# Webcomponents

## Standardisierte und wiederverwendbare Komponenten fürs Web

**Webtechnologien altern schnell. Zudem wechseln Technologie-Stacks von Projekt zu Projekt. Deshalb lassen sich Komponenten der einen Anwendung selten in anderen wiederverwenden. Webcomponents lösen das Dilemma auf, indem sie etablierte Standards verwenden, Funktionalität und Technologie kapseln sowie die grundlegende Abhängigkeit von Frameworks ablösen.**

### Definition

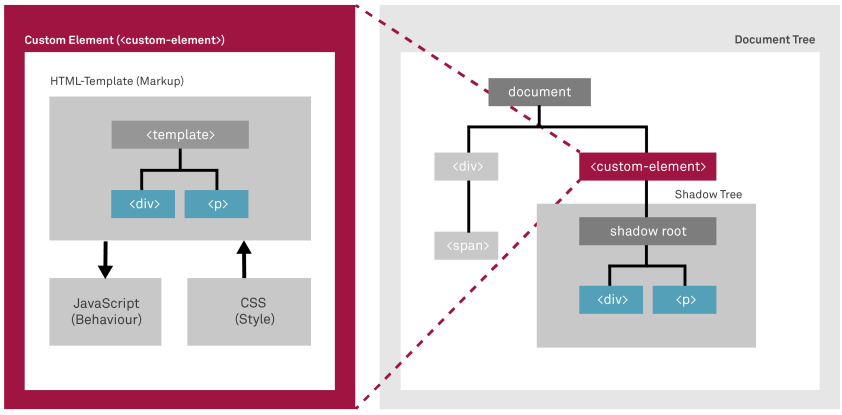
Die Webcomponents-Spezifikation fußt auf den drei standardisierten Webtechnologien #CustomElements, #HTMLTemplates und #ShadowDOM. Die Custom Elements-Spezifikation ist dabei Dreh- und Angelpunkt. Sie klammert das Verhalten (JavaScript), die Struktur (HTML) und das Aussehen (CSS) zu einer wiederverwendbaren Webkomponente. Die so entstehenden Webkomponenten lassen sich dann genau wie die im HTML-Standard vorgesehenen, primitiven Komponenten verwenden.

HTML-Templates deklarieren wiederverwendbare HTML-Fragmente, die sich etwa als Vorlage für die Unterstrukturen eines Custom Elements eignen. Ein Custom Element kann also aus einem oder mehreren anderen primitiven oder Custom Elements bestehen. Die Spezifikation des Shadow-DOMs wiederum bewirkt, dass eine Webcomponent - genauer deren Verhalten, Struktur und Aussehen - gekapselt und in sich geschlossen bleibt. Dazu herrscht innerhalb des Shadow-DOMs ein eigener Gültigkeitsbereich, der isoliert vom primären DOM der Webseite ist. Der Shadow-DOM schafft auf diesem Weg eine vom DOM unabhängige, diesem aber trotzdem untergeordnete Struktur. Das vermeidet ungewollte Wechselwirkungen mit anderen Komponenten als auch mit anderen Instanzen derselben Webcomponent.

Hilfreich: Custom Elements sind dem Webbrowser zunächst unbekannt und müssen registriert werden. Entsprechende Mechanismen zur Registrierung sieht der Standard deshalb vor. Zur besseren Unterscheidung von primitiven und Custom Elements gilt zudem die Regel, dass der Name eines Custom Elements einen Bindestrich enthalten muss, während primitive niemals einen Bindestrich enthalten werden.

### Referenzszenario

Die Entwicklung einer weiteren Webanwendung steht an. Aus mehreren Gründen ist der bisher verwendete Technologie-Stack ungeeignet und zum geplanten neuen Stack inkompatibel. Dennoch sollen bereits für andere Webanwendungen entwickelte GUI-Komponenten wiederverwendet werden. Anstatt diese auf Basis des neuen Stacks neu zu implementieren, lassen sie sich als Webcomponent kapseln, portieren und einsetzen.




### Potenzial

UI-Frameworks lösen bereits seit einiger Zeit die hohen Anforderungen und die steigende Komplexität in der Client-Entwicklung. Sie alle verfolgen dabei eigene Ziele, gehen unterschiedlich vor, haben unterschiedlich lange Lebensspannen und sind zueinander nicht kompatibel. Langlebige Unternehmensanwendungen leiden unter Umständen darunter, weil Erweiterungen und Wartungen sehr kostenintensiv ausfallen können.

Webcomponents schließen als universelle, standardisierte Lösung die Lücke der Framework-spezifischen, inkompatiblen Komponenten. Es reicht aus, eine Version einer UI-Komponente zu entwickeln und zu pflegen. Sie ermöglichen sogar die Kombination unterschiedlicher Frameworks, stellen ein einheitliches Gefüge bei Aussehen und Bedienung sicher, was sich positiv auf Anwenderzufriedenheit auswirkt.

.msg

### Webtechnologie

- Entwicklung austauschbarer Komponenten für webbasierte Anwendungen
- stellt höhere Anforderungen an Interaktivität im Frontend, wodurch die Komplexität wächst

### Microfrontends

- schneiden große Anwendungen in fachliche, modulare Teile
- haben unterschiedliche Technologie-Stacks

**WBC**

### ES-Modules

- offizielles, standardisiertes Modulsystem für Javascript
- gute Basis für Custom Elements
- nicht im Zuge der Webcomponents-Spezifikation entstanden

### Frameworks

- Angular, React, VueJS
- Integration von externen Webcomponents möglich
- Export von Komponenten als Webcomponents möglich

## Reifegrad

Bereits 2013 veröffentlichte Google mit Polymer ein auf Webcomponents aufbauendes Framework. Die ersten APIs waren sehr komplex zu implementieren, es mangelte an Unterstützung durch die Webbrowser und die notwendigen Polyfills resultierten in geringer Performanz und hoher Netzwerklast durch den zusätzlich benötigten Javascript-Code.

In Form der drei Webstandards steht ein natives Komponentenmodell zur Verfügung, das alle wichtigen Webbrowser unterstützen. Der Einsatz von Polyfills entfällt. Zudem lassen sich die Webcomponents direkt im Webbrowser debuggen, wenn auch noch mit einigen Hürden.

Die Entwicklung von Webcomponents bedingt noch viel technischen Code sowie umfangreiche manuelle Maßnahmen, falls keine Hilfsbibliotheken genutzt werden. Es bleibt ungewiss, ob der Standard irgendwann einen Reifegrad erreicht, der Bibliotheken und Frameworks verzichtbar macht. Dem mehrwertigen Einsatz von Webcomponents steht dies aber nicht im Weg.

## Marktübersicht

Die bekannten Frameworks Angular, React und Vue exportieren UI-Komponenten bei Bedarf als Webcomponents. Sie können aber auch als Webcomponents entwickelte UI-Bibliotheken unterschiedlicher Quellen integrieren und mischen, etwa die beliebten Open UI5 und SAP UI5.

Der Überschuss an noch immer notwendigem technischem Code (Boilerplate) lässt sich mit dem Einsatz von Hilfsbibliotheken wie Litelement, Slim.js oder Smart.js kompensieren. Das reduziert Zeit, Aufwand und Verzögerungen in der Entwicklung.

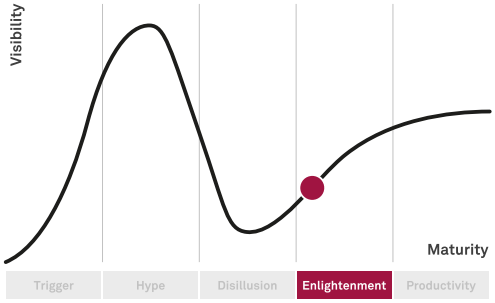
## Alternativen

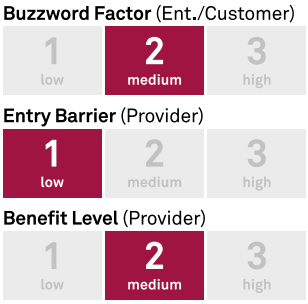
Zur Entwicklung von Webwendungen sind Frameworks wie Angular, React oder Vue weiterhin geeignet. Während Webcomponents dabei ausschließlich Komponenten abbilden, beherrschen die Frameworks auch die Komplexität der Architektur selbst. Webcomponents sind daher in diesem Zusammenhang eine Ergänzung statt reiner Alternative.

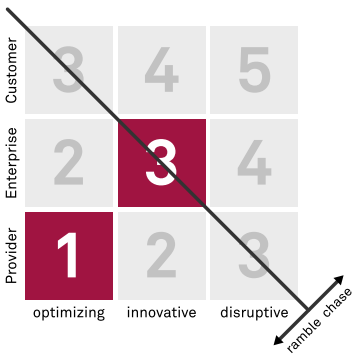
Die Eigenschaften der Kapselung, Wiederverwendbarkeit und Framework-Unabhängigkeit lassen sich auch mit iFrames abbilden. Als Herausforderung gilt dabei die Kommunikation der iFrames untereinander und mit der eigentlichen Webanwendung.

## Fazit

- \+ basieren auf Webstandards
- \+ sind unabhängig von Frameworks
- \+ vermeiden Interferenzen von UI-Komponenten
- \+ sind austausch- und wiederverwendbar
- \+ trennen Entwicklungsteams organisatorisch
- \- hängen im selben Namespace
- \- erfordern viel technischen Code
- \- schreien nach Hilfsbibliotheken
- \- lassen sich nur schwierig debuggen




https://msg.direct/techrefresh

Stand: September 2020

**msg systems ag**

Robert-Bürkle-Straße 1 | 85737 Ismaning/München | Telefon: +49 89 96101-0 | Fax: +49 89 96101-1113 | www.msg.group | info@msg.group