

Webkomponenten zur Erstellung von Javascript basierenden Serveranwendungen

Andreas Hahn



BACHELORARBEIT

Nr. S1510238020-A

eingereicht am
Fachhochschul-Bachelorstudiengang

Medientechnik und -design

in Hagenberg

im Februar 2018

Diese Arbeit entstand im Rahmen des Gegenstands

Bachelorarbeit 1

im

Sommersemester 2017

Betreuung:

Rimbert Rudisch-Sommer

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Hagenberg, am 28. Februar 2018

Andreas Hahn

Inhaltsverzeichnis

Erklärung	iii
Kurzfassung	vi
Abstract	vii
1 Einleitung	1
1.1 Motivation	1
1.2 Theoretischer Hintergrund und Stand der Forschung	1
1.3 Forschungsfrage	2
1.4 Methodik	2
1.5 Erwartete Ergebnisse	2
1.6 Aufbau der Arbeit	3
2 Technische Grundlagen	4
2.1 Webkomponenten	4
2.1.1 Custom-Elements	4
2.1.2 HTML Import	5
2.1.3 HTML Templates	5
2.1.4 Shadow DOM	5
2.2 Representational State Transfer API	6
2.2.1 Restriktionen	6
2.2.2 Praxis	7
3 State of the Art	8
3.1 Komponentenorientierte Javascript-Frameworks und -Bibliotheken	8
3.1.1 Angular	9
3.1.2 Polymer	10
3.1.3 React	11
3.1.4 Vue	12
3.2 MEAN Stack ¹	12
4 Grundlegendes Konzept	14
4.1 Aufbau einer Webanwendung	14
4.2 Verwendung von Komponenten im Web	15

¹<http://mean.io/>

4.3	Eigene Idee, technisches Design	16
4.4	Anforderung an die komponentenorientierte Webanwendung	16
4.4.1	Darstellung	17
4.4.2	Logik	17
4.4.3	Datenverwaltung	17
5	Implementierung der Webanwendung	18
5.1	Umsetzung mit Dependencies Electron / Scram-Engine	18
5.1.1	Electron	18
5.1.2	Scram-Engine	18
5.2	Implementierung der Komponenten	19
5.2.1	Frontend	19
5.2.2	Server	19
5.2.3	Datenverwaltung	21
6	Evaluierung	24
6.1	Nutzen	24
6.2	Flexibilität	24
6.3	Wiederverwendbarkeit	24
6.4	Testbarkeit	25
7	Zusammenfassung	26
7.1	Lernkurve	26
7.2	Leistung	26
7.3	Markup	26
7.4	Prototyp	27
	Quellenverzeichnis	28
	Literatur	28
	Online-Quellen	28

Kurzfassung

Webkomponenten tragen zur Atomisierung der Webprogrammierung bei, da diese den WebentwicklerInnen die Möglichkeiten bieten, bestehende HTML Elemente zu erweitern und eigene zu kreieren. Dieser unscheinbarer Zusatz ermöglicht ProgrammiererInnen ihre Software aus mächtigen, zusammengesetzten Komponenten zu erstellen. Diese Bachelorarbeit konzipiert ein Gerüst aus Webkomponenten in Kombination mit anderen gängigen Webtechnologien, welche die Erstellung einer Serveranwendung weitgehend mit Webkomponenten ermöglicht und testet dieses anhand eines für diese Arbeit entwickelten Prototypen.

Abstract

Webcomponents aim towards atomarizing web programming by giving developers the possibility to extend already existng HTML elemnts and to create own components. This modest addition enables programmers to build their software out of combined, powerfull components. This bachelor thesis focuses on the development of a framework of web components in combination with other popular web technologies to develop a server application using webcomponents. This framework will then be evaluated based on a prototype implemented for this project.

Kapitel 1

Einleitung

1.1 Motivation

Die Idee, die Webprogrammierung komponentenorientiert zu gestalten, gibt es seit 1998 [13]. Jedoch hat sich keine dieser Ideen bis heute durchgesetzt. Seit einigen Jahren gibt es dazu wieder Ansätze, welche von größeren Firmen, wie beispielsweise Google mit Angular¹ oder Facebook mit React², mittels eigen entwickelten komponentenbasierten Frameworks versuchen, die Webentwicklung voran zu treiben. Allerdings ist seit neuestem die grundlegende Idee der nativen Webkomponenten³, welche künftig nur als Webkomponenten bezeichnet werden, zurück. Da diese, sobald gängige Webbrowser alle Technologien von Webkomponenten – HTML Imports, Templates, Custom Elements, Shadow DOM – unterstützen, einen weiteren Schritt in standardisierte, wiederverwendbare und unabhängige Webbausteine bedeutet. Zur Zeit ist es noch nicht, oder erschwert möglich, clientseitig und serverseitig auf Webkomponenten zu setzen. Diese Bachelorarbeit beleuchtet jene Problematik und stellt Lösungsvorschläge anhand einer konkreten CRUD-Anwendung dar.

1.2 Theoretischer Hintergrund und Stand der Forschung

Eine CRUD-Anwendung (Create, Read, Update, Delete) ist eine Anwendung, welche die 4 grundlegenden Operationen implementiert, um Daten zu manipulieren. BenutzerInnen können durch eine Benutzeroberfläche Datensätze erstellen, lesen, bearbeiten und löschen. Angular oder React wären für die Umsetzung einer solchen Anwendung geeignet. Aus heutiger Sicht zählen diese Technologien zu den gängigsten Frameworks und werden von zahlreichen EntwicklerInnen stetig verbessert. Diese besagten Frameworks sind mit einer steilen Lernkurve behaftet und benötigen Zeit um sich damit vertraut zu machen.

Im Gegensatz dazu sind Webkomponenten logisch deklariert und bauen lediglich auf Kenntnissen von nativem HTML, CSS und Javascript auf. Die Unterstützung wird noch nicht von jedem Browser gewährleistet und verursacht aus diesem Grund ein Zögern bei

¹<https://angular.io/>

²<https://facebook.github.io/react/>

³<https://www.webcomponents.org/>

einigen EntwicklerInnen, um komplett auf Webkomponenten zu setzen. Google hat mit Polymer⁴ eine Javascript-Bibliothek geschaffen, welche die Kreierung und Nutzung von Webkomponenten ungemein vereinfacht. Darüber hinaus nutzt diese Bibliothek Polyfills, die Funktionen, welche Polymer benötigt, bei Bedarf liefert und wird dadurch nicht nur in allen gängigen Webbrowsern – Chrome, Firefox, Edge, Safari – sondern auch in deren älteren Versionen, die die notwendigen Funktionen nicht implementiert haben, unterstützt [6]. Native Unterstützung ist bereits bei den meisten Browserherstellern in Entwicklung und würde Polyfills überflüssig machen.

1.3 Forschungsfrage

Aus diesen Ansätzen ergibt sich folgende Forschungsfrage für diese Bachelorarbeit:

Wie muss ein Webanwendung programmatisch konzipiert werden, um die grundlegenden Funktionen weitgehend mittels server- und clientseitigen Webkomponenten zu realisieren?

1.4 Methodik

Um diese Frage zu beantworten, soll die Bachelorarbeit als eine Kombination von Literaturarbeit und praktischer bzw. prototypischer Umsetzung realisiert werden.

Zunächst soll aus bestehender Literatur (erweiternd zu Abschnitt 1.2) erörtert werden, wie mit dem Thema der Webkomponenten und mit komponentenorientierte Frameworks umgegangen wird. Gemeinsame Faktoren sollen daraus extrahiert werden und als Grundlage für ein eigenes, theoretisches Konzept dienen. Die Vorteile und Limitierungen von Webkomponenten werden dabei erörtert. Dieses Konzept soll schlussendlich eine Liste von Kerntechniken enthalten, welche die Umsetzung einer CRUD-Anwendung mittels Webkomponenten ermöglicht.

Überprüft soll die Anwendbarkeit dieses Konzepts, durch einen eigenen, im Rahmen dieser Arbeit entwickelten Prototypen, werden.

1.5 Erwartete Ergebnisse

Als konkretes Ergebnis wird ein Gerüst mit Webkomponenten erstellt, welches als Grundlage für die Erstellung einer Webanwendung dienen soll. Es wird erwartet, dass sich solche konkreten Techniken finden und umsetzen lassen.

Bei der praktischen Umsetzung der Webkomponenten in einer CRUD-Anwendung wird ebenfalls eine positive Evaluierung erwartet, da es bereits erfolgreiche Konzepte basierend auf server- und clientseitigen Webkomponenten gibt und somit auf deren Erfahrungen aufgebaut werden kann.

⁴<https://www.polymer-project.org/>

1.6 Aufbau der Arbeit

Im ersten Teil dieser Arbeit werden technische Grundlagen bezüglich Webkomponenten erörtert, und wie eine REST-Architektur aufgebaut ist. Im dritten Kapitel werden State of the Art Technologien besprochen. Im Anschluss wird das grundlegende Konzept der praktischen Arbeit erläutert. Im fünften Kapitel wird die konkrete Implementierung der Webanwendung behandelt. Im darauffolgenden Abschnitt – Evaluierung – werden objektiv die Vor- und Nachteile, der im Fokus stehenden Technologie, aufgezeigt. Im siebten und letzten Kapitel wird zusammenfassend die persönliche Meinung, bezüglich der Thematik, einfließen.

Kapitel 2

Technische Grundlagen

2.1 Webkomponenten

Wie der Name – Webkomponente – suggeriert, ermöglicht diese Technologie die Webentwicklung in kleinere, wiederverwendbare, modulare Container zu „komponentisieren“. Der klare Vorteil besteht darin, dass diese Komponenten, unabhängig von einem Framework, vollständig mit „vanilla“ HTML, CSS und Javascript kreiert werden können, und wie in Kap. 4.2 erwähnt, Code wartbar machen.

Webkomponenten bestehen aus 4 separaten W3C Standards¹:

- Custom Elements - APIs um neue HTML Elemente zu definieren
- HTML Imports - deklarative Methode um HTML Dokumente in andere zu importieren
- HTML Templates - `<template>`-Element, welches Dokumenten eigenen DOM ermöglicht
- Shadow DOM - DOM und Styling abkapseln

An einer Standardisierung der Webkomponenten wird bereits seitens der W3C gearbeitet. Mehr dazu findet sich in [16]. Jedoch um diese derzeit in allen größeren Browsern zu nutzen, gibt es sogenannte Polyfills [18]. Dieses Polyfill ist ein Codestück, welches Technologie zur Verfügung stellt, die der Browser nicht nativ unterstützt.

2.1.1 Custom-Elements

Custom-Elements gibt den EntwicklerInnen die Fähigkeit, eigene, voll funktionsfähige DOM-Elemente zu erstellen, existierende HTML-Tags zu erweitern und/oder von anderen EntwicklerInnen erstellte Elemente zu nutzen.

Diese Spezifikation ist Teil eines größeren Projekts, das Internet zu rationalisieren, im Hinblick auf EntwicklerInnen zugänglichen Schnittstellen, wie beispielsweise die Custom-Elements-Definition. Jedoch gibt es immer noch Limitierungen, welche das komplette Potenzial – semantisch und funktional – einschränken, um das Verhalten bestehender HTML-Elemente vollständig, mit selbst erstellten Elementen, zu beschreiben. Mehr dazu findet sich in [19].

¹https://www.w3.org/standards/techs/components#w3c_all

Programm 2.1: Exemplarische Definition eines Custom-Elements

```
1 class HelloWorld extends HTMLElement {...};  
2 customElements.define("hello-world", HelloWorld);
```

2.1.2 HTML Import

HTML Importe ermöglichen das Inkludieren und Wiederverwenden von HTML-Dokumenten in anderen Dokumenten. Dieser Import beinhaltet all jene Elemente, die in einem HTML-Dokument definiert sind (Javascript, CSS, HTML, ...). Diese Möglichkeit, mehrere Technologien in einem Import, sozusagen eine URL, zu bündeln, ist klein, aber wesentlich. Aktuell wird es von kaum einem Browser, außer Chrome 62, unterstützt². Mehr über *HTML Imports* in [17].

Programm 2.2: Exemplarischer HTML Import

```
1 <link rel="import" href="hello-world.html">
```

2.1.3 HTML Templates

Das Template Element kapselt Bestandteile einer Webseite ab, welche durch Skripte geklont und zur Laufzeit in das Dokument eingefügt werden können. Die Darstellung des Elements ist beim Laden und erstmaligen rendern der Seite leer, und wird zur Laufzeit mit Javascript gerendert. Dieses Element kann als Inhaltsplatzhalter angesehen werden, welcher für spätere Verwendung gespeichert wird.

Programm 2.3: Exemplarisches HTML Template

```
1 <template id="template">  
2   <p>Hello World!</p>  
3 </template>
```

Das p-Element, wie im Programm 2.3 gezeigt, ist kein Kindelement des Template Elements im DOM. Es ist ein Kind des DocumentFragment, welches vom content IDL Attribut des Templates zurückgegeben wird, wie in [20] erläutert.

2.1.4 Shadow DOM

Der Shadow Dom bietet eine API zur Erstellung eines separaten Baumes, Shadow Tree genannt, welcher an ein Element angehängt werden kann. Das bringt die Abkapselung von DOM-Elementen und CSS mit sich. Beispielsweise kann unorganisierter CSS-Code von einem Bereich in den anderen durchsickern und wiederum einen Konflikt erzeugen, welcher unbeabsichtigtes Verhalten hervorbringt.

²<https://caniuse.com/#search=HTML%20Import>

Shadow DOM löst dieses Problem durch Isolierung der CSS-Bereiche und Abkapselung des DOM. Gemeinsam mit der Custom-Element-API ermöglicht es das Schreiben von Komponenten mit in sich geschlossenem HTML, CSS und Javascript.

Um den Shadow DOM eines Elements zu erstellen muss diesem lediglich der Shadow Root angehängt werden.

Programm 2.4: Exemplarisches HTML Template

```
1 const p = document.createElement('p');  
2 const shadowRoot = p.attachShadow({mode: 'open'});
```

2.2 Representational State Transfer API

In diesem Abschnitt wird der Begriff der Representational State Transfer (REST) Architektur, welcher von Roy Thomas Fielding im Jahre 2000 eingeführt wurde, beschrieben [1]. Der als REST bezeichnete Architekturansatz beschreibt die grundlegenden Regeln, wie verteilte Systeme miteinander kommunizieren können. Fielding hat für diese Architektur Restriktionen zusammengefasst, welche in dem folgendem Abschnitt erläutert werden.

2.2.1 Restriktionen

Null Stil

Der Null Stil beschreibt den Status eines Systems, wenn jenes keine Abgrenzungen der enthaltenen Komponenten aufweist. Von diesem Status weg wird REST definiert.

Client-Server

Nach dem Client-Server-Architektur Stil werden Aufgabenbereiche auf Client und Server aufgeteilt. Meist wird vom Client eine Aufgabe gestellt, und diese vom Server verarbeitet.

Zustandslosigkeit

Jegliche REST-Zugriffe sind zustandslos. Das bedeutet, jede Anfrage enthält alle benötigten Informationen, die der Endpunkt braucht, um diese zu verstehen und zu verarbeiten. Unabhängig davon, wer oder was die Anfrage getätigt hat.

Einheitliche Schnittstelle

Eine standardisierte Schnittstelle soll durch die Trennung der Zuständigkeiten die Einfachheit fördern, sowie die darunter liegende Implementierung und Kommunikation verbergen.

Caching

Die effizienteste Netzwerkanfrage ist jene, die das Netzwerk nicht benutzt. Soll heißen, dass eine wiederverwendbare, gecachte Antwort die Performanteste ist.

Stufen Systeme

Das Zielsystem soll, wie in Kap. 4.1 weiter behandelt, mit n-Stufen aufgebaut sein, damit der Anwender lediglich auf eine Schnittstelle, einer Stufe, zugreifen muss, und die anderen verborgen bleiben können.

2.2.2 Praxis

In der Praxis wird das REST-Paradigma bevorzugt per HTTP/S realisiert. Services werden über eine URL/einen URI angesprochen. Wie eine Anfrage bearbeitet werden soll, kann per HTTP-Methode (GET, POST, PUT, etc.) verfeinert werden.

Kapitel 3

State of the Art

Um heutzutage eine vollständige Webanwendung – Datenspeicher, Logik und Darstellung – zu erstellen, gibt es unterschiedlichste Herangehensweisen. Um diesen Prozess zu vereinfachen, wird in den meisten Fällen ein Framework, wie in Kap. 3.1 erläutert, verwendet. Javascript Frameworks werden bevorzugt für das Frontend herangezogen. Das Backend kann, unabhängig vom Frontend, in jeder beliebigen Programmiersprache umgesetzt werden (Javascript, PHP, Python, ...), aber auch hier bietet die Nutzung eines Frameworks Vorteile. Um die Entwicklungstechnologien für vollständige Anwendungen kompakt zu halten, gibt es sogenannte Softwarestacks, welche Technologien zu einer Plattform vereinen und die Interoperabilität dieser gewährleisten. Mehr dazu in Kap. 3.2.

3.1 Komponentenorientierte Javascript-Frameworks und -Bibliotheken

Sogenannte Javascript Frameworks oder Bibliotheken – der größte Unterschied liegt darin, dass eine Anwendung die Bibliothek einbindet und ein Framework die Anwendung an sich – sind Werkzeuge, um den EntwicklerInnen ein Grundgerüst an Funktionalität zur Verfügung zu stellen. Ein wichtiger Aspekt ist dabei auch die Effizienz und Verwertbarkeit von erfahrenen ProgrammiererInnen entworfenen Code [5]. Eine konkrete Definition eines Frameworks findet sich in [3]:

A framework is a semi-complete application. A framework provides a reusable, common structure to share among applications. Developers incorporate the framework into their own application and extend it to meet their specific needs. Frameworks differ from toolkits by providing a coherent structure, rather than a simple set of utility classes.

Im Anschluss befindet sich eine Auflistung von solchen, zu diesem Zeitpunkt populären Javascript Frameworks und Bibliotheken¹. Diese arbeiten alle zum Großteil mit Komponenten im Frontend.

- Angular

¹<http://www.npmtrends.com/angular-vs-react-vs-vue-vs-@angular/core>

- Polymer
- REACT
- Vue.js

Alle aufgelisteten Frameworks und Bibliotheken sind verfügbar unter der MIT-Lizenz und werden im Anschluss genauer behandelt.

3.1.1 Angular

Angular ist ein TypeScript basierendes Framework, entwickelt und gewartet von Google, beschrieben als „Superheroic JavaScript MVW Framework“. Angular (auch „Angular 2+“, „Angular 2“ oder „ng2“) ist der von Grund auf neu geschriebene, größtenteils inkompatible Nachfolger von AngularJS (auch „Angular.js“ oder „AngularJS 1.x“). AngularJS wurde im Oktober 2010 veröffentlicht und bekommt weiterhin bug-fixes, etc.. Das neue Angular wurde im September 2016 als Version 2 veröffentlicht. Die aktuellste Version ist 4.3.6 (Stand 23. August 2017). Die Versionsnummer 3 wurde übersprungen, da eines der NPM-Pakete von Angular 2 bereits die Version v3.3.0 trug².

Angular Komponente

Das Kernkonzept von Angular ist die Komponente. Eine komplette Angular-Anwendung kann als Baum, bestehend aus solchen Komponenten, modelliert werden. Die Definition laut der offiziellen Angular-Dokumentation [7]:

A component controls a patch of screen called a view. You define a component's application logic—what it does to support the view—inside a class. The class interacts with the view through an API of properties and methods.

Programm 3.1: Angular Komponente

```
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'hello-world',
5    template: '<p>Hello, {{text}}!</p>',
6  })
7  export class HelloComponent {
8    text: string;
9    constructor() {
10      this.text = 'World';
11    }
12  }
```

Angewendet wird diese Komponente wie folgt:

```
<hello-world></hello-world>
```

²<http://angularjs.blogspot.co.at/2016/12/ok-let-me-explain-its-going-to-be.html>

Was folgende Ausgabe rendert:

```
<p>Hello World</p>
```

3.1.2 Polymer

Polymer (auch „Polymer-Project“) ist eine Open Source Javascript-Bibliothek zur Erstellung von Webanwendungen mit Webkomponenten. Erstmals erschienen am 29. Mai 2015 und wird – ebenso wie Angular – von Google entwickelt. Der letzte stabile Release (Stand 2. Februar 2018) ist die Version 2.4.0.

Polymer Komponente

Polymer baut auf der Custom-Element Spezifikation auf und fügt dem eine Reihe an Features hinzu wie beispielsweise [14]:

- Instanzmethoden für allgemeine Aufgaben
- Automatisierung für die Handhabung von Eigenschaften und Attributen, z. B. das Festlegen einer Eigenschaft basierend auf dem entsprechenden Attribut.
- Erstellen von Shadow DOM Bäumen für Elementinstanzen basierend auf einem `<template>`.
- Ein Datensystem, welches Datenbindung, Observers für Eigenschaftenänderungen und berechnete Eigenschaften unterstützt.

Programm 3.2: Polymer Komponente

```
1  <link rel='import' href='bower_components/polymer/polymer.html'>
2  <dom-module id='hello-world'>
3  <template>
4  <p>Hello {{text}}</p>
5  </template>
6  <script>
7  class HelloWorld extends Polymer.Element {
8    static get is() {
9      return 'hello-world';
10   }
11   constructor() {
12     super();
13     this.text = 'World';
14   }
15 }
16 customElements.define(HelloWorld.is, HelloWorld);
17 </script>
18 </dom-module>
```

Angewendet wird diese Komponente wie folgt:

```
<link rel='import' href='hello-world.html'>
<hello-world></hello-world>
```

Was folgende Ausgabe rendert:

```
<p>Hello World</p>
```

3.1.3 React

React (auch „React.js“ oder „ReactJS“) wird als „JavaScript Bibliothek zur Entwicklung von User-Interfaces“³ bezeichnet. Es wurde erstmals im März 2013 von Facebook veröffentlicht – zuvor nur firmenintern verwendet – und wird seither entwickelt und gewartet. Die neuste Version (Stand 26. September 2017) ist 16.0.0.

React Komponente

Aus der Sicht von React sind Komponenten ident mit Javascript Funktionen. Sie können beliebig viele Inputs („props“ genannt) besitzen und geben React Elemente zurück, die beschreiben, was am Bildschirm ausgegeben werden soll. Die Definition entsprechend der offiziellen React Dokumentation [8]:

Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.

Programm 3.3: React Komponente

```
1  import React, {Component} from "react";
2
3  class HelloWorld extends Component {
4    text = '';
5    constructor(props) {
6      super(props);
7      this.text = 'World';
8    }
9
10   render() {
11     return <p>Hello {this.text}</p>;
12   }
13 }
14 export default HelloWorld;
```

Angewendet wird diese Komponente wie folgt:

```
import Hello from './HelloWorld';
<Hello/>
```

Was folgende Ausgabe erzeugt:

```
<p>Hello World</p>
```

³<https://reactjs.org/blog/2013/06/05/why-react.html>

3.1.4 Vue

Vue (englische Aussprache [/vju/]), auch „Vue.js“ genannt, beschreibt sich selbst als ein „progressives Framework zur Entwicklung von User-Interfaces“⁴. Vue wurde im Februar 2014 vom Ex-Google-Mitarbeiter Evan You veröffentlicht. Ein markantes Merkmal von Vue ist, dass es hauptsächlich von einer einzelnen Person, ohne die Stütze einer großen Firma, erschaffen wurde. Nur ein paar EntwicklerInnen arbeiten für Evan an der Weiterentwicklung des Frameworks. Die aktuellste Version ist 2.5.13 (Stand 2. Februar 2018).

Vue Komponente

Die Definition laut der Vue Dokumentation [9]:

Components are one of the most powerful features of Vue. They help you extend basic HTML elements to encapsulate reusable code. At a high level, components are custom elements that Vue's compiler attaches behavior to. In some cases, they may also appear as a native HTML element extended with the special `is` attribute.

Programm 3.4: Vue Komponente

```
1  Vue.component('hello-world', {
2    template: '<p>Hello {{ text }} </p>',
3    data: function () {
4      return {text:"World"};
5    }
6  })
7
8  new Vue({
9    el: '#container'
10 })
```

Angewendet wird diese Komponente wie folgt:

```
<div id="container">
  <hello-world></hello-world>
</div>
```

Was folgende Ausgabe erzeugt:

```
<div id="container">
  <p>Hello World </p>
</div>
```

3.2 MEAN Stack⁵

Der MEAN Stack ist ein Javascript Software Stack, um Webanwendung zu bauen. MEAN ist ein Akronym für MongoDB, Express.js, Angular und Node.js. Node.js ist

⁴<https://vuejs.org/v2/guide/>

⁵<http://mean.io/>

eine Ausführungsumgebung um Javascript serverseitig auszuführen und als Servertechnologie zu nutzen. Die Architektur ermöglicht asynchronen Input und Output. Um Serveranwendungen vereinfacht entwickeln zu können, wird Express als Framework für Node genutzt. Angular dient als Frontend Framework und wurde bereits in Kap. 3.1.1 behandelt. MongoDB ist eine dokumentenorientierte NoSQL-Datenbank [2]. Diese speichert Daten im binären JSON Format ab und kann ebenso Javascript Code ausführen.

Durch die durchgängige Sprache wird der Datenaustausch zwischen Server und Client primitiv gehalten und erspart Kontextwechsel, welche beim Verwenden von unterschiedlichen Programmiersprachen entstehen. Da das Umdenken in den verschiedenen Sprachen wegfällt, werden nicht zwingend Spezialisten in jedem Bereich benötigt und es können so Anwendungen von weniger EntwicklerInnen realisiert werden.

Kapitel 4

Grundlegendes Konzept

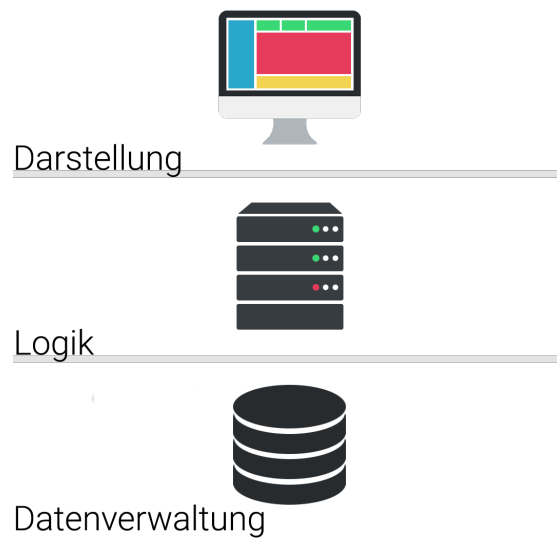
4.1 Aufbau einer Webanwendung

Das Konzept der Schichtenarchitektur ist in der Softwareentwicklung eine Client-Server-Architektur, welche die Darstellung, Anwendungslogik und Datenverwaltung physikalisch trennt. Das Konzept der abgestuften Architektur bei der Anwendungsentwicklung lässt sich von einem 1-Stufen-Ansatz bis hin zu einem n-Stufen-Ansatz einteilen. Jeder dieser Ansätze hat seine Vorzüge, doch für eine webbasierte Anwendung eignet sich der n-Stufen-Ansatz. Das Grundkonzept einer solchen Architektur besteht darin, eine Anwendung in unabhängige Ebenen mit definierten Rollen aufzuteilen. Die mehrstufige Anwendungsarchitektur ermöglicht es den EntwicklerInnen flexible, wiederverwendbare und austauschbare Anwendungskomponenten zu erstellen, welche zu einer Schichtenanwendung verknüpft werden können. Diese mögen sich auf derselben Maschine befinden, werden aber meist auf unterschiedliche ausgelagert [4].

Beispielsweise sind bei dem 1-Stufen-Ansatz alle für die Anwendung relevanten Funktionen – Darstellung, Logik, Datenverwaltung, etc.– auf der Clientmaschine vereint. Durch die Limitierung, dass die komplette Software auf einem Gerät betrieben wird, und die daraus folgenden Skalierungsprobleme, ist dieser Ansatz kaum für Webanwendungen geeignet. Der gebräuchlichste Stufenansatz ist der 3-stufige, siehe Abb. 4.1, welcher in der Regel die Darstellung, Anwendungslogik und Datenverwaltung entkoppelt. Ein Webbrowser, zur Anzeige der Benutzeroberfläche, wäre zum Beispiel die erste Stufe. Ein Webserver, welcher die Logik ausführt, repräsentiert die Zweite. Und als dritte Stufe, zur Datenverwaltung, käme eine Datenbank infrage.

Obwohl der 3-Stufen-Ansatz die Skalierbarkeit erhöht und eine Trennung der Anwendungslogik von den Anzeige- und Datenverwaltungsschichten einführt, trennt er die Anwendung nicht wirklich in spezialisierte Schichten. Für Prototyp- oder einfache Webanwendungen kann eine dreistufige Architektur ausreichen. Bei komplexen Anforderungen an solche Anwendungen ist ein 3-stufiger Ansatz jedoch in mehreren Schlüsselbereichen, einschließlich Flexibilität und Skalierbarkeit, unzureichend, und es wäre sinnvoll, einen n-Stufen-Ansatz zu wählen [11].

Der größte Nachteil von diesen Ansätzen ist der zusätzliche Overheadaufwand und eine höhere Latenz, was wiederum die der NutzerInnen wahrgenommen Geschwindigkeit mindert.

**Abbildung 4.1:** 3-Stufen-Architektur

Die Trennung von Logik ist in der Webentwicklung ein wesentliches Konzept. Wie in diesem Kapitel die Abstraktion durch Stufen, oder Ebenen, wird im folgenden Kapitel 4.2 die Trennung von Logik, mittels Komponenten, behandelt.

4.2 Verwendung von Komponenten im Web

Das Web besteht aus Bausteinen, sogenannten Elementen. Wenn EntwicklerInnen beispielsweise einen Link mittels einem `a`-Tag nutzen, wird erwartet, dass dieser sich wie ein Link-Element verhält. Dieses Element hat seine standardmäßige blaue Farbe, einen Handzeiger, wenn sich die Maus über diesem befindet und vor allem die Funktionalität, dass bei einem Klick auf das Element zu einer neuen URL weitergeleitet wird. Dieses Verhalten und Styling wird ohne jegliches Einwirken der EntwicklerInnen bereitgestellt. Jedes HTML-Element funktioniert nach diesem Prinzip, welches HTML Code einfacher zu schreiben und verständlicher macht. Durch Kombination solcher Elemente, mit eigens definierten Stylesheets können komplexe Gerüste aus HTML-Elementen, mit komplexem CSS/Javascript entstehen.

Damit EntwicklerInnen nicht die komplette Struktur im Kopf behalten müssen, und bei Problemen nur diese, das aufgetretene Problem beheben können, hat es sich bewährt, Code in kleine, überschaubare Teile herunter zu brechen. In Einheiten, welche das gesamte Gerüst wartbar machen. Diese Teile können einfachere Funktionen, Module, Komponenten oder viele andere Ansätze sein, welche Systeme in atomare Einzelteile aufteilen [15].

Komponentenbasierende UI-Bibliotheken waren der Standard, um komplexe Anwendungen zu bauen und moderne Webframeworks wie Angular oder Vue ermöglichen den EntwicklerInnen die Nutzeroberfläche aus wiederverwendbaren Blöcken zusammenzusetzen. Der große Unterschied zwischen diesen Frameworks und Webkomponenten ist, dass Webkomponenten die Komponentisierung auf dem DOM-Level durchführen, sodass

Custom-Elemente ohne ein Framework, wie Standard HTML-Elemente genutzt werden können.

Um also die Definition der, in dieser Arbeit fokussierten Technologie – einer Webkomponente – zusammenzufassen: Eine entkoppelte Sammlung von Funktionalität oder Prozessen und Logik, mit einer verständlichen Schnittstelle oder API um der Komponente Funktionalität abzurufen.

4.3 Eigene Idee, technisches Design

Durch den Webkomponenten-Standard wird es möglich sein, Komponenten ausschließlich zur Erstellung einer Webanwendung zu nutzen. Um die Grenzen der Nutzung von herkömmlichen Webkomponenten zu überschreiten, und die Verwendung auszuweiten, entwickelte sich die Idee, Webkomponenten gänzlich, oder Teile des Standards, zum Schreiben von CRUD-Anwendungen, und wie in Kap. 4.1 erwähnt, die einzelnen Schichten durch Webkomponenten zu realisieren, oder mit den Ebenen durch diese Technologie zu kommunizieren. Sozusagen sollen die Schichten – Darstellung, Logik, Datenverwaltung – , welche in Abb. 4.1 gezeigt sind, mit Webkomponenten realisiert werden.

Darstellung: Für diese gibt es bereits konkrete Ansätze und zahlreiche, vorgefertigte Komponenten zur Wiederverwendung¹.

Logik: Die Umsetzung der Logik mit Webkomponenten ist noch nicht zur Gänze möglich, da diese im Grunde „nur“ eine abgekapselte Schicht aus Custom-Elements auf reinen Javascript Code setzen. Ebenso abhängig ist die Realisierung von dem gewählten Technologie Stack. Da das Web bevorzugt Javascript unterstützt, wird auch eine reine Javascript Serverumgebung, beispielsweise Node.js², empfohlen. Jedenfalls gibt es durch die Scram-Engine³, welche in Kap. 5.1.2 genauer behandelt wird, die Möglichkeit, HTML mit dessen Javascript serverseitig, außerhalb des Browsers, zu parsen, auszuführen und zu rendern. Dadurch werden das Aufsetzen und das Betreiben von Servern, durch die Nutzung von Webkomponenten, ermöglicht.

Datenverwaltung Diese Schicht per se durch Webkomponenten zu realisieren ist genau so weit möglich, wie die der Logikschicht. Es kann ein Gerüst aus Custom-Elements über die Datenbank Zugriffsschicht gelegt werden, um so Datenbankabfragen mittels Webkomponenten zu tätigen und weiter zu verarbeiten.

4.4 Anforderung an die komponentenorientierte Webanwendung

Um nun den Grundgedanken der Webkomponenten zu vertiefen, und diese Technologie ausschließlich zur Erstellung einer CRUD-Anwendung zu nutzen, werden folgende Anforderungen gestellt, gruppiert nach Stufen:

¹<https://www.webcomponents.org/>

²<https://nodejs.org/>

³<https://github.com/scramjs/scram-engine>

4.4.1 Darstellung

Die Anwendung soll eine rudimentäre Benutzeroberfläche zur Verfügung stellen, um alle CRUD-Operationen dem Benutzer zu ermöglichen. Für das Erstellen soll ein Formular das Eingeben und Bestätigen von Daten ermöglichen. Das Auflisten aller vorhandenen Datensätze in einer Tabelle gewährleistet das Lesen. Ein Formular, wie beim Erstellen, soll mit den Daten des spezifischen Datensatzes ausgefüllt sein, und das Aktualisieren den Nutzenden überlassen. Das Löschen soll über einen einfachen Auslöser machbar sein.

4.4.2 Logik

So weit es die Webkomponenten zulassen, soll der Server mit diesen strukturiert und konfiguriert werden.

4.4.3 Datenverwaltung

Die Interaktion mit der Datenbank soll durch Webkomponenten das Erstellen, Lesen, Aktualisieren und Löschen eines Datensatzes ermöglichen. Das bedeutet, alle CRUD-Operationen sollen durch Webkomponenten ausführbar sein.

Kapitel 5

Implementierung der Webanwendung

Die AnwenderInnen sollen zu Beginn Datensätze anlegen können. Im Prototypen wurde das Datenmodell von Personen mit Benutzername, E-Mail und Passwort herangezogen. Nachdem die AnwenderInnen das Formular für eine Person ausgefüllt haben, kann dieser Datensatz angelegt werden. Nach elementaren Validierungen werden die Daten an den Server übermittelt. Dieser speichert diese wiederum in der Datenbank ab. In der Listenansicht sehen die AnwenderInnen alle bereits angelegten Personen in einer Tabelle und per Klick auf den Namen wird zur Detailansicht weitergeleitet. Hier wird ein Formular mit den Nutzerdaten ausgefüllt, welche verändert und aktualisiert werden können. Auf der Detailseite findet sich ebenso ein Lösch-Knopf, um eine bestehende Person zu entfernen. Dadurch sind alle CRUD-Operation abgedeckt.

5.1 Umsetzung mit Dependencies Electron / Scram-Engine

Um die Nutzung von Webkomponenten am Server zu ermöglichen, werden zwei konkrete Technologien benötigt, da Webkomponenten grundsätzlich nur im Browser funktionieren.

5.1.1 Electron

Electron ist ein Framework, um native Cross-Plattform-Applikationen mit Web-Technologien wie JavaScript, HTML und CSS zu erstellen. Es basiert auf dem Chromium¹ Webbrowser und Node.js.

5.1.2 Scram-Engine

Das Scram-Engine-Projekt, erarbeitet von Jordan Last, ermöglicht es, eine HTML-Datei einem vorkonfigurierten Electronstartscript mit minimalem Aufwand, zur Verfügung zu stellen. Es wird lediglich Electron von der Kommandozeile aufgerufen und eine Startdatei mitgegeben. Die Vorkonfiguration vereinfacht das Arbeiten mit Electron durch beispielsweise das Anbieten von Kommandozeilenargumenten, um das Electronfenster zu verstecken, und somit das bequeme Starten von serverseitigen Anwendungen aus

¹<https://www.chromium.org/>

der Kommandozeile heraus. Electron ist notwendig, da es Chromium mit Node.js kombiniert. Die Scram-Engine nutzt Chromiums Fähigkeit, Webkomponenten in einen interaktiven DOM zu parsen, welcher manipuliert werden kann. Diese Webkomponenten können dadurch jeglichen Node.js Code nutzen und haben die Möglichkeit mit dem Betriebssystem zu interagieren, Datenbankaufrufe zu tätigen, prinzipiell alles, was Node.js möglich ist. Ein reiner Node.js Server würde nicht ausreichen, da dieser HTML, Custom-Elemente oder Webkomponenten generell nicht parsen kann. Durch die Kombination der beiden Technologien können Webkomponenten weitaus mehr leisten, als in einem Webbrowser.

Da universelle Webkomponenten auf Electron als deren Plattform angewiesen sind, muss das System, welches diese betreibt, relativ leistungsstark sein. Jordan Last arbeitet an einer Lösung, um die Electron-Abhängigkeit – und dadurch die Chromium-Abhängigkeit – loszuwerden. Jegliche Interessenten, die an dieser Optimierung mitwirken möchten, können ihn kontaktieren².

5.2 Implementierung der Komponenten

Der für diese Arbeit entwickelte Prototyp kann in die Hauptteile Frontend und Backend eingeteilt werden. Wobei das Backend nochmals in Server und Datenbank unterteilt wird.

5.2.1 Frontend

Im Frontend wurde hauptsächlich auf bereits vorhandene Webkomponenten aufgebaut. Es wurden Standard HTML-Elemente durch paper-input Elemente³ ersetzt, um ein abgestimmtes Gesamtbild zu erzeugen, ohne sich auf das Design konzentrieren zu müssen. So wurde bei dem Anlegen einer neuen Person, wie in Abb. 5.1 ersichtlich, auf `<paper-input>` Elemente mit einem `<iron-icon>` Element gesetzt. Das Aktualisieren eines Nutzers ist das selbe Formular, lediglich mit anderen `<paper-button>` Komponenten. Die Auflistung, aller bereits erstellten Personen, wurde durch ein `<paper-datatable-api>` Element erstellt. Jegliche Kommunikation zwischen Frontend und Backend wurde über eine REST-API gehandhabt. Die Anfragen wurden durch `<iron-ajax>` Komponenten vollzogen, welche eine Webkomponente für einfache ajax-Anfragen ist.

5.2.2 Server

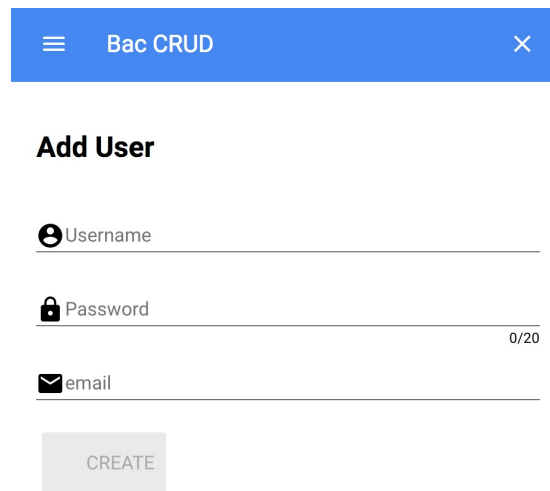
Ein Node.js Server kann mit Webkomponenten, wie im Programm 5.1 abgekürzt ersichtlich, konfiguriert und funktionstüchtig eingesetzt werden. Durch `<express-app>`, `<express-config>`, `<express-middleware>` und `<express-router>` mit `<express-route>` Komponenten⁴ kann die komplette Serverumgebung aufgesetzt werden.

In dieser Serveranwendung sind alle Endpunkte und Middlewares, welche die Express-Anwendung manipulieren, unter einer `<express-app>` Webkomponente verschachtelt. In

²<https://github.com/lastmjs>

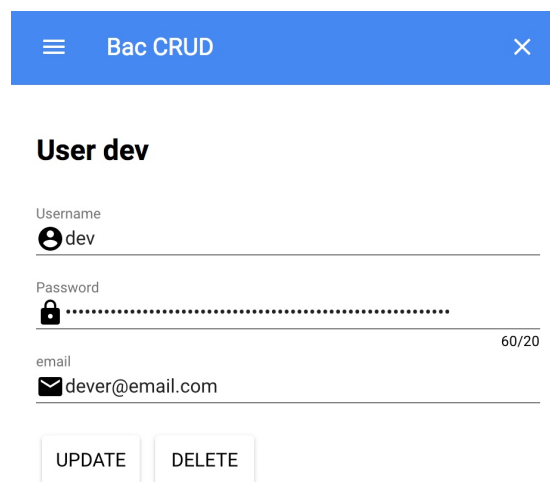
³<https://www.webcomponents.org/collection/PolymerElements/paper-input-elements>

⁴<https://github.com/scramjs/express-web-components>



The screenshot shows a web application interface with a blue header bar containing a hamburger menu icon, the text 'Bac CRUD', and a close icon. Below the header, the title 'Add User' is displayed. The form consists of three input fields: 'Username' with a person icon, 'Password' with a lock icon and a character count '0/20', and 'email' with an envelope icon. A 'CREATE' button is located at the bottom of the form.

Abbildung 5.1: User erstellen



The screenshot shows the same web application interface, but the title is 'User dev'. The form fields are pre-filled: 'Username' with 'dev', 'Password' with a masked password (dots) and a character count '60/20', and 'email' with 'dever@email.com'. At the bottom, there are two buttons: 'UPDATE' and 'DELETE'.

Abbildung 5.2: User bearbeiten

der Reihenfolge, wie die Middlewares definiert werden, werden diese auch in die Anwendung eingepflegt. In einem `<express-router>` Element können ebenfalls `<express-route>` Elemente und Middlewares verschachtelt, und so komplexe Routen-System erstellt werden.

Anstatt dass REST-Routen in Javascript imperativ definiert, können diese deklarativ in HTML zusammengesetzt werden. Es wird so eine visuelle Hierarchie der vorhanden Routen erstellt. Durch die Visualisierung mit HTML behält man nicht nur den Überblick, es ist auch einfacher zu verstehen als das Äquivalent in Javascript. Hier wird beispielsweise im Programm 5.2 der *indexHandler* (siehe Programm 5.3) als *Callback* an ein `<express-middleware>` Element übergeben, welches eine Methode(*get*) und einen Pfad(/) besitzt. Diese Middleware befindet sich unter dem `<express-router>` mit der

↓ ID	↓ 🔍 Username	↓ 🔍 Email
4	dev	dever@email.com
5	Test69	test@gmx.at
2	Test68	test68@gmx.at
11	Test66	test66@mail.com
12	New Worker Today	test@mail.test
6	FrankTheTank	frankyTanky@hanky.com

Abbildung 5.3: User Liste

Programm 5.1: Exemplarische Express-App mit Webkomponenten

```

1 <express-app port="3000">
2   <express-config callback="[[staticMW]]"></express-config>
3   <express-middleware callback="[[bodyParserMW]]"></express-middleware>
4   <express-router path="/">
5     <express-middleware method="get" path="/" callback="[[indexHandler]]"></express-
      middleware>
6   </express-router>
7   <db-router></db-router>
8 </express-app>

```

Basisroute als Pfad. Das bedeutet, dass dieser Router für seinen Pfad unter sich Routen haben kann, die ebenso einen Pfad besitzen. Da der *indexHandler* auf die Basisroute angewendet werden soll, wird dieser Middleware ebenfalls die Basisroute als Pfad mitgegeben. Nach diesem Prinzip können jegliche verschachtelte Routen erstellt werden. Wobei Subrouten immer unter der übergeordneten Route verschachtelt werden sollen. So soll die Route */index/list* unter dem Router für die Route */index* als Route */list* geschachtelt werden.

Die gezeigte Route im Code 5.2 ist rein zur Definition, welche URL, welche Seite rendern soll. Für REST-Anfragen, die Daten aus der Datenbank manipulieren, wurde eine eigene Komponente `<db-router>` erstellt, welche alle relevanten Datenbankabfragen über zuvor definierte Routen verarbeitet. Diese Komponente, welche selber `<express-router>` Elemente implementiert, kapselt so Datenbank spezifische Endpunkte von der Serverkonfiguration ab. Dieser `<db-router>` ist, ebenso wie ein `<express-router>` Element, ein Kind der `<express-app>` Komponente. Ersichtlich im Programm 5.1.

5.2.3 Datenverwaltung

Der im Programm 5.1 ersichtliche `<db-router>` verarbeitet jegliche Datenbank spezifischen Anfragen und triggert `<db-query>` Komponenten, um Daten der Datenbank

Programm 5.2: Indexroute-Konfiguration mit Webkomponenten

```

1 <express-router path="/">
2   <express-middleware method="get" path="/" callback="[[indexHandler]]"></express-
     middleware>
3 </express-router>

```

Programm 5.3: indexHandler Funktion

```

1 indexHandler(req, res) {
2   res.sendFile(path.resolve('./client/index.html'));
3 }

```

Programm 5.4: Datenverwaltung mit Komponenten

```

1 <db-query id="queryListUsers" db-model="[[userModel]]">
2   <db-find></db-find>
3   <db-sort sort2-query="{[_sort]}" sort-query="[[_sort]]" sort-property="username"
4     sort-direction="asc"></db-sort>
5 </db-query>
6
7 <db-query id="queryUserDetails" db-model="[[userModel]]">
8   <db-find find-query="[[findUserQuery]]"></db-find>
9 </db-query>
10
11 <db-query id="querySaveUser" db-model="[[userModel]]">
12   <db-save save="[[newUser]]"></db-save>
13 </db-query>
14
15 <db-query id="queryUpdateUser" db-model="[[userModel]]">
16   <db-find find-query="[[findUserQuery]]"></db-find>
17   <db-save save="[[updateUser]]"></db-save>
18 </db-query>
19
20 <db-query id="queryDeleteUser" db-model="[[userModel]]">
21   <db-find find-query="[[findUserQuery]]"></db-find>
22   <db-delete></db-delete>
23 </db-query>

```

zu manipulieren. Siehe Programm 5.4. Diesem `<db-query>` Element wird durch Kindelemente gesagt, was es für Abfragen tätigen soll. Besitzt das `<db-query>` Element beispielsweise lediglich ein `<db-find>` Element, wird davon ausgegangen, dass ein Datensatz von der Datenbank gesucht und zurückgegeben werden soll. Besitzt es aber zusätzlich zu dem `<db-find>`, ein `<db-delete>`, so wird das `<db-query>` Element nicht den gefunden Datensatz zurückgeben, sondern löschen.

DB Query Komponente

Um die interne Arbeitsweise zu verdeutlichen, wird die im Zuge dieser Arbeit entwickelte Webkomponente erläutert. Der exakte Programmcode kann auf GitHub nachgelesen werden⁵. Als Parameter benötigt das `<db-query>` Element eine eindeutige ID, welche zum Auslösen der Abfrage benötigt wird, und das zugehörige Datenbankmodell – da dieses Element für mongoDB mit mongoose entworfen wurde. Die Komponente ist so konzipiert, dass diese zu Beginn durch alle Kindelemente traversiert, die unterschiedlichen Typen überprüft und abhängig von den vorhandenen DB-Element-Typen⁶ die Element-Parameter zur eigentlichen *Query* zusammenfügt. Mit einer Funktion(*execute*), welche als Parameter eine Callback-Funktion mitbekommt, kann die *Query* ausgelöst und auf das Resultat reagiert werden.

⁵<https://github.com/drdreo/webcomponent-cms/blob/master/server/components/db-query.html>

⁶DB wurde als Präfix für die Datenbank Webkomponenten herangezogen

Kapitel 6

Evaluierung

In diesem Kapitel wird objektiv die Webkomponenten Technologie evaluiert.

6.1 Nutzen

Durch die Inkompatibilität der Webkomponenten mit diversen Browsern, können diese für manche EntwicklerInnen noch nicht als Produktionstechnologie verwendet werden. Jedoch haben bereits namhafte Firmen – YouTube, GitHub, etc. – Webkomponenten im Einsatz [10]. Da Webkomponenten noch nicht richtig – in der Industrie – in Verwendung sind, gibt es noch eine überschaubare Gemeinschaft die hinter den Webkomponenten steht.

Wurde die Technologie erstmals durchschaut, kann durch das Wiederverwenden der erstellten Komponenten Zeit gespart, und so die Produktivität gesteigert werden. Jedoch sollte bedacht werden, dass der Funktionsumfang, welcher ein Framework bietet, die gleichen und darüber hinaus noch weitere Funktionen zur Verfügung stellen kann. Alle Ziele, die mit Webkomponenten erreicht werden können, kann gleichermaßen auch mit einem Framework erzielt werden. Lediglich auf eine andere Art und Weise.

6.2 Flexibilität

Der Vorteil der Webkomponenten liegt darin, sie nach belieben einsetzen zu können. Damit ist gemeint, dass in bestehenden Projekten Teile durch Webkomponenten ausgetauscht und nachgebessert werden können. Egal, ob es ein „vanilla“ Projekt, oder ein mit beispielsweise Angular aufgesetztes Projekt, ist. Bei einer Anwendung, welche nicht mit dem gewünschten Framework erstellt wurde, können meist keine Elemente dieses Frameworks genutzt werden. Bei Webkomponenten fällt dieser Nachteil weg.

6.3 Wiederverwendbarkeit

Die Abkapselung der Webkomponenten vom restlichen Dokument und prinzipiell auch dem Projekt selbst, macht diese wiederverwendbar. Hat man eine Komponente für eine bestimmte Anwendung programmiert, so ist diese nicht zwingend an diese gebunden und

kann auch in künftigen, oder vergangenen Projekten, verwendet werden. Durch ihre Flexibilität wird auch die Wiederverwendbarkeit gesteigert, da man Webkomponenten auch in Projekten, welche auf einem Framework aufbauen, wieder verwenden kann. So wird das endlose Reimplementieren der selben Funktionen in unterschiedlichen Projekten obsolet. Ebenso ist es möglich Komponenten anderen EntwicklerInnen zur Verfügung zu stellen, sodass man eine große Palette an bereits vorhandenen Komponenten hat, um Anwendungen zu realisieren.

Das Wiederverwenden ist möglich, da Webkomponenten auf derzeitigen oder geplanten Webstandards basieren, wie in Kap. 2.1 erwähnt, welche alle größeren Browser versuchen zu implementieren. Das bedeutet, dass Webkomponenten nicht auf ein Framework oder eine Bibliothek angewiesen sind, und universell im Web funktionieren können.

6.4 Testbarkeit

Da Komponenten abgeschlossene Systeme sind, können sie als Blackbox betrachtet werden. Diese sind von deren Umgebung komplett unabhängig. Diese Unabhängigkeit bringt den Idealfall für Programmtests mit sich, da es stets bequemer ist, abgeschottete Einheiten zu testen, als den ganzen Komplex.

Scram.js hat eine Option(-d), die es ermöglicht ein Electron Fenster während des Debuggens zu öffnen. Dadurch hat man alle Chrome-Entwicklerwerkzeuge für das Debuggen am Server zur Verfügung.

Um Webkomponenten zu testen gibt es vom Polymer-Team den sogenannten „Web Component Tester“¹. Dieser ist eine End-to-End Testumgebung, welcher das lokale Testen von selbst erstellten Elementen ermöglicht.

¹<https://www.polymer-project.org/1.0/docs/tools/tests>

Kapitel 7

Zusammenfassung

Webkomponenten sind mächtig. Sie tragen wesentlich zu der Standardisierung der Webplattform bei. Clientseitig sieht man die Vorteil bereits und ist auf einem guten Weg, die Nutzung solcher Komponenten voran zu treiben. Ebenso gibt es zahlreiche Vorteile für die serverseitige Nutzung. Die Distanz zwischen client- und serverseitiger Komponenten-Nutzung wird geringer und Webkomponenten sind ein wichtiger Schritt in die richtige Richtung.

7.1 Lernkurve

Es gibt genügend EntwicklerInnen, UI/UX-DesignerInnen, AnfängerInnen und andere Personen, welche nichts mit Backend-Programmierung anfangen können, aber grundlegendes Wissen von HTML, CSS und Javascript besitzen. Durch die serverseitige Nutzung von Webkomponenten könnten solche Personen bekannte Techniken anwenden und vor allem, durch die semantische Sprache der Custom-Elemente, würden diese fähiger sein, serverseitigen Code zu erstellen und zu verstehen. Dies wäre mit dem Senken der Lernkurve gleichzusetzen.

7.2 Leistung

Durch die Abhängigkeit von Electron, und die daraus resultierende Leistung und Stabilität der Serverumgebung, könnten noch Probleme verursachen. Besonders wenn man die Anwendung auf ein reales Szenario skalieren möchte. Unabhängig davon, wäre meiner Meinung nach nicht die zur Verfügung stehende Leistung das Problem. Da Electron nur einen Prozess startet, welcher Node.js Code ausführt, der mehr oder weniger genau so wie ein *vanilla* Node.js Prozess laufen würde. Ob die Chromium Laufzeitumgebung stabil genug ist, um Monate lang, ohne Unterbrechung, laufen zu können, wäre die wichtigere Frage. Wie es Jordan Last beschrieben hat [12].

7.3 Markup

Ebenso kann ich mir vorstellen, dass geübte ProgrammiererInnen von dem wortreichen Stil der Webkomponenten abgeschreckt werden. Es benötigt mehr Codezeilen um die

selbe Aufgabe mit serverseitigen Webkomponenten zu realisieren, als mit *vanilla* Javascript, aufgrund der Markup-Sprache.

7.4 Prototyp

Die Entwicklung des Prototypen, welcher alle CRUD-Operationen für die Datenmanipulation einer mongoDB-Datenbank, sowie ein Nutzermanagement mit Login-Mechanismus implementiert, gewährte einen großen Einblick auf die Funktionen und Möglichkeiten der Webkomponenten. Durch das neu Erschaffen der DB-Custom Elemente wurde ersichtlich, das aufgrund der Natur von Javascript, so einiges möglich ist. Für mich brachte die Nutzung der Webkomponenten eher eine Abkapselung von Programmcode als visuelle Schicht, als einen programmiertechnischen Vorteil. Es musste erst ein Grundgerüst geschaffen, und dann mit Funktion ausgestattet werden, welches sowieso in Javascript geschrieben wurde.

Für geübte EntwicklerInnen würde es kaum einen Nutzen geben, Programm-Logik über Webkomponenten zu handhaben, außer die modulähnliche Arbeitsweise. Für AnfängerInnen jedoch, kann ich mir vorstellen, wäre es einfacher – durch die Visualisierung von Code durch Komponenten – in die Welt der Webentwicklung einzusteigen.

Quellenverzeichnis

Literatur

- [1] Roy Thomas Fielding. „Architectural Styles and the Design of Network-based Software Architectures“. Doctoral dissertation. University of California, Irvine, 2000. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm#fig_5_8 (siehe S. 6).
- [2] N. Leavitt. „Will NoSQL Databases Live Up to Their Promise?“ *Computer* 43.2 (Feb. 2010), S. 12–14 (siehe S. 13).
- [3] Petar Tahchiev u. a. *JUnit in Action*. 2. Aufl. Greenwich: Manning, 2011 (siehe S. 8).

Online-Quellen

- [4] Nov. 2017. URL: https://en.wikipedia.org/wiki/Multitier_architecture#Three-tier_architecture (siehe S. 14).
- [5] Samuel Andras. *JavaScript Frameworks, why and when to use them*. Feb. 2017. URL: <https://blog.hellojs.org/javascript-frameworks-why-and-when-to-use-them-43af33d0608d> (siehe S. 8).
- [6] Polymer Authors. *Browser compatibility*. Google. Juli 2017. URL: <https://www.polymer-project.org/1.0/docs/browsers> (siehe S. 2).
- [7] Angular Docs. *Architecture Overview*. Nov. 2017. URL: <https://angular.io/guide/architecture#components> (siehe S. 9).
- [8] React Docs. *Components and Props*. Nov. 2017. URL: <https://reactjs.org/docs/components-and-props.html> (siehe S. 11).
- [9] Vue.js Docs. *Components*. Okt. 2017. URL: <https://vuejs.org/v2/guide/components.html> (siehe S. 12).
- [10] Marcus Hellberg. *Web Components in production use – are we there yet?* Juni 2017. URL: <https://vaadin.com/blog/-/blogs/web-components-in-production-use-are-we-there-yet-> (siehe S. 24).
- [11] Krunal. *Benefits of using the n-tiered approach for web applications*. Sep. 2008. URL: <http://krunal-ajax-javascript.blogspot.co.at/2008/09/benefits-of-using-n-tiered-approach-for.html> (siehe S. 14).

- [12] Jordan Last. *Server-Side Web Components How and Why?* Juli 2016. URL: <https://scotch.io/tutorials/server-side-web-components-how-and-why> (siehe S. 26).
- [13] Travis Leithead und Arron Eicholz. *Bringing componentization to the web: An overview of Web Components*. Microsoft Edge Team. Juli 2017. URL: <https://blogs.windows.com/msedgedev/2015/07/14/bringing-componentization-to-the-web-an-overview-of-web-components/#H5ykveDs14lpvXEr.97> (siehe S. 1).
- [14] Polymer-Project. *Custom element concepts*. 2017. URL: <https://www.polymer-project.org/2.0/docs/devguide/custom-elements> (siehe S. 10).
- [15] Fredrik Söderquist. *The benefits of Component Driven Web Development*. Nov. 2017. URL: <https://dev.to/fregu/the-benefits-of-component-driven-web-development> (siehe S. 15).
- [16] W3C. *WEB COMPONENTS CURRENT STATUS*. Nov. 2017. URL: https://www.w3.org/standards/techs/components#w3c_all (siehe S. 4).
- [17] W3C, Dimitri Glazkov und Hajime Morrita. *HTML Imports*. Nov. 2017. URL: <https://www.w3.org/TR/html-imports/> (siehe S. 5).
- [18] *What is a Polyfill?* Okt. 2010. URL: <https://remysharp.com/2010/10/08/what-is-a-polyfill/> (siehe S. 4).
- [19] WHATWG. *HTML - Living Standard - Custom elements*. Dez. 2017. URL: <https://html.spec.whatwg.org/multipage/custom-elements.html#custom-elements-core-concepts> (siehe S. 4).
- [20] WHATWG. *HTML - Living Standard - Scripting*. Dez. 2017. URL: <https://html.spec.whatwg.org/multipage/scripting.html#the-template-element> (siehe S. 5).

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —