Erstellung von adaptiven Web Components

Christoph Kleber

24. November 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

1 Inhaltsverzeichnis

2 Adaptivität

2.1 Begriffsklärung

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 breiten 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, dies kann beispielsweise eine Person oder ein System sein, sich an seine Umgebung anzupassen. Diese Anpassung basiert auf bestimmten Einflüssen, so kann sich eine Datenbank an äußere Einflüsse, wie beispielsweise ihre Einsatzumgebung anpassen oder ein Softwaresystem an die Vorlieben seines Nutzers anpassen. Dies kann automatisch geschehen. So passt sich beispielsweise der Inhalt der Seite „Facebook“ aufgrund eines Algorithmus an den einzelnen Nutzer an, ohne dass dieser bestimmte Einstellun-

gen vornehmen muss.[4, vgl.] Die Möglichkeit zur Adaptivität kann jedoch auch dem Nutzer bereitgestellt werden. Ist dies der Fall, kann der Nutzer beispielsweise die Benutzeroberfläche an seine eigenen Vorlieben anpassen. So kann zum Beispiel die Benutzeroberfläche der Entwicklungsumgebung „JetBrains PhpStorm“ anhand persönlicher Präferenzen angepasst werden und Farbstile und Kontraste angepasst werden.
Hier stellt sich jedoch die Frage, wie im Zusammenhang von Internetanwendungen eine Adaptivität bereitgestellt werden kann und insbesondere, an welche Aspekte sie sich anpassen soll. Hierfür muss die Umgebung, der Nutzer und auch dessen Arbeitsaufgaben erforscht und definiert werden.

2.2 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.

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 sollten die Ziele der Nutzer beschrieben werden, auf der anderen Seite die Gesamtziele des Systems. Danach sollten 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.[5, vgl. S.15 ff.]

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.

3 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, vgl. S. 1] Sie setzen sich zusammen aus den vier Technologien *Custom Elements*, *Hypertext Markup Language (HTML) Imports*, *Templates* und *Shadow DOM*. Das Nutzen dieser Technologie soll Applikationen im Web leichter wiederverwertbar, wartbar, unabhängiger und kapselbar machen.[6, vgl. S.2]

3.1 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, vgl.] 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.

3.1.1 Custom Elements !v0 und v1!

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 unterstützt.[8, vgl.] Im *Android* Browser wird diese Version schon seit 2014 unterstützt, in der *Android* Version 4.4.4. Im *Samsung Internet* wird sie seit 2016 in der Version 4 genutzt.[9, vgl.] Die Version v0 wird von der Version v1 abgelöst, hier ergeben sich einige Änderungen in der Syntax der Advanced Programming Interface (API).[10, vgl.] Derzeit wird sie nicht per Standardeinstellung von anderen Browsern unterstützt.[11, vgl.] Die Version v1 wird derzeit nur von den Browsern *Chrome* und *Opera* unterstützt. Das erste mal wurde sie im Jahr 2016 in der *Chrome* Version 54 und in der *Opera* Version 41 genutzt. [12, vgl.]

3.1.2 HTML Imports

HTML Imports wurden in den Browsern *Chrome* und *Opera* zuerst 2014 unterstützt. 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, vgl.] Der *Android* Browser unterstützt *HTML Imports* seit 2016 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, vgl.]

3.1.3 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 dieser 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.“]Cooney2013 Auf einer aktuellen Übersichtsseite des Konsortiums wird die *Decorators* Technologie nicht mehr im Zusammenhang mit *Web Components* erwähnt.[15, vgl.]

3.1.4 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 unterstützt.[16, vgl.][17, vgl.] Im Jahr 2015 wurden sie dann vom *Edge* Browser unterstützt, in der Version 13.[18, vgl.] 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, vgl.] 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, vgl.]

3.1.5 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.[19, vgl.] Die mobilen Varianten der Browser unterstützen das *Shadow DOM* seit 2016, *Opera* in der Version 37 und *Chrome* in der Version 53, somit auch der *Android* Browser.[20, vgl.] Die Version v0 wird von der Version v1 abgelöst, welche verschiedene Neuerungen in der Syntax, aber auch in der Unterstützung 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.[21, vgl.] 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 unterstützt, jeweils seit 2016. In den mobilen Versionen wird sie nur von dem *Chromium* und somit auch auf *Android* in der Version 53 unterstützt.[22, vlg.]

3.2 Vergleich Webentwicklung mit Web Components und ohne

Das *Frontend* von modernen Webseiten basiert heutzutage hauptsächlich auf den Technologien HTML, Cascading Style Sheets (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“.[23, S.15] Ein *Framework* ist somit ein System, das den Entwicklern bestimmte Funktionalitäten zur Verfügung stellt, ohne dass dieser sie selbst programmieren muss. Diese 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 diese sich gegenseitig beeinflussen. So können die Stil-Regeln verschiedener Teile der Webseite sich unbeabsichtigt beeinflussen oder das JavaScript, welches eine bestimmte Funktion hat, an einer anderen Stelle für welche es nicht programmiert wurde, Einfluss nehmen. 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, welche eine Kapselung, eine Wiederverwendung und eine leichtere Wartbarkeit von Programmcode ermöglichen sollen.

3.2.1 Vorteil: Wenn Browserunterstützung gegeben: native, kein Framework

Aufgrund der Erkenntnisse der Geschichte von *Web Components* in Kapitel 3.1 ist eine große 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 werden und weniger, beziehungsweise unter Umständen keine *Frameworks* genutzt werden. Dies verkleinert das Laden von externen Programmcode. Darüber hinaus 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.

3.2.2 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, vgl. S.2] Dies hat einige Folgen für das Verhalten einer Applikation. Zuerst ist der *Shadow 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. Des weiteren hat das CSS nur Zugriff auf den DOM des eigenen Geltungsbereichs, weder von außerhalb des *Shadow DOM* können Stil Regeln Einfluss auf diesen nehmen, noch können Stil Regeln von innerhalb nach außen Einfluss nehmen. Ein Vorteil an dieser Eigenschaft ist, dass man atomare, also sehr einfache, CSS Bezeichner innerhalb des *Shadow DOM* verwenden kann und dieselben Bezeichner gleichzeitig außerhalb dieses nutzen kann.[24, vgl.] Darüber hinaus wird die Gestaltung der Erscheinung der Applikation konsistenter, sie erfolgt einzeln für jedes *Web Component* und für den Bereich außerhalb der *Web Components*.

3.2.3 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.

3.2.4 Wartbarkeit

Die Wartbarkeit von Web-Applikationen wird erleichtert, da die *Web Components* in *Templates* organisiert sind.S.2]patel2015learning Das sorgt dafür dass der Programmcode einzelner Komponenten separat gespeichert wird und somit leichter wiedergefunden und geändert werden kann.

3.2.5 Browserunterstützung

Auch wenn *Web Components* sehr viele Vorteile aufweisen, ist es ein Problem wenn die Techniken (noch) nicht in allen Browsern unterstützt werden oder unterschiedlich implementiert sind. Wie in Kapitel 3.1 dargelegt, sind 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. Dies kann jedoch umgangen werden, indem *Polyfills* verwendet werden. Das sind in diesem Zusammenhang Programmcodes, welche die Funktionen oder Teile einer Technologie, die nativ noch nicht von einem Browser unterstützt wird, nachstellen. Diese können dann verwendet werden um Nutzern aller Browser den Gebrauch Technologien zu ermöglichen.[25, vgl. S.4] Das *webcomponents.js* ist ein Set von *Polyfills* und ermöglicht den Nutzern die Verwendung von *Web Components* in allen modernen Browsern.[26, vgl.]

3.3 Technik der Web Components

3.3.1 Custom Elements

Das *Custom Element* ist eine *API*, welches das Bilden eigener, voll funktionstüchtiger *DOM* Elemente ermöglicht.[27, vgl.] Die API beschreibt in diesem Zusammenhang eine Schnittstelle, welche einem anderen Programm ein Werkzeug zur Verfügung stellt, um sich an das eigene Softwaresystem anbinden zu können.[28, vgl.] 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.[29, vgl.] 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 3.1 ist ein JavaScript Programmcode dargestellt, welcher ein leeres *Custom Element* definiert und und unter dem Namen „new-custom-element“ registriert wird. Für *Custom Elements* sind

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

Listing 3.1: Custom Element JavaScript

mehrere *Callbacks* verfügbar. *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.[29, vgl.]

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. Sie wird jedoch nur für Attribute aufgerufen, welche in einer statischen *get* Funktion mit Namen *observedAttributes* definiert wurden.

3.3.2 HTML Imports

HTML Imports ist eine Technologie zum Importieren von externen HTML Dokumenten in ein HTML Dokument. Hier ist zu unterscheiden zwischen importierenden und importierten HTML Dokumenten. Die importierenden Dokumente besitzen einen Link, welcher mindestens die Uniform Resource Locator (URL) des *Imports* und die Eigenschaft *rel=„import“* besitzt,also ein Link eines bestimmten Typ ist, siehe Listing 3.2.[30, vgl.] Die importierten Dokumente haben keinen außergewöhnlichen Aufbau

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

Listing 3.2: Standard HTML Import

im Vergleich zu normalen HTML Dokumenten, sie können aus HTML, CSS oder JavaScript Elementen bestehen. Es kann auch die *Template* Technologie verwendet werden, dazu mehr in Kapitel 3.3.3. Um auf den Inhalt des importierten Dokuments zuzugreifen wird JavaScript verwendet. Wie in Listing 3.3 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 *„elemt“* gespeichert. Hier wird ein div Element, welches die Klasse *„element“* besitzt gespeichert. Dieses kann dann in der importierenden Seite genutzt werden.

3.3.3 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.[31, vgl. S.177] Dies bedeutet, dass

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

Listing 3.3: JavaScript Code für Zugriff auf Inhalt des importierten Dokuments

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.[32, vgl.] In 3.4 sieht man den *JavaScript* Programmcode

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

Listing 3.4: JavaScript Code für das Hinzufügen eines Templates in das DOM

um ein vorhandenes *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 auch die Bilder des *Templates* geladen und eventuelle Skripte ausgeführt.

3.3.4 Shadow DOM

„Das *Shadow DOM* beschreibt die Fähigkeit eines Browsers, eine neue, völlig abgekapselte Knotenstruktur im bestehenden DOM zu erzeugen.“[33, 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*. Dies ist in Abbildung 3.1 dargestellt.[6, vgl. S.22] Die Folge dieser Kapselung ist, dass alles was dem *Shadow tree* hinzugefügt wird, nur lokal Einfluss auf diesen 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

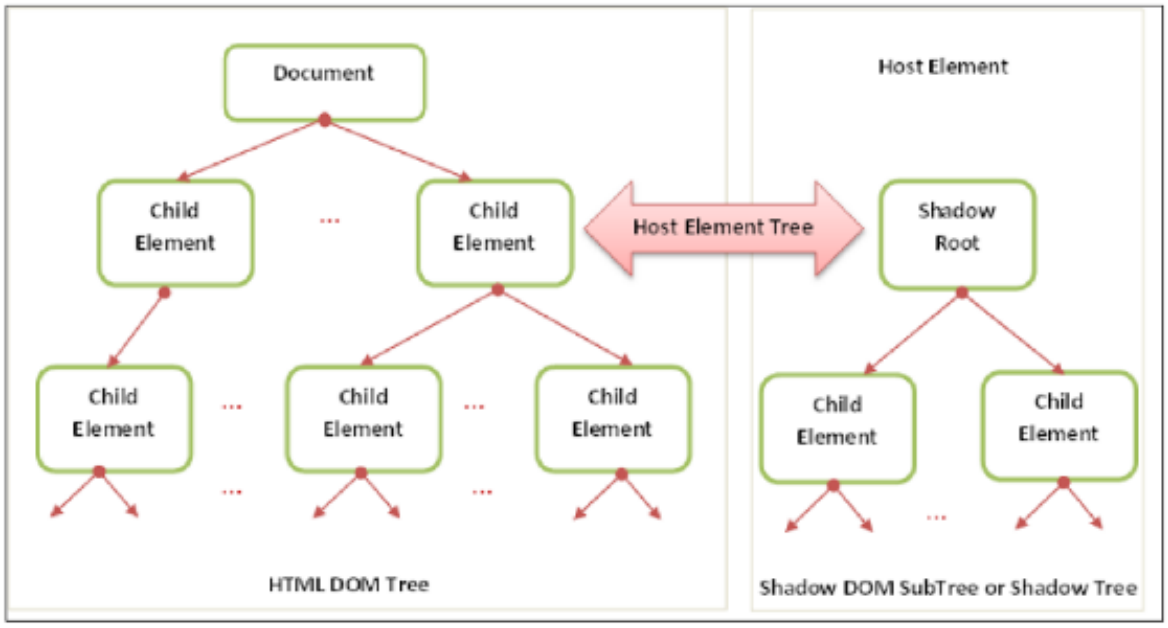

Abbildung 3.1: DOM und Shadow DOM [6, S. 22]

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.[24, vgl.] In Listing 3.5 ist dargestellt, wie Mithilfe

```
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 3.5: JavaScript Code für das Erstellen eines Shadow DOM

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 Über-

schrift in den *Shadow DOM* eingefügt.

3.3.4.1 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 3.6 ist der HTML Programmcode des *Web Com-*

```
1 <ul id='contacts'>
2    <li>
3       <slot name='title'>Kein Titel</slot>
4       <slot name='name'>
5          <p>Achtung<p>
6          <h1>Kein Name<h1>
7       </slot>
8    </li>
9 </ul>
```

Listing 3.6: Nutzung von Slot Platzhalter-Elementen im Shadow DOM

ponent 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 eigenen DOM Baum bestehen, wie im *Slot* „name“ zu sehen ist.[24, vgl.] In Listing 3.7 werden die, in Listing 3.6 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, welche 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 Listing 3.8 ist der von den zwei vorhergehenden Listings kombinierte und gerenderte Inhalt zu sehen. Dies ist der DOM, den man beispielsweise in

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

Listing 3.7: Befüllen der Slot Elemente im DOM

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 übersetzt.

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

Listing 3.8: Gerenderter DOM

4 Methodik dieser Arbeit

5 Adaptivität an den Nutzer

Das Reaktion einer Internetseite, beziehungsweise von deren *Web Components* die sich adaptiv an den Nutzer anpassen basiert auf drei 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, adaptive Elemente von Webseiten, aussehen wird.

5.1 Präferenzen von Nutzern

Um eine Adaptivität zu ermöglichen, muss definiert werden, welche Bestandteile der Anwendung sich dynamisch verhalten, also an den Nutzer anpassen. Dazu mehr in Kapitel 6.2.1.1. Vorher muss definiert werden, wie die Applikation insgesamt reagiert, also inwiefern sie ihren Zustand an den jeweiligen Nutzer anpasst. Hierfür

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

5.2 Identifikation passender Preference Terms

Jeder Nutzer hat eigene Vorlieben und Einschränkungen beim Nutzen einer Anwendung. Somit unterscheidet sich die gesamte Nutzerschaft. Um diese abzubilden werden verschiedene Bezeichner definiert. Jeder Nutzer hat bei jedem dieser Bezeichner einen Wert. Somit wird ein Bezeichner-Wert Paar gebildet. Hier stellt sich die Frage, auf welcher Basis die Vorlieben und Einschränkungen definiert und organisiert werden.

5.2.1 WCAG Guidelines

Die Web Content Accessibility Guidelines 2.0 (WCAG 2.0) versuchen das Internet für alle Personen, unabhängig ihrer Einschränkung, zugänglicher zu machen. Hierfür 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.

5.2.1.1 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.

5.2.1.2 Richtlinie 1.2 zeitbasierte Medien

Es sollen Alternativen für zeitbasierte Medien geschaffen werden.

5.2.1.3 Richtlinie 1.3 Anpassbar

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

5.2.1.4 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.

5.2.1.5 Richtlinie 2.1 Zugänglichkeit mit der Tastatur

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

5.2.1.6 Richtlinie 2.2 Ausreichend Zeit

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

5.2.1.7 Richtlinie 2.3 Anfälle

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

5.2.1.8 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.

5.2.1.9 Richtlinie 3.1 Lesbarkeit

Der textbasierte Inhalt sollte lesbar und verständlich sein.

5.2.1.10 Richtlinie 3.2 Vorhersehbar

Internetseiten sollen in einer vorhersehbaren Art und Weise auftreten und funktionieren.

5.2.1.11 Richtlinie 3.3 Hilfestellung bei der Eingabe

Dem Nutzer soll geholfen werden, Fehler zu vermeiden und zu korrigieren.

5.2.1.12 Richtlinie 4.1 Kompatibel

Die Kompatibilität mit heutigen und zukünftigen Benutzeragenten und unterstützenden Technologien soll maximiert werden. [34, vgl.]

5.2.2 Folgerung für Merkmale der Anwendung

5.2.2.1 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.[35, vgl S.234]

5.2.2.2 Schrift

Die Wahl einer serifen oder grotesken, also serifenlosen Schriftart hat, wenn davon ausgegangen wird eine als allgemein lesbar bekannte Schriftart zu verwenden, keinen bestätigten Einfluss auf die Richtlinie 3.1, die Lesbarkeit eines Textes.[36, vgl. 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 auch 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.[36, vgl. S.152] Somit ermöglicht ein Bezeichner Schriftgröße, zu welchem der Nutzer einen Wert wählen kann eine nützliche Adaptivität der Anwendung.

5.2.2.3 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, geschrieben [37, vgl S.148]

5.2.2.4 Anordnung der Elemente

Kongruent zum Kapitel 5.2.2.3 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.[38, vgl. S.47 f.] Ein Beispiel dazu bilden die deutsche und ägyptische Nachrichtenseite „Zeit Online“ und „Al Shaab“, dargestellt in Abbildung 5.1. 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 angeordnet sind. Im deutschsprachigem Internetauftritt sind die Elemente nach links ausgerichtet und der Name der Webseite ist weit oben links.

Abbildung 5.1: Einfluss der Leserichtung auf die Anordnung der Elemente[39][40]

5.2.2.5 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.

5.2.2.6 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 manchen 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 nutzen.

5.2.3 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.

5.2.4 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.

5.2.4.1 Was bedeutet REST

Representational State Transfer (REST) ist ein Protokoll um Daten in einer verteilten Umgebung auszutauschen. Es basiert auf den folgenden Prinzipien.

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. [41, vgl. S.77]

5.2.4.2 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.[42, vgl.] Wie in Listing 5.1 dargestellt führt die Funktion „restCall“ eine asyn-

```
1  function restCall (url){
2      jQuery.ajax({
3          //URL setzen, um Wert von bestimmtem Nutzer anzufragen
4          url: url,
5          data: {
6              format: 'json'
7          },
8          error: function(request, status, error) {
9              //Fehlermeldung ausgeben
10         },
11         dataType: 'json',
12         success: function(data) {
13             //Speichern der empfangenen Beispieldaten
14             var reading_direction = data.reading_direction;
15             var contrast = data.contrast;
16             //Seite entsprechend der empfangenen Präferenzen ändern
17             update(reading_direction, contrast);
18         },
19         type: 'GET'
20     });
21 }
```

Listing 5.1: Programmcode zum Abfragen der Nutzerpräferenzen

chrone Anfrage 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.

6 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.[43]

6.1 Die Elemente

Die erste Auswahl ist das Element „google-map“. Es ist ein *Web Component*, der die Anzeige einer *Google Map*, also einer Karte, ermöglicht. Diese Karte kann personalisiert und durch Wegbeschreibungen oder Markierungen ergänzt werden.[44]

6.2 Anpassung dieser Web Components

6.2.1 google-map

6.2.1.1 Konzeption zur Adaptivität

Abbildung 6.1 zeigt die Ansicht einer „Google Map“, im weiteren Karte genannt. 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

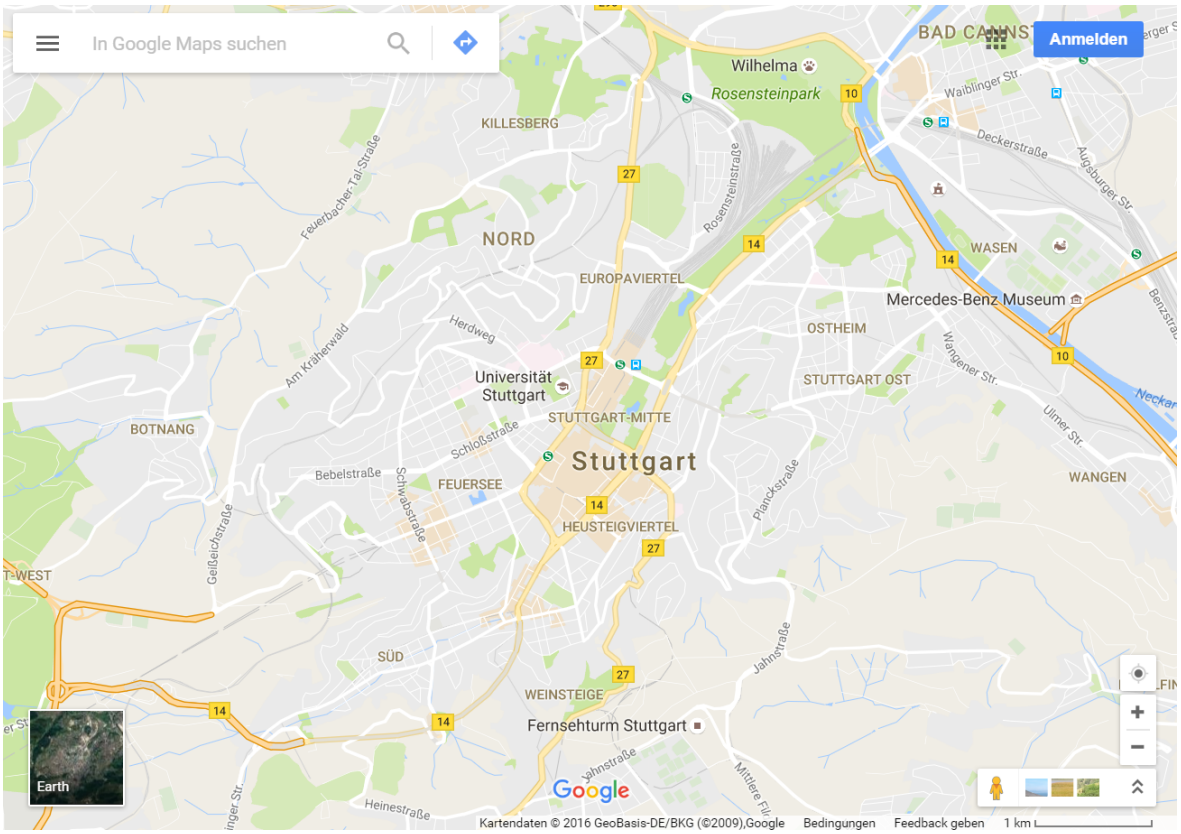

Abbildung 6.1: Google Map Ansicht[45]

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 und in der Suchleiste angleichen. Hier sollte es dem Nutzer ermöglicht werden zwischen einer kleinen oder großen und einer serifen oder grotesken Schriftart zu wählen. Um sich an die Leserichtung des Anwenders anzupassen muss zum einen der Text angepasst werden und zum anderen der Aufbau des *Web Components* sich verändern. So sollten das wichtigste Element, die Suchleiste inklusive der Einstellungen, 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 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.

6.2.1.2 Umsetzung Programmierung

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.[46] In Listing 6.1 ist der Programmcode, 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.

6.2.2 Web Component Zwei

6.2.3 Web Component Drei

```
var stylesArray = [{
   featureType: 'water',
   stylers: [
   {'color': '#000000'}
   ]},{
   featureType: 'landscape',
   stylers: [
   {'color': '#000000'}
   ]},{
   featureType: 'road',
   stylers: [
   {'color': '#ffffff'}
   ]},{
   featureType: 'transit',
   stylers: [
   {'color': '#ffffff' }
   ]
   },{
   featureType: 'poi',
   stylers: [
   {'visibility': 'off'}
   ]},{
   featureType: 'administrative',
   stylers: [
   {'color': '#ffff00'}
   ]}
];
```

Listing 6.1: Der Programmcode um die Gestaltung an einen erhöhten Kontrastwert anzupassen

7 Vergleich

7.1 Polymer

Die *Polymer* Bibliothek stellt eine Sammlung von Funktionen bereit um *custom elements* zu erstellen. Diese Elemente sollen letztendlich wie normale DOM Bestandteile funktionieren.[47] Die verwendeten Technologien der Bibliothek ähneln den *Web Components*, oder es werden direkt die Technologien der *Web Components* genutzt.

7.1.1 Polymer Technologie

Literatur

[1] W Wöller und J Kruse. *Tiefenpsychologisch fundierte Psychotherapie: Basisbuch und Praxisleitfaden*. Schattauer, 2014. ISBN: 9783794530694.

[2] H Loeser. *Web-Datenbanken: Einsatz objekt-relationaler Datenbanken f{ü}r Web-Informationssysteme*. Informationstechnologien f{ü}r die Praxis. Springer Berlin Heidelberg, 2013. ISBN: 9783642594908.

[3] Alfred Kobsa. „Adaptivit{ä}t und Benutzermodellierung in interaktiven Softwaresystemen“. In: *Grundlagen und Anwendungen der K{ü}nstlichen Intelligenz: 17. Fachtagung f{ü}r K{ü}nstliche Intelligenz Humboldt-Universit{ä}t zu Berlin 13.–16. September 1993*. Hrsg. von Otthein Herzog, Thomas Christaller und Dieter Schütt. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, S. 152–166. ISBN: 978-3-642-78545-0. DOI: 10.1007/978-3-642-78545-0_9. URL: http://dx.doi.org/10.1007/978-3-642-78545-0%7B%5C_%7D9.

[4] Kim Rixecker. *So entsteht unser Newsfeed: Der Facebook-Algorithmus im Detail*. 2016. URL: http://t3n.de/news/facebook-newsfeed-algorithmus-2-577027/ (besucht am 10. 11. 2016).

[5] A Hoffmann und S Niemczyk. *Die VENUS-Entwicklungsmethode: Eine interdisziplin{ä}re Methode f{ü}r soziotechnische Softwaregestaltung*. ITeG Technical Reports. Kassel University Press, 2014. ISBN: 9783862195503.

[6] Sandeep Kumar Patel. *Learning Web Component Development*. Community experience distilled. Packt Publishing Ltd, 2015. ISBN: 9781784395568.

[7] Dominic Cooney und Dimitri Glazkov. *Introduction to Web Components*. 2012. URL: https://www.w3.org/TR/2012/WD-components-intro-20120522/ (besucht am 02. 11. 2016).

[8] Dominic C. *Custom Elements v0 - Chrome Platform Status*. 2015. URL: https://www.chromestatus.com/feature/4642138092470272 (besucht am 03. 11. 2016).

[9] ACHTUNGcaniuse. *Custom Elements v0*. 9999. URL: http://caniuse.com/%7B%5C#%7Dfeat=custom-elements (besucht am 07. 11. 2016).

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

[11] ACHTUNGfirefox. *Firefox Platform Status*. 9999. URL: https://platform-status.mozilla.org/%7B%5C#%7Dcustom-elements (besucht am 03. 11. 2016).

[12] Dominic C. *Custom Elements v1 - Chrome Platform Status*. 2016. URL: https://www.chromestatus.com/feature/4696261944934400 (besucht am 03. 11. 2016).

[13] Morrita. *HTML Imports - Chrome Platform Status*. 2015. URL: https://www.chromestatus.com/feature/5144752345317376 (besucht am 03. 11. 2016).

[14] ACHTUNGcaniuse. *HTML templates*. 9999. URL: http://caniuse.com/%7B%5C#%7Dsearch=templates (besucht am 03. 11. 2016).

[15] ACHTUNGw3c. *WEB COMPONENTS CURRENT STATUS*. 9999. URL: https://www.w3.org/standards/techs/components%7B%5C#%7Dw3c%7B%5C_%7Dall (besucht am 03. 11. 2016).

[16] Rafael W und Adam K. *<template> Element - Chrome Platform Status*. 2015. URL: https://www.chromestatus.com/feature/5207287069147136 (besucht am 03. 11. 2016).

[17] ACHTUNGfirefox. *Firefox Platform Status*. 9999. URL: https://platform-status.mozilla.org/%7B%5C#%7Dhtml-templates (besucht am 03. 11. 2016).

[18] ACHTUNGmicrosoft. *Windows 10 build 10547*. 9999. URL: https://developer.microsoft.com/en-us/microsoft-edge/platform/changelog/desktop/10547/ (besucht am 03. 11. 2016).

[19] Ryosuke Niwa. *Webkit Feature Status*. 9999. URL: https://webkit.org/status/%7B%5C#%7Dfeature-shadow-dom (besucht am 07. 11. 2016).

[20] Hayato. *Shadow DOM v0*. 2016. URL: https://www.chromestatus.com/feature/4507242028072960 (besucht am 07. 11. 2016).

[21] Hayato Ito. *What's New in Shadow DOM v1 (by examples)*. 2016. URL: http://hayato.io/2016/shadowdomv1/ (besucht am 09. 11. 2016).

[22] Hayato. *Shadow DOM v1*. 2016. URL: https://www.chromestatus.com/feature/4667415417847808 (besucht am 07. 11. 2016).

[23] P.M.P.P.M.P.P.M.I.R.M.P. Stuart Brunt. *A Roadmap to Cracking the PMP{®} Exam: A PMP Exam Preparation Study Guide*. Trafford Publishing, 2013. ISBN: 9781466985209.

[24] Eric Bidelman. *Shadow DOM v1: Self-Contained Web Components*. 2016. URL: https://developers.google.com/web/fundamentals/getting-started/primers/shadowdom (besucht am 02. 11. 2016).

[25] B Satrom. *Building Polyfills*. O'Reilly Media, 2014. ISBN: 9781449370718.

[26] O. V. *Polyfills*. URL: http://webcomponents.org/polyfills/ (besucht am 07. 11. 2016).

[27] Domenic Denicola. *Custom Elements*. 2016. URL: https://www.w3.org/TR/2016/WD-custom-elements-20161013/ (besucht am 01. 11. 2016).

[28] Björn Behrendt. *Application-Programming-Interface (API) Definition*. 2016. URL: http://www.gruenderszene.de/lexikon/begriffe/application-programming-interface-api (besucht am 01. 11. 2016).

[29] Andreas Argelius. *Create reusable and future-proof UI components with Custom Elements v1 and Web Components*. 2016. URL: https://onsen.io/blog/create-reusable-future-proof-ui-components-with-custom-elements-v1-web-components/ (besucht am 01. 11. 2016).

[30] Dimitri Glazkov und Hajime Morrita. *HTML Imports*. 2016. URL: https://www.w3.org/TR/html-imports/ (besucht am 02. 11. 2016).

[31] Dane Cameron. *HTML5, JavaScript, and jQuery 24-Hour Trainer*. 2015.

[32] Denis Potschien. *HTML5: Wie das Template-Element komplexe Vorlagen ermöglicht*. 2013. URL: https://www.drweb.de/magazin/html5-wie-das-template-element-komplexe-html-vorlagen-ermoeglicht-40414/ (besucht am 02. 11. 2016).

[33] Peter Gasston. *Moderne Webentwicklung: Geräteunabhängige Entwicklung – Techniken und Trends in HTML5, CSS3 und JavaScript*. dpunkt.verlag, 2014. ISBN: 9783864914652.

[34] Ben Caldwell u. a. *Web Content Accessibility Guidelines (WCAG) 2.0*. 2008. URL: https://www.w3.org/TR/WCAG20/ (besucht am 16. 11. 2016).

[35] H Balzert, U Klug und A Pampuch. *Webdesign /& Web-Usability: Basiswissen für Web-Entwickler*. W3L-Verlag, 2009. ISBN: 9783868340112.

[36] Timm Bremus. *Barrierefreiheit*. 2013. ISBN: 978-3-86802-095-3.

[37] C Emrich. *Interkulturelles Marketing-Management: Erfolgsstrategien – Konzepte – Analysen*. Springer Fachmedien Wiesbaden, 2013. ISBN: 9783658030339.

[38] O Meidl. *Global Website: Webdesign im internationalen Umfeld*. Springer Fachmedien Wiesbaden, 2013. ISBN: 9783658028671. URL: https://books.google.de/books?id=stEwwZ71rWoC.

[39] O. V. *El Shaab*. 2016. URL: http://www.elshaab.org/ (besucht am 17. 11. 2016).

[40] O. V. *Zeit Online*. 2016. URL: http://www.zeit.de/index (besucht am 17. 11. 2016).

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

[42] O. V. *jQuery.ajax()*. URL: http://api.jquery.com/jquery.ajax/ (besucht am 15. 11. 2016).

[43] *Custom Elements*. URL: https://customelements.io/ (besucht am 21. 11. 2016).

[44] *google-map*. URL: https://elements.polymer-project.org/elements/google-map (besucht am 21. 11. 2016).

[45] *Google Maps*. URL: https://www.google.de/maps/@48.7791843,9.1780412,13z (besucht am 21. 11. 2016).

[46] *Style Reference*. 2016. URL: https://developers.google.com/maps/documentation/javascript/style-reference%7B%5C#%7Dstyle-elements.

[47] *Polymer Library - Polymer Project*. URL: https://www.polymer-project.org/1.0/docs/devguide/feature-overview (besucht am 24. 11. 2016).