

Bachelorarbeit

im Studiengang Mobile Medien

vorgelegt von

Christoph Kleber

Matrikelnummer: 26982

am 31.01.2017

an der Hochschule der Medien Stuttgart
zur Erlangung des akademischen Grades
eines Bachelor of Science

**Konzeption und Entwicklung
von adaptiven Web Components**

Erst-Prüfer:

Prof. Dr. Gottfried Zimmermann

Zweit-Prüfer:

Darius Morawiec

Eidesstattliche Erklärung

Hiermit versichere ich, Christoph Kleber, ehrenwörtlich, dass ich die vorliegende Bachelorarbeit mit dem Titel: „Konzeption und Entwicklung von adaptiven Web Components“ selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Ich habe die Bedeutung der ehrenwörtlichen Versicherung und die prüfungsrechtlichen Folgen (§26 Abs. 2 Bachelor-SPO (6 Semester), § 24 Abs. 2 Bachelor-SPO (7 Semester), § 23 Abs. 2 Master-SPO (3 Semester) bzw. § 19 Abs. 2 Master-SPO (4 Semester und berufsbegleitend) der HdM) einer unrichtigen oder unvollständigen ehrenwörtlichen Versicherung zur Kenntnis genommen.

Stuttgart, den _____

(Christoph Kleber)

Kurzfassung

In dieser Bachelorarbeit wird ein relativ junges Themengebiet der *Frontend* Webentwicklung, die *Web Components* behandelt. Parallel wird auf die Problemstellung der adaptiven Webentwicklung, also der Anpassung des Webauftritts an den jeweiligen Nutzer und dessen Präferenzen eingegangen und diese Problemstellung auf die *Web Components* angewendet. Zuerst wird konzipiert, wie sich eine Adaptivität ausdrücken und wie sie umgesetzt werden kann. Hier ist erarbeitet worden, dass Nutzer einen Satz an Präferenzen mit jeweiligem Wert besitzen. Dieser Satz wird gespeichert und beim Aufruf der Webseite abgefragt. Anhand der Werte soll das Aussehen und der Aufbau der Komponenten einer Webseite angepasst werden. In den Kapiteln der *Web Components* wird zuerst deren Geschichte und Technologie behandelt. Danach wird auf verschiedene *Frameworks* aus diesem Segment eingegangen. Aufgrund der vorhergehenden Ergebnisse der gesamten Kapitel werden dann vorhandene *Web Components* gewählt und diese zur erfolgreichen Adaptivität angepasst. Zuletzt wird evaluiert inwiefern diese Anpassung erfolgen konnte und dass *Web Components*, unter guten Voraussetzungen, zur Adaptivität geeignet sind.

Abstract

The subject of this bachelor thesis is a relative new topic of frontend web development, the *Web Components*. Parallel the problem statement of adaptive web development, the adaption of a website to its user and his preferences, is addressed and conveyed to *Web Components*. First a concept is designed, how adaptivity can be expressed and implemented. The outcome of this is that users have a set of preferences with a certain value to each preference. This set is saved and requested by invocation of the website. Based on these values the appearance and structure of the website should be adapted. In the chapters about *Web*

Components first their history and technology is covered. Subsequently different frameworks of this subject are approached. On grounds of preceding outcomes of all chapters, existing *Web Components* are chosen and adjusted to successful adaptivity. Last it is evaluated how far these adjustments could be ensued and that, under good preconditions, *Web Components* are applicable to adaptivity.

Listing Verzeichnis

5.1. Beispiel eines Preference Sets	22
5.2. Abfragen der Nutzerpräferenzen	25
6.1. Erstellung Custom Element	30
6.2. Standard HTML Import	31
6.3. Zugriff auf importiertes Dokument	31
6.4. Hinzufügen eines Templates	32
6.5. Erstellung eines Shadow DOM	33
6.6. Slot Elemente im Shadow DOM	33
6.7. Befüllen der Slot Elemente im DOM	34
6.8. Übersetzter DOM	34
6.9. Der :host Selektor	35
6.10. Beispiel Web Component	36
6.11. Nutzung des example-component	37
7.1. Polymer custom element Registrierung	42
7.2. Polymer properties Objekt	44
7.3. Polymer Instanz-Methoden	44
7.4. Polymer dom-module	45
7.5. Benutzerdefinierte CSS Eigenschaften	46
8.1. Web Component basierend auf Darwin.js	48
8.2. Erstellung des shadow roots	49
8.3. Bilden des Web Components	49
8.4. Das awc-image Element	51
8.5. CSS Programmcode awc-image	51
8.6. Das awc-input Element	52
8.7. JavaScript Programmcode awc-input	52
9.1. CSS des Paper Date Picker für serife Schriftart	62
9.2. Die Methode _changeFontSize	63

9.3. Anpassung zur Erhöhung des Kontrastwerts	67
9.4. adaptive Methode der Google Map	71

Abbildungsverzeichnis

4.1. Einfluss Leserichtung auf Anordnung der Elemente	21
6.1. Ansicht example-component	37
8.1. Ansicht awc-input	53
9.1. Iron Data Table	55
9.2. Adaptiver Iron Data Table	58
9.3. Paper Date Picker	59
9.4. Adaptiver Paper Date Picker	64
9.5. Google Map Web Component	65
9.6. Adaptive Google Map	72

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

Kurzfassung	3
Listing Verzeichnis	5
Abbildungsverzeichnis	7
Abkürzungsverzeichnis	8
1. Einleitung	13
2. Methodik dieser Arbeit	14
3. Adaptivität	15
3.1. Nutzungskontext	16
4. Adaptivität an den Nutzer	17
4.1. Präferenzen der Nutzer	17
4.2. WCAG Richtlinien	18
4.3. Folgerung für Präferenzen	19
4.3.1. Kontrast	19
4.3.2. Schrift	19
4.3.3. Leserichtung	20
4.3.4. Anordnung der Elemente	20
4.3.5. Text-Alternativen	21
4.3.6. Geschwindigkeit	21
5. Preference Sets	22
5.1. Speichern der Nutzer Präferenzen	22
5.2. Herunterladen der Nutzer Präferenzen	23
5.2.1. Was bedeutet REST	23
5.2.2. Der Client	24

6. Web Components	27
6.1. Geschichte der Web Components	27
6.1.1. Custom Elements	27
6.1.2. HTML Imports	28
6.1.3. Decorators	28
6.1.4. Templates	28
6.1.5. Shadow DOM	29
6.2. Technik der Web Components	29
6.2.1. Custom Elements	30
6.2.2. HTML Imports	31
6.2.3. Templates	32
6.2.4. Shadow DOM	32
6.2.4.1. Slots	33
6.2.4.2. Shadow DOM Gestaltungsanweisungen	35
6.3. Ein vollständiges Web Component	36
6.4. Webentwicklung mithilfe der Web Components	37
6.4.1. Native Browser-Unterstützung	38
6.4.2. Kapselung	39
6.4.3. Wiederverwendung	39
6.4.4. Wartbarkeit	40
6.4.5. Browser-Unterstützung	40
6.5. Adaptivität der Web Components?	40
7. Polymer	42
7.1. Polymer Technologie	42
7.1.1. Ein Element registrieren	42
7.1.2. Laufzeit der Elemente	43
7.1.3. Eigenschaften der Elemente	43
7.1.4. Methoden der Elemente	44
7.1.5. Polymer DOM	45
8. Das AWC Framework	47
8.1. Darwin.js Technologie	47
8.1.1. Registrierung eines Elements	47
8.1.2. Eigenschaften der Elemente	48
8.1.3. Methoden der Elemente	48
8.1.4. Bilden eines Elements	49
8.1.5. Adaptivität der Elemente	50

8.2. Beispiel eines AWC Web Component	52
9. Adaptivität von bestehenden Web Components	54
9.1. Iron Data Table	54
9.1.1. Konzeption zur Adaptivität	54
9.1.2. Einrichtung des Iron Data Table	55
9.1.3. Umsetzung der Programmierung	56
9.1.3.1. Kontrast	56
9.1.3.2. Text-Alternativen	56
9.1.3.3. Schriftart	57
9.1.3.4. Schriftgröße	57
9.1.3.5. Leserichtung	57
9.1.4. Beispiel-Ergebnis der Adaptivität	58
9.2. Paper Date Picker	59
9.2.1. Konzeption zur Adaptivität	59
9.2.2. Einrichtung des Paper Date Picker Web Component	60
9.2.3. Umsetzung der Programmierung	60
9.2.3.1. Kontrast	61
9.2.3.2. Text-Alternativen	61
9.2.3.3. Schriftart	62
9.2.3.4. Schriftgröße	62
9.2.3.5. Leserichtung	63
9.2.4. Beispiel-Ergebnis der Adaptivität	64
9.3. Google Map	65
9.3.1. Konzeption zur Adaptivität	65
9.3.2. Einrichtung des Google Map Web Component	66
9.3.3. Umsetzung der Programmierung	66
9.3.3.1. Kontrast	66
9.3.3.2. Text-Alternativen	68
9.3.3.3. Schriftgröße	69
9.3.3.4. Schriftart	69
9.3.3.5. Leserichtung	70
9.3.3.6. Die adaptive Methode	70
9.3.4. Beispiel-Ergebnis der Adaptivität	72
10. Evaluation der Adaptivitäts-Anpassungen	73
10.1. Iron Data Table	73
10.2. Paper Date Picker	74

10.3. Google Map	74
11. Adaptive Web Components sind möglich	76
Literatur	78
Anhang	85

1. Einleitung

Im Folgenden wird der Aufbau dieser Arbeit vorgestellt. Im ersten Kapitel wird der Aufbau der Abhandlung beschrieben. Daraufhin wird im zweiten Kapitel die genutzte Methodik dieser Arbeit beschrieben. Im dritten Kapitel wird auf die Adaptivität eingegangen. Hier wird dieser Begriff und dessen Umsetzung erläutert. Danach wird der Nutzungskontext im Zusammenhang der Adaptivität beschrieben. Im vierten Kapitel wird die Adaptivität einer Internetseite an einen Nutzer konzipiert. Hier werden die *Web Content Accessibility Guidelines 2.0* (WCAG 2.0) im Zusammenhang mit Präferenzen von Nutzern besprochen und daraus Folgerungen für die Präferenzwerte zur Adaptivität erstellt. Das fünfte Kapitel erörtert die Zusammenstellung dieser Präferenzen. Es wird referiert wie diese aufgebaut sind, gespeichert und heruntergeladen werden können, um dann genutzt zu werden. Das sechste Kapitel behandelt die *Web Components*. Es wird auf die Geschichte, die Technik und die Webentwicklung mithilfe dieser Elemente eingegangen. Das siebte Kapitel berichtet über die „Polymer“ Bibliothek und deren Technik. Das achte Kapitel behandelt das *AWC Framework* und dessen Technologie, insbesondere wie die Adaptivität ermöglicht wird. Im neunten Kapitel werden bestehende Web Components zur Ermöglichung der Adaptivität erweitert. Zu jedem Element wird jeweils die Adaptivität konzipiert und im Programmcode umgesetzt. Das zehnte Kapitel evaluiert die Möglichkeit der Anpassung zur Adaptivität der vorhergehenden Elemente. Das elfte Kapitel beschreibt ein Fazit auf Basis der vorhergehenden Kapitel.

2. Methodik dieser Arbeit

Im Folgenden wird die Methodik dieser Abhandlung beschrieben. Sie lässt sich in zwei Bereiche aufteilen, „Adaptivität“ und „Web Components“. Beginnend mit dem Bereich „Adaptivität“ wird zuerst vorhandene Literatur gesammelt. Diese besteht zum Großteil aus Abhandlungen, Webseiten, Büchern und insbesondere auch Richtlinien zur barrierefreien Gestaltung. Aus dieser Literatur werden Ergebnisse zusammengetragen und mithilfe von persönlicher Interpretation verdichtet. Dadurch werden Präferenzen zur möglichen Adaptivität von Elementen erstellt und evaluiert.

Der Bereich „Web Components“ setzt sich aus einem Theorie- und einem Praxisteil zusammen. Im Theorie-Teil wird wiederum vorhandene Literatur gesammelt. Diese besteht aus Abhandlungen und zum großen Teil aus Programmdokumentationen und Anleitungen. Diese wird als Basis genutzt, die Technologien der *Web Components* und mögliche nutzbare *Frameworks* zu erforschen. Mit diesen Technologien und *Frameworks* werden verschiedene Tests durchgeführt und Beispiel-Komponenten erstellt, um die Technologien zu erschließen. Die Ergebnisse, eine Kombination aus der Literatur und der Arbeit am Programmcode, werden daraufhin gesammelt. Der Praxisteil beginnt mit der Sammlung und dem Testen von vorhandenen *Web Components*. Von diesen werden einige ausgewählt um diese zu erweitern. Anhand einer Kombination des Theorie-Teils des Bereichs „Web Components“, also der Arbeit mit den Technologien, und den aus dem Bereich der „Adaptivität“ gezogenen Schlüsse, werden Folgerungen gezogen wie eine Adaptivität konzipiert und insbesondere wie sie im Programmcode umgesetzt werden kann. Dadurch entstehen bei Gelingen adaptive *Web Components*. Daraufhin werden die Komponenten und der Prozess der Erstellung dieser evaluiert.

3. 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, Erfahrungsgraden 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, also ohne aktive Einwirkung des Nutzers, geschehen. So gleicht sich zum Beispiel 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.

3.1. 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.

4. Adaptivität an den Nutzer

Nachdem evaluiert wurde an welche Merkmale sich eine Adaptivität anpassen kann wird folgend die Anpassung selbst erläutert. 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 immer und von überall erreichbar sein. Als Nächstes müssen die Elemente der Internetseite darauf vorbereitet werden sich anzupassen. Dafür werden Funktionen und Ressourcen bereitgestellt, die bei Bedarf ausgeführt werden oder sich 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 alle adaptiven Elemente der Internetseite erteilt. Dieser beauftragt, die Funktionen auszuführen, die zu den jeweiligen Präferenzen und somit Nutzern passen. 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.

4.1. 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*).

4.2. WCAG Richtlinien

Um mehr über mögliche Präferenzen in Erfahrung zu bringen werden im folgenden die 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 sich Anwendungen adaptiv an verschiedene Nutzer anpassen sollten. Im Folgenden werden die Richtlinien aufgezählt und kurz beschrieben [6].

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 Zeit-basierte 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.

4.3. Folgerung für Präferenzen

Auf Basis der vorhergehenden Kapiteln, insbesondere den WCAG 2.0 Richtlinien und weiterer Beobachtungen wird nun erforscht, welche Merkmale eines *Web Component* sich in welcher Art und Weise anpassen sollen, um eine sinnvolle Adaptivität zu erreichen.

4.3.1. 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, der 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 [7, S.234].

4.3.2. Schrift

Wenn davon ausgegangen wird, dass eine als allgemein lesbar bekannte Schriftart verwendet wird, hat die Wahl einer serifen oder serifenlosen Schriftart keinen bestätigten Einfluss auf die Richtlinie 3.1, die Lesbarkeit eines Textes [8, S.157-161]. Jedoch kann, um eine Adaptivität der Schriftart zu ermöglichen, dem Nutzer die Wahl gelassen werden, ob er eine serife oder serifenlose 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 [8, S.152]. Somit ermöglicht ein Bezeichner Schriftgröße, zu welchem der Nutzer einen Wert wählen kann, eine nützliche Adaptivität der Anwendung.

4.3.3. 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 [9, S.148].

4.3.4. Anordnung der Elemente

Kongruent zum Kapitel 4.3.3 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 [10, S.47 f.]. Ein Beispiel dazu bilden die deutsche und ägyptische Nachrichtenseite „Zeit Online“ und „Al Shaab“, dargestellt in Abbildung 4.1. 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 4.1.: Einfluss der Leserichtung auf die Anordnung der Elemente [11] [12]

4.3.5. 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. Dies betrifft bei nicht textbasierten Inhalten vorrangig Steuerungselemente wie zum Beispiel Pfeile in verschiedene Richtungen. Bei Textbasierten Inhalten ist es vom einzelnen Fall abhängig ob ein bestimmter Text durch ein eindeutiges Symbol ersetzt werden kann.

4.3.6. 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 verstehen und betätigen.

5. Preference Sets

Die in Kapitel 4.3 erarbeiteten Präferenzen betreffen die Nutzer der adaptiven Webseite. Jeder Nutzer hat zu verschiedenen Präferenzen verschiedene Werte. Deshalb ist es notwendig die gesamten Präferenzen jedes einzelnen Nutzers zu speichern um sie später bei Aufruf der Webseite zu nutzen. Diese Sammlung der Präferenzen wird im weiteren Zusammenhang der Präferenzen beziehungsweise *Preference Set* genannt.

5.1. 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. Exemplarisch werden diese über die Versionsverwaltung *Git* auf der Internetseite „GitHub“ gespeichert. Dafür wurde ein *Repository*, also eine Art neues Projekt auf „GitHub“, mit Namen „data“ erstellt. In diesem werden für einzelne Nutzer und damit deren Zusammenstellung der Präferenzen einzelne Dateien gespeichert. Diese können dann über eine bestimmte Adresse aufgerufen werden und sind so später beim gewünschten Herunterladen der Präferenzen ansteuerbar. In Listing 5.1 ist ein Beispiel dieser Zusammenstellung dargestellt. Sie kann über die Adresse

```
1  [  
2    {  
3      "more-contrast": "true",  
4      "font-size": "xl",  
5      "reading-direction": "rtl"  
6    }  
7  ]
```

Listing 5.1: Preference Set eines Nutzers

„<https://raw.githubusercontent.com/christophkleber/data/master/1>“ aufgerufen werden. Hier ist zu beachten, dass der einzelne Nutzer durch eine Identifikationsnummer repräsentiert wird, hier die Nummer „1“. Im späteren realen Gebrauch würde jeder Nutzer eine eindeutige, unter Umständen längere, Identifikationsnummer erhalten. Dies ermöglicht, dass der Nutzer selbst anonym bleibt und dessen Präferenzen nur anhand dieser Nummern abgerufen werden können. Zu jeder vom Nutzer genutzten Präferenz, also dem Bezeichner, wird ein Wert zugeordnet. Beispielsweise ist in Zeile drei zur Präferenz „more-contrast“ der Wert „true“ gesetzt.

5.2. 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.

5.2.1. Was bedeutet REST

Um einen geeigneten *REST-Client* zu erstellen muss genauer auf das *REST* Protokoll eingegangen werden. *Representational State Transfer* (REST) ist ein Protokoll um Daten in einer verteilten Umgebung auszutauschen. Es basiert auf den folgenden Prinzipien [13, 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 *Hypertext Markup Language* (HTML) Protokoll. Es werden die von der *Hypertext Transfer Protocol* (HTTP)-Technologie bekannten Methoden verwendet. Dies macht das 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.

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

5.2.2. Der Client

Die Funktion um die Präferenzen eines bestimmten Nutzers abzufragen basiert auf dem „XMLHttpRequest“ JavaScript Objekt. Es bietet die Möglichkeit eine Anfrage an einen Speicherort, basierend auf dem HTTP Protokoll, zu stellen um Daten abzufragen. Die in Listing 5.2 dargestellte Funktion „loadPref“ führt eine Anfrage an einen Speicherort aus und speichert die danach erhaltenen Daten. Diese erwartet den Parameter „prefSet“, der die Identifikationsnummer des jeweiligen Nutzers darstellt. Zu Beginn wird die leere Variable „xhrreq“ erstellt. Daraufhin wird versucht diese in Zeile fünf als ein „XMLHttpRequest“ zu erstellen. Falls hier ein Fehler auftritt, deutet dies darauf hin, dass es sich bei dem verwendeten Browser nicht um einen modernen Browser handelt und deshalb das „XMLHttpRequest“ nicht verwendet werden kann. Daraufhin wird in Zeile zehn versucht „xhrreq“ als ein „ActiveXObject“ der Art „Msxml2.XMLHTTP“ zu erstellen. Tritt hier wiederum ein Fehler auf, handelt es sich wahrscheinlich um eine Version des „Internet Explorer“ mit einer Versionsnummer kleiner als fünf. Dann wird in Zeile 13 versucht „xhrreq“ als ein „ActiveXObject“ der Art „Microsoft.XMLHTTP“ zu erstellen. Kann auch dies nicht ohne Fehler funktionieren wird das Laden der Präferenzen nicht vom Browser unterstützt und die Funktion abgebrochen.

```

1 function loadPref(prefSet) {
2     var xhttpreq = new XMLHttpRequest();
3     try{
4         //Moderner Browser
5         xhttpreq = new XMLHttpRequest();
6     }catch (e){
7         //Internet Explorer 5+
8         try{
9             xhttpreq = new ActiveXObject('Msxml2.XMLHTTP');
10        }catch (e) {
11            //Internet Explorer 5
12            try{
13                xhttpreq = new ActiveXObject('Microsoft.XMLHTTP');
14            }catch (e){
15                //Wird nicht unterstützt
16                console.log('XMLHTTP nicht unterstützt: keine preference
17                    ↪ sets');
18                return false;
19            }
20        }
21    }
22    xhttpreq.onreadystatechange = function() {
23        if (this.readyState == 4 && this.status == 200) {
24            var jsonObj = JSON.parse(xhttpreq.responseText);
25            //Aufruf einer Funktion um mit Präferenzen weiter zu
26            ↪ arbeiten
27        }
28    };
29    xhttpreq.open('GET', 'Speicherort/' + prefSet, true);
30    xhttpreq.send();
31 }

```

Listing 5.2: JavaScript Programmcode zum Abfragen der Nutzerpräferenzen

Diese Objektzuweisung versucht die Funktion unabhängig vom genutzten Browser nutzbar zu machen, obwohl die *Web Components* allgemein nur auf modernen Browsern unterstützt werden. Jedoch können auch *Web Components* unter bestimmten Voraussetzungen und mit der Nutzung von *Polyfills* auch auf älteren Browsern genutzt werden. Mehr dazu in Kapitel 6.4.5. Unabhängig vom letztendlich zugewiesenen Objekt an die Variable „xhttpreq“, außer die Funktion wurde abgebrochen, wird die Funktion „onreadystatechange“ aufgerufen. Sie wartet auf einen Statuswechsel und prüft ob der „readyState“ im Wert „4“ vorliegt und

gleichzeitig der Status im Wert „200“. Dies bedeutet, dass der Status der Anfrage „done“ ist und die HTTP Anfrage auch erfolgreich war und somit die Daten abgerufen werden konnten. Daraufhin werden die übertragenen Daten in der Variable „jsonObj“ gespeichert und können weiter verwendet werden. Um die asynchrone Anfrage letztendlich abzuschicken und in den auf Antwort wartenden Zustand zu wechseln, werden in den letzten zwei Zeilen die Funktionen „open“ und „send“ aufgerufen. „open“ enthält als Argumente die Art des Aufrufs, hier „get“ und als letztes Argument „true“, damit der Aufruf asynchron durchgeführt wird. Ein weiteres Argument ist die *Uniform Resource Locator* (URL). Sie wird in diesem Listing stellvertretend als „Speicherort“ bezeichnet. Dieses Argument wird gesetzt um einen entsprechenden Speicherort der Zusammenstellung der Präferenzen anzufragen. Zu diesem „Speicherort“ wird der Parameter „prefSet“ angefügt. Dieser beschreibt dann die Auswahl der Präferenzen des einzelnen Nutzers. Ein Beispiel dieses Speicherorts könnte „http://www.speicherort.de/praeferenzen/1“ sein [14].

6. Web Components

Im Folgenden wird auf den Themenbereich der späteren Anwendung der Adaptivität eingegangen, den *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 [15, S.1]. Sie setzen sich zusammen aus den vier Technologien *Custom Elements*, *HTML Imports*, *Templates* und *Shadow Document Object Model* (DOM). Das Nutzen dieser Technologie soll Applikationen im Web leichter wiederverwertbar, wartbar, unabhängiger und kapselbar machen [15, S.2].

6.1. 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 [16]. 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 Browser-Kompatibilität eingegangen.

6.1.1. Custom Elements

Eine Technologie der *Web Components*, die *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 [17]. 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 [18]. Die Version v0 wird von der Version v1 abgelöst, hier ergeben sich einige Änderungen in der Syntax der *Advanced Programming Interface* (API) [19]. Derzeit wird sie nicht per Standardeinstellung von anderen Browsern unterstützt [20], 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 [21].

6.1.2. 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“ [22]. 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 [23].

6.1.3. 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“ [16]. 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“ [24]. Auf einer aktuellen Übersichtsseite des Konsortiums wird die *Decorators* Technologie nicht mehr im Zusammenhang mit *Web Components* erwähnt [25].

6.1.4. Templates

Templates werden schon über einen längeren Zeitraum in den verschiedenen Browsern unterstützt. Zuallererst wurden sie im „Chrome“ im Jahr 2013 verwen-

det, 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 [26] [27]. Im Jahr 2015 wurden sie dann vom „Edge“ Browser unterstützt, in der Version 13 [28]. Auf den Browsern des „Macintosh“ Betriebssystems wurden *Templates* das erste Mal 2014 verwendet, in der „Safari“ Version 7.1 und der „Safari & Chrome for iOS“ Version 8 [23]. 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 [23].

6.1.5. 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 *Shadow DOM* seit 2016 genutzt werden [29]. 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 [30]. Die Version v0 wird von der Version v1 abgelöst, die 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 [31]. 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 [32].

6.2. Technik der Web Components

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

6.2.1. Custom Elements

Das Bilden voll funktionstüchtiger DOM Elemente ermöglicht die *Custom Element* API [33]. Eine 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 [34]. Somit ermöglicht eine API einen Austausch von Informationen zwischen verschiedenen Programmen oder Systemen. Die *Custom Element* API ermöglicht den Nutzern die Auszeichnungssprache HTML zu erweitern [35]. 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 erstellten Programmiercodes in Elemente. In Listing 6.1 ist JavaScript Programmcode dargestellt, der ein leeres *Custom Element* definiert und unter dem Namen „new-custom-element“ registriert wird.

```

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

```

Listing 6.1: JavaScript Programmcode zum Erstellen eines Custom Elements

Hier wird das Standard „HTMLElement“ erweitert, es wird kein spezielles Element erweitert wie beispielsweise ein „Button“. Für *Custom Elements* sind mehrere *Callbacks* verfügbar. *Callback* Funktionen beschreiben hier Funktionen, die bei bestimmten Ereignissen des Lebenskreislafs der Applikation von außerhalb des *Custom Elements* aufgerufen werden. Im Folgenden werden diese Funktionen aufgelistet [35].

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

6.2.2. 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 URL des *Imports* und die Eigenschaft „rel='import'“ besitzt, also ein Link eines bestimmten Typ ist [36](siehe Listing 6.2). Die importierten Dokumente haben keinen außergewöhnlichen Aufbau im

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

Listing 6.2: HTML Programmcode zum Standardmäßigen 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 6.2.3. Um auf den Inhalt des importierten Dokuments zuzugreifen wird JavaScript verwendet. Wie in Listing 6.3 dargestellt, wird zuerst nach dem Link Element gesucht, das 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]'); //durchsucht
  ↳ nach zu importierendem Link
2 var importedDocument = link.import; //importiert den Inhalt des
  ↳ Links
3 var elem = importedDocument.querySelector('div.element');
  ↳ //Selektiert Element der importierten Komponente
```

Listing 6.3: JavaScript Programmcode für Zugriff auf Inhalt des importierten Dokuments

6.2.3. Templates

Die *Hypertext Markup Language Version 5* (HTML5) Funktion *Template* ermöglicht Teile einer Seite unabhängig vom DOM zu erstellen. Diese können dann später programmatisch zum DOM hinzugefügt werden [37, 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 [38]. In Listing 6.4 sieht man den JavaScript

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

Listing 6.4: JavaScript Programmcode für das Hinzufügen eines Templates in das DOM

Programmcode 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 ebenso die Bilder des *Templates* geladen und eventuelle Skripte ausgeführt.

6.2.4. Shadow DOM

„Das Shadow DOM beschreibt die Fähigkeit eines Browsers, eine neue, völlig abgekapselte Knotenstruktur im bestehenden DOM zu erzeugen“ [39, 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 des *shadow roots* 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 [40]. In Listing 6.5 ist dargestellt, wie mithilfe von JavaScript ein *Shadow*

```

1 var header = document.createElement('header');
2 var shadowRoot = header.attachShadow({mode: 'open'}); //shadow root
   ↳ an header angefügt
3 var headline = document.createElement('h1');
4 var headlineText = document.createTextNode('headline');
5 headline.appendChild(headlineText);
6 shadowRoot.appendChild(headline); //Elemente als Kinder in shadow
   ↳ root angefügt

```

Listing 6.5: JavaScript Programmcode für die Erstellung eines Shadow DOM

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.

6.2.4.1. Slots

Shadow DOM kann auch mit anderen DOM Elementen erweitert werden. Der Entwickler kann dem Nutzer seines *Web Component* ermöglichen, diesen zu erweitern. Hierfür werden *Slots* verwendet. In Listing 6.6 ist der HTML Programmco-

```

1 <ul id='contacts'>
2   <li>
3     <slot name='title'>Kein Titel</slot> <!-- Slot Element title
   ↳ mit Platzhalter 'Kein Titel' -->
4     <slot name='name'> <!-- Slot Element mit Platzhalter -->
5       <p>Achtung</p>
6       <h1>Kein Name</h1>
7     </slot>
8   </li>
9 </ul>

```

Listing 6.6: HTML Programmcode zur Nutzung von Slot Platzhalter-Elementen im Shadow DOM

de des *Web Component* dargestellt, der im *Shadow DOM* später ausgeführt 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 [40]. In Listing 6.7 werden die, in Listing 6.6 erstellten Platzhalter-Elemente, beim Verwenden des *Web Component* in beispielsweise einer Webseite befüllt.

```
1 <span slot='title'>Dr.</span> <!-- Element mit Slot Attribut -->
2 <span slot='title'>Phil.</span> <!-- Element mit Slot Attribut -->
3 <span slot='name'>Michael</span> <!-- Element mit Slot Attribut -->
```

Listing 6.7: HTML Programmcode zum Befüllen der Slot Elemente im DOM

Wie hier in Zeile eins und zwei 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 6.8 ist der, von den zwei vorhergehenden Listings, kombinierte und berechnete Inhalt zu sehen. Dies ist der DOM, den man beispielsweise in einer Webseite sehen würde. Hier

```
1 <ul id='contacts'>
2   <li>
3     <slot name='title'> <!-- Übersetztes Slot Element title -->
4       <span>Dr.</span>
5       <span>Phil.</span>
6     </slot>
7     <slot name='name'> <!-- Übersetztes Slot Element name -->
8       <span>Michael</span>
9     </slot>
10  </li>
11 </ul>
```

Listing 6.8: Übersetzter DOM

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.

6.2.4.2. Shadow DOM Gestaltungsanweisungen

Wie auf den vorherigen Seiten erklärt, ist der *shadow tree* vom normalen DOM ab gekapselt. Dies hat zur Folge, dass CSS Gestaltungsanweisungen, definiert im eigenen *Component*, nur Einfluss auf den *shadow tree* des eigenen *Web Component* haben. Es können CSS Selektoren verwendet werden, die auch außerhalb des Element bestehen. Trotzdem haben diese nur Einfluss auf die Elemente innerhalb des *Web Component*.

Eine Besonderheit der Gestaltungsanweisungen 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, die 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 von außen nach eigenen Wünschen anzupassen. Darüber hinaus kann der „:host“ Selektor erweitert werden. So können über die Form „:host(<selector>)“ bestimmte Stil Eigenschaften gesetzt werden, die nur Anwendung finden wenn das *Web Component* bestimmte Eigenschaften erfüllt. In Listing 6.9 ist dargestellt in welcher Form Einfluss genommen werden kann.

```
1 <style>
2 :host(.active) { /* :host Selektor bei Vorhandensein der Klasse
   ↳ active */
3     color: green;
4 }
5 :host(.blue) { /* :host Selektor bei Vorhandensein der Klasse blue
   ↳ */
6     color: blue;
7 }
8 </style>
```

Listing 6.9: CSS Gestaltungsanweisungen: Der :host Selektor

So spricht der Selektor in Zeile zwei nur den *Web Component* an, der die Klasse „active“ besitzt. Der Selektor in Zeile fünf spricht nur den *Web Component* an, wenn dieser die Klasse „blue“ besitzt.

6.3. Ein vollständiges Web Component

Die in den vorherigen Kapiteln erwähnten Technologien werden nun kombiniert, um beispielhaft ein vollständiges *Web Component* zu erstellen. In Listing 6.10

```
1 <template>
2   <style>
3     p {color: red}
4   </style>
5   <p>Dies ist ein Web Component.</p>
6 </template>
7 <script>
8 class Example extends HTMLElement {
9   constructor() {
10     super();
11     var template =
12       ↪ document.currentScript.ownerDocument.querySelector('template');
13     var clone = document.importNode(template.content, true);
14     this.createShadowRoot().appendChild(clone);
15   }
16 }
17 customElements.define('example-component', Example);
18 </script>
```

Listing 6.10: Beispiel Web Component

ist der HTML Programmcode dieser Komponente dargestellt. Hier werden die vorher erwähnten Technologien *Custom Elements*, *Templates* und *Shadow DOM* genutzt. In Zeile eins wird ein *Template* erstellt. Der Inhalt dessen besteht aus den CSS Gestaltungsanweisungen und einem Paragraph mit dem Text „Dies ist ein Web Component.“. Ab Zeile sieben wird der JavaScript Programmcode definiert. Dieser erstellt in Zeile acht die Klasse „example“, die ein „HTMLElement“ erweitert. Diese besitzt einen Konstruktor. In diesem wird zuerst die „super“ Funktion des erweiterten Element aufgerufen. Danach wird das *Template* Element ausgewählt, dessen Inhalt kopiert und in den neu erstellen *shadow root* eingefügt. In der Zeile 16 wird dann die definierte Komponente zu den „customElements“ unter dem Namen „example-component“ hinzugefügt. Daraufhin kann diese, nachdem die hier erstellte HTML Datei importiert wurde, in HTML Dokumenten unter dem Bezeichner „example-component“ genutzt werden.

```
1 <head>
2   <link rel='import' href='comps/example-component.html'>
3 </head>
4 <body>
5   <p>Normaler Inhalt der Internetseite.</p>
6   <example-component></example-component>
7 </body>
```

Listing 6.11: Nutzung des example-component

In Listing 6.11 ist der gesamte „body“ und ein Ausschnitt des „head“ einer HTML Datei dargestellt. Diese importiert das in Listing 6.10 erstellte *Web Component* und nutzt es dann unter dem definierten Bezeichner „example-component“. Abbildung 6.1 zeigt dann die im Browser angezeigte Ansicht des in Listing 6.11 genutzten *Web Components*.

Normaler Inhalt der Internetseite.

Dies ist ein Web Component.

Abbildung 6.1.: Ansicht des Beispiel Web Components Quelle: Bildschirmaufnahme des eigenen, vom Browser übersetzten, Programmcode

Hier wird der Text „Dies ist ein Web Component.“ als ein Paragraph angezeigt. Nur dieser Paragraph, also das eigentliche *Web Component* wird in rot dargestellt, da diese Gestaltungsanweisung innerhalb der Komponente gesetzt wurde und somit nur diese betrifft.

6.4. Webentwicklung mithilfe der Web Components

Aktuell werden *Web Components* in der Webentwicklung in geringer Häufigkeit genutzt. 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 define application interfaces, the interrelationships between applications, and internal communications between clients and external to online platforms“

[41, S.15]. 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 diese sich gegenseitig beeinflussen. So können die Gestaltungsanweisungen verschiedener Teile der Webseite sich unbeabsichtigt beeinträchtigen. Auch kann der JavaScript Programmcode, der eine bestimmte Funktion hat, an einer anderen nicht beabsichtigten, Stelle Einfluss nehmen. An dieser Stelle könnte er eine vollkommen andere Funktion ausüben und Inkonsistenzen hervorrufen. 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 möglicherweise 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.

6.4.1. Native Browser-Unterstützung

Aufgrund der Erkenntnisse der Geschichte von *Web Components* in Kapitel 6.1 ist eine gewisse Wahrscheinlichkeit gegeben, dass die Technologien der *Web Components* in Zukunft nativ von den verschiedenen Browsern unterstützt werden. Diese Konklusion basiert vor allem auf der Tatsache, dass verschiedene Techniken der *Web Components* schon in manchen Browsern nativ 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 vom externen Programmcode. Außerdem ist die Syn-

tax 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.

6.4.2. 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 [15, S.2]. Dies hat einige Folgen für das Verhalten einer Applikation. Zuerst ist der *Shadow DOM* isoliert. Er kann nicht von außerhalb angesprochen werden. 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 Gestaltungsanweisungen 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 [40]. Atomare Bezeichner haben den großen Vorteil dass sie kurz und prägnant den eigentlichen Nutzen oder Funktion des Elements beschreiben. 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*.

6.4.3. 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 [15, S.2]. Dadurch ist man nicht an ein bestimmtes *Framework* gebunden und kann auch mit Elementen außerhalb dieses Ökosystems interagieren und Komponenten wiederverwenden. Zum anderen können die *Web Components* aufgrund ihrer Definitionsart an theoretisch allen Stellen einer Anwendung wiederverwendet werden, was eine Arbeitserleichterung und auch Verminderung des Programmcodes hervorruft.

6.4.4. Wartbarkeit

Die Wartbarkeit von Web-Applikationen wird erleichtert, da die *Web Components* in *Templates* organisiert sind [15, S.2]. Das sorgt dafür, dass der Programmcode einzelner Komponenten separat gespeichert und somit leichter wiedergefunden und geändert werden kann. Insbesondere bei großen Projekten führt dies zu einer großen Arbeitserleichterung.

6.4.5. Browser-Unterstützung

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 6.1 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, welche 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 [42, S.4]. Das „webcomponents.js“ ist ein Set von *Polyfills* und ermöglicht den Nutzern die Verwendung von *Web Components* in allen modernen Browsern [43].

6.5. Adaptivität der Web Components?

Die Technik und die Vor- und Nachteile von *Web Components* allgemein wurden in den vorhergehenden Kapiteln evaluiert. Daraufhin stellt sich nun die Frage, inwiefern die *Web Components* zur Adaptivität geeignet sind. Um eine Komponente adaptiv zu gestalten, muss es möglich sein auf den Aufbau, das Aussehen und unter Umständen den Inhalt dieser Einfluss nehmen zu können.

Den Aufbau und das Aussehen adaptiv anzupassen könnten die CSS Gestaltungsanweisungen innerhalb der *Web Components* ermöglichen. Hier könnten je nach Nutzer andere Gestaltungsanweisungen erfolgen, um die Komponente an den Nutzer anzupassen. Um Einfluss auf den Inhalt nehmen zu können, könnten Methoden innerhalb der Komponente definiert werden, die Inhalt, anhand den Präferenzen des Nutzers, hinzufügen, weglassen oder ändern. Durch den gekapselten Zustand könnten sich bei richtiger Anwendung nur die *Web Components*

an den Nutzer adaptiv anpassen und keine anderen Teile der Webseite ungewollt verändern. Dies würde auch ermöglichen jedes einzelne *Web Component* unabhängig zu verändern. Da unter Umständen jedes einzelne *Web Component* anderweitig angepasst werden müsste um die Präferenzen der Nutzer zu erfüllen ist dies eine wichtige Eigenschaft. Um jedoch diese Anpassung innerhalb der *Web Components* durchzuführen muss auch auf innerhalb der Komponenten zugegriffen werden können. Dazu können HTML-Attribute genutzt werden. Diese würden von „außen“ dem *Web Component* hinzugefügt werden und dann innerhalb dieses verwendet werden um bestimmte Anpassungen durchzuführen. Falls alle diese Techniken korrekt genutzt werden und sich nicht gegenseitig negativ beeinflussen, kann es möglich sein *Web Components* zur Adaptivität zu erweitern.

7. Polymer

Die „Polymer“ Bibliothek bietet eine Möglichkeit *Web Components* zu nutzen. Sie stellt eine Sammlung von Funktionen bereit um *custom elements* zu erstellen und bietet darüber hinaus verschiedene nützliche Funktionen an. Die mithilfe dieser Bibliothek erstellten Elemente sollen letztendlich wie normale DOM Bestandteile funktionieren [44].

7.1. Polymer Technologie

Die verwendeten Technologien der Bibliothek um *Web Components* zu erstellen ähneln den der *Web Components* oder es werden direkt die Technologien der *Web Components* genutzt. Im Folgenden wird die Technik der Version 1.0 erläutert.

7.1.1. 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 7.1: JavaScript Programmcode der 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 7.1 in Zeile zwei zu sehen, spezifi-

ziert 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 7.1 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');“ [45].

7.1.2. Laufzeit der Elemente

Um auf die Laufzeit und die Reaktion der Elemente auf deren Zustände im Lebenszyklus zu reagieren stellt „Polymer“ verschiedene *Callback* bereit. Diese werden vom Element selbst ausgeführt, wenn bestimmte Zustände vorliegen. Die „ready“ Funktion wird beispielsweise ausgeführt, wenn der gesamte lokale DOM des Elements initialisiert und die Eigenschaftswerte gesetzt wurden. Ein weiteres Beispiel ist die „attached“ Funktion, sie wird aufgerufen wenn das Element an das „document“ hinzugefügt wird. Sie wird in jedem Fall nicht ausgeführt, bevor die „ready“ Rückruffunktion ausgeführt wurde [45].

7.1.3. 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“ Schlüssel definiert beispielsweise den Typ des Attributs, in Listing 7.2 ist die Eigenschaft „user“ beispielsweise vom Typ „String“ und die Eigenschaft „isHappy“ vom Typ „Boolean“.

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

Listing 7.2: JavaScript Programmcode des Polymer properties Objekt

Als weiteres Beispiel definiert der „value“ Schlüssel den Standardwert der Eigenschaft, bei der Eigenschaft „count“ die in der Funktion erstellte leere Datengruppe [46].

7.1.4. Methoden der Elemente

Um den späteren Instanzen Methoden zur Verfügung zu stellen, können diese dem Prototyp des Elements hinzugefügt werden.

```
1 Polymer({
2   is: 'my-element',
3   speak: function(){ //Methode des Elements
4     console.log('speak');
5   }
6 });
7 //Nachdem das Element instanziiert ist, kann die Methode aufgerufen
  ↪ werden.
8 var elem = document.querySelector('my-element');
9 elem.speak();
```

Listing 7.3: JavaScript Programmcode der Polymer Instanz-Methoden

In Listing 7.3 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 [47].

7.1.5. 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 [48]. In Listing 7.4 ist der Programmcode zur

```
1 <dom-module id='my-element'> <!-- Wird zu lokalem DOM übersetzt -->
2   <template>I am my-element!</template> <!-- Template Inhalt des
   ↳ Elements -->
3   <script> // Skript des Elements
4     Polymer({
5       is: 'my-element'
6     });
7   </script>
8 </dom-module>
```

Listing 7.4: Polymer dom-module

Erstellung eines 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 Gestaltungsanweisungen 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 Gestaltungsanweisungen 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 [49]. Diese werden in der in Listing 7.5 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-

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

Listing 7.5: Benutzerdefinierte CSS Eigenschaften zur Polymer Web Component Erstellung

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.

8. Das AWC Framework

Das *Adaptive Web Components Framework* (AWC Framework) ist eine weitere Möglichkeit mit *Web Components* zu arbeiten. Es 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 [50]. Im Folgenden wird die verwendete Technik des AWC Framework auf Basis des auf „GitHub“ vorhandenen Programmcodes erläutert [51].

8.1. 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 tritt eine Fehlermeldung auf. Danach wird mit den übergebenen Parametern weitergearbeitet.

8.1.1. 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 „HTMLElement“, dieses kann auch expandiert werden wenn kein bestimmtes, schon bestehendes, Element erweitert werden soll. In Listing 8.1 ist der Programmcode dargestellt, der ein neues Element erstellt. Hier wird das Element

unter dem Bezeichner „awc-image“ und als Erweiterung des „HTMLElement“ registriert.

```
1 new Component('awc-image', HTMLElement,
2   // Modell des Elements:
3   {
4     'contrast': 'normal',
5     'size': 'normal',
6     'reading_direction': 'ltr'
7   },
8   // Verhalten des Elements:
9   {
10    onAdaptiveChange: function() {
11      console.log('onAdaptiveChange');
12      console.log(this);
13    }
14  });
```

Listing 8.1: JavaScript Programmcode zum Erstellen eines Web Components basierend auf Darwin.js

8.1.2. Eigenschaften der Elemente

Dem „_data“ Parameter werden die initialen Modelldaten 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 8.1 werden beispielsweise die Präferenzen „contrast“, „size“ und „reading_direction“ gesetzt.

8.1.3. 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 8.1 wird beispielsweise die Methode „onAdaptiveChange“ übergeben. Diese wird somit dem *Web Component* zugeordnet.

8.1.4. 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 8.2 dargestellt. Daraufhin wird der zu im-

```

1  this.root = this.attachShadow({
2    mode: 'open' //Offener shadow root
3  });

```

Listing 8.2: JavaScript Programmcode zum Erstellen des shadow roots

portierende *Link*, passend zum „_htmlTag“, gewählt. Von diesem *Link* wird das *Template* Element gespeichert. Daraufhin wird die Funktion „_buildHtml“ mit dem Parameter „data“ aufgerufen. Diese ist in Listing 8.3 dargestellt. Hier wird in Zei-

```

1  _buildHtml(data) {
2    this.root.innerHTML = ''; //leerer shadow root
3    let instance = this.template.content.cloneNode(true); //Kopieren
    ↪ des Inhalts
4    let div = document.createElement('div');
5    div.appendChild(instance); //Inhalt wird als Kind an das div
    ↪ angehängt
6    let handlebar = window.Handlebars.compile(div.innerHTML); //Inhalt
    ↪ des div wird mit Handlebars kompiliert
7    let buildHtml = handlebar(data); //Kompilierter Code wird mit data
    ↪ Werten ausgeführt
8    this.root.innerHTML = buildHtml; //shadow root wird befüllt
9  }

```

Listing 8.3: JavaScript Programmcode zum Bilden eines Web Components basierend auf Darwin.js

le eins der Inhalt 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 geschweiften Klammern gesetzt werden, beispielsweise wie: „{{Inhalt}}“. Diese funktionieren als Platzhalter und werden beim Kompilieren durch die gesetzten Werte

ersetzt [52]. 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.

8.1.5. 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, die zur Adaptivität benötigt werden, besitzt. Wenn dies zutrifft werden alle Elemente gesammelt, die 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 zu Beginn die Methode „_pipeCallback“ mit dem Parameter „onAdaptiveChange“ aus, dadurch werden etwaige *Callback* Funktionen des Elements mit dem Wert 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 Werten für die Eigenschaften der Elemente. Wird sie aufgerufen prüft sie zu Beginn auch über die Methode „_hasAdaptiveWebComps“, ob der Browser die Funktionalitäten, die 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 8.1.5 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 8.4 ist der HTML Inhalt des „awc-image“ dargestellt. In diesem

```

1 <template>
2    <!-- Element mit
    ↳ CSS Klassen und Handlerbar Platzhaltern -->
3 </template>

```

Listing 8.4: HTML Programmcode des Templates des awc-image Elements

besitzt 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 8.5 sind ein Ausschnitt der Dar-

```

1 .contrast_, .contrast_normal { /* Standardkontrastwert */
2   -webkit-filter: none;
3   filter: none;
4   background: #fff;
5 }
6 .contrast_high { /* Erhöhter Kontrastwert */
7   -webkit-filter: grayscale(100%) contrast(130%);
8   filter: grayscale(100%) contrast(130%);
9   background: #000;
10 }

```

Listing 8.5: Ausschnitt der CSS Gestaltungsanweisungen des Templates des awc-image

stellungseigenschaften 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.

8.2. Beispiel eines AWC Web Component

Mithilfe des AWC *Frameworks* wird im Folgenden ein *Web Component* erstellt. Es wird unter dem Namen „awc-input“ registriert und hat die Funktion einen Benutzer nach einer Frage zu einem von ihm zu wählenden Thema zu befragen. In

```

1 <div class="contrast_{{ contrast }} reading_direction_{{
  ↳ reading_direction }} size_{{ size }}">
2   <p>Hallo{{ name }}, bitte wähle dein Thema:</p> <input
  ↳ name="topic" type="text">
3   <p>Deine Frage ist:</p> <textarea name="question" cols="40"
  ↳ rows="5"></textarea>
4   <button type="button">Absenden</button>
5 </div>

```

Listing 8.6: HTML Programmcode des awc-input Element

Listing 8.6 ist ein Ausschnitt des Elements, der HTML Programmcode des Elements, dargestellt. Hier wird in Zeile zwei ein „input“ Feld erstellt, das den Nutzer nach dem gewünschten Thema fragt, in Zeile drei wird ein „textarea“ Feld erstellt, das den Nutzer nach seiner Frage bittet. Zuletzt wird in Zeile vier ein „button“ erstellt, der, bei Anbindung an das *Backend*, die Frage abschickt. In Listing 8.7 ist der JavaScript Programmcode des Elements dargestellt. Hier wird das Element

```

1 new Component('awc-input', HTMLElement,
2   // Modell:
3   {
4     'contrast': 'normal',
5     'reading_direction': 'ltr',
6     'size': 'normal'
7   },
8   // Verhalten:
9   {
10    onAdaptiveChange: function() {
11      console.log('onAdaptiveChange');
12      console.log(this);
13    }
14  });

```

Listing 8.7: JavaScript Programmcode des awc-input

unter dem Namen „awc-input“ registriert und seine standardmäßigen Werte der Präferenzen gesetzt. Die Präferenzen des Elements können dann über das AWC *Framework* verändert werden, dabei werden seine Klassen geändert. Daraufhin ändert das Element sein Aussehen aufgrund dieser Klassen, anhand von Gestaltungsanweisungen. In Abbildung 8.1 ist das Element dargestellt.

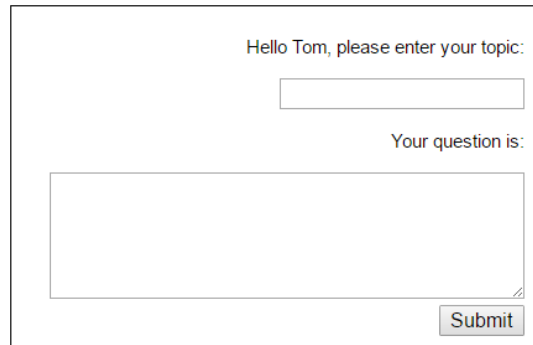
The image shows a web form within a rectangular border. At the top, the text "Hello Tom, please enter your topic:" is displayed. Below this is a single-line text input field. Further down, the text "Your question is:" is shown. Below this is a large, multi-line text area. In the bottom right corner of the form, there is a button labeled "Submit".

Abbildung 8.1.: Ansicht des awc-input Elements Quelle: Bildschirmaufnahme des eigenen, vom Browser übersetzten, Programmcode

Hier wird es von einem Nutzer aufgerufen, der eine normale Schriftgröße, einen normalen Kontrastwert und eine Leserichtung von rechts nach links in seinen Präferenzen festgelegt hat.

9. 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 [53]. In dieser Ausarbeitung wird nur deutschsprachiger Text beachtet, da eine korrekte Übersetzung sehr vieler möglicher Sprachen schwer durchführbar ist und für eine beispielhafte Konzeption und Programmierung von Adaptivität nicht notwendig ist.

9.1. Iron Data Table

Die erste Auswahl fiel auf das Element „iron-data-table“ [54]. Dieses *Web Component* stellt Tabellen dar. Die Tabelleninhalte können frei definiert werden.

9.1.1. Konzeption zur Adaptivität

In der Abbildung 9.1 ist das Element, befüllt mit Beispieldaten, dargestellt. Der Aufbau besteht aus der Kopfzeile im oberen Bereich mit den Bezeichnern der Spalten. Darunter ist der eigentliche Inhalt: Für jedes Datenelement eine Zeile mit den Werten des Elements zum Bezeichner der jeweiligen Spalte. Jede Zeile besitzt auf der linken Seite ein Quadrat, das angewählt werden kann und die Funktion besitzt, das Element der jeweiligen Zeile auszuwählen. Das Quadrat der Kopfzeile besitzt die Funktion alle Spalten anzuwählen.

<input type="checkbox"/>	Anrede	Vorname	Nachname	Alter
<input type="checkbox"/>	Frau	Donna	Davis	25
<input type="checkbox"/>	Herr	Samuel	Maier	43
<input type="checkbox"/>	Frau	Katja	Butler	67
<input type="checkbox"/>	Frau	Maria	Zeisig	19
<input type="checkbox"/>	Herr	Reinhard	Marx	48

Abbildung 9.1.: Iron Data Table Web Component Ansicht [54]

Das Element nutzt Symbole und Texte. Diese sollten sich adaptiv an den Nutzer anpassen. So sollte sich im Bezug auf die Texte Einfluss auf die Schriftgröße, die Schriftart und Text-Alternativen genommen werden. In diesem Element können keine Symbole als Text-Alternativen genutzt werden. Dies liegt daran, dass die Text-Elemente dieser Komponente sich in jeder Tabelle ändern können und somit nicht eindeutig und verlustfrei durch Bilder ersetzt werden können. Im Bezug auf die Symbole sollten Alternativen für diese bereitgestellt werden. Allgemein sollte sich der Kontrast des *Web Components* anpassen und der Aufbau an die Leserichtung des Nutzers anpassen lassen. Die Leserichtung hat einen Einfluss auf die Textelemente, hier werden jedoch nur einzeln stehende Wörter verwendet, die darüber hinaus sich, je nach Inhalten auch verändern. Deshalb lassen sich diese nicht adaptiv anpassen. Jedoch verändert die Leserichtung den Aufbau der Komponente, bisher spiegelt der Aufbau die Relevanz der Spalten wieder, von links nach rechts. Links ist das wichtigste Element, das Quadrat zum Anwählen einer Zeile. Bei einer Leserichtung von rechts nach links wird dieser Aufbau gespiegelt.

9.1.2. Einrichtung des Iron Data Table

Das *Iron Data Table Web Component* ist auf „GitHub“ gespeichert [54]. Hier wurde ein „fork“ durchgeführt um am Programmcode dieses Elements weiterarbeiten zu können. Das bedeutet, dass der Stand der eigentlichen Entwicklern kopiert wird und eigene Änderungen im Weiteren darauf aufbauen. Die Abhängigkeiten werden über „Bower“ installiert. „Bower“ ist ein Paketverwaltungsprogramm, welches das Aktualisieren und Installieren von Bibliotheken und *Frameworks* unter-

stützt. Dabei wird „Bower“ als ein Kommandozeilenprogramm genutzt [55]. Der „Iron Data Table“ selbst wird aus verschiedenen *Web Components* zusammengesetzt. In der hier verwendeten Zusammenstellung werden die Elemente „iron-data-table“, „iron-list“, „data-table-colum“, „data-table-row“, „data-table-checkbox“ und „data-table-cell“ genutzt.

9.1.3. Umsetzung der Programmierung

Im Folgenden wird die konzipierte Adaptivität der Komponenten im Programmcode umgesetzt.

9.1.3.1. Kontrast

Um den Kontrast des Elements anzupassen wird die Methode „_changeContrast“ des „iron-data-table“ Elements genutzt. Diese fragt den Wert des Parameters „moreContrast“ ab, bei einem „true“ Wert wird zu Beginn die Hintergrundfarbe des Hintergrunds auf weiß gesetzt, damit die Elemente sich besser von diesem abheben. Daraufhin werden allen Elementen mit den Bezeichnern „iron-data-table“, „data-table-row“ und „data-table-checkbox“ die CSS Klasse „more-contrast“ hinzugefügt. Diese werden dann von den einzelnen Komponenten über Gestaltungsanweisungen genutzt. Das umschließende Element „iron-data-table“ wird angewiesen die Schriftfarbe auf schwarz zu setzen um den Kontrast zwischen Text-Elementen und Hintergrund zu erhöhen. Die Komponente „data-table-row“ wird angewiesen einen Rahmen um die Kopfzeile der Tabelle zu erstellen, um eine bessere Abgrenzung dieser vom Rest der Tabelle zu ermöglichen. Auch wird die Farbe des Hintergrunds der Kopfzeile auf weiß gesetzt um den Kontrast zum Text zu erhöhen. Darüber hinaus wird der „Fokus“ Zustand der Zeilen verändert. Dieser wird, vorher dargestellt durch einen blauen Text auf grauem Hintergrund, nun dargestellt durch einen weißen Text auf schwarzem Hintergrund. Dabei wird die blaue Linie unter der Zeile entfernt. Die Komponente „data-table-checkbox“ wird dahingehend verändert, über Gestaltungsanweisungen das Anwählen eines Feldes über einen schwarzen Haken auf weißem Grund statt eines blauen Hakens auf weißem Grund darzustellen.

9.1.3.2. Text-Alternativen

In diesem Elemente können Texte wie in Kapitel 9.1.1 dargestellt, nicht durch Symbole ersetzt werden. Jedoch können die vorhandenen Symbole adaptiv durch

Texte ersetzt werden. Dies betrifft die Symbole für eine angewählte Zeile, ein Quadrat mit Haken in diesem und die Symbole für eine nicht ausgewählte Zeile, ein Quadrat. Hierfür wird die Methode „_changeTextAlternatives“ innerhalb des „iron-data-table“ Elements genutzt. Diese fragt zu Beginn den Wert der Eigenschaft „textAlternatives“ ab, ist dieser „text“, werden alle „data-table-checkbox“ Elemente gesammelt und jedem dieser die CSS Klasse „text“ hinzugefügt und der Inhalt dieses Element durch den Text „auswählen“ ersetzt. Darüber hinaus wird die Methode „_changeTextAlternatives“ innerhalb des „data-table-checkbox“ Element genutzt. Diese wird von der schon vorhandenen Methode „_checkedChanged“ aufgerufen. Danach sammelt sie alle „data-table-checkbox“ Elemente und ersetzt den Inhalt dieser durch den Text „gewählt“, wenn die sie die Klasse „text“ und das Attribut „checked“ enthalten oder durch den Text „auswählen“, wenn sie nur die Klasse „text“ enthalten und nicht das Attribut „checked“.

9.1.3.3. Schriftart

Die Schriftart sollte sich adaptiv an den Nutzer anpassen. Um dies zu erreichen wird die Methode „_changeFontStyle“ des „iron-data-table“ Element genutzt. Hier wird der Parameter „fontStyle“ abgefragt. Ist dieser auf den Wert „serif“ gesetzt, wird dem „iron-data-table“ Element die CSS Klasse „serif“ hinzugefügt. Diese wird dann in den Gestaltungsanweisungen genutzt um die Anweisung zur „font-family“ zu ändern.

9.1.3.4. Schriftgröße

Auch die Größe der Schrift sollte sich adaptiv an den Nutzer anpassen. Hierfür wird die Methode „_changeFontSize“ des „iron-data-table“ Elements genutzt. Sie fragt den Wert des Parameters „fontSize“ ab. Dessen Standardwert ist „m“. Ist der Wert auf „l“ oder „xl“ gesetzt wird die Schriftgröße um zwei oder vier Pixel erhöht. Dies wird ermöglicht durch das Hinzufügen von den CSS Klassen „l“ oder „xl“ zu dem „iron-data-table“ Element. Diese werden dann über Gestaltungsanweisungen des *Web Component* genutzt um die Anweisung zur „font-size“ zu ändern.

9.1.3.5. Leserichtung

An die Leserichtung des Nutzers wird sich über die Methode „_changeReadingDirection“ des „iron-data-table“ Elements angepasst. Sie fragt den Wert des Parameters „readingDirection“ ab. Ist dieser „rtl“, werden alle „data-table-row“ Ele-

mente gesammelt und jedem dieser die CSS Klasse „rtl“ hinzugefügt. Im den „data-table-row“ Elementen werden diese Klassen in den Gestaltungsanweisungen genutzt, den Wert der „flex-direction“ auf den Wert „row-reverse“ zu setzen um den Aufbau der Zeile zu verändern. Dadurch wird dieser vertikal gespiegelt und das wichtigste Element befindet sich bei einer Leserichtung von rechts nach links auf der rechten Seite.

9.1.4. Beispiel-Ergebnis der Adaptivität

In der Abbildung 9.2 ist das adaptive „Iron Data Table“ Element dargestellt. Die Abbildung basiert auf dem vorher ausgewählten und daraufhin zur Adaptivität erweiterten Element.

Alter	Nachname	Vorname	Anrede	<input type="checkbox"/>
25	Davis	Donna	Frau	<input type="checkbox"/>
43	Maier	Samuel	Herr	<input type="checkbox"/>
67	Butler	Katja	Frau	<input type="checkbox"/>
19	Zeisig	Maria	Frau	<input type="checkbox"/>
48	Marx	Reinhard	Herr	<input type="checkbox"/>

Abbildung 9.2.: Adaptives Iron Data Table Web Component Ansicht Quelle: Bildschirmaufnahme des eigenen, vom Browser übersetzten, Programmcode

Das Element wurde hier beispielhaft an einen Nutzer angepasst, dessen Präferenzwerte einen erhöhten Kontrast, eine erhöhte Schriftgröße und eine Leserichtung von rechts nach links enthalten.

9.2. Paper Date Picker

Als nächstes wurde das „paper-date-picker“ Element ausgewählt [56]. Es ermöglicht die Anzeige eines Kalenders. Von diesem Kalender kann ein einzelner Tag vom Nutzer ausgewählt werden, dieser wird dann mit erweiterten Informationen angezeigt.

9.2.1. Konzeption zur Adaptivität

Das *Web Component* ist in Abbildung 9.3 dargestellt. Es nutzt Texte und Symbole. Die Adaptivität dieses Elements sollte durch eine mögliche Veränderung des Kontrastes, der Schrift, wenn möglich der Bereitstellung von Text- und Symbol-Alternativen und der Leserichtung und dadurch wiederum der Anordnung der Elemente ermöglicht werden.

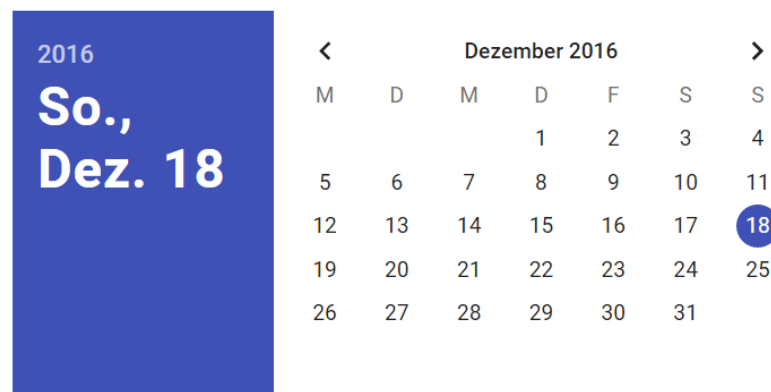


Abbildung 9.3.: Paper Date Picker Web Component Ansicht [57]

Der Kontrast der Texte, also der Datenelemente, gegenüber dem Hintergrund sollte sich an die Bedürfnisse der Nutzer anpassen. Diese sollten sich hochgradig voneinander unterscheiden lassen können. Es sollten, wenn möglich, Alternativen für Texte oder im Gegensatz dazu, zu Darstellungen, geboten werden. Um Text-Alternativen für Text-Elemente zu ermöglichen muss es möglich sein diese zu ersetzen ohne ein Verlust an Informationen zu erleiden. Die Text-Komponenten dieses Elements bestehen aus Bezeichnungen für Tage, Monate, Jahre und Zahlen. Es ist nicht möglich, diese durch eindeutige Symbole zu ersetzen. Deshalb können in diesem *Web Component* keine Text-Alternativen angeboten werden. Es werden jedoch auch Symbole in diesem *Web Component* genutzt. Um zwischen den Monaten des Datums-Wählers zu wechseln werden Vergleichszeichen genutzt. Um zu einem Monat vorher zu wechseln wird das Kleiner-als-Zeichen

und um zu einem Monat später zu wechseln wird das Größer-als-Zeichen genutzt. Jeweils bei einem Klick auf eines dieser Symbole wird dessen Funktionalität ausgeführt. Eine Alternative für diese Symbole kann durch verschiedene Sätze bereitgestellt werden. So kann das Symbol für den vorherigen Monat durch den Ausdruck „voriger Monat“ und für den nächsten Monat durch den Ausdruck „nächster Monat“ ersetzt werden. Auf die Schrift der Texte sollte Einfluss genommen werden können. So sollte die Schriftart und insbesondere die Schriftgröße anpassbar sein. Die Schriftart sollte zwischen einer serifen und einer serifenlosen Art wechseln, die Größe sollte sich zwischen der Standardgröße, einer großen und einer sehr großen Schriftgröße wechseln lassen. Hiervon sind Datenelemente und die Bezeichner des Kalenders betroffen. Auch der Text des derzeitig ausgewählten Datums sollte sich anpassen können. Um sich an die Leserichtung des Nutzers anzupassen, sollte Einfluss auf die Anordnung der Texte und auch Elemente genommen werden. Der Standardzustand der Komponente ist die Leserichtung von links nach rechts, hier wird das ausgewählte Datum auf der linken Seite und der Kalender auf der rechten Seite dargestellt. Auch werden die Wochentage des Kalenders von links nach rechts gelesen. Diese Elemente sollten sich bei einer Leserichtung von rechts nach links verschieben lassen.

9.2.2. Einrichtung des Paper Date Picker Web Component

Das *Paper Date Picker Web Component* ist auf „GitHub“ gespeichert [57]. Es wurde wiederum ein „fork“ durchgeführt um an diesem weiterarbeiten zu können und es zu erweitern. Für die Abhängigkeiten des Elements wurde wieder „Bower“ genutzt. Das „Paper Date Picker“ Element wird selbst aus verschiedenen *Web Components* zusammengesetzt. In der hier verwendeten Zusammenstellung werden insgesamt die Elemente „paper-date-picker“, „paper-calendar“, „paper-year-list“, „iron-media-query“, „neon-animated-pages“ und „neon-animatable“ genutzt. Nachdem es eingebunden ist und alle Abhängigkeiten installiert sind kann das *Web Component* unter dem HTML Bezeichner „<paper-date-picker>“ genutzt werden.

9.2.3. Umsetzung der Programmierung

Im Folgenden wird die konzipierte Adaptivität des Elements im Programmcode umgesetzt.

9.2.3.1. Kontrast

Um den Kontrast adaptiv zu erhöhen wird die Eigenschaft „moreContrast“ und die Methode „_changeMoreContrast“ des „paper-date-picker“ verwendet. Die Eigenschaft ist vom Typ Boolean und der Standardwert ist „false“. Die Methode fragt zu Beginn den Wert der Eigenschaft ab, ist dieser „true“ wird die Hintergrundfarbe der linken Hälfte des *Web Component* auf schwarz geändert, um den Kontrast zwischen der Farbe des Texts und der Farbe des Hintergrunds zu erhöhen. Das „paper-date-picker“ Element nutzt innerhalb seiner selbst noch das „paper-calendar“ Element. Wenn nun, um den Kontrast zu erhöhen, die Hintergrundfarbe des Rahmens des derzeitigen Datums geändert werden soll, ist dies nicht ohne weitere Anpassung möglich. Wenn die Methode „_changeMoreContrast“ in die „ready“ Methode des *Web Component* eingefügt wird, wird sie ausgeführt nachdem der lokale DOM des Elements initialisiert wurde, jedoch ist nicht garantiert, dass der lokale DOM des inneren Elements, des „paper-calendar“ schon vollständig initialisiert wurde. Deshalb wird das Ereignis mit dem Bezeichner „selectedDate“ vom „paper-calendar“ versendet, wenn sich das gewählte Datum des Kalenders ändert. Dies wird realisiert, indem das Ereignis von der Methode „_getDayClass“ versendet wird. Die Methode „_changeContrast“ des „paper-date-picker“ wartet auf dieses Ereignis und fügt daraufhin die CSS Klasse „moreContrast“ an das derzeit selektierte Datumselement. Die CSS Klasse wird im „paper-calendar“ genutzt um diesem Element eine schwarze Hintergrundfarbe hinzuzufügen. Die Methode „_changeMoreContrast“ des „paper-date-picker“ wird zur vorhandenen „ready“ Methode des „Polymer“ *Web Component* hinzugefügt um nach der Initialisierung des lokalen DOMs ausgeführt zu werden.

9.2.3.2. Text-Alternativen

Es ist möglich die Symbole dieses Elements durch Texte zu ersetzen. Um eine Adaptivität dieser Symbole und deren Alternativen zu erreichen wird die Eigenschaft „textAlternatives“ hinzugefügt. Diese ist vom Typ String. Die Methode „_changeTextAlternatives“ des „paper-date-picker“ fragt zu Beginn ab, ob die Eigenschaft „textAlternatives“ den Wert „text“ enthält, ist dies der Fall wird das Symbol für den vorherigen Monat durch den Text „voriger Monat“ und das Symbol für den nächsten Monat durch den Text „nächster Monat“ ersetzt. Die Methode „_changeTextAlternatives“ wird wiederum in die „ready“ Methode des *Web Components* eingefügt, damit sie nach Initialisierung des lokalen DOMs ausgeführt wird.

9.2.3.3. Schriftart

Um die Art der Schrift adaptiv zwischen einer serifen oder serifenlosen Schriftart zu wechseln wird die Methode „_changeFontStyle“ des „paper-date-picker“ genutzt. Im Standardzustand ist die Schriftart des *Web Component* eine serifenlose Schriftart. Die Methode fragt zu Beginn den Wert des Parameters „fontStyle“ ab. Ist dieser gesetzt und enthält den Wert „serif“ werden zum äußeren *Web Component*, dem „paper-date-picker“ und zu den inneren *Web Components*, den „paper-calendar“ und „paper-year-list“ Elementen die CSS Klasse „serif“ hinzugefügt. Diese Klassen werden in den Gestaltungseigenschaften der einzelnen Komponenten genutzt, um die Schriftart der Textelemente auf eine serife Schriftart zu wechseln. In Listing 9.1 sind ein Ausschnitt der Gestaltungsanweisungen des „paper-date-picker“ dargestellt. Sie weisen bestimmte Elemente des *Web Com-*

```
1 :host(.serif) span, :host(.serif) #heading.paper-date-picker
  ↳ .year.paper-date-picker{
2   font-family: 'Times New Roman', Times, serif';
3 }
```

Listing 9.1: Ausschnitt der CSS Gestaltungsanweisungen des Paper Date Picker für eine serife Schriftart

ponent, hier die Textelemente, an, eine serife Schriftart zu nutzen. Dies soll jedoch nur durchgeführt werden, wenn das *host* Element die Klasse „serif“ besitzt.

9.2.3.4. Schriftgröße

Die Schriftgröße sollte sich auch adaptiv anpassen. So sollten alle Textelemente sich je nach Nutzer normal groß, größer oder sehr groß darstellen lassen. Dies wird über die Methode „_changeFontSize“ des „paper-date-picker“ Elements erreicht. Diese fragt den Wert des Parametes „fontSize“ ab. Dieser kann die Werte „m“, „l“ oder „xl“ enthalten. „m“ steht für den Standardwert der Größe der Schrift des *Web Component*. Bei dem Wert „l“ werden alle Textelemente der Komponente um die Menge „2px“ vergrößert und „xl“ nochmals um diesen Wert vergrößert. Die Vergrößerung wird über CSS Klassen realisiert. Die Methode „_changeFontSize“ fügt je nach Wert des Parameters „fontSize“ die Klasse „m“, „l“ oder „xl“ zu den „paper-date-picker“ und „paper-calendar“ Elementen hinzu. In Listing 9.2 ist diese Methode dargestellt. Sie verwendet ein *Switch Statement* das, je nach Wert des „fontSize“ Parameters die CSS Klassen hinzufügt. Da der Wert „m“ den

```
1  _changeFontSize: function () {  
2      switch (this.fontSize){  
3          case 'm':  
4              return;  
5              break;  
6          case 'l':  
7              document.querySelector('paper-date-picker').className += ' l';  
8              document.querySelector('paper-calendar').className += ' l';  
9              break;  
10         case 'xl':  
11             document.querySelector('paper-date-picker').className += ' xl';  
12             document.querySelector('paper-calendar').className += ' xl';  
13             break;  
14         default:  
15             break;  
16     }  
17 }
```

Listing 9.2: Die Methode _changeFontSize

Standardwert darstellt wird hier keine Klasse hinzugefügt. Diese CSS Klassen werden von den Gestaltungsanweisungen der Komponenten „paper-date-picker“ und „paper-calendar“ dazu genutzt die Größe der Textelemente anzupassen. Darüber hinaus muss bei der Größe „xl“ der Abstand mancher Elemente zueinander vergrößert werden, um ein sich gegenseitiges Überdecken dieser zu verhindern.

9.2.3.5. Leserichtung

Die Standard Leserichtung der Komponente ist von links nach rechts. Um diese zu ändern wird die Methode „_changeReadingDirection“ des „paper-date-picker“ Elements verwendet. Sie fragt zu Beginn den Wert des Parameters „reading-Direction“ ab. Enthält dieser den Wert „rtl“, also eine erwünschte Leserichtung von rechts nach links, werden zum äußeren *Web Component*, dem „paper-date-picker“ und zu dem inneren *Web Component*, dem „paper-calendar“ die CSS Klasse „rtl“ hinzugefügt. In den Gestaltungsanweisungen dieser Komponenten werden die Elemente angewiesen, bei der Existenz der Klasse „rtl“ die Eigenschaft „flex-direction“ bestimmter Elemente auf den Wert „row-reverse“ zu setzen. Dadurch wird der Aufbau des *Web Component* für Nutzer ausgerichtet, die von

rechts nach links lesen.

9.2.4. Beispiel-Ergebnis der Adaptivität

In der Abbildung 9.4 ist das adaptive „Adaptiver Paper Date Picker“ Element dargestellt.

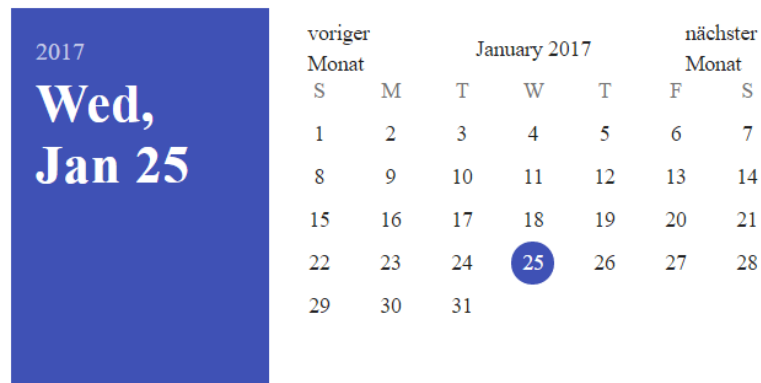


Abbildung 9.4.: Adaptiver Paper Date Picker Web Component Ansicht Quelle: Bildschirmaufnahme des eigenen, vom Browser übersetzten, Programmcode

Das Element wurde hier beispielhaft an einen Nutzer angepasst, dessen Präferenzwerte eine serife Schriftart und Text-Alternativen statt Symbole enthalten.

9.3. Google Map

Zuletzt wurde das Element „google-map“ ausgewählt. Es ist ein *Web Component*, das die Anzeige einer „Google Map“, also einer Karte, ermöglicht [58].

9.3.1. Konzeption zur Adaptivität

Abbildung 9.5 zeigt die Ansicht einer „Google Map“. Sie nutzt einerseits farbige Abbildungen und Symbole, andererseits auch verschiedene Texte.



Abbildung 9.5.: Google Map Web Component Ansicht [59]

Um eine Adaptivität zu ermöglichen, sollte Einfluss auf den Kontrast, Text- und Symbol-Alternativen, die Schrift und die Leserichtung und damit auf die Anordnung der Elemente genommen werden. Der Kontrast der Kartenelemente sollte sich adaptiv an den Nutzer anpassen. Hier sollten die Straßen, Gebäude, Grünflächen und ähnliche Elemente, die 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 Einstellungsleiste, sich bei einer Leserichtung von links nach rechts in der linken oberen Ecke befinden und bei einer Leserichtung von rechts nach links in der rechten oberen Ecke befinden. Dementsprechend würde 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, die eine Funktion erfüllen und nur durch Abbildungen oder Symbole dargestellt werden, auch als Texte angeboten werden.

9.3.2. Einrichtung des Google Map Web Component

Das *Google Map Web Component* ist auf „GitHub“ gespeichert [59]. Um die Möglichkeit zu erlangen es anzupassen wurde wiederum ein „fork“ durchgeführt. Um die Abhängigkeiten des *Web Component* zu installieren wurde „Bower“ genutzt. Die „Google Map“ selbst wird aus verschiedenen *Web Components* zusammengesetzt. In der hier verwendeten Zusammenstellung werden die Elemente „google-map“ und „google-maps-api“ genutzt. Falls „Marker“ gesetzt werden wird auch das Element „iron-selector“ genutzt. 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“ Benutzerkonto generiert und daraufhin in das *Web Component* eingebunden.

9.3.3. Umsetzung der Programmierung

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

9.3.3.1. 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 das Aussehen dieser Elemente oder Merkmale über verschiedene Gestaltungsoptionen angepasst werden [60].

```
1 gmapstyles =
2 [
3   {
4     "featureType": "administrative",
5     "elementType": "labels.text.fill",
6     "stylers": [{
7       "color": "#000000"
8     }]
9   },
10  {
11    "featureType": "administrative",
12    "elementType": "labels.text.stroke",
13    "stylers": [{
14      "color": "#ffff00"
15    }]
16  },
17  {
18    "featureType": "landscape",
19    "stylers": [{
20      "color": "#000000"
21    }]
22  }
23 ]
```

Listing 9.3: JavaScript Programmcode zur Gestaltung und Anpassung eines erhöhten Kontrastwerts

In Listing 9.3 ist ein Ausschnitt des Programmcodes der einen erhöhten Kontrast der Karte erreicht, dargestellt. Die hier erstellte Datengruppe wird als „styles“ Parameter beim Instanzieren 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 Boolean und der Standardwert ist „falsch“. Wird sie beim Einbinden des *Web Component* gesetzt, wechselt ihr Wert auf „wahr“. Die Methode „_getMapOptions“ ist in dem Element „google-map“ schon vor der Anpassung zur Adaptivität vorhanden. Diese erstellt eine Datengruppe „mapOptions“. Der wird die „styles“ Datengruppe übergeben. Um eine Adaptivität zu ermöglichen wird hier der Datengruppe der Rückgabe-Wert der Methode „_changeContrast“ des

„google-map“ Elements übergeben. Diese fragt 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.

9.3.3.2. Text-Alternativen

In einem adaptiven *Web Component* sollte es ermöglicht werden, Symbole oder Abbildungen durch Texte darzustellen oder Texte durch Symbole zu ersetzen, falls dies möglich ist. In diesem *Web Component* wird dafür die Methode „_changeTextAlternatives“ des „google-map“ Elements verwendet. Diese wartet zu Beginn auf die Benachrichtigung des Ereignisses „idle“ der „Google Map“. Diese bedeutet, dass alle Elemente der Karte nach einer Aktion wieder inaktiv sind. Dieses Ereignis wurde gewählt, da vor diesem Ereignis von der „Google Map“ oft Änderungen zur Laufzeit entstehen und auf diese reagiert werden müssen. Danach fragt 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 in nächstem Abschnitt 9.3.3.3, 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 überdeckt 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 [61]. 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“, eine gefaltete Karte, genutzt und für die Einstellung Satellit das Symbol „fa-globe“, ein Globus. Diese werden dann bei Aufruf der Methode „_changeTextAlternatives“ des „google-map“ Elements und einem mit „Symbol“ versehenen Wert des „textAlternatives“ Parameter gesetzt. Die Methode wird in die „_initGMap“ Methode hinzugefügt, hiermit wird erreicht, dass die Methode zur Adaptivität vom *Web Component* selbst ausgeführt wird, denn die Methode „_initGMap“ wird in der „attached“ *Callback* Funktion der „Polymer“ Bibliothek aufgerufen, wenn das Element an das *document* angefügt wurde. Siehe mehr dazu in Kapitel 7.1.2.

9.3.3.3. Schriftgröße

Um die Adaptivität der Schriftgröße zu ermöglichen wird die Methode „_changeFontSize“ genutzt. Sie wartet auch auf die Benachrichtigung des Ereignisses „idle“ der „Google Map“. 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 sich gegenseitiges Überdecken dieser zu verhindern. Die Methode „_changeFontSize“ wird auch in die „_initGMap“ Methode eingefügt, um zu erreichen dass sie ausgeführt wird, wenn das Element an das *document* angefügt wurde.

9.3.3.4. Schriftart

Die Schriftart kann sich auch an den Nutzer anpassen. Dies ermöglicht die Methode „_changeFontStyle“ des „google-map“ Elements. Diese wartet auch auf die Benachrichtigung des Ereignisses „idle“ der „Google Map“. Daraufhin wird abgefragt ob der Parameter „fontStyle“ 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. Eine Änderung findet nur statt wenn der Wert auf „serif“ gesetzt ist, da im Standardzustand des *Web Component* eine serifenlose Schriftart genutzt wird. Die Methode wird auch zur „_initGMap“ Methode hinzugefügt.

9.3.3.5. 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“ des „google-map“ Elements wartet auf die Benachrichtigung des Ereignisses „idle“ 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 Verkleinerungs-Ereignis die Gestaltungseigenschaften der Elemente und damit ihre Anordnung im *Web Component* neu berechnet, müssen die Auswirkungen der Leserichtung nach jedem Vergrößerungs- oder Verkleinerungs-Ereignis neu gesetzt werden. Deshalb 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. Auch die Methode „_changeReadingDirection“ wird zur Methode „_initGMap“ hinzugefügt.

9.3.3.6. Die adaptive Methode

Eine notwendige Besonderheit des „google-map“ Elements ist die Abweichung der „adaptive“ Methode. Jedes der zur Adaptivität erweiterten Elemente besitzt zwar diese Methode, jedoch besitzt sie in den anderen Elemente keine erweiterte Funktion, sondern zählt nur die zur Adaptivität benötigten Methoden auf und führt diese aus. In Listing 9.4 ist der Programmcode dieser Methode dargestellt. In Zeile fünf bis acht werden, wie in den anderen erweiterten *Web Components*, die Methoden zur Adaptivität aufgerufen. Die Abweichung dieser Methode beginnt

jedoch schon in Zeile zwei. Hier wird der Kontrast-Wert gesetzt. Es wird abgefragt ob das „map“ schon existiert.

```
1 adaptive: function(){
2     if (this.map) {
3         this.map.setOptions({styles: this._changeContrast()});
4     }
5     this._changeTextAlternatives();
6     this._changeFontSize();
7     this._changeFontStyle();
8     this._changeReadingDirection();
9     var thisMap = this.map;
10    google.maps.event.trigger(thisMap, 'idle');
11    google.maps.event.addListener(thisMap, 'tilesloaded', function(){
12        setTimeout(function(){
13            google.maps.event.trigger(thisMap, 'idle');
14        }, 100);
15    }.bind(this));
16 }
```

Listing 9.4: JavaScript Programmcode der Methode adaptive des google-map Elements

Wenn dies der Fall ist, bedeutet das, dass die „styles“ der Karte schon gesetzt werden. In diesem Fall werden die „styles“ über die Methode „setOptions“ neu gesetzt. Da die Methoden zur Adaptivität auf das „idle“ Ereignis warten, wird nach dem Einbinden der Methoden, in Zeile zehn das „idle“ Ereignis gesendet. Dadurch vollzieht das Element das erste Mal die Anpassung an die Präferenzen des Nutzers. Danach wird definiert, dass, wenn die Meldung „tilesloaded“ eintritt, die Meldung „idle“ nach einem sehr kurzen *TimeOut*, also einer Wartezeit, erfolgen soll. Dadurch wird garantiert, dass die zur Laufzeit überschriebenen Werte auch nach dem Wechseln der Karten-Ansicht, also dem Wechsel zwischen der Satelliten- und der Karten-Ansicht, wieder an die Präferenzen der Nutzer angepasst werden. Die Wartezeit muss genutzt werden, da das Ereignis kurz vor der Anpassung zur Laufzeit durch die „google-map“ erfolgt und somit sonst wieder überschrieben werden würde.

9.3.4. Beispiel-Ergebnis der Adaptivität

In der Abbildung 9.6 ist das adaptive „Google Map“ Element dargestellt.

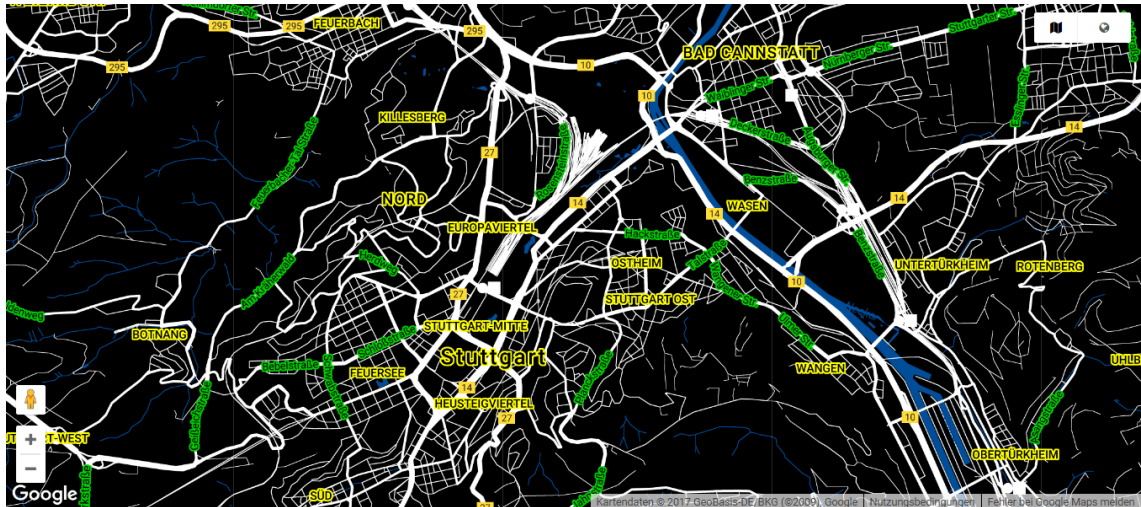


Abbildung 9.6.: Adaptive Google Map Web Component Ansicht Quelle: Bildschirmaufnahme des eigenen, vom Browser übersetzten, Programmcode

Das Element wurde hier beispielhaft an einen Nutzer angepasst, dessen Präferenzwerte einen erhöhten Kontrast, eine Leserichtung von rechts nach links und Symbol-Alternativen statt Texte enthalten.

10. Evaluation der Adaptivitäts-Anpassungen

Nachdem einige vorhandene *Web Components*, realisiert mithilfe der „Polymer“ Bibliothek, zur Adaptivität angepasst wurden, wird im folgenden darauf eingegangen, inwiefern die Adaptivität möglich ist und ob Hindernisse oder Möglichkeiten vorliegen.

10.1. Iron Data Table

Eine Erweiterung zur Adaptivität ist möglich. Sie wird über das programmatische Hinzufügen von CSS Klassen und der darauf folgenden Reaktion über Gestaltungsanweisungen realisiert. Je nach gewählter Präferenz werden verschiedene Klassen an die *Web Components* angehängt und diese innerhalb der Komponenten genutzt um das Aussehen, den Aufbau oder den Inhalt anzupassen. Dies stellt jedoch das erste Problem dar. Jede Tabelle ist aus mehreren *Web Components* zusammengesetzt. Es ist nicht möglich die Gestaltungsanweisungen in einer Komponente für alle Komponenten zu setzen, da *Web Components* über den *Shadow DOM* gekapselt sind. Somit können Gestaltungsanweisungen von einer Komponente keinen Einfluss auf eine andere Komponente nehmen. Deshalb müssen die Methoden zum Hinzufügen der CSS Klassen in einer Komponente definiert werden, während die Methoden die Klassen auch zu anderen *Web Components* hinzufügen müssen. Dies führt dazu, dass die Adaptivität nur vollständig gewährleistet ist, wenn die Zusammenstellung der verschiedenen *Web Components* in einer bestimmten Form vorliegt. Zweitens müssen die Gestaltungsanweisungen zu jeder Komponente innerhalb dieser gesetzt werden. Dies führt dazu, dass der Programmcode der eine Adaptivität ermöglicht, auf verschiedene Komponenten und damit Dateien verteilt werden muss und somit schwerer wartbar ist.

10.2. Paper Date Picker

Auch die Adaptivität dieses Elements wird über das programmatische Hinzufügen von CSS Klassen und eine Reaktion auf diese realisiert. Wiederum werden die Präferenzen des Nutzers anhand von CSS Klassen an die Komponenten hinzugefügt. Hier entsteht dasselbe Problem wie im Kapitel 10.1, da der „Paper Date Picker“ auch aus verschiedenen *Web Components* besteht. So sind auch die Methoden zum Hinzufügen der Klassen in nur einer Komponente, die wiederum diese Klassen an andere Komponenten hinzufügen. Deshalb sollten die verschiedenen *Web Components* in einer bestimmten Zusammenstellung vorliegen. Auch müssen die Gestaltungsanweisungen in jeder Komponente einzeln gesetzt werden was wiederum die Wartbarkeit mindert. Ein weiteres Hindernis zur Ermöglichung von Adaptivität dieser Komponente ist die Erstellung von Elementen zur Laufzeit. Die Komponente erstellt beispielsweise die Hintergrund-Farbänderung des derzeit ausgewählten Datums über das Hinzufügen der Klasse „selected“ bei dem Auswählen dieses Datums. Das Aussehen dieses Elements, also nur dem gewählten Datum, sollte sich bei ermöglichter Adaptivität auch anpassen. Deshalb mussten hier neue Ereignisse erstellt und deren Meldungen abgewartet werden. Beispielsweise wird die Meldung „selectedDate“ ausgegeben wenn ein Datum gewählt wird und bei erhöhter Kontrast-Präferenz des Nutzers diesem Datum die Klasse „moreContrast“ hinzugefügt.

10.3. Google Map

Die Anpassung zur Adaptivität dieses Elements ist problematischer. Es wird nicht alleine im *Template* des *Web Components* definiert, sondern zur Laufzeit bei der Initialisierung über die API erstellt. Dies geschieht über die Methode „_initGMap“. Hier wird ein neues „google.maps.Map“ Objekt erstellt. Dieses erstellt die Elemente der Karte dann programmatisch. Dadurch liegt die Karte nicht von Beginn an vor, sondern wird erst zur Laufzeit der Applikation erstellt. Bei diesem Vorgang wird das Karten-Element auch über die CSS Klasse „style-scope“ gekapselt. Dies führt zu einigen Hindernissen. Zuerst führt die Kapselung dazu, dass Gestaltungsanweisungen keinen Einfluss auf Elemente innerhalb des Karten-Elements haben. Da auf den Programmcode des zur Laufzeit erstellten Karten-Elements, also der API, kein Zugriff besteht und dieser deshalb nicht verändert werden kann, können, im Gegensatz zu den in vorherigen Kapiteln beschriebenen *Web Components*, keine CSS Klassen zur Ermöglichung der Ad-

aptivität genutzt werden. Die Gestaltungsanweisungen müssen zur Laufzeit über JavaScript erstellt werden. Ein weiteres Problem der Erstellung zur Laufzeit des Karten-Elements ist, dass die einzelnen Teile der Karte nicht beim ersten Laden der Internetseite vorhanden sind. So werden viele Elemente der Karte erst im Nachhinein nachgeladen und können nicht von Beginn an über JavaScript angesprochen werden. Deshalb müssen Ereignisse genutzt werden. Diese Ereignisse müssen abgewartet werden und erst nachdem die Meldung erbracht wird, dass sie vollzogen sind kann eine Methode zur Adaptivität ausgeführt werden. Die Karte wird jedoch nicht nur zur Laufzeit erstellt, sondern auch zur Laufzeit aktualisiert. So werden viele Gestaltungsanweisungen der Elemente der Karte bei verschiedenen Aktionen, beispielsweise dem vergrößern oder verkleinern, neu gesetzt oder überschrieben. Deshalb müssen diese Aktionen wieder über bestimmte Ereignisse und deren Meldungen abgewartet werden. Nachdem die Meldungen eintreffen und somit eine Aktion, welche die Komponente verändert hat, erfolgt ist, müssen die Gestaltungsanweisungen zu bestimmten Präferenzen der Nutzer wieder neu gesetzt werden. Zuletzt verwendet die zur Laufzeit erstellte Karte sehr wenig eindeutige Identifikatoren (Ids) oder CSS Klassen. Dadurch ist es schwer, bestimmte Elemente eindeutig anzuwählen und mit diesen zu Arbeiten. Insgesamt ist es komplex das „Google Map“ *Web Component* zur Adaptivität zu erweitern, jedoch ist eine Adaptivität ermöglicht worden.

11. Adaptive Web Components sind möglich

Mithilfe von *Web Components* ist eine kompakte Wiederverwendbarkeit von Komponenten möglich. Diese können beliebig oft und an verschiedenen Stellen von Webanwendungen eingesetzt werden. Diese Abhandlung führt zu dem Schluss, dass es insgesamt möglich ist diese zur Adaptivität zu erweitern. Dafür wird Programmcode innerhalb der Definition des *Web Component* genutzt. In Kapitel 6.2.4 ist dargelegt, dass diese Definition nur Einfluss auf diese eine bestimmte Art eines *Web Component* hat, es kann bei richtiger Anwendung der Technologien ausgeschlossen werden dass Seiteneffekte von diesem auf andere Elemente einer Anwendung auftreten. Der Programmcode reagiert auf von außerhalb gesetzte Eigenschaften, die Präferenzen des Nutzers. Diese Präferenzen können in einer eindeutigen Art und Weise gespeichert werden, in der Form von *Preference Sets*, um später die Eigenschaften der *Web Components* zu bestimmen. Das Herunterladen dieser *Preference Sets* erfolgt über einen *REST-Call*, dargelegt in Kapitel 5.2. Insgesamt können somit *Web Components* über das Definieren von Methoden und Variablen erweitert werden, sich an die Präferenzen des Nutzers anzupassen. Diese Definition basiert auf dem Kapitel 9. Diese Anpassung zur Adaptivität erfolgt beim Aufruf einer Internetseite für jedes *Web Component* einzeln und jedes dieser Komponenten kann beliebig oft in einer Internetseite eingebunden werden.

Jedoch ist eine lückenlose Adaptivität nur unter bestimmten Voraussetzungen möglich. Zuerst müssen die Präferenzen eindeutig definiert werden. Die Bezeichnungen der Präferenzen und auch die Werte dieser müssen eine allgemeingültige, also für jeden Nutzer gleiche, Bedeutung haben. Des weiteren müssen einige Konventionen bei der Erstellung von *Web Components* beachtet werden. Erstens sollten einzelne *Web Components* nicht aus zu vielen Komponenten zusammengesetzt werden. Wie in Kapitel 10.1 dargelegt, muss die Erweiterung zur Adaptivität in jedem dieser Elemente einzeln definiert werden, dies vermindert die Wartbarkeit und es können bei bestimmten Zusammenstellungen dieser einzel-

nen Komponenten ungewollte Seiteneffekte entstehen. Zweitens sollte möglichst die gesamte Komponente im *Template* definiert werden und möglichst wenig zur Laufzeit erweitert oder verändert werden. Dadurch liegt das *Web Component* zu Beginn im (möglichst) vollständigen Zustand vor und die Konzipierung zur Erweiterung zur Adaptivität kann das gesamte Element beachten. Wie Kapitel 10.3 aufgezeigt, ist es problematisch wenn einzelne und insbesondere unterschiedliche Teile der Komponente zur Laufzeit nachgeladen werden, da diese die Definition von Adaptivität im Programmcode erschweren. Dieses Kapitel legt dar, dass Veränderungen zur Laufzeit bestimmte Anpassungen zur Adaptivität überschreiben und die Adaptivität annullieren können.

In der Zukunft der Webentwicklung könnte, wie in Kapitel 6.4.1 dargestellt, eine native Browser-Unterstützung der Technologien der *Web Components* von allen modernen Browsern erfolgen. Dies würde dazu führen dass eindeutige und einheitliche Normen und *Best Practices*, also eine Art Übersicht der besten Methoden und Praktiken, entstehen. Dies würde dazu führen dass Komponenten in ähnlicher oder gleicher Form definiert werden. Dadurch könnte die Definition von Adaptivität für viele verschiedene *Web Components* einheitlich erfolgen. Der Programmcode zur Adaptivität müsste dann nicht extra für jede einzelne Komponente erstellt werden sondern könnte nur in einer Konfiguration angepasst werden. Eine wichtige Frage ist, ob die Adaptivität als eigene Eigenschaft oder Technologie in die *Web Components* aufgenommen werden kann. Da die *Web Components* immer wieder verschiedene Entwicklungen durchgemacht haben und verschiedene Technologien hinzugefügt oder herausgenommen wurden, wie dargestellt in Kapitel 6.1.3, ist es möglich dass die Adaptivität in Zukunft hinzugefügt wird. Dadurch würde der Aufbau von *Web Components* so angepasst werden, dass eine Adaptivität leichter und konsistenter ermöglicht wäre. Eine konsistente Adaptivität von *Web Components* wäre somit garantiert.

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.
- [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] 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/>.
- [7] Balzert, H., Klug, U. & Pampuch, A. (2009). *Webdesign /& Web-Usability: Basiswissen für Web-Entwickler*. W3L-Verlag.
- [8] Bremus, T. (2013). *Barrierefreiheit*. entwickler.press.
- [9] Emrich, C. (2013). *Interkulturelles Marketing-Management: Erfolgsstrategien – Konzepte – Analysen*. Springer Fachmedien Wiesbaden.
- [10] Meidl, O. (2013). *Global Website: Webdesign im internationalen Umfeld*. Springer Fachmedien Wiesbaden.

-
- [11] (2016). *El Shaab*. Abgerufen am 17.11.2016
Abgerufen von
<http://www.elshaab.org/>.
- [12] (2016). *Zeit Online*. Abgerufen am 17.11.2016
Abgerufen von
<http://www.zeit.de/index>.
- [13] Chauhan, S. (2014). *ASP.NET MVC Interview Questions and Answers: Dot Net Tricks*.
- [14] Fscholz et al. (2017). *XMLHttpRequest - Web API Referenz | MDN*. Abgerufen am 3.1.2017
Abgerufen von
<https://developer.mozilla.org/de/docs/Web/API/XMLHttpRequest>.
- [15] Patel, S. K. (2015). *Learning Web Component Development*. Packt Publishing Ltd.
- [16] 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/>.
- [17] C., D. (2015). *Custom Elements v0 - Chrome Platform Status*. Abgerufen am 3.11.2016
Abgerufen von
<https://www.chromestatus.com/feature/4642138092470272>.
- [18] *Custom Elements v0*. Abgerufen am 7.11.2016
Abgerufen von
<http://caniuse.com/%7B%5C#%7Dfeat=custom-elements>.
- [19] Bidelman, E. (2016). *Custom Elements v1: Reusable Web Components*. Abgerufen am 3.11.2016
Abgerufen von
<https://developers.google.com/web/fundamentals/getting-started/primers/customelements>.
- [20] *Firefox Platform Status*. Abgerufen am 3.11.2016
Abgerufen von
<https://platform-status.mozilla.org/%7B%5C#%7Dcustom-elements>.

-
- [21] C., D. (2016). *Custom Elements v1 - Chrome Platform Status*. Abgerufen am 3.11.2016
Abgerufen von
<https://www.chromestatus.com/feature/4696261944934400>.
- [22] Morrita. (2015). *HTML Imports - Chrome Platform Status*. Abgerufen am 3.11.2016
Abgerufen von
<https://www.chromestatus.com/feature/5144752345317376>.
- [23] *HTML templates*. Abgerufen am 3.11.2016
Abgerufen von
<http://caniuse.com/%7B%5C#%7Dsearch=templates>.
- [24] 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>.
- [25] *WEB COMPONENTS CURRENT STATUS*. Abgerufen am 3.11.2016
Abgerufen von
https://www.w3.org/standards/techs/components%7B%5C#%7Dw3c%7B%5C_%7Dall.
- [26] W., R. & K., A. (2015). *<template> Element - Chrome Platform Status*. Abgerufen am 3.11.2016
Abgerufen von
<https://www.chromestatus.com/feature/5207287069147136>.
- [27] *Firefox Platform Status*. Abgerufen am 3.11.2016
Abgerufen von
<https://platform-status.mozilla.org/%7B%5C#%7Dhtml-templates>.
- [28] *Windows 10 build 10547*. Abgerufen am 3.11.2016
Abgerufen von
<https://developer.microsoft.com/en-us/microsoft-edge/platform/changelog/desktop/10547/>.
- [29] Niwa, R. *Webkit Feature Status*. Abgerufen am 7.11.2016
Abgerufen von
<https://webkit.org/status/%7B%5C#%7Dfeature-shadow-dom>.

-
- [30] Hayato. (2016). *Shadow DOM v0*. Abgerufen am 7.11.2016
Abgerufen von
<https://www.chromestatus.com/feature/4507242028072960>.
- [31] Ito, H. (2016). *What's New in Shadow DOM v1 (by examples)*. Abgerufen am 9.11.2016
Abgerufen von
<http://hayato.io/2016/shadowdomv1/>.
- [32] Hayato. (2016). *Shadow DOM v1*. Abgerufen am 7.11.2016
Abgerufen von
<https://www.chromestatus.com/feature/4667415417847808>.
- [33] Denicola, D. (2016). *Custom Elements*. Abgerufen am 1.11.2016
Abgerufen von
<https://www.w3.org/TR/2016/WD-custom-elements-20161013/>.
- [34] 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>.
- [35] 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/>.
- [36] Glazkov, D. & Morrita, H. (2016). *HTML Imports*. Abgerufen am 2.11.2016
Abgerufen von
<https://www.w3.org/TR/html-imports/>.
- [37] Cameron, D. (2015). *HTML5, JavaScript, and jQuery 24-Hour Trainer*.
- [38] 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-ermoeeglicht-40414/>.
- [39] Gasston, P. (2014). *Moderne Webentwicklung: Geräteunabhängige Entwicklung – Techniken und Trends in HTML5, CSS3 und JavaScript*.

-
- [40] 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>.
- [41] Brunt, S. et al. (2013). *A Roadmap to Cracking the PMP® Exam: A PMP Exam Preparation Study Guide*. Trafford Publishing.
- [42] Satrom, B. (2014). *Building Polyfills*. O'Reilly Media.
- [43] *Polyfills*. Abgerufen am 7.11.2016
Abgerufen von
<http://webcomponents.org/polyfills/>.
- [44] 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>.
- [45] 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>.
- [46] Polymer, Authors. (2016). *Declared Properties - Polymer Project*. Abgerufen am 28.11.2016
Abgerufen von
<https://www.polymer-project.org/1.0/docs/devguide/properties>.
- [47] 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>.
- [48] 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>.
- [49] 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>.

-
- [50] REMEX. (2016). *Adaptive Web Components*. Abgerufen am 15.12.2016
Abgerufen von
<http://darwin.gpii.eu/>.
- [51] REMEX. (2016). *REMEXLabs/AWC: Adaptive Web Components Framework*. Abgerufen am 15.12.2016
Abgerufen von
<https://github.com/REMEXLabs/AWC>.
- [52] Handlebar, Authors. *Handlebars.js: Minimal Templating on Steroids*. Abgerufen am 16.12.2016
Abgerufen von
<http://handlebarsjs.com/>.
- [53] *Custom Elements*. Abgerufen am 21.11.2016
Abgerufen von
<https://customelements.io/>.
- [54] Tähkäpää, Sauli. (2016). *Saulis/iron-data-table*. Abgerufen am 30.12.2016
Abgerufen von
<https://github.com/Saulis/iron-data-table>.
- [55] *Bower - a package manager for the web*. Abgerufen am 8.12.2016
Abgerufen von
<https://bower.io/>.
- [56] Media, Ben Davis. *bendavis78/paper-date-picker: Material design date picker component for polymer*. Abgerufen am 18.12.2016
Abgerufen von
<https://customelements.io/bendavis78/paper-date-picker/>.
- [57] Media, Ben Davis. (2016). *bendavis78/paper-date-picker: Material design date picker component for polymer*. Abgerufen am 18.12.2016
Abgerufen von
<https://github.com/bendavis78/paper-date-picker>.
- [58] *google-map*. Abgerufen am 21.11.2016
Abgerufen von
<https://elements.polymer-project.org/elements/google-map>.
- [59] Google, Authors. (2016). *GoogleWebComponents/google-map: Google Maps web components*. Abgerufen am 21.11.2016
Abgerufen von
<https://github.com/GoogleWebComponents/google-map>.

- [60] (2016). *Style Reference*. Abgerufen am 24.11.2016
Abgerufen von
<https://developers.google.com/maps/documentation/javascript/style-reference%7B%5C#%7Dstyle-elements>.
- [61] Gandy, Dave. *Font Awesome, the iconic font and CSS toolkit*. Abgerufen am 16.12.2016
Abgerufen von
<http://fontawesome.io/>.

Anhang

Im Folgenden wird eine Übersicht der Daten auf der beigelegten CD gegeben. Die Ordnerstruktur setzt sich wie folgt zusammen.

1. Programmcode

1.1 Globale Lizenz des erstellten Programmcodes

1.2 Beispielprogrammcode

1.2.1 Genutzte Beispiele

- imports
- polymer

1.2.2 AWC

- assets
- comps
- lib
- spec
- src

1.3 Elemente zur Ermöglichung von Adaptivität

1.3.1 REST-client

1.3.2 Preference Sets

1.4 Erweiterte Web Components

1.4.1 iron-data-table

- bower_components
- demo
- test

1.4.2 paper-date-picker

- bower_components

- demo

- test

1.4.3 google-map

- bower_components

- css

- demo

- test

- tmp

1.5 Demonstrationswebseite

- bower_components

- css

- imports

- javascript

2. Demonstrationsvideo

3. GitHub Links

4. Thesis