HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

BACHELORARBEIT

Herr
Robert Keller

Vergleich von MV* Frameworks der Skriptsprache JavaScript

2016

Fakultät Angewandte Computer- und Biowissenschaften

BACHELORARBEIT

Vergleich von MV* Frameworks der Skriptsprache JavaScript

Autor:
Herr Robert Keller

Studiengang:
Medieninformatik & interaktives Entertainment

Seminargruppe:
MI12-w1B

Erstprüfer:
Prof. Alexander Marbach

Zweitprüfer:
Christian Roschke B.Sc.

Einreichung:
Mittweida, 23.05.2016

Faculty of Applied Computer Sciences & Biosciences

BACHELOR THESIS

Comparing MV* frameworks of the scripting language JavaScript

author:
Mr. Robert Keller

course of studies:
Media Informatics and Interactive Entertainment

seminar group:
MI12-w1B

first examiner:
Prof. Alexander Marbach

second examiner:
Christian Roschke B.Sc.

submission:
Mittweida, 23.05.2016

Bibliografische Angaben:

Keller, Robert:

Vergleich von MV* Frameworks der Skriptsprache JavaScript

Comparing MV* frameworks of the scripting language JavaScript

2016 - 79 Seiten

Mittweida, Hochschule Mittweida (FH), University of Applied Sciences,

Fakultät Angewandte Computer- und Biowissenschaften, Bachelorarbeit, 2016

Abstract

Die vorliegende Bachelorarbeit gibt einen Überblick über die aktuellen Konzepte modularer Programmierung und deren Umsetzung mit JavaScript. Hierfür werden einige JavaScript Frameworks für die Erstellung einer einfachen clientseitigen Anwendung geprüft und unter verschiedenen Kriterien ausgewertet. Als Basis dieser Arbeit dienen Fachliteratur, die Dokumentation der jeweiligen Frameworks und die persönlichen Erkenntnisse und Erfahrungen des Authors. Der Verfasser erhofft sich dadurch einen Erkenntnisgewinn zur sauberen und modularen Programmierung mit JavaScript und wie diese bei kleinen und clientseitigen Anwendungen durch Frameworks erleichtert werden kann.

Inhaltsverzeichnis

Abkürzungsverzeichnis

Asynchronous JavaScript and XML
...AJAX

Asynchronous Module Definition
...AMD

Cascading Stylesheet
...CSS

Document Object Modell
...DOM

Extensible Markup Language
...XML

Hyper Text Markup Language
...HTML

JavaScript Object Notation
...JSON

model-view-controller
...MVC.

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Immer mehr Unternehmen der IT-Industrie entwickeln ihre Systeme basierend auf Webtechnologien. Besonders die zunehmende Notwendigkeit Anwendungen mobil und auf verschiedensten Endgeräten nutzen zu können unterstützt diesen starken Aufwärtstrend.

In der Entwicklung wird hierbei, besonders im Bereich Frontend-Programmierung die Skriptsprache JavaScript genutzt. Zur Entwicklung setzt man auch im Falle von JavaScript auf die Verwendung zahlreicher Frameworks oder Bibliptheken welche komplexe Programmstrukturen vereinfachen sollen und Lösungen für oft auftauchende Probleme bieten sollen.

Ziel dieser Bachelorarbeit ist es, einen Leitfaden für die Verwendung des optimalen JavaScript-Frameworks für kleine und clientseitige Anwendungen zu erarbeiten, um aufgrund des großen Überangebots ein sinnvolles Framework auszuwählen.

Dieser Leitfaden orientiert sich an festgelegten Vergleichsparametern und nutzt neben den unmittelbaren Kriterien der Programmierung außerdem aktuelle Marktanalysen.

Der Autor hat bereits während des Studiums ersten Kontakt mit JavaScript gehabt und konnte diesen während seines Praxissemesters vertiefen. Die Bachelorarbeit soll ihn somit außerdem für die berufliche Zukunft unterstützen und einen noch intensiveren Einblick in JavaScript bieten.

2 Grundlagen

Zunächst sollen in diesem Kapitel einige Grundlagen aus dem Bereich der Webentwicklung erläutert werden. Der Author erachtet dies als zwingend notwendig, da nur so sicher gestellt werden kann das der in dieser Arbeit angelegte Vergleich wissenschaftlich fundiert und umfassend ist. Zu diesen erfassten Grundlagen zählen sowohl wirtschaftliche und analytische Grundlagen als auch fachliche Grundlagen.

2.1 Marktanalyse

Um eine Zweckmäßigkeit und Aktualität dieser Arbeit zu Gewährleisten ist zunächst eine Marktanalyse und die Betrachtung aktueller Webtechnologien notwendig.

Besonders der Faktor Kompatibilität ist aufgrund der Vielfalt sowohl im Bereich Software als auch Hardware ein tiefgreifendes Element um sich für bestehende Technologien zu entscheiden.

Der Markt im Bereich Webentwicklung ist hierbei durch viele Faktoren geprägt welche auch in den folgenden Abschnitten erläutert werden sollen.

2.1.1 Software

Der Bereich Software deckt im Themengebiet dieser Arbeit vor allem die Frage nach der Verwendung der Webbrowser ab.

2016	Chrome	IE	Firefox	Safari	Opera
March	69.9 %	6.1 %	17.8 %	3.6 %	1.3 %
February	69.0 %	6.2 %	18.6 %	3.7 %	1.3 %
January	68.4 %	6.2 %	18.8 %	3.7 %	1.4 %

Abbildung 1: Browserstatistik des Jahres 2016 | Quelle: http://www.w3schools.com/browsers/browsers_stats.asp

Eine der weltweit größten Web Entwickler Seiten : w3schools.com bietet hierfür eine online einsehbare Statistik an, welche aufzeigt wie viel Prozent der Internetnutzer die jeweiligen Browser verwenden.

Besonders in der zur Zeit rasanten Entwicklung ist es wichtig die Interessen der Nutzer zu beachten, welche die Möglichkeiten der verschiedenen Software Angebote nutzen. Anhand der Browserstatistik ist bereits erkennbar, dass die Browser Chrome und Firefox an erster Stelle für die Nutzer stehen. Der Browser wird von rund 70% der in der Statistik erfassten Nutzer verwendet und der an zweiter Stelle stehende Firefox von rund 18%. Außerdem ist in dieser Tabelle auch ein Aufwärtstrend des Chrome und ein Abwärtstrend des Firefox zu verzeichnen.

Um diese Statistik zu verifizieren nutzt der Verfasser außerdem die auf den deutschen Raum ausgelegte Statistik von: browser-statistik.de welche ein ähnliches Bild aufzeigt.

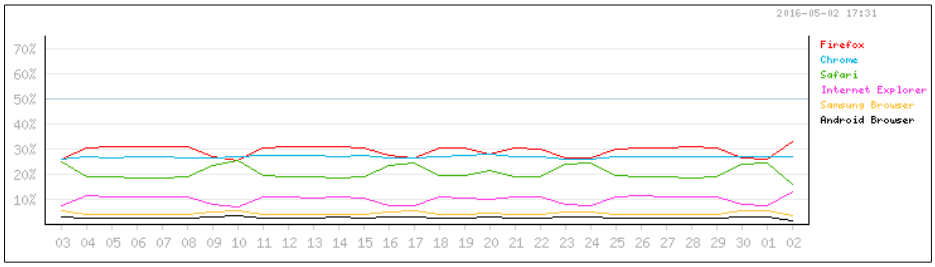

Abbildung 2: Browserstatistik Deutschland vom 02.05.2016 | Quelle: https://www.browser-statistik.de/statistiken/

Der zweiten Statistik ist allerdings zu entnehmen, das im deutschen Raum zur Zeit der Browser Firefox von mehr Nutzern verwendet wird. Dieser weist außerdem zum Zeitpunkt der Statistik einen Aufwärtstrend auf.

Aus eigener Erfahrung kann der Verfasser diese Ergebnisse nur bestätigen da er zum Zeitpunkt dieser Arbeit neben Chrome auch Firefox nutzt.

Diese Hypothese des Verfassers lässt sich durch seine unmittelbare Umgebung, sprich seinen Freundeskreis und seinen Kollegenkreis nur untermauern. Auch dort werden überwiegend die Browser Chrome und Firefox verwendet.

Der aus den Statistiken außerdem resultierende Browser Safari wird durch die große Zahl von Nutzern der Apple Hardware genutzt. Doch auch für MacOSX und Linux gibt es die Browser Chrome und Firefox.

Ein weiteres für Webentwickler oft kritisches Thema ist der noch oft genutzte Internet Explorer von Microsoft. Durch die Auslieferung mit jedem Microsoft Betriebssystem nutzen viele Anwender diesen ohne sich mit dem Thema Browser weiter auseinander zu setzen. Doch im Bereich Webentwicklung ist es oftmals der Internetbrowser der bestimmte Funktionen nicht unterstützt und somit die Browserkompatiblität problematisch gestaltet.

2.1.2 Deaktivierung von JavaScript

Neben der Verwendung eines kompatiblen Browsers ist für den JavaScript-Entwickler noch ein weiteres Kriterium von enormer Bedeutung : die Deaktivierung von JavaScript.

In den vergangen Jahren genoss JavaScript nicht den heute so guten Ruf. Es galt zu Zeiten in welchen vor allem die Technologie Flash das Web dominierte mehr oder weniger mehr als Helfer des Webs für unnötige Effekte wie herabfallende Sterne die dem Cursor folen oder Helferaufgaben wie Uhren auf Websites oder kleinere Informationsmeldungen.[1]

Zu diesem Thema gibt es zwar kaum aktuelle statistische oder wissenschaftliche Quellen. Ein Vergleich über Google Trends zeigt aber das man nicht aus den Augen lassen darf, das verschiedenste Nutzer durchaus weiterhin JavaScript aufgrund möglicher Sicherheitslücken deaktivieren.

Auch Journalist Klint Finley beschreibt in einem Artikel des Onlinemagazins Wired vom 18.11.2015 wie er in einem Selbstversuch eine Woche JavaScript deaktiviert. In seinem Artikel zeigt er auf das die Deaktivierung zu einem wesentlich „sauberem“ Interneterlebnis führt er allerdings auf viele Probleme stoßen musste. So war es ihm beispielsweise nicht mehr möglich GoogleDocs zu verwenden oder die Streamingplatform Netflix. [2]

Um trotz dieser wenigen Quellen einen Eindruck zu schaffen wie relevant das Thema in der heutigen Zeit ist hat der Verfasser sich für eine Analyse mit Google Trends entschieden. Der Dienst Google Trends stellt Informationen bereit, welche Suchbegriffe wann und wie oft in die Suchmaschine Google eingegeben werden. Da die Suchmaschine Google für die meisten Anwender die erste Anlaufstelle ist entschied sich der Verfasser mithilfe dieses Tools eine eigene Analyse beziehungsweise Statistik zu erheben um die Problemstellung von deaktiviertem JavaScript auf die heutige Zeit zu projezieren.

---

1 Vgl.: Fletzberger, Stephan: „The State of JavaScript: Der Ist-Zustand“, Lichtenecker, unter: https://liechtenecker.at/the-state-of-javascript-der-ist-zustand/ (abgerufen am 23.04.2016)

2 Vgl.: Finley, Klint: „I turned off JavaScript for a Whole Week and It Was Glorius“, WIRED, unter: http://www.wired.com/2015/11/i-turned-off-javascript-for-a-whole-week-and-it-was-glorious/#slide-1 (abgerufen am 07.05.2016)

Der Verfasser nutzt hierfür einen Vergleich der Suchbegriffe:[3]

- *“JavaScript deaktivieren”*
- *“deactivate JavaScript”*
- *“noscript”*

In diesem Vergleich ist sowohl auffällig, dass ein Interesse der Nutzer an der Deaktivierung von JavaScript in ihrem Browser existiert. Diese jedoch relative gering ist und einen Abwärtstrend aufweist.

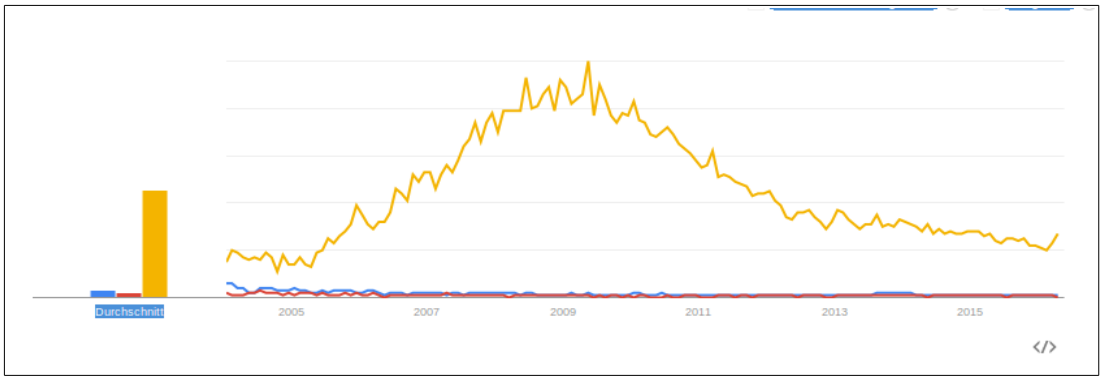

Abbildung 3: Google Trend Analyse für No Javascript vom 02.05.2016 | Quelle: https://www.google.de/trends/ erstellt durch Robert Keller

Abbildung 3 stellt die Google Trend Analyse für die Suchbegriffe: noscript(gelb), javascript deaktivieren(blau) und deactivate javascript(rot) dar. Erkennbar ist hierbei, das alle Kurven einen Abwärtstrend aufweisen. Der extreme Ausschlag der gelben Kurve kann auch dem Hintergrund entspringen, das die Formulierung noscript auch im HTML zur Definition eines Bereiches genutzt wird, der Nutzern ohne JavaScript angezeigt wird.

Neben der Frage ob JavaScript deaktiviert ist könnte es durchaus auch interessant sein ob der Zielbrowser überhaupt JavaScript mitbringt. Dazu nutzt der Verfasser die Website “caniuse.com“ welche Webentwickler recherchieren lässt welche Features mit welchen Browsern kompatibel sind.

Nutzt man hierbei das Schlagwort „javascript“ so sieht man, das mit aktuellen Browsern die Verwendung von JavaScript problemlos möglich ist. Lediglich der Browser Opera Mini scheint einige Probleme aufzuweisen [4].

3 Vgl.: Analyse von Google Trends unter https://www.google.de/trends/ (abgerufen am 02.05.2016)

4 Vgl.: CanIUse mit dem Suchbegriff Javascript, unter http://caniuse.com/#search=javascript (abgerufen am 04.05.2016)

2.1.3 Entwicklertrends

Als letzten Punkt der kurzen marktanalytischen Betrachtung steht nun noch eine eher spekulative Prognose des Verfassers. So lassen sich die in den folgenden Kapiteln erfassten Frameworks auch nach Suchanfragen und Anfragen auf den bekanntesten Entwicklerseiten analysieren.

Hierbei zeigt ein Artikel von Uri Shaked, in welchem er die Frameworks AngularJS BackboneJS und emberJS vergleicht das JavaScript Frameworks zunehmend an Aufmerksamkeit gewinnen[5].

Der Kernpunkt dieser letzten spekulativen Analyse zielt auf den zukünftigen Support der Frameworks ab. So liegt die Wahrscheinlichkeit sehr hoch, dass ein Framework welches oft auf Google gesucht oder auf diversen Foren diskutiert wird noch sehr lange Support und Weiterentwicklung erhält. Außerdem kann der Webentwickler neben den Standardressourcen auch mithilfe großer Communities seine Problemlösungen diskutieren.

2.1.4 Fazit

Aus diesem Abschnitt erhält der Verfasser wichtige Erkenntnisse über die zu beachtenden Standards und Probleme. Der Verfasser erkennt hierbei das eine Entwicklung für die aktuellen Browser Chrome und Firefox für den gegebenen Anwendungsfall ausreichen wird, da Statistiken zeigen das diese Browser am beliebtesten sind und am meisten genutzt werden. Auch die Problematik von deaktiviertem JavaScript verliert zunehmend an Bedeutung und kann damit außer acht gelassen werden.

5 Vgl.: Shaked, Uri: „AngularJS vs. Backbone.js vs. Ember.js“, AirPair, unter https://www.airpair.com/js/javascript-framework-comparison (abgerufen am 04.05.2016)

2.2 Stand aktueller Webtechnologien

Als Vorraussetzung für diese Arbeit dient der Trend zum Modebegriff “Web 2.0”.

> “Web 2.0 generally refers to a set of social, architectural, and design patterns resulting in the mass migration of business to the Internet as a platform.” [6]

Der Begriff Web 2.0 beschreibt nicht wirklich eine neue Version des Internet sondern eine neue Art der Nutzung von Webangeboten. Diese evolutionäre Veränderung zeigt sich vor allem darin, dass User vor der Zeit des Web 2.0 primär Websiten zur Informationsgewinnung nutzten. Mit dem Wandel des Nutzungsverhalten ist es nun außerdem von Bedeutung das der Nutzer einen Zusatznutzen und eine starke Interaktionsmöglichkeit mit den Webangeboten hat.

2.2.1 Multidevice

Durch die besondere Entwicklung des oben benannten Nutzungsverhaltens ist eine rasante technologische Entwicklung unaufhaltsam. Durch neue Interaktionsmöglichkeiten entsteht der zunehmende Drang von Nutzern ihre Webinhalte immer und überall zu konsumieren. Durch dieses Verlangen existieren bereits heute zahlreiche Endgeräte wie Smartphones, Tablets, Smartwatches, Laptops und PC’S welche unterschiedlichste Spezifikationen aufweisen.

Aus diesem Grund ist es von enormer Bedeutung, dass vorhandene Webapplikationen auf sämtlichen Geräten möglichst reibungslos funktionieren.

Gerät	Höhe in Pixel	Breite in Pixel	Auflösung in ppi
Iphone 6	1334	750	326
Iphone 6 Plus	1920	1080	401
Ipad Air	2048	1536	264
MacBookPro 13“	1280	800	

Tabelle 1: Auflösungen von Apple Geräten

6 Governor, James: Web 2.0 Architectures, 1. Auflage, O'Reilly Verlag, Sebastopol 2009, Seite ix (Preface)

Tabelle 1 zeigt welche Vielzahl von Displaymaßen und Auflösungen alleine Endgeräte von Apple liefern. Diese Darstellung soll lediglich verdeutlichen das es für Webentwickler von enormer Bedeutung ist jedes zur Zeit der Entwicklung und zukünfitges Endgerät so gut es geht zu berücksichtigen.

2.2.2 HTML5 und CSS3

HTML, die Hypertext Markup Language, befindet sich zum Zeitpunkt dieser Arbeit in der Version 5. Mit dieser Version bieten sich neue Funktionen, welche auch den Entwicklungsaufwand minimieren. Zu diesen Neuerungen gehören unter anderem Video- und Audiofunktionalitäten, lokale Speicherung aber auch direkt integrierte dynamische 2D- beziehungsweise 3D-Grafiken.[7]

Diese zusätzlichen Funktionen ermöglichen unter anderem auch den Verzicht auf zusätzliche Plugins wie zum Beispiel Adobe Flash.

Als Auszeichnungssprache dient HTML5 zur Auszeichnung und Vernetzung von Texten und Inhalten.

Um diese Inhalte dem Nutzer in einer ansprechenderen und noch besser lesbaren Form darzustellen nutzt man außerdem CSS.

CSS steht als Abkürzung für Cascading Style Sheets und steht zum Verfassungszeitpunkt dieser Arbeit in Version 3. Mit dieser Stylesheet-Sprache werden Gestaltungsanweisungen für HTML erstellt.

```
1  <!DOCTYPE html>
2  <html lang="de">
3    <head>
4      <meta charset="utf-8">
5       <meta name="viewport" content="width=device-width, initial-
   scale=1.0">
6      <title>Titel</title>
7    </head>
8    <body>
9    </body>
10 </html>
```

Dieses Beispiel zeigt das Grundgerüst einer HTML Seite.

7 Vgl.: Fain, Rasputins: „Enterprise Web Development“, 1. Auflage, O‘Reilly, Seite xxiii

2.2.3 XML

XML steht als Abkürzung für Extensible Markup Language und bedeutet übersetzt so viel wie eine erweiterbare Auszeichnungssprache. XML ist eine Technologie, welche zur Strukturierung und Beschreibung von Daten genutzt wird..

Der Begriff erweiterbar bedeutet bei XML, dass der Entwickler beziehungsweise Anwender das Dokument unter Beachtung der Syntax selber strukturieren kann und eine eigene Hierarchie entwickeln kann.

In anderen Worten kann XML auch als eine Syntax beschrieben werden welche man nutzt um eine eigene Sprache zu erstellen.[8]

```xml
<?xml version="1.0" encoding="UTF-8"?>
<person>
  <name>robert</name>
  <alter>22</alter>
  <größe>1.75</größe>
  <herkunft>Deutschland</herkunft>
</person>
<person>
  <name>max</name>
  <alter>25</alter>
  <größe>1.73</größe>
  <herkunft>Italien</herkunft>
</person>
```

In diesem Beispiel erstellte der Verfasser eine XML Datei zur Speicherung von zwei Personen.

8 Vgl.: Hunter, Rafter: Beginning XML, 4. Auflage, Wiley Publishing Inc, S. 7

2.2.4 JavaScript Object Notation - JSON

Wie in den meisten Programmiersprachen vorhanden so beschreibt JSON ein Datenaustauschformat welches einfach strukturiert ist. Diese Struktur besteht lediglich aus Objekten und Arrays[9].

Im Buch Beginning JSON wird JSON als nicht nur simples Daten Format sondern auch als Kern moderner Applikationen beschrieben.[10]

```
{
  "robert": {
    "alter": 22,
    "größe": 1.75,
    "herkunft": "Deutschland"
  },
  "max": {
    "alter": 25,
    "größe": 1.73,
    "herkunft": "Italien"
  }
}
```

Der Verfasser zeigt hier an einem einfachen Beispiel wie JSON für die Speicherung zweier Personen aussehen könnte. In diesem Beispiel wird mit JSON Objekten gearbeitet.

Dieses Beispiel baut auf dem Beispiel aus dem Abschnitt XML auf.

9 Vgl.: Reimers,Thies: PHP 5.4 & MySQL 5.5, 4. Auflage, Galileo Computing Verlag, S. 749

10 Vgl.: Smith, Ben: Beginning JSON, 1. Auflage, Springer Science+Business Media New York, S. 337

2.2.5 AJAX

Die Abkürzung AJAX steht für Asynchronous JavaScript+XML.

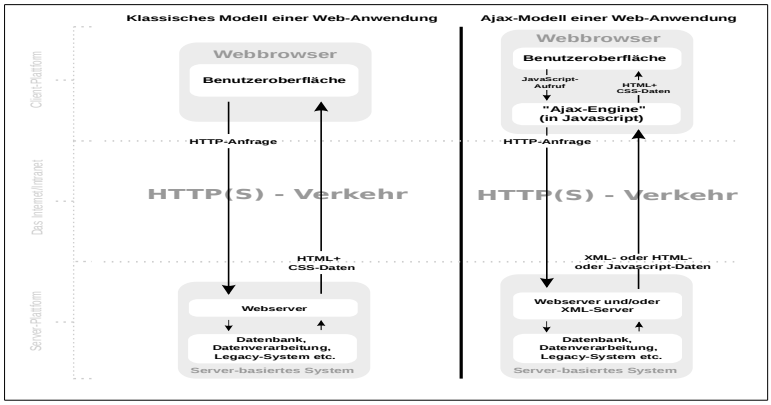

Abbildung 4: Vergleich Klassische zu Ajax Anwendung | Quelle: https://commons.wikimedia.org/wiki/File:Ajax-vergleich.svg

Dieses Konzept einer asynchronen Datenübertragung schafft die Möglichkeit, während eine HTML-Seite angezeigt wird, HTTP-Anfragen durchzuführen und ohne neues Laden die Seite zu verändern[11].

2.2.6 Fazit

Unter der Beachtung aktueller Technologien ist es für den Verfasser wichtig die Frameworks auf die Tauglichkeit dieser Technologien im Rahmen der Notwendigkeit für kleinere Anwendungen zu prüfen. Besonders von Bedeutung ist, das sich in der heutigen Zeit zahlreiche Endgeräte finden welche jeweils unterschiedlichste Bedingung für eine Webanwendung bieten.

11 Vgl.: Wenz, Christian: JavaScript und AJAX Das umfassende Handbuch, Galilio Computing Verlag, S 391

3 Begriffsklärung

In der Erstellung modularer Webanwendung gibt es einige grundlegende Prinzipien denen man folgen sollte um eine nachhaltige Anwendung zu erstellen. In diesem Abschnitt soll geklärt werden welche Grundprinzipien sich im Laufe des technologischen Fortschrittes herauskristallisiert haben und wie diese im speziellen Fall von Javascript umgesetzt werden können.

3.1 Modularisierung

Clientseitige JavaScript Anwendungen arbeiten innerhalb des HTML-Dokument indem sie auf das DOM zugreifen. Das Bedeutet das der Programmcode als Kontext das globale Objekt, das “window”-Objekt besitzt. Dieses gibt somit den obersten Namensraum vor welchen sich globales Objekt und DOM mit anderen Scripten teilen. Damit kann kein Script Bestandteile dieser für sich beanspruchen.

Ohne die Nutzung von Modularisierung in Javascript Code werden sämtliche Variablen Global angelegt sprich sie sind Teil des “window“-Objektes wie der Verfasser an diesem abstrahierten Beispiel zeigt:

```
1  var wert1 = “wert1”;
2  function funk1()
3  {
4
5  };
```

In diesem Codebeispiel wären sowohl Variable “wert1” als auch Funktion “funk1” Teil des “window”-Objektes sprich:

```
1  window.wert1;
2  window.funk1();
```

Das bedeutet zum einen, das diese Variable und Funktion von überall zugänglich sind und der Entwickler sich über die Namensvergabe extrem viele Gedanken machen muss da die Namen global vergeben sind. Diese Tatsache bietet einige Sicherheitslücken und sollte deswegen möglichst vermieden werden.

Um für bestimmte häufige Funktionen eine gute Lösung zu finden wurde für Funktionen wie:

- onload();
- onclick();
- onmouseover();

Das sogenannte unaufdringliche JavaScript eingeführt.

In diesem wird hierzu fortgeschrittenes Event-Handling verwendet welches somit mögliche Überschneidungen reduziert.

Um für alle anderen Funktionen, Variablen und Objekte einer Applikation die geschilderten Probleme zu umgehen und für eine Kapselung zu sorgen nutzt man deshalb die Modularisierung.[12]

In der Programmierung lässt sich der Begriff Modularisierung als ein Prinzip ähnlich mit Legosteinen beschreiben. Hierbei wird ein System aus wiederverwendbaren Teilen Strukturiert welche als Module bezeichnet werden.

Die Vorteile dienen hierbei zum einen in einer erhöhten Struktur außerdem wird aber auch eine unnötigen Duplizierung vermieden. Code der vorher an mehreren Stellen gebraucht wurde und dort erneut geschrieben wurde kann mithilfe der Modularisierung durch die Verwendung des Bausteines ausgelagert und dann einfach aufgerufen werden.

3.2 Modularisierungs Standards

Für den korrekten Aufbau eines Modules haben sich zunehmend bestimmte Standards bewährt. Diese bauen auf den Grundprinzipien modularer Programmierung auf.

Im Gegensatz zur Objektorientierten Programmierung bieten Module aber zum Beispiel keine Vererbung beziehungsweise Polymorphie.[13]

12 Vgl.: molily: „Organisation von JavaScripten: Module und Kapselung“, unter: http://molily.de/js/organisation-module.html (abgerufen am 02.04.2016)

13 Vgl.: Eisenblätter, Ludwig: Module und Bibliotheken, Uni Hamburg, Folie 7

3.2.1 Objekt Literal

In der Objekt Literal Notation wird ein Objekt als eine Sammlung von kommagetrennten Name/Werte Paaren in geschweiften Klammern beschrieben.

Die Namen innerhalb des Ojektes können entweder Strings oder Bezeichner sein auf welche ein Doppelpunkt folgt.

Auf das letzte Paar sollte kein Komme folgen, da dies sonst zu unerwarteten Fehlern führen kann.[14]

Als Beispiel für die Verwendung der Objekt Literal Notation hat der Verfasser ein Modul mit seinem Namen und einer Funktion, welche diesen ausgibt erstellt.

```
1  myModule = {
2        name: "robert",
3        testFunktion: function()
4        {
5             alert(this.name);
6        },
7  };
```

3.2.2 Privater Funktionsscope

Durch die Nutzung eines globalen Objektes als Namensraum im Objekt-Literal um darin eigene Objekte unterzubringen gibt es keine Trennung zwischen öffentlichen und privaten Daten. Diese Tatsache ist sinnvoll für Methoden welche von außen aufrufbar sind, für Eigenschaften ist es allerdings problematisch, da diese weiter von außen auf-ruf-, les- und manipulierbar sind.

Um diese Problematik zu umgehen muss also eine Gültigkeitsbereich für die Variablen definiert werden. Diese werden also gekapselt und es entsteht ein privater Scope. Ein Scope lässt sich in JavaScript nur durch eine Funktion erzeugen. Aus diesem Grund wird für den "Privaten Funktions Scope" das gesamte Script gekapselt. Darin angelegte Variablen und Funktionen tasten somit nicht mehr den globalen Scope an.[15]

14 Vgl.: Osmani, Addy: Learning JavaScript Design Patterns, 1. Auflage, O'Reilly, S. 27

15 Vgl.: molily: „Organisation von JavaScripten: Module und Kapselung“, unter: http://molily.de/js/organisation-module.html (abgerufen am 02.04.2016)

3.2.3 AMD

AMD steht als Abkürzung für “Asynchronous Module Definition” und bezeichnet ein Modell in welchem die Module und die Abhängigkeiten asynchron geladen werden können.

Dies bedeutetet das mehrere Prozesse parallel und unabhängig von Zeitintervallen ausgeführt werden.

Dieses Modell besteht aus zwei Hauptfunktionen. Zum einen besitzt es die define()-Funktion welche das Modul mit seiner ID, Abhängigkeiten und eine Funktion definiert.

Neben dieser Funktion besitzt es außerdem die require()-Funktion. Diese Funktion beinhaltet ein Array mit ID’s welche geladen werden sollen

AMD wird von Zahlreichen Projekten wie Dojo, MooTools, Firebug und sogar jQuery genutzt.

```
1  define(['jquery'], function ($) {
2      //    Methoden
3      function myFunc(){};
4
5      //    öffentliche Methoden
6      return myFunc;
7  });
```

3.2.4 CommonJS

CommonJS dient primär als simple API zum serverseitigen Zuweisen von Modulen. Im Gegensatz zu AMD enthält CommonJS mehr Zugriffsmöglichkeiten auf Eingaben, Ausgaben, Dateisystemzugriffe und mehr.[16]

Aus struktureller Sicht ist ein CommonJS Modul ein wiederverwendbarer Schnipsel JavaScript welcher spezifische Objekte exportiert die für jedem abhängigen Code verfügbar gemacht werden können.

Typischerweise sind diese Module nicht in Funktionen verschachtelt wie es bei AMD in der define()-Funktion der Fall ist.

Allgemein beinhalten CommonJS drei Hauptteile: eine freie Variable beziehungsweise ein Objekt bezeichnet als exports welche die Objekte beinhaltet die ein Modul benötigt

16 Vgl.: Osmani, Addy: Learning JavaScript Design Patterns, 1. Auflage, O‘Reilly, S. 141

um für andere Module verfügbar zu sein, eine require Funktion welche Verweise auf notwendige Module deklariert und eine module Variable in welcher die Metadaten des Moduls abgelegt werden.

```
1  var $ = require('jquery');
2  function myFunc(){};
3  module.exports = myFunc;
```

3.2.5 UMD

UMD auch Universal Module Definition wird als die Schnittstelle und Vereinigung von AMD und CommonJS bezeichnet. Es bietet ein universelles Pattern welches beide Stile unterstützt.

Die Folge dieser Vielseitigkeit ist eine sehr chaotisch wirkende Struktur

```
1  (function (root, factory) {
2      if (typeof define === 'function' && define.amd) {
3          // AMD
4          define(['jquery'], factory);
5      } else if (typeof exports === 'object') {
6          // Node, Ähnlich Common JS
7          module.exports = factory(require('jquery'));
8      } else {
9          // Globalen, root ist das window Objekt
10         root.returnExports = factory(root.jQuery);
11     }
12 }(this, function ($) {
13     //    Methoden
14     function myFunc(){};
15
16     //    öffentliche Methoden
17     return myFunc;
18 }));
```

3.2.6 ES6-Modules

Die ECMAScript 6 Spezifikation ist ein sich in der Entwicklung befindlicher Entwurf welcher die Änderungen und Features der nächsten Version von JavaScript umreißt. Aufgrund des Entwicklungsstandes ist eine breite Browserkompatibilität daher nicht gegeben.[17]

Bei der Verwendung von ES6 Modulen gibt es im Vergleich zum bisherigen Umgang mit Modulen einige Unterschiede. Wichtig ist hierbei vor allem, das ES 6 Module dateibasiert sind. Das bedeutet das pro Modul eine Datei erstellt wird. [18]

Der Vorteil hierbei ist, dass man bei einer größeren Anwendung nicht wie sonst eine große Datei mit allen Modulen sondern jetzt für jedes Modul eine kleine einzelne Datei lädt. Dies bring einen Vorteil in Sachen Optimierung und Performance.

3.2.7 Modulloader

Bei einer “normalen“ Website nutzt man <script>-Tags um verschiedene Skripte zu laden. Dabei wird allerdings die Website solange blockiert, bis alle JavaScript Skripte geladen wurden. Zwar kann diese Wartezeit reduziert werden, indem man alles in einer Datei ablegt welche dann auch noch minifiziert wird, doch muss die Seite immernoch auf das Laden des Skriptes warten.

An dieser Stelle wurde die Verwendung sogenannter Modulloader eingeführt mit welchen es möglich wird Skripte asynchron und parallel zu laden.

Die meisten Modulloader folgen einem ähnlichen Prinzip in ihrer Verwendung. Als erstes werden die Dateinamen übergeben welche geladen werden sollen:

```
1  genericScriptLoader.load('jquery.js', 'underscore.js');
```

Durch das asynchrone oder parralele hinzu laden von Skripten ist es jedoch nicht möglich einfach Funktionen oder Code auszuführen welcher von den externen Resourcen abhängig ist da nicht garantiert werden kann ob die Abhängigkeiten bereits geladen wurden. Die Lösung die Modulloader dafür bieten, ist eine Methode welche ausgelöst wird wenn alle Skripte geladen wurden:[19]

17 Vgl.: Fain, Rasputnis: „Enterprise Web Development“, 1. Auflage, O‘Really, S. 229

18 Vgl.: Simpson, Kyle: „You Don‘t Know JS: ES6 and Beyond“, 1. Auflage, O‘Really, S. 100

```
genericScriptLoader.ready(function(){
 //Eigener Code der von Skripten abhängig ist
});
```

3.2.7.1 HeadJS

HeadJS ist ein Modulloader welcher seit langem kein Update bekommen hat wie man auf der HeadJS Seite sehen kann. Trotzdem wird es aber von Jack Franklin als ein exzellenter Loader bezeichnet und wird deshalb vom Verfasser mit behandelt.

Dieser Loader ermöglicht es neben JavaScript außerdem CSS asynchron und parralel zu laden.

Die Verwendung erfolgt durch hinzufügen der HeadJS Datei. Als Attribut lässt sich außerdem die zu ladende Datei bei der Initiierung definieren:

```
<head>
<script src="head.min.js" data-headjs-load="init.js"></script>
</head>
```

In der definierten Datei init.js finden sich dann die Verweise auf die externen Skripte:

```
head.load("skript1.js", "skript2.js");
```

Mit einer wie im vorherigen Abschnitt beschriebenen Methode kaann dann der eigentliche Code ausgeführt werden wenn ein jeweiliges Skript fertig geladen wurde:

```
head.ready("skript1.js", function() {});
head.ready("skript2.js", function() {});
```

3.2.7.2 RequireJS

RequireJS wird ähnlich wie HeadJS mit einem Skript Tag hinzugefügt. Dabei enthält es neben dem Source Attribute welches auf die RequireJS Datei verweist außerdem ein Attribut welches auf eine Skript Datei zur initialisierung verweist:

```
<script data-main="main.js" src="require.js"></script>
```

In der hier verwendeten Datei main.js werden dann alle weiteren Konfigurationen vorgenommen. Hier werden die externen Skripte geladen und darin enthaltende Methoden gegebenenfalls ausgeführt:

---

19 Vgl.: Franklin, Jack: „Essential JavaScript: the top five script loader“, unter: http://www.creativebloq.com/javascript/essential-javascript-top-five-script-loaders-8122862 (abgerufen am 06.05.2016)

```
1  require(["skript1"],function(skript1){
2    skript1.funktion1();
3  });
4  require(["skript2"],function(skript2){
5    skript2.funktion1();
6  });
```

3.2.8 Entwurfsmuster

3.2.8.1 Model View Controller

Das Entwurfsmuster Model View Controller besteht aus 3 Komponenten. Es besitzt zum einen den View, die Präsentationsschicht. Diese stellt benötigte Daten aus dem Modell dar und nimmt Aktionen des Benutzers entgegen. Desweiteren besitzt es das Model welches die Daten enthält und Datenänderungen mithilfe des Observer Pattern meldet. Zuletzt gibt es noch den Controller welcher die Steuerung im Modell übernimmt. Diese Steuerung dient zur Verwaltung der Views und reagiert auf die Benutzereingaben.

3.2.8.2 Model View Presenter

Das Entwurfsmuster Model View Presenter besitzt ähnlich wie Model View Controller drei Teile. Allerdings besitzt dieses statt dem Controller den Presenter. Dieser übernimmt eine ähnliche Funktion des Controllers und verbindet Model und View. Im Falle von MVP werden für den Presenter allerdings meistens Interfaces verwendet.

3.2.8.3 Model View ViewModel

Dieses Entwurfsmuster gilt als eine Abwandlung des MVC Musters. Hierbei wird auf eine Abkapselung der Programmlogik von der Darstellung abgezielt. Im Gegensatz zu den Beiden vorrangegangenen Entwurfsmustern besitzt das MVVM als dritten Teil das ViewModel. Dieses ermöglicht eine Verbindung von View und Model. Zum einen in der Form das es vom Model Informationen erhält. Zum anderen gibt es dem View öffentlich definierte Eigenschaften und Befehle mit.

Dieses Entwurfsmuster liefert die Möglichkeit, Oberflächendesigner unabhängig von Entwicklern arbeiten zu lassen.

3.2.9 Fazit

In diesem Abschnitt wurde erkennbar das das Gebiet der modularen Programmierung vor allem sehr weit gefächert ist und es einige Prinzipien gibt die sich durchgesetzt haben und heute als Standard gelten. Für den Verfasser ist es wichtig im Verlauf dieser Arbeit zu analysieren an welchen Stellen die in den vorherigem Abschnitt beschriebenen Prinzipien zur Verwendung kommen.

Außerdem sollte zum Ende der Arbeit geklärt werden auf welchen Entwurfsmustern jeweilige Frameworks basieren.

4 Frameworks

Um dem Entwickler die Modularisierung und das Objektorientierte arbeiten zu erleichtern gibt es die Möglichkeit Frameworks beziehungsweise Bibliotheken zu nutzen. Neben diesem Vorteil gibt es aber auch weiter Gründe Frameworks zu nutzen. Zum einen die Möglichkeit des URL Routings aber auch das nutzen von Data Bindings.

4.1 Betrachtete Frameworks

In dieser Arbeit hat sich der Verfasser für 5 Frameworks entschieden welche unter gezielten Kriterien verglichen werden sollen.

4.1.1 AngularJS

AngularJS ist ein weitumfassendes Javascript Framework nachdem Entwurfsmuster MVC.

Diese Framework macht es dem Entwickler sehr einfach schnell Anwendungen zu konstruieren welche sowohl auf Desktop als auch auf mobilen Platformen stabil und gut laufen.

Die mit AngularJS konstruierten Anwendungen sind Client seitige Anwendungen. Somit laufen alle Teile der Anwendung wie model, view, controller und alle HTML Teile auf dem Endgerät. Lediglich nötige Daten müssen sich dann mittels REST Service oder ähnlichem geholt werden.

Eine Kernfunktionalität in AngularJS sind Controller, welche in einigen anderen Frameworks gänzlich fehlen.

4.1.2 Backbone JS

BackboneJS ist eine Bibliothek welche nach dem Entwurfsmuster MVP arbeitet.

Neben der kompakten Größe ist die größte Stärke von Backbone.js seine Vielseitigkeit. Anders als andere Frameworks ist Backbone.js nicht besonders eigensinnig.

So bringt es zum Beispiel keine eigene Templating Engine mit sich, was dem Entwickler ermöglicht selbst zu wählen was sich bei einem Projekt am besten anbietet.

Da Backbone.js so leichtgewichtig ist empfiehlt es sich vor allem durch seine hohe Geschwindigkeit für einfache Projekte, zum Beispiel Single Page Apps oder Widgets als Teil einer Website.

BackboneJS ist außerdem von der JavaScript-Bibliothek Underscore.js abhängig.

4.1.3 Ember.js

Ember ist auch ein Framework welches auf dem Entwurfsmuster MVC basiert.

Das Framework nutzt für sein Data Binding eine modifizierte Version der Template-Engine Handlebars.js und sorgt damit auch für eine automatische Aktualisierung des HTML-Dokumentes bei Änderungen am Datenmodell.

4.1.4 Vue.js

Der Verfasser hat sich neben relativ bekannten und lange existierenden Frameworks außerdem für Vue.js entschieden. Vue.js lässt sich zwar nicht als komplettes Framework sondern mehr als Bibliothek beschreiben liefert aber für kleine Clientseitige Anwendungen alle Funktionen.

Dieses Framework arbeitet nach dem Entwurfsmuster MVVM und ist stark von AngularJS inspiriert.

4.2 Designpattern

Modulare Programmierung richtet sich nach Designpattern welche zur sauberen Struktur von Modulen dienen. Diese Designpattern sind wiederverwendbare Lösungen für häufig auftretende Problem beim Softwaredesign. Diese kann man auch als Vorlagen für die Lösung von Problemen beschreiben.

Der Vorteil dieser Pattern ist das sie bewiesene Lösungen für Probleme sind welche man einfach verwenden kann und wiederholen kann.

4.2.1 Constructor Pattern

Als Konstruktor bezeichnet man in der objektorientierten Programmierung eine spezielle Methode welche zur Initialisierung neu erstellter Objekte genutzt wird. Da in JavaScript fast alles Objekte sind nutzt man daher Objekt-Konstruktoren.[20]

```
1  function Student( name, alter, semester ) {
2       this.name = name;
3       this.alter = alter;
4       this.semester = semester;
5       this.toString = function () {
6            return this.name + " ist im " + this.semester + ". Se-
   mester";
7       };
8  }
9  var rob = new Student( "Robert Keller", 22, 8 );
10 var maxi = new Student( "Max Mustermann", 24, 8 );
11
12 console.log( rob.toString() );
13 console.log( maxi.toString() );
```

In diesem Beispiel wird das Constructor Pattern zur Erstellung von zwei Studenten verwendet welche dann mit eigener toString() Methode in der Konsole ausgegeben werden.

20 Vgl.: Osmani, Addy : „Learning JavaScript Design Patterns“, unter: http://addyosmani.com/resources/essentialjsdesignpatterns/book/#antipatterns (abgerufen am 22.04.2016)

4.2.2 Module Pattern

In der Softwaretechnik sollte das Module Pattern ursprünglich zur Abkapselung wiederverwendbaren Codes genutzt werden. In Javascript wird dieses zur Darstellung des Klassenkonzeptes genutzt. Hiermit ist es möglich sowohl public und private Methoden als auch Variablen in einem einzelnen Objekt zu erzeugen welche den gekapselten Code anderen globalen Scope Objekten unzugänglich machen.

Als Scope bezeichnet man in der JavaScript Programmierung eine Satz Variablen, Objekte und Funktionen zu denen man Zugriff hat.[21]

Diese Abkapselung ermöglicht es den Konflikt zu umgehen, Funktionsnamen zu definieren welche in anderen Scripten im selben Anwendungs Scope genutzt werden.

Das Module Pattern lässt sich zwar sehr gut auf eher kleine Anwendungen anwenden, in größeren Projekten sollte man darauf aber nicht zurück greifen. Dies liegt vor allem daran, dass man bei einer großen Zahl von Modulen viele zusätzlichen Codebausteine hinzufügen und das Vorhandensein von Objekten im globalen Scope für jedes Modul überprüfen muss.

Außerdem ist hier eine besondere Vorsicht mit Namensgebungen notwendig um versehentliche Konflikte im Globalen Scope zu umgehen.

Der größte Nachteil des Module Pattern ist das man trotz allem selbst die Abhängigkeiten mithilfe von <script> Tags verwalten muss.

21 Vgl.: w3schools.com: „JavaScript Scope“, unter: http://www.w3schools.com/js/js_scope.asp (abgerufen am 23.04.2016)

4.2.3 Revealing Module Pattern

Als Revealing Module Pattern wird ein Entwurfsmuster bezeichnet welches Objekt-Literal und Kapselfunktion kombiniert. Mithilfe der Kapselfunktion wird hierbei ein Objekt nach draußen gegeben bevor dies beendet wird über welches einzelne Methoden aufgerufen werden können.

Das zurückgegebene Objekt mit den verfügbaren Funktionen kann hierbei als öffentliche API beschrieben werden.

Durch diese Kombination von Objekt-Literal und Kapselung ist es möglich innerhalb eines Modules private und öffentliche Objekte zu kombinieren und nur notwendige Objekte öffentlich zugänglich zu machen.[22]

```
1  var Modul = (function () {
2
3      /* ... private Objekte ... */
4
5      /* Gebe öffentliche API zurück: */
6      return {
7          öffentlicheMethode : function () { ... }
8      };
9
10 })();
```

22 Vgl.: Timms, Simon: „Mastering JavaScript Design Patterns“, 1. Auflage, Packt Publishing, S. 39

4.2.4 Singleton Pattern

Betrachtet man die meisten Pattern so wird schnell auffällig das es einfach möglich ist, mehrere Objekte einer Klasse zu erstellen. Das Singleton Pattern befasst sich mit dieser Problematik und schafft die Möglichkeit, Klassen zu entwickeln von denen nur ein Objekt erstellt werden kann.

Der Vorteil hierbei ist, das der Speicherbedarf reduziert wird und eine verspätete initialisierung ermöglicht wird

Recherchiert man nach der Ausführung des Singleton Pattern in JavaScript so stößt man auf zahlreiche Varianten. Als einfache Schreibweise findet sich im Buch „Learning JavaScript Design Patterns“ ein Objekt-Literal gruppiert mit seinen zugehörigen Methoden und Eigenschaften.[23]

```
1  var meinSingleton = {
2       eigenschaft1: "einzigartig",
3       eigenschaft2: "speicherarm",
4       methode1: function () {
5            console.log('Mein Singleton!');
6       }
7  };
```

23 Vgl.: Osmani, Addy: „Learning JavaScript Design Patterns“, 1. Auflage, O‘Reilly, S. 25

4.2.5 Mediator Pattern

Das Mediator Pattern dient dazu, ein Objekt mit allen verschiedenen Teilen einer Applikation zu teilen. Häufig wird dieses Pattern genutzt, um die Kommunikation zwischen verschiedenen Programmteilen und Funktionen zu ermöglichen.

Dieses Pattern wird dann genutzt wenn in einer Anwendung verschiedene Module zu viele Beziehung zu anderen Modulen beziehungsweise untereinander besitzen. Der Mediator ist dann der zentrale Punkt der Kommunikation.[24]

4.2.6 Prototype Pattern

Das Prototype Pattern ermöglicht es, Objekte basierend auf Vorlagen zu erstellen. In diesem Pattern wird die Vorlage im Prinzip geklont und ein neues Objekt daraus erstellt.

Im ECMAScript 5 lässt sich dieses Pattern durch Object.create realisieren:

```
1  var einStudent = {
2  lernen: function() {},
3  name: 'Max'
4  };
5
6  var andererStudent = Object.create( einStudent );
7  console.log(andererStudent.name);
```

4.2.7 Command Pattern

Mit dem Command Pattern ist es möglich, Operationen zu einer Warteschlange zu sammeln. Dies wird mithilfe von Kapselung jedes Befehles erreicht. Durch die Verwendung einer Warteschlange ist es somit außerdem möglich, ein Log zu erstellen und einzelne Operationen rückgängig zu machen.

Der Vorteil dieses Pattern liegt mit der Möglichkeit Veränderungen durchzuführen klar auf der Hand, doch durch die Verwendung der Kapselung bedeuten beim Command Pattern viele Operationen auch gleichzeitig viele Klassen und die Umsetzung kann schnell unübersichtlich werden.

---

24 Vgl.: Osmani, Addy: „Learning JavaScript Design Patterns“, 1. Auflage, O‘Reilly, S. 21

4.2.8 Observer Pattern

Möchte man in einer Anwendung implementieren das mehrere Anwendungsbestandteile über Änderungen eines Subjektes informiert werden, so ist es normalerweise notwendig, das sich ändernde Subjekt regelmäßig anzufragen. Da diese Problemlösung allerdings eine extreme Rechenzeit verursachen würde wurde das Observer Pattern entwickelt. Das Observer Pattern ist eine Weiterführung des Command Pattern.

Dieses Pattern sorgt dafür, das bei Änderung des Subjektes die anderen Anwendungsbestandteile informiert werden.

Philipp Hauer erklärt dieses Pattern auf seiner Website mithilfe eines Zeitungsabonements.

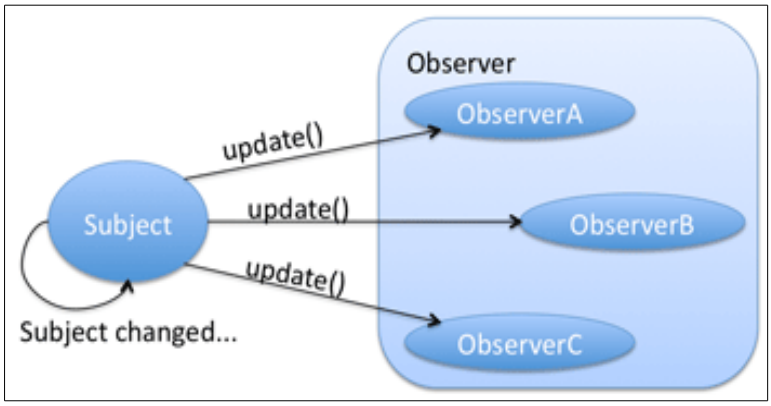

Abbildung 5: Observer Pattern nach Philipp Hauer | Quelle: http://www.-philipphauer.de/study/se/design-pattern/observer.php

4.2.9 Facade Pattern

Das englische Wort Facade lässt sich mit dem deutschen Fassade übersetzen. Dieses wird im deutschen als die Darstellung nach außen beschrieben. Dieses Designpattern liefert ein Interface zu einem großem Teil Code ohne dabei die eigentliche Komplexität des Codes preiszugeben.[25]

Verwendet wird dieses Pattern unter anderem auch in der Bibliothek jQuery.

Addy Osmani zeigt in seinem Buch „Learning JavaScript Design Patterns“ aus der Bibliothek jQuery ein Beispiel des Facade Pattern anhand der $(document).ready() Funktion.

Als Nutzer verwendet man diese Funktion welche innerhalb von jQuery eigentlich die Methode binReady() nutzt. In dieser werden unter anderem verschiedene Browser berücksicht um eine Browserkompatibiliät der Bibliothek zu gewährleisten.

4.2.10 Factory Pattern

Müssen in einer Anwendung Objekte mit hoher Komplexität erstellt werden, verschiedene Instanzen abhängig von der Umgebung oder kleine Objekte mit den selben gemeinsamen Vorgaben erzeugt werden so empfiehlt sich die Verwendung des Factory Pattern. [26]

Das Factory Pattern löst diese Problmatik indem man für die Erstellung der Objekte zunächst eine Schnittstelle definiert in welcher wiederum Unterklassen vorgeben welche Klasse instanziert wird.

Häufig findet sich dieses Pattern in JavaScript Frameworks.

25 Vgl.: Osmani, Addy: „Learning JavaScript Design Patterns“, 1. Auflage, O‘Reilly, S. 56

26 Vgl.: Osmani, Addy: „Learning JavaScript Design Patterns“, 1. Auflage, O‘Reilly, S. 59

4.2.11 Mixin Pattern

Möchte man ein Objekt erzeugen und ihm Beispielsweise später eine Sammlung von Funktionen hinzufügen so empfiehlt sich das Mixin Pattern. In JavaScript wird dieses Pattern mittels generischer Programmierung umgesetzt.

4.2.12 Decorator Pattern

Als Alternative zur Bildung vieler Unterklassen lässt sich das Decorator Pattern verwenden. Als Beispiel für Unterklassen lässt sich das folgende Zeigen:

```
1  var Mensch = function( vorname , nachname ){
2        this.vorname = vorname;
3        this.nachname = nachname;
4        this.geschlecht = 'männlich'
5  };
6
7
8  var robert = new Mensch( "Robert" , "Keller" );
9
10 var Student = function( vorname, nachname , themen ){
11       Mensch.call(this, vorname, nachname);
12       this.themen = themen;
13 }
14 Student.prototype = Object.create(Mensch.prototype);
15 var robertStudent = new Student( "Robert" ,"Keller" , ['pro-
   grammierung','mathe']);
16 console.log(robertStudent); //Ergibt Neben den Student Parame-
   tern auch das Geschlecht
```

Bei einem solch kleinen Beispiel ist die Verwendung noch legitim. Geht es aber um zahlrieche neue Eigenschaften oder Funktionen so ist die Verwendung von Unterklassen sehr unpraktikabel.

Hierfür werden jetzt die Decorator verwendet, die unser Interface lediglich erweitern.

Phillip Hauer erklärt dieses Pattern auf seiner Website anhand von Gerichten in einem Restaurant. Hierbei gibt es das Interface für das eigentliche Gericht, die Basisgerichte als Komponenten und die Beilagen als Decorators.

5 Anforderung und Analyse

Um zu verifizieren welches Framework wie gut für kleine, primär clientseitige Anwendungen verwendet werden könnte hat der Verfasser einen Vergleich angestellt. Hierzu wurde ein Vergleichsprojekt mit Vergleichskriterien simuliert.

Wichtig hierbei war es, ein sehr grundlegendes und allgemeingültiges Beispiel zu erstellen.

5.1 Vergleichsanwendung: To-Do-App

Für den Vergleich der Frameworks erstellte der Verfasser eine App zur Aufzeichnung von zu erledigenden Aufgaben, eine sogenannte To-Do-App.

Vor dem erstellen dieser App entwickelte der Verfasser eine Konzeption. Zunächst war entscheidend welche Funktionen die App bieten soll.

Dem Verfasser war es hierbei wichtig, weitestgehend Grundfunktionalitäten zu demonstrieren um Frameworks zu vergleichen. Die ToDo App sollte folgende Funktionen mitbringen:

- hinzufügen von Aufgaben

- Kategorisieren von Aufgaben

- Löschen von Aufgaben

- Speichern der Aufgabenliste

Außerdem sollte die App mit jedem Framework das selbe optische Grundgerüst beinhalten welches durch Bootstrap unterstützt wurde.

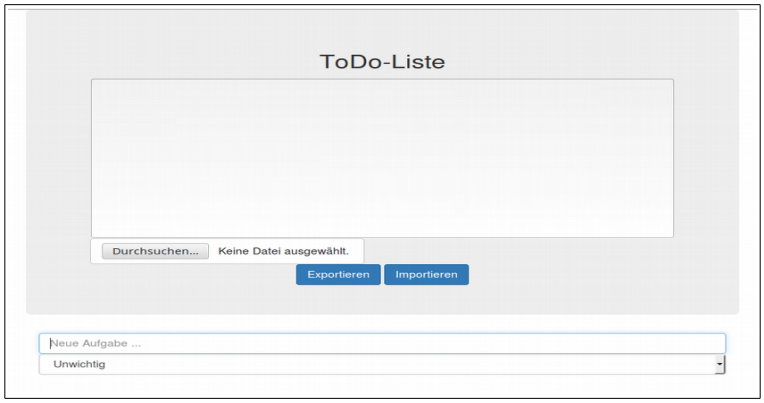

Abbildung 6: Grundstruktur der ToDo Anwendung des Verfassers

Jede Umsetzung beinhaltet neben dem Framework außerdem die verkleinerte Skriptdatei von JQuery, die verkleinerte CSS von Bootstrap und eine vom Verfasser erstellte CSS Datei.

Der Verfasser nutzt bei der Erstellung der Analyse außerdem die Website todomvc.-com auf welcher MV* Frameworks anhand einer ToDo-App vorgestellt werden.

Mit der Nutzung dieser funktionierenden Quelle geht der Verfasser sicher, dass bei eventuellen Problemen in der praktischen Arbeit trotz immer eine Analyse möglich ist.

Die ToDo-App dieser Website bietet hierbei folgende Funktionen:

- Anlegen von Aufgaben

- Abschließen von Aufgaben

- Löschen von Aufgabenlist

- Temporäres Speichern der Liste

- Anzeigen von Beendeten, Aktiven oder allen Aufgaben mittels Routings

5.2 Vergleichskriterien

Um die in den vorherigen Kapiteln betrachteten Frameworks sinnvoll zu vergleichen, ist es nötig einige Vergleichskriterien zu definieren.

5.2.1 UI Binding und Data Binding

Das Data Binding oder auch im deutschen die Datenbindung sorgt für einen automatischen Datenaustausch zwischen Objekten. Hierbei werden vor allem Elemente der Benutzeroberfläche an ein Model gebunden. Realisiert wird diese Funktionalität oftmals mit dem aus den vorherigen Kapiteln erklärten Observer Pattern.

Im Vergleich zu diesem Kriterium möchte der Verfasser klären, ob das Framework Data Binding unterstützt und in welcher Form.

5.2.2 Frameworkgröße und Lauf-/Ladezeiten

Beim Test der Frameworkgröße hat der Verfasser verschiedene Testfälle konzipiert um die Frameworkgrößen zu überprüfen.

Testfall 1: Vergleich der Ordnergrößen des Kompletten Projektes des Verfassers und von ToDo-MVC.

Testfall 2: Vergleich der Dateigrößen der jeweiligen Frameworks..

Testfall 3: Vergleich der Dateigrößen der jeweiligen Frameworks mit notwendigen Abhängigkeiten.

Zur Laufzeitan- und Ladezeitenanalyse hat der Verfasser einige Testfälle erstellt um eine möglichst objektive Betrachtung aller Frameworks zu erstellen.

Neben den regulären Seitenaufrufen gehören dazu auch Seitenaufrufe unter größerer Belastung.

Testfall 1: Der Author ruft jede Seite unter regulären Bedingungen in den Browsern Google Chrome und Mozilla Firefox auf. Er testet dann die benötigten Ladezeiten mit Hilfe der Entwicklertools der jeweiligen Browser.

Testfall 2: Der Author speichert eine Liste von Aufgaben in 3 verschiedenen Längen: 10, 20 und 60 Aufgaben. Auch dabei wird die Laufzeit gemessen.

Testfall 3: Der Author importiert die 3 gespeicherten Aufgabenlisten und führt auch hierbei eine Laufzeitanalyse durch.

5.2.3 Funktionen

In diesem Kriterium prüft der Verfasser auf das Vorhandensein bestimmter Funktionen im Framework. Diese Umfassen vor allem die Erkenntnisse aus den vorherigen Kapiteln, besonders dem Abschnitt aktuelle Webtechnologien.

Zu den zu prüfenden Funktionen gehören aus dem Abschnitt aktueller Technologien:

- AJAX / XMLHttp Requests

- Mobile Unterstützung

- Browser Unterstützung

Die Überprüfung auf diese Funktionen erfolgt sowohl über eine Recherche als auch über einen Umfangreichen Test mit folgenden Testfällen:

Testfall 1: Der Verfasser testet die eigens entwickelte Anwendung und die Anwendung von ToDo-MVC in den Browsern Firefox, Chrome und Internet Explorer.

Testfall 2: Der Verfasser testet die eigens entwickelte Anwendung und die Anwendung von ToDo-MVC auf mehreren Endgeräten. Dazu gehören Smartphones und Tablets sowohl mit Android als auch mit iOS System und jeweils den Systemeigenen Browsern und dem Browser Chrome.

5.3 Umsetzung

5.3.1 AngularJS

Das erste Projekt der ToDo-App setzte der Verfasser mit dem Framework AngularJS um. Dabei nutzte er ein vorher erstelltes HTML und CSS Template um die immer gleiche Optik aller Anwenungen zu garantieren.

5.3.1.1 Aufbau des Frameworks

Zur Verwendung von Angular ist es dem Entwickler durch sogenannte Direktiven möglich ein reguläres HTML Template zu nutzen. Diese Direktiven werden zusätzlich zu normalen Attributen in die jeweiligen Tags eingeschlossen und ermöglichen damit bestimmte Angular Bezüge.

Zum Templating nutzt AngularJS hierbei ein eigenes Markup welches stark an Handlebars.js erinnert. Hierbei werden die Bezüge in doppelten geschweiften Klammern geschrieben. So lässt sich zum Beispiel der Aufgabentitel aus der ToDo-App mittels: {{task.title}} ausgeben.

Einige wichtige Direktiven hat der Author im folgendem aufgezeigt:

Die Direktive „ng-app“ bestimmt hierbei welcher Teil des HTML-Templates vom Framework verwaltet werden soll. Verwendet man hierbei zum Beispiel direkt den Tag <html> und ändert ihn in <html ng-app> so wird die gesammte Oberfläche durch Angular verwaltet.

Eine weitere Direktive ist „ng-model“. Diese sorgt für das Databinding an bestimmte Modellattribute. So ist es möglich diese Direktive zum Beispiel an einem Input Feld zu nutzen um daraus den Inhalt zu übernehmen. Ein Beispiel aus der ToDo-App wäre hierfür: <input ng-model=“todo.title“>

Außerdem wichtig ist eine Direktive welche dem Entwickler das Eventhandling erleichtert. Mit der Direktive „ng-click“ ist es möglich dem Klick-Event von zum Beispiel einem Button eine Funktion zuzuweisen.

5.3.1.2 Praktische Verwendung

Zur Erstellung der ToDo-Anwendung mit AngularJS hat der Verfasser zunächst das HTML-Template erstellt und die nötigen direktiven eingearbeitet. Das Template findet sich unter anderem in „Anlage 1: Template Angular App“.

Zu diesem Template erstellte der Verfasser einen passenden Controller welcher die nötigen Funktionen für eine ToDo-App Verwaltet. Auch dieser findet sich in den Anlagen unter „Anlage 2: Angular App Javascript“.

Durch das Zwei-Wege Databinding war es dem Verfasser sehr schnell möglich die nötigen Funktionen und Daten der App einzuarbeiten. Große Erleichterung brachte dabei auch die Verwendung der Handlebar Syntax.

5.3.1.3 Fazit

Während der Erarbeitung dieser Bachelorarbeit hatte der Verfasser den ersten wirklichen Kontakt mit Frameworks. Bei der praktischen Ausarbeitung war AngularJS das erste verwendete Framework. Gerade deshalb erstaunte es den Anwender das er alle gewünschten Funktionen ohne größere Schwierigkeiten implementieren konnte und äußerst schnell zum gewünschten Ziel gelangte. Nach der Erarbeitung dieser Anwendung kann der Verfasser sich vorstellen, in zukünftigen Projekten AngularJS zu verwenden.

Auch der Export und Import der Aufgabenliste funktioniert hierbei Problemlos und macht es dem Anwender möglich, seine Liste zu sichern.

5.3.2 EmberJS

5.3.2.1 Aufbau des Frameworks

Auch EmberJS bietet zum Aufbau des Codes und der Oberfläche zahlreiche Komponenten. Hier ist es allerdings nicht möglich wie bei Angular mit Direktiven zu arbeiten.

EmberJS setzt hierbei auf die Verwendung von Templates und nutzt dafür Handlebars. Handelbars.js ist eine Bibliothek welche es ermöglicht, dynamische Templates mit wenigem Code zu erstellen. Zur Verwendung dieser wird regulärer HTML Code mit Handlebar Ausdrücken kombiniert welche dazu führen, Das bestimmte Abschnitte beispielsweise Wiederholt werden.

Zur Erstellung von Oberflächenspezifischen Code nutzt Ember den View. Dieser repräsentiert hierbei ein bestimmtes Element des DOM und enthält darin unter anderem das zugehörige Eventhandling.

Zur Beschreibung von Objekten und deren Zustand verwendet Ember das Model. Hier können unter anderem Attribute für das Model festgelegt werden welche die Anwendungen dann Verarbeiten kann um Objekte anhand des Models zu erstellen.

Als Element zwischen dem Model und dem View steht bei Ember der Controller. Ember unterscheidet hierbei die Objekt und die Array Controller. Zur Verwaltung einzelner Modelle nutzt man hierbei den Objekt-Controller und für mehrere Modelle den Array-Controller.

Für die Abbildung bestimmter Templates auf bestimmte Pfade und zugehörige Modelle nutzt man in Ember den Router. Hierbei ist es demzufolge möglich an Routen sowohl Templates zu binden als auch passende Daten was wiederum durch einen Controller übernommen wird.

5.3.2.2 Praktische Verwendung

Zur Umsetzung der ToDo-Anwendung orientierte sich der Verfasser sehr stark an der ToDo-Anwendung von ToDo-MVC. Dies war nötig da er trotz der umfangreichen Dokumentation oft auf Probleme sties.

Zunächst passte der Verfasser hierfür das für Angular erstellt Template an und erstellte die hierfür nötigen Handlebar-Templates aus welchen Ember dann die Seite generiert. Dieses Template findet sich in „Anlage 2: ToDo-App Ember Template.

Der Verfasser erstellte dann ein Model für jede einzelne Aufgabe welches die drei Attribute für Titel, Priorität und Beendet enthält.

Um verschiedene Funktionen für jede einzelne Aufgabe zu ermöglichen erstellte der Verfasser einen Controller für die einzelne Aufgabe. Dieser enthält die nötigen Funktionen um die Aufgabe zu entfernen und abzuschließen.

Ein weiterer Controller wurde für die Sammlung aller Aufgaben erstellt. Dieser enthält die nötigen Funktionen um die Werte aus der Benutzeroberfläche mithilfe des Modells in eine Liste von allen Aufgaben abzulegen. Das anlegen dieser Liste verwendet die Speicherfunktion von Ember mit der zusätzlichen Bibliothek für lokales abspeichern dem „Localstorage Adapter“.

Außerdem wurde ein View angelegt für die Komponente des Input Feldes. In diesem wurde auch implementiert, das nach erfolgreichen anlegen der Aufgabe das Feld wieder geleert wird.

Auch der Quellcode dieser Anwendung findet sich in den Anlagen unter : „Anlage 3: EmberJS App“.

5.3.2.3 Fazit

Nach dem guten Ergebnis bei der Nutzung von AngularJS fiel es dem Author zunächst schwer mit Ember die gleiche Anwendung umzusetzen. Durch das Beispiel der Website ToDo-MVC gelang is ihm aber anhand der Vorlage das gewünschte Projekt zumindest in den Grundfunktionen umzusetzen.

Die vom Author gewünschte Funktion die Liste zu Importieren und zu Exportieren konnte er nicht Umsetzen.

5.3.3 BackboneJS

5.3.3.1 Aufbau des Frameworks

Im Falle von BackboneJS werden die Aufgaben die normalerweise ein Controller besitzt durch die Views übernommen. Zwischen diesen Views und den Models wird hierbei auf eine Eventgetriebene Kommunikation aufgebaut.

Zur Erstellung von Templates nutzt Backbone hier die externe Templating-Engine Underscore.

Backbone besitzt außerdem auch Router mit welchen es wie bei anderen Frameworks auch möglich ist URL‘s mit Methoden zu verknüpfen.

Durch das wegfallen des Controller nimmt Backbone eigentlich eine Sonderstellung in Sachen Entwurfsmuster ein und entwickelt eine eigene Form eines MV*-Pattern ähnlich dem MVP Muster.

5.3.3.2 Praktische Verwendung

Auch im Falle der Backbone Anwendung entschied sich der Author dafür unter anderem die ToDo-MVC Anwendung als Vorlage zu nutzen.

Aus dem Angular Template entwickelte er ein für Backbone funktionierendes Template. Diese besteht zum einem aus dem Formularbereich welcher in normalem HTML geschrieben ist. Zusätzlich besitzt die Datei aber auch ein Template welches die Struktur jeder Aufgabe innerhalb der Liste vorgibt.

Neben dieser HTML Datei besitzt auch die Backbone Anwendung ein Model. Dieses enthält wie bei Ember die Attribute für ein Aufgaben Objekt und gibt dafür auch Standardwerte vor. Außerdem enthält das Model die nötige Funktion um das Ändern des Status auch der Aufgabe innerhalb der Liste zuzuweisen.

Zum Abspeichern der Aufgabenliste wird eine Collection verwendet. In dieser wird zunächst das angelegte Model zugewiesen. Die Collection wird außerdem Lokal unter dem Namen ‘todos‘ abgespeichert.

Desweiteren wurde zwei Views angelegt. Zum einen existiert der View für die Anwendung. In ihm werden die Funktionen für die Oberfläche definiert. So zum Beispiel das hinzufügen einer Aufgabe.

Als weiteren View gibt es den View für jedes einzelne Aufgaben Element. Dieser nutzt das Angelegte Template aus der HTML Datei. Er enthält die für einzelne Aufgaben relevanten Funktionen und Events. Dazu gehört unter anderem das Beenden einer Aufgabe oder das löschen dieser aus der Liste.

Auch der Quellcode dieser Anwendung findet sich in den Anlagen unter : „Anlage 4: BackboneJS App“.

5.3.3.3 Fazit

Auch im Falle von Backbone nutzte der Verfasser als Vorlage die Website ToDo-MVC. Diese Vorlage erleichterte ihm die Erstellung des Beispiels enorm. Auch die Erfahrungen aus der Erstellung der Ember App konnten ihm hierbei bei der Lösung einiger Probleme helfen.

Den sich vorher gestellten Anspruch an die Anwendung konnte der Verfasser hierbei fast vollkommen erfüllen. Lediglich das importieren einer JSON Datei ist hierbei nicht möglich.

Die Funktion zum exportieren der Datei konnte hierbei aus der Angular Anwendung durch einige Modifizierungen übernommen werden.

5.3.4 VueJS

5.3.4.1 Aufbau des Frameworks

Durch die Anlehnung von VueJS ist schnell eine Ähnlichkeit mit AngularJS erkennbar. Auch VueJS benutzt zum Templating ein reguläres HTML Template in welches man zur Verwendung bestimmter Variablen Handlebar Code hinzufügt. Damit ist es zum Beispiel möglich, das vorhandene Angular Template weiter zu verwenden und nur geringfügig anzupassen.

Auch VueJS benutzt außerdem wie AngularJS Direktiven die im folgenden aufgezeigt sind:

Die Direktive „app“ welche als reguläre HTML-Id vergeben wird, bestimmt welcher Teil des HTML-Templates vom Framework verwaltet werden soll.

Als weiter Direktive findet sich „v-model“. Diese ist für das Databinding an bestimmte Modellattribute zuständig. So ist es auch hier wie bei Angular möglich diese Direktive zum Beispiel an einem Input Feld zu nutzen um daraus den Inhalt zu übernehmen. Ein Beispiel aus der ToDo-App wäre hierfür: <input v-model=“newTask“>

Außerdem wichtig ist eine Direktive welche dem Entwickler das Eventhandling erleichtert. Mit der Direktive „v-on:“ ist es dem Entwickler möglich, verschiedenste Events an Funktionen zu binden. Für einen Button mit Klick-Event würde das dann Beispielsweise so aussehen: <button v-on:click=“klickEvent()>Klick mich</button>

5.3.4.2 Praktische Verwendung

Neben den sehr bekannten Frameworks entschied sich der Verfasser auch für das eher unbekanntere Framework VueJS. Wie bereits im theoretischen Teil beschrieben lehnt sich dieses Framework sehr stark unter anderem an AngularJS an.

Als Template konnte der Verfasser aufgrund dieser Tatsache das Angular Template fast komplett übernehmen. Es war hierbei nur notwendig die Direktiven an VueJS anzupassen. Oftmals bedeutete das nur die Buchstaben „ng“ durch „v“ zu ersetzen.

Neben dem Template wurde lediglich eine JavaScript Datei angelegt. Diese enthält die Definition aller nötigen Funktionen. Außerdem werden die Daten definiert.

Auch der Quellcode dieser Anwendung findet sich in den Anlagen unter : „Anlage 5: VueJS App“.

5.3.4.3 Fazit

Nach der Erfahrung mit der Erstellung einer Angular App hatte der Verfasser wenige Probleme selbe Anwendung mit VueJS umzusetzen. Lediglich die für den Dateiupload nötige Direktive brachte Probleme so dass sich JSON Dateien nur mittel Textfeld importieren lassen. Der Verfasser konnte vor allem seine anfänglichen Zweifel wegen der relativen Unbekanntheit dieses Frameworks konnte er komplett ablegen.

5.4 Vergleich

5.4.1 UI und Databinding

BackboneJS bietet dem Entwickler nicht die Möglichkeit Databinding zu nutzen. Stattdessen nutzt Backbone eine Daten Model und nutzt zur Kommunikation jQuery Funktionen. Dies erschwert zwar in einigen Situationen die Anwendungsentwicklung allerdings sorgt es dafür, das jedes Modul unabhängig ist. Damit empfiehlt sich Backbone besonders für größere Webanwendungen.

EmberJS bietet dem Entwickler ein Databinding. Mithilfe doppelter geschweifter Klammern kann dieses Binding innerhalb des HTML Templates umgesetzt werden. Hiermit werden Variablen an ein Model gebunden. Allerdings ist es bei Ember notwendig, Abhängigkeiten genau zu definieren.

AngularJS bietet dem Entwickler ein extrem gut funktionierendes zwei Wege Databinding. Dabei werden sämtliche Daten der Anwendung ohne die Angabe von Abhängigkeiten neu berechnet und aktualisiert. Dies bringt den Vorteil das der Code stark vereinfacht wird und leichter lesbar ist.

Ähnlich wie AngularJS implementiert auch VueJS ein zwei Wege Databinding. Hier finden sich die doppelt geschweiften Klammern wieder. Das Databinding lässt sich sehr stark mit dem in Angular vorhanden vergleichen.

Die verschiedenen Arten von Databindings innerhalb der Frameworks sorgen dafür, das sich die Frameworks bereits damit in Anwendungsbereiche einordnen lassen. Durch das fehlen von Databinding in Backbone und die damit stärkere Unabhängigkeit lässt sich Backbone vor allem für größere Anwendungen verwenden. Die nötige Deklaration von Abhängigkeiten bei Ember führt zu einem Anwendungsbereich innerhalb kleiner bis mittlerer Anwendungen. Die Frameworks Angular und Vue können durch ihr zwei Wege Databinding vor allem in kleinen Anwendungen genutzt werden, was jedoch keinen Ausschluss aus größeren Komplexen Anwendungen bedeuten soll.

5.4.2 Frameworkgröße und Lauf-/Ladezeiten

Anhand der Ergebnisse der einzelnen Testfälle listete der Verfasser die Ergebnisse dieser Tests in den folgenden drei Tabellen auf.

	Angular	**Backbone**	**Ember**	**Vue**
Größe Ordner	139 kB	407,2 kB	2,4 MB	88,8 kB
Geladene Daten Firefox	Bootstrap: 119,67 KB 0,05 s	-||-	-||-	-||-
Geladene Daten Chrome Network	Alle: 454 B, 585 ms	Alle: 454 B, 571 ms	Alle: 454 B, 1,84 s	Alle: 454 B, 428 ms

Tabelle 2: Ordnergrößen und Ladezeiten einer eigenen ToDo-App

	Angular	**Backbone**	**Ember**	**Vue**
Größe Ordner	1,2 MB	462,3 kb	2,7 MB	320,4 kB
Geladene Daten Firefox	/	/	/	/
Geladene Daten Chrome Network	Alle: 0 B, 555 ms	Alle: 0 B, 308 ms	Alle: 0 B, 1,4 s	Alle: 0 B, 334 ms

Tabelle 3: Ordnergrößen und Ladezeiten der ToDo-MVC App

	Angular	**Backbone**	**Ember**	**Vue**
Hauptdatei	1,0 MB	72,4 kB	1,9 MB	244,7 kB
Zusätzliche Abhängigkeiten	26,9 kB + 36,0 kB = 62,9 kB	7,4 kB + 247,6 kB + 52,0 kB = 307 kB	413,5 kB + 15,2 kB + 100,9 kB + 247,4 kB = 777 kB	20,3 kB
Gesamt	1,062 MB	379,4 kB	2,677 MB	265 kB

Tabelle 4: Dateigrößen und Abhängigkeiten der Frameworks gemessen anhand der ToDo-MVC App

Aufgrund der unvollständigen Anwendung des Verfassers konnte Testfall 3 der Lauf- und Ladezeitenanalyse nicht durchgeführt werden.

Anhand dieser Tests fällt zunächst auf das die Performance des VueJS Frameworks auf dem ersten Platzt steht. Dies liegt vor allem an seiner geringen Größe und den we-

nigen nötigen Abhängigkeiten. Besonders auffällig ist vor allem die enorme Größe und Ladezeit welche sich bei der Verwendung von EmberJS findet.

5.4.3 Funktionen

Auch hier hat der Verfasser nach durchführung der Testfällte eine Tabelle mit allen wichtigen Ergebnissen angelegt.

	Angular	**Backbone**	**Ember**	**Vue**
AJAX / XMLHttp Requests	Ja	Ja	Nein(Nutzt jQuery Ajax)	Nein(Nutzt jQuery Ajax oder das vue-resource Plugin)
Mobile Unterstützung	Ja	Ja	Ja	Ja
Alle Browser unterstützt	Ja	Ja	Ja	Ja

Tabelle 5: Grundlegender funktionaler Vergleich der Framworks

Es lässt sich hier generell Sagen das die in dieser Arbeit getesteten Frameworks alle den modernen Anforderungen an clientseitigen Webapplikationen gerecht werden. Allerdings werden bei Ember und Vue Technologien wie Ajax durch jQuery realisiert.

Auffällig war außerdem das beim Testen mit dem iOS Browser Safari gelegentliche Probleme auftreten welche bei sonst keinem anderen Browser auftreten. Auch sind diese teils nicht reproduzierbar. Allerdings funktionieren die Anwendungen bei Benutzung von Chrome für iOS ohne Probleme.

Die Ergebnisse hierbei eigen, dass sich auch die Entwickler der Frameworks durchaus bewusst sind welchen Technologien und Funktionen für moderne Anwendungen zu beachten sind.

5.4.4 Schwierigkeit und Support

Dieser Vergleichsabschnitt soll unter anderem auch als Subjektiver Vergleich der Frameworks dienen. Durch die Erstellung der eigenen ToDo-Anwendung hat der Verfasser persönliche Erfahrungen mit den Frameworks gemacht und einen eigenen Eindruck gewonnen.

Während der Erstellung der Anwendung fielen dem Verfasser besonders die Frameworks AngularJS und VueJS sehr leicht. Durch die dortige Implementierung des zwei Wege Databinding und den damit verbundenen Verzicht auf die deklaration von Abhängigkeiten konnte er damit sehr schnell die Anwendung umsetzen.

Im Gegensatz dazu ist dem Verfasser die Umsetzung der Anwendung mit den komplexeren Frameworks Backbone und Ember sehr schwer gefallen.

Der Verfasser würde damit AngularJS als ein sehr einfach zu lernendes Framework deklarieren.

Um einen weiteren Vergleich in Sachen Support anzulegen hat der Verfasser die Website github.com genutzt und hierbei für jedes Framework die vergebenen Sterne analysiert. Dabei ergab sich folgendes Ergebnis:

BackboneJS: 24,780 Sterne

VueJS: 18.835 Sterne

EmberJS: 16.204 Sterne

AngularJS: 11.922 Sterne

Der Verfasser stellt hierbei die These auf, das sich anhand der vergebenen Sterne auch die Community bemessen lässt. Dies Bedeutet das wir auch jeden Stern als einen möglichen Entwickler zählen können der uns bei Problemen zur Seite steht.

Desweiteren untersucht der Verfasser die möglichen Ergebnisse einer einfachen Google Suche. Hierbei wird nach jedem Framework gesucht und die Zahl der Ergebnisse verglichen. Zu beachten hierbei ist, das dieses Ergebnis stark durch die Suchmaschinenoptimierung beeinflusst sein kann. Hierbei finden sich folgende Ergebnisse:

AngularJS: 58.500.000

BackboneJS: 32.100.000

VueJS: 17.600.000

EmberJS: 460.000

Vergleicht man diese beiden Analysen so lässt sich nur für BackboneJS eine hohe Stellung feststellen. Bei anderen Frameworks weichen die Ergebnisse sehr stark voneinander ab.

6 Schlussbetrachtung

In der Erstellung dieser Arbeit hat der Verfasser neue Erkenntnisse über die modulare Entwicklung von JavaScript Anwendungen erlangen können. Die Ergebnisse dieser Arbeit zeigen vor allem das es durch die rasante Weiterentwicklung unserer Technologien notwendig ist, auch die Anwendungsentwicklung daran anzupassen.

Zu Beginn dieser Arbeit stand der Anspruch JavaScript Frameworks auf ihre Tauglichkeit für kleine und clientseitige Anwendungen zu prüfen. Der Verfasser kann aus den Erkenntnissen dieser Arbeit und auf Basis seiner eigenen Kriterien ein relativ klares Ergebniss finden.

Aufgrund der relativ geringen Dateigröße, dem großen Funktionsumfang und dem gut funktionierendem Databinding empfiehlt es sich für den Verfasser solche kleinen clientseitigen Anwendungen vor allem mit AngularJS umzusetzen. Durchaus erscheint es ihm auch als sinnvoll dafür VueJS zu verwenden.

Das Framework BackboneJS ist hierbei eher für größere Anwendungen sinnvoll.

Die Umsetzung dieser Arbeit hat dem Verfasser außerdem aufgezeigt, dass die Verwendung von Frameworks neben zahlreichen Erleichterungen in der Entwicklung außerdem zu zahlreichen Problemen führen kann. Diese können unter anderem auch der zunehmenden Komplexität durch die Frameworks geschuldet sein.

Als zukünftige Perspektive wird der Verfasser einige Projekte mit den neu gewonnen Erkenntnissen umsetzen.

Literaturverzeichnis

Buchquellen:

- Governor, James: Web 2.0 Architectures, 1. Auflage, O'Reilly Verlag, Sebastopol
- Hunter, Rafter: Beginning XML, 4. Auflage, Wiley Publishing Inc
- Reimers,Thies: PHP 5.4 & MySQL 5.5, 4. Auflage, Galileo Computing Verlag
- Smith, Ben: Beginning JSON, 1. Auflage, Springer Science+Business Media New York
- Wenz, Christian: JavaScript und AJAX Das umfassende Handbuch, Galilio Computing Verlag
- Eisenblätter, Ludwig: Module und Bibliotheken, Uni Hamburg
- Osmani, Addy: Learning JavaScript Design Patterns, 1. Auflage, O‘Reilly
- Fain, Rasputnis: „Enterprise Web Development“, 1. Auflage, O‘Really
- Simpson, Kyle: „You Don‘t Know JS: ES6 and Beyond“, 1. Auflage, O‘Really
- Timms, Simon: „Mastering JavaScript Design Patterns“, 1. Auflage, Packt Publishing

Internetquellen:

- w3schools, unter: http://www.w3schools.com/browsers/browsers_stats.asp (abgerufen am 05.04.2016)
- Fletzberger, Stephan: „The State of JavaScript: Der Ist-Zustand“, Lichtenecker, unter: https://liechtenecker.at/the-state-of-javascript-der-ist-zustand/ (abgerufen am 23.04.2016)
- Finley, Klint: „I turned off JavaScript for a Whole Week and It Was Glorius“, WIRED, unter: http://www.wired.com/2015/11/i-turned-off-javascript-for-a-whole-week-and-it-was-glorious/#slide-1 (abgerufen am 07.05.2016)

- Google Trends, unter: https://www.google.de/trends (abgerufen am 02.05.2016)
- CanIUse mit dem Suchbegriff Javascript, unter http://caniuse.com/#search=javascript (abgerufen am 04.05.2016)
- Shaked, Uri: „AngularJS vs. Backbone.js vs. Ember.js“, AirPair, unter https://www.airpair.com/js/javascript-framework-comparison (abgerufen am 04.05.2016)
- molily: „Organisation von JavaScripten: Module und Kapselung“, unter: http://molily.de/js/organisation-module.html (abgerufen am 02.04.2016)
- Franklin, Jack: „Essential JavaScript: the top five script loader“, unter: http://www.creativebloq.com/javascript/essential-javascript-top-five-script-loaders-8122862 (abgerufen am 06.05.2016)
- Osmani, Addy : „Learning JavaScript Design Patterns“, unter: http://addyosmani.com/resources/essentialjsdesignpatterns/book/#antipatterns (abgerufen am 22.04.2016)
- w3schools.com: „JavaScript Scope“, unter: http://www.w3schools.com/js/js_scope.asp (abgerufen am 23.04.2016)

Anlagen

Anlage 1: Template Angular App

```
1  <!DOCTYPE html>
2  <html ng-app='todo'>
3      <head>
4          <title>ToDo App mit Angular JS</title>
5          <link rel="stylesheet" href="https://maxcdn.bootstrapcd-
   n.com/bootstrap/3.3.5/css/bootstrap.min.css" type="text/css" />
6          <link rel="stylesheet" href="css/style.css"
   type="text/css" />
7          <script type="text/javascript"
   src="js/angular.min.js"></script>
8      </head>
9      <body>
10         <div id="todoapp" ng-app="todo" ng-controller="ToDoCtrl"
   class="container-fluid">
11             <div class="col-md-8 col-md-offset-2 jumbotron text-
   center">
12                             <h2>ToDo-Liste</h2>
13                            <div class="col-md-12">
14                                <textarea id="importField" ng-
   model="text" class="col-md-12" rows="12"></textarea>
15                                <input type="file" class="btn btn-
   default" on-read-file="displayFileContents(contents)" />
16                             </div>
17                 <div class="col-md-12">
18                     <button class="btn btn-primary" ng-
   click="saveListJSON()">Exportieren</button>
19                 <button class="btn btn-primary" ng-click="im-
   portList()">Importieren</button>
20                 </div>
21                 </div>
22             <div id="" class="col-md-8 col-md-offset-2 form-
   group">
23                 <input type="text" name="add-task" class="col-
   md-8 form-control" placeholder="Neue Aufgabe ..." ng-
   model="taskTitle" ng-keyup="addTask($event)"/>
24                 <select ng-model="priority" class="col-md-4
   form-control">
25                     <option value='unwichtig'>Unwichtig</option>
26                     <option value='mittel'>Mittel</option>
27                     <option value='wichtig'>Wichtig</option>
```

```
28                  </select>
29              </div>
30              <div class="col-md-12"><hr /></div>
31              <ul id="todo-list" class="col-md-8 col-md-offset-2">
32                  <li ng-repeat="task in tasklist track by $index"
   ng-mouseenter="item= 'focus'" ng-mouseleave="item = 'nofocus'"
   ng-class='item'>
33                      <input class="col-md-1" type="checkbox"
   name="checkbox" ng-model="task.completed" ng-checked="complete-
   Task()"/>
34                      <span class="completed-{{task.completed}}
   col-md-6">{{task.title}}</span>
35                      <span class="priority {{task.priority}} col-
   md-4">{{task.priority}}</span>
36                      <button class="glyphicon glyphicon-remove
   col-md-1" title="Aufgabe entfernen" ng-click="deleteTask(tas-
   klist.indexOf(task))"></button>
37                      <div class="col-md-12"><hr /></div>
38                  </li>
39              </ul>
40            </div>
41          <script type="text/javascript" src="js/app.js"></script>
42      </body>
43 </html>
```

Anlage 2: Angular App Javascript

```
1  var toDoApp = angular.module('todo', [])
2
3  toDoApp.directive('onReadFile', function ($parse) {
4      return {
5          restrict: 'A',
6          scope: false,
7          link: function (scope, element, attrs) {
8              element.bind('change', function (e) {
9                  var onFileReadFn = $parse(attrs.onReadFile);
10                 var reader = new FileReader();
11                 reader.onload = function () {
12                     var fileContents = reader.result;
13                     scope.$apply(function () {
14                         onFileReadFn(scope, {
15                             'contents': fileContents
16                         });
17                     });
18                 };
19                 reader.readAsText(element[0].files[0]);
20             });
21         }
22     };
23 });
24
25
26 toDoApp.controller('ToDoCtrl', ['$scope', function($scope,
   $http) {
27     $scope.tasklist = [];
28
29     $scope.priority = 'unwichtig';
30
31     $scope.addTask = function(event) {
32         if(event.keyCode == 13 && $scope.taskTitle) {
33             $scope.tasklist.push({"title": $scope.taskTitle,
   "completed": false, "priority":
   $scope.priority});
34             $scope.taskTitle = "";
35             $scope.priority = 'unwichtig';
36
37         }
```

```
38      }
39       $scope.displayFileContents = function (contents) {
40          console.log(contents);
41          $scope.text = contents;
42      };
43
44      //Modularisierung CommonJS
45
46      $scope.deleteTask = function(index) {
47          $scope.tasklist.splice(index, 1);
48      }
49
50      $scope.importList = function () {
51          $scope.tasklist = angular.fromJson($scope.text);
52      }
53
54      $scope.saveListJSON = function () {
55              var data = $scope.tasklist;
56              var filename = "aufgabenliste.json";
57          filename = prompt("Bitte gib deiner Liste einen Namen:",
    filename);
58            if (!data) {
59              console.error('No data');
60              return;
61            }
62
63            if (!filename) {
64              filename = 'download.json';
65            }
66
67            if (typeof data === 'object') {
68              data = JSON.stringify(data, undefined, 2);
69            }
70
71            var blob = new Blob([data], {type: 'text/json'}),
72              e = document.createEvent('MouseEvents'),
73              a = document.createElement('a');
74
75            a.download = filename;
76            a.href = window.URL.createObjectURL(blob);
```

```
77           a.dataset.downloadurl = ['text/json', a.download,
   a.href].join(':');
78           e.initEvent('click', true, false, window,
79               0, 0, 0, 0, 0, false, false, false, false, 0,
   null);
80           a.dispatchEvent(e);
81         };
82
83 }])
```

Anlage 3: EmberJS App

```
1  <!doctype html>
2  <html data-framework="emberjs">
3          <head>
4                  <meta charset="utf-8">
5                  <title>To Do App mit Ember</title>
6                  <link rel="stylesheet" href="https://maxcdn.-
   bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css"
   type="text/css" />
7                  <link rel="stylesheet" href="css/style.css"
   type="text/css" />
8          </head>
9          <body>
10                 <script type="text/x-handlebars" data-template-
   name="todo-list">
11                         {{#if length}}
12                                 <section id="main">
13                                         <ul id="todo-list"
   class="col-md-8 col-md-offset-2">
14                                                 {{#each}}
15                                                         <li>
16 {{input type="checkbox" class="toggle col-md-1" checked=isCom-
   pleted}}
17 {{#if isCompleted }}<span class="col-md-6 completed-true" {{ac-
   tion "editTodo" on="doubleClick"}}>{{title}}</span>
   {{else}}<span class="col-md-6 completed-false" {{action "editTo-
   do" on="doubleClick"}}>{{title}}</span>{{/if}}
18 <span {{bind-attr class=':col-md-4 :priority priority'}}>{{prio-
   rity}}</span>
19 <button {{action "removeTodo"}} class="glyphicon glyphicon-remo-
   ve col-md-1" title="Aufgabe entfernen"></button>
20 <div class="col-md-12"><hr/></div>
21 </li>
22 {{/each}}
23                                         </ul>
24                                 </section>
25                         {{/if}}
26                 </script>
27                 <script type="text/x-handlebars" data-template-
   name="todos">
28                         <section id="todoapp">
```

```
29                      <div class="col-md-8 col-md-offset-2
   jumbotron text-center">
30                <h2>ToDo-Liste</h2>
31                <div class="col-md-12">
32                    <textarea id="importField" ng-model="text"
   class="col-md-12" rows="12"></textarea>
33                    <input type="file" class="btn btn-default"
   on-read-file="displayFileContents(contents)" />
34                </div>
35                <div class="col-md-12">
36                    <button class="btn btn-primary" {{action
   "exportList"}}>Exportieren</button>
37                    <button class="btn btn-primary" {{action
   "importList"}}>Importieren</button>
38                </div>
39            </div>
40
41            <div id="" class="col-md-8 col-md-offset-2 form-
   group">
42                {{todo-input class="col-md-8 form-control"
   type="text" value=newTitle action="createTodo" placeholder="Neue
   Aufgabe..."}}
43                {{view Ember.Select
44                            content=priorityList
45                            value=newPriority
46                            valueBinding="priority"
47                            class="col-md-4 form-control"
48                }}
49            </div>
50                        </section>
51                </script>
52                <script src="js/jquery.js"></script>
53                <script src="js/handlebars.js"></script>
54                <script src="js/ember.js"></script>
55                <script src="js/ember-data.js"></script>
56                <script
   src="js/localstorage_adapter.js"></script>
57                <script src="js/app.js"></script>
58                <script src="js/router.js"></script>
59                <script src="js/models/todo.js"></script>
```

```
60                <script
   src="js/controllers/todos_controller.js"></script>
61                <script src="js/controllers/todos_list_control-
   ler.js"></script>
62                <script
   src="js/controllers/todo_controller.js"></script>
63                <script
   src="js/views/todo_input_component.js"></script>
64                <script src="js/helpers/pluralize.js"></script>
65        </body>
66 </html>
```

ToDo-Model:

```
1  (function () {

2          'use strict';
3
4          Todos.Todo = DS.Model.extend({
5                  title: DS.attr('string'),
6                  priority: DS.attr('string'),
7                  isCompleted: DS.attr('boolean')
8          });
9  })();
```

ToDo-Controller:

```
1  (function () {
2        'use strict';
3        Todos.TodoController = Ember.ObjectController.extend({
4              actions: {
5                    removeTodo: function () {
6                          this.removeTodo();
7                    }
8              },
9              removeTodo: function () {
10                   var todo = this.get('model');
11                   todo.deleteRecord();
12                   todo.save();
13             },
14             saveWhenCompleted: function () {
15                   this.get('model').save();
```

```
16              }.observes('isCompleted')
17        });
18 })();
```

Anlage 4: BackboneJS App

```
1  <!doctype html>

2  <html>
3          <head>
4                  <meta charset="utf-8">
5                  <title>ToDo App mit BackboneJS</title>
6                  <link rel="stylesheet" href="css/style.css">
7                     <link rel="stylesheet" href="https://maxcdn.-
   bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css"
   type="text/css" />
8          </head>
9          <body>
10         <div class="container-fluid todoapp">
11          <div class="col-md-8 col-md-offset-2 jumbotron text-cen-
   ter">
12                            <h2>ToDo-Liste</h2>
13
14                          <div class="col-md-12">
15                                        <textarea id="importField"
   class="col-md-12" rows="12"></textarea>
16                                <input type="file" class="btn btn-
   default"/>
17                            </div>
18                 <div class="col-md-12">
19                       <button class="btn btn-primary export">Ex-
   portieren</button>
20                  <button class="btn btn-primary import">Importie-
   ren</button>
21                 </div>
22     </div>
23                 <section class="container-fluid">
24                                <div id="" class="col-md-8 col-md-
   offset-2 form-group">
25
26                                   <input class="new-todo col-md-8
   form-control" placeholder="Neue Aufgabe..." autofocus>
27                                     <select class="col-md-4 form-
   control priority-list">
```

```
28                      <option selected value='unwichtig'>Unwich-
   tig</option>
29                  <option value='mittel'>Mittel</option>
30                  <option value='wichtig'>Wichtig</option>
31              </select>
32              </div>
33                          <div class="col-md-12"><hr /></div>
34
35                      <section class="main">
36
37                              <ul id="todo-list" class="todo-
   list col-md-8 col-md-offset-2"></ul>
38                      </section>
39              </section>
40
41              <script type="text/template" id="item-template">
42                      <li class="view">
43                              <input class="col-md-1 toggle"
   type="checkbox" <%= completed ? 'checked' : '' %>>
44                              <span class="completed-<%- com-
   pleted %> col-md-6"><%- title %></span>
45                             <span class="col-md-4 priority <
   %- priority %>"><%- priority %></span>
46                             <button class="destroy glyphicon
   glyphicon-remove col-md-1" title="Aufgabe entefernen"></button>
47                                  <div class="col-md-12"><hr
   /></div>
48
49                      </li>
50              </script>
51              </div>
52              <script src="js/jquery.js"></script>
53              <script src="js/underscore.js"></script>
54              <script src="js/backbone.js"></script>
55                                                      <script
   src="js/backbone.localStorage.js"></script>
56              <script src="js/models/todo.js"></script>
57              <script src="js/collections/todos.js"></script>
58              <script src="js/views/todo-view.js"></script>
59              <script src="js/views/app-view.js"></script>
60              <script src="js/app.js"></script>
```

```
61         </body>
62 </html>
```

ToDo-Model:

```
var app = app || {};

(function () {
        'use strict';
        app.Todo = Backbone.Model.extend({

                defaults: {
                        title: '',
                        completed: false,
                        priority: 'Unwichtig'
                },
                toggle: function () {
                        this.save({
                                completed: !this.get('completed')
                        });
                }
        });
})();
```

ToDo-View:

```
1  var app = app || {};
2
3  (function ($) {
4          'use strict';
5
6          app.TodoView = Backbone.View.extend({
7
8                  tagName:  'li',
9
10
11                                 template:  _.template($('#item-
   template').html()),
12
13                 events: {
14                         'click .toggle': 'toggleCompleted',
```

```
15                          'click .destroy': 'clear'
16                  },
17
18
19                  initialize: function () {
20                              this.listenTo(this.model, 'change', this.render);
21                              this.listenTo(this.model, 'destroy', this.remove);
22                  },
23                  render: function () {
24
25                              this.$el.html(this.template(this.model.toJSON()));
26                          this.$el.toggleClass('completed', this.model.get('completed'));
27                          this.$input = this.$('.edit');
28                          return this;
29                  },
30                  toggleCompleted: function () {
31                          this.model.toggle();
32                  },
33                  clear: function () {
34                          this.model.destroy();
35                  }
36          });
37 })(jQuery);
```

Anlage 5: VueJS App

```
1  <!DOCTYPE html>

2  <html>
3      <head>
4          <title>ToDo App mit Vue JS</title>
5           <link rel="stylesheet" href="https://maxcdn.bootstrapcd-
   n.com/bootstrap/3.3.5/css/bootstrap.min.css" type="text/css" />
6                      <link  rel="stylesheet"  href="css/style.css"
   type="text/css" />
7
8                                    <script    type="text/javascript"
   src="js/vue.min.js"></script>
9
10     </head>
11     <body>
12         <div id="app" class="container-fluid">
13              <div class="col-md-8 col-md-offset-2 jumbotron text-
   center">
14                              <h2>ToDo-Liste</h2>
15
16                            <div class="col-md-12">
17                                      <textarea id="importField" v-
   model="jsonFile" class="col-md-12" rows="12"></textarea>
18                                   <input type="file" class="btn btn-
   default" v-model="fileInput" />
19                              </div>
20                 <div class="col-md-12">
21                                <button class="btn btn-primary" v-
   on:click="saveListJSON()">Exportieren</button>
22                   <button class="btn btn-primary" v-on:click="im-
   portListJSON(jsonFile)">Importieren</button>
23                 </div>
24                 </div>
25
26
27
28             <div class="col-md-8 col-md-offset-2 form-group">
29                    <input class="col-md-8 form-control" placehol-
   der="Neue  Aufgabe  ..."  v-model="newTask"  v-on:keyup.enter="add-
```

```
Task"/>
30                        <select class="col-md-4 form-control" v-
model="newPriority">
31                       <option selected value='unwichtig'>Unwich-
tig</option>
32                    <option value='mittel'>Mittel</option>
33                    <option value='wichtig'>Wichtig</option>
34                </select>
35            </div>
36            <div class="col-md-12"><hr /></div>
37            <ul id="todo-list" class="col-md-8 col-md-offset-2">
38                <li v-for="task in tasks">
39
40                          <input class="col-md-1" type="checkbox"
name="checkbox" v-model="task.completed" />
41                         <span class="completed-{{task.completed}}
col-md-6">{{ task.title }}</span>
42                     <span class="priority {{task.priority}} col-
md-4">{{ task.priority }}</span>
43                         <button class="glyphicon glyphicon-remove
col-md-1" title="Aufgabe entfernen" v-on:click="removeTask($in-
dex)"></button>
44                    <div class="col-md-12"><hr /></div>
45                </li>
46            </ul>
47                    </div>
48
49
50        <script type="text/javascript" src="js/app.js"></script>
51    </body>
52 </html>
```

App:

```
1  new Vue({
2    el: '#app',
3    data: {
4      newTask: '',
5      tasks: []
6    },
7    methods: {
```

```
8       addTask: function () {
9         var title = this.newTask.trim();
10        var priority = this.newPriority.trim();
11
12        if (title && priority) {
13           this.tasks.push({ title: title, priority: priority, com-
   pleted: false})
14          this.newTask = ''
15        }
16      },
17      removeTask: function (index) {
18        this.tasks.splice(index, 1)
19      },
20      editTask: function (index) {
21        var completed = this.newCompleted;
22        this.tasks[index].completed = completed;
23      },
24      importListJSON: function (file) {
25        this.tasks = JSON.parse(file);
26      },
27      saveListJSON: function () {
28        var data = this.tasks;
29              var filename = "Aufgabenliste.json";
30            filename = prompt("Gib deiner Liste einen Namen:", fi-
   lename);
31            if (!data) {
32              console.error('Keine Daten vorhanden!');
33              return;
34            }
35
36            if (!filename) {
37              filename = 'Aufgabenliste.json';
38            }
39
40            if (typeof data === 'object') {
41              data = JSON.stringify(data, undefined, 2);
42            }
43
44            var blob = new Blob([data], {type: 'text/json'}),
45              e = document.createEvent('MouseEvents'),
46              a = document.createElement('a');
```

```
47
48          a.download = filename;
49          a.href = window.URL.createObjectURL(blob);
50               a.dataset.downloadurl = ['text/json', a.download,
   a.href].join(':');
51          e.initEvent('click', true, false, window,
52                  0, 0, 0, 0, 0, false, false, false, false, 0,
   null);
53          a.dispatchEvent(e);
54
55     }
56
57
58   }
59 })
```

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe. Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Ort, den TT. Monat JJJJ Robert Keller