Webkomponenten zur Entwicklung von 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

Vorwort

Dies ist **Version 2017/02/28** der LaTeX-Dokumentenvorlage für verschiedene Abschlussarbeiten an der Fakultät für Informatik, Kommunikation und Medien der FH Oberösterreich in Hagenberg, die mittlerweile auch an anderen Hochschulen im In- und Ausland gerne verwendet wird.

Während dieses Dokument anfangs ausschließlich für die Erstellung von Diplomarbeiten gedacht war, sind nunmehr auch *Masterarbeiten*, *Bachelorarbeiten* und *Praktikumsberichte* abgedeckt, wobei die Unterschiede bewusst gering gehalten wurden.

Bei der Zusammenstellung dieser Vorlage wurde versucht, mit der Basisfunktionalität von LaTeX das Auslangen zu finden und – soweit möglich – auf zusätzliche Pakete zu verzichten. Das ist nur zum Teil gelungen; tatsächlich ist eine Reihe von ergänzenden „Paketen“ notwendig, wobei jedoch nur auf gängige Erweiterungen zurückgegriffen wurde. Selbstverständlich gibt es darüber hinaus eine Vielzahl weiterer Pakete, die für weitere Verbesserungen und Finessen nützlich sein können. Damit kann sich aber jeder selbst beschäftigen, sobald das notwendige Selbstvertrauen und genügend Zeit zum Experimentieren vorhanden sind. Eine Vielzahl von Details und Tricks sind zwar in diesem Dokument nicht explizit angeführt, können aber im zugehörigen Quelltext jederzeit ausgeforscht werden.

Zahlreiche KollegInnen haben durch sorgfältiges Korrekturlesen und konstruktive Verbesserungsvorschläge wertvolle Unterstützung geliefert. Speziell bedanken möchte ich mich bei Heinz Dobler für die konsequente Verbesserung meines „Computer Slangs“, bei Elisabeth Mitterbauer für das bewährte orthographische Auge und bei Wolfgang Hochleitner für die Tests unter Mac OS.

Die Verwendung dieser Vorlage ist jedermann freigestellt und an keinerlei Erwähnung gebunden. Allerdings – wer sie als Grundlage seiner eigenen Arbeit verwenden möchte, sollte nicht einfach („ung'schaut“) darauf los werken, sondern zumindest die wichtigsten Teile des Dokuments *lesen* und nach Möglichkeit auch beherzigen. Die Erfahrung zeigt, dass dies die Qualität der Ergebnisse deutlich zu steigern vermag.

Der Quelltext zu diesem Dokument sowie das zugehörige LaTeX-Paket sind in der jeweils aktuellen Version online verfügbar unter

https://github.com/Digital-Media/HagenbergThesis.

Trotz großer Mühe enthält dieses Dokument zweifellos Fehler und Unzulänglichkeiten – Kommentare, Verbesserungsvorschläge und passende Ergänzungen sind daher stets willkommen, am einfachsten per E-Mail direkt an mich:

Dr. Wilhelm Burger, Department für Digitale Medien,
Fachhochschule Oberösterreich, Campus Hagenberg (Österreich)

wilhelm.burger@fh-hagenberg.at

Übrigens, hier im Vorwort (das bei Diplom- und Masterarbeiten üblich, bei Bachelorarbeiten aber entbehrlich ist) kann kurz auf die Entstehung des Dokuments eingegangen werden. Hier ist auch der Platz für allfällige Danksagungen (z. B. an den Betreuer, den Begutachter, die Familie, den Hund, ...), Widmungen und philosophische Anmerkungen. Das sollte allerdings auch nicht übertrieben werden und sich auf einen Umfang von maximal zwei Seiten beschränken.

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 im fünften Semester entwickelten Prototypen.

Abstract

Webcomponents aim towards atomarizing web programming by giving developers the possibility to extend already exisitng 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 as a project in the fifth semester.

Kapitel 1

Einleitung

1.1 Motivation

Die Idee, die Webprogrammierung komponentenorientiert zu gestalten, gibt es seit 1998 [17]. 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 Anuglar[1] oder Facebook mit React[2], mittels eigen entwickelten komponentenbasierten Frameworks versuchen, die Webentwicklung voran zu treiben. Allerdings ist seit neuestem die grundlegende Idee der nativen Webkomponenten[3], 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 eines konkreten Content-Management-System dar.

1.2 Theoretischer Hintergrund und Stand der Forschung

Ein Content-Management-System (CMS) ist ein System, welches den NutzerInnen entsprechende Inhalte verwalten lässt. Seien es beispielsweise die Kunden, die Projekte oder aber interne Statistiken einer Firma. BenutzerInnen mit Zugriffsrechten können dieses System meist ohne Programmier- oder HTML-Kenntnisse bedienen. Angular oder React wären für die Umsetzung eines solchen CMS 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, was einige EntwicklerInnen noch zögern lässt

[1] https://angular.io/
[2] https://facebook.github.io/react/
[3] https://www.webcomponents.org/

um komplett auf Webkomponenten zu setzen. Google hat mit Polymer[4] eine Javascript-Bibliothek geschaffen, welche die Kreierung und Nutzung von Webkomponenten ungemein vereinfacht. Darüber hinaus nutzt diese Bibliothek Polyfills, was 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 [15]. Native Browserunterstützung ist bereits bei den meisten in Entwicklung und würde Polyfills überflüssig machen.

1.3 Forschungsfrage

Aus diesen Ansätzen ergibt sich die folgende Forschungsfrage für diese Bachelorarbeit:

> Wie muss ein CMS 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 umgegangen wird, und wie mit komponentenorientierte Frameworks. Gemeinsame Faktoren wie Mechaniken 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 eine Umsetzung eines CMS mittels Webkomponenten ermöglicht.

Überprüft soll die Anwendbarkeit dieses Konzept durch einen eigenen, im Rahmen des fünften Semesters entwickelten, Prototypen werden.

1.5 Erwartete Ergebnisse

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

Bei den Tests der praktischen Umsetzung der Webkomponenten in einem CMS wird ebenfalls eine positive Evaluierung erwartet, da es bereits erfolgreiche Konzepte basierend auf server- und clientseitigen Webkomponenten gibt, auf deren Erfahrungen aufgebaut werden kann.

[4] https://www.polymer-project.org/

Kapitel 2

Technische Grundlagen

2.1 Webkomponenten

Webkomponenten bestehen aus 4 Bestandteilen:

- 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 Seiten der W3C gearbeitet. Mehr dazu findet sich in [3]. Jedoch um diese derzeitig in allen größeren Browsern zu nutzen gibt es sogenannte Polyfills [2]. Dieses Polyfill ist ein Codestück, welches Technologie zur Verfügung stellt, die der Browser nicht nativ unterstützt. 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.

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 [4].

Exemplarische Definition:

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

2.1.2 HTML Imports

HTML Imports sind einer der vier Webkomponenten Spezifikation. Sie ermöglicht das Inkludieren und Wiederverwenden von HTML Dokumenten in anderen. Dieser Import kann all jenes enthalten, was in diesem HTML Dokument definiert ist (JavaScript, CSS, HTML, ...). Diese Möglichkeit, mehrere Technologien in einem Import, quasi eine URL, zu bündeln, ist eine kleine, aber wesentliche. Aktuell wird es von kaum einem Browser, außer Chrome 62, unterstützt.[1] Genaueres ist in [5] nachzulesen.

Exemplarischer 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.

Exemplar:

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

Das p-Element in diesem Template ist kein Kind dessen im DOM. Es ist ein Kind des DocumentFrgament welches vom Template content IDL Attribut zurückgegeben wird, wie in [6] erläutert.

2.1.4 Shadow DOM

Der Shadow Dom bringt das Abkapseln von DOM Elementen und CSS mit sich. Beispielsweise kann unorganisierte CSS-Code von einem Bereich in den anderen durchsickern, was wiederum einen Konflikt erzeugt, welcher unbeabsichtigtes Verhalten hervorbringen kann.

Shadow DOM löst dieses Problem durch Isolierung der CSS-Bereiche und Abkapselung des DOM. Gemeinsam mit der Custom Element API ermöglicht dies 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.

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

2.2 Komponentenorientierte JavaScript-Frameworks und -Bibliotheken

Sogenannte Javascript Frameworks oder Bibliotheken – der größte Unterschied liegt darin, dass eine Applikation die Bibliothek einbindet und ein Framework die Anwendung an

[1] https://caniuse.com/#search=HTML%20Import

sich – sind Werkzeuge um dem/der EntwicklerIn ein Grundgerüst an Funktionalität zur Verfügung zu stellen. Ein wichtiger Aspekt ist dabei auch die Effizienz und Verwertbarkeit von erfahrenen EntwicklerInnen entworfenen Code [7]. Eine konkrete Beschreibung findet sich in [8]:

> *Ein Framework gibt somit in der Regel die Anwendungsarchitektur vor. Dabei findet eine Umkehrung der Steuerung (Inversion of Control) statt: Der Programmierer registriert konkrete Implementierungen, die dann durch das Framework gesteuert und benutzt werden, statt – wie bei einer Klassenbibliothek – lediglich Klassen und Funktionen zu benutzen. Wird das Registrieren der konkreten Klassen nicht fest im Programmcode verankert, sondern „von außen" konfiguriert, so spricht man auch von Dependency Injection.*

Im Anschluss befindet sich eine Auflistung von solchen, zu diesem Zeitpunkt populären Javascript Frameworks und Bibliotheken[2]. Diese arbeiten alle zum Großteil mit Komponenten im Frontend.

- Angular
- Polymer
- REACT
- Vue.js

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

2.2.1 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 inkompatibler 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 trug[3].

Angular Komponente

Das Kernkonzept von Angular ist die Komponenten. Eine komplette Angular Anwendung kann als Baum aus diesen Komponenten modelliert werden. Die Definition aus der Angular Dokumentation [9]:

> *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.*

[2] http://www.npmtrends.com/angular-vs-react-vs-vue-vs-@angular/core
[3] http://angularjs.blogspot.co.at/2016/12/ok-let-me-explain-its-going-to-be.html

```
import { Component } from '@angular/core';

@Component({
  selector: 'hello-world',
  template: '<p>Hello, {{text}}!</p>',
})
export class HelloComponent {
  text: string;
  constructor() {
    this.text = 'World';
  }
}
```

Angewendet wird diese Komponente wie folgt:

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

Was folgende Ausgabe rendert:

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

2.2.2 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 war am 18. Oktober 2017 mit der Version 2.2.0.

Polymer Komponente

Polymer baut auf der Custom-Element Spezifikation auf und fügt dem eine Reihe an Features hinzu wie beispielsweise [10]:

- 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.

```
<link rel='import' href='bower_components/polymer/polymer.html'>
<dom-module id='hello-world'>
<template>
<p>Hello {{text}}</p>
</template>
<script>
class HelloWorld extends Polymer.Element {
  static get is() {
    return 'hello-world';
  }
```

```
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>
```

2.2.3 React

React (auch „React.js“ oder „ReactJS“) wird als „JavaScript Bibliothek zur Entwicklung von User-Interfaces“[4] 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 der React Dokumentation [11]:

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

```
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;
```

Angewandt wird diese Komponente wie folgt:

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

[4] https://reactjs.org/blog/2013/06/05/why-react.html

Was folgende Ausgabe erzeugt:

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

2.2.4 Vue

Vue (englische Aussprache [/vju/]) (auch „Vue.js“ genannt) beschreibt sich selbst als ein „progressives Framework zur Entwicklung von User-Interfaces“[5]. 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 2.5.2 (Stand 13. Oktober 2017).

Vue Komponente

Die Definition der Vue Dokumentation [12]:

> *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.*

```
1  // register
2  Vue.component('hello-world', {
3    template: '<p>Hello {{ text }} </p>',
4    data: function () {
5      return {text:"World"};
6    }
7  })
8  // root instance
9  new Vue({
10   el: '#container'
11 })
```

Angewandt 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>
```

[5] https://vuejs.org/v2/guide/

2.3 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 [16]. 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.

2.3.1 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, welche vom Server verarbeitet wird.

Zustandslosigkeit

Jegliche REST-Zugriffe sind zustandslos. Heißt, 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.

2.3.2 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

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, ohne die komplette Endanwendung umschreiben zu müssen. Diese mögen sich auf der selben Maschine befinden, werden aber meist auf unterschiedliche ausgelagert [13].

Beispielsweise sind bei dem 1-Stufen-Ansatz alle für die Anwendung relevanten Funktionen – Darstellung, Logik, Datenverwaltung, etc.– auf der Client-Maschine 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 Darstellung, Anwendungslogik und Datenverwaltung entkoppelt. Ein Webbrowser, zur Anzeige der Benutzeroberfläche, wäre zum Beispiel die erste Stufe. Ein Webserver, welcher die Logik ausführt, die Zweite und als dritte Ebene, zur Datenverwaltung, käme eine Datenbank in Frage.

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 [1].

Der größte Nachteil von diesen Ansätzen ist der zusätzliche Overhead-Aufwand und eine höhere Latenz, was wiederum die der NutzerInnen wahrgenommen Geschwindigkeit

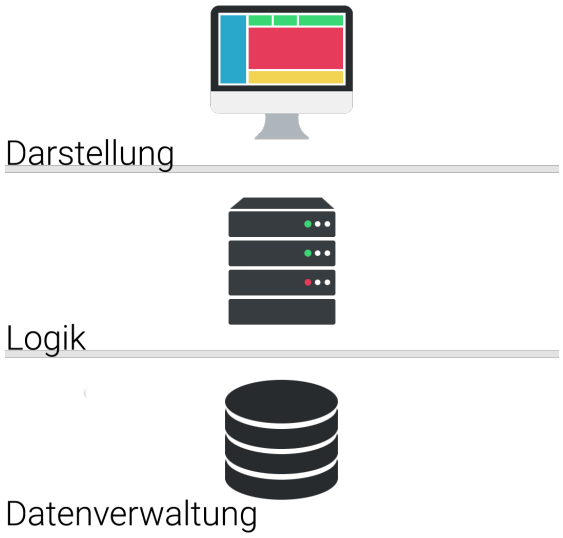

Abbildung 4.1: 3-Stufen-Architektur

mindert.

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 eigene standardmäßige blaue Farbe, einen Handzeiger bei hover 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, aber auch verständlicher macht. Durch Kombination solcher Elemente, mit eigens definierten Stylesheets können komplexere Gerüste aus HTML Elementen, mit komplexeren CSS/Javascript entstehen.

Damit die EntwicklerInnen nicht die komplette Struktur im Kopf behalten müssen, und bei Problemen nur diese, das aufgetretene Problem beheben können, hat es sich bewehrt, 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 komplexe Strukturen in atomare Einzelteile aufteilen [14].

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

4.3 Anforderung an die komponentenorientierte Webanwendung

4.4 Eigene Idee, technisches Design

Kapitel 5

Implementierung der Webanwendung

5.1 Struktur der Webanwendung

5.2 Umsetzung mit Dependencies Electron / ScramEngine

5.3 Implementierung der Komponenten

Kapitel 6

Evaluierung

6.1 Aufwand / Nutzen

6.2 Wiederverwendbarkeit

6.3 Skalierbarkeit, Flexibilität

6.4 Testbarkeit

Kapitel 7

Zusammenfassung

7.1 Kosten-Nutzen Faktor

Quellenverzeichnis

Literatur

[1] Sep. 2008. URL: http://krunal-ajax-javascript.blogspot.co.at/2008/09/benefits-of-using-n-tiered-approach-for.html (siehe S. 12).

[2] Okt. 2010. URL: https://remysharp.com/2010/10/08/what-is-a-polyfill/ (siehe S. 3).

[3] Nov. 2017. URL: https://www.w3.org/standards/techs/components#w3c_all (siehe S. 3).

[4] Dez. 2017. URL: https://html.spec.whatwg.org/multipage/custom-elements.html#custom-elements-core-concepts (siehe S. 3).

[5] Nov. 2017. URL: https://w3c.github.io/webcomponents/spec/imports/ (siehe S. 4).

[6] Dez. 2017. URL: https://html.spec.whatwg.org/multipage/scripting.html#the-template-element (siehe S. 4).

[7] Feb. 2017. URL: https://blog.hellojs.org/javascript-frameworks-why-and-when-to-use-them-43af33d0608d (siehe S. 5).

[8] März 2017. URL: https://de.wikipedia.org/wiki/Framework (siehe S. 5).

[9] Nov. 2017. URL: https://angular.io/guide/architecture#components (siehe S. 5).

[10] 2017. URL: https://www.polymer-project.org/2.0/docs/devguide/custom-elements (siehe S. 6).

[11] Nov. 2017. URL: https://reactjs.org/docs/components-and-props.html (siehe S. 7).

[12] Okt. 2017. URL: https://vuejs.org/v2/guide/components.html (siehe S. 8).

[13] Nov. 2017. URL: https://en.wikipedia.org/wiki/Multitier_architecture#Three-tier_architecture (siehe S. 12).

[14] Nov. 2017. URL: https://dev.to/fregu/the-benefits-of-component-driven-web-development (siehe S. 13).

[15] Polymer Authors. *Browser compatibility*. Google, Juli 2017. URL: https://www.polymer-project.org/1.0/docs/browsers (siehe S. 2).

[16] 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. 9).

[17] 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/#H5ykveDs14IpvXEr.97 (siehe S. 1).