Erstellung von adaptiven Web Components

Christoph Kleber

9. November 2016

Kurzfassung

In dieser Arbeit geht es um Web Components.

Abstract

This thesis is about Web Components.

Listing Verzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

API Advanced Programming Interface

DOM Document Object Model

HTML Hypertext Markup Language

URL Uniform Resource Locator

div division

HTML5 Hypertext Markup Language Version 5

CSS Cascading Style Sheets

W3C World Wide Web Consortium

1 Inhaltsverzeichnis

2 Adaptivität

2.1 Begriffsklärung

Das ist erster Text Test

2.2 Adaptivität bei User Interfaces

3 Web Components

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

3.2 Geschichte von Web Components

Web Components wurden vom W3C das erste Mal im Jahr 2012 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 und stattdessen wird die *HTML Import* Technologie verwendet.[3] 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 eingegangen.

3.2.1 Custom Elements !v0 und v1!

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

[1] vgl. Patel. 2015, S. 1.
[2] vgl. Patel. 2015, S. 2.
[3] vgl. Cooney und Glazkov. 2012.

Opera in der Version 20 unterstützt. In den mobilen Varianten dieser Browser wird sie seit 2016 von *Opera for Android* in der Version 37 und von *Chrome for Android* in der Version 53 unterstützt.[4] Im *Android* Browser wird diese Version schon seit 2014 unterstützt, in der *Android* Version 4.4.4. Im *Samsung Internet* werden sie seit 2016 in der Version 4 genutzt.[5] Die Version v0 wird von der Version v1 abgelöst, hier ergeben sich einige Änderungen in der Syntax der Advanced Programming Interface (API).[6] Derzeit wird sie nicht per Standardeinstellung von anderen Browsern unterstützt.[7] Die Version v1 wird derzeit nur von den Browsern *Chrome* und *Opera* unterstützt. Das erste mal wurde sie im Jahr 2016 in der *Chrome* Version 54 und in der *Opera* Version 41 genutzt.[8]

3.2.2 HTML Imports

HTML Imports wurden in den Browsern *Chrome* und *Opera* zuerst 2014 unterstützt. Die Imports wurden als Erstes in der *Chrome* Version 36 und in der *Opera* Version 23 genutzt. Gegenwärtig wird die Technologie von den zuvor erwähnten Browsern auch in den mobilen Browser-Varianten unterstützt, in den Versionen 53 des *Chromium* und 37 des *Opera for Android*.[9] Der *Android* Browser unterstützt *HTML Imports* seit 2016 in der Version 53. Der Browser des *Android* Betriebssystems ist ab dem Jahr 2016 in der Version 53 kongruent mit dem Chromium Browser. *Samsung Internet* unterstützt die Imports seit 2016 in der Version 4.[10]

3.2.3 Decorators

Decorators erscheinen nur in Dokumenten und Artikeln, sie werden 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“.[11] Dies zeigt, dass an dieser Stelle noch keine Implementierung dieser Technologie vorliegt. Auch in einem *Working Draft* des W3C vom Jahr 2013 wird davon gesprochen, dass „Decorators, unlike other parts of Web

[4] vgl. C. 2015.
[5] vgl. ACHTUNGcaniuse. 9999.
[6] vgl. Bidelman. 2016.
[7] vgl. ACHTUNGfirefox. 9999.
[8] vgl. C. 2016.
[9] vgl. Morrita. 2015.
[10] vgl. ACHTUNGcaniuse. 9999.
[11] Cooney und Glazkov. 2012.

Components, do not have a specification yet.“[12] Auf einer aktuellen Übersichtsseite des Konsortiums wird die *Decorators* Technologie nicht mehr im Zusammenhand mit *Web Components* erwähnt.[13]

3.2.4 Templates

Templates werden schon über einen längeren Zeitraum in den verschiedenen Browsern unterstützt. Zuallererst wurden sie im *Chrome* im Jahr 2013 verwendet, in der Version 26. Im selben Jahr wurden sie vom *Firefox* Browser in der Version 22 und vom *Opera* Browser in der Version 15 unterstützt.[14][15] Im Jahr 2015 wurden sie dann vom *Edge* Browser unterstützt, in der Version 13.[16] Auf den Browsern des *Macintosh* Betriebssystems wurden *Templates* zuerst 2014 verwendet, in der *Safari* Version 7.1 und der *Safari & Chrome for iOS* Version 8.[17] In den meisten mobilen Varianten der Browser werden Templates seit 2016 bereitgestellt. So erschien dieses Feature 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, seit der Version 4.4[18]

3.2.5 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 Safari Version 10 kann der *Shadom DOM* seit 2016 genutzt werden.[19] Die mobilen Varianten der Browser unterstützen das *Shadow DOM* seit 2016, *Opera* in der Version 37 und *Chrome* in der Version 53, somit auch der *Android* Browser.[20] Die Version v0 wird von der Version v1 abgelöst, welche verschiedene Neuerungen in der Syntax, aber auch in der Unterstützung von bestimmten Funktionen aufweist. So kann beispielsweise in der v0 ein *shadow root* immer nur als „open“ definiert werden,

[12] Cooney und Glazkov. 2013.
[13] vgl. ACHTUNGw3c. 9999.
[14] vgl. W und K. 2015.
[15] vgl. ACHTUNGfirefox. 9999.
[16] vgl. ACHTUNGmicrosoft. 9999.
[17] vgl. ACHTUNGcaniuse. 9999.
[18] vgl. ACHTUNGcaniuse. 9999.
[19] vgl. Niwa. 9999.
[20] vlg. Hayato. 2016.

in der v1 kann er auch als „closed“ *shadow root* erstellt werden.[21] Die Version v1 wird noch nicht so großem Ausmaß wie die Version v0 unterstützt. Vollständig wird sie nur vom *Chrome* 53 und *Opera* 40 Browser unterstützt, jeweils seit 2016. In den mobilen Versionen wird sie nur von dem *Chromium* und somit auch auf *Android* in der Version 53 unterstützt.[22]

3.3 Vergleich Webentwicklung mit Web Components und ohne

Webapplikationen basieren auf vielen verschiedenen Technologien, beispielsweise HTML, Cascading Style Sheets (CSS) und *JavaScript*. Meistens werden darüber hinaus noch verschiedene *Frameworks* verwendet. „A computer system framework has a layered structure that can incorporate servers, operating systems, client machines, and applications. The framework can provide a set of functions to definde application interfaces, the interrelationships between applications, and internal communications between clients and external to online platforms“.[23] Ein *Framework* ist somit ein System, das den Entwicklern bestimmte Funktionalitäten zur Verfügung stellt, ohne dass dieser sie selbst programmieren muss. Diese können beispielsweise die Hilfe bei der Interaktion mit dem DOM sein, wie das *JavaScript Framework* „jQuery“, ein Slide-Element bereitstellen wie das *Framework* „Slider“ oder das Backend einer Webseite definieren wie das *Framework* „TYP03 CMS“. Hier ergibt sich großes ein Problem. Wenn in einer Applikation mehrere *Frameworks* und Technologien für verschiedene Funktionen verwendet werden, können diese sich gegenseitig beeinflussen. So können die *Style* Regeln verschiedener Teile der Webseite sich unbeabsichtigt beeinflussen oder das *JavaScript*, welches eine bestimmte Funktion hat, an einer anderen Stelle für welche es nicht programmiert wurde, Einfluss nehmen. Darüber hinaus können viele Teile der Webseite weder wiederverwendet, noch gut gewartet werden können, da sie großen Einfluss aufeinander nehmen und somit sehr ineinander verschachtelt sind. Die *Web Components* versuchen diese Probleme durch eine (in Zukunft) native Implementierung verschiedener Techniken anzugehen, welche eine Kapselung, eine Wiederverwendung und eine leichtere Wartbarkeit von Programmcode ermöglichen.

[21] vgl. Ito. 2016.
[22] vlg. Hayato. 2016.
[23] Stuart Brunt. 2013, S. 15.

3.3.1 Vorteil: Wenn Browserunterstützung gegeben: native, kein Framework

Wenn es in der Zukunft der Fall sein wird, dass *Web Components* nativ von allen Browsern unterstützt werden, ergibt sich daraus ein großer Vorteil. Es muss bei der Nutzung nativer, also von den Browsern implementierten Techniken, kein externer Programmcode genutzt werden um bestimmte Funktionen abzudecken. Viele Funktionalitäten können einfach über die Nutzung nativer Methoden abgebildet werden. Deshalb müssen weniger Funktionen selbst geschrieben werden und weniger, beziehungsweise unter Umständen keine *Frameworks* genutzt werden. Dies verkleinert das Laden von externen Programmcode. Darüber hinaus ist die Syntax und Funktionsweise bei nativen Funktionen bekannt und eindeutig. Daraus ergeben sich weniger Inkonsistenzen in der Programmierung und eine leichtere Verständlichkeit. Im Gegensatz dazu muss bei vielen *Frameworks* eine jeweils eigene Syntax benutzt werden.

3.3.2 Kapselung

Ein Mechanismus zur Datenkapselung wird vom *Shadow DOM* bereitgestellt. Dieser ermöglicht, dass der Programmcode des *Web Components* vom Rest der Applikation getrennt werden kann. Dadurch wird ein privater *Scope*, also ein Geltungsbereich der Applikation und dessen Variablen, Methoden, Bezeichnern und ähnliches, genutzt.[24] Dies hat einige Folgen für das Verhalten einer Applikation. Zuerst ist der *Shadow DOM* isoliert, er kann nicht von außerhalb angesprochen werden, beispielsweise über die Funktion *document.querySelector()*. Dies hat den Vorteil, dass die Funktionalität des *Web Component* nicht von außen beeinträchtigt werden kann. Des weiteren hat das CSS nur Zugriff auf den DOM des eigenen Geltungsbereichs, weder von außerhalb des *Shadow DOM* können *Style* Regeln Einfluss auf diesen nehmen, noch können *Style* Regeln von innerhalb nach außen Einfluss nehmen. Ein Vorteil an dieser Eigenschaft ist, dass man atomare, also sehr einfache, CSS Bezeichner innerhalb des *Shadow DOM* verwenden kann und dieselben Bezeichner gleichzeitig außerhalb dieses nutzen kann.[25] Darüber hinaus wird das *Styling* der Applikation konsistenter, es erfolgt einzeln für jede *Web Component* und für den Bereich außerhalb der *Web Components*.

[24] vgl. Patel. 2015, S.2.
[25] vgl. Bidelman. 2016.

3.3.3 Wiederverwendung

Die Theorie der *Web Components* ist die Erstellung von verschiedenen Komponenten, die immer wieder wiederverwendet werden können. Dies hat zum einen den großen Vorteil, dass es eine Interoperabilität zwischen *Frameworks* ermöglicht.[26] Dadurch ist man nicht an ein bestimmtes *Framwork* gebunden und kann auch mit außerhalb dieses Ökosystems interagieren und Komponenten wiederverwenden. Zum anderen können auch innerhalb eines *Frameworks* oder nativ, ohne das Nutzen eines *Frameworks*, Teile einer Anwendung wiederverwendet werden, was eine Arbeitserleichterung und Verminderung des Programmcodes hervorruft.

3.3.4 Wartbarkeit

Die Wartbarkeit von Web-Applikationen wird erleichtert, da die *Web Components* in *Templates* organisiert sind.[27] Das sorgt dafür dass der Programmcode einzelner Komponenten separat gespeichert wird und somit leichter wiedergefunden und geändert werden kann.

3.3.5 Nachteil: Browserunterstützung

Auch wenn *Web Components* sehr viele Vorteile aufweisen, ist es ein Problem wenn die Techniken nicht in allen Browsern unterstützt werden oder unterschiedlich implementiert sind. Wie in Kapitel 3.2 dargelegt, sind einige der Techniken noch nicht von allen Browsern unterstützt, oder unterscheiden sich in deren Ausführung. Dies kann zu Inkonsistenzen oder dem nicht funktionieren einer Applikation führen. Dies kann jedoch umgangen werden, indem *Polyfills* verwendet werden. Das sind in diesem Zusammenhang Programmcodes, welche die Funktionen oder Teile einer Technologie, die nativ noch nicht von einem Browser unterstützt wird, nachstellen. Diese können dann verwendet werden um Nutzern aller Browser die Technologien gebrauchen zu lassen.[28] Das *webcomponents.js* ist ein Set von *Polyfills* und ermöglicht den Nutzern die Verwendung von *Web Components* in allen modernen Browsern.[29]

[26] vgl. Patel. 2015, S.2.
[27] vgl. Patel. 2015, S.2.
[28] vgl. Satrom. 2014, S. 4.
[29] vgl. V.

3.4 Technik der Web Components

3.4.1 Custom Elements

Das *Custom Element* ist eine *API*, welches das Bilden eigener, voll funktionstüchtiger *DOM* Elemente ermöglicht.[30] Die *API* beschreibt in diesem Zusammenhang eine Schnittstelle, welche einem anderen Programm ein Tool zur Verfügung stellt, um sich an das eigene Softwaresystem anbinden zu können.[31] Somit ermöglicht eine *API* einen Austausch von Informationen zwischen verschiedenen Programmen oder Systemen. Die *Custom Element API* ermöglicht den Nutzern die Auszeichnungsprache *HTML* zu erweitern.[32] Es können bestehende *HTML* Elemente erweitert, oder neue hinzugefügt werden. Jedes neue oder erweiterte Element wird unter einem *Tag* Namen registriert. Dies ermöglicht eine Kapselung des erstellen Programmiercodes in Elemente. In Listing 3.1 ist ein *JavaScript* Programmcode dargestellt, welcher ein leeres *Custom Element* definiert und und unter dem Namen „new-custom-element“ registriert wird. Für *Custom Elements* sind mehrere *Callbacks* verfügbar. *Callback*

```
class NewCustomElement extends HTMLElement {
        constructor() {
                super();
        }
}
customElements.define('new-custom-element', NewCustomElement);
```

Listing 3.1: Custom Element JavaScript

Funktionen beschreiben hier Funktionen, die bei bestimmten Ereignissen des *Lifecycle* von außerhalb des *Custom Elements* aufgerufen werden. Im folgenden werden diese Funktionen aufgelistet.[33]

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.

[30] vgl. Denicola. 2016.
[31] vgl. Behrendt. 2016.
[32] vgl. Argelius. 2016.
[33] vgl. Argelius. 2016.

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

3.4.2 HTML Imports

HTML Imports ist eine Technologie zum Importieren von externen HTML Dokumenten in ein HTML Dokument. Hier ist zu unterscheiden zwischen importierenden und importierten HTML Dokumenten. Die importierenden Dokumente besitzen einen Link, welcher mindestens die Uniform Resource Locator (URL) des *Imports* und die Eigenschaft *rel=„import“* besitzt,also ein Link eines bestimmten Typ ist, siehe Listing 3.2.[34] Die importierten Dokumente haben keinen außergewöhnlichen Aufbau im Vergleich

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

Listing 3.2: Standard HTML Import

zu normalen HTML Dokumenten, sie können aus HTML, *Style* oder *Script* Elementen bestehen. Es kann auch die *Template* Technologie verwendet werden, dazu mehr in 3.4.3. Um auf den Inhalt des importierten Dokuments zuzugreifen wird *JavaScript* verwendet. Wie in Listing 3.3 dargestellt, wird zuerst nach dem Link Element gesucht, welches die Eigenschaft *rel=„import“* besitzt. Daraufhin wird dieses Dokument importiert und ein bestimmter Teil des Dokuments als *JavaScript* Variable *„elemt“* gespeichert. Hier wird ein division (div) Element, welches die Klasse *„element“* besitzt gespeichert. Dieses kann dann in der importierenden Seite genutzt werden.

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

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

[34] vgl. Glazkov und Morrita. 2016.

3.4.3 Templates

Das Hypertext Markup Language Version 5 (HTML5) Feature *Templates* ermöglicht Teile einer Seite unabhängig vom DOM zu erstellen. Diese können dann später programmatisch zum DOM hinzugefügt werden.[35] Dies 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.[36] In 3.4 sieht man den *JavaScript* Code um ein vorhandenes *Template* zum DOM hin-

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

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

zuzufügen. 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 Inhalt hinzugefügt zu werden. In diesem Moment werden auch die Bilder des *Templates* geladen und eventuelle Skripte ausgeführt.

3.4.4 Shadow DOM !SLOTS!

„Das *Shadow DOM* beschreibt die Fähigkeit eines Browsers, eine neue, völlig abgekapselte Knotenstruktur im bestehenden DOM zu erzeugen.“[37] Dies bedeutet, dass neben dem normalen *Document tree*, wessen Wurzelknoten ein Dokument ist, noch der *Shadow tree* besteht. Der Wurzelknoten des letzteren ist kein Dokument, sondern der *Shadow root*. Dies ist in Abbildung 3.1 dargestellt.[38] Die Folge dieser Kapselung ist, dass alles was dem *Shadow tree* hinzugefügt wird, nur lokal Einfluss auf diesen hat. Die Gestaltung von Webelementen im *Shadow root* wird dadurch vereinfacht. CSS Selektoren können nicht von außerhalb des *Shadow roots* auf diesen zugreifen und Selektoren, die innerhalb dieses definiert werden haben keinen Einfluss auf den normalen DOM. Genauso verhält es sich mit dem Zugriff auf die DOM Elemente des *Shadow root*. Sie können nicht von außerhalb angesprochen werden, beispielsweise

[35] vgl. Cameron. 2015, S. 177.
[36] vgl. Potschien. 2013.
[37] Gasston. 2014, Kap. 11.1.4.
[38] vgl. Patel. 2015, S. 22.

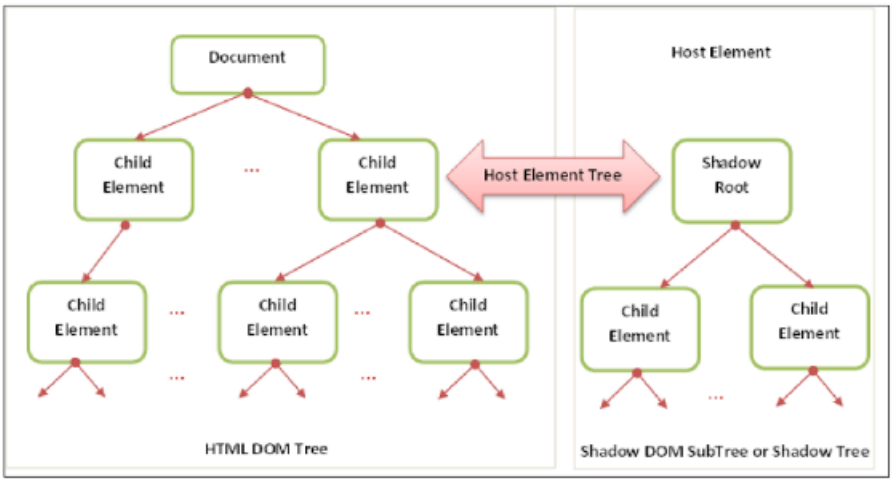

Abbildung 3.1: DOM und Shadow DOM

durch die Funktion *document.querySelector()*, sondern können nur von Funktionen innerhalb des *Shadow root* angesprochen werden.[39] In Listing 3.5 ist dargestellt, wie

```
var header = document.createElement('header');
var shadowRoot = header.attachShadow({mode: 'open'});
var headline = document.createElement("h1");
var headlineText = document.createTextNode("headline");
headline.appendChild(headlineText);
shadowRoot.appendChild(headline);
```

Listing 3.5: JavaScript Code für das Erstellen eines Shadow DOM

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

3.4.4.1 Slots

Shadow DOM kann auch mit anderen DOM erweitert werden. Der Entwickler kann hier gestatten, dass sein *Web Component* mit Elementen, welche der Nutzer seines *Web Component* definiert, erweitert wird. Hierfür werden *Slots* verwendet. In Listing 3.6 ist HTML Code des *Web Component* dargestellt, welcher im *Shadow DOM* später

[39] Bidelman. 2016.

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

Listing 3.6: Nutzung von Slot im Shadow DOM

gerendert wird. Hier ist die Verwendung des *Slot* Elements interessant. Dieses kann beim Einbinden des *Web Component* später ausgestattet werden. Wird das Element nicht befüllt, wird das *Fallback*, der Inhalt innerhalb des Elements, hier beispielsweise „Kein Titel“ genutzt. Das *Fallback* kann auch aus einem eigenen DOM Baum bestehen, wie im *Slot* „name“ zu sehen ist.[40]

[40] Bidelman. 2016.

4 Methodik dieser Arbeit

5 Adaptive Web Components

5.1 Identifikation passender Web Components

5.1.1 Identifikation Web Components

5.1.2 Identifikation passender Preference Terms

5.2 Preference Sets zur Adaptivität

5.3 Adaptivität der bestehenden Web Components

5.3.1 Web Component Eins

5.3.1.1 Konzeption zur Adaptivität

5.3.1.2 Umsetzung Programmierung

5.3.2 Web Component Zwei

5.3.3 Web Component Drei

6 Vergleich

6.1 Vergleich mit Polymer

Literatur

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[15] Dominic Cooney und Dimitri Glazkov. *Introduction to Web Components*. 2013. URL: https://www.w3.org/TR/2013/WD-components-intro-20130606/%7B%5C#%7Ddecorator-section (besucht am 03. 11. 2016).

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

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

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

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

[20] Hayato. *Shadow DOM v1*. 2016. URL: https://www.chromestatus.com/feature/4667415417847808.

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

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

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

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

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

[26] B Satrom. *Building Polyfills*. O'Reilly Media, 2014. ISBN: 9781449370718. URL: https://books.google.de/books?id=PpbiAgAAQBAJ.

[27] P.M.P.P.M.P.P.M.I.R.M.P. Stuart Brunt. *A Roadmap to Cracking the PMP{®} Exam: A PMP Exam Preparation Study Guide*. Trafford Publishing, 2013. ISBN: 9781466985209. URL: https://books.google.de/books?id=6nsNsUH6EBgC.

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

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