCallback() Diese Funktion wird aufgerufen, wenn das *Custom Element* vom DOM wieder losgelöst wird.

attributeChangedCallback(name, prevValue, newValue) Diese Funktion wird aufgerufen, wenn sich ein Attribut ändert; jedoch nur für Attribute aufgerufen, die in einer statischen *get* Funktion mit Namen *observedAttributes* definiert wurden.

2.2.2 HTML Imports

HTML Imports ist eine Technologie zum Importieren von externen HTML Dokumenten in ein HTML Dokument. Hier unterscheidet man zwischen importierenden und importierten HTML Dokumenten. Die importierenden Dokumente besitzen einen Link, der mindestens die Uniform Resource Locator (URL) des *Imports* und die Eigenschaft *rel=„import“* besitzt,also ein Link eines bestimmten Typ ist[27].(siehe Listing 2.2) Die importierten Dokumente haben keinen außergewöhnlichen Aufbau im

```
1 <link rel='import' href='/imports/imported-document.html'>
```

Listing 2.2: Standard HTML Import

Vergleich zu normalen HTML Dokumenten, sie können aus HTML, Cascading Style Sheets (CSS) oder JavaScript Elementen bestehen. Es kann auch die *Template* Technologie verwendet werden, dazu mehr in Kapitel 2.2.3. Um auf den Inhalt des importierten Dokuments zuzugreifen wird JavaScript verwendet. Wie in Listing 2.3 dargestellt, wird zuerst nach dem Link Element gesucht, welches die Eigenschaft rel=„import“ besitzt. Daraufhin wird dieses Dokument importiert und ein bestimmter Teil des Dokuments als JavaScript Variable „elem“ gespeichert. Hier wird ein div Element, das mit der Klasse „element“ ausgezeichnet ist, gespeichert. Anschließend kann dies in der importierenden Seite genutzt werden.

```
1 var link = document.querySelector('link[rel=import]');
2 var importedDocument = link.import;
3 var elem = importedDocument.querySelector('div.element');
```

Listing 2.3: JavaScript Programmcode für Zugriff auf Inhalt des importierten Dokuments

2.2.3 Templates

Das Hypertext Markup Language Version 5 (HTML5) Feature *Template* ermöglicht Teile einer Seite unabhängig vom DOM zu erstellen. Diese können dann später programmatisch zum DOM hinzugefügt werden[28, S.177]. Das bedeutet, dass der Inhalt des *Templates*, bis er zum DOM hinzugefügt wird, nicht in der Webanwendung angezeigt wird und auch nicht über DOM Selektoren angesteuert werden kann. Gegebenenfalls im *Template* enthaltene Bilder werden nicht geladen und Skripte nicht ausgeführt[29]. In Listing 2.4 sieht man den JavaScript Programmcode um ein vor-

```
1 var inhalt = document.querySelector('template').content;
2 document.querySelector('body').appendChild(inhalt);
```

Listing 2.4: JavaScript Programmcode für das Hinzufügen eines Templates in das DOM

handenes *Template* zum DOM hinzuzufügen. In Zeile eins wird der Inhalt des *Templates* zur JavaScript Variable „inhalt“ hinzugefügt, um dann in der nächsten Zeile an den *body* der Seite, also dem eigentlichen Inhalt hinzugefügt zu werden. In diesem Moment werden ebenso die Bilder des *Templates* geladen und eventuelle Skripte ausgeführt.

2.2.4 Shadow DOM

„Das *Shadow DOM* beschreibt die Fähigkeit eines Browsers, eine neue, völlig ab gekapselte Knotenstruktur im bestehenden DOM zu erzeugen“[30, Kap. 11.1.4]. Dies bedeutet, dass neben dem normalen *Document tree*, dessen Wurzelknoten ein Dokument ist, noch der *Shadow tree* besteht. Der Wurzelknoten des letzteren ist kein

Dokument, sondern der *Shadow root*. Die Folge dieser Kapselung ist, dass alles was dem *Shadow tree* hinzugefügt wird, nur lokal Einfluss hat. Die Gestaltung von Webelementen im *Shadow root* wird dadurch vereinfacht. CSS Selektoren können dadurch nicht von außerhalb des *Shadow roots* auf diesen zugreifen und Selektoren, die innerhalb dieses definiert werden haben keinen Einfluss auf den normalen DOM. Genauso verhält es sich mit dem Zugriff auf die DOM Elemente des *Shadow root*. Sie können nicht von außerhalb angesprochen werden, beispielsweise durch die Funktion „document.querySelector()“, sondern können nur von Funktionen innerhalb des *Shadow root* angesprochen werden[31]. In Listing 2.5 ist dargestellt,

```
1 var header = document.createElement('header');
2 var shadowRoot = header.attachShadow({mode: 'open'});
3 var headline = document.createElement('h1');
4 var headlineText = document.createTextNode('headline');
5 headline.appendChild(headlineText);
6 shadowRoot.appendChild(headline);
```

Listing 2.5: JavaScript Programmcode für die Erstellung eines Shadow DOM

wie Mithilfe von JavaScript ein *Shadow DOM* erstellt wird. In Zeile zwei wird zuerst einem bestehendem Element ein *Shadow root* hinzugefügt. Daraufhin wird eine Überschrift und deren Text erstellt und zusammengefügt. Zuletzt wird in Zeile sechs die Überschrift in den *Shadow DOM* eingefügt.

2.2.4.1 Slots

Shadow DOM kann auch mit anderen DOM erweitert werden. Der Entwickler kann dem Nutzer seines *Web Component* ermöglichen, diesen zu erweitern. Hierfür werden *Slots* verwendet. In Listing 2.6 ist der HTML Programmcode des *Web Component* dargestellt, welcher im *Shadow DOM* später gerendert wird. Hier ist die Verwendung des *Slot* Elements interessant. Dieses kann beim Einbinden des *Web Component* später ausgestattet werden. Wird das Platzhalter-Element später nicht befüllt, wird das *Fallback*, also die Ersatzfunktion, der Inhalt innerhalb des Elements, hier beispielsweise „Kein Titel“ genutzt. Die Ersatzfunktion kann auch aus einem ei-

```
<ul id='contacts'>
   <li>
      <slot name='title'>Kein Titel</slot>
      <slot name='name'>
         <p>Achtung<p>
         <h1>Kein Name<h1>
      </slot>
   </li>
</ul>
```

Listing 2.6: Nutzung von Slot Platzhalter-Elementen im Shadow DOM

genen DOM Baum bestehen, wie im *Slot* „name“ zu sehen ist[31]. In Listing 2.7 werden die, in Listing 2.6 erstellten Platzhalter-Elemente, beim Verwenden des *Web Component* in beispielsweise einer Webseite befüllt. Wie hier in Zeile fünf und sechs zu sehen ist, können einzelne Slots auch mit mehreren Elementen befüllt werden. Beim Übersetzen des Programmcodes der Applikation werden alle Elemente, die die passenden *Slot* Attribute aufweisen in den DOM übersetzt. Wenn im *Shadow DOM* ein *Slot* Platzhalter ohne ein „name“ Attribut definiert wird, werden alle vom Nutzer innerhalb des *Web Components* erstellten Elemente in den *DOM* geschrieben. In

```
<span slot='title'>Dr.</span>
<span slot='title'>Phil.</span>
<span slot='name'>Michael</span>
```

Listing 2.7: Befüllen der Slot Elemente im DOM

Listing 2.8 ist der von den zwei vorhergehenden Listings kombinierte und gerenderte Inhalt zu sehen. Dies ist der DOM, den man beispielsweise in einer Webseite sehen würde. Hier ist gut zu erkennen, wie die *Slot* Elemente des *Shadow DOM* mit den später erstellten Elementen befüllt werden. Alle Elemente innerhalb des *Web Component* mit dem Attribut „slot“ werden in diesen übertragen.

```
<ul id='contacts'>
   <li>
      <slot name='title'>
             <span>Dr.</span>
             <span>Phil.</span>
          </slot>
      <slot name='name'>
             <span>Michael</span>
      </slot>
   </li>
</ul>
```

Listing 2.8: Übersetzter DOM

2.2.4.2 Shadow DOM Stil-Regeln

Wie auf den vorherigen Seiten erklärt, ist der *Shadow tree* vom normalen DOM abgekapselt. Dies hat zur Folge das CSS Stil-Regeln, definiert im eigenen *Component*, nur Einfluss auf den *Shadow tree* des *Web Component* haben. Es können CSS Selektoren verwendet, die auch außerhalb des Element bestehen. Trotzdem haben diese keinen Einfluss auf außerhalb.
Eine Besonderheit der Stil-Regeln des *Shadow DOM* sind dessen Selektoren. Der „:host“ Selektor definiert den Stil für das Element, in welchem der *shadow tree* bereitgestellt wird, somit für den *Web Component*. Dieser wirkt nur in dem Kontext eines *shadow root*. Dadurch kann der Stil des eigenen *Web Component* verändert werden. Eine Eigenschaft des „:host“ Selektors ist, dass die Seite, welche das *Web Component* enthält, stärkeren Einfluss auf diesen Selektor besitzt und somit den Stil überschreiben kann. Damit ermöglicht es den Nutzern eines *Web Component*, diesen nach eigenen Wünschen anzupassen. Darüber hinaus kann der „:host“ Selektor erweitert werden. So können über die Form „:host(<selector>)“ bestimmte Stil Eigenschaften gesetzt werden, welche nur Anwendung finden wenn das *Web Component* bestimmte Eigenschaften erfüllt. In Listing 2.9 ist dargestellt in welcher Form Einfluss genommen werden kann. So spricht der Selektor in Zeile zwei nur den *Web Component* an, während die Maus sich über diesem befindet. Der Selektor in Zeile

```
<style>
:host(:hover) {
   opacity: 1;
}
:host(.blue) {
   color: blue;
}
</style>
```

Listing 2.9: Der :host Selektor

fünf spricht nur den *Web Component* an, wenn dieser die Klasse „blue“ erhalten hat.

2.3 Webentwicklung mithilfe der Web Components

Das *Frontend* von modernen Webseiten basiert heutzutage hauptsächlich auf den Technologien HTML, CSS und JavaScript. Meistens werden darüber hinaus noch verschiedene *Frameworks* verwendet. „A computer system framework has a layered structure that can incorporate servers, operating systems, client machines, and applications. The framework can provide a set of functions to definde application interfaces, the interrelationships between applications, and internal communications between clients and external to online platforms“[**stuart2013roadmap**]. Ein *Framework* ist somit ein System, das den Entwicklern bestimmte Funktionalitäten zur Verfügung stellt, ohne dass dieser sie selbst programmieren muss. Sie können beispielsweise die Hilfe bei der Interaktion mit dem DOM sein, wie das JavaScript *Framework* „jQuery“, ein „Slide-Element“ bereitstellen wie das *Framework* „Slider“ oder zur Diagrammerstellung genutzt werden wie das *Framework* „D3“. Hier ergibt sich ein Problem. Wenn in einer Applikation mehrere *Frameworks* und Technologien für verschiedene Funktionen verwendet werden, können sich diese gegenseitig beeinflussen. So können die Stil-Regeln verschiedener Teile der Webseite sich unbeabsichtigt beeinträchtigen oder das JavaScript, das eine bestimmte Funktion hat, an einer anderen Stelle für die es nicht programmiert wurde, Einfluss nehmen. Viele dieser Probleme resultieren daraus, dass verschiedene *Frameworks* für Variablen

oder Funktionen die gleiche Benennung nutzen und somit manche Elemente doppelt benannt und damit auch doppelt angesprochen werden. Darüber hinaus können viele Teile der Webseite weder wiederverwendet, noch gut gewartet werden, da sie großen Einfluss aufeinander nehmen und somit sehr ineinander verschachtelt sind. Die *Web Components* versuchen diese Probleme durch eine (in Zukunft) native Implementierung verschiedener Techniken anzugehen, die eine Kapselung, eine Wiederverwendung und eine leichtere Wartbarkeit von Programmcode ermöglichen sollen. Nachfolgend wird auf die daraus resultierenden Vor- und Nachteile eingegangen.

2.3.1 Native Browserunterstützung

Aufgrund der Erkenntnisse der Geschichte von *Web Components* in Kapitel 2.1 ist eine gewisse Wahrscheinlichkeit gegeben, dass die Technologien der *Web Components* in Zukunft nativ von den verschiedenen Browsern unterstützt werden. Sollte dieser Fall eintreffen ergibt sich daraus ein großer Vorteil. Es muss bei der Nutzung nativer, also von den Browsern implementierten Techniken, kein externer Programmcode genutzt werden um bestimmte Funktionen abzudecken. Viele Funktionalitäten können einfach über die Nutzung nativer Methoden abgebildet werden. Deshalb müssen weniger Funktionen selbst geschrieben und weniger, beziehungsweise unter Umständen keine *Frameworks* genutzt werden. Dies verkleinert das Laden von externen Programmcode. Außerdem ist die Syntax und Funktionsweise bei nativen Funktionen bekannt und eindeutig. Daraus ergeben sich weniger Inkonsistenzen in der Programmierung und eine leichtere Verständlichkeit. Im Gegensatz dazu muss bei vielen *Frameworks* eine jeweils eigene Syntax benutzt werden.

2.3.2 Kapselung

Ein Mechanismus zur Datenkapselung wird vom *Shadow DOM* bereitgestellt. Dieser ermöglicht, dass der Programmcode des *Web Components* vom Rest der Applikation getrennt werden kann. Dadurch wird ein privater *Scope*, also ein Geltungsbereich der Applikation und dessen Variablen, Methoden und Bezeichnern, genutzt[6, S.2]. Dies hat einige Folgen für das Verhalten einer Applikation. Zuerst ist der *Sha-*

dow DOM isoliert. Er kann nicht von außerhalb angesprochen werden, beispielsweise über die Funktion „document.querySelector()“. Dies hat den Vorteil, dass die Funktionalität des *Web Component* nicht von außen beeinträchtigt werden kann. Außerdem hat das CSS nur Zugriff auf den DOM des eigenen Geltungsbereichs; weder von außerhalb noch von innerhalb nach außen des *Shadow DOM* können Stil-Regeln Einfluss nehmen. Ein Vorteil an dieser Eigenschaft ist, dass man atomare, also sehr einfache, CSS Bezeichner innerhalb des *Shadow DOM* verwenden und dieselben Bezeichner gleichzeitig außerhalb dieses nutzen kann[31]. Darüber hinaus wird die Gestaltung der Erscheinung der Applikation beständiger. Sie erfolgt einzeln für jedes *Web Component* und für den Bereich außerhalb der *Web Components*.

2.3.3 Wiederverwendung

Die Theorie der *Web Components* ist die Erstellung von verschiedenen Komponenten, die wiederverwendet werden können. Dies hat zum einen den großen Vorteil, dass es eine Interoperabilität zwischen *Frameworks* ermöglicht[6, S.2]. Dadurch ist man nicht an ein bestimmtes *Framwork* gebunden und kann auch mit Elementen außerhalb dieses Ökosystems interagieren und Komponenten wiederverwenden. Zum anderen können auch innerhalb eines *Frameworks* oder nativ, ohne das Nutzen eines *Frameworks*, Teile der Anwendung wiederverwendet werden, was eine Arbeitserleichterung und Verminderung des Programmcodes hervorruft.

2.3.4 Wartbarkeit

Die Wartbarkeit von Web-Applikationen wird erleichtert, da die *Web Components* in *Templates* organisiert sind[6, S.2]. Das sorgt dafür, dass der Programmcode einzelner Komponenten separat gespeichert und somit leichter wiedergefunden und geändert werden kann.

2.3.5 Browserunterstützung

Auch wenn *Web Components* sehr viele Vorteile aufweisen, ist es ein Problem wenn die Techniken (noch) nicht in allen Browsern unterstützt werden oder unterschied-

lich implementiert sind. Wie in Kapitel 2.1 dargelegt, werden einige der Techniken noch nicht von allen Browsern unterstützt oder unterscheiden sich in deren Umsetzung. Dies kann zu Inkonsistenzen oder dem nicht Funktionieren einer Applikation führen. Durch Verwendung von *Polyfills* kann man dies jedoch umgehen. Das sind in diesem Zusammenhang Programmcodes, die die Funktionen oder Teile einer Technologie, die nativ noch nicht von einem Browser unterstützt wird, nachstellen. Sie können dann verwendet werden um Nutzern aller Browser den Gebrauch der Technologien zu ermöglichen[32, S.4]. Das *webcomponents.js* ist ein Set von *Polyfills* und ermöglicht den Nutzern die Verwendung von *Web Components* in allen modernen Browsern[33].

3 Polymer

Die *Polymer* Bibliothek stellt eine Sammlung von Funktionen bereit um *custom elements* zu erstellen. Diese Elemente sollen letztendlich wie normale DOM Bestandteile funktionieren[34].

3.1 Polymer Technologie

Die verwendeten Technologien der Bibliothek ähneln den *Web Components* oder es werden direkt die Technologien der *Web Components* genutzt. Im Folgenden wird die Technologie der Version 1.x erläutert.

3.1.1 Ein Element registrieren

Ähnlich der *Web Components* Technologie, muss mithilfe der *Polymer* Bibliothek ein vom Entwickler neu erstelltes *custom element* zuerst registriert werden. Die

```
1  MyElement = Polymer({
2     is: 'my-element'
3  });
4  // Instanziierung mit createElement
5  var elem = document.createElement('my-element');
6  // Instanziierung über Konstruktor
7  var elem2 = new MyElement();
```

Listing 3.1: Polymer custom element Registrierung

„Polymer“ Funktion erstellt einen Prototypen eines Elements, registriert diesen im

Browser und gibt einen Konstruktor zurück, der genutzt werden kann um Instanzen des Elements zu erstellen. Wie in Listing 3.1 in Zeile zwei zu sehen, spezifiziert das „is“ Argument der Funktion den Namen der HTML Kennzeichnung (*tag*). In Zeile fünf und sieben sind die beiden Möglichkeiten dargestellt ein Element zu registrieren, also eine Instanz zu erstellen. Einmal über die „createElement“ Funktion oder über den „new“ Operator. In der „Polymer“ Funktion kann das Argument „extends“ hinzugefügt werden. Dieses kann mit einem Wert befüllt werden, der einem Standard HTML Element Bezeichner entspricht, beispielsweise „input“. Dadurch kann dieses Standard HTML Element erweitert werden. Bei der Instanziierung über den Konstruktor wird dann genauso wie in Listing 3.1 verfahren, bei der Instanziierung durch „createElement“ muss als Erstes Argument der Funktion zusätzlich das zu erweiternde Element und als zweites Argument der „is“ Wert des neu erstellten *custom element* übergeben werden, beispielsweise „var elem = document.createElement(’input’, ’my-input’); “[35].

3.1.2 Eigenschaften der Elemente

Der Prototyp kann mit Eigenschaften versehen werden. Hierfür wird das „properties“ Objekt genutzt. Es können beliebig viele Eigenschaften des Elements definiert und zu jeder Eigenschaft verschiedene Schlüssel hinzugefügt werden. Der „type“

```
1  Polymer({
2     is: 'my-element',
3     properties: {
4        user: String,
5        isHappy: Boolean,
6        count: {
7           type: Array,
8           value: function() { return {}; }
9        }
10    }
11 });
```

Listing 3.2: Polymer properties Objekt

Schlüssel definiert beispielsweise den Typ des Attributs, in Listing 3.2 ist die Eigenschaft „user“ beispielsweise vom Typ „String“ und die Eigenschaft „isHappy“ vom Typ „Boolean“. Als weiteres Beispiel definiert der „value“ Schlüssel den Standardwert der Eigenschaft, bei der Eigenschaft „count“ die in der Funktion erstellte leere Datengruppe[36].

3.1.3 Methoden der Elemente

Um den späteren Instanzen Methoden zur Verfügung zu stellen, können diese dem Prototyp des Elements hinzugefügt werden. In Listing 3.3 wird dem Prototyp die Methode „speak“ zur Verfügung gestellt. In Zeile neun wird dann die Methode von einer Instanz aufgerufen. Zudem stehen viele verschiedene Methoden standardmäßig jedem Prototyp zur Verfügung. Diese können für verschiedene Zwecke genutzt werden[37].

```
Polymer({
   is: 'my-element',
   speak: function(){
   console.log('speak');
   }
});
//Nachdem das Element instanziiert ist, kann die Methode aufgerufen
 ↪  werden.
var elem = document.querySelector('my-element');
elem.speak();
```

Listing 3.3: Polymer Instanzmethoden

3.1.4 Polymer DOM

Polymer unterstützt verschiedene DOM Implementierungen. Wird vom genutzten Browser *shadow DOM*, die Technologie aus dem Bereich der *Web Components* unterstützt, kann diese verwendet werden. Unterstützt der Browser dies nicht, wird eine angepasste Implementierung in *Polymer* genutzt, der *shady DOM*. Insgesamt sind beide Möglichkeiten in *Polymer* dem lokalen DOM zugeordnet und werden im weiteren so genannt[38]. In Listing 3.4 ist der Programmcode zur Erstellung eines

```
<dom-module id='my-element'>
   <template>I am my-element!</template>
   <script>
      Polymer({
      is: 'my-element'
      });
   </script>
</dom-module>
```

Listing 3.4: Polymer dom-module

lokalen DOM von *Polymer* dargestellt. Das Element zur Erstellung dieses wird „dom-module“ genannt. Das Element muss im „id“ Attribut den gleichen Wert enthalten wie der Prototyp im „is“ Argument, um eine Zuordnung zu ermöglichen. Es enthält ein

„template“ Element in Zeile zwei, dessen Inhalt wird dann in den lokalen DOM der Instanz des Prototyps geschrieben.
Polymer Elemente können durch Stil-Regeln ergänzt werden. Diese sollten, wie standardmäßig bei der Verwendung von CSS, innerhalb „<style>“ und darüber hinaus innerhalb des „<template>“ Tags gesetzt werden.
Kongruent zum *Shadow DOM*, ermöglicht der *Shady DOM*, eine Kapselung der Stil-Eigenschaften. Dadurch haben Stil-Regeln innerhalb des *Polymer* Elements keinen Einfluss auf außerhalb und umgekehrt. Um dem Nutzer eines *Polymer Web Component* jedoch zu ermöglichen Einfluss auf den Stil dessen zu nehmen, können benutzerdefinierte CSS Eigenschaften genutzt werden[39]. Diese werden in der in

```
1  <style>
2  .title {
3          color: var(--my-element-title-color);
4  }
5  .smalltext {
6          color: var(--my-element-title-color, blue);
7  }
8  </style>
```

Listing 3.5: Benutzerdefinierte CSS Eigenschaften zur Polymer Web Component Erstellung

Listing 3.5 dargestellten Form, im Bereich der Stil-Eigenschaften, bei der Erstellung eines *Polymer* Elements definiert. Wie in Zeile sechs zu sehen ist kann auch, falls der Nutzer der Eigenschaft keinen Wert zuweist ein Standard Wert gesetzt werden. Hier ist der Standardwert „blue“. An einer Stelle oberhalb des DOM kann dann die CSS Eigenschaft „–my-element-title-color“ vom Nutzer des Elements mit einem Wert belegt werden. Dadurch wird ab dann jedes Element das mit dieser benutzerdefinierten Eigenschaft belegt ist, mit dem vom Nutzer gesetzten Wert ausgezeichnet. Beispielsweise kann der „:host“ Selektor mit der Eigenschaft „–my-element-title-color: green“ ausgezeichnet werden. Dadurch werden alle Elemente innerhalb des *Shadow root*, mit der benutzerdefinierten Eigenschaft „–my-element-title-color“ auf den Wert „green“ gesetzt.

4 Das AWC Framework

Das Adaptive Web Components Framework (AWC Framework) wurde von der *REMEX* Forschungsgruppe entwickelt. Das JavaScript Framework wird auch mit der Bezeichnung *Darwin.js* beschrieben. Es soll die Implementierung von adaptiven Benutzeroberflächen ermöglichen, basierend auf der Technik der *Web Components*. Die Oberflächen richten sich somit nach den Bedürfnissen und der Zusammenstellung der Präferenzen eines Nutzers, um die Zugänglichkeit, Gebrauchstauglichkeit und das Nutzungserlebnis zu verbessern[40][41]. Im folgenden wird die verwendete Technik des AWC Framework erläutert.

4.1 Darwin.js Technologie

Um ein neues adaptives *Web Component* zu erstellen wird die Klasse „Component“ und von dieser der Konstruktor genutzt. Diesem können vier Parameter übergeben werden. Sie setzen sich aus „_htmlTag“, „_baseClass“, „_data“ und „_options“ zusammen. Zu Beginn nutzt der Konstruktor die Methode „testBrowserSupport()“ um abzufragen, ob die Technologien der *Web Components* vom Browser unterstützt werden. Ist dies nicht der Fall wird ein Fehler geworfen. Danach wird mit den übergebenen Parametern weitergearbeitet.

4.1.1 Registrierung eines Elements

Der Parameter „_htmlTag“ definiert den Bezeichner unter dem das neue Element registriert wird, während der Parameter „_baseClass“ definiert, welches HTML Element als Basis genutzt wird und erweitert wird. Hier kann zum Beispiel das „Button“ Element definiert werden. Das Elternelement aller HTML Elemente ist das „HTML-Element“, dieses kann auch ausgebaut werden, wenn kein bestimmtes, schon be-

stehendes Element erweitert werden soll. In Listing 4.1 ist der Programmcode dargestellt, der ein neues Element erstellt. Hier wird das Element unter dem Bezeichner „awc-image“ und als Erweiterung des „HTMLElement“ registriert.

```
new Component('awc-image', HTMLElement,
    // Model:
    {
        'contrast': 'normal',
        'size': 'normal',
        'reading_direction': 'ltr'
    },
    // Behaviour:
    {
        onAdaptiveChange: function() {
        console.log('onAdaptiveChange');
        console.log(this);
    }
});
```

Listing 4.1: JavaScript Programmcode zum Erstellen eines Web Components basierend auf Darwin.js

4.1.2 Eigenschaften der Elemente

Dem „_data“ Parameter werden die initialen Modell Daten übergeben. Hier können die eigentlichen Präferenzen oder Einstellungen der späteren Nutzer mit initialen Werten gespeichert werden. Es können beliebig viele Präferenzen-Wert Paare übergeben werden. In Listing 4.1 werden beispielsweise die Präferenzen „contrast“, „size“ und „reading_direction“ gesetzt.

4.1.3 Methoden der Elemente

Dem „_options“ Parameter können optional Methoden übergeben werden. Dadurch erhält das *Web Component* initial diese Funktionalität und kann zur Laufzeit verwendet werden. Bei der Erstellung des „awc-image“ in Listing 4.1 wird beispielsweise die

Methode „onAdaptiveChange“ übergeben. Diese wird somit dem *Web Component* zugeordnet.

4.1.4 Bilden eines Elements

Um das Element letztendlich zu bilden werden verschiedene Schritte im Konstruktor der „Component“ Klasse durchgeführt. Zu Beginn muss ein *Shadow root* geschaffen werden, das wird in Listing 4.2 dargestellt. Daraufhin wird der zu importierende

```
this.root = this.attachShadow({
   mode: 'open' //Offener Shadow Root
});
```

Listing 4.2: JavaScript Programmcode zum Erstellen des Shadow roots

Link, passend zum „_htmlTag“, gewählt. Von diesem *Link* wird das *Template* Element gespeichert. Daraufhin wir die Funktion „_buildHtml“ mit dem Parameter „data“ aufgerufen. Diese ist in Listing 4.3 dargestellt. Hier wird in Zeile eins der Inhalt

```
_buildHtml(data) {
   this.root.innerHTML = ''; //leerer Shadow Root
   let instance = this.template.content.cloneNode(true); //Kopieren
    ↪ des Inhalts
   let div = document.createElement('div');
   div.appendChild(instance); //Inhalt wird als Kind an das div
    ↪ angehängt
   let handlebar = window.Handlebars.compile(div.innerHTML);
    ↪ //Inhalt des div wird mit Handlebars kompiliert
   let buildHtml = handlebar(data); //Kompilierter Code wird mit
    ↪ data Werten ausgeführt
   this.root.innerHTML = buildHtml; //Shadow Root wird befüllt
}
```

Listing 4.3: JavaScript Programmcode zum Bilden eines Web Components basierend auf Darwin.js

des *Shadow Root* geleert. Daraufhin wird der Datenknoten des Inhalts des *Templates* kopiert. Dieser wird dann als Kind eines neu erstellten div Elements in Zeile fünf hinzugefügt. Daraufhin wird der Inhalt dieses divs mithilfe des *Handlebar* Frameworks kompiliert. *Handlebar* ermöglicht semantische *Templates* zu erstellen. In diesen *Templates* können Ausdrücke innerhalb vier geschweifter Klammern gesetzt werden, beispielsweise wie: „{{Inhalt}}“. Diese funktionieren als Platzhalter und werden beim kompilieren durch die gesetzten Werte ersetzt[42]. In Zeile sieben wird daraufhin der kompilierte Inhalt mit dem „data“ Objekt als Parameter ausgeführt. Dadurch werden alle Platzhalter des *Templates* durch die richtigen Werte ersetzt. Zuletzt wird der Inhalt des *Shadow Root* mit dem kompilierten Inhalt, also dem Programmcode des *Web Component*, befüllt.

4.1.5 Adaptivität der Elemente

Die Adaptivität der Elemente wird über verschiedene Aktualisierungsmethoden ermöglicht. Diese werden in der Klasse „Components“ definiert. Die Methode „update“ ändert die Eigenschaften aller *Web Components* einer bestimmten Art, zum Beispiel alle „awc-image“ Elemente. Sie erwartet zwei Parameter, den „htmlTag“ der zu verändernden Elemente und „data“, die neuen Werte der Elemente. Zu Beginn wird die Methode „_hasAdaptiveWebComps“ genutzt um abzufragen, ob der Browser die Funktionalitäten, welche zur Adaptivität benötigt werden, besitzt. Wenn dies zutrifft werden alle Elemente gesammelt welche den bestimmten „htmlTag“ besitzen. Daraufhin wird an jedes dieser Elemente ein Ereignis mit der Bezeichnung „adaptiveUpdate“ gesendet, dem das „data“ Objekt angehängt wird. In der Klasse „Component“ wird über die Registrierung eines „EventListener“ für jedes *Web Component* dieses Ereignis empfangen. Daraufhin wird die Methode „_adaptiveChangeCallback“ aufgerufen. Diese führt die zu Beginn die Methode „_pipeCallback“ mit dem Parameter „onAdaptiveChange“ aus, dadurch werden etwaige Callback Funktionen des Elements mit dem Namen dieses Parameters ausgeführt. Danach fügt sie die neuen „data“ Werte mit den vorhandenen des *Web Component* zusammen und bildet das Element mit den Werten neu.
Die Methode „updateAll“ ändert die Eigenschaften aller auf der Seite vorhandenen *Web Components*. Als Parameter erwartet sie das „data“ Objekt mit den neuen Wer-

ten für die Eigenschaften der Elemente. Wird sie aufgerufen prüft sie zu Beginn auch über die Methode „_hasAdaptiveWebComps“, ob der Browser die Funktionalitäten, welche zur Adaptivität benötigt werden, besitzt. Daraufhin werden alle *Web Components* der Anwendung gesammelt und wiederum das Ereignis „adaptiveUpdate“ an jedes dieser Elemente gesendet. Bei diesem Ereignis werden auch die neuen Werte in Form des „data“ Objekts angehängt. Dieses Ereignis wird wie in Kapitel 4.1.5 von den *Web Components* empfangen und verarbeitet. Letztendlich wird das Element mit den neuen Werten auch wieder neu gebildet und hat sich somit adaptiv angepasst.
Zuletzt muss jedes *Web Component*, beziehungsweise dessen Darstellung, auf diese Anpassung reagieren. Im Werkszustand besitzen diese verschiedene CSS-Klassen. In Listing 4.4 ist der HTML Inhalt des „awc-image“ dargestellt. In diesem besitzt

```
1 <template>
2    <img class="contrast_{{ contrast }} reading_direction_{{
     ↪  reading_direction }} size_{{ size }}"
     ↪  src="/assets/img/carole.png" alt="Carole">
3 </template>
```

Listing 4.4: HTML Programmcode des Templates des awc-image

das img-Element einige CSS Klassen. Klassen die zur Bereitstellung der Adaptivität genutzt werden, starten mit dem Namen der Präferenz und einem Unterstrich, beispielsweise „contrast_“ und sind gefolgt von einem *Handlebar* Platzhalter, dessen Inhalt wiederum der Name der Präferenz ist, hier „{{contrast}}“. Jede Neubildung oder Änderung des Elements ändert mithilfe des *Handlebar* Frameworks die CSS-Klassen des Elements. Je nach gesetzter CSS Klasse wird das Element dann anders dargestellt. In Listing 4.5 sind die Darstellungseigenschaften des *Web Component* gezeigt. Sie bestimmen inwiefern sich die Darstellung ändert. So wird in der Standarddarstellung des Kontrastwerts („contrast_“ und „contrast_normal“) ein weißer Hintergrund und kein CSS-Filter genutzt. Ist jedoch die CSS Klasse „contrast_high“ gesetzt, ist die Präferenz des Nutzers ein erhöhter Kontrastwert. Dann wird ein CSS Filter „grayscale(100%) contrast(130%)“ und eine schwarze Hintergrundfarbe genutzt. Dies erhöht den Kontrast der Darstellung des Elements.

```
.contrast_, .contrast_normal {
    -webkit-filter: none;
    filter: none;
    background: #fff;
}
.contrast_high {
    -webkit-filter: grayscale(100%) contrast(130%);
    filter: grayscale(100%) contrast(130%);
    background: #000;
}
```

Listing 4.5: CSS Programmcode des Templates des awc-image

5 Methodik dieser Arbeit

Kapitel über die Methodik

6 Adaptivität an den Nutzer

Die Reaktion einer Internetseite, beziehungsweise von deren *Web Components*, die sich adaptiv an den Nutzer anpassen basiert auf mehreren Schritten. Als Erstes werden Einstellungen und Vorlieben der Nutzer gesammelt und in einer für Maschinen verständlichen Form gespeichert. Der Ort an dem diese gespeichert werden, muss aus dem Netz erreichbar sein. Als Nächstes müssen die Elemente der Internetseite darauf vorbereitet werden sich anzupassen. Dafür werden Funktionen und Ressourcen bereitgestellt, die sich bei Bedarf anpassen können. Nachdem dies erfolgt ist, kann die Internetseite, bei Aufruf durch einen bestimmten Nutzer eine Anforderung an den Speicherort stellen, die Informationen des bestimmten Nutzers zu übertragen. Daraufhin wird ein Befehl an die Elemente der Internetseite gestellt, die zu den jeweiligen Präferenzen und damit Nutzern, passenden Funktionen auszuführen. Dabei verändert sich das Aussehen und unter Umständen der Inhalt der Seite und beweist seine Adaptivität.
Nachfolgend wird in dieser Arbeit darauf eingegangen in welcher Art sich an den Nutzer angepasst werden soll und wie diese Werte gespeichert und abgerufen werden. Als Nächstes wird konfiguriert, wie sich die *Web Components* an die entsprechenden Präferenzen anpassen werden und zuletzt wie das Ergebnis, die adaptiven Elemente einer Internetseite, aussehen wird.

6.1 Präferenzen der Nutzer

Jeder Nutzer hat eigene Vorlieben und Einschränkungen beim Nutzen einer Anwendung. Somit unterscheidet sich die gesamte Nutzerschaft. Um diese Unterscheidung abzubilden werden verschiedene Bezeichner definiert. Jeder Nutzer hat bei jedem dieser Bezeichner einen Wert. Somit wird ein Bezeichner-Wert Paar gebildet. Diese Paare werden gesammelt für jeden einzelnen Nutzer gespeichert. Hierfür

werden alle Präferenzen eines Nutzers in einer Zusammenstellung gesammelt und notiert(*Preference Set*).

6.1.1 WCAG Richtlinien

Um mehr über mögliche Präferenzen in Erfahrung zu bringen werden im folgenden die Web Content Accessibility Guidelines 2.0 (WCAG 2.0) Richtlinien betrachtet. Diese versuchen das Internet für alle Personen, unabhängig ihrer Einschränkung, zugänglicher zu machen. Dafür schaffen sie Prinzipien und Richtlinien, auf Basis derer Internetseiten konzipiert werden können. Diese können sehr gut verwendet werden um herauszufinden, in welcher Art und Weise Anwendungen sich adaptiv an verschiedene Nutzer anpassen sollten. Im Folgenden werden die Richtlinien aufgezählt und kurz beschrieben[43].

Richtlinie 1.1 Text-Alternativen Für Texte einer Anwendung sollen nicht aus Text bestehende Alternativen angeboten werden, beispielsweise große Druckbuchstaben, Brailleschrift, Sprache, Symbole oder einfachere Sprache. Parallel dazu sollen nicht textbasierte Inhalte auch durch eine textbasierte Alternative dargestellt werden können.

Richtlinie 1.2 zeitbasierte Medien Es sollen Alternativen für zeitbasierte Medien geschaffen werden.

Richtlinie 1.3 Anpassbar Die Möglichkeit den Inhalt der Internetseite in verschiedenen Arten darzustellen, ohne Informationen oder Struktur zu verlieren, soll vorhanden sein.

Richtlinie 1.4 Unterscheidbar Den Nutzern soll es erleichtert werden den Vordergrund vom Hintergrund zu unterscheiden. Dies soll bei visuellen und Audio-Elementen kongruent erfolgen.

Richtlinie 2.1 Zugänglichkeit mit der Tastatur Die gesamte Funktionalität der Anwendung sollte auch mit der Tastatur verwendet werden können.

Richtlinie 2.2 Ausreichend Zeit Den Nutzern soll genügend Zeit zur Verfügung gestellt werden um den Inhalt zu lesen und nutzen.

Richtlinie 2.3 Anfälle Der Inhalt sollte nicht in der Weise dargestellt werden, die bekannt dafür ist Anfälle auszulösen.

Richtlinie 2.4 Navigierbar Es soll den Nutzern ermöglicht werden in der Anwendung zu navigieren, Inhalt zu finden und zu ermitteln an welcher Stelle sie sich befinden.

Richtlinie 3.1 Lesbarkeit Der textbasierte Inhalt sollte lesbar und verständlich sein.

Richtlinie 3.2 Vorhersehbar Internetseiten sollen in einer vorhersehbaren Art und Weise auftreten und funktionieren.

Richtlinie 3.3 Hilfestellung bei der Eingabe Dem Nutzer soll geholfen werden, Fehler zu vermeiden und zu korrigieren.

Richtlinie 4.1 Kompatibel Die Kompatibilität mit heutigen und zukünftigen Benutzeragenten und unterstützenden Technologien soll maximiert werden.

6.1.2 Folgerung für Präferenzen

Auf Basis der vorhergehenden Kapiteln und weiterer Beobachtungen wird nun erforscht, welche Merkmale des *Web Component* sich in welcher Art und Weise anpassen soll, um eine sinnvolle Adaptivität zu erreichen.

6.1.2.1 Kontrast

Um die WCAG 2.0 Richtlinie 1.4 zu erfüllen, sollte der Kontrast sich an den Nutzer anpassen. Der Bezeichner Kontrast definiert, ob und inwiefern der Nutzer einen Kontrastwert nutzen möchte, welcher vom Standard Kontrastwert abweicht. Beispielsweise erleichtert ein hoher Kontrast farbenblinden Nutzern Objekte vom Hintergrund zu unterscheiden. Für Nutzer ohne diese Einschränkung sind zu hohe Kontraste jedoch störend, da sie das Auge schnell ermüden[44, S.234].

6.1.2.2 Schrift

Wenn davon ausgegangen wird, dass eine als allgemein lesbar bekannte Schriftart verwendet wird, hat die Wahl einer serifen oder serifenlosen Schriftart keinen be-

stätigten Einfluss auf die Richtlinie 3.1, die Lesbarkeit eines Textes[45, S.157-161]. Jedoch kann, um eine Adaptivität der Schriftart zu ermöglichen, dem Nutzer die Wahl gelassen werden, ob er eine serife oder groteske Schriftart nutzen möchte. Die Schriftgröße kann jedoch Einfluss auf die Richtlinie 3.1 nehmen. So kann eine zu große Schriftgröße, genauso wie eine zu kleine Schriftgröße, die Lesbarkeit beeinträchtigen. Insbesondere älteren Menschen oder Nutzern mit Sehbeeinträchtigungen wird das Lesen eines Textes durch eine anpassbare Schriftart erleichtert[45, S.152]. Somit ermöglicht ein Bezeichner Schriftgröße, zu welchem der Nutzer einen Wert wählen kann eine nützliche Adaptivität der Anwendung.

6.1.2.3 Leserichtung

Um die Richtlinie 3.1, die Lesbarkeit, zu erfüllen, sollte die Leserichtung sich an den Nutzer anpassen. Ohne diese Anpassung könnten Missverständnisse und Irritationen entstehen, da Texte von einer anderen Seite gelesen werden. So wird beispielsweise die arabische und hebräische von rechts oben nach links unten, die westliche von links oben nach rechts unten aufgebaut[46, S.148].

6.1.2.4 Anordnung der Elemente

Kongruent zum Kapitel 6.1.2.3 ist zu bemerken, dass die Sprache des Nutzers und damit seine Leserichtung nicht nur einen Einfluss auf die Texte haben sollte, sondern insgesamt einen Einfluss auf die Gestaltung der Internetseite haben sollte. Aufgrund der Leserichtung fallen Elemente eines Internetauftritts an verschiedenen Stellen unterschiedlich auf. So ziehen bei einer Leserichtung von links nach rechts Elemente in der linken oberen Ecke eher die Aufmerksamkeit auf sich als Elemente in der rechten oberen Ecke, da diese erst danach gelesen werden. Im Gegensatz dazu fallen bei der Leserichtung von rechts nach links Elemente in der rechten oberen Seite schneller auf[47, S.47 f.]. Ein Beispiel dazu bilden die deutsche und ägyptische Nachrichtenseite „Zeit Online“ und „Al Shaab“, dargestellt in Abbildung 6.1. Hier ist im oberen Bereich die Kopfzeile der arabischsprachigen und darunter die Kopfzeile der deutschsprachigen Internetseite abgebildet. Hier sieht man ganz klar, wie im arabischsprachigen Bereich die Elemente eher nach rechts ausgerichtet sind und wichtige Elemente, wie beispielsweise der Name der Zeitung ganz rechts oben

Abbildung 6.1: Einfluss der Leserichtung auf die Anordnung der Elemente[48][49]

angeordnet sind. Im deutschsprachigem Internetauftritt sind die Elemente nach links ausgerichtet und der Name der Webseite ist weit oben links.

6.1.2.5 Text-Alternativen

Die Richtlinie 1.1 definiert, dass zu jedem Text eine Alternative geboten wird. Darüber hinaus sollen die nicht textbasierten Inhalte nach Bedarf durch einen Text dargestellt werden können. Dem Nutzer sollte ermöglicht werden in seinen Präferenzen anzugeben, ob er den Text, oder dessen Alternativen angezeigt bekommen möchte.

6.1.2.6 Geschwindigkeit

Die Richtlinie 2.2 sichert dem Nutzer ausreichend Zeit zu, den Inhalt zu lesen und zu nutzen. Dies sollte adaptiv an den Nutzer angepasst werden, da manche Nutzer mehr oder weniger Zeit für eine bestimmte Aufgabe benötigen. Beispielsweise werden bestimmte Navigationselemente durch ein Verweilen der Maus auf diesen ausgeklappt. Sobald die Maus nicht mehr auf dem Element verweilt wird es wieder eingeklappt. Hier sollte es dem Nutzer ermöglicht werden die Zeitspanne anzupassen bevor das Element wieder einklappt, da es ihm andernfalls nicht möglich sein könnte das Element schnell genug zu betätigen.

6.1.3 Speichern der Nutzer Präferenzen

Die Präferenzen der verschiedenen Nutzer, also die Bezeichner-Wert Paare müssen gespeichert werden. Der Speicherort muss an das Netz angeschlossen sein, da die Werte zur Laufzeit der Anwendung, beziehungsweise beim Laden der Internetseite abgerufen werden müssen.

6.1.4 Herunterladen der Nutzer Präferenzen

Um die Präferenzen der Nutzer zu verwenden und auf Basis dieser die Webseite anzupassen, muss eine Möglichkeit geschaffen werden, die Zusammenstellung der Präferenzen beim Aufruf der Anwendung abzufragen. Hierfür wird ein *REST-Client* verwendet. Dieser sendet beim Aufruf der Anwendung einen Befehl an einen bestimmten, an das Netz angeschlossenen, Ort. Dabei wird diesem Ort mitgeteilt, für welche Person die Zusammenstellung angefragt wird. Daraufhin erhält die Anwendung als Antwort alle Präferenzen der mitgeteilten Person und kann auf Basis dieser die Anwendung anpassen.

6.1.4.1 Was bedeutet REST

Representational State Transfer (REST) ist ein Protokoll um Daten in einer verteilten Umgebung auszutauschen. Es basiert auf den folgenden Prinzipien[50, S.77].

Adressierbarkeit von Ressourcen Jede Ressource sollte von einem einzigartigen Bezeichner identifiziert werden können.

Einfache und einheitliche Schnittstelle Das REST Protokoll basiert auf dem HTML Protokoll. Es werden die von der Hypertext Transfer Protocol (HTTP)-Technologie bekannten Methoden verwendet. Dies macht REST Protokoll simpel und einheitlich.

Repräsentation Die Ressourcen können in verschiedenen Formen repräsentiert werden. Bei Änderung oder Anfrage der Ressource wird immer eine Repräsentation genutzt. Somit kann in einer Anfrage definiert werden, in welcher Form der Repräsentation die Antwort erfolgen soll.

Zustandslos Es werden keine Zustände auf dem *Server* gespeichert. Zustandsinformationen werden vom *Client* gehandhabt und bei Bedarf an den Server gesendet.

Cashing möglich Der *Client* sollte die Möglichkeit haben, die Antworten für späteren Gebrauch zu speichern.

6.1.4.2 Der Client

Die Funktion um die Präferenzen eines bestimmten Nutzers abzufragen basiert auf der *jQuery.ajax()* Funktion. Dies ist eine Funktion des *jQuery Frameworks* , welche eine asynchrone Anfrage an einen Ort stellt, basierend auf dem HTTP Protokoll.[51] Wie in Listing 6.1 dargestellt führt die Funktion „restCall“ eine asynchrone Anfra-

```
function restCall (url){
    jQuery.ajax({
        //URL setzen, um Wert von bestimmtem Nutzer anzufragen
        url: url,
        data: {
            format: 'json'
        },
        error: function(request, status, error) {
            //Fehlermeldung ausgeben
        },
        dataType: 'json',
        success: function(data) {
            //Speichern der empfangenen Beispieldaten
            var reading_direction = data.reading_direction;
            var contrast = data.contrast;
            //Seite entsprechend der empfangenen Präferenzen ändern
            update(reading_direction, contrast);
        },
        type: 'GET'
    });
}
```

Listing 6.1: JavaScript Programmcode zum Abfragen der Nutzerpräferenzen

ge aus. Sie erwartet ein Argument „url“. Dieses Argument wird gesetzt um einen entsprechenden Speicherort der Zusammenstellung der Präferenzen und den gewünschten Nutzer anzufragen. Ein Beispiel dieses Speicherorts könnte „http://www.speicherort.de/praeferenzen/nutzer1“ sein. In Zeile fünf und sechs wird das Datenformat der erwarteten Antwort definiert, hier wird das Format JavaScript Object Notation (JSON) genutzt. Ab Zeile acht wird die Fehlerbehandlung definiert, hier wird gewählt, was bei einem Fehler, beispielsweise einem nicht Erreichen des Speicherorts, passiert. Bei erfolgter Antwort des Speicherorts wird die „succes“ Funktion aufgerufen. Hier werden die empfangenen Daten zuerst gespeichert, um sie dann in der „update“ Funktion nutzen. Diese passt das *Web Component* adaptiv, entsprechend der Präferenzen des Nutzers an. In Zeile 19 wird definiert, welche Art eines REST Aufrufs erfolgen soll. Hier wird ein *GET* Aufruf genutzt, da Daten vom Speicherort abgerufen werden sollen.

7 Adaptivität von bestehenden Web Components

In den folgenden Kapiteln werden bestehende *Web Components* ausgewählt, untersucht und um eine Adaptivität zu ermöglichen, angepasst. Hierfür werden Elemente aus dem Katalog der Internetseite „customelements.io“ gewählt[52]. Die erste Auswahl ist das Element „google-map“. Es ist ein *Web Component*, das die Anzeige einer *Google Map*, also einer Karte, ermöglicht[53].

7.1 google-map

7.1.1 Konzeption zur Adaptivität

Abbildung 7.1 zeigt die Ansicht einer „Google Map“. Sie nutzt einerseits farbige Abbildungen und Symbole, andererseits auch verschiedene Texte. Um eine Adaptivität zu ermöglichen, sollte Einfluss auf den Kontrast, die Schrift, die Leserichtung und damit die Anordnung der Elemente und Text-Alternativen genommen werden. Der Kontrast der Kartenelemente sollte sich adaptiv an den Nutzer anpassen. Hier sollten die Straßen, Gebäude, Grünflächen und ähnliche Elemente, welche durch Farbunterschiede dargestellt werden, sich stark voneinander abheben können, um allen Nutzern das Lesen der Karte zu ermöglichen. Des weiteren sollte sich die Schrift, also die Bezeichner der Kartenelemente, wie Städte, Straßen oder besondere Gebäude anpassen. Auch sollte sich die Schrift auf den Buttons einstellen lassen. Hier sollte es dem Nutzer ermöglicht werden zwischen einer kleinen, großen und einer serifen oder grotesken Schriftart zu wählen. Um sich an die Leserichtung des Anwenders anzupassen muss zum einen der Text und zum anderen der Aufbau des *Web Components* sich verändern. So sollten das wichtigste Element, die Einstel-

Abbildung 7.1: Google Map Web Component Ansicht

lungsleiste, bei einer Leserichtung von links nach rechts in der linken oberen Ecke befinden und bei einer Leserichtung von rechts nach links in der rechten oben Ecke befinden. Dementsprechend würden sich der derzeitige, standardmäßige, Aufbau der Karte bei einer Leserichtung von rechts nach links spiegeln. Zuletzt sollten für die Texte Text-Alternativen geboten werden und Bestandteile der Karte welche eine Funktion erfüllen und nur durch Bilder oder Symbole dargestellt werden, auch als Text angeboten werden.

7.1.2 Einrichtung des Google Map Web Component

Das *Google Map Web Component* ist auf GitHub gespeichert. Um die Möglichkeit zu erlangen es anzupassen wurde ein „fork“ durchgeführt. Das bedeutet, dass der Stand der eigentlichen Entwicklern kopiert wird und eigene Änderungen im weiteren darauf aufbauen. Um die Abhängigkeiten des *Web Component* zu installieren wurde Bower genutzt. *Bower* ist ein Paketverwaltungsprogramm, welches das Aktualisieren und Installieren von Bibliotheken und Frameworks unterstützt. Dabei wird *Bower* als ein Kommandozeilenprogramm genutzt[54]. Die Abhängigkeiten setzen sich aus den folgenden Elementen zusammen:

- „Polymer/polymer#1̂.2.3“,
- „GoogleWebComponents/google-apis#1̂.1.1“,

- „PolymerElements/iron-resizable-behavior#1̂.0.0“
- „PolymerElements/iron-selector#1̂.0.5“

Wenn das *Web Component* eingebunden ist, kann es unter dem HTML Tag „<google-map>“ genutzt werden. Dieses muss mit einem *Google* API Schlüssel, mit dem Namen „Google Maps JavaScript API“ instanziiert werden. Dieser wurde über ein *Google* Konto generiert und daraufhin in das *Web Component* eingebunden.

7.1.3 Umsetzung Programmierung

Im weiteren wird die konzipierte Adaptivität des bestehenden *Web Component* im Programmcode umgesetzt.

7.1.3.1 Kontrast

Zuerst wird die Anpassung des Kontrastwerts vorgenommen. Hier sollte zu Beginn auf die Erstellung einer „Google Map“ genauer eingegangen werden. Dabei wird eine Instanz der Klasse „google.maps.Map“ erstellt. Diese kann mithilfe optionaler Parameter erstellt werden. Einer dieser Parameter sind die sogenannten „styles“. Hier können einzelne Merkmale oder die gesamte Karte mit eigenen Gestaltungseinstellungen versehen werden. Ein Merkmal sind bestimmte geographische Eigenschaften der Karte, wie beispielsweise Straßen, Parks oder Seen, sie werden als „featureType“ definiert. Danach können noch einzelne Elemente dieser Eigenschaften gewählt werden, wie beispielsweise Bezeichner oder geographische Elemente. Diese werden als „elementType“ bezeichnet. Zuletzt kann somit das Aussehen dieser Elemente der Merkmale über verschiedene Gestaltungsoptionen angepasst werden[55]. In Listing 7.1 ist ein Ausschnitt des Programmcodes der einen erhöhten Kontrast der Karte erreicht, dargestellt. Die hier erstellte Datengruppe wird als „styles“ Parameter beim Instanziieren der „google.maps.Map“ Klasse übergeben. Den Merkmalen der Karte werden verschiedene Farben zugeordnet, diese werden so gewählt, dass sie einen hohen Kontrast erzeugen und sich somit gut voneinander unterscheiden lassen. Diese Datengruppe soll jedoch nur verwendet werden wenn ein erhöhter Kontrast für den Nutzer erwünscht ist. Dafür wird dem „google-map“ Element eine neue Eigenschaft „moreContrast“ hinzugefügt. Diese ist vom Datentyp

```
var stylesArray = [{featureType: 'water', stylers: [
        {'color': '#004d99'}
        ]},{        featureType: 'landscape', stylers: [
        {'color': '#000000'}
        ]},{        featureType: 'road', stylers: [
        {'color': '#ffffff'}
        ]},{        featureType: 'road',
        elementType: 'labels.text.stroke', stylers: [
        {'color': '#00cc00'}
        ]},{        featureType: 'road',
        elementType: 'labels.text.fill', stylers: [
        {'color': '#000000'}
        ]},{        featureType: 'transit', stylers: [
        {'color': '#ffffff' }
        ]},{        featureType: 'poi',        stylers: [
        {'visibility': 'off'}
        ]},{        featureType: 'administrative',
        elementType: 'labels.text.stroke', stylers: [
        {'color': '#ffff00'}
        ]},{         featureType: 'administrative',
        elementType: 'labels.text.fill', stylers: [
        {'color': '#000000'}
        ]}
];
```

Listing 7.1: JavaScript Programmcode zur Gestaltung und Anpassung eines erhöhten Kontrastwerts

Boolean und der Standardwert ist „falsch“. Wird sie beim Einbinden des *Web Component* gesetzt, wechselt ihr Wert auf „wahr“. Die Methode „_getMapOptions()“ ist in diesem Element „google-map“ schon vorhanden. Diese erstellt eine Datengruppe „mapOptions“. Dieser wird der Parameter „styles“ übergeben. Um eine Adaptivität zu ermöglichen wird hier dem Parameter der Rückgabewert der Methode „_changeContrast()“ übergeben. Diese frägt zu Beginn ab, ob die Eigenschaft „moreContrast“ auf „wahr“ gesetzt ist. Ist dies der Fall wird die vorher beschriebene Datengruppe für den erhöhten Kontrast an den „styles“ Parameter übergeben, andernfalls wird der Standardwert überreicht. Somit wird insgesamt ein erhöhter Kontrast beim Setzen

des Parameters „moreContrast“ erreicht.

7.1.3.2 Schriftgröße

Um die Adaptivität der Schriftgröße zu ermöglichen wird die Methode „_changeFontSize()“ genutzt. Diese wartet zu Beginn auf die Benachrichtigung des Ereignisses „tilesloaded“ der „Google Map“. Diese bedeutet, dass alle Elemente der Karte geladen wurden. Daraufhin fragt sie den im Parameter „fontSize“ gesetzten Wert ab. Anhand dieses Wertes werden Schriftgrößen für die verschiedenen Elemente der Karte gesetzt. Der Parameter „fontSize“ kann die Werte „m“„l“ oder „xl“ enthalten. Von „m“ bis „xl“ werden die Schriftgrößen immer größer. Der Standardwert ist „m“, in diesem Fall wird die Schriftgröße nicht geändert. Danach werden die gesetzten Schriftgrößen auf die Kartenelemente übernommen. Darüber hinaus werden die Elemente bei Bedarf verschoben um ein Überlappen dieser zu verhindern.

7.1.3.3 Schriftart

Die Schriftart kann sich auch an den Nutzer anpassen. Dies ermöglicht die Methode „_changeFontStyle()“. Diese wartet auch auf die Benachrichtigung des Ereignisses „tilesloaded“ der „Google Map“. Daraufhin wird abgefragt ob der Parameter „fontFamily“ des *Web Component* gesetzt ist. Ist dies der Fall wird der Wert des Parameters genutzt um die Schriftart der „google-map“ auf eine serife oder serifenlose zu ändern.

7.1.3.4 Leserichtung

Die Leserichtung des Nutzers hat primär Einfluss auf den Aufbau von Texten. Da dieses *Web Component* nur einzelne Wörter enthält und keine ganzen Sätze hat die Leserichtung in diesem Zusammenhang keine große Bedeutung. Jedoch hat sie einen Einfluss auf die Anordnung der Elemente. Die Methode „_changeReadingDirection()“ wartet auf die Benachrichtigung des Ereignisses „tilesloaded“ der „Google Map“. Danach fragt sie den Wert des Parameters „readingDirection“ ab und ändert die Anordnung der Elemente falls dieser den Wert „rtl“, also Leserichtung von rechts nach links, enthält. Der Standardwert des Parameters ist „ltr“, standardmäßig ist die

Anordnung der Elemente nach diesem Wert ausgerichtet. Da die *Google Maps* API bei jedem Vergrößerungs- oder Verkleinerungsereignis die Gestaltungseigenschaften der Elemente und damit ihre Anordnung im *Web Component* neu berechnet, müssen die Auswirkungen der Leserichtung nach jedem Vergrößerungs- oder Verkleinerungsereignis neu gesetzt werden. Hierfür wird auf das Ereignis „idle“ gewartet und danach die Elemente, passend zur Leserichtung, wieder neu geordnet. Von dieser Änderung der Anordnung betroffen sind die beiden Schalter der Ansichtseinstellungen der Karte. Hier kann zwischen dem „Karte“ und „Satellit“ Wert gewählt werden. Darüber hinaus sind die Einstellungen der Vergrößerungsstufe und das Symbol der „Google Street View“ von der Änderung betroffen.

7.1.3.5 Text-Alternativen

In einem adaptiven *Web Component* sollte es ermöglicht werden, Symbole durch Texte darzustellen oder Texte durch Symbole zu ersetzen. In diesem *Web Component* wird dafür die Methode „_changeTextAlternatives“ verwendet. Sie wartet auch auf die Benachrichtigung des Ereignisses „tilesloaded“ der „Google Map“. Danach frägt sie den Wert des Parameters „textAlternatives“ ab. Dieser ist ein Parameter vom Datentyp String. Wenn dieser Parameter den Wert „text“ enthält, werden die Symbole der Kartensteuerung durch Text ersetzt. Die Steuerung der Vergrößerungsstufe erfolgt standardmäßig über ein Plus und ein Minus Symbol, wobei das Plus für Vergrößerung und das Minus für Verkleinerung steht. Diese werden durch die Texte „Vergrößern“ und „Verkleinern“ ersetzt. Da diese Texte je nach gewählter Schriftgröße eine unterschiedliche Breite und Höhe haben, siehe Kapitel 7.1.3.2, müssen die Größen der umgebenden Elemente angepasst werden. Diese bilden einen Rahmen um den Text oder ändern die Hintergrundfarbe der Texte. Dafür wird der Wert des Parameters „fontSize“ abgefragt und dementsprechend die Höhe und Breite angepasst. Die Elemente der „Google Map“ sind absolut positioniert. Deshalb läuft ein zu langer Text aus dem Sichtbereich des Nutzers oder überlappt andere Elemente. Um dies zu unterbinden wird in der „_changeTextAlternatives“ Methode abgefragt welche Leserichtung derzeit verwendet wird und aufgrund dieser Auskunft werden die Text-Alternativen so positioniert, dass sie immer komplett sichtbar sind. Das Symbol der kleinen Person für die Steuerung der „Street-View“ Funktion kann

nicht durch Text ersetzt werden. Da dieses Symbol mit der Maus auf eine bestimmte Stelle bewegt werden muss um die Funktion an genau dieser Stelle zu aktivieren, kann kein Text verwendet werden, da dieser zu breit wäre und die Stelle nicht eindeutig identifizierbar wäre.
Ist der Wert des Parameters „textAlternatives“ „symbol“, sollten alle Steuerungselemente die aus Text bestehen und durch Symbole ersetzt werden können, durch solche ausgetauscht werden. Dies trifft auf die Steuerung des Aussehens der Karte, die Elemente „Karte“ und „Satellit“ zu. Die Bibliothek „Font Awesome“ wird als Quelle für die Symbole genutzt. Diese wird als eine Formatvorlage in die HTML Seite geladen und einzelne Symbole können dann über CSS Selektoren ausgewählt werden. Für die Einstellung Karte wird das Symbol „fa-map“ genutzt und für die Einstellung Satellit das Symbol „fa-globe“.

8 Vergleich

Kapitel über den Vergleich

Literatur

[1] Wöller, W. & Kruse, J. (2014). *Tiefenpsychologisch fundierte Psychotherapie: Basisbuch und Praxisleitfaden*. Schattauer.

[2] Loeser, H. (2013). *Web-Datenbanken: Einsatz objekt-relationaler Datenbanken für Web-Informationssysteme*. Springer Berlin Heidelberg.

[3] Kobsa, A. „Adaptivität und Benutzermodellierung in interaktiven Softwaresystemen“. In: *Grundlagen und Anwendungen der Künstlichen Intelligenz: 17. Fachtagung für Künstliche Intelligenz Humboldt-Universität zu Berlin 13.–16. September 1993*. Hrsg. von Otthein Herzog, Thomas Christaller & Dieter Schütt. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, S. 152–166.

[4] Rixecker, K. (2016). *So entsteht unser Newsfeed: Der Facebook-Algorithmus im Detail*. Abgerufen am 10.11.2016
Abgerufen von http://t3n.de/news/facebook-newsfeed-algorithmus-2-577027/.

[5] Hoffmann, A. & Niemczyk, S. (2014). *Die VENUS-Entwicklungsmethode: Eine interdisziplinäre Methode für soziotechnische Softwaregestaltung*. Kassel University Press.

[6] Patel, S. K. (2015). *Learning Web Component Development*. Packt Publishing Ltd.

[7] Cooney, D. & Glazkov, D. (2012). *Introduction to Web Components*. Abgerufen am 2.11.2016
Abgerufen von https://www.w3.org/TR/2012/WD-components-intro-20120522/.

[8] C., D. (2015). *Custom Elements v0 - Chrome Platform Status*. Abgerufen am 3.11.2016
Abgerufen von https://www.chromestatus.com/feature/4642138092470272.

[9] *Custom Elements v0*. Abgerufen am 7.11.2016
Abgerufen von http://caniuse.com/%7B%5C#%7Dfeat=custom-elements.

[10] Bidelman, E. „Custom Elements v1: Reusable Web Components“. In: (2016). https://developers.google.com/web/fundamentals/getting-started/primers/customelements.

[11] *Firefox Platform Status*. Abgerufen am 3.11.2016
Abgerufen von https://platform-status.mozilla.org/%7B%5C#%7Dcustom-elements.

[12] C., D. (2016). *Custom Elements v1 - Chrome Platform Status*. Abgerufen am 3.11.2016
Abgerufen von https://www.chromestatus.com/feature/4696261944934400.

[13] Morrita. (2015). *HTML Imports - Chrome Platform Status*. Abgerufen am 3.11.2016
Abgerufen von https://www.chromestatus.com/feature/5144752345317376.

[14] *HTML templates*. Abgerufen am 3.11.2016
Abgerufen von http://caniuse.com/%7B%5C#%7Dsearch=templates.

[15] Cooney, D. & Glazkov, D. (2013). *Introduction to Web Components*. Abgerufen am 3.11.2016
Abgerufen von https://www.w3.org/TR/2013/WD-components-intro-20130606/%7B%5C#%7Ddecorator-section.

[16] *WEB COMPONENTS CURRENT STATUS*. Abgerufen am 3.11.2016
Abgerufen von https://www.w3.org/standards/techs/components%7B%5C#%7Dw3c%7B%5C_%7Dall.

[17] W., R. & K., A. (2015). *<template> Element - Chrome Platform Status*. Abgerufen am 3.11.2016
Abgerufen von https://www.chromestatus.com/feature/5207287069147136.

[18] *Firefox Platform Status*. Abgerufen am 3.11.2016
Abgerufen von https://platform-status.mozilla.org/%7B%5C#%7Dhtml-templates.

[19] *Windows 10 build 10547*. Abgerufen am 3.11.2016
Abgerufen von https://developer.microsoft.com/en-us/microsoft-edge/platform/changelog/desktop/10547/.

[20] Niwa, R. *Webkit Feature Status*. Abgerufen am 7.11.2016
Abgerufen von https://webkit.org/status/%7B%5C#%7Dfeature-shadow-dom.

[21] Hayato. (2016). *Shadow DOM v0*. Abgerufen am 7.11.2016
Abgerufen von https://www.chromestatus.com/feature/4507242028072960.

[22] Ito, H. (2016). *What's New in Shadow DOM v1 (by examples)*. Abgerufen am 9.11.2016
Abgerufen von http://hayato.io/2016/shadowdomv1/.

[23] Hayato. (2016). *Shadow DOM v1*. Abgerufen am 7.11.2016
Abgerufen von https://www.chromestatus.com/feature/4667415417847808.

[24] Denicola, D. (2016). *Custom Elements*. Abgerufen am 1.11.2016
Abgerufen von https://www.w3.org/TR/2016/WD-custom-elements-20161013/.

[25] Behrendt, B. (2016). *Application-Programming-Interface (API) Definition*. Abgerufen am 1.11.2016
Abgerufen von http://www.gruenderszene.de/lexikon/begriffe/application-programming-interface-api.

[26] Argelius, A. (2016). *Create reusable and future-proof UI components with Custom Elements v1 and Web Components*. Abgerufen am 1.11.2016
Abgerufen von https://onsen.io/blog/create-reusable-future-proof-ui-components-with-custom-elements-v1-web-components/.

[27] Glazkov, D. & Morrita, H. (2016). *HTML Imports*. Abgerufen am 2.11.2016
Abgerufen von https://www.w3.org/TR/html-imports/.

[28] Cameron, D. (2015). *HTML5, JavaScript, and jQuery 24-Hour Trainer*.

[29] Potschien, D. (2013). *HTML5: Wie das Template-Element komplexe Vorlagen ermöglicht*. Abgerufen am 2.11.2016
Abgerufen von https://www.drweb.de/magazin/html5-wie-das-template-element-komplexe-html-vorlagen-ermoeglicht-40414/.

[30] Gasston, P. (2014). *Moderne Webentwicklung: Geräteunabhängige Entwicklung – Techniken und Trends in HTML5, CSS3 und JavaScript*. dpunkt.verlag.

[31] Bidelman, E. (2016). *Shadow DOM v1: Self-Contained Web Components*. Abgerufen am 2.11.2016
Abgerufen von https://developers.google.com/web/fundamentals/getting-started/primers/shadowdom.

[32] Satrom, B. (2014). *Building Polyfills*. O'Reilly Media.

[33] *Polyfills*. Abgerufen am 7.11.2016
Abgerufen von http://webcomponents.org/polyfills/.

[34] Polymer, Authors. (2016). *Polymer Library - Polymer Project*. Abgerufen am 24.11.2016
Abgerufen von https://www.polymer-project.org/1.0/docs/devguide/feature-overview.

[35] Polymer, Authors. (2016). *Registration and lifecycle - Polymer Project*. Abgerufen am 27.11.2016
Abgerufen von https://www.polymer-project.org/1.0/docs/devguide/registering-elements.

[36] Polymer, Authors. (2016). *Declared Properties - Polymer Project*. Abgerufen am 28.11.2016
Abgerufen von https://www.polymer-project.org/1.0/docs/devguide/properties.

[37] Polymer, Authors. (2016). *Instance methods - Polymer Project*. Abgerufen am 28.11.2016
Abgerufen von https://www.polymer-project.org/1.0/docs/devguide/instance-methods.

[38] Polymer, Authors. (2016). *Local DOM Basics and API - Polymer Project*. Abgerufen am 6.12.2016
Abgerufen von https://www.polymer-project.org/1.0/docs/devguide/local-dom.

[39] Polymer, Authors. (2016). *Styling local DOM - Polymer Project*. Abgerufen am 6.12.2016
Abgerufen von https://www.polymer-project.org/1.0/docs/devguide/styling.

[40] REMEX. (2016). *REMEXLabs/AWC: Adaptive Web Components Framework*. Abgerufen am 15.12.2016
Abgerufen von https://github.com/REMEXLabs/AWC.

[41] REMEX. (2016). *Adaptive Web Components*. Abgerufen am 15.12.2016 Abgerufen von http://darwin.gpii.eu/.

[42] Handlebar, Authors. *Handlebars.js: Minimal Templating on Steroids*. Abgerufen am Abgerufen von http://handlebarsjs.com/.

[43] Caldwell, B. et al. (2008). *Web Content Accessibility Guidelines (WCAG) 2.0*. Abgerufen am 16.11.2016
Abgerufen von https://www.w3.org/TR/WCAG20/.

[44] Balzert, H., Klug, U. & Pampuch, A. (2009). *Webdesign /& Web-Usability: Basiswissen für Web-Entwickler*. W3L-Verlag.

[45] Bremus, T. (2013). *Barrierefreiheit*. entwickler.press.

[46] Emrich, C. (2013). *Interkulturelles Marketing-Management: Erfolgsstrategien – Konzepte – Analysen*. Springer Fachmedien Wiesbaden.

[47] Meidl, O. (2013). *Global Website: Webdesign im internationalen Umfeld*. Springer Fachmedien Wiesbaden.

[48] (2016). *El Shaab*. Abgerufen am 17.11.2016
Abgerufen von http://www.elshaab.org/.

[49] (2016). *Zeit Online*. Abgerufen am 17.11.2016
Abgerufen von http://www.zeit.de/index.

[50] Chauhan, S. (2014). *ASP.NET MVC Interview Questions and Answers:* Dot Net Tricks.

[51] *jQuery.ajax()*. Abgerufen am 15.11.2016
Abgerufen von http://api.jquery.com/jquery.ajax/.

[52] *Custom Elements*. Abgerufen am 21.11.2016
Abgerufen von https://customelements.io/.

[53] *google-map*. Abgerufen am 21.11.2016
Abgerufen von https://elements.polymer-project.org/elements/google-map.

[54] *Bower - a package manager for the web*. Abgerufen am 8.12.2016
Abgerufen von https://bower.io/.

[55] (2016). *Style Reference*. Abgerufen am 24.11.2016
Abgerufen von https://developers.google.com/maps/documentation/javascript/style-reference%7B%5C#%7Dstyle-elements.