

Entwicklung einer Web-Components-Library

Manuel Mauky





Manuel Mauky

 @manuel_mauky



<https://www.zeiss.com/digital-innovation>

Kontext

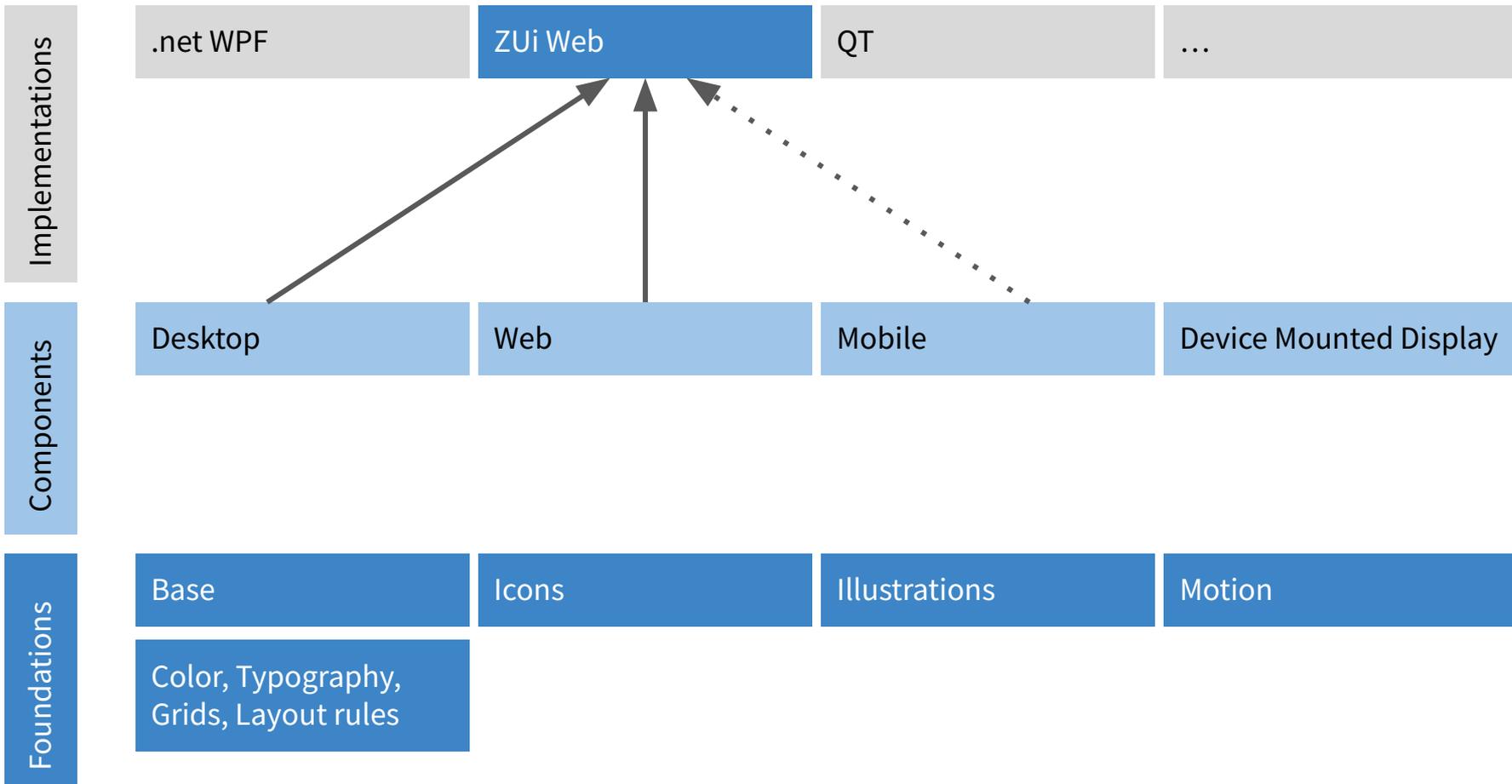
- ZEISS baut optische Geräte, u.a.
 - Medizintechnik
 - Industrielle Messtechnik
 - Microscopie
- Heutzutage (fast) immer dabei: Computer und Software
- Ziel: Einheitliches Look&Feel vermittelt hohe Qualität, steigende User Experience durch einheitliche Bedienkonzepte und Funktionsweisen

ZEISS UI

ZUi Design System

"A **design system** is a complete set of standards intended to manage design at scale using reusable components and patterns."

Therese Fessenden (Nielsen Norman Group)



ZUi Web

ZUi Web

- Web-Component Library
- wird momentan von ~18 ZEISS-Projekten verwendet
 - mehrheitlich Angular, einige React
- Start Entwicklung der Library: Dezember 2019; Februar 2021 Go Live
- zwischen 3-8 Leute, ca. 3 Jahre Entwicklung
- ca. 80 Komponenten
- 170.865 lines of code, davon 61.535 TypeScript

ZUi-Web in Anwendungen

The screenshot displays the ZEISS FORUM Cloud Viewer web application. At the top, there is a search bar with the text "Search for cases" and a magnifying glass icon. On the right side of the top bar, it says "CARin Test" next to a blue circular icon with "CT".

The main interface is divided into several sections:

- Left Sidebar:** Contains navigation icons for home, calendar, play, list, and a profile icon. The profile icon is labeled "DU".
- Top Left:** Shows the ZEISS logo and the text "FORUM Cloud Viewer".
- Left Panel:** A list of patients under the heading "6 PATIENTEN". The patients listed are:
 - Chap, Behoerchen (06.06.2018 (3) | 001)
 - Chip, Ahoerchen (06.06.2018 (3) | AVeryLongPatientID)
 - Dagobert, Duck (06.06.1875 (146) | -1)
 - Gustav, Gans (06.06.1885 (136) | 77)
 - TC, 13340 (18.05.1952 (69) | TC13340)
 - Validation, iPP01 (06.06.2018 (3) | 35)
- Center Panel:** Displays the selected patient "Chip, Ahoerchen" with details: "06.06.2018 (3) | AVeryLongPatientID". It shows a grid of image thumbnails for different dates and views:
 - 20.04.2018:** Three thumbnails labeled "Bild", "OD", and "OS".
 - 02.02.2018:** Three thumbnails labeled "Bild", "OD", and "OS Macular Thickness".
 - 02.02.2017:** Three thumbnails labeled "Bild", "OD", and "COLOR".
- Right Panel:** A large, detailed view of an eye scan from "20.04.2018 | 08:00 | BILD | OD". It shows a fundus image with a red overlay indicating retinal vessels. A search and navigation toolbar is visible at the bottom of this panel.

ZUI-Web im Styleguide und Library

The screenshot displays the ZUI-Web styleguide and library interface. The top navigation bar shows the current page as "00 Styleguide / Styleguide - Web" and includes a "Share" button and a 50% zoom level. The left sidebar, titled "Pages", lists various components such as "Responsive Grid", "Avatar", "Focus View", "Header Bar", "Menu Bar", "Page Header", "Search Field", "Tag", and "WIP". The "Header Bar" component is currently selected, and its sub-components, including "Headerbar", "Headerbar Variations", "Icon bar", "Icons", "Inline Message", "InteractiveIcon", "InteractiveText", "List", "Menu", "Menu bar", "PageHeader", "Pagination", and "Picker", are listed in the main content area.

The central canvas area shows a preview of the "ZUI web component library" header bar. The top navigation bar includes the ZUI logo, the text "ZUI web component library", a search bar, and a user profile for "chief surgeon Dr. Jane Montgomery-D...". Below the canvas, the "Actions" section shows "Knobs (18)" and "Snapshots". The "Container width (px)" knob is set to 1194 / 1920. The "Icon (slot: 'icon')" knob shows the code: `<zui-icon-zeiss size='xl' slot='icon'></zui-icon-zeiss>`. The "ProductName - productName" knob is set to "ZUI", and the "ProductName - productSubName" knob is set to "web component library".

The right-hand configuration panel allows for customizing the header bar's appearance and behavior. It includes the following settings:

- Has searchbar:**
- Show search results:**
- Show iconButton 1:**
- Show iconButton 2:**
- Show iconButton 3:**
- Show NotificationButton:**
- NotificationButton - has-notifications:**
- NotificationButton - emphasis:** default selected
- NotificationButton - size:** m l
- Show UserMenuButton:**
- UserMenuButton - avatar-only:**

At the bottom right of the configuration panel, there are "Copy" and "Reset" buttons.

Web-Components

Web Components - Standards

Wiederverwendbare Komponenten auf Basis von Web-Standards

- **Custom Elements**
 - JavaScript-Basisklasse für Komponenten
 - Life-Cycle
- **Shadow DOM**
 - isolierter DOM-Baum für Komponente
 - CSS-Styling unabhängig von globalen Stylesheets
- HTML templates

```
class MySlider extends HTMLElement {
  constructor() {
    super()

    this.attachShadow({ mode: "open"})
    this.shadowRoot.innerHTML = `
      <div>
        ...
      </div>
    `
  }
}
window.customElements.define("my-slider", MySlider)
```

// Usage

<form>

 <**my-slider** value="50" min="0" max="100"></**my-slider**>

 ...

</form>

Unser Tech-Stack

- Lit
- TypeScript
- SCSS
- rollup
- Azure DevOps (CI/CD)
- Mocka, Playwright, Web-Test-Runner
- Storybook

Warum Web-Components?

Warum Web-Components?

- Web-Standard mit breiter Browser-Unterstützung
- Kein Vendor Lock-in
- Nutzbar in allen SPA-Frameworks*

Warum selber bauen?

Warum selber bauen?

- teilweise recht spezielle Anforderungen und Wünsche an Komponenten
- Versuche mit Customizing existierender Frameworks gescheitert
- mehr Flexibilität und Freiheit bei Entwicklung
- Kontrolle über Weiterentwicklung

Einschränkungen, Herausforderungen

Accessibility

Action

The screenshot shows the Chrome DevTools Accessibility panel. The left pane displays the DOM tree with the following structure:

```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <noscript>You need to enable
    JavaScript to run this app.
    </noscript>
    <div id="root">
      <div class="App">
        <wc-my-button> == $0
          #shadow-root (open)
            "Action"
          </wc-my-button>
        </div>
      </div>
    </div>
  </html>
```

The right pane shows the Accessibility tree with the following structure:

- Accessibility Tree
 - WebArea "Example App"
 - generic
 - generic
 - Ignored
- ARIA Attributes
 - No ARIA attributes
- Computed Properties
 - Accessibility node not exposed
 - Element not interesting for accessibility.

Accessibility

- Web-Components unterstützen "normale" WAI ARIA Mechanismen
- Allerdings:
 - Shadow-DOM kann Probleme machen
 - nicht alle Möglichkeiten von Standard-HTML Elementen stehen zur Verfügung
- Spec: "Accessibility Object Model" aktuell in Arbeit
- Frühzeitig betrachten!

Zusammenspiel mit React

React

- React unterstützt Web-Components im Prinzip
- Allerdings:
 - Teilweise unangenehm in der Benutzung

Problem 1: Attribute vs. Properties

- Web-Components (genau wie Standard HTML Elemente) unterscheiden zwischen "Attribut" und "Property"

Problem 1: Attribute vs. Properties

- Web-Components (genau wie Standard HTML Elemente) unterscheiden zwischen "Attribut" und "Property"

```
<input type="text" value="hallo" />
```

Attribute



Problem 1: Attribute vs. Properties

- Web-Components (genau wie Standard HTML Elemente) unterscheiden zwischen "Attribut" und "Property"

```
<input type="text" value="hallo" />
```

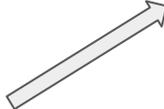
Attribute



```
const input = document.querySelector("input"
```

```
input.value = "hallo"
```

Property



Problem 1: Attribute vs. Properties

- Web-Components (genau wie Standard HTML Elemente) unterscheiden zwischen "Attribut" und "Property"
- React kennt nur "Props"

```
// React
```

```
<input
```

```
    value="hallo"           // attribute or property?
```

```
/>
```

Problem 1: Attribute vs. Properties

- Web-Components (genau wie Standard HTML Elemente) unterscheiden zwischen "Attribut" und "Property"
- React kennt nur "Props"

```
// React
```

```
<input
```

```
    value="hallo"           // attribute or property?
```

```
/>
```

Problem 1: Attribute vs. Properties

- Web-Components (genau wie Standard HTML Elemente) unterscheiden zwischen "Attribut" und "Property"
- React kennt nur "Props"

```
// React
```

```
<my-element
```

```
  value="hallo"           // attribute!
```

>

Problem 1: Attribute vs. Properties

- Web-Components (genau wie Standard HTML Elemente) unterscheiden zwischen "Attribut" und "Property"
- React kennt nur "Props"
- React kann deklarativ nur Attribute setzen, keine Properties.

Problem 2: Events

React hat eigenes Event-System

Web-Components nutzen Standard-HTML-Events

React unterstützt keine deklarativen Listener für Custom-Events aus Web-Components

"Lösung"

- React Refs erlauben imperativen Zugriff auf Instanzen von Web-Components
 - Event-Listener
 - Attribute / Properties explizit setzen
- Allerdings:
 - Nicht der "React-way"
 - Umständlich und unangenehm. React-Devs wollen deklarativ arbeiten, nicht imperativ

Unsere Lösung

- separates Package mit React-Komponenten
- Wrapper-Komponente kümmert sich um Attribute vs. Properties und deklarative Event-Listener
- volle TypeScript-Unterstützung
- React-Komponenten werden automatisch generiert

Form-Elemente

HTML Form-Elemente sind "besonders"

- übergeben Daten an Form
- reagieren auf "reset"
- Validierung
- Interaktion mit anderen Elementen innerhalb eines Forms
 - z.B. `<input type="radio">`

Ideale Lösung: "Form-Associated Custom-Elements"

Spec "Form-Associated Custom-Elements"

- APIs für vollständige Form-Integration
- Allerdings:
 - aktuell nur in Chrome

Mittelmäßige Lösung: "form-data" Event

- Form sendet "form-data"-Event, wenn Formular abgesendet wird
- Elemente können ihre Daten ergänzen
- Unterstützt in Chrome und Firefox
- Allerdings:
 - nicht in Safari

"dirty" Lösung: Stellvertreter Form-Element

- Custom Element erzeugt ein `<input>` Element als Stellvertreter und fügt dieses dem Form hinzu.
- Daten: Custom-Element → Stellvertreter → Form
- Nachteil:
 - unschöner Code
 - korrektes Handling muss sichergestellt werden (z.B. Aufräumen des Stellvertreters ...)

Forms: unsere Lösung

Kombination aus 2 + 3

- wenn Firefox/Chrome → form-data-Event
- wenn Safari → Stellvertreter-Element

Zusatz-Herausforderung: Angular Forms

Angular Forms

Angular hat 2 Forms-Varianten:

- Reactive Forms
- Template-Driven Forms

By default keine Unterstützung für Web-Components

Lösung

Angular bringt Directive `[ngDefaultControl]` mit

Voraussetzung:

- `value` property
- `disabled` property
- `blur` event bei focus lost
- `input` event bei value change
- `value` property muss leeren String korrekt behandeln

Microfrontends

Microfrontends

Idee: Wie Microservices, nur fürs Frontend

- jede App/Microservice hat sein eigenes UI
- AppShell komponiert Apps zu einer Gesamt-Anwendung
- unterschiedliche Microfrontends können unterschiedliche Frameworks nutzen

→ unterschiedliche Microfrontends

→ unterschiedliche Web-Component-Library-Version?

Problem

```
window.customElements.define("my-slider", MySlider)
```

Problem

```
window.customElements.define("my-slider", MySlider)
```

- Tag darf nur 1 mal pro Seite registriert werden
- Klasse darf nur genau einem Tag zugeordnet werden

Ideale Lösung: Scoped Custom Element Registries

Idee: Custom-Elements nicht mehr global registrieren, sondern in selbst definierter "Custom-Registry".

Custom-Registry kann an einen Shadow-DOM angehängt werden

<https://github.com/WICG/webcomponents/blob/gh-pages/proposals/Scoped-Custom-Element-Registries.md>

Ideale Lösung: Scoped Custom Element Registries

Problem:

- Status: Proposal
- noch keine Browser-Unterstützung
- Polyfill existiert, aber "Work in Progress" → production-ready?

Lessons Learned

Abstimmung Styleguide-Team & App-Teams

Styleguide Team



Vorgaben



Library Team



Styleguide Team



Library Team



Product Designer



Product Designer

Styleguide Team



Product Designer

Styleguide Team



Analyst (Library Team)

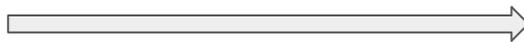


Product Designer

Styleguide Team



Analyst (Library Team)



Library Team



App-Entwicklung \neq Library-Entwicklung

App-Entwicklung != Library-Entwicklung

- Zielgruppe sind Entwickler*innen
- Technische Doku sehr wichtig
- API-Stabilität wichtig
 - Refactorings und Deprecation erzeugen Extra-Aufwand
 - Versionierung
- Anpassbarkeit: Komponenten müssen in vielfältigen Anwendungsfällen funktionieren
- Vorteil: Oft sehr gute Bug-Reports, Zusammenarbeit beim Bugfixing

Community-Management

- public Issue-Board für Bug-Reports und Feature-Wünsche
- wöchentliche "Developer-Sprechstunde"
 - wir zeigen, woran wir gerade arbeiten, welche Bugs wir gerade fixen usw.
 - Leute können ihre Fragen stellen
- öffentliche Sprint-Reviews
- Tech-Support-Kanal in Teams
- transparente Roadmap

Komponentenschnitt

Beispiel "Headerbar"



Eine große Komponente mit vielen Attributen

Pro:

- einfach einbinden und fertig

Con:

- wenig flexibel

Beispiel "Headerbar"



Eine große Komponente mit vielen Attributen

Pro:

- einfach einbinden und fertig

Con:

- wenig flexibel

VS.

Ein Baukasten aus vielen Einzelteilen

Pro:

- sehr flexibel

Con:

- viel Potential für "falsche" Benutzung
- kompliziertere Programmierung

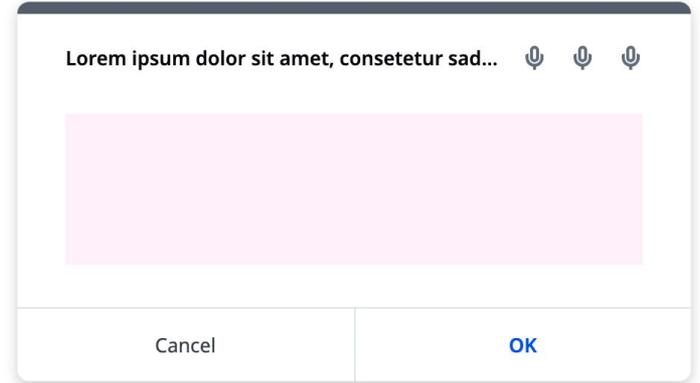
Komponenten-Schnitt: Beispiel 2 "Dialog Buttons"

Erste Version:

Attribute: "cancel-label", "accept-label"

Später: "Wir wollen Buttons disablen"

→ "disable-cancel", "disable-accept"



Komponenten-Schnitt: Beispiel 2 "Dialog Buttons"

Erste Version:

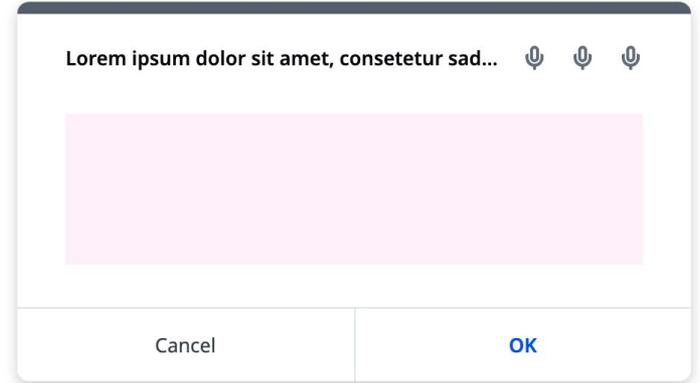
Attribute: "cancel-label", "accept-label"

Später: "Wir wollen Buttons disablen"

→ "disable-cancel", "disable-accept"

Noch Später: "Wir wollen Tooltips auf Buttons"

→ "cancel-tooltip", "accept-tooltip"



Komponenten-Schnitt: Beispiel 2 "Dialog Buttons"

Erste Version:

Attribute: "cancel-label", "accept-label"

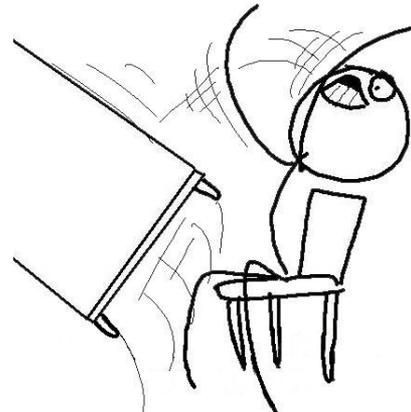
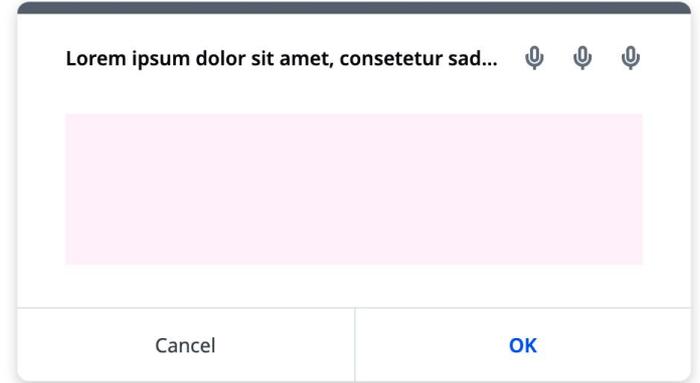
Später: "Wir wollen Buttons disablen"

→ "disable-cancel", "disable-accept"

Noch Später: "Wir wollen Tooltips auf Buttons"

→ "cancel-tooltip", "accept-tooltip"

Noch etwas Später: "Wir wollen 3 Buttons"



Komponenten-Schnitt: Beispiel 2 "Dialog Buttons"

Bessere Lösung:

```
<zui-dialog>  
  <zui-dialog-buttons slot="footer" primary-position="auto">  
    <zui-dialog-button slot="primary">OK</zui-dialog-button>  
    <zui-dialog-button slot="secondary">Cancel</zui-dialog-button>  
    <zui-dialog-button>Some other Action</zui-dialog-button>  
  </zui-dialog-buttons>  
</zui-dialog>
```

Komponenten-Schnitt: Lessons learned

- einfache, "dumme" UI-Basis-Komponenten
- zusätzlich: "Feature-Komponenten" mit Logik und wohl definierten Use-Cases

Ziel: Wenn unsere Feature-Komponente nicht passt, *könnten* App-Devs ihre eigenen Feature-Komponenten aus unseren Basis-Komponenten bauen

Fazit

Fazit

- Web-Components sorgen für Extra-Aufwand (vgl. reine React/Angular Library)
 - Kompatibilität sicherstellen
 - Shadow-DOM erfordert ein paar Umwege
- Teilweise: Einfach aussehende Komponenten sind tatsächlich sehr kompliziert zu bauen (aber nicht nur bei Web Component)
- Nutzung in Apps ist etwas umständlicher
- einige nützliche Zusatz-Specs noch nicht einsatzbereit
- Framework-spezifische Zusatz-Packages in Betracht ziehen
 - aktuell React-Support-Package
 - in Zukunft auch Angular-Support-Package geplant

Fazit

- Framework-agnostischer Einsatz!
- Bei großer Anzahl an Apps bzw. heterogenen SPA-Frameworks lohnen sich Web-Components
- Bei kleinen Apps / einzelnen Apps eventuell besser auf React/Angular/\$framework only setzen
 - Alternativ: SPA direkt mit Web-Components(-Framework) bauen

Manuel Mauky

 @manuel_mauky

Fragen?



<https://www.zeiss.com/digital-innovation>