WEB COMPONENTS ODER REACT

Unterschiede, Gemeinsamkeiten und Anwendungsbeispiele

Bachelor-Arbeit von Simone Ritscher

Bachelorarbeit

WEB COMPONENTS ODER REACT

Unterschiede, Gemeinsamkeiten und Anwendungsbeispiele

von
Simone Ritscher

Studiengang: Mediendesign und digitale Gestaltung
Hochschule: Ravensburg-Weingarten University of Applied Sciences (RWU)

Betreuer I: Prof. Dr. rer. nat. Marius Hofmeister
Betreuer II: Prof. Dr.-Ing. Thorsten Weiss

Bearbeitungszeitraum: Oktober 2020 bis März 2021
Abgabetermin: 18. März 2021

Abstract

In dieser Arbeit werden zwei verschiedene Ansätze beleuchtet, um eine Webanwendung in Komponenten aufzuteilen. Die Frontend Bibliothek React steht dabei dem Web Standard Web Components gegenüber.

Für den Vergleich werden drei eigenständige Komponenten mit beiden Ansätzen entwickelt. Danach werden diese anhand von Kriterien aus Sicht des Komponentenentwicklers, des Komponentennutzers und eines Webseitenbesuchers analysiert. Bei diesem Vergleich werden die Gemeinsamkeiten und Unterschiede, sowie Vor- und Nachteile der beiden Ansätze untersucht.

Aus dem Vergleich ergibt sich, dass die größten Vorteile von Web Components in ihrer Wiederverwendbarkeit in anderen Anwendungen liegen. Sie sind plattformübergreifend nutzbar und bieten eine Abtrennung von Styling und Inhalten. Für den Entwickler können sie jedoch keine vergleichbare Erfahrung zu einem Frontend Framework bieten.

Der Vergleich zeigt dadurch, dass die verschiedenen Stärken von React und Web Components sich ergänzen können. Eine Kombination aus Web Components und React ermöglicht, Komponenten deklarativ zu entwickeln und in vielen Anwendungen wiederzuverwenden. Dies gilt nicht nur für die Kombination mit React, sondern auch für Vue, Angular und spezielle Web Components Frameworks wie LitElement oder Stencil.

Inhaltsverzeichnis

1

Einleitung

1.1 Motivation

1.2 Problemstellung

1.1 Motivation

Das Internet besteht **grundlegend aus drei Technologien: HTML, CSS und Javascript.** HTML, kurz für Hypertext Markup Language, definiert die Inhalte einer Webseite. CSS, kurz für Cascading Style Sheets, legt das Aussehen einer Webseite fest. Javascript erweitert Webseiten um dynamisches Verhalten. Mit diesen drei grundlegenden Technologien sind alle Aspekte einer Webseite – Inhalt, Aussehen und Verhalten – abgedeckt.

Aber natürlich ist die Technologie-Landschaft im Web nicht ganz so simpel. Serverseitige Skriptsprachen wie PHP sind ebenfalls grundlegende Technologien des Webs, sie werden nur vom Server anstatt vom Browser interpretiert (vgl. 1&1 IONOS SE, 2020).
Mit CSS Präprozessoren wie zum Beispiel SASS oder LESS kann die CSS Syntax um verschiedene Funktionalitäten erweitert werden. So können zum Beispiel Funktionen, mathematische Operationen und verschachtelte Selektoren durch CSS Präprozessoren erreicht werden (vgl. CSS preprocessor - MDN Web Docs Glossary: Definitions of Web-related terms | MDN, 2020).

Ähnliche Erweiterungen gibt es für JavaScript. Ein verbreitetes Beispiel ist TypeScript, das JavaScript um Typen erweitert (vgl. Microsoft, o. J.).

Darüber hinaus gibt es tausende Frameworks und Bibliotheken, die Entwicklern Basisfunktionen und Softwaregerüste bereitstellen.
Eines der verbreitetsten JavaScript Frameworks ist jQuery, das unter anderem die Document Object Model (DOM) Manipulation in JavaScript stark vereinfacht (vgl. JQuery Introduction, o. J.).
UI Frameworks wie Bootstrap oder Semantic UI erleichtern die Gestaltung von Webseiten.
Frontend Bibliotheken wie Vue, Angular oder React ermöglichen eine einfachere Entwicklung von dynamischen Frontends.

Im Gegensatz zu den Grundpfeilern des Webs ist die **Zukunft dieser Erweiterungen jedoch immer in gewissem Maße unsicher.**

Das beste Beispiel für dieses schnelle Schicksal ist Adobe Flash. Noch im Jahr 2006 war das Lego Baustein Icon des Flash Players nahezu überall zu sehen. Das

hatte auch gute Gründe: Flash brachte wie keine andere Technologie das interaktive Web voran. Mit Flash konnten Video- und Audioinhalte einfach in eine Seite eingebettet werden. Mit der eigenen Skriptsprache ActionScript, die auf dem ECMAScript Standard basierte, konnten auch interaktivere Erlebnisse wie Spiele oder ganze Webseiten gestaltet werden. Zu dieser Zeit war Flash so weit verbreitet im Web, dass Browser oft zusammen mit dem entsprechenden Plug-in ausgeliefert wurden (vgl. Hoffmann, 2017).

Blickt man 2020 erneut auf die Online-Welt, findet man kaum eine aktuelle Webseite, die noch Flash Elemente verwendet. Seit 2017 ist bekannt, dass Adobe den Flash Player komplett einstellen wird. Die meisten Browser blockieren Flash Player Inhalte schon länger (vgl. Holland, 2017). Wer das Programm noch auf dem Rechner besitzt, wird seit Oktober 2020 zur Deinstallation aufgerufen. Am 31.12.2020 verschwand Flash endgültig von der Bedienoberfläche (vgl. Adobe, 2020). **Was ist in 14 Jahren passiert, dass der Flash Player von einer der wichtigsten Online Technologien zu einem längst vergessenen Dinosaurier wurde?**

Der Anfang vom Ende war das Jahr 2007: Das iPhone wurde veröffentlicht – ohne Flash Player Support (vgl. Hoffmann, 2017). Steve Jobs begründete 2010 die Entscheidung gegen Flash Player mit geringer Performance, Sicherheitsproblemen und wenig Anpassung an die neuen mobilen Verhältnisse.

Nicht zuletzt entschied sich Apple aber gegen Flash, weil es als proprietäres Produkt von Adobe von nur einer Firma abhängig war. Im Gegensatz dazu stand der neue Standard HTML5.
Durch HTML5 wurde es möglich Video- und Audioelemente, moderne Grafiken und Animationen ohne weitere Plug-ins umzusetzen (vgl. Jobs, 2010). HTML5 beendete die Alternativlosigkeit des Flash Players. Große Sicherheitsprobleme führten letztendlich sogar dazu, dass die Abschaltung des Plug-ins eingefordert wurde (vgl. Braun, 2015).

Diese lange Entwicklung war 2006 noch nicht absehbar. Eben diese Gefahr betrifft auch moderne Frameworks. Ein Anwendungsfall, der zuvor nur von Frameworks abgedeckt werden konnte, kann in den Standard integriert werden und dadurch Frameworks überflüssig machen.

Eine Funktion, die zurzeit von mehreren Frameworks geliefert wird, ist ein Komponentenmodell. Die Frontend Bibliotheken Angular, Vue und React bieten alle ihre eigene Variante des gleichen Konzepts: Eine

Anwendung in wiederverwendbare, anpassbare Komponenten aufzuteilen. Zusätzlich liefern sie weitere Funktionen, die sich von Framework zu Framework unterscheiden. Der Kern jeder Anwendung bleiben jedoch die Einzelteile, aus denen sie sich komponieren lässt. Diese Komponenten sollen eigenständig sein, das heißt ihre Struktur, Styling und Logik ist separat von der Anwendung selbst. Dadurch lassen sich deutlich einfacher als in klassischem HTML Abschnitte verschieben und wiederverwenden (vgl. Abbildung 1). HTML Komponenten, die aus vielen `div` Elementen mit speziell benannten Klassen zusammengesetzt sind, nehmen viel Platz ein und sind unübersichtlich. Eindeutig benannte Komponenten machen die Anwendung lesbarer und verständlicher (vgl. Abbildung 1).

Diese angestrebte Aufteilung lässt sich aber nicht nur durch Frameworks realisieren. Der native Ansatz dafür nennt sich Web Components. Web Components setzen sich aus mehreren Standards zusammen, denen lange der weitläufige Browser Support fehlte. Doch seit 2020 unterstützen alle modernen Browser von Google, Mozilla, Apple und Microsoft die Web Components Standards.

```
<div class="dropdown">
  <a class="btn btn-secondary dropdown-toggle" href="#"
     role="button" id="dropdownMenuLink" data-bs-toggle="dropdown"
     aria-expanded="false">
    Dropdown link
  </a>

  <ul class="dropdown-menu" aria-labelledby="dropdownMenuLink">
    <li><a class="dropdown-item" href="#">Action</a></li>
    <li><a class="dropdown-item" href="#">Another action</a></li>
    <li><a class="dropdown-item" href="#">Something else here</a></li>
  </ul>
</div>
```

```
<DropdownButton variant="secondary" id="dropdown-basic-button"
                title="Dropdown button">
  <Dropdown.Item href="#/action-1">Action</Dropdown.Item>
  <Dropdown.Item href="#/action-2">Another action</Dropdown.Item>
  <Dropdown.Item href="#/action-3">Something else</Dropdown.Item>
</DropdownButton>
```

Abbildung 1: Code eines Bootstrap Dropdown Menüs, in grün React Bootstrap

Kann damit ein neues Zeitalter der nativen Komponenten eingeleitet werden? Ein drastischer Umschwung der Technologien ist in nächster Zeit nicht zu erwarten. Auch im Beispiel des Adobe Flash Players dauerte es mehrere Jahre, bis das Plug-in wirklich von den Computern verschwand. Dennoch soll im Rahmen dieser Arbeit schon heute untersucht werden, welche Möglichkeiten der Standard im Vergleich zu einem Frontend Framework bietet.

1.2 Problemstellung

Frontend Frameworks und Web Components haben eine verschiedene Zielsetzung und einen verschiedenen Umfang. Der größte Vergleichspunkt ist die Umsetzung des Komponentenmodells. An dieser Stelle soll der Vergleich hauptsächlich ansetzen.
Dabei sollen Vorteile und Nachteile der verschiedenen Ansätze aus verschiedenen Perspektiven ermittelt werden. Das soll es erleichtern festzustellen, in welchen Situationen welcher Ansatz sinnvoll sein kann.

Besonderer Fokus liegt dabei auf Anwendungsmöglichkeiten für den Web Components Standard. Da dieser noch nicht im gleichen Maße wie zum Beispiel React verbreitet ist, können hier viele Konstellationen getestet werden.

Web Components können unter Anderem alleinstehend, in Kombination mit bestehenden Frontend-Frameworks oder mit speziellen Web Component Frameworks verwendet werden.
Das Ziel ist es einen möglichst weiten Überblick über die Möglichkeiten von Web Components zu erhalten, um eine informierte Prognose über die Zukunft von Komponentenmodellen im Browser aufzustellen.

Dabei sollen folgende Fragen beantwortet werden:

- **Bieten Web Components ähnliche Vorteile wie Frontend Frameworks?**
- **In welchen Use Cases ist es besser auf etablierte Frameworks zu setzen?**
- **In welchen Situationen können Web Components Vorteile bieten?**
- **Welche Funktionen könnte der Standard in der Zukunft erfüllen?**

Um das zu klären, wird eine der verbreiteten Frontend Bibliotheken mit Web Components verglichen. Die Wahl der Frontend Bibliothek fiel aus persönlicher Präferenz und Erfahrung auf React.

Um den Vergleich möglichst fair zu gestalten, werden zuerst die Grundkonzepte und Hintergründe der verschiedenen Ansätze theoretisch beleuchtet. Daraufhin werden feste Kriterien aus verschiedenen Sichten aufgestellt werden. Im praktischen Test werden drei verschiedene Komponenten mit beiden Ansätzen erstellt und anhand der Kriterien verglichen werden. Das Fazit soll darstellen, welche Erkenntnisse aus diesem Test über die Zukunft von Web Components und React gewonnen wurden.

2

Grundlagen

2.1. Was ist React?

2.2. Was sind Web Components?

2.1 Was ist React?

React ist eine JavaScript-Bibliothek für die Frontend Webentwicklung. Das heißt React erleichtert es, Benutzeroberflächen zu entwickeln. Die Library wurde von Facebook entwickelt und wird weiterhin für interne Projekte verwendet (vgl. Design Principles - React, o. J.). Seit 2013 ist React auch als Open-Source-Lösung zugänglich. Das ermöglicht die Nutzung von React für viele verschiedene bekannte Web Anwendungen: Facebook selbst entwickelte Instagram komplett mit React, doch auch bei Webseiten wie Netflix oder Airbnb ist React im Einsatz (vgl. Zeigermann/Hartmann, 2016, S.3).

2.1.1 Konzepte

Um den Aufbau und die Vorteile von React besser zu verstehen, sollen im Folgenden einige Grundkonzepte vorgestellt werden.

Deklarative Programmierung

Das erste Prinzip, das prominent auf der offiziellen Seite von React erwähnt wird, ist Deklarative Programmierung (vgl. React – A JavaScript library for building user interfaces, o. J.).
Deklarative Programmierung ist ein Progammierparadigma, bei dem im Code nur das gewünschte Endergebnis beschrieben werden soll. Das Gegenteil dazu ist die Imperative Programmierung, bei der die einzelnen Schritte, die zum Endergebnis führen, aufgeschrieben werden.

Tyler McGinnis von ui.dev vergleicht dies mit der Suche nach einem bestimmten Ort. Nach dem imperativen Ansatz würde Schritt für Schritt der Weg von einem Ort zum anderen Ort erklärt werden. Die deklarative Erklärung wäre nur eine Adresse. Der Empfänger der Adresse muss in diesem Fall bereits wissen, wie er weiter vorgehen muss, um zum Ziel zu gelangen, beispielsweise indem

er die Adresse in ein GPS System eingibt. **Oft liegt der deklarativen Programmierung also ein imperatives System zu Grunde,** das bereits weiß, welche Schritte für ein Ergebnis notwendig sind (vgl. McGinnis, 2016).

Ein Code-Beispiel für deklarative Programmierung in JavaScript wäre die Verdopplung von Zahlen in einem Array (vgl. Abbildung 2). Mit Hilfe von Standard JavaScript Funktionen kann hier ein großer Teil abstrahiert werden. Die Schritte, um ein neues Array zu erstellen und alle Werte des alten Arrays zu durchlaufen, müssen im deklarativen Ansatz nicht angegeben werden, da sie durch die JavaScript Funktion `map()` abgedeckt werden.

Die Vorteile eines deklarativen Ansatzes sind seine relative Kürze und Einfachheit, wie auch am Beispiel in Abbildung 2 erkennbar ist. Das Programmiermodell wird mehr auf den menschlichen Leser als auf die internen Prozesse der Maschine ausgerichtet. **Dadurch wird der Code einfacher und nachvollziehbarer.**

Wie bereits oben erwähnt liegt deklarativem Code jedoch oft eine imperative Abstraktion zu Grunde. Bedient man sich dieser Abstraktion, wird die Kontrolle über Teile des Codes abgegeben; dadurch kann die Funktion nicht auf jeden einzelnen Anwendungsfall perfekt optimiert werden. Gerade bei komplexen Anwendungen kann deklarativer Code durch das hohe Maß an Abstraktion mehr Einarbeitung erfordern als imperativer Code. Oft werden deshalb imperative und deklarative Ansätze gemischt (vgl. Deklarative Programmierung, 2020; McGinnis, 2016).

```
function double(numbers){
     newArr = [ ];
     for (i = 0; i < numbers.length; i++){
          newArr[i] = numbers[i]*2;
     }
     return newArr;
}
```

```
function double(numbers){
     return numbers.map((value)=> value*2);
}
```

Abbildung 2: Verdopplung eines Array in imperativer (oben) und deklarativer (unten) Weise

Natürlich ist Code, der mit Hilfe von React geschrieben wurde, nicht automatisch deklarativ. JavaScript Code kann sowohl imperativ als auch deklarativ sein (vgl. Abbildung 2). **React verwendet jedoch ein besonderes Software Muster, das sogenannte Virtual DOM.** Dieses ermöglicht es, die Entwicklung von Oberflächen und deren Synchronisierung – die grundlegende Aufgabe von React – deklarativ zu gestalten. Mit Hilfe des Virtual DOM muss innerhalb des Codes nur angegeben werden, wie eine App zu einem bestimmten Zeitpunkt aussehen soll. Der Entwickler muss sich nicht mit den Operationen befassen, die nötig sind, um eine Darstellung zu einer anderen zu transformieren. So kann die Benutzeroberfläche deklarativ, auf das Ergebnis ausgerichtet, beschrieben werden. React führt automatisch die Schritte aus, die die Anwendung von einem Zustand in den anderen versetzten. Das entspricht der imperativen Abstraktion.

Um die notwendigen Operationen von einem Zustand zum anderen zu ermitteln, kommt der Virtual DOM zum Einsatz. Im Virtual DOM wird immer eine ideale Version der Oberfläche gespeichert, wie sie laut Code sein soll. Kommt eine neue Version des Virtual DOM hinzu, ausgelöst durch einen State Change, wird diese mit der vorherigen Version verglichen und daraus die nötigen Operationen abgeleitet, um zum neuen idealen Zustand zu gelangen (vgl. Virtual DOM and Internals, o. J.; Zeigermann/Hartmann, 2016, S.23f.).

Um den idealen Zustand der Oberfläche in React zu beschreiben, wird meist die JavaScript Erweiterung JSX (JavaScript XML) verwendet. Diese ermöglicht es XML-artigen Code direkt in JavaScript zu schreiben. Im Vergleich zu einem Methoden-Aufruf in klassischem JavaScript wird die Lesbarkeit und deklarative Programmierung dadurch weiter gefördert (vgl. Abbildung 3).
Diese Vermischung von Logik und Markup wird jedoch teilweise kritisch betrachtet, da sie mit bestehenden Best Practices, also bewährten Vorgehensweisen, bricht (vgl. Introducing JSX, o. J.; Zeigermann/Hartmann, 2016, S.5f. und S.21f.).

Komponenten

Ein weiteres wichtiges Konzept von React ist die Aufteilung einer Anwendung in Komponenten (engl. Components). Die Aufteilung in eigenständige, wiederverwendbare Komponenten hat, wie bereits in der Einleitung der Arbeit erwähnt, viele Vorteile. **Anwendungen können dadurch schneller und flexibler aus ihren Einzelteilen komponiert werden** (vgl. Components and Props, o. J.).

```
class HelloMessage extends React.Component {
  render() {
    return React.createElement(
      "div",
      null,
      "Hello ",
      this.props.name
    );
  }
}

ReactDOM.render(
React.createElement(HelloMessage,
{ name: "Taylor" }),
document.getElementById('hello-example'));
```

```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="Taylor" />,
  document.getElementById('hello-example')
);
```

Abbildung 3: Vergleich von React mit Methodenaufrufen (oben) und React mit JSX (unten)

React empfiehlt in seiner Dokumentation, zur Aufteilung einer Anwendung in Komponenten nach dem **Single Responsibility Prinzip** vorzugehen. Dieses Prinzip besagt, dass eine Komponente idealerweise nur einen Zweck oder eine Aufgabe erfüllen sollte. Wird die Komponente größer und erfüllt mehrere Aufgaben, muss sie in mehrere kleinere Komponenten aufgeteilt werden (vgl. Thinking in React, o. J.).

Komponenten können in React entweder als Funktion oder als Klasse dargestellt werden.

Eine Funktionen Komponente darf dabei nur ein Objekt `props` (kurz für engl. „Properties", deutsch „Eigenschaften") als Übergabewert akzeptieren und muss ein React Element zurückgeben. Um eine Klasse als Komponente darzustellen, muss die Klasse von `React.Component` erben und eine `render()` Methode implementieren, die ein React Element zurückgibt. Diese Darstellungen sind aus Sicht von React gleichwertig (vgl. Components and Props, o. J.).

Der Datenfluss von Eltern- zu Kind-Komponenten funktioniert über die bereits oben genannten Properties. Diese werden wie Attribute in der Eltern-Komponente übergeben und können in der Kind-Komponente, egal ob diese als Funktion oder als Klasse dargestellt wird, über das Objekt `props` oder `this.props` aufgerufen werden. Eine Komponente darf aber nie das `props` Objekt selbst verändern (vgl. Components and Props, o. J.).

Außer den Properties kann jede Komponente auch einen State besitzen. Dieser ist privat und kann nur von der Komponente selbst kontrolliert werden. Er wird im Konstruktor einer Klassenkomponente als Objekt mit beliebigen Eigenschaften definiert. Um diesen State zu verändern, darf nur die Funktion `this.setState()` verwendet werden. Dadurch kann React auf jede State Veränderung reagieren und wenn nötig DOM Elemente neu laden, wodurch die Interaktivität von Komponenten gewährleistet wird (vgl. State and Lifecycle, o. J.).

Wie in klassischem HTML werden auch in React Events wie `onClick` oder `onChange` verwendet, um User Input zu überwachen. **React greift dabei nicht auf die Browser-Events zurück, sondern liefert eigene Synthetic-Events.** Dadurch soll eine einheitliche Entwicklung unabhängig vom Browser ermöglicht werden. Die Unterschiede zu den Browser-Events sind jedoch minimal.

In React werden Events außerdem oft genutzt, um die Kommunikation von Kind zu Eltern-Komponenten zu ermöglichen. Dies ist notwendig, weil React grundsätzlich nur den Datenfluss in eine Richtung erlaubt. Der Zustand der Anwendung, der für alle Komponenten gleich sein soll, sollte daher in einer der obersten Komponenten festgelegt sein. Diese Komponente wird dann durch Events über Änderungen informiert. Um die-

se Benachrichtigung zu ermöglichen kann eine Eltern-Komponente eine Funktion als Property an eine Kind-Komponente übergeben. Im Falle einer Änderung, auf die die Eltern-Komponente reagieren sollte, zum Beispiel einer Änderung in einem Input-Feld oder einem Klick auf einen Button, kann ein Event die Funktion aus der Eltern Komponente über `props` (oder `this.props`) aufrufen und so die übergeordnete Komponente benachrichtigen (vgl. Lifting State Up, o. J.; Handling Events, o. J.; Zeigermann/Hartmann, 2016, S.128ff.).

Learn once, Write Anywhere

Für größere Anwendungen oder verschiedene Use Cases kann React mit weiteren Technologien kombiniert werden.

Als reine Frontend Bibliothek soll dabei keine Einschränkung in Bezug auf weitere Technologien stattfinden.

Erweiterungen, die häufig mit React zusammen genutzt werden, sind zum Beispiel Node.js für Server Anbindung und Paket Management, React Router für das Routing von URLs zu Komponenten, Webpack und Babel für Build Management, Redux oder MobX für State Management. Zusätzlich gibt es noch viele weitere Pakete, die speziell für React entwickelt wurden (vgl. Zeigermann/Hartmann, 2016, S.8ff.; Create a New App, o. J.).

Diese freie Wahl von weiteren Technologien wird von React als eines der positiven Merkmale der Bibliothek herausgestellt (vgl. React – A JavaScript library for building user interfaces, o. J.).

```
class Component extends React.Component {
  render() {
    return <p>This is a class component </p>
  }
}
```

```
function Component(props){
  return <p>This is a Function Component</p>
}
```

Abbildung 4: Vergleich einer React Klassenkomponente (oben) und einer Funktionskomponente (unten)

2.1.2 Geschichte

React wurde 2011 vom Software Ingenieur Jordan Walke für Facebook entwickelt. Facebook hatte zu dieser Zeit Probleme mit den Facebook Ads Anwendungen, da diese immer komplexer wurden. **Deshalb wurde ein Prototyp entwickelt, der den Code vorhersehbarer und die Wartung des Codes einfacher machen sollte. Aus diesem Prototyp entstand die Library React.**

Als Facebook 2012 Instagram aufkaufte, wurde React, das bis dahin nur innerhalb von Facebook verwendet worden war, aus Facebooks Technologie-Setup losgelöst. Dadurch konnte instagram.com mit React umgesetzt werden, ohne im Hintergrund die gleichen Technologien wie Facebook zu verwenden (vgl. Facebook Developers, 2015).

2013 wurde die Bibliothek als Open-Source Lösung veröffentlicht (vgl. Zeigermann/Hartmann, 2016, S.3). React ist damit Teil von Facebook Open Source. React wird auch weiterhin intern von Facebook genutzt. Die Weiterentwicklung richtet sich dabei oft nach den Bedürfnissen des Facebook Teams (vgl. Design Principles, o. J.). Diese Herangehensweise, die eigenen Produkte zu verwenden, um sie zu verbessern, wird Dogfooding genannt (vgl. Cambridge University Press, o. J.).
Laut React garantiert Dogfooding das Fortbestehen der Bibliothek und eine zielgerichtete Vision für die Zukunft (vgl. Design Principles, o. J.).

2.1.3 Nutzung

Die Ursprünge und Weiterentwicklung von React sind eindeutig abhängig Facebook. Doch auch außerhalb dieser Anwendung hat React **eine große Open Source Community** (vgl. Zeigermann/Hartmann, 2016, S.7f.). Im offiziellen React Tag „reactjs“ auf Stack Overflow finden sich 275,142 Fragen (vgl. Newest „reactjs“ Questions, 2021). Auf Github wird React in 5.399.098 Repositories verwendet (vgl. facebook, 2021).

Auch im Vergleich mit anderen Web Technologien ist React beliebt. Bei einer Stack Overflow Umfrage zu den verwendeten Web Technologien von 2019 gaben 31.3% an, dass sie React verwenden. Einen höheren Wert erzielt nur die JavaScript Bibliothek jQuery (vgl. Stack Overflow, 2019).

Im Online Ranking hotframeworks.com, das Statistiken auf Stack Overflow und Github von verschiedenen Frameworks vergleicht, belegt React sogar den ersten Platz (vgl. Web framework rankings | HotFrameworks, o. J.). **Bei diesen Vergleichen wird React oft in eine Reihe mit den JavaScript Frameworks Angular und Vue gestellt.**
Dabei stellt sich die Frage, ob React als Framework bezeichnet werden kann. Eine endgültige Antwort dazu ist schwer zu finden. Gerade bei Vergleichen werden Libraries und Frameworks oft mehr nach Gefühl oder individuellem Funktionsumfang unterschieden. **Ein großes Problem dabei ist, dass es keine eindeutige Definition gibt, wann eine Bibliothek zu einem Framework wird.** Ein mögliches Kriterium: Eine Library ist eine Sammlung an Funktionen und Klassen, die der Entwickler selbst aufruft. Ein Framework hingegen kann vorgesehene Funktionen auch selbstständig aufrufen (vgl. Domin, 2018).

In React kann jede Komponente sogenannte Lifecycle Methoden implementieren. Die Methode `componentDidMount()` wird zum Beispiel aufgerufen, sobald die betreffende Komponente im DOM gerendert wurde. Diese Methoden müssen nicht explizit vom Entwickler aufgerufen werden.
Offiziell wird React ausschließlich als Bibliothek bezeichnet. Dies wird als Feature ausgelegt, da React flexibel und anpassbar sei (vgl. Byron et al., 2017). Für den Rest dieser Arbeit wird React deshalb auch nur als Bibliothek oder Library bezeichnet. Gerade weil die Trennung der Begriffe schwierig ist, ist es am einfachsten die Wünsche des Entwicklerteams zu berücksichtigen.

2.2 Was sind Web Components?

Web Components sind eine Sammlung von verschiedenen Standards, mit denen gekapselte und wiederverwendbare Komponenten nur mit Hilfe von HTML, JavaScript und CSS gebaut werden können. Damit ist es möglich eine Komponentenarchitektur ohne ein Framework oder eine Bibliothek umzusetzen. Web Components können allerdings auch mit Frameworks oder Libraries kombiniert werden. So können einzelne Komponenten plattformübergreifend genutzt werden, ohne sich in die Eigenheiten aller Frameworks einzuarbeiten (vgl. Rauber, 2020a).

2.2.1 Konzepte

Web Components setzen sich aus den folgenden Standards zusammen:

Custom Elements

Custom Elements sind das Herzstück der wiederverwendbaren Komponenten. **Dieser Standard ermöglicht es mit Hilfe von JavaScript neue unbekannte HTML-Tags zu registrieren und deren Verhalten zu definieren.** So kann die eigene Komponente in HTML exakt wie jeder andere Standard HTML Tag verwendet werden. Um ein neues Element zu definieren wird das globale Objekt `customElements` verwendet. Mit der Methode `define(tagName, constructor, options)` wird ein neues Element definiert (vgl. Abbildung 5). Der Name des Elements muss dabei immer einen Bindestrich (-) enthalten. Dadurch ist das Element als benutzerdefiniert gekennzeichnet. Außerdem soll verhindert werden, dass zukünftige neue HTML Standard Elemente denselben Namen wie Komponenten tragen.

Selbstschließende HTML-Tags sind dabei nicht möglich; alle Custom Elements müssen mit einem End-Tag geschlossen werden.

Um das Verhalten einer Komponente zu bestimmen, sind Custom Element Reactions hilfreich. Diese sind mit den Lifecycle Methoden von Frontend Frameworks vergleichbar: So wird die Funktion `connectedCallback()` immer aufgerufen, wenn die Komponente zum DOM hinzugefügt wird. Die Reaction `attributeChangedCallback(attrName, oldValue, newValue)` wird aufgerufen, wenn ein Attribut der Komponente geändert wird.

Diese Funktion hat die Besonderheit, dass sie nicht für alle Attribute aufgerufen wird. Die gewünschten Attribute müssen zuerst dem Array `observedAttributes` hinzugefügt werden. Dies soll die Performance verbessern, indem keine unnötigen Callbacks entstehen.

Der Standard Custom Elements befindet sich in der zweiten Version. Die erste Version, genannt v0, war bereits vom Browser Chrome implementiert worden. Dort verwendete man zur Erstellung von neuen Elementen den Ausdruck `document.registerElement()`. Dieser Standard wurde durch die v1, wie die aktuelle zweite Version genannt wird, ersetzt (vgl. Bidelman, 2020a).

```
<head>
    <script>
        window.customElements.define(
        'my-component' Tag-Name
        class extends HTMLElement { ... });
        Komponenten Klasse
    </script>
</head>
<main>
    <my-component></my-component> Komponente
</main>
```

Abbildung 5: Registrieren eines Custom Elements und Verwendung in HTML

Shadow DOM

Das Shadow DOM ist dafür zuständig, dass Komponenten unabhängig und abgekapselt vom Rest einer Anwendung sind. Wird einem Element ein Shadow DOM angehängt, entsteht ein neuer DOM-Baum der unabhängig vom Haupt DOM ist. Dadurch können im Shadow DOM weitere HTML-Elemente, Styling und Logik hinzugefügt werden, ohne dass dies für Entwickler direkt sichtbar ist. Alles, was innerhalb des Shadow DOMs definiert wird, hat keine Auswirkungen auf den Haupt DOM. Im Gegenzug haben CSS-Regeln und JavaScript Operationen aus dem Dokumenten DOM keine Auswirkungen auf das lokale Shadow DOM.

In einer Web Component werden die tatsächlichen Kind-Elemente des Custom Elements als Light DOM im Gegenteil zum Shadow DOM bezeichnet.

Um einem Custom Element einen Shadow DOM hinzuzufügen muss lediglich im Konstruktor der Custom Element Klasse die Funktion `this.attachShadow(…)` aufgerufen werden. Dabei gibt es einen offenen und einen geschlossen Modus, der mit `{mode: 'open'}` oder `{mode: 'closed'}` beim Initialisieren festgelegt werden kann.

Der geschlossene Modus ist jedoch für die Entwicklung von Web Components fast immer zu restriktiv und bietet keine echten Sicherheitsvorteile. Deshalb kann dieser größtenteils vernachlässigt werden.

Um benutzerdefinierten Content in den Shadow DOM zu integrieren, können `<slot>` Tags im Template des DOMs verwendet werden. Auf den HTML `<template>` Tag wird im nächsten Unterpunkt genauer eingegangen. **Mit einem Slot kann eine Stelle markiert werden, an dem Content aus dem Light DOM gerendert werden soll.**
Um Inhalte direkt einem von mehreren Slot Elementen zuzuordnen, können mit dem `name` Attribut Slots benannt werden. Im Light DOM wird bei dem Element, das im Slot platziert werden soll, der entsprechende Slot Name mit dem Attribut `slot="..."` angegeben.
Werden keine Namen angegeben, werden die Slots der Reihe nach mit dem Inhalt des Light DOM gefüllt (vgl. Bidelman, 2020b).

Elemente des Light-DOMs die in einen Slot eingefügt wurden, werden als slotted bezeichnet. Elemente im Light-DOM, die theoretisch einen Platz in einem Slot einnehmen könnten, als slotable (vgl. Using templates and slots, 2020).

Sein Styling erhält das Shadow DOM wie

in normalem HTML über ein `<style>` Element oder ein externes Stylesheet. Für das Shadow DOM gibt es aber zusätzlich besondere Pseudo-Selektoren.

Ein Beispiel für diese Selektoren ist `:host`, der das Element selektiert, an den das Shadow DOM angehängt ist. Da das Host-Element sich im Haupt DOM befindet, wird das Styling des `:host` Selektors jedoch – falls vorhanden – von äußerem CSS überschrieben (vgl. Bidelman, 2020b).

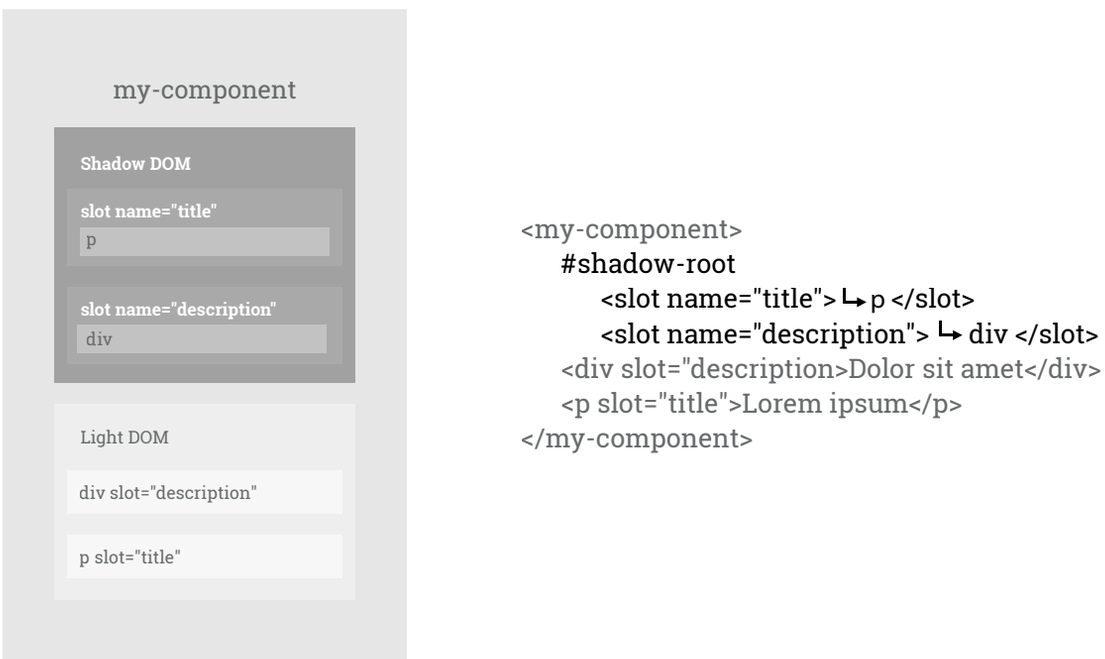

Abbildung 6: Schematische Darstellung Shadow DOM und Light DOM mit Slots

HTML Templates

HTML Templates sind wie Shadow DOM und Custom Elements nicht grundsätzlich Teil von Web Komponenten. **Sie können aber dabei helfen, Inhalte in den Shadow DOM zu laden.**

Template Tags können an jeder beliebigen Stelle in einem Dokument stehen. Ihr Inhalt wird jedoch nicht gerendert, bis er aktiviert wird.

Um ein Template zu aktivieren, wird es geklont und als Kind an anderer Stelle, zum Beispiel an den Shadow DOM, angehängt. **Der Inhalt des Shadow DOMs sollte aus Performance Gründen auf diese Weise und nicht durch das Setzen der `innerHTML` Property festgelegt werden.**

Wird die `innerHTML` Property des Shadow DOM gesetzt, wird der Inhalt für jede Instanz des Custom Elements erneut geparset, also von Text in ein DOM übersetzt. Beim Verwenden eines Templates geschieht das jedoch nur einmal (vgl. Bidelman, 2020a).

Ein weiterer Vorteil des `<template>` Tags ist die Verwendung von Slots, die bereits im Punkt Shadow DOM erläutert wurde. HTML Templates werden seit 2013 von allen großen Browsern unterstützt (vgl. Bidelman, 2013).

2.2.2 Geschichte

Die Idee von Web Components wurde zuerst auf der Fronteers Konferenz 2011 von Alex Russell vorgestellt.

Fronteers ist ein Zusammenschluss von niederländischen Frontend-Webentwicklern, der jährlich eine Konferenz zum Thema Frontend-Webentwicklung veranstaltet.
Der Vortragende Alex Russel ist ein Software Ingenieur bei Google. Dort arbeitet er aktuell (Stand 2021) am Browser Chrome, der Rendering Engine Blink . Früher entwickelte er die Standards für Web Components mit. Außerdem war er von 2013 bis 2019 Mitglied der W3C Technical Architecture Group, die sich mit Prinzipien und Problemen in der Web Architektur befasst (vgl. Russell, 2020; The W3C Technical Architecture Group (TAG), o. J.).

In seinem Vortrag erklärt er die Möglichkeiten, Spezifikationen der Web Standards als Webentwickler mitzubeeinflussen.
Als Beispiel wird die Einführung von neuen Elementen in HTML5 genannt, die unter Anderem aus häufig verwendeten CSS Klassen abgeleitet wurden.
Web Components sollen eine neue Möglichkeit dieser Einflussnahme darstellen.
Anstatt die gewünschten Elemente in JavaScript Dateien zu beschreiben, die von außen keinen Aufschluss über den Zweck oder die Funktion des Elements geben, sollen Custom Elemente verwendet werden, die auch im DOM erkennbar sind. Damit könne ein Signal an Browser Betreiber gesendet werden, welche Elemente von Entwicklern gewünscht sind (vgl. Russell, 2011).

Google hat seitdem die Entwicklung und Verwendung von Web Components maßgeblich vorangetrieben. Offiziellen Browser-Support von anderen Browsern gab es lange Zeit nicht. Dies sollte durch Polyfills ausgeglichen werden. Tatsächlich wurde die Nutzung aber erst durch weitläufigeren Browser-Support möglich (vgl. Page, 2015).

Erst seit 2020 werden die aktuellen Web Components Standard von allen großen Browsern (Edge, Firefox, Chrome und Safari) mit der Veröffentlichung des neuen auf Chromium basierten Edge Browsers unterstützt.

In Firefox sind alle Standards seit 2018 verfügbar. In Safari sind Web Components seit 2017 nutzbar; hier fehlt lediglich der Support für Custom Elements, die Browser-

eigene Elemente erweitern können. Diese Erweiterung ist eigentlich darauf ausgelegt, Accessibility Features von bestehenden Elementen zu übernehmen. Sobald jedoch ein Shadow-DOM zu einem Element hinzugefügt wird, gehen diese Features verloren. Dadurch sind die Anwendungsmöglichkeiten sehr begrenzt, weshalb sich der fehlende Support wenig auf die Verwendung von Web Components auswirkt (vgl. Page, 2015; Deveria, o. J. a).

Seit den Anfängen von Web Components hat sich auch der Umfang der Spezifikationen verändert. Während heute nur noch Custom Elements, Shadow DOM und HTML Templates zu den Standards gezählt werden, war vor einigen Jahren noch der Standard HTML Imports angedacht. Dieser sollte es ermöglichen, ein HTML Dokument und alle darin importierten Dateien in ein anderes zu importieren. Hier haben sich jedoch JavaScript Module Imports als Alternative durchgesetzt. Selbst Browser, die bereits HTML Imports unterstützt hatten, haben diese Spezifikation wieder entfernt (vgl. Rauber, 2020a; Deveria, o. J. a).

2.2.3 Nutzung

Auf webcomponents.org werden Components und ganze Component Collections geteilt. Die Seite bietet auch einen Startpunkt, um Web Components kennen zu lernen.
Für Web Components muss oft derselbe Code mehrmals geschrieben werden. **Um diesen sogenannten „Boilerplate Code“ zu reduzieren, können verschiedene Bibliotheken für Web Components eingesetzt werden** (vgl. webcomponents.org, o. J.).

Eine beliebte Bibliothek ist die Polymer Library. Sie wurde von Google entwickelt und sollte die Entwicklung von Web Components erleichtern. Die Polymer Library ist in der dritten Version verfügbar. Weitere Versionen wird es jedoch nicht geben, da Google 2018 die Nachfolger Bibliothek LitElement vorgestellt hat.

LitElement basiert auf lit-html, einer JavaScript Bibliothek, die ebenfalls von Google entwickelt wurde.
Damit können HTML Templates mit dynamischen Daten gerendert werden (vgl. The Polymer Project, o. J.; Lit-html, o. J.). LitElement bietet zusätzlich weitere Erleichterungen in der Arbeit mit Web Components.

Eine weitere Bibliothek für die Entwicklung von Web Components ist Stencil. Stencil bezeichnet sich selbst als Compiler für schnelle Web Components, die den Webstandards entsprechen. Auch Ansätze von React finden sich in Stencil wieder. Stencil arbeitet mit einem Virtual DOM, um Änderungsoperationen gering zu halten. Das asynchrone Rendering von Stencil ist direkt von Reacts Technologie Fiber inspiriert. Außerdem wird auch in Stencil die JavaScript Erweiterung JSX verwendet, um Templates zu erstellen. Stencil Komponenten werden laut eigener Aussage von Firmen wie Apple, Amazon und Porsche eingesetzt (vgl. Stencil, o. J.; Stencil – Introduction, o. J.; Rauber, 2020b).

Weitere Bibliotheken, die auf webcomponents.org weniger oft vertreten sind, sind z.B. Hybrids, Slim.js oder Skate.js.

3

Testanwendung

3.1. Architektur

3.2 Unterschiedliche Entwicklertools

3.3 Vergleichskriterien

3.1 Architektur

Die Testanwendung soll eine Webanwendung für ein Urlaubsziel sein. Dafür soll zuerst eine statische Seite mit Hilfe von HTML und CSS entwickelt werden, in der die Komponenten getestet werden. Für die Anwendung sollen drei Komponenten entwickelt werden. Die erste Komponente stellt eine Karte dar, die die Anfahrt zu einem festgelegten Ziel von einem frei wählbaren Startpunkt darstellt. Eine weitere Komponente zeigt Events in einem Kalender an. Die letzte Komponente stellt Wetterinformationen an einer kleinen Auswahl von Tagen dar. Bei all diesen Komponenten sollte es von außen möglich sein, ihr Styling an die Seite anzupassen.

3.1.1 Anfahrt Komponente

Für diese Komponente ist ein User Input notwendig. Der Besucher sollte eine Adresse oder einen Ort eingeben können. Dieser Ort muss in Geokoordinaten umgewandelt werden. Das kann mit der Geocoding API von HERE Technologies erreicht werden. Mit Hilfe der Koordinaten soll dann eine Karte mit einer markierten Route dargestellt werden. Dafür werden die JavaScript API und die Routing API von HERE Technologies verwendet. Die HERE APIs werden verwendet, weil diese alle notwendigen Funktionen bieten, verständlich und bei einer kleinen Zahl an Aufrufen gratis verwendbar sind.

Ein Entwickler, der die Komponente in seiner Anwendung verwenden will, muss daher aber auch die API Keys ändern können. Auch der Endpunkt muss außerhalb der Komponente anpassbar sein. Außerdem soll eine Schnittstelle geboten werden, um Kartenkontrollelemente ein- und auszublenden und von der vereinfachten Kartendarstellung auf Satellitendarstellung zu wechseln.

3.1.2 Kalender Komponente

Für jedes Event im Kalender sollen die Informationen Name, Datum und optional Beschreibung und Link zu weiteren Infos bereitgestellt werden. Diese Informationen würden in einem echten Use Case wahrscheinlich aus einer Datenbank auf einem Server stammen. Für diese Komponente soll deshalb ein REST-Zugriff ausgeführt werden, der ein JSON-Objekt zurückliefert, das die Daten der Events enthält. Im Kalender werden dann Name und Datum angezeigt. Klickt der User auf ein Event, öffnet sich ein Overlay, dass die lange Beschreibung und den Link zu weiteren Event Infos anzeigt. Es soll zusätzlich die Möglichkeiten einer Tages-, Wochen-, oder Monatsansicht geben. Der erste Tag der Woche soll ebenfalls anpassbar sein.

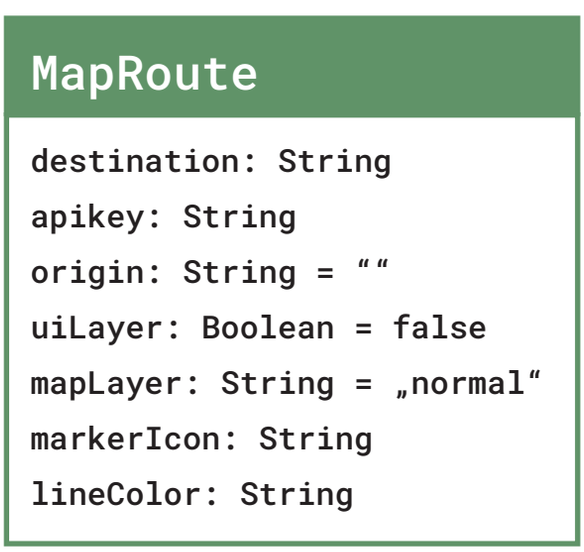

Abbildung 7: Schaubild der benötigten Daten für die Anfahrt Komponente

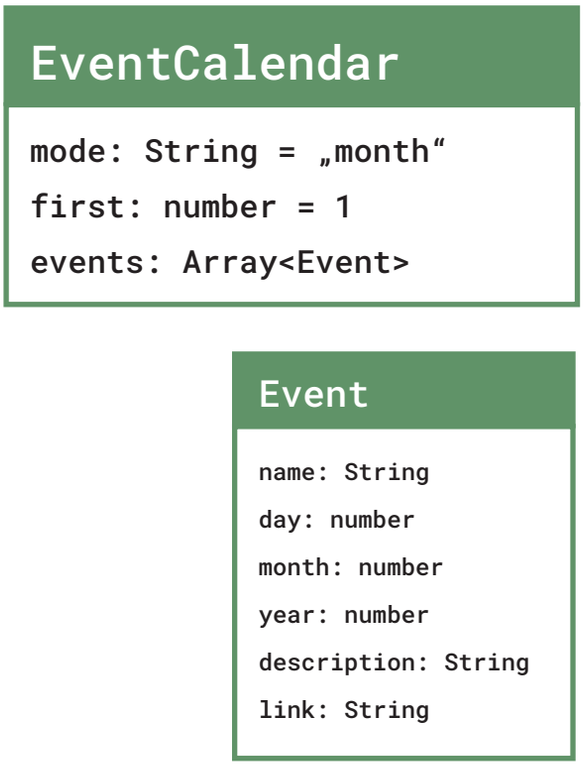

Abbildung 8: Schaubild der benötigten Daten für die Kalender Komponente

3.1.3 Wetter Komponente

Für die Wetterkomponente soll der User zwischen verschiedenen Vorhersage Zeiträumen wechseln können: Heute, 3-Tages-Vorhersage, 5-Tages-Vorhersage und 8-Tages-Vorhersage. Die Vorhersage wird von der Open Weather Map One Call API bereitgestellt. Diese API ist bis zu einer gewissen Abrufzahl kostenfrei, aber auch hier muss ein Entwickler die Möglichkeit haben, von außen die API Keys einzugeben. Um passende Daten zu erhalten, müssen außerdem Standpunkt, Sprache und Einheiten anpassbar sein. Ein Entwickler soll außerdem verschiedene Wetterdaten ein- und ausblenden können. Hier soll er aus den Daten Temperatur, Regen- und Schneevorhersage, Wind und Sonnenaufgang und -untergang auswählen können. Ein Wetter Icon und eine kurze Beschreibung sollen immer angezeigt werden. Zusätzlich soll es möglich sein, ein eigenes Icon-Set für die Wetter-Icons zu verwenden.

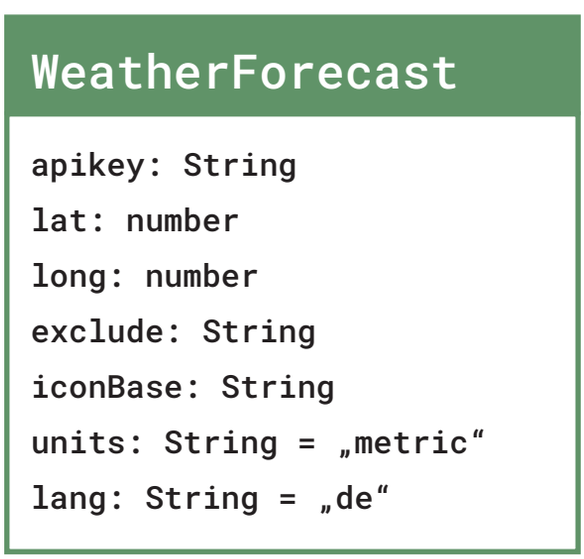

Abbildung 10: Schaubild der benötigten Daten für die Wetter Komponente

3.2 Unterschiedliche Entwicklertools

Für einen schnellen Start in die Entwicklung der React Komponenten wird **Create-React-App** verwendet. Dieses Tool wurde von Facebook entwickelt, um Entwicklern die anfängliche Konfiguration der Entwicklungsumgebung abzunehmen (vgl. Create React App, o. J.).

Außer den dafür benötigen Modulen sollen – soweit möglich – keine weiteren Packages verwendet werden, um den Entwicklungsaufwand für eine komplett selbst-programmierte Komponente fair vergleichen zu können.

Für die Entwicklung mit Web Components wird auf eine Library erst einmal verzichtet, um tatsächlich einen Unterschied zwischen der Library React und dem nativen Ansatz Web Components ziehen zu können. In **Kapitel 5.5 Weitere Möglichkeiten für Web Components** soll aber auf die Entwicklung mit Stencil und LitElement eingegangen werden.

Das HTML-Template, in das die Komponenten eingefügt werden, wird mit Hilfe von Bootstrap erstellt. Für React würde sich die Nutzung von React Bootstrap anbieten, das speziell für React entwickelt wurde. Im Rahmen dieser Arbeit wird jedoch darauf verzichtet, da die beiden Versionen nicht exakt deckungsgleich sind. Für einen fairen Vergleich soll mit demselben Styling gearbeitet werden.

3.3 Vergleichskriterien

Die Ansätze von React und Web Components unterscheiden sich deutlich in den verwendeten Methoden und der Zielsetzung. Um diese verschiedenen Ansätze dennoch zu vergleichen, werden acht verschiedene Kriterien angewendet, aufgeteilt in die Gruppen, denen sie den größten Nutzen bringen.

3.3.1 Komponenten-Entwickler Kriterien

Die erste Person, die sich mit jeder Komponente befasst, ist der Entwickler. Für diesen ist es besonders wichtig, dass der Komponenten Code simpel und verständlich ist, denn er muss die Komponente später verstehen und warten. Um diese Anforderungen vergleichen zu können, werden drei Kriterien angesetzt.

Wie viel „Boilerplate Code“ muss für jede Komponente geschrieben werden?

Code, der für jede Anwendung wiederholt werden muss, wird Boilerplate Code genannt. Dieser sollte so gut wie möglich vermieden werden, um dem Entwickler repetitive Schreibarbeit abzunehmen. Außerdem wird der Quellcode so kürzer und übersichtlicher. Der Boilerplate Code soll ausgemacht werden, indem alle drei Komponenten in den beiden Ansätzen miteinander verglichen werden. Die Zeilen Code, die bis auf wenige Änderungen wiederholt werden müssen, sind der Boilerplate Code.

Wie einfach können Anpassungen vom Entwickler unterstützt werden?

Wenn eine Komponente auch von anderen Entwicklern wiederverwendbar sein soll, ist es wichtig, dass sie auch ohne genau Kenntnisse der internen Logik an verschiedene Bedürfnisse anpassbar ist. Die Schnittstellen für diese Anpassung muss der Komponenten-Entwickler liefern. Dabei soll verglichen werden, wie viele Zeilen Code allein für die Programmierung der Schnittstellen geschrieben werden muss.

Welcher Code weist mehr deklarative Merkmale auf?

Deklarative Programmierung ist eines der Grundkonzepte von React. Die Vorteile von deklarativer Programmierung gegenüber imperativer Programmierung wurden bereits in Punkt 2.1.1 erklärt. Für dieses Kriterium soll verglichen werden, welcher Code im direkten Vergleich öfter deklarative Merkmale aufweist.

3.3.2 Komponenten-Nutzer Kriterien

Gerade für Web Components ist ein häufiger Anwendungsfall das Nutzen einer Komponente, ohne den Code oder sogar den Entwickler zu kennen. Dafür wird zum Beispiel die Plattform webcomponents.org zum Veröffentlichen von Komponenten genutzt. Aber auch bei der Arbeit im Team kann es vorkommen, dass Komponenten von einem Entwickler in eine Anwendung integriert werden müssen, der die Komponente nicht selbst geschrieben hat.

Die Komponente sollte also auch von einem Entwickler, der kein JavaScript Experte ist, eingesetzt werden können. Dadurch ist es einfacher, die Komponente wiederzuverwenden und verschiedene Kenntnisgrade im Team auszugleichen.

Damit das möglich ist, sollten folgende Kriterien möglichst gut erfüllt sein.

Wie einfach sind Anpassungen für einen Entwickler, der die Komponenten verwendet, umzusetzen?

Ein Komponenten-Nutzer sollte Komponenten von außen verändern können, ohne sich in die zugrunde liegende Programmierung einarbeiten zu müssen. Gut wäre es hier, wenn die Schnittstellen auch für Nicht-JavaScript Programmierer verständlich aufgebaut sind.

Wie einfach können Komponenten-Styles vor allgemeinen Styles geschützt werden?

Gerade beim Styling kommt es oft zu Problemen, wenn Code außerhalb des eigentlichen Einsatzbereichs wiederverwendet werden soll. Wichtige CSS-Werte werden überschrieben und müssen angepasst werden (vgl. Stoll, 2020). Bei gekapselten Komponenten ist dies besonders unerwünscht. Um hier vergleichen zu können, werden die Komponenten in die gleiche `index.html` mit dem gleichen allgemeinen Style-Sheet eingebunden. Dann wird bewertet, wie viele Styles überschrieben werden und wie viel Aufwand es ist, das Styling wieder anzupassen.

Wie einfach lässt sich eine Komponente in ein anderes Framework einbauen?

Plattform-übergreifende Komponenten haben im Web einen großen Vorteil. Web Technologien können sich schnell verändern, Teams können ihr Framework wechseln oder auch mit mehreren Frameworks arbeiten (vgl. Springer, 2020). Dann ist es gut, wenn Komponenten, die bereits programmiert wurden, nicht für jedes andere Framework stark angepasst werden müssen und auch in neuere Frameworks integrierbar sind.

Um dies zu testen, sollen die Komponenten aus beiden Ansätzen in das Framework Vue integriert werden. Danach wird der Aufwand für eventuelle Anpassungen verglichen. Die Wahl fiel auf das Framework Vue, da es – mit React und Angular – zu den verbreitetesten Frontend Libraries angehört. Die nächste Wahl für diesen Test wäre Angular gewesen. Dieses Framework ist jedoch ein Projekt von Google. Da Google schon maßgeblich am Voranschreiten des Web Component Standards beteiligt ist, soll hier mit Vue ein unabhängigeres Framework bevorzugt werden. Außerdem soll überprüft werden, wie gut Web Components in React integriert werden können.

3.3.3 Besucher Kriterien

Für den Webseiten Besucher sollte es idealerweise nicht auffallen, welche Technologie grundlegend verwendet wird. Hier ist vor allem wichtig, dass der Nutzer keine Einschränkungen durch die verwendete Technologie erfährt.

Welche Seite lädt schneller?

Geschwindigkeit ist ein wichtiger Teil der User Experience (vgl. Pavic et al., 2020). Deshalb sollte eine möglichst schnelle Geschwindigkeit bevorzugt werden.

Die Webseiten Performance wird mit der Google Page Speed Insights API analysiert. Dieser Test basiert auf Lighthouse, einem automatisierten Tool, welches die Geschwindigkeit einer Webseite und verschiedene Verbesserungsmöglichkeiten testet (vgl. About PageSpeed Insights, 2018; Lighthouse | Tools for Web Developers, 2020).

Werden Accessibility Tools unterstützt?

Für Nutzer mit Einschränkungen ist der Support von Screen Readern oder Anpassungen durch User Stylesheets sehr wichtig. Nicht alle dieser Funktionen werden automatisch unterstützt. So gibt es zum Beispiel Probleme mit User Stylesheets, wenn Elemente im Shadow DOM nicht erreicht werden können.

Hier soll verglichen werden, welche barrierefreien Funktionen von den Komponenten unterstützt werden. Falls manche Funktionen nicht unterstützt werden, soll der Aufwand für die Nachrüstung verglichen werden.

4

Entwicklung

4.1 Entwicklung des HTML Template

Die beiden Ansätze React und Web Components sollen innerhalb des gleichen HTML Templates getestet werden.

Das Template wurde mit Bootstrap entwickelt, um die vorgefertigten Funktionen für ein Raster-System und responsive Navigation zu nutzen. Bootstrap ist eine HTML, CSS und JavaScript Library, die ursprünglich bei Twitter entstand (vgl. Otto et al., o. J.). Über 3 Millionen Webseiten verwenden mittlerweile Bootstrap (vgl. SimilarTech, 2020). Gerade weil Bootstrap so populär ist und über 10.000 Zeilen CSS Code mitbringt (Version 4.5.3), ist es ein perfektes Framework für das HTML Template: Es zeigt einen realistischen Use Case, mit dem sich leicht die Kapselung von CSS Styles testen lässt.

Das Template besteht aus einer Navigation, einem Header Bild, drei Sektionen, die später die Komponenten enthalten sollen, und einem Footer.

4.2 Entwicklung mit Web Components

4.2.1 Setup

Für das Setup der Web Components müssen nur im HTML Template die drei Skripte für die jeweiligen Komponenten eingebunden werden. Diese können dann wie normale HTML Tags in den dafür vorgesehenen Sektionen eingefügt werden.

Zum Testen wird das Node Modul http-server verwendet. Dies ist hier sogar notwendig, da einige Browser lokal geladene JavaScript Dateien unterdrücken. Das würde verhindern, dass die Web Komponenten überhaupt geladen werden.
Für die Entwicklung wurde die Google Developer Doku unter https://developers.google.com/web/fundamentals/web-components und die MDN Web Dokumentation für Benutzerdefinierte Events und das Javascript Date-Objekt verwendet. Außerdem wurden Informationen von W3Schools und den jeweiligen API Dokumentation (HERE Maps und Open Weather Map API) verwendet.

Die Komponenten können unter https://web-components-bachelor.netlify.app/ getestet werden.

4.2.2 Anfahrt Komponente

Die notwendigen Skripte von HERE werden als Module direkt in der JavaScript Datei importiert. Im Konstruktor der Komponente wird die ShadowRoot erstellt. Darauf wird der Inhalt des Templates dem Shadow DOM hinzugefügt.
Das Template enthält das eigene Styling, das Stylesheet für die HERE Maps, so wie die Container für die Karte und die Zeitanzeige. **Das Initialisieren des Shadow DOM sollte immer im Konstruktor gemacht werden** (vgl. Custom Element Best Practices | Web Fundamentals, 2019).

Darauf werden die `observedAttributes` gesetzt. Die überwachten Attribute in diesem Fall sind nur `origin` und `destination`. Alle weiteren Attribute sollen nur vom Entwickler in der HTML Datei gesetzt werden können, daher ist hier keine Überwachung während der Laufzeit notwendig.

Der Key für die HERE API wird ebenfalls direkt als Attribut übergeben. Da ab gewissen Abrufzahlen für die Nutzung der API bezahlt werden muss, könnte dies zunächst wie ein Sicherheitsrisiko wirken. Da der API-Key bei clientseitigem Rendering auch in den URLs der Geocode Anfragen herauszulesen ist, wäre eine andere Variante aber nicht sicherer. Eine Möglichkeit den API Key trotzdem zu schützen, ist es, diesen in den HERE Einstellungen auf bestimmte Seiten zu begrenzen.

Im `connectedCallback()` wird ein `origin` Attribut erstellt, falls keines existiert. Das gewährleistet die Funktion, auch wenn kein Standard-Startpunkt festgelegt wurde. Die HERE Plattform wird initialisiert, um direkt die Koordinaten des Endpunkts anhand der festgelegten Adresse zu erhalten.
Existieren die Koordinaten des Endpunkts wird dieser direkt als Mittelpunkt der Karte festgelegt. Wurde kein Endpunkt angegeben wird nur eine Standardkarte erstellt. Außerdem wird der `EventListener` für das Map-Input Event festgelegt.

In der `createMap()` Funktion wird die Karte initialisiert. Dabei werden die Optionen, die in den Attributen festgelegt wurden, wie zum Beispiel die Kartenansicht oder die Ein- und Ausblendung der UI Elemente, berücksichtigt. In der `createMap()`Funktion wird die Funktion `drawRoute()` aufgerufen. Wenn Start- und Endpunkt definiert

sind, wird eine Route gezeichnet. Ist nur der Endpunkt definiert, wird an dieser Stelle ein Marker gezeichnet. Ändert sich das Attribut `origin` oder `destination`, wird das `attributeChangedCallback(name, oldValue, newValue)` aufgerufen. Dort wird zuerst die alte Route entfernt, bevor `drawRoute()` erneut aufgerufen wird, um eine neue Route zu erstellen.

Um die alte Route zu entferne wird die Funktion `removeObjectById()` verwendet. Diese entfernt alle Marker oder Linien von der Karte, deren ID mit dem Argument übereinstimmt. Als ID wird während der Funktion `drawRoute()` für alle Objekte der aktuelle Startpunkt festgelegt. Ändert sich der Startpunkt, werden alle Objekte, deren ID mit dem `oldValue` des Startpunkts übereinstimmt, entfernt. Ändert sich der Endpunkt werden alle Objekte mit dem aktuellen Startpunkt als ID entfernt.

Das zweite Custom Element Map-Input, das für die Eingabe des Startpunkts verantwortlich ist, hat nur das Attribut `origin`, das von der Map-Route Komponente übergeben wird. Auch hier wird im Konstruktor eine ShadowRoot erstellt, die in diesem Fall ein `<div>` Element mit einem `<input>` Feld und einem Button enthält.
Im `connectedCallback()` werden hier Event-Listener für den Button und das Input-Feld festgelegt. Auf Klick des Button oder beim Drücken von Enter im Input-Feld wird ein CustomEvent ausgelöst. Hier ist es sehr wichtig, dass sowohl die Optionen `bubbles` und `composed` als `true` angegeben werden. Sonst kann das Event nicht außerhalb des Shadow DOM erkannt werden. Wird dieses Event in der Komponente Map-Route empfangen, wird das Attribut `origin` auf den Wert des Input-Felds gesetzt. **Während der Entwicklung wurde außerdem deutlich, wie wichtig eine ausführliche Dokumentation für den Nutzer der Komponente ist.**

Bei der Verwendung der HERE API gibt es die Möglichkeit, ein eigenes Icon als PNG oder SVG Datei zur Darstellung als Marker zu verwenden. Dies wird jedoch ab einer Größe von 256px nicht mehr dargestellt. Bis die entsprechende Stelle in der Dokumentation gefunden wurde, sorgte dies für viel Verwirrung. Solche Probleme sollten bei der Dokumentation der Komponente unbedingt vermieden werden. Hier muss auf jeden Fall vermerkt werden, welche Attribute verwendet werden können, welche Werte möglich sind, welche Attribute notwendig sind und welche Auswirkungen die Attribute haben.

Der vollständige Code mit Dokumentation kann auf Github unter dem Link: https://github.com/ssimone175/map-route eingesehen werden.

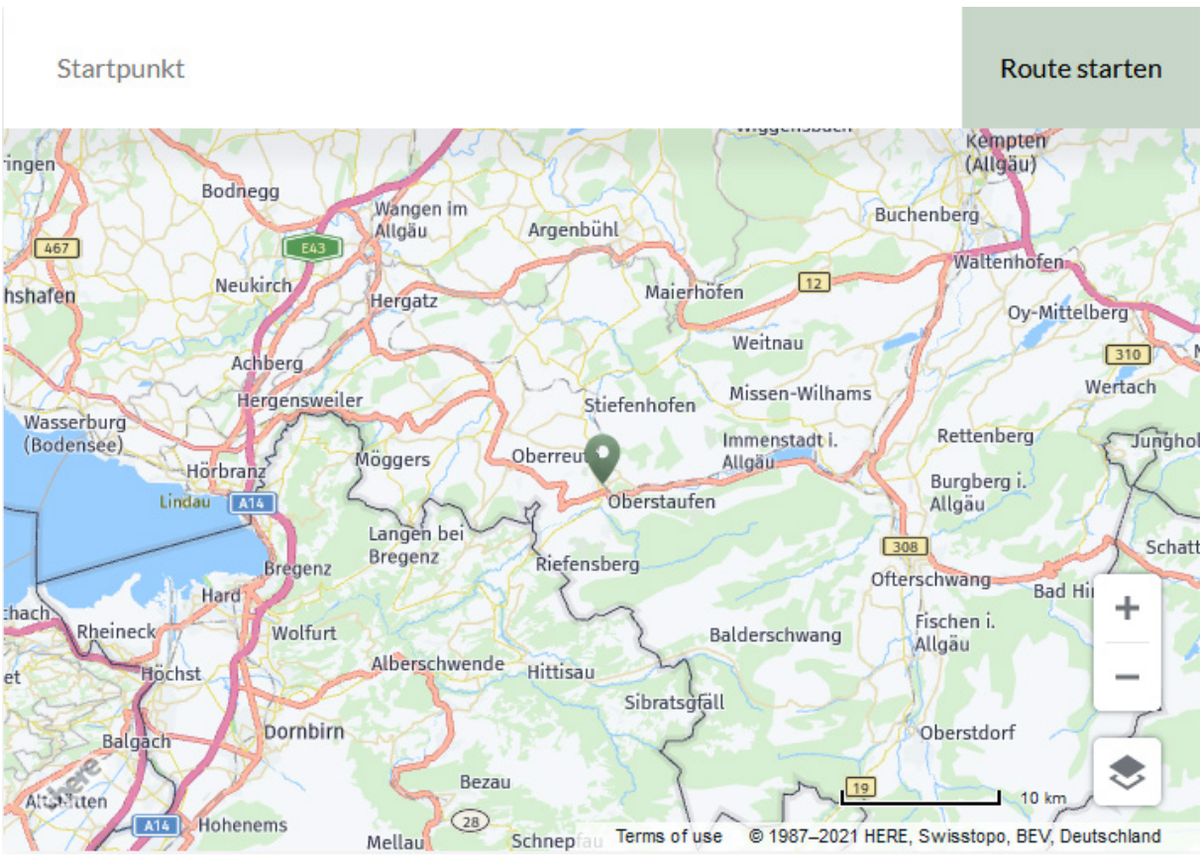

Abbildung 11: Anfahrt Web Component

4.2.3 Kalender Komponente

Die Kalender Komponente kann in zwei Sektionen aufgeteilt werden: Der Kalender Header enthält zwei Buttons, um das Kalenderblatt zu wechseln, und den Namen des aktuell angezeigten Monats und Jahres. Darunter wird eine Reihe an Daten und Events dargestellt.

Für diese Komponente wurde ein Template verwendet, das den größtenteils unveränderlichen Header darstellt. Die Daten und Events werden durch DOM Operationen erst später hinzugefügt.

Im `connectedCallback()` erhält der Header seine Funktion. Nachdem das aktuelle Datum als Start für den Kalender generiert wurde, wird der Titel des Headers auf den aktuellen Monat und das aktuelle Jahr gesetzt. Danach wird für die beiden Buttons die Funktion `onCalendarChange()` als `onclick` Event festgelegt. In dieser Funktion wird das Startdatum, auf Basis dessen die Kalenderdaten generiert werden, um entweder einen Tag, eine Woche oder einen Monat verändert.
Je nachdem welcher Button aktiviert wurde, ist diese Veränderung positiv oder negativ. Danach werden die Daten und der Titel des Kalenders neu generiert.

Im `connectedCallback()` werden außerdem die notwendigen Klassen des Kalender-Elements dynamisch gesetzt und das Events-Array mit Hilfe der `fetch` API initialisiert. Sowohl im `connectedCallback()` also auch in der `onCalendarChange()` Funktion werden die Daten mit Hilfe der `createDays()` Funktion generiert.

In `createDays()` wird ein Array erstellt, das alle anzuzeigenden Daten in der richtigen Reihenfolge enthält, abhängig von dem gewählten Modus – Tagesansicht, Wochenansicht oder Monatsansicht – und dem ersten Tag der Woche – Montag oder Sonntag. Dieses Array wird daraufhin durchlaufen. Für jeden Tag, dessen Datum nicht mit einem Event-Datum übereinstimmt, wird ein `<div>` Element erzeugt, dass die benötigten CSS-Klassen für das richtige Styling und das Datum als Inhalt erhält.
Stimmt das Datum mit einem Event-Datum überein, wird ein Custom-Element mit dem Namen `<calendar-event>` erzeugt. Diesem Element werden die CSS-Klassen und der Wochentag des Events als Attribute übergeben. Diese sind für das Styling notwendig.

Die Inhalte der `<calender-event>` Komponente werden mit Hilfe von Slots gesetzt. Dem Light-DOM des `<calendar-event>` Custom Elements werden das Datum und der Name des Events als `<p>` Elemente hinzugefügt. Für die zusätzlichen Informationen, die erst nach einem Klick sichtbar werden, wird ein `<div>` Element übergeben, das mindestens ein Custom-Element des Typs `<event-info>` enthält. Auch die Inhalte für das `<event-info>` Element werden mit Hilfe von Slots geladen. Auf die Vor- und Nachteile von Slots wird in der Wetter Komponente und der Zusammenfassung genauer eingegangen.
Probleme bereitete das Styling der Komponenten. Um die Event-Komponente in Abhängigkeit der Kalender-Optionen zu stylen wurde zuerst die CSS-Pseudo-Klasse `:host-context` verwendet. Dadurch kann dem Shadow-DOM einer Komponente abhängig von den Klassen einer Eltern-Komponente CSS Eigenschaften hinzugefügt werden, obwohl er vor anderen Styles geschützt ist.
Diese Pseudo-Klasse wird im Moment aber weder von Firefox noch von Safari unterstützt (vgl. Deveria, o. J. b).

Deshalb mussten alle CSS Klassen die das gesamte Kalender-Styling betreffen, für die `<calender-event>` Elemente noch einmal gesetzt werden.

Der vollständige Code mit Dokumentation kann auf Github unter dem Link: https://github.com/ssimone175/event-calendar eingesehen werden.

Abbildung 12: Kalender Web Component

4.2.4 Wetter Komponente

Für die Wetterkomponente werden zwei Komponenten erstellt: `<weather-forecast>` und `<daily-forecast>`. Das Template der Weather-Forecast Komponente besteht aus 4 Buttons und einem leeren `<div>` Element, in das später die `<daily-forecast>` Elemente geladen werden. Im Konstruktor wird dieses Template dem ShadowRoot hinzugefügt.

Im `connectedCallback()` wird mit dem Befehl `fetch()` die Open Weather Map One Call API aufgerufen. Wenn dieser Aufruf beendet ist, wird das zurückgelieferte Array mit 8 Tagesvorhersagen in eine Klassen Variable gespeichert. Danach kann die Funktion `updateWeather()` aufgerufen werden, die die `<daily-forecast>` Elemente aktualisiert. Außerdem werden im `connectedCallback()` die Start-Anzahl der anzuzeigenden Tagesvorhersagen und die `onclick` Event-Listener für die Buttons gesetzt.

In der Funktion `updateWeather()` wird geprüft, wie viele `<daily-forecast>` Elemente im Moment im dafür vorgesehenen `<div>` Element vorhanden sind.
Sind es mehr als die aktuelle Zahl der anzuzeigenden Elemente werden die letzten Elemente gelöscht, bis die Zahlen übereinstimmen.
Sind weniger Elemente als vorgegeben vorhanden, wird die Funktion `loadWeather()` aufgerufen.

In dieser Funktion wird in einer For-Schleife für jedes Element des Response Arrays, das nicht bereits existiert und dessen Index kleiner als die Zahl der anzuzeigenden Tage ist, ein neues `<daily-forecast>` Element erstellt. Die Wetterinformationen werden als Property an das Custom Element weitergegeben.

Als letzter Schritt in der Funktion `updateWeather()` wird überprüft, ob Elemente mit dem `id` Attribut „show“ und mit der Klasse „chosen“ existieren. Mit Klick auf eines der `<daily-forecast>` Elemente wird diesem die Klasse „chosen“ hinzugefügt, das aktuelle Show-Element wird entfernt und mit einem Klon des Chosen-Element ersetzt. Bevor ein Element geklickt wurde oder falls das Chosen-Element entfernt wurde, wird standardmäßig das erste Element ausgewählt.

Das Custom Element `<daily-forecast>` hat ein eigenes Template, das dem Shadow-DOM dieses Elements hinzugefügt wird. In der `connectedCallback()` Methode werden die Informationen, die übergeben wurden in das Template eingefügt. Sollte eine Information durch das `exclude` Attribut ausgeschlossen sein, wird der entsprechende Platzhalter im Template gelöscht.

Für die Darstellung der Vorhersagen werden mehrere Icons verwendet. Diese werden von einem anderen Server geladen. Um ein eigenes Icon-Set verwenden zu können, muss der Nutzer der Komponente die Icons mit den gleichen Namen benennen und das Attribut `iconBase` verändern.

Weil die Basis URL der Icons variabel sein muss, werden die `src` Attribute der Icons erst zusammen mit den anderen Informationen über DOM Operationen verändert. Hier wird ein großer Nachteil in der Entwicklung mit Web Components deutlich: Best Practice ist es, das Template einer Komponente außerhalb dieser zu definieren. So wird sichergestellt, dass das Template nur einmal geparst wird und somit Performance Vorteile ausgenutzt werden (vgl. Bidelman, 2020a). **Dadurch gibt es aber nur zwei Möglichkeiten die Inhalte des Templates dynamisch zu setzen: Slots und DOM Operationen.**

Slots haben den Vorteil, dass Informationen automatisch an eine bestimmte Stelle gesetzt werden. Es sind also keine weiteren Operationen nötig. **Für die Kommunikation zwischen Komponenten haben Slots aber auch Nachteile.** Es muss ein Element erstellt werden, dann Informationen und ein Slot Attribut hinzugefügt werden, um es dann dem Light DOM der Komponente anzuhängen.

Alternativ dazu kann eine Information als ein Attribut gesetzt werden. Für den Use-Case dieser Komponente, in dem sich eigentlich nur der vordere Teil einer URL ändern soll, aber das für viele Elemente, sind **Slots zusätzlich umständlich, da mit ihnen nur ganze Elemente und keine Attribute in Templates ersetzt werden können**. Das heißt für jedes Icon müsste ein eigener Slot gefüllt werden.

Deshalb wurde hier auf Slots verzichtet und stattdessen die Basis URL als Attribut übergeben und darauf die `src` Attribute der entsprechenden Icons angepasst.

Der vollständige Code mit Dokumentation und Standard-Icons kann auf Github unter dem Link: https://github.com/ssimone175/weather-forecast eingesehen werden.

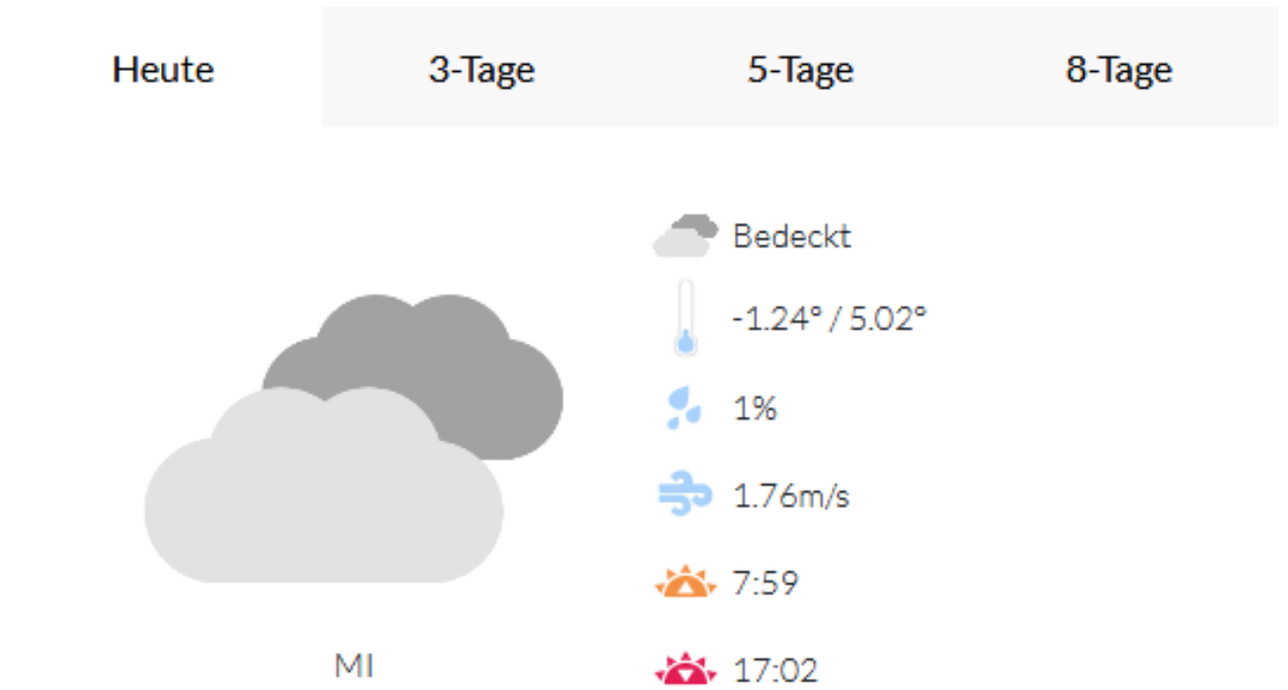

Abbildung 13: Wetter Web Component

4.3 Entwicklung mit React

4.3.1 Setup

Für das initiale Setup wird mit Create React App eine Beispiel App erstellt.
Create React App stellt bereits eine Build-Pipeline mit Webpack und Babel bereit (vgl. Create a New App, o. J.). Auch notwendige Dateien für eine Progressive Web App werden mitgeliefert, wie die Datei `manifest.json`. In dieser Arbeit werden diese jedoch nicht weiter beachtet, da der Fokus auf einer klassischen Webanwendung liegt.

Die generierte `index.html` Datei wird durch das zuvor erstellte HTML Template ersetzt. Das `<main>` Element, in das die Komponenten später eingebettet werden, wird durch ein `<div>` Tag mit der `id` „app“ ersetzt. Dieser wird später durch React mit der App Komponente ersetzt, die den Startpunkt für die weiteren Komponenten darstellt. In der App Komponente wird die `<main>` Sektion wie im Template eingefügt und an der passenden Stelle die entsprechenden Komponenten aufgerufen.

Für die Entwicklung wurde die React Dokumentation unter https://reactjs.org/docs/getting-started.html, W3Schools für das JavaScript Date Objekt und die jeweilige API Dokumentation (HERE Maps und Open Weather Map API) verwendet.

Der vollständige Code aller React Komponenten kann auf Github unter dem Link https://github.com/ssimone175/react-components eingesehen werden.

Die React Komponenten können auf https://react-components-bachelor.netlify.app/ getestet werden.

4.3.2 Anfahrt Komponente

Eine Herausforderung, um diese Komponente in React zu entwickeln, war die HERE Maps API und deren Dokumentation. Diese bezieht sich nahezu ausschließlich auf pures JavaScript, das nicht ohne kleine Änderungen auf React übertragen werden kann.

Um ein Map-Objekt zu erstellen, muss zum Beispiel ein Element angegeben werden, in dem die Karte dargestellt werden kann. Elemente aus dem `return` Statement der `render()` Funktion einer React Komponente können aber nicht wie in JavaScript mit Methoden wie `getElementById()` ausgewählt werden.
Hier muss stattdessen eine Referenz im `return` Statement gesetzt werden, mit der während anderer Lifecycle Methoden auf das Element zugegriffen werden kann. Eine große Hilfe war schließlich ein Artikel, der die Grundlagen der Einbindung in React erklärte (vgl. Add a HERE Web Map into your React application, 2020).

Außerdem wird das Node Package here-js-api verwendet, dass es ermöglicht, die notwendigen HERE Skripte mit dem Befehl `import` einzubinden (vgl. Hacker, 2020). Die von HERE empfohlene Alternative wäre das einfügen der Skripte in der `index.html` gewesen (vgl. Add a HERE Web Map into your React application, 2020). In diesem Fall hätte ein Nutzer nicht nur wissen müssen, wie eine React Komponente grundsätzlich eingebunden wird, sondern auch, welche Skripte für das Funktionieren dieser speziellen Komponente notwendig sind. Das bietet zusätzliches Fehlerpotenzial. Deshalb wurde hier das Installieren eines einzelnen Packages bevorzugt.

Die Formular Werte, die für die Eingabe des Startpunktes notwendig sind, wurden nicht, wie in React üblich, über den State der Komponente verwaltet. **Stattdessen wurde mit sogenannten Uncontrolled Components gearbeitet** (vgl. Uncontrolled Components, o. J.). Grund dafür war, dass nicht mit jedem Tastendruck ein Rerender ausgelöst werden sollte, sondern erst beim Abschicken des Formulars. Deshalb wird hier stattdessen eine unkontrollierte Komponente verwendet. Das ist möglich in dem das Input Feld, dessen Wert verwendet werden soll, eine Referenz erhält, auf die beim Einreichen des Formulars zugegriffen werden kann, um den aktuellen Wert zu erhalten. Erst beim Submit wird dann tatsächlich der State verändert.

Ein großer Unterschied zur Entwicklung mit Web Components ist der Umgang mit den Properties einer Komponente. Dieser sieht auf den ersten Blick sehr ähnlich aus. Beim Aufrufen werden die Properties wie HTML-Attribute übergeben. In React kann dann mit dem Objekt `this.props` auf diese zugegriffen werden. **Während es sich bei Web Components empfiehlt Attribute und Properties synchron zu halten, kann dies in React ausgeklammert werden** (vgl. Custom Element Best Practices, 2019). Das erspart im Vergleich zur Web Components Anfahrt Komponente sehr viel Getter und Setter Code.

Eine weitere Abweichung ist das Löschen von Kartenelementen. Wird eine neue Route gezeichnet, muss zuerst die alte wieder von der Karte entfernt werden. Bei den Web Components war dies mit der Reaction `attributeChangedCallback(name, oldValue, newValue)` möglich. Für jede Route wurde als ID der angegebene Startpunkt verwendet. Sobald sich dieser verändert, kann im `attributeChangedCallback(...)` auf den alten Wert zugegriffen werden und somit alle Marker der alten Route gelöscht werden. Das hat den Vorteil, dass andere Map Objekte nicht betroffen sind. In React muss dafür der alte Wert in einer Variablen gespeichert und bei jeder Änderung angepasst werden.

In React ist das Styling der Komponente nicht gekapselt vom Styling des Templates. Für das Input-Feld mussten die Bootstrap Styles width, `height`, `padding`, `border` und `box-shadow` überschrieben werden, um die gleiche Darstellung zu erhalten wie bei der Web Component. Eine Alternative wäre es, einen spezifischeren Klassennamen zu verwenden, um diesem Konflikt auszuweichen. Dieser ist dann aber möglicherweise weniger präzise.

4.3.3 Kalender Komponente

In React besteht die Kalender Komponente aus drei Komponenten, die ineinander verschachtelt sind. Die oberste Komponente `Calendar` ist dabei die wichtigste.

In der `render()` Funktion der Calendar Komponente wird abhängig vom Ansichtsmodus und des ersten Wochentags ein Array generiert, das alle anzuzeigenden Daten in der richtigen Reihenfolge enthält. Dieser Schritt ist nahezu identisch wie beim Web Components Ansatz. Einen großen Unterschied macht aber die Nutzung von JSX. Anstatt das Array zu durchlaufen und für jedes Datum ein neues Element im DOM hinzuzufügen, kann in React das Array an Datum-Objekten mit Hilfe der `Array.map()` Funktion in ein Array an Day Komponenten umgewandelt werden. Dieses Array wird dann im `return` Statement der `render()` Funktion an der richtigen Stelle eingesetzt.

In der Day Komponente wird überprüft, welches Styling das Datum erhalten soll, und ob es sich um ein Event handelt. Ist dies der Fall wird eine Event Komponente zurückgegeben, ansonsten wird ein einfaches `<div>` mit dem entsprechenden Styling erstellt.

Bei der Erstellung dieser Komponente bot React einige große Vorteile. **Mit Hilfe von JSX kann nicht nur wie oben erwähnt einfach ein Array mit JSX-Elementen dargestellt werden, sondern auch Objekte einfach wie Attribute an Kind Komponenten übergeben.** Das verkürzt den Aufruf der Event Komponente im Vergleich zu Web Components enorm. Außerdem kann auch das Array an Events, das der Komponenten-Nutzer liefern muss, einfach an die Komponente übergeben werden. Dies ist mit Web Components nicht möglich, da Attribute nur primitive Datentypen übergeben können. Deshalb kann hier nur der Link für eine `fetch()` Anfrage weitergegeben werden.

Schwierigkeiten bereitete hingegen das Styling. Einige Styles von anderen Komponenten wurden ungewollt auf die Kalender Komponente angewendet und mussten überschrieben werden, zum Beispiel der Style für Button-Elemente aus der Anfahrt-Komponente. Ein weiterer Nachteil der ungewollten Styles ist, dass sie der Wiederverwendbarkeit der Komponente im Weg stehen. **Wird die Komponente in einer Anwendung entwickelt, die eigene Styles definiert oder Frameworks, wie zum Beispiel Bootstrap, einsetzt, die ebenfalls**

CSS Styles festlegen, kann es sein, dass diese in einer anderen Anwendung fehlen. Als Beispiel kann hier der Style des Kalender Headers gewählt werden. Im Template wird für den Header bereits die Schriftfarbe Weiß festgelegt, deshalb ist dies im Kalender Header nicht mehr nötig. Wird die Komponente aber in eine anderen Anwendung kopiert, muss dieser Style nachträglich ergänzt werden.

Da diese Komponente vor dem Web Components Gegenstück entwickelt wurde, war hier auch die Einarbeitung in JavaScript Datum Objekte notwendig. Einerseits werden damit sehr hilfreiche Funktionen wie zum Beispiel `getDay()`, das den Wochentag eines Datums zurückgibt, geliefert. Andererseits sind die vielen Funktionen teilweise auch verwirrend. Ein Beispiel ist die sehr ähnliche Benennung von Funktionen bei `getDay()` und `getDate()`. Auch die Rückgabewerte sind nicht immer selbsterklärend. So gibt `getDate()` und `getFullYear()` den Tag des Monats und die Jahreszahl in der menschlich gewohnten Form zurück. Die Funktion `getMonth()` hingegen gibt die Monatszahlen verschoben wieder, da sie für den Januar mit der Zahl 0 startet. Dadurch erhält zum Beispiel der Monat November die Zahl 10 und nicht 11 (vgl. JavaScript Date Reference, o. J.). Diese kleinen Besonderheiten mussten für die Erstellung der Kalenderdaten berücksichtigt werden und hatten so an dieser Stelle einen großen Einfluss auf die Entwicklungsarbeit.

Das Komponenten Modell und die JSX Erweiterung von React waren für die Entwicklung dieser Komponente sehr hilfreich. Einzig die Kapselung von Styles hätte hier die Wiederverwendbarkeit der Komponente noch steigern können.

4.3.4 Wetter Komponente

Die React Wetter Komponente ist der Web Component sehr ähnlich, kann aber an vielen Stellen durch React Eigenheiten erheblich vereinfacht werden.

Im State der Wetter Klasse werden folgende Variablen initialisiert:

- `days`, für die Anzahl der anzuzeigenden Tage,
- `dayClass`, als Styling Klasse abhängig von der Anzahl der Tage,
- `response`, das Array der Vorhersagen und
- `chosenItem`, die aktuell ausgewählte Vorhersage.

Jede Änderung dieser Variablen löst somit automatisch ein neues Rendern der Komponente aus.

In der `componentDidMount()` Funktion wird die Open Weather Map One Call API aufgerufen. Das Array der Vorhersagen wird in die `this.state.response` Variable gespeichert. Als `this.state.chosenItem` wird das erste Objekt des Arrays gesetzt. Dadurch wird ein neuer Render ausgelöst, sobald die Antwort verfügbar ist. In der `render()` Funktion wird ein Array an Daily Komponenten erzeugt, das alle Vorhersagen erhält, deren Index kleiner als die Zahl der anzuzeigenden Tage ist. Als letztes Item im Array wird eine Daily Komponente erstellt, die das `id` Attribut mit Wert „show“ erhält und als Objekt immer `this.state.chosenItem` darstellt.

Ein sehr großer Vorteil von React ist hier das Virtual DOM Pattern. Wenn sich beim Web Components Ansatz der Zeitraum der Vorhersage ändert, muss überprüft werden, welche Objekte vorhanden sind, welche entfernt und welche neu hinzugefügt werden müssen. Dadurch entsteht viel Fehlerpotential, dass entweder zu viele oder zu wenig Objekte dargestellt werden. Bei der Entwicklung mit React hingegen wird automatisch die kleinstmögliche Anzahl an Änderungen ausgeführt. **Das erspart Entwicklungsaufwand, um alle Änderungsfälle zu berücksichtigen.**

Auch JSX hat die Entwicklung dieser Komponente sehr vereinfacht. Anstatt in der Tagesvorhersage für jedes Attribut eine Methode zu programmieren, die die Informationen an den vorgesehen Platz im Template setzt, kann direkt im `return` Statement die Property Variable an der richtigen Stelle verwendet werden.

Außerdem kann in React die URL des Icon-Sets einfach über eine Property gesetzt werden. Mit dem Web Components Ansatz ist dies nicht ohne Weiteres möglich, da die Inhalte zuerst in einem HTML Template festgelegt werden. Innerhalb dieses Templates gibt es noch keinen Zugriff auf die Attribute des Custom Elements. Aus Performance Gründen ist es empfehlenswert HTML Templates zu verwenden.
In diesem Fall muss aber nachdem die Komponente im DOM eingefügt wurde das `src` Attribut jedes `<img>` Tags noch einmal verändert werden. React bietet hier mit JSX Templates einen eindeutigen Vorteil.

4.4 Zusammenfassung

Obwohl die vorab definierten Kriterien noch nicht angewandt wurden, soll hier eine kurze Zusammenfassung der subjektiven Eindrücke bei der Entwicklung Platz finden.

Aus der Sicht eines Komponentenentwicklers haben Web Components mehr Nachteile. Die Verwendung des HTML `<template>` Tags bietet dynamische Inhalte, kann aber nicht mit den Vorzügen der Verwendung von JSX in React mithalten. Es gibt nur zwei Möglichkeiten Inhalte innerhalb eines Templates dynamisch zu setzen: DOM Operationen und Slots.

Slot-Elemente wirken auf den ersten Blick, als könnten sie die Rolle von Variablen erfüllen. **Durch einen Slot können aber nur HTML-Elemente eingefügt werden. Attribute wie Klassen oder `src` müssen mit Hilfe von DOM Operationen gesetzt werden,** wenn nicht das gesamte Element mit einem Slot gesetzt werden soll.

Auch das Styling von Slots ist eher verwirrend als intuitiv. Die Elemente die in die Slots des Templates geladen werden, befinden sich im Light DOM im Gegensatz zum Shadow DOM. Das heißt, dass das Styling des Shadow DOM keine Auswirkung auf das Element direkt hat. Trotzdem können aber Styles von Elementen im Shadow DOM geerbt werden. Um Elemente, die in einem Slot eingesetzt werden, direkt zu selektieren kann der `::slotted()` Selektor verwendet werden. Dieser kann aber nicht verwendet werden, um auf Nachfahren des geslotteten Elements zugreifen zu können.
Die Slots sind damit zwar nützlich, aber bieten auch ihre eigenen Herausforderungen, die beachtet werden müssen.

Die Alternative ist, die Elemente, die dynamische Inhalte erfordern, mit Hilfe von `id` Attributen oder anderen Selektoren zu erreichen. Dann können in den Custom Element Reactions Inhalte verändert werden.

Der offensichtliche Nachteil dabei ist, dass sehr viele Inhalte des Templates während der Laufzeit noch einmal geändert werden müssen. Bei React können ganz simpel Variablen im Markup verwendet werden. Für den Entwickler ist das eindeutig anschaulicher.

Ein ebenfalls arbeitsintensiver Schritt beim Ansatz Web Components ist es Veränderungen ressourcenschonend umzusetzen. Als Beispiel kann hier die Wetter

Komponente genannt werden. Beim Klick auf einen Button ändert sich die Anzahl der angezeigten Wetterberichte. Dabei sollen aber die bereits vorhandenen Elemente nicht gelöscht und erneut gerendert werden müssen. Deshalb muss hier überprüft werden, welche Elemente bereits im Shadow-DOM der Komponente existieren, welche entfernt und welche hinzugefügt werden müssen. React setzt auf das Virtual DOM Pattern, das bei Aufruf der `render()` Funktion automatisch die geringstmögliche Anzahl an Änderungen im DOM ausführt (vgl. Virtual DOM and Internals, o. J.).

Eine weitere Eigenheit der Web Components ist die Best Practice Attribute und Properties, wo möglich, synchron zu halten (vgl. Custom Element Best Practices | Web Fundamentals, 2019). **Das heißt für jedes Attribut muss eine `get` Funktion und eine `set` Funktion implementiert werden, damit die Information von einem anderen Entwickler sowohl wie ein Attribut als auch wie eine Property der Komponente behandelt werden kann.** Das erzeugt Boilerplate Code, der in React komplett wegfällt.

Trotz diesen Einschränkungen in der Entwicklung bieten Web Components jedoch auch Vorteile. **Die Kapselung von Styles innerhalb einer Komponente war erfrischend im Vergleich zur Arbeit mit globalem CSS.** Trotz der Verwendung von Bootstrap konnten sehr generische CSS-Selektoren verwendet werden, ohne ungewollte Styles überschreiben zu müssen, und ganz ohne die Verwendung der `!important` Regel.

Auch die Custom Element Reactions, die vergleichbar mit den React Lifecycle Methoden sind, haben ihre Vorteile. Das `attributeChangedCallback(oldValue,newValue,name)` gewinnt vor allem dadurch, dass der alte und der neue Wert eines Attributs verfügbar sind. Das ermöglicht es auf Änderungen spezifischer zu reagieren, ohne den alten Wert eines Attributs in einer eigenen Variablen festhalten zu müssen. Das `attributeChangedCallback` hat keine Entsprechung im React Lifecycle.

Im Folgenden wird in der Analyse auf die zuvor aufgestellten Kriterien zurückgegriffen.

5

Bewertung

5.1. Komponenten-Entwickler Kriterien

5.2. Komponenten-Nutzer Kriterien

5.3. Besucher Kriterien

5.4. Zusammenfassung Kriterien

5.5. Weitere Möglichkeiten für Web Components

5.1 Komponenten-Entwickler Kriterien

Wie viel „Boilerplate Code“ muss für jede Komponente geschrieben werden?

Der Boilerplate Code wird hier als Code, der für jede Komponente wiederholt werden muss und keinen besonderen Wert für die Komponentenlogik liefert, definiert.

Von allen Komponenten werden die Gesamtlänge des Codes und die Zeilenzahl an Boilerplate Code gezählt. Zum Gesamtcode zählen dabei alle React Components oder Custom Elements, die die Endkomponente ausmachen.

Da alle Gesamtkomponenten Anfahrt, Wetter und Kalender aus zwei bis drei kleineren Komponenten bestehen, kann sich Boilerplate Code wie zum Beispiel das Initialisieren des Shadow DOM auch innerhalb der Komponente mehrmals wiederholen. Diese Wiederholungen werden auch mehrmals gezählt.
In die Gesamtlänge des Codes fließt auch der CSS Code mit ein. Bei den Web Components ist dieser im Template integriert, um die Komponente nicht mit relativen Pfaden zu Stylesheets zu verkomplizieren. In React befindet sich der CSS Code in separaten Stylesheets.

Im Anhang sind alle Zeilen Code, die als Boilerplate Code gewertet wurden aufgelistet.

Für die React Komponenten mussten nur die Klassendefinition und der Import der React Skripte für alle Komponenten wiederholt werden.
In Zeilen Code bedeutet das:

Die Anfahrt Komponente besteht aus insgesamt 281 Zeilen, davon sind 5 Zeilen Boilerplate Code.

Die Kalender Komponente besteht aus insgesamt 347 Zeilen, davon sind 9 Zeilen Boilerplate Code.

Die Wetter Komponente besteht aus insgesamt 279 Zeilen, davon sind 6 Zeilen Boilerplate Code.

Im Durchschnitt sind die **React Komponenten** also **insgesamt 302 Zeilen lang** und enthalten gerundet **7 Zeilen Boilerplate** Code.

Für die Web Components Komponenten mussten die Template Definition, die Klassen Definitionen, das Initialisieren des Shadow DOM und Get und Set Funktionen der Attribute für alle Komponenten wiederholt werden. Daraus ergeben sich in Zeilen Code diese Zahlen:

Die Anfahrt Komponente besteht aus insgesamt 379 Zeilen, davon sind 68 Zeilen Boilerplate Code. Insgesamt ist die Web Component 98 Zeilen länger und enthält 63 Zeilen mehr Boilerplate Code als die React Komponente.

Die Kalender Komponente besteht aus insgesamt 524 Zeilen, davon sind 47 Zeilen Boilerplate Code. Insgesamt ist die Web Component 177 Zeilen länger und enthält 38 Zeilen mehr Boilerplate Code als die React Komponente.

Die Wetter Komponente besteht aus insgesamt 387 Zeilen, davon sind 42 Zeilen Boilerplate Code. Insgesamt ist die Web Component 108 Zeilen länger und enthält 36 Zeilen mehr Boilerplate Code als die React Komponente.

Im Durchschnitt ergibt sich dabei eine Länge von **430 Zeilen, also 128 Zeilen mehr als für die gleichwertigen React Komponenten.** Die Komponenten enthalten durchschnittlich gerundet **52 Zeilen Boilerplate Code. Das sind 45 Zeilen Boilerplate Code mehr als bei den React Komponenten** (vgl. Abbildung 10).

Daraus ergeben sich außerdem folgende Zahlen: Im Durchschnitt bestehen die **Web Components aus 12,1 % Boilerplate Code**, bei **React Komponenten sind es nur 2,3%**.

Mit nur wenigen Zeilen Boilerplate Code zeigt sich React hier als der deutlich ausdrucksstärkere Ansatz, der Inhalt über Code Quantität priorisiert.

Der deutlich höhere Umfang der Web Components Komponenten ist auch wenig überraschend. Schon bei der Programmierung war ersichtlich, dass React hier durch den Virtual DOM und das direkte Verwenden von Variablen in JSX viele Abschnitte verkürzen konnte.
Inwiefern die Menge an Boilerplate Code für Web Components mit Hilfe von externen Libraries reduziert werden kann, wird in Punkt 5.5 erforscht.

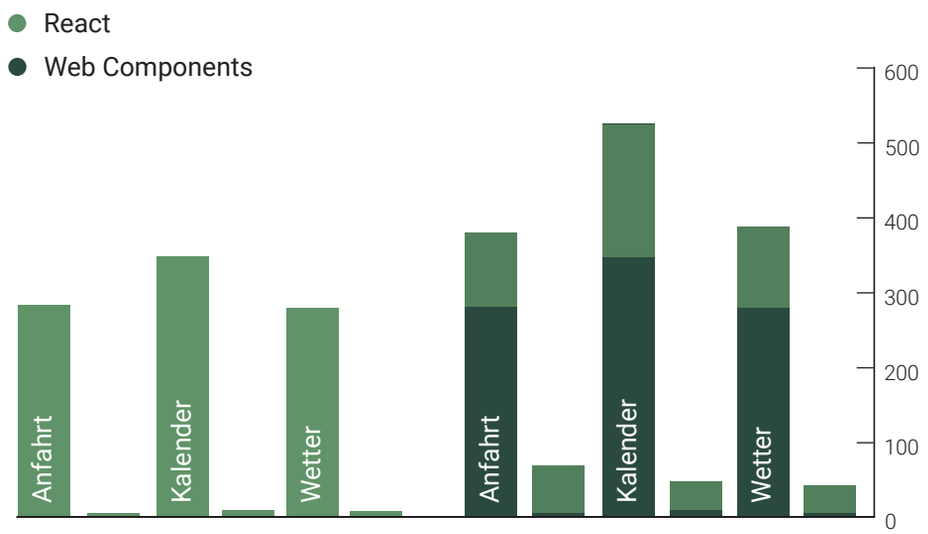

Abbildung 14: Vergleich der Länge des Gesamtcodes und Zeilen Boilerplate Code in React und mit Web Components

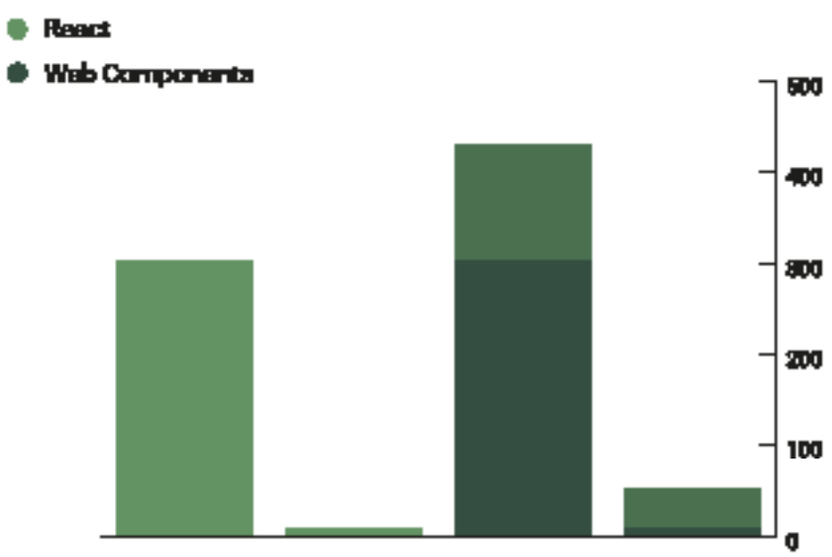

Abbildung 15: Vergleich vom Durchschnitt der Länge des Gesamtcodes und Zeilen Boilerplate Code in React und mit Web Components

Wie einfach können Anpassungen vom Entwickler unterstützt werden?

Um eine Komponente wiederverwendbar zu machen, muss diese auch von außen anpassbar sein. Für diese Anpassungen müssen vom Entwickler die entsprechenden Schnittstellen bereitgestellt werden.

An dieser Stelle soll der Aufwand für die Programmierung dieser Schnittstellen bewertet werden. Dazu wird die genaue Anzahl der Zeilen an Code bestimmt, die nur für Anpassungen von außen notwendig sind. **Beispiele hierfür sind das Setzen von Standardwerten, falls ein Wert nicht von außen gesetzt wurde oder bei Web Components die Get- und Set-Funktionen für Attribute.** In React ergeben sich dafür diese Zahlen:

Für die Anfahrt Komponente werden 19 Zeilen Code nur für Anpassungen benötigt. Insgesamt sind es 282 Zeilen Code.

Für die Kalender Komponente werden 40 Zeilen Code nur für Anpassungen benötigt. Insgesamt sind es 347 Zeilen Code.

Für die Wetter Komponente werden 15 Zeilen Code nur für Anpassungen benötigt. Insgesamt sind es 279 Zeilen Code.

Daraus ergeben sich **durchschnittlich gerundet 25 Zeilen Code für Anpassungen**. Im Durchschnitt machen die Anpassungen **8,3% des Gesamtcodes** aus.

Bei Web Components werden die Getter und Setter für die Attribute der obersten Komponente gezählt. Das heißt in einer Komponente, die aus mehreren verschachtelten Custom Elements besteht, werden nur die Getter und Setter der Komponente gezählt, die tatsächlich im HTML vom Entwickler verwendet wird. Die Attribute der tieferliegenden Custom Elements sind nicht für Entwickleranpassungen zuständig, sondern für die Kommunikation zwischen Custom Elements. Daraus ergeben sich die folgenden Zahlen:

Die Anfahrt Komponente besteht aus 379 Zeilen Code, davon sind 82 Zeilen für Nutzeranpassungen.

Die Kalender Komponente besteht aus 524 Zeilen Code, davon sind 52 Zeilen für Nutzeranpassungen.

Die Wetter Komponente besteht aus 430 Zeilen Code, davon sind 36 Zeilen für Nutzeranpassungen.

Im Durchschnitt enthalten die Web Components **gerundet 57 Zeilen Code für Nutzeranpassungen**. Damit machen Schnittstellen für Nutzeranpassungen **13,3% des Gesamtcodes aus.**

Web Components sind aus dieser Sicht im Nachteil, weil durch Getter und Setter der Attribute viel Boilerplate Code entsteht. Dabei muss jedoch beachtet werden, dass diese nur gesetzt werden, um dem Komponentennutzer mehrere Möglichkeiten zu geben mit der Komponente zu interagieren. In React können die Properties nur in der übergeordneten Komponente direkt wie Attribute gesetzt werden. Dies wird in den Komponentennutzer Kriterien berücksichtigt. Für den Komponenten Entwickler ist es zusätzlicher Aufwand, der in diesem Fall negativ bewertet werden muss.

Zusätzlicher Aufwand entsteht außerdem dadurch, dass Variablen nicht direkt im Template verwendet werden können. Slots können zwar als Platzhalter für dynamische Inhalte eingesetzt werden, aber nicht im gleichen Maße wie in React. Dort können Variablen direkt im JSX Rückgabe-Wert verwendet werden. Das macht die Anpassungen für den Komponentenentwickler sehr viel einfacher und auch (siehe nächster Punkt) deklarativer.

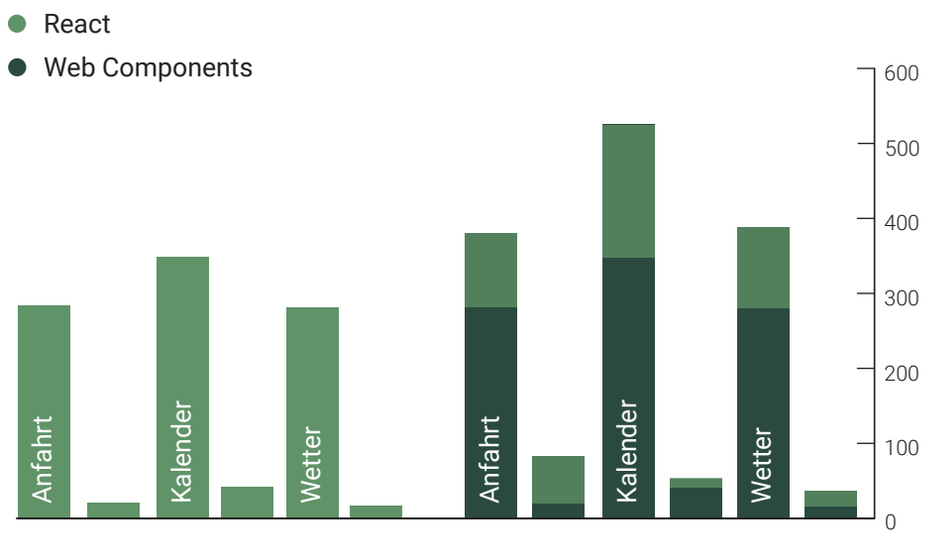

Abbildung 16: Vergleich der Länge des Gesamtcodes und Zeilen Anpassungs Code in React und mit Web Components

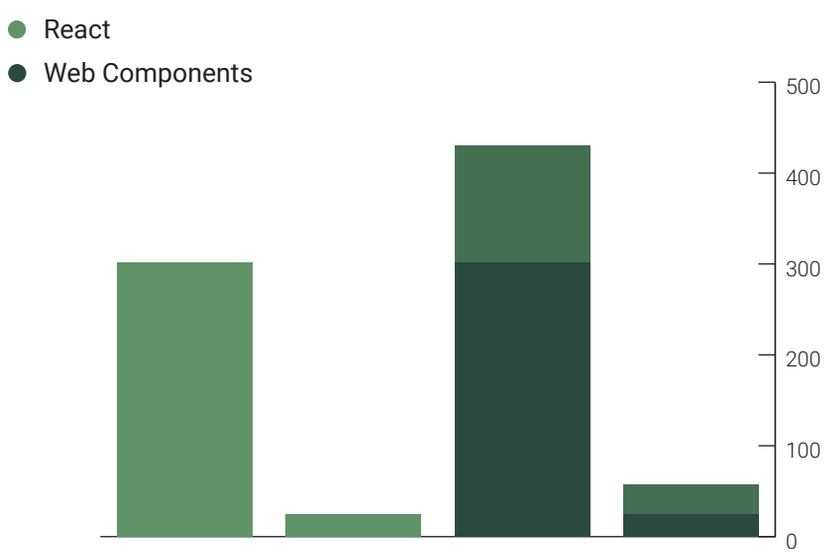

Abbildung 17: Vergleich vom Durchschnitt der Länge des Gesamtcodes und Zeilen Anpassungs Code in React und mit Web Components

Welcher Code weist mehr deklarative Merkmale auf?

Deklarative Programmierung kann verschieden ausgelegt werden. In Rückblick auf Kapitel 2.1.1 wird hier **deklarative Programmierung als ein Paradigma gesehen, nach dem Code ergebnisorientiert und entwicklerorientiert sein soll**. Dies soll dazu führen, dass die Programmierung verständlicher ist und somit einfacher bearbeitet und gewartet werden kann. Da React sich selbst als deklarativ präsentiert, ist es wenig verwunderlich, dass bei diesem Ansatz tatsächlich mehr deklarative Merkmale auftreten. Im Vergleich zu Web Components machen vor allem der Virtual DOM und die Verwendung von JSX den Unterschied.

Durch das Virtual DOM Pattern ist es in React möglich ergebnisorientiert zu programmieren. Als Beispiel kann die Wetter Komponente herangezogen werden. In dieser befinden sich vier Buttons, um den Zeitraum der Wettervorhersage zu ändern. Wird der Zeitraum zum Beispiel von fünf Tagen auf acht Tage verlängert, wäre es ideal die ersten fünf Vorhersagen nicht neu rendern zu müssen. Um dies mit Web Components umzusetzen, muss überprüft werden, welche Vorhersagen bereits vorhanden sind und welche neu gerendert werden müssen. Darauf müssen genaue Anweisungen folgen, inwiefern das DOM verändert werden muss. In React reicht es beim Klick auf den Button zu definieren, dass die Anzahl verändert wurde. Im Virtual DOM wird darauf der Idealzustand mit 8 Vorhersagen festgehalten. Beim Vergleich mit dem vorherigen Virtual DOM werden die notwendigen Operationen – das Hinzufügen von drei Vorhersagen – errechnet. Die Schritte zur Überprüfung und Veränderung können in React abstrahiert werden.

Mit Hilfe von JSX kann in React entwicklerorientierter Code geschrieben werden. Im XML-artigen JSX-Code können vorher gesetzte Variablen direkt verwendet werden. Das macht es für einen Entwickler einfacher zu verstehen, welcher Wert genau an einer bestimmten Stelle verwendet wird. Außerdem macht es dynamische Inhalte durch einen Unterschied in der Schreibweise direkt sichtbar.
Diese Unterschiede machen den React Code einerseits verständlicher. Wie in der Auswertung der vorherigen Kriterien bereits zu sehen war, ist der React Code außerdem deutlich kürzer. **Im Durchschnitt konnten in React 150 Zeilen eingespart werden. Diese Einsparungen sind mit Sicherheit auch auf die deklarativen Aspekte zurückzuführen.**

```
onClick={()=> {
this.setState({chosenItem: this.state.response[i]})
}};
```

```
forecast.onclick=(e)=>{
this.shadowRoot.querySelector(".chosen").removeAttribute("class");
this.shadowRoot.getElementById("show").remove();
let clone = e.target.cloneNode(true);
e.target.setAttribute("class","chosen");
clone.setAttribute("id","show");
this.shadowRoot.getElementById("weather").appendChild(clone);
};
```

Abbildung 18: Einsparung von Code mit deklarativer Programmierung in React (oben) im Vergleich zu Web Components (unten)

5.2 Komponenten-Nutzer Kriterien

Wie einfach sind Anpassungen für einen Entwickler, der die Komponenten verwendet, umzusetzen?

Auf den ersten Blick sind die Anpassungsmöglichkeiten für den Komponentennutzer bei beiden Ansätzen sehr ähnlich.

In React können sowohl primitive Datentypen als auch Javascript Objekte wie HTML Attribute als Props übergeben werden. Bei Web Components sind eben diese Attribute eine der wichtigsten Interaktionsmöglichkeiten.

Im Unterschied zu React können mit Attributen aber nur primitive Datentypen wie Strings oder Zahlen übergeben werden. Für Objekte können Properties in Javascript gesetzt werden. Attribute und Properties sollten laut Best Practices synchron gehalten werden, so dass alle Attribute auch wie Properties behandelt werden können (vgl. Custom Element Best Practices, 2019). Außerdem können ganze HTML-Stücke mit einem Slot-Attribut übergeben werden.

React setzt eher auf Einheitlichkeit. In React können zwar auch die Kinder einer Komponente an einer bestimmten Stelle eingesetzt werden. Mehrere Elemente für verschiedene Stellen zu markieren, wie es mit Slots möglich ist, geht in React aber nur mit der Verwendung von Props. Dadurch ist die Verwendung von Kind Elementen in React eher unüblich (vgl. Composition vs Inheritance, o. J.).

Web Components auf der anderen Seite müssen mehrere verschiedene Möglichkeiten bieten. Slotables entsprechen normalen Kind Elementen und sind dadurch sehr intuitiv, auch für Nicht-JavaScript Experten. **Der Unterschied zwischen Attributen und Properties ist jedoch eine eindeutige Einschränkung.** Dadurch, dass komplexe Datentypen nicht als Attribute gesetzt werden können, müssen Anpassungen eventuell an zwei sehr verschiedenen Stellen gesetzt

werden. Das ist technisch notwendig, aber sehr unintuitiv. Für die Synchronisierung von Attributen und Properties muss man sich außerdem auf die Genauigkeit des Komponentenentwicklers verlassen. Falls dieser nicht konsequent Getter und Setter für alle Attribute implementiert hat, kann es zu unerwartetem Verhalten kommen, wenn Attribute und Properties einer Komponente nicht synchron sind.

Das Problem von Attributen und Properties wird oft in der Verwendung in Kombination mit Frontend Frameworks gelöst. **Bei der Verwendung eines Custom Elements in Vue können zum Beispiel Properties genau wie Attribute übergeben werden.** Dadurch ist keine Trennung der Nutzeranpassungen notwendig.

CSS Anpassungen können größtenteils vernachlässigt werden. Sowohl in React als auch in Web Components ist dies am einfachsten mit CSS Variablen zu erreichen, die vom Entwickler im eigenen Stylesheet gesetzt werden können. Hier sind keine Unterschiede anzutreffen.

Wie einfach können Komponenten-Styles vor allgemeinen Styles geschützt werden?

Wenn ein Stück HTML-Markup außerhalb seines eigentlichen Kontexts wiederverwendet wird, kann es oft zu Styling Problemen kommen. Das Styling des neuen Kontexts verändert den neuen Inhalt, und der neue Inhalt und das dafür eingefügte CSS verändern den Inhalt der Seite.

Das gewünschte Ergebnis beim Nutzen dieser Komponente wäre, dass diese Styles sich nicht beeinflussen. Wenn Styles angepasst werden müssen, sollte dies willentlich vom Komponentennutzer hervorgerufen werden.

In welchem Ausmaß diese Probleme bei den verschiedenen Ansätzen auftreten, wird mit einer `test.html` Datei mit neuem Stylesheet getestet. An den folgenden Abbildungen ist zu erkennen, wo die verschiedenen Styles sich beeinflussen.

Abbildung 19: Test HTML Datei ohne eingefügte Komponenten

Abbildung 20: Test HTML Datei mit Web Components: Es sind keine Style Unterschiede erkennbar

Abbildung 21: Test HTML Datei mit React Komponenten: In der Navigation und im Kalender Header sind Probleme zu erkennen

Bei den Web Components gibt es keine Styling Konflikte. Die Schriftart innerhalb der Komponente verändert sich aufgrund von Vererbung (vgl. Abbildung 14).

Bei den React Komponenten treten einige Konflikte auf (vgl. Abbildung 15). Folgende Werte des Test Stylesheets wurde überschrieben:

- `.btn-dark{background-color, border-color}`
- `button {width}`
- `.form-control{height; min-height; padding; border; background-color; border-radius}`
- `form {width; height; box-shadow}`

Folgende Werte der Komponenten wurden vom Test-Stylesheet überschrieben:

- `button{font-size}`
- `header::before {content; width; height; background; position}`

Das heißt in React müssten an sechs verschiedenen Stellen genauere Selektoren verwendet werden, um keinen Einfluss auf ungewollte Regionen zu haben. **Hier zeigen sich die Vorteile des Shadow DOM. Aufgrund der Style Kapselung braucht es keine weitere Arbeit, um die Komponente in einem anderen Kontext zu verwenden.** Anpassungen sind über die vom Entwickler definierten Schnittstellen weiterhin möglich.

Bei den React Komponenten gibt es auch gute Möglichkeiten diesen Konflikten zu entgehen.

Spezifischere Selektoren sind meist Standard. Vier der Selektoren, die von den Style Konflikten betroffen waren, sind Element Selektoren. Die weiteren zwei sind Klassen, die von Bootstrap verwendet werden. Mit leichten Anpassungen könnten auch hier viele Konflikte vermieden werden. Je nach Größe einer Anwendung und des Entwicklerteams können so aber nie alle Konflikte ausgeschlossen werden.

Eine andere Möglichkeit in React Style Konflikte zu verhindern ist CSS-in-JS Lösungen zu verwenden. Diese wurden in diesem Fall nicht getestet, da sie nicht von React selbst bereitgestellt werden. Mit diesen Lösungen ist es möglich Styles innerhalb von JavaScript mit einzigartigen Namen generieren zu lassen. Dadurch werden Style Konflikte vermieden, obwohl simple Selektoren verwendet werden. Beispiele für solche Erweiterungen sind JSS, Styled Components oder emotion (vgl. Comparison with Other Frameworks, 2020; JSS, 2020).

Wie einfach lässt sich eine Komponente in ein anderes Framework einbauen?

Für die Einarbeitung in Vue wurde Vue zuerst nur über einen `<script>` Tag eingebunden, um eine minimale Version ohne Build-Setup testen zu können. Dadurch konnten keine Single File Components verwendet werden. In dieser Variante konnten Web Components sehr einfach über einen `<script>` Tag im HTML oder über ein `import` Statement in Javascript eingebunden werden.

Für eine skalierbarere Version wurden die Web Components ebenfalls in einer App mit Webpack Setup getestet. **Diese wurde mit Hilfe des Vue CLI (Vue Command Line Interface) erstellt. Dieses Tool ermöglicht es, von der Konsole aus eine Vue Anwendung mit fertigem Build-Setup zu erstellen.** Mit diesem Setup war es möglich mit Single-File-Components zu arbeiten. Diese stellen eine besondere Notation dar, da dort Template, Logik und Styling nacheinander in derselben Datei Platz finden.

Bei der Einbindung einer Web Component in diese Anwendung mussten einige kleine Anpassungen getroffen werden. Wird ein individueller Tag innerhalb einer Komponente verwendet, wird das von der Evaluierung als Vue Komponente interpretiert. Ist keine Komponente unter diesem Tag-Namen registriert, wird ein Fehler in der Konsole ausgeworfen. **Dies kann verhindert werden, indem in der Main-Datei die Tag Namen zu den ignorierten Elementen hinzugefügt werden** (vgl. Custom Elements Interop | Vue.js, 2020). Die Komponenten wurden über die `index.html` eingebunden. Die strenge Evaluierung durch ESLint, einem Plug-in das die Code Qualität erhöhen soll, lässt in der Standard Konfiguration den Import innerhalb der Komponente nicht zu (vgl. OpenJS Foundation and other contributors, 2020).
Die React Komponente wurde nur im Vue CLI Setup mit Single File Components getestet. **Da React Komponenten als Klassen oder Funktionen dargestellt werden und Vue Komponenten als Objekte, mussten hier viele Anpassungen vorgenommen werden.** Die Reihenfolge des Codes innerhalb der Komponente wurde durch diesen Unterschied in der Darstellung gänzlich verändert. Auf interne Daten wird in Vue nicht mit `this.state` zugegriffen, dementsprechend muss auch nicht `this.setState()` für Änderungen aufgerufen werden. Das wurde mit `this` und einer normalen Zuweisung ersetzt.

Auch auf `props` wird nicht mit `this.props`, sondern nur mit `this` zugegriffen. Es gibt keine `render()` Funktion sondern ein Template und es wird kein JSX verwendet. Sowohl `render()` Funktionen als auch JSX können zwar durchaus in Vue eingesetzt werden. Aufgrund des anderen Aufbaus der `render()` Funktion hätte diese jedoch trotzdem stark angepasst werden müssen, weswegen hier die sehr viel weiterverbreitete Variante eines Templates verwendet wurde (vgl. Comparison with Other Frameworks, 2020).
Arrays mit JSX-Objekten wurden hier durch die `v-for` Funktionalität ersetzt. Komponenten, Variablen und Props konnten mit gewissen Syntax-Änderungen im Template ähnlich wie in JSX verwendet werden.

Web Components können in React ähnlich wie in Vue integriert werden. Die ESLint Evaluierung beim Create-React-App Setup geht jedoch nicht so weit, dass die unbekannten Tags einen Fehler werfen würden. Deshalb ist nur der Import in der `index.html` oder in der Komponenten-Datei notwendig, um die Komponente verwenden zu können.

Im Vergleich zu anderen Frontend Bibliotheken gibt es jedoch 2 Probleme:
Aufgrund des synthetischen Event Systems von React kann nicht direkt auf DOM Events reagiert werden, die in einer Web Component ausgelöst werden. Daher muss dort immer manuell ein Event Listener gesetzt werden.
In anderen Frameworks ist es außerdem möglich komplexe Datentypen als Properties zu übergeben. In React werden diese immer als Attribute behandelt und dadurch unbrauchbar (vgl. Dodson, o. J.).

Web Components bieten also einen sehr schnellen und einfachen Weg eine fremde Komponente in Vue zu verwenden. Mit nur drei Zeilen Code kann eine Komponente verwendet werden. Für eine React Komponente ist dies nicht möglich. Hier muss sowohl Kenntnis von Vue als auch von der Funktion der Komponente vorhanden sein, um sie an die anderen Konzepte anzupassen. Obwohl Vue und React viele Ähnlichkeiten aufweisen, müssen einige Syntax Unterschiede ausgeglichen werden (vgl. Comparison with Other Frameworks, 2020).

Auch in React können Web Components einfach integriert werden. React bietet jedoch keine Komfort-Lösungen für Event Handling und Properties, wie andere Frontend Bibliotheken es tun (vgl. Dodson, o. J.).

Der gesamte Code der Vue Komponente kann auf Github unter https://github.com/ssimone175/bachelor-vue-test eingesehen werden.

5.3 Besucher Kriterien

Welche Seite lädt schneller?

Beim Google Page Speed Insights Test wird die Geschwindigkeit der Seite anhand von verschiedenen Metriken analysiert. **Der Wert hängt von vielen Faktoren ab und ist dementsprechend nicht jedes Mal exakt gleich.** Die Werte hier sind als nur als ein mögliches Wertpaar zu begreifen, das einen Trend in der Geschwindigkeit der Seite anzeigen kann.

Die Ergebnisse bei diesem Page Speed Test lauten wie folgt:
Ergebnis Web Components:
34 Mobile, 74 Desktop
Ergebnis React:
28 Mobile, 63 Desktop

Da der Page Speed Test nicht immer das gleiche Ergebnis liefert wurden mehrere Testläufe durchgeführt. **Aus diesen ergab sich ein kleiner Performance Vorteil der Web Components vor allem bei Desktop Ladezeiten.** Bei mobilen Ladezeiten unterschieden sich die Ergebnisse so oft, dass kein besonderer Unterschied in der Performance festgestellt werden kann.
Von anderen Testern online werden ebenfalls schnellere Ladezeiten für Web Components festgestellt (vgl. Kamboj, 2020; Hellberg, 2017). **Dieses Ergebnis ist nicht sehr überraschend, da Web Components als Standard immer leichtgewichtiger sein werden als eine Bibliothek oder ein Framework.** Dabei darf aber nicht unterschätzt werden, dass Web Components vom Entwickler optimiert werden müssen. Für Veränderungen in React wird der Virtual DOM verwendet. Bei einem nativen Ansatz wie Web Components müssen Zustandsänderungen vom Entwickler überwacht werden. Web Components sind also keine Garantie für eine bessere Performance, wenn die Optimierung vom Entwickler abhängt. Die Performance Unterschiede werden innerhalb dieses Tests also vernachlässigt.

Werden Accessibility Tools unterstützt?

Im Rahmen dieses Tests wurde die **Browser Extension Dark Mode für Chrome verwendet, die es ermöglicht Seiten mit einem User-Stylesheet zu verdunkeln.** Das ist sehr nützlich für Menschen mit lichtsensitiver Migräne. Mit User-Stylesheets können außerdem Schriftgrößen und Text-Kontraste erhöht werden. Das kann Menschen mit Sehschwächen oder Lese-Rechtschreib-Schwächen das Lesen erleichtern.

Bei React funktionieren diese Anpassungen wie gewünscht, mit **Web Components verhindert der Shadow DOM die korrekte Funktionsweise (vgl. Abbildung 23).** Damit Menschen mit eingeschränktem Sehvermögen Computer benutzen können, verwenden sie meist einen sogenannten Screenreader. Dieser gibt den Inhalt einer Webseite oder eines Dokuments entweder als Audio oder über ein Braille-Lesegerät aus. **Für den Screenreader Test wurde JAWS als eines der häufigsten Programme in der Testversion in Kombination mit Google Chrome verwendet** (vgl. WebAIM, 2019).

Screenreader brauchen leider viel Einarbeitungszeit, da einige Tastenkombinationen auswendig gelernt werden müssen (vgl. Aktion Mensch e. V., o. J.). Dadurch war es nicht möglich, weitere Screenreader oder mit anderen Browsern zu testen.

Trotz der Kapselung des Shadow DOM konnten in diesem Fall keine Probleme fest-

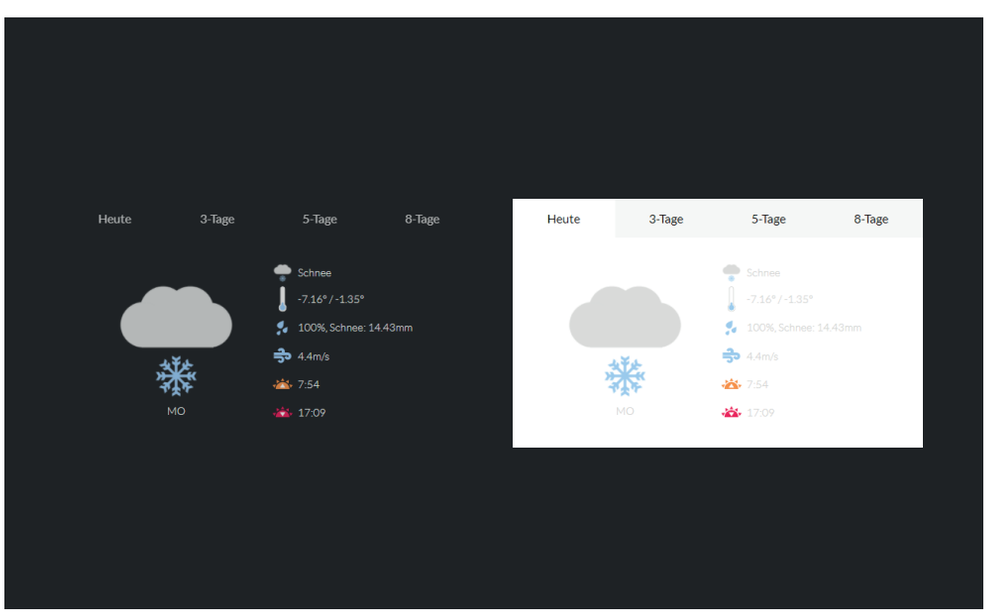

Abbildung 22: Funktion der Dark Mode Browser Extension, links in React, rechts mit Web Components

gestellt werden. Der Screenreader arbeitet zwar mit dem Document Object Model, das eigentlich separat vom Shadow DOM eines Elements ist. Im Browser wird jedoch beim Rendern ein flaches Model erzeugt. Dieses enthält sowohl das DOM des Markup als auch das Shadow DOM von Komponenten. Dadurch hat der Screenreader nicht nur Zugriff auf das Shadow DOM, sondern ordnet es auch richtig im Dokumentenfluss ein (vgl. Sutton, 2014).

Buttons, Links und Input Felder konnten mit der Tastatur navigiert und bedient werden.

Accessibility in Web Components ist ein diskutiertes Thema. Standard HTML Elemente wie `<button>` oder `<input>` bieten Accessibility Funktionen, die für eingeschränkte Nutzer notwendig sind, um sich im Web zurechtzufinden. Oft ist aber das Styling dieser Standard Elemente sehr komplex. Deshalb werden diese Elemente manchmal nachgebaut, um ein individuelles Styling verwenden zu können. **Wenn jedoch mit einem Custom Element Standard Elemente nachgebaut werden, gehen eingebaute Accessibility Funktionen verloren.** Diese wiederherzustellen, hat für die meisten Entwickler wenig Priorität.

Im Chrome Browser ist es wegen diesem Problem möglich, ein Custom Element das Verhalten eines Standard Elements erben zu lassen. Das geerbte Verhalten bleibt aber nur bestehen, solange das Custom Element keinen eigenen Shadow DOM erhält. Dadurch sind die Style Anpassungen wieder erheblich eingeschränkt. Diese Funktion wird außerdem von anderen Browsern nicht implementiert (vgl. Dodson, 2017).

Das größte Problem für Accessibility ist der Shadow DOM. Das ausdrückliche Ziel des Shadow DOM ist es, Komponenten abzukapseln. Dadurch können aber auch User-Stylesheets Elemente im Shadow DOM nicht verändern.

Im Test dieser Komponenten gab es keine Probleme mit Screenreadern oder Tastatur-Navigation. Bei kleineren Komponenten kann dies aber durchaus vorkommen.

Als Beispiel kann ein Formular mit einem Custom Element Input-Feld und ein Custom-Element Label dienen. Haben beide Elemente einen Shadow DOM, kann das Label nicht über das `for` Attribut mit dem Input Feld in Verbindung gebracht werden. Dazu braucht es nämlich ein eindeutiges `id` Attribut . IDs sind jedoch für jeden Shadow DOM gekapselt. Ein Screenreader kündigt in diesem Fall bei der Navigation mit Tab nicht an welche Beschriftung das Input Feld hat. Das macht es für einen Besucher fast unmöglich ein Formular auszufüllen (vgl. Dodson, 2017; Powell, 2019).

An diesem Test zeigen sich die Nachteile der vollständigen Style Kapselung einer Komponente. **Was für einen Komponentennutzer vielleicht wünschenswert ist, ist für manche Besucher einer Webseite eher ein Problem.** Mit Hilfe von CSS Variablen sind Style Anpassungen durch ein User Stylesheet zwar technisch möglich. Der Aufwand, auf jeder Seite mit Shadow DOM die Variabelnamen herauszufinden und dementsprechend das Stylesheet zu bearbeiten, ist jedoch unverhältnismäßig. Dies setzt zusätzlich voraus, dass diese Variablen vom Entwickler unterstützt wurden.

Die Probleme des Shadow DOM können nicht vom Entwickler ausgeglichen werden. Sie erfordern neue Ansätze, wie zum Beispiel das Accessibility Object Model (vgl. Dodson, 2017). Mit diesem wäre es unter Anderem möglich Beziehungen zwischen zwei Elementen ohne `id` Referenzen auszudrücken (vgl. Boxhall et al., 2020). Diese Ansätze werden jedoch kaum unterstützt (vgl. Boxhall/Mazzoni, 2020). **Ein Entwickler muss also die Wahl zwischen der Verwendung von Shadow DOM und einer vollständigen Barrierefreiheit treffen.**

5.4 Kriterien Zusammenfassung

Der größte Vorteil von Web Components liegt im weitläufigen Support des Standards. Dadurch sind sie plattformübergreifend nutzbar. Außerdem kann durch die Verwendung des Shadow DOM eine Kapselung der Komponenten erreicht werden, die sie leicht wiederverwendbar machen. **Diese Vorteile werden jedoch durch den höheren Entwicklungsaufwand und fehlende Unterstützung von Accessibility Tools überschattet.**

Einige Vorteile können in React sehr einfach nachgerüstet werden: So ist das Kapseln von Styling für nur eine Komponente mit mehreren CSS-in-JS Lösungen möglich. Diese generieren zur Laufzeit eindeutige Klassennamen, die Styling-Konflikte verhindern (vgl. Comparison with Other Frameworks, 2020; JSS, 2020). **Allein die plattformübergreifende Nutzung kann React nicht in gleicher Weise bieten.**

Hier kommen Kombinationsmöglichkeiten von Web Components und Frontend-Frameworks ins Spiel. Angular bietet mit Angular Elements eine direkte Möglichkeit, Komponenten die innerhalb des Frameworks erstellt wurden, als Web Component zu verpacken. Mit einer minimalen Distribution des Frameworks wird die Funktionalität der Komponente hergestellt. Diese kann dann wie eine Web Component in verschiedenen Frameworks und in purem HTML verwendet werden (vgl. Angular - Angular Elements Overview, 2020).
Bei Vue ist dieses Verhalten mit der Bibliothek Vue Custom Elements möglich (vgl. Bemenderfer, 2020).

Auch React Komponenten können als Web Components verpackt werden. Für diesen Prozess kann zum Beispiel die Bibliothek Direflow verwendet werden (vgl. Høiberg, 2020). Hier gibt es jedoch keine so verbreitete Möglichkeit wie in Vue oder Angular. Im Hinblick auf den Support von Web Components in React gibt es größere Probleme. Das synthetische Event System von React erzwingt, Event Listener hinzuzufügen, um auf DOM Events innerhalb einer Web Component reagieren zu können. Außerdem können komplexe Datentypen nicht direkt als Properties übergeben werden (vgl. Dodson, o. J.).

Der Test zeigt, dass Web Components große Möglichkeiten bieten, um Komponenten an Nutzer auszulieferen, die sie einsetzen möchten, ohne sie verstehen zu müssen. Für Entwickler und Besucher einer Webseite bieten Web Components weniger Vorteile als ein klassisches Frontend Framework. Für Besucher kann jedoch der Vorteil der Style Kapselung aufgegeben werden, um Accessibility Tools nicht zu blockieren. Dies erfordert dann etwas mehr Disziplin für den Entwickler möglichst einzigartige CSS-Selektoren und ID-Attribute zu verwenden, so dass keine Konflikte entstehen. **Im nächsten Kapitel werden zwei Frameworks getestet, die die großen Nachteile des Entwicklers für Web Components ausgleichen können sollen.**

5.5 Weitere Möglichkeiten für Web Components

Web Components haben durchaus Vorteile und Anwendungsmöglichkeiten. **Das größte Problem der Web Components ist aber die komplizierte Entwicklung.** Wer gerne mit Angular oder Vue arbeitet, kann mit diesen Frameworks Web Components erstellen. Somit können hier alle Entwicklervorteile der Frameworks und die einfache Auslieferung von Web Components genutzt werden. Wer jedoch nicht mit den genannten Frontend Frameworks arbeiten möchte, kann möglicherweise auf die folgenden beiden Web Component Frameworks zurückgreifen.

Stencil

Stencil ist ein Framework das von Ionic entwickelt wurde, um einfacher mit Web Components arbeiten zu können (vgl. Stencil – Introduction, o. J.). **Stencil setzt dabei auf ähnliche Konzepte wie React.** Das Virtual DOM Pattern und `render()` Methoden mit JSX sind beides Merkmale von React, die Stencil übernimmt. Ein Unterschied ist jedoch die zwingende Verwendung von TypeScript, um JavaScript um explizite Typen zu ergänzen. **Stencil kann damit als Möglichkeit gesehen werden mit React ähnlichen Tools zu arbeiten und Komponenten als Web Components auszuliefern.** Ähnliche Möglichkeiten gibt es wie bereits oben erwähnt in anderen Frontend Frameworks von Haus aus.

Bei der Reduktion von Boilerplate Code kann Stencil punkten, in dem Shadow DOM Initialisierung und das Registrieren des Custom Elements automatisch vorgenommen werden. Auch das Behandeln von Properties und Attributen kann mit einem Decorator um einiges verkürzt werden.
Im Vergleich zu Web Components ohne Framework kann so der Code um 87 Zeilen von 387 auf 300 Zeilen verkürzt werden.

Ein weiterer großer Vorteil von Stencil ist, dass beim Build der Komponente automatisch eine Dokumentation ausgegeben wird.

Diese enthält Informationen über Properties und Events mit Typ und Kommentar falls vorhanden. Zusätzlich wird eine Visualisierung der Komponentenbeziehungen generiert (vgl. Rauber, 2020b). **Um die Komponenten für andere Entwickler wiederverwendbar zu machen, ist eine gute Dokumentation ein wichtiges Hilfsmittel. Hier kann Stencil dem Entwickler einen weiteren Teil Arbeit abnehmen.**

Der Code der Stencil Komponente kann unter dem Link https://github.com/ssimone175/bachelor-stencil-test eingesehen werden.

LitElement

LitElement arbeitet ebenfalls mit einem Template in einer `render()` Funktion. **Im Unterschied zu Stencil basieren diese Templates jedoch auf lit-html.**

Die Templates werden in der `render()` Funktion mit Template Strings angegeben. Diese verwenden das Accent Grave Zeichen (`) anstatt Anführungszeichen. Dadurch können sie mehrere Zeilen überspannen und Variablen, die mit einem Dollarzeichen und geschweiften Klammern markiert werden, enthalten (vgl. Template literals (Template strings) – JavaScript, 2020).

Die html Funktion von lit-html übersetzt diese Template Strings in ein TemplateResult Objekt.

Für das Rendern des Templates wird in LitElement kein Virtual DOM verwendet. Im Virtual DOM wird der gesamte Inhalt neu gerendert, um dann die notwendigen Änderungen zu identifizieren. Stattdessen werden in LitElement nur die Teile neu gerendert, die die Variablen verwenden. Das soll zu Performance Vorteilen führen (vgl. Templates – LitElement, o. J.; Lit-html, o. J.).

LitElement bietet wie Stencil eine Kurzschreibweise für Properties. TypeScript ist in diesem Fall aber optional und nicht Pflicht.

Im Vergleich zu Stencil bringt Lit-Element weniger fremde Konzepte mit. Dadurch ist der Code gefühlt näher am Web Components Standard. Aber auch mit Lit-Element kann Boilerplate Code eingespart werden. **Anstatt 387 Zeilen sind in Lit-Element nur 310 Zeilen für die gleiche Komponente notwendig.**

Der Code der LitElement Komponente kann unter https://github.com/ ssimone175/bachelor-litelement-test eingesehen werden.

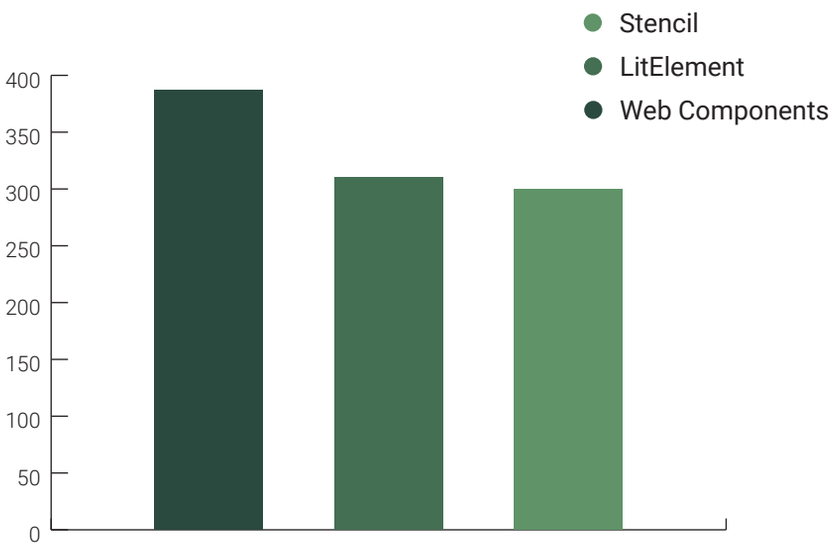

Abbildung 23: Länge der Wetter Komponente mit Web Components, Stencil und LitElement im Vergleich

6

Fazit

Keine Konkurrenz

Web Components und React bieten Überschneidungen, aber sie verfolgen nicht die gleichen Ziele.
React auf der einen Seite soll die Frontend-Entwicklung allgemein vereinfachen, mit deklarativen und komponentenbasierten Techniken.
Web Components streben grundsätzlich keine Vereinfachung an. Als JavaScript Standard verfolgen sie das Ziel, Komponenten in purem JavaScript überhaupt möglich zu machen. Zusätzliche Funktionen wie ein Virtual DOM würden hier den Rahmen sprengen, da Bibliotheken, die diese Funktion abdecken, weit verbreitet sind.

Dementsprechend bieten diese Ansätze auch verschiedene Vorteile. **Während React dem Komponentenentwickler viel Arbeit abnimmt, können Web Components von fremden Entwicklern sehr einfach in andere Anwendungen integriert werden.** Diese plattformübergreifende Nutzung ist der größte Vorteil von Web Components.

Ist diese nicht notwendig, lohnt es sich auch nicht auf Web Components zu setzen. Wer also eine Möglichkeit sucht effizient Web Anwendungen zu entwickeln, sollte sich an die etablierten Frameworks halten.

Es gibt gute Gründe, wieso sich Web Components hier noch nicht durchgesetzt haben.

Wer nach Möglichkeiten sucht, Komponenten anwendungs- und plattformübergreifend zu verwenden, sollte sich definitiv Web Components anschauen. Dabei muss man als Entwickler auch nicht auf Vorteile der Frameworks verzichten.
Sowohl Vue als auch Angular bieten komfortable Möglichkeiten Komponenten als Web Components auszuliefern (vgl. Angular - Angular Elements Overview, 2020; Bemenderfer, 2020).

Stencil bietet mit einer Kombination aus Virtual DOM und JSX eine ähnliche Entwicklungsexperience wie React, mit einigen zusätzlichen Konzepten wie TypeScript.

Wer näher am Standard arbeiten möchte, aber trotzdem dynamischen Content im Template direkt schreiben möchte, findet vielleicht mit Lit-Element oder lit-html eine gute Bibliothek.

Bevor Web Components sich in größerem Maße durchsetzen werden, werden wahrscheinlich noch einige Anpassungen folgen

müssen. React ist eine der wenigen Bibliotheken, die noch gewisse Schwierigkeiten bei der Integration von Web Components vorweist (vgl. Dodson, o. J.).
Zuerst bietet React keine Möglichkeit Objekte als Properties anstatt als HTML Attribute an die Web Component zu übergeben. Dadurch werden Objekte und Arrays beim Übergeben zu Strings gemacht und können nicht verwendet werden.

Außerdem spielt eine Rolle, dass React eigene Synthetic Events definiert. Dadurch kann außerhalb auf kein DOM Event, das in einer Web Component ausgesendet wird, reagiert werden. Um dies zu umgehen, können mit Hilfe einer Referenz Event Listener manuell gesetzt werden. Dies ist leider zusätzlicher Aufwand (vgl. Dodson, o. J.).

React steht einer komplementären Verwendung mit Web Components jedoch nicht entgegen (vgl. Web Components, o. J.). Für die Zukunft kann hier also auf eine Verbesserung der genannten Probleme gehofft werden.
Im Moment sind Web Components also nur für spezielle Use Cases sinnvoll, in der eine anwendungs- oder plattformübergreifende Nutzung der Komponenten klar im Vordergrund steht.

Konzepte, die Web Components ausmachen, können auch in Frameworks nachgerüstet werden. Eine Style Kapselung in React ist zum Beispiel durch CSS-in-JS Lösungen wie JSS, Styled Components oder emotion möglich.

Das Ergebnis diese Arbeit ist mit Sicherheit, dass React und Web Components keine Konkurrenten sind. Die Ansätze verfolgen gänzlich verschiedene Ziele und Ideen. Anstatt sich für das Eine oder das Andere entscheiden zu müssen, ist viel mehr die Frage, wie die beiden Ansätze sich positiv beeinflussen können.
Durch eine bessere Integration von Web Components in React könnte ein weiterer Markt an Third-Party Komponenten für React erschlossen werden.
Mit einer einfacheren Möglichkeit React Komponenten als Web Components auszuliefern, könnten die Vorteile beider Ansätze kombiniert werden.

Je mehr Frameworks die Nutzung als Custom Element zulassen, desto mehr verliert auch die Wahl des Frontend Frameworks an Bedeutung. Der Austausch unter verschiedenen Framework Nutzern würde dadurch einfacher und weniger zur Glaubensfrage. **Vielleicht könnten Web Components so zu einem harmonischeren Web beitragen.**

Gendervermerk

In der vorliegenden Bachelorarbeit wird aus Gründen der besseren Lesbarkeit die gewohnte männliche Sprachform bei personenbezogenen Substantiven und Pronomen verwendet.

Dies impliziert jedoch keine Benachteiligung des weiblichen oder anderweitigen Geschlechtern, sondern soll im Sinne der sprachlichen Vereinfachung als geschlechtsneutral zu verstehen sein.

Eidesstaatliche Versicherung

Hiermit versichere ich, dass diese Abschlussarbeit von mir persönlich ver-fasst ist und dass ich keinerlei fremde Hilfe in Anspruch genommen habe. Ebenso versichere ich, dass diese Arbeit oder Teile daraus weder von mir selbst noch von anderen als Leistungsnachweise andernorts eingereicht wurden. Wört-liche oder sinngemäße Übernahmen aus anderen Schriften und Veröffentlichungen in gedruckter oder elektronischer Form sind gekennzeichnet. Sämtliche Sekun-därliteratur und sonstige Quellen sind nach- gewiesen und in der Bibliographie aufgeführt. Das Gleiche gilt für graphi-sche Darstellungen und Bilder sowie für alle Internet-Quellen. Ich bin ferner damit einverstanden, dass meine Arbeit zum Zwecke eines Plagiatsabgleichs in elektronischer Form anonymisiert ver-sendet und gespeichert werden kann. Mir ist bekannt, dass von der Korrektur der Arbeit abgesehen werden kann, wenn die Erklärung nicht erteilt wird.

Ort/Datum

Unterschrift

Literaturverzeichnis

1&1 IONOS SE (2020): Server-Side- und Client-Side-Scripting: Die Unterschiede, in: *IONOS Digitalguide*, [online] https://www.ionos.de/digitalguide/websites/web-entwicklung/server-side-und-client-side-scripting-die-unterschiede/ [06.01.2021].

About PageSpeed Insights (2018): in: *Google Developers*, [online] https://developers.google.com/speed/docs/insights/v5/about [21.10.2020].

Add a HERE Web Map into your React application (2020): in: *HERE Developer*, [online] https://developer.here.com/tutorials/react/ [25.10.2020].

Adobe (2020): Produkteinstellung von Adobe Flash Player, in: *adobe.com*, [online] https://www.adobe.com/de/products/flashplayer/end-of-life.html [25.11.2020].

Aktion Mensch e. V. (o. J.): Screenreader nutzen – Anleitung für NVDA: Einfach für Alle AccessBlog, in: *Einfach Für Alle Blog*, [online] https://www.einfach-fuer-alle.de/blog/id/2741/ [08.12.2020].

Angular - Angular Elements Overview (2020): in: *Angular*, [online] https://angular.io/guide/elements [06.01.2021].

Bemenderfer, Joshua (2020): Building Native Web Components with Vue.js, in: *DigitalOcean,* [online] https://www.digitalocean.com/community/tutorials/vuejs-custom-elements [06.01.2021].

Bidelman, Eric (2013): HTML's New Template Tag: standardizing client-side templating, in: *HTML5 Rocks - A resource for open web HTML5 developers*, [online] https://www.html5rocks.com/en/tutorials/webcomponents/template/ [13.10.2020].

Bidelman, Eric (2020a): Custom Elements v1: Reusable Web Components, in: *Google Developers*, [online] https://developers.google.com/web/fundamentals/web-components/customelements [13.10.2020].

Bidelman, Eric (2020b): Shadow DOM v1: Self-Contained Web Components, in: *Google Developers*, [online] https://developers.google.com/web/fundamentals/web-components/shadowdom [13.10.2020].

Boxhall, Alice/James Craig/Dominic Mazzoni/Alexander Surkov (2020): Accessibility Object Model, in: *GitHub*, [online] https://github.com/WICG/aom/blob/gh-pages/explainer.md [08.12.2020].

Boxhall, Alice/Dominic Mazzoni (2020): Can I Use Accessibility Object Model (AOM), in: *GitHub*, [online] https://github.com/WICG/aom/blob/gh-pages/caniuse.md [08.12.2020].

Braun, Herbert (2015): Kommentar: Weg mit Flash!, in: *heise online*, [online] https://www.heise.de/security/meldung/Kommentar-Weg-mit-Flash-2751583.html [25.11.2020].

Byron, Lee / Taras Mankovski / Rob Wormald (2017): The Strengths of Ember, Angular & React Explored, in: *InfoQ*, ab 29:50 [online] https://www.infoq.com/presentations/ember-angular-react/ [13.10.2020].

Cambridge University Press (Hrsg.) (o. J.): Dogfooding, *Cambridge Dictionary,* [online] https://dictionary.cambridge.org/de/worterbuch/.

Comparison with Other Frameworks (2020): in: *VueJS.org*, [online] https://vuejs.org/v2/guide/comparison.html [06.01.2021].

Components and Props (o. J.): in: *React*, [online] https://reactjs.org/docs/components-and-props.html [13.10.2020].

Composition vs Inheritance (o. J.): in: *React,* [online] https://reactjs.org/docs/composition-vs-inheritance.html [11.01.2021].

Create a New App (o. J.): in: *React*, [online] https://reactjs.org/docs/create-a-new-react-app.html [13.10.2020].

Create React App (o. J.): in: *create-react-app.dev*, [online] https://create-react-app.dev/ [13.10.2020].

CSS preprocessor - MDN Web Docs Glossary: Definitions of Web-related terms | MDN (2020): in: *MDN Web Docs*, [online] https://developer.mozilla.org/en-US/docs/Glossary/CSS_preprocessor [06.01.2021].

Custom Element Best Practices (2019): in: *Google Developers,* [online] https://developers.google.com/web/fundamentals/web-components/best-practices [25.10.2020].

Custom Elements Interop | Vue.js (2020): in: *VueJS.org*, [online] https://v3.vuejs.org/guide/migration/custom-elements-interop.html#_2-x-syntax [06.01.2021].

Deklarative Programmierung (2020): in: *IONOS Digital Guide,* [online] https://www.ionos.de/digitalguide/websites/web-entwicklung/deklarative-programmierung/ [13.10.2020].

Design Principles - React (o. J.): in: *React*, [online] https://reactjs.org/docs/design-principles.html [13.10.2020].

Deveria, Alexis (o. J. a): Can I use... Support tables for HTML5, CSS3, etc, in: *Can I use... Support tables for HTML5, CSS3, etc,* [online] https://caniuse.com/?search=components [13.10.2020].

Deveria, Alexis (o. J. b): Can I use :host-context ?, in: *Can I use... Support tables for HTML5, CSS3, etc,* [online] https://caniuse.com/?search=%3Ahost-context [02.11.2020].

Dodson, Rob (o. J.): Custom Elements Everywhere, in: *Custom Elements Everywhere*, [online] https://custom-elements-everywhere.com/#react [06.01.2021].

Dodson, Rob (2017): The future of accessibility for custom elements, in: *Rob Dodson,* [online] https://robdodson.me/the-future-of-accessibility-for-custom-elements/ [08.12.2020].

Domin, Andreas (2018): Library vs. Framework: Das sind die Unterschiede, in: *t3n – digital pioneers*, [online] https://t3n.de/news/library-vs-framework-unterschiede-1022753/ [13.10.2020].

facebook (2021): Network Dependents - facebook/react, in: *GitHub*, [online] https://github.com/facebook/react/network/dependents?package_id=UGFja2FnZS0xMzM2NDkxNg%3D%3D [12.01.2021].

Facebook Developers (2015): React.js Conf 2015 Keynote - Introducing React Native, in: *YouTube,* [online] https://www.youtube.com/watch?v=KVZ-P-ZI6W4&t=0s&list=PLb0IAmt7-GS1cbw4qonlQztYV1TAW0sCr&index=1 [13.10.2020].

Forms (o. J.): in: *React,* [online] https://reactjs.org/docs/forms.html [25.10.2020].

Hacker, Bernd (2020): Npm: here-js-api, in: npm, [online] https://www.npmjs.com/package/here-js-api [25.10.2020].

Handling Events (o. J.): in: *React,* [online] https://reactjs.org/docs/handling-events.html [13.10.2020].

Hellberg, Marcus (2017): Simplifying Performance with Web Components, in: *Vaadin*, [online] https://vaadin.com/blog/simplifying-performance-with-web-components [06.01.2021].

Hoffmann, Jay (2017): Flash And Its History On The Web, in: *The History of the Web,* [online] https://thehistoryoftheweb.com/the-story-of-flash/ [25.11.2020].

Høiberg, Simon (2020): React and Web Components - ITNEXT, in: *Medium*, [online] https://itnext.io/react-and-web-components-3e0fca98a593 [06.01.2021].

Holland, Martin (2017): Adobe verabschiedet sich von Flash: 2020 ist Schluss, in: *heise online,* [online] https://www.heise.de/newsticker/meldung/Adobe-verabschiedet-sich-von-Flash-2020-ist-Schluss-3783264.html [25.11.2020].

Introducing JSX (o. J.): in: *React,* [online] https://reactjs.org/docs/introducing-jsx.html [13.10.2020].

Jadhwani, Prateek (2019): Getting Started with Web Components: Build modular and reusable components using HTML, CSS and JavaScript, Birmingham, UK: Packt Publishing.

JavaScript Date Reference (o. J.): in: *W3Schools*, [online] https://www.w3schools.com/jsref/jsref_obj_date.asp [02.11.2020].

Jobs, Steve (2010): Thoughts on Flash, in: *Apple,* [online] https://web.archive.org/web/20200109015107/https://www.apple.com/hotnews/thoughts-on-flash/ [25.11.2020].

JQuery Introduction (o. J.): in: *W3Schools*, [online] https://www.w3schools.com/jquery/jquery_intro.asp [06.01.2021].
JSS (2020): in: cssinjs.org, [online] https://cssinjs.org/react-jss/?v=v10.5.0 [06.01.2021].

Kamboj, Sheeshpaul (2020): Web Components Basics and Performance Benefits - Sheeshpaul Kamboj, in: *Medium*, [online] https://medium.com/@spkamboj/web-components-basics-and-performance-benefits-f7537c908075 [06.01.2021].

Lifting State Up (o. J.): in: *React,* [online] https://reactjs.org/docs/lifting-state-up.html [13.10.2020].

Lighthouse | Tools for Web Developers (2020): in: *Google Developers,* [online] https://developers.google.com/web/tools/lighthouse [21.10.2020].

Lit-html (o. J.): in: *The Polymer Project,* [online] https://lit-html.polymer-project.org/ [13.10.2020].

McGinnis, Tyler (2016): Imperative vs Declarative Programming - ui.dev, in: *UI.dev*, [online] https://ui.dev/imperative-vs-declarative-programming/ [13.10.2020].

Microsoft (o. J.): Typed JavaScript at Any Scale., in: *typescriptlang.org*, [online] https://www.typescriptlang.org/ [06.01.2021].

Newest „reactjs" Questions (2021): in: *Stack Overflow*, [online] https://stackoverflow.com/questions/tagged/reactjs [12.01.2021].

OpenJS Foundation and other contributors (2020): ESLint - Pluggable JavaScript linter, in: *ESLint - Pluggable JavaScript linter*, [online] https://eslint.org/ [06.01.2020].

Otto, Mark/Bootstrap Contributors/Jacob Thornton (o. J.): About, in: *getbootstrap.com,* [online] https://getbootstrap.com/docs/4.5/about/overview/ [08.12.2020].

Page, Wilson (2015): The state of Web Components, in: *Mozilla Hacks – the Web developer blog*, [online] https://hacks.mozilla.org/2015/06/the-state-of-web-components/ [13.10.2020].

Pavic, Bojan / Chris Anstey / Jeremy Wagner (2020): Why does speed matter?, in: *web.dev,* [online] https://web.dev/why-speed-matters/ [21.10.2020].

Powell, Kevin (2019): Why I won't use ShadowDOM - simple examples, in: *Codepen,* [online] https://codepen.io/kevinmpowell/pen/JQwOyE [08.12.2020].

Rauber, Manuel (2020a): Native im Web ohne Framework, in: *Windows Developer*, Jg. 2020, Nr. 4, [online] https://kiosk.entwickler.de/windows-developer-magazin/windows-developer-magazin-4-2020-2/native-im-web-ohne-framework/.

Rauber, Manuel (2020b): Web Components auf Steroiden, in: *Windows Developer,* Jg. 2020, Nr. 7, [online] https://kiosk.entwickler.de/windows-developer-magazin/windows-developer-magazin-7-2020/web-components-auf-steroiden/.

React – A JavaScript library for building user interfaces (o. J.): in: *React,* [online] https://reactjs.org [13.10.2020].

Russell, Alex (2020): About Me, in: *Infrequently Noted*, [online] https://infrequently.org/about-me/#content [13.10.2020].

Russell, Alex (2011): Web Components and Model Driven Views by Alex Russell, in: *Fronteers,* [online] https://fronteers.nl/congres/2011/sessions/web-components-and-model-driven-views-alex-russell [13.10.2020].

SimilarTech (2020): Bootstrap Market Share and Web Usage Statistics, in: *SimilarTech*, [online] https://www.similartech.com/technologies/bootstrap [08.12.2020].

Springer, Sebastian (2020): Bausteine für das Web, in: *PHP Magazin*, Jg. 2020, Nr. 2, [online] https://entwickler.de/online/web/bausteine-fuer-das-web-komponentenbasierte-architektur-mit-web-components-579928139.html.

Stack Overflow (2019): Stack Overflow Developer Survey 2019, in: *Stack Overflow*, [online] https://insights.stackoverflow.com/survey/2019#technology-_-web-frameworks [13.10.2020].

State and Lifecycle (o. J.): in: *React,* [online] https://reactjs.org/docs/state-and-lifecycle.html [13.10.2020].

Stencil (o. J.): in: *stenciljs.com*, [online] https://stenciljs.com/ [13.10.2020].

Stencil - Introduction (o. J.): in: *stenciljs.com,* [online] https://stenciljs.com/docs/introduction [13.10.2020].

Stoll, Kathrin (2020): Web Components: So verwendest du Code plattformübergreifend wieder, in: *t3n – digital pioneers*, [online] https://t3n.de/news/web-components-verwendest-code-1200044/ [21.10.2020].

Sutton, Marcy (2014): Accessibility and the Shadow DOM | MarcySutton.com, in: *Marcy Sutton Blog,* [online] https://marcysutton.com/accessibility-and-the-shadow-dom/ [08.12.2020].

Template literals (Template strings) - JavaScript (2020): in: *MDN Web Docs*, [online] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals [06.01.2021].

Templates – LitElement (o. J.): in: *LitElement*, [online] https://lit-element.polymer-project.org/guide/templates [06.01.2021].

The Polymer Project (o. J.): @polymer/polymer Our original Web Component library, in: *webcomponents.org,* [online] https://www.webcomponents.org/element/@polymer/polymer [13.10.2020].

The W3C Technical Architecture Group (TAG) (o. J.): W3C Technical Architecture Group, in: *w3.org,* [online] https://www.w3.org/2001/tag/ [13.10.2020].

Thinking in React (o. J.): in: *React,* [online] https://reactjs.org/docs/thinking-in-react.html [13.10.2020].

Uncontrolled Components (o. J.): in: *React*, [online] https://reactjs.org/docs/uncontrolled-components.html [25.10.2020].

Using templates and slots (2020): in: *MDN Web Docs,* [online] https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_templates_and_slots [19.11.2020].

Virtual DOM and Internals (o. J.): in: *React*, [online] https://reactjs.org/docs/faq-internals.html [13.10.2020].
Web Components (o. J.): in: React, [online] https://reactjs.org/docs/web-components.html [06.01.2021].

Web framework rankings | HotFrameworks (o. J.): in: *HotFrameworks*, [online] https://hotframeworks.com/ [13.10.2020].

WebAIM (2019): WebAIM: Screen Reader User Survey #8 Results, in: *WebAIM,* [online] https://webaim.org/projects/screenreadersurvey8/#usage [08.12.2020].

webcomponents.org (o. J.): Resources - webcomponents.org, in: *webcomponents.org*, [online] https://www.webcomponents.org/resources [13.10.2020].

Zeigermann, Oliver / Nils Hartmann (2016): React: Die praktische Einführung in React, React Router und Redux, 1. Aufl., Heidelberg, Deutschland: Dpunkt.Verlag GmbH.

Abbildungsverzeichnis

Anhang

Anfahrt Web Component Boilerplate

```
let tmpl = document.createElement(‚template‘);
    class MapRoute extends HTMLElement {
    constructor() {
        super();
        const shadowRoot = this.attachShadow({mode: ‚open‘});
        shadowRoot.appendChild(mapTmp.content.cloneNode(true));
    }
        get origin() {
            return this.getAttribute(‚origin‘);
    }
        set origin(val) {
            this.setAttribute(‚origin‘, val);
    }
        get destination() {
            return this.getAttribute(‚destination‘);
    }
        set destination(val){
            this.setAttribute(„destination“,val);
    }
        get apikey(){
            return this.getAttribute(‚apikey‘);
    }
        set apikey(val){
            this.setAttribute(„apikey“,val);
    }
        get markerIcon(){
            return this.getAttribute(‚markerIcon‘);
    }
        set markerIcon(val){
            this.setAttribute(„markerIcon“,val);
    }
        get lineColor(){
            return this.getAttribute(‚lineColor‘);
    }
        set lineColor(val){
            this.setAttribute(„lineColor“,val);
    }
        get uiLayer(){
            return this.hasAttribute(‚uiLayer‘);
    }
```

```
        set uiLayer(val){
            this.setAttribute(„uiLayer“,val);
    }
        get mapLayer(){
            if(this.getAttribute(‚mapLayer‘)==“Satellite“){
                return „satellite“;
            }
            else{
                return „normal“;
            }
    }
        set mapLayer(val){
            this.setAttribute(„mapLayer“,val);}
    }
class MapInput extends HTMLElement {
    constructor() {
        super();
        const shadowRoot = this.attachShadow({mode: ‚open‘});
        shadowRoot.appendChild(tmpl.content.cloneNode(true));
    }
    get origin() {
        return this.getAttribute(‚origin‘);
    }
    set origin(val){
        this.setAttribute(„origin“,val);
    }}
window.customElements.define(‚map-route‘, MapRoute);
window.customElements.define(‚map-input‘, MapInput);
```

Anfahrt Web Component Anpassungsaufwand

```
static get observedAttributes() {
    return [‚origin‘, ‚destination‘];
}
get origin() {
    return this.getAttribute(‚origin‘);
}
set origin(val) {
    this.setAttribute(‚origin‘, val);
}
get destination() {
    return this.getAttribute(‚destination‘);
}
set destination(val){
    this.setAttribute(„destination“,val);
}
get apikey(){
    return this.getAttribute(‚apikey‘);
}
set apikey(val){
    this.setAttribute(„apikey“,val);
}
get markerIcon(){
    return this.getAttribute(‚markerIcon‘);
}
set markerIcon(val){
    this.setAttribute(„markerIcon“,val);
}
get lineColor(){
    return this.getAttribute(‚lineColor‘);
}
set lineColor(val){
    this.setAttribute(„lineColor“,val);
}
get uiLayer(){
    return this.hasAttribute(‚uiLayer‘);
}
set uiLayer(val){
    this.setAttribute(„uiLayer“,val);
}
get mapLayer(){
    if(this.getAttribute(‚mapLayer‘)==“Satellite“){
        return „satellite“;
    }else{
        return „normal“;
    }
}
```

```
set mapLayer(val){
    this.setAttribute(„mapLayer“,val);
}
if(!this.origin){
    let att = document.createAttribute(‚origin‘);
    att.value = „ „;
    this.setAttributeNode(att);
}
let type;
if(this.mapLayer ==“satellite“){
    type = defaultLayers.raster.satellite.map;
}else{
    type = defaultLayers.vector.normal.map;
}
if(this.uiLayer){
    var ui = H.ui.UI.createDefault(map, defaultLayers, „de-DE“);
}
let icon;
if(this.markerIcon){
    icon = new H.map.Icon(this.markerIcon);
}
let lineColor;
if(this.lineColor){
    lineColor = this.lineColor;
}else{
    lineColor= ‚black‘;
}
get origin() {
    return this.getAttribute(‚origin‘);
}
set origin(val){
    this.setAttribute(„origin“,val);
}
if(this.origin){
    this.shadowRoot.querySelector(„input“).value = this.origin;
}
```

Kalender Web Component Boilerplate

```
let tmpl = document.createElement(‚template‘);
class Calendar extends HTMLElement {
    constructor() {
        super();
        const shadowRoot = this.attachShadow({mode: ‚open‘});
        shadowRoot.appendChild(tmpl.content.cloneNode(true));
    }
    get mode() {
        return this.getAttribute(‚mode‘) || „month“;
    }
    set mode(val){
        this.setAttribute(‚mode‘,val);
    }
    get firstWeekDay() {
        return (this.getAttribute(‚first‘)?parseInt(this.getAttribu-
te(‚first‘)):1);
    }
    set firstWeekDay(val){
        this.setAttribute(‚first‘,val);
    }}
let evTempl = document.createElement(‚template‘);
    class Event extends HTMLElement{
    constructor(props){
        super(props);
        const shadowRoot = this.attachShadow({mode:‘open‘});
        shadowRoot.appendChild(evTempl.content.cloneNode(true));
    }
    get classes(){
        return this.getAttribute(„classes“)||““;
    }
    set classes(val){
        this.setAttribute(‚classes‘,val);
    }
    get weekday(){
        return this.getAttribute(„weekday“)||““;
    }
    set weekday(val){
        this.setAttribute(‚weekday‘,val);}
    }
class Info extends HTMLElement{
    constructor(props){
        super(props);
        const shadowRoot = this.attachShadow({mode:‘open‘});
        shadowRoot.appendChild(infoTempl.content.cloneNode(true));}
    }
window.customElements.define(‚calendar-event‘, Event);
window.customElements.define(‚event-info‘, Info);
window.customElements.define(‚event-calendar‘, Calendar);
```

Kalender Web Component Anpassungsaufwand

```
switch(this.mode){
    case „day“:
        this.startDate = new Date(tmp.getFullYear(), tmp.getMonth(), tmp.
getDate() + 1*factor);
        break;
    case „week“:
        this.startDate = new Date(tmp.getFullYear(), tmp.getMonth(), tmp.
getDate() + 7*factor);
        break;
    case „month“:
    default:
        this.startDate = new Date(tmp.getFullYear(), tmp.get-
Month()+1*factor);
        break;
}
get mode() {
    return this.getAttribute(‚mode‘) || „month“;
}
set mode(val){
    this.setAttribute(‚mode‘,val);
}
get firstWeekDay() {
    return (this.getAttribute(‚first‘)?parseInt(this.getAttribu-
te(‚first‘)):1);
}
set firstWeekDay(val){
    this.setAttribute(‚first‘,val);
}
switch(this.mode){
    case „day“:
        days.push(this.startDate);
        break;
    case „week“:
        let firstDay = new Date(this.startDate.getTime());
        while(firstDay.getDay()!== this.firstWeekDay){
            firstDay.setDate(firstDay.getDate() - 1);
        }
        while(days.length < 7){
            days.push(new Date(firstDay.getTime()));
            firstDay.setDate(firstDay.getDate()+1);
        }
        break;
    case „month“:
        let first = new Date(this.startDate.getFullYear(), this.startDa-
te.getMonth(), 1);
        while(first.getDay()!== this.firstWeekDay){
```

```
            first.setDate(first.getDate() - 1);
        }
        while(first.getMonth()===this.startDate.getMonth() || first.get-
Month()===(this.startDate.getMonth()>0?this.startDate.getMonth()-1:11)){
            days.push(new Date(first.getTime()));
            first.setDate(first.getDate()+1);
        }
        break;
    default:
        break;
}
dayClass += „ „ + this.mode + „ firstDay-“ + this.firstWeekDay;
```

Wetter Web Component Boilerplate

```
let tmpl = document.createElement(‚template‘);
    class Weather extends HTMLElement {
    constructor() {
        super();
        const shadowRoot = this.attachShadow({mode: ‚open‘});
        shadowRoot.appendChild(tmpl.content.cloneNode(true));}
        get lat(){
            return this.getAttribute(„lat“);}
        set lat(val){
            this.setAttribute(„lat“,val);}
        get lon(){
            return this.getAttribute(„lon“);}
        set lon(val){
            this.setAttribute(„lon“,val);}
        get apikey(){
            return this.getAttribute(„apikey“);}
        set apikey(val){
            this.setAttribute(„apikey“,val);}
        get units(){
            return this.getAttribute(„units“) || „metric“;}
        set units(val){
            this.setAttribute(„units“,val);}
        get lang(){
            return this.getAttribute(„lang“) || „en“;}
        set lang(val){
            this.setAttribute(„lang“,val);}
        get exclude(){
            return this.getAttribute(„exclude“) || [];}
        set exclude(val){
            this.setAttribute(„exclude“,val)}
get iconBase(){
    return this.getAttribute(„iconBase“)||“http://localhost:3030/“}
set iconBase(val){
    this.setAttribute(„iconBase“,val);}}
window.customElements.define(‚weather-forecast‘, Weather);
window.customElements.define(‚daily-forecast‘, Daily);
let weather = document.createElement(‚template‘);
class Daily extends HTMLElement {
    constructor() {
        super();
        const shadowRoot = this.attachShadow({mode: ‚open‘});
        shadowRoot.appendChild(weather.content.cloneNode(true));}}
```

Wetter Web Component Anpassungsaufwand

```
get lat(){
    return this.getAttribute(„lat“);}
set lat(val){
    this.setAttribute(„lat“,val);}
get lon(){
    return this.getAttribute(„lon“);}
set lon(val){
    this.setAttribute(„lon“,val);}
get iconBase(){
    return this.getAttribute(„iconBase“)||“http://localhost:3030/“}
set iconBase(val){
    this.setAttribute(„iconBase“,val);}
get apikey(){
    return this.getAttribute(„apikey“);}
set apikey(val){
    this.setAttribute(„apikey“,val);}
get units(){
    return this.getAttribute(„units“) || „metric“;}
set units(val){
    this.setAttribute(„units“,val);}
get lang(){
    return this.getAttribute(„lang“) || „en“;}
set lang(val){
    this.setAttribute(„lang“,val);}
get exclude(){
    return this.getAttribute(„exclude“) || [];}
set exclude(val){
    this.setAttribute(„exclude“,val);}
get exclude(){
    return this.getAttribute(„exclude“)  || „ „;
}
forecast.setAttribute(„exclude“, this.exclude);
get iconBase(){
    return this.getAttribute(„iconBase“);
}
forecast.setAttribute(„iconBase“, this.iconBase);
```

Anfahrt React Component Boilerplate

```
import React from „react“;
export default class MapRoute extends React.Component {
    constructor(props){
        super(props);
    }}
```

Anfahrt React Component Anpassungen

```
let layer;
if(this.props.mapLayer !== „Satellite“){
    layer =defaultLayers.vector.normal.map;
}else{
    layer = defaultLayers.raster.satellite.map
}
if(this.props.uiLayer){
    var ui = H.ui.UI.createDefault(map, defaultLayers, ‚de-DE‘);
}
this.icon = undefined;
if(this.props.icon){
    this.icon = new H.map.Icon(this.props.icon);
}
let icon = this.icon;
let startVal = „Startpunkt“;
if(this.props.origin){
    startVal = this.props.origin;
}
let color = this.props.lineColor!=undefined?this.props.lineColor:“blue“;
```

Kalender React Component Boilerplate

```
import React from „react“;
class Event extends React.Component{
    constructor(props){
        super(props);
    }}
export default class Calendar extends React.Component{
    constructor(props){
        super(props);
    }}
```

Kalender React Component Anpassungsaufwand

```
switch(this.props.mode) {
    case „day“:
        this.setState({startDate: new Date(tmp.getFullYear(), tmp.get-
Month(), tmp.getDate() + 1 * factor)});
        break;
    case „week“:
        this.setState({startDate: new Date(tmp.getFullYear(), tmp.get-
Month(), tmp.getDate() + 7 * factor)});
        break;
    case „month“:
    default:
        this.setState({startDate: new Date(tmp.getFullYear(), tmp.get-
Month() + 1 * factor)});
        break;
}
let firstWeekDay = (this.props.firstDay!==undefined?this.props.first-
Day:1);
switch(this.props.mode){
    case „day“:
        days.push(this.state.startDate);
        break;
    case „week“:
        let firstDay = new Date(this.state.startDate.getTime());
        while(firstDay.getDay()!== firstWeekDay){
            firstDay.setDate(firstDay.getDate() - 1);
        }
        while(days.length < 7){
            days.push(new Date(firstDay.getTime()));
            firstDay.setDate(firstDay.getDate()+1);
        }
        break;
    case „month“:
    default:
        let first = new Date(this.state.startDate.getFullYear(), this.
state.startDate.getMonth(), 1);
        while(first.getDay()!== firstWeekDay){
            first.setDate(first.getDate() - 1);
        }
        while(first.getMonth()===this.state.startDate.getMonth() ||
first.getMonth()===(this.state.startDate.getMonth()>0?this.state.startDa-
te.getMonth()-1:11)){
            days.push(new Date(first.getTime()));
            first.setDate(first.getDate()+1);
        }
        break;
}
let mode = this.props.mode!=undefined?this.props.mode:“month“;
```

Wetter React Component Boilerplate

```
import React from „react“;
class Daily extends React.Component{}
export default class Weather extends React.Component{
    constructor(props){
        super(props);
    }}
```

Wetter React Component Anpassungen

```
if(this.props.lat && this.props.lon && this.props.apikey) {
    let unit = (this.props.units ? this.props.units : „metric“);
    let lang = (this.props.lang ? this.props.lang : „en“);
}
let exclude = this.props.exclude!=undefined?this.props.exclude:““;
let iconBase = this.props.iconBase?this.props.iconBase:“http://local-
host:3030/“;
exclude={exclude}
iconBase={iconBase}
iconBase={iconBase}
exclude={exclude}
{!this.props.exclude.includes(„temperature“)?:““}
{!this.props.exclude.includes(„rain“)?:““}
{!this.props.exclude.includes(„wind“)?:““}
{!this.props.exclude.includes(„sun“)?:““}
```