Seminararbeit

Von der Webseite zur Webapplikation

Neue Möglichkeiten in der Webentwicklung am Beispiel von JavaScript Frameworks

Eingereicht bei:	Prof. Dr. Klaus-Dieter Krägeloh
Eingereicht von:	Paul Mölders
Anschrift:	Bergerstr. 38, 58452 Witten
Studiengang:	Bachelor in Information Technology
Semester:	SoSe 2013
Matrikel-Nr:	2009024
Abgabe:	Juli 2013

Inhaltsverzeichnis

Zusammenfassung

In der vorliegende Seminararbeit wird untersucht, ob die Umsetzung von Webapplikationen durch den Einsatz von JavaScript Frameworks vereinfacht werden kann. Dazu werden die beiden Frameworks AngularJS und BackboneJS beschrieben, eingesetzt und bewertet.

Der Vorteil und die Notwendigkeit für den Einsatz der vorgestellten Framework-Technologien werden vor dem Hintergrund der Historie der Webapplikationen betrachtet.

Zur Evaluierung der Praxistauglichkeit, sowie zur Demonstration der jeweiligen Vor- und Nachteile wird eine Beispielapplikation umgesetzt. Diese zeigt ebenfalls die Veränderung gegenüber herkömmlichen Webanwendungen. Bei der Entwicklung der Beispielanwendung werden die beiden Frameworks erklärt und die unterschiedlichen Ansätze zur Implementierung von typischen Anforderungen an Webapplikationen geschildert.

In der Diskussion werden die Erkenntnisse aus der Umsetzung gegeneinander abgewogen. Zusätzlich werden Erweiterbarkeit und Popularität der Frameworks beleuchtet.

Als Resultat der Seminararbeit ergibt sich, dass der Einsatz eines JavaScript Framework bei der Umsetzung einer Webanwendung sinnvoll ist, wenn man die Gütekriterien für Webapplikationen in hohem Maß erfüllen will. Eine generelle Entscheidung für BackboneJS oder AngularJS kann nicht ausgesprochen werden, da die Ansätze sehr unterschiedlich sind. Beide Ansätze haben ihre Berechtigung und bringen jeweils Vor- und Nachteile mit, die vor der Entwicklung einer Webanwendung abgewogen werden sollten.

1 Einleitung

1.1 Problemstellung

Webanwendungen haben durch die rasante Verbreitung des Internets eine zentrale Rolle in vielen Unternehmen gewonnen. Durch die Vielzahl von Endgerät-Kategorien ist die stetige Erreichbarkeit sowie die Möglichkeit der Zusammenarbeit verschiedenster Nutzer ein schlagkräftiges Argument für webbasierte Anwendungen.

Die Technologie der Webanwendungen hat sich in den letzten Jahren ebenfalls erheblich weiterentwickelt, um neuen Herausforderungen adäquat nachzukommen. Die Anforderungen an Webanwendungen sind sehr variabel und vielseitig. Die Nutzerlast schwankt sehr stark, die Einsatzgebiete werden vielfältiger und zugleich werden die Nutzerbedürfnisse spezieller.

Für Webentwickler bedeutet dies, dass sie sich stets mit aktuellen Technologien auseinandersetzen müssen, um Chancen und Möglichkeiten zu erkennen und zu nutzen.

1.2 Zielsetzung

Die vorliegende Arbeit zeigt den technologischen Wandel bei der Erstellung von Webanwendungen. Hierbei werden die neuen Möglichkeiten in der Webentwicklung dargestellt, die im Vergleich mit herkömmlichen Technologien besonders in den Aspekten überzeugen, die durch die stark gestiegene Verbreitung und Nutzung in den Vordergrund getreten sind und somit primäre Attribute von Webanwendungen geworden sind.

Webentwickler haben die Grenzen der herkömmlichen Technologien erkannt und diese durch neue Wege und Möglichkeiten verblassen lassen.

1.3 Fragestellung

In der folgenden Seminararbeit wird ein Ansatz zur Entwicklung von modernen, auf JavaScript basierende Webapplikationen auf dem Client, beschrieben. Durch die Entwicklung einer Beispielapplikation wird verdeutlicht, dass eine deutliche Verbesserung, gemessen an den Gütekriterien für Webapplikationen (Calero, C., Ruiz, J., & Piattini, M., 2005) durch die Nutzung von JavaScript Frameworks wie AngularJS oder BackboneJS möglich ist. Dabei wurden zwei Frameworks ausgewählt, die auf sehr unterschiedliche Weise dasselbe Ziel

verfolgen: Erstellen von Webapplikationen mit hoher Performance und hoher Wartbarkeit, die dem Benutzer eine intuitive Bedienbarkeit und vielseitige Funktionalität bieten.

1.4 Aufbau der Arbeit

In Kapitel 1 werden die Problemstellung sowie die Zielsetzung und der Aufbau der Arbeit beschrieben.

Im zweiten Kapitel werden alte und neue Technologien zur Entwicklung von Webanwendungen verglichen. Es wird eine Beispielapplikation mit zwei neuen Technologien, AngularJS und BackboneJS, erstellt. Die Beispielapplikation zeigt die Vor- und Nachteile die durch den Einsatz einer dieser Technologien entstehen und verdeutlicht die unterschiedlichen Ansätze beider Technologien. Zum Vergleich dienen herkömmliche Ansätze zur Entwicklung von Webseiten mittels Technologien wie Active Server Pages (ASP.NET), Java Server Pages (JSP) oder PHP: Hypertext Preprocessor (PHP).

Im Anschluss wird in Kapitel 3 die Umsetzung auf Basis der vorgestellten Technologien verglichen und bewertet. Es wird ebenfalls auf Rahmenfaktoren wie Erweiterbarkeit, Umfang und Popularität von AngularJS und BackboneJS eingegangen.

2 Methode

2.1 Webanwendungen als Nachfolger von Desktop-Applikationen

Durch die Verbreitung des Internets wurden Softwareentwickler vor die Herausforderung gestellt, Webanwendungen zu erstellen, die auch solche Aufgaben übernehmen, die bisher von Desktop-Applikationen bewältigt wurden. Die Vorteile von Webanwendungen wie Benutzbarkeit ohne spezielle Hardware und Installationsaufwand, stetige Verfügbarkeit, flexible Nutzungsmöglichkeiten auf verschiedenen Plattformen sind der Grund dafür, dass die Technologie zur Entwicklung von Desktop-Applikationen vergleichsweise stagniert. Webtechnologien entwickeln sich schnell weiter und drängen mehr und mehr Desktop-Applikationen in den Hintergrund.

Der herkömmliche Ansatz zur Entwicklung von Web-Anwendungen basiert auf dem bereits bekannten Vorgehen zur Erstellung von Desktop-Applikationen. Dieser Weg war bekannt und wurde von den Entwicklern bewältigt. Das Vorgehen sah wie folgt aus:

Der Webserver bekommt die Anfrage (http-Request) vom Benutzer und schickt ihm eine Antwort (http-Response) zurück. Die Antwort ist eine fertige HTML Seite, die alle Informationen zur Darstellung bereits ausgewertet enthält. Der Browser übernimmt die Darstellung des fertigen HTML-Gerüsts. Sobald der Benutzer eine Aktion auf der HTML Seite durchführt, wird diese an den Webserver gesendet. Dieser verarbeitet die Nutzeraktion und sendet ein Resultat in Form einer neuen oder modifizierten (vollständigen) HTML Seite zurück.

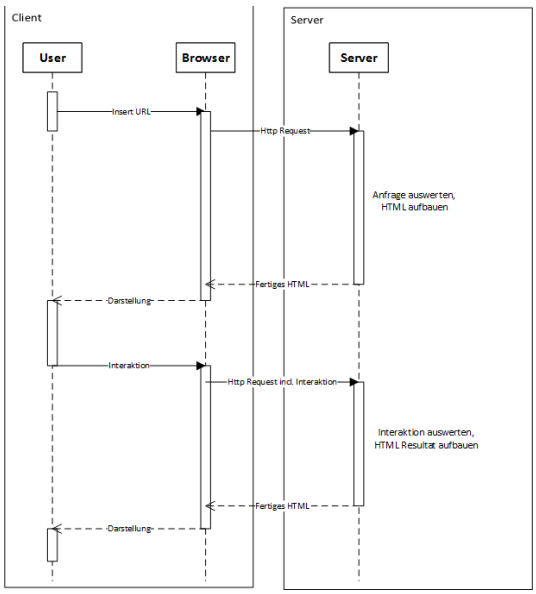

ABB. 1 Prozess HTML Request

Bei diesem Modell weiß der Webserver stets, an welchem Punkt sich der Benutzer befindet und wie die Seite für den Benutzer im Browser aussieht. Dies ist vergleichbar zu einer Desktop Applikation, da es hier keine Trennung zwischen Server und Benutzer gibt und daher der Server jederzeit die Darstellung kennt.

Bei Desktop-Applikationen wurde das MVC-Modell eingeführt, das die Aufgaben einer Applikation in Einheiten unterteilt: *Model, View, Controller* (Leff, & Rayfield, 2001).

Das *Model* beschreibt und enthält die Daten. Im *View* ist die Darstellung definiert, ohne Steuerung von Interaktionen oder Reaktionen. Der Controller kennt *View* und *Model* und stellt die Daten des *Models* so bereit, wie der *View* sie erwartet.

Dieses Konzept hat sich bewährt und ist auch in der Webentwicklung weit verbreitet. Die zusätzliche Herausforderung der Client-Server Infrastruktur wurde zunächst in der Webentwicklung nur als Kanal zur Übermittlung gesehen. Somit wurde der Client als *View-Layer* gesehen. Die Anwendungslogik wird auf dem Server implementiert und ausgeführt.

Durch die Unterstützung von JavaScript im Browser wird jedoch nicht nur die Darstellung, sondern ebenfalls das Nachladen, Verarbeiten und Darstellen von Daten zur Aufgabe auf dem Client.

In dieser Seminararbeit wird beschrieben, wie die Client-Server Infrastruktur in Verbindung mit der stark gestiegenen Verbreitung und Weiterentwicklung von JavaScript zu einem Umdenken in der Webentwicklung geführt hat. Die Rollen von Webserver und Browser werden neu definiert um den veränderten Anforderungen gerecht zu werden.

2.2 Gütekriterien von Webanwendungen

Bei den Anforderungen an Webanwendungen, müssen verschiedene Faktoren betrachtet werden. Die Gütekriterien sind nicht eindeutig definiert, in allen Definitionen finden sich die folgenden Elemente: Verfügbarkeit, Verlässlichkeit und Wartbarkeit (Fielding, 2000). Diese Elemente werden bereits in der herkömmlichen Webentwicklung berücksichtigt. Die hier vorgestellten Ansätze zeigen neue Möglichkeiten zur Gewährleistung der Gütekriterien in einer Form, die mit den bisherigen Ansätzen nahezu unmöglich gewesen wäre.

2.3 Webentwicklung mit JavaScript und schlanken Webservern

Im Gegensatz zu der herkömmlichen Webentwicklung löst sich der neue Ansatz von dem Gedanken, dass der Server weiß, was dem Benutzer im Browser gerade dargestellt wird. Der Server beschränkt sich auf die Auswertung, das Speichern und die Bereitstellung der Daten. Die Anwendungslogik verschiebt sich somit zum Teil zum Client.

2.3.1 Server: Datenverarbeitung und Bereitstellung

Der Server erfüllt im neuen Modell die folgenden Kernaufgaben:

1. Bereitstellen von HTML-Seiten auf Anfrage
2. Bereitstellen, verarbeiten und abspeichern von Daten auf Anfrage

Im Vergleich zum herkömmlichen Modell sind die Bereiche 1 & 2 klar getrennt und enthalten keine Zusammenhänge.

2.3.1.1 Bereitstellen von HTML-Seiten auf Anfrage

Wenn ein Benutzer eine HTML-Seite anfordert (http GET *.html), sendet der Webserver den HTML Code der Seite zurück. Die HTML-Seite wird auf dem Webserver nur minimal interpretiert. Im optimalen Modell sendet der Webserver eine statische Ressource, da dies die schnellstmögliche Antwortzeit bedeutet.

Der HTML-Code dieser Seite enthält alle weiteren Informationen für den Browser, die Darstellungselemente (via Cascading Style Sheets, CSS) und Interaktionsmöglichkeiten (mit JavaScript) beinhalten. Der Webserver wertet diese jedoch nicht aus.

2.3.1.2 Datenhaltung und Verarbeitung

Neben den statischen Ressourcen werden ebenfalls Anfragen nach inhaltlichen Ressourcen aus einer Datenbank benötigt (http GET Request). Hierbei kommt es zu lesenden und schreibenden Anfragen, die über eine Schnittstelle abgerufen werden. Für die Schnittstelle wird zumeist *Representational State Transfer* (REST) (Fielding, 2000) verwendet, da dieses Protokoll auf http basiert und die Implementierung sehr einfach ist. REST wird oftmals verwendet um Standardoperationen für Ressourcen wie Erstellen (CREATE), Lesen (READ), Ändern (UPDATE) und Löschen (DELETE) umzusetzen. Diese Standardoperationen werden zusammenfassend CRUD-Operationen genannt.

Über diese Schnittstelle werden die Daten, beispielsweise Suchergebnisse oder Listeneinträge gesendet. Bei lesenden Anfragen (READ) sendet der Webserver lediglich die Daten in einem Austauschformat zurück, ohne ergänzende Informationen zur Darstellung oder Einbettung in die Seite. Als Austauschformate dienen *Extensible Markup Language* (XML) oder *JavaScript Object Notation* (JSON) Dokumente. READ Anfragen werden mit einem http GET Request ausgeführt. Durch die Übertragung von Inhalten in einem dieser Austauschformate wird garantiert, dass die Antwort schlank (und somit schnell) ist, da keine Informationen zur Darstellung mitgesendet werden.

Im neuen Modell werden somit Datenbankabfragen losgelöst von Anforderungen von HTML-Seiten behandelt.

Bei schreibenden Zugriffen (CREATE / UPDATE / DELETE) werden die Befehle ebenfalls in einer eigenen Anforderung gesendet. Dies erfordert, dass der Browser die Benutzerinterak-

tion zunächst interpretiert und nur die eigentliche Anforderung (Speichern eines Datensatzes) an den Webserver übergibt (siehe Client: Darstellung und Benutzerinteraktion).

Der Webserver verarbeitet diesen Befehl und führt die nötigen Schritte auf der Datenbank aus. Als Antwort bekommt der Client einen Statuscode („OK“ / „FEHLER“). Die Antwort enthält somit ebenfalls keine Informationen zur Darstellung (wie es im herkömmlichen Modell der Fall gewesen wäre).

Bei der Bereitstellung von REST-Services ist es üblich, die Anfragen wie folgt zu behandeln: Das Erstellen eines neuen Datensatzes / einer neuen Ressource (CREATE) erfolgt mittels http POST-Operation. Das Ändern eines Datensatzes erfolgt mittels PUT-Operation (UPDATE). Für das Löschen eines Datensatzes wird die http DELETE-Operation ausgeführt (Richardson & Ruby, 2007).

Ein weiterer Vorteil einer Webapplikation, welche für die Datenhaltung überwiegend mittels eines REST Services kommuniziert, ist die Austauschbarkeit des Clients. So kann der REST Service ebenfalls von einer Desktop-Applikation oder einer Smartphone App angesprochen werden.

Der Server kann zusammenfassend in zwei Funktionsbereiche aufgeteilt werden.

Bereich 1 sorgt für die Bereitstellung von HTML-Seiten. Hierbei werden die Ressourcen möglichst wenig interpretiert, so dass die Geschwindigkeit maximiert und der Rechenaufwand auf dem Server minimiert wird. Dieser Bereich entspricht im Optimalfall einem File-Server, der die Ressourcen unverändert und somit statisch an den Client übergibt.

Bereich 2 beinhaltet die Datenhaltung und Verarbeitung. Hier werden Daten aus der Datenbank gelesen, Datenänderungen vorgenommen und Datensätze gelöscht. Dieser Bereich beschäftigt sich nicht mit der Darstellung oder der Bereitstellung von statischen Ressourcen.

Durch diese Trennung entsteht ein deutlicher Performancegewinn, da die Daten nicht auf dem Server mit der Darstellung zusammengeführt werden müssen.

2.3.2 Client: Darstellung und Benutzerinteraktion

Der Client erfüllt in dem hier dargestellten Modell ein wesentlich größeres Spektrum an Aufgaben als im herkömmlichen Modell. Diese Entwicklung ist erst durch die Verbreitung, Standardisierung und letztendlich die Beschleunigung von JavaScript in Browsern möglich geworden.

Durch diese Entwicklung ist es möglich, Webseiten als eigene Applikation zu betrachten welche ohne einen Webserver existieren kann, nachdem sie erstmalig zum Browser übertragen wurde. Nur wenn Daten auf dem Server gespeichert oder nachgeladen werden sollen muss eine Verbindung zum Server aufgebaut werden.

Im neuen Modell ändert sich die Rollenverteilung grundlegend. Der Client erhält die Vorgaben zur Darstellung (also die HTML-Seite) als statischen Inhalt. Die möglichen Nutzerinter-

aktionen werden per JavaScript verarbeitet. Die vom Webserver bereitgestellten Daten werden im Browser in die Darstellung eingebettet.

Der Browser hat demnach im neuen Modell folgende Aufgaben:

1. Aufbau der HTML-Seite
2. Verarbeiten von Benutzer-Interaktionen
3. Nachladen von Daten

2.3.2.1 Aufbau der HTML Seite

Wenn ein Nutzer eine Anfrage nach einer Seite über seinen Browser an einen Webserver sendet, bekommt der Browser im neuen Modell ein HTML Grundgerüst und JavaScript zurück.

Der Browser stellt zunächst das HTML Grundgerüst dar. Anschließend wird das zugehörige JavaScript ausgewertet. Hierbei werden die Daten in das Grundgerüst mittels Vorlagen (*Templates*) eingebunden und die Einstiegspunkte für Benutzerinteraktionen definiert (*Events*). Sobald diese Schritte durchgeführt wurden, steht die Seite dem Benutzer zur Verfügung.

2.3.2.2 Verarbeiten von Benutzer Interaktionen

Sobald ein definiertes Event vom Benutzer ausgeführt wird, reagiert der zugehörige Event-Handler. Hierbei werden die Parameter des Events (z.B. Eingaben in Textfelder, …) im Browser gelesen und ausgewertet. Im JavaScript ist definiert, ob ausgehend von Event und Parameter eine Kommunikation mit dem Webserver nötig ist. Falls ja, wird eine Anfrage an den Webserver gesendet, die nur die unbedingt notwendigen Inhalte beinhaltet. Abhängig von der Antwort des Servers wird die Darstellung angepasst, beispielsweise um eine Fehlermeldung an den Benutzer zu senden.

In diesem Modell entscheidet also JavaScript im Browser ob eine Anfrage an den Server gesendet werden muss, oder ob die Benutzerinteraktion ohne Kommunikation zum Webserver verarbeitet werden kann.

Für das Einblenden eines Popups ohne Daten ist beispielsweise keine Verbindung zum Webserver nötig. Das Popup kann bereits (für den Benutzer zunächst unsichtbar) im HTML-Grundgerüst enthalten sein.

Sobald das Popup ausgefüllt wurde und der Benutzer die Daten speichern möchte, wird eine Anfrage an den Webserver mit den eingegeben Daten gesendet. In Abhängigkeit von der Antwort des Webservers (Erfolgsmeldung oder Fehlermeldung) wird das Popup ausgeblendet oder nicht.

In diesem Modell ist somit wesentlich weniger Kommunikation mit dem Webserver erforderlich, wodurch die Anwendung für den Benutzer erheblich schneller und flüssiger wird.

2.3.2.3 Nachladen von Daten

Bei Benutzerinteraktionen, die Daten vom Webserver benötigen, wird die Verbindung per *Asynchronous JavaScript and XML* (AJAX) aufgebaut. Sobald die Antwort vom Server vorliegt, werden die Daten in der Darstellung eingebettet.

Die Webanwendung ist somit auch während des Ladevorgangs für den Benutzer sichtbar, da kein Neuladen der Webseite, sondern nur ein partielles Aktualisieren eines Bereichs der Seite notwendig ist.

Beim Nachladen von Daten sendet der Server nur die Daten und keine Informationen zur Darstellung. Im Browser entscheidet das zugehörige JavaScript, wie die Daten angezeigt werden und an welchen Orten die Daten erscheinen sollen.

2.4 Vergleich der Modelle

Im herkömmlichen Modell wird die Applikationslogik vollständig auf dem Server behandelt. Der Browser leitet jegliche Benutzeraktion an den Server weiter. Die Möglichkeiten zur Auswertung der Benutzeraktionen mittels JavaScript werden kaum bzw. meist gar nicht genutzt. Im neuen Modell wird die Applikationslogik zum Großteil auf den Client verschoben. Mittels JavaScript werden die Aktionen des Benutzers interpretiert und behandelt. Wenn Daten gelesen, geschrieben, verändert oder gelöscht werden müssen, wird dies als einzelner Befehl an die REST Schnittstelle des Servers gesendet.

Dies ermöglicht, dass die Serverfunktionen wesentlich „schlanker“ und Anfragen wie Antworten weniger komplex sind. In der Folge können die Clients schneller und effizienter bedient werden. Die Rechenzeit, die im alten Modell für die Darstellung und Bearbeitung von Benutzerinteraktionen (je Benutzer) auf dem Server verwendet werden musste, wird somit auf dem Server gewonnen.

Der Browser muss für die initiale Darstellung der Seite mehr Rechenzeit aufwenden als im traditionellen Modell, ist dafür jedoch weitgehend autonom sobald die Seite erstmalig zur Verfügung steh. Zum Nachladen von Daten wird eine punktuelle und gezielte Verbindung zum Webserver benötigt anstelle eines vollständigen Neuladens der Webseite.

Somit wird ein JavaScript Layer (Ajax Engine) auf dem Client eingeführt, welcher die Kommunikation mit dem Server steuert. Der Vergleich der Modelle sieht wie folgt aus:

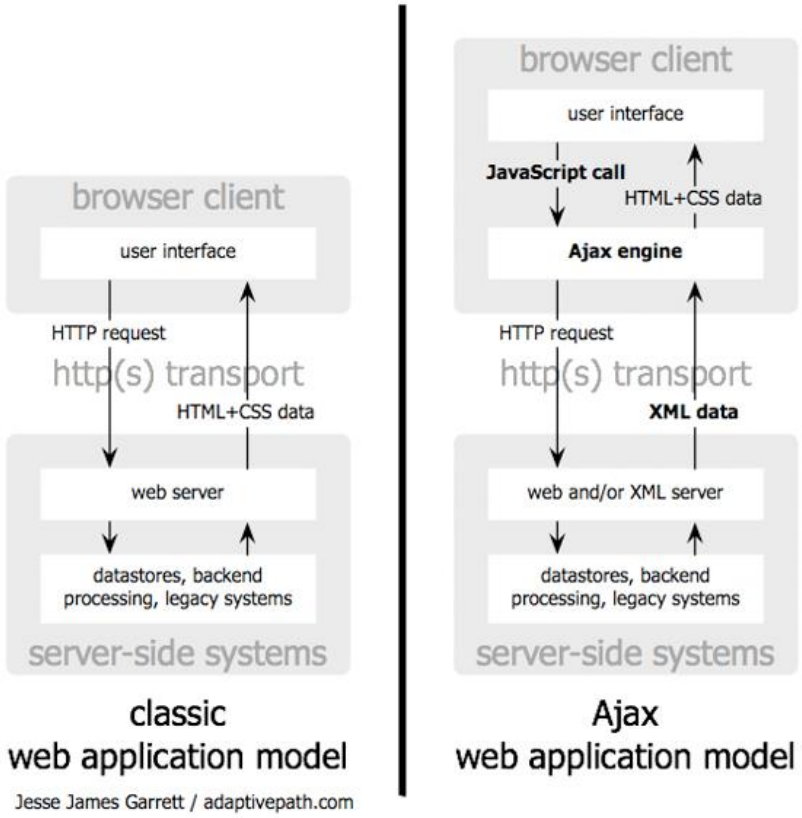

ABB. 2 Vergleich: Webapplikation mit AJAX und ohne AJAX (Garrett, 2005)

Zur Umsetzung des auf AJAX basierenden Modells existieren zahlreiche JavaScript Frameworks, die den Entwickler bei der Erstellung von Webanwendungen unterstützen. In dieser Seminararbeit werden AngularJS und BackboneJS vorgestellt und verglichen.

2.5 JavaScript Frameworks

Infolge der schnellen Verbreitung und ständigen Weiterentwicklung von Web-Anwendungen sind zahlreiche JavaScript Frameworks entstanden. Die hier vorgestellten Frameworks zählen zu den besonders populären JavaScript Frameworks, sie sind bei Entwicklern etabliert und werden in vielen Projekten verwendet.

2.5.1 Backbone JS

Backbone JS verfolgt primär das Ziel, Struktur in JavaScript gestützte Webapplikationen zu bringen. Seit dem 20.03.2013 ist die Version 1.0.0 von Backbone JS veröffentlicht und kann entsprechend der in der Lizenz angegebenen Bedingungen („BackboneJS License“) verwendet werden.

Aufgrund der geringen Größe und Komplexität des Frameworks sind die Anwendungen sehr vielseitig. Der standardmäßige Funktionsumfang ist gering. In Folge dessen bedeutet die Nutzung des Frameworks für einen konkreten Anwendungsfall viel Programmierung. Dies ist beabsichtigt, da das Ziel von BackboneJS die klare Strukturierung ist. Wiederkehrende Aufgaben werden nicht durch Hilfsmittel unterstützt. Inhaltliche Vorgaben könnten Einschränkungen bedeuten. Der Entwickler erhält bei BackboneJS inhaltlich alle Freiheiten, lediglich die Strukturierung der Applikation ist vorgegeben.

Um die Ordnung in Webapplikationen vorzugeben, gibt BackboneJS die folgenden Komponenten zur Strukturierung vor:

- Model
- Collection
- Router
- View

Im Beispiel in Kapitel 3 werden Model, Collection & View implementiert. Ein Router wird in dieser Beispielapplikation nicht benötigt.

2.5.1.1 Model

Ein Model repräsentiert einen Datensatz, welcher CRUD Operationen unterliegt. Ein Datensatz löst ein Event aus, wenn er sich verändert. Über Events kann auf Änderungen reagiert werden.

2.5.1.2 Collection

Durch eine Collection wird eine Menge von Datensätzen beschrieben. Jeder Datensatz entspricht den Model, welches für die Collection definiert wurde.

Collections werden über eine REST-Schnittstelle mit dem Server synchronisiert, d.h. bei Änderungen von Daten innerhalb einer Collection wird die Liste der Änderungen an den Server gesendet. Diese Synchronisation ist eine Standardvorgehensweise, kann aber auch je Collection an die Anforderungen angepasst werden, falls beispielsweise eine Collection nicht auf dem Server gespeichert werden muss.

2.5.1.3 Router

Der Router beschreibt einen Pfad zu einer Ressource. Wird beispielsweise das Profil eines Benutzers angefordert, muss ein Router definiert sein, der auf diese Anfrage reagiert.

2.5.1.4 View

In einem View wird beschrieben, wie ein Datensatz im HTML Gerüst dargestellt werden soll. Neben der Darstellung werden Events deklariert. Mit diesen wird sichergestellt, dass Datenänderungen für den Benutzer sichtbar sind.

2.5.2 AngularJS

AngularJS ist ein Framework, das von Entwicklern von Google erstellt und weiterentwickelt wird. Am 03.04.2013 wurde Version 1.1.4 („quantum-manipulation“) unter der MIT Lizenz („AngularJS License“) veröffentlicht.

AngularJS hebt sich stark von anderen JavaScript Frameworks ab, indem es den HTML Code um eigene Attribute erweitert. Diese Attribute haben semantische Funktion und werden zur Laufzeit im Browser ausgewertet.

Neben den Attributen im HTML Code existieren viele JavaScript Methoden und Komponenten, die den Entwickler bei vielen Anforderungen unterstützen. AngularJS verfolgt hierbei einen deklarativen Ansatz. Der Entwickler definiert nicht den Weg zum Resultat (imperativer Ansatz), sondern beschreibt, bei welcher Situation welches Resultat erwartet wird.

Zur Realisierung dieser deklarativen Beschreibung stellt AngularJS unter anderem folgende Komponenten bereit:

2.5.2.1 View

Bei AngularJS ist der View der HTML Code der Seite. Der HTML Code wird erweitert, um Attribute, die AngularJS interpretiert.

- **ng-model** bindet einen Datensatz an ein HTML Element und seine Unterelemente. Die Werte des Datensatzes können ausgegeben werden.
- **ng-repeat** erzeugt eine Schleife, welche für eine bestimmte Menge durchlaufen wird. Oftmals wird somit eine Liste von Datensätzen durchlaufen und jeder Datensatz wird dargestellt.

2.5.2.2 Controller

Der Controller steuert das Verhalten in der Webanwendung. Bei AngularJS wird der Controller für eine Gruppe von Elementen definiert. Die Elemente in diesem Controller können auf die Methoden und Variablen dieses Controllers zugreifen.

2.6 Beispielapplikation

Um die Vorteile, Unterschiede und Möglichkeiten der Frameworks gegeneinander, aber auch gegenüber herkömmlichen Technologien zur Webentwicklung zu demonstrieren wurde eine Beispielapplikation erstellt.

Als Szenario wurde eine Umfrage gewählt. Der Benutzer kann eine eigene Antwort hinzufügen. Benutzer können mehrere Stimmen abgeben. Die Resultate werden als Balken dargestellt.

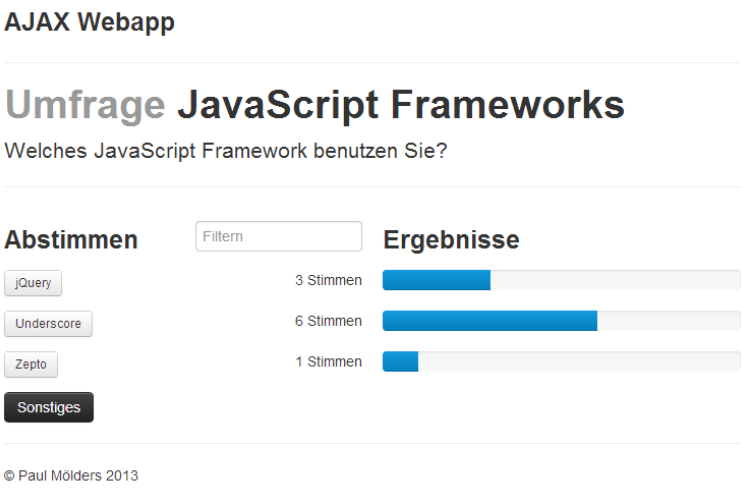

ABB. 3 Beispielapplikation: Ausgangssituation

2.6.1 Ausgangssituation

Die Ausgangssituation enthält ein statisches Layout (siehe ABB. 1) ohne Interaktionsmöglichkeiten. Das Layout wurde mit Twitter Bootstrap (http://twitter.github.io/bootstrap/) erstellt.

2.6.2 Use Cases

Die Anwendungsfälle werden als Use Cases formuliert. Dabei wird aus der Sicht des Benutzers geschildert, welches Ziel er mit der Nutzung der Anwendung verfolgt und wie er zu diesem Ziel gelangt.

Folgende Anwendungsfälle (Use Cases) werden abgebildet:

2.6.2.1 Use Case 1 - Abstimmen

„Als Benutzer möchte ich für eine Option abstimmen, indem ich auf den Namen der Option klicke. Durch meine Stimme erhöht sich die Anzahl der Stimmen dieser Option.“

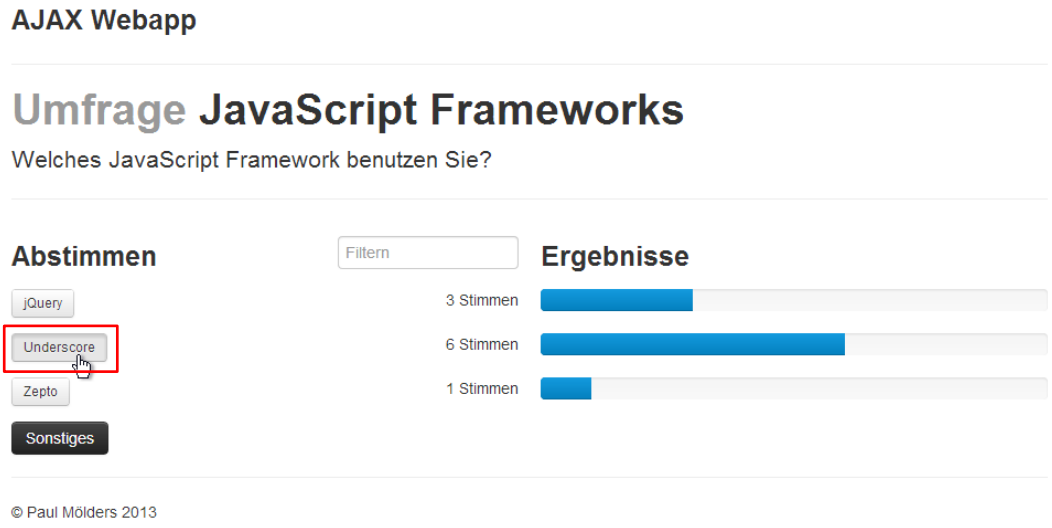

ABB. 4 Abstimmen

Durch mehrmaliges Klicken kann der Benutzer mehrere Stimmen abgeben.

2.6.2.2 Use Case 2 - Option hinzufügen

„Als Benutzer möchte ich eine neue Option hinzufügen, falls meine gewünschte Option noch nicht vorhanden ist. Dazu klicke ich auf „Sonstiges“ und gebe den Namen der Option ein. Durch den Klick auf „Hinzufügen“ wird die Option hinzugefügt. Die hinzugefügte Option hat eine Stimme.“

Nach Klick auf „Sonstiges“ wird ein Textfeld und der „Hinzufügen“ Knopf eingeblendet.

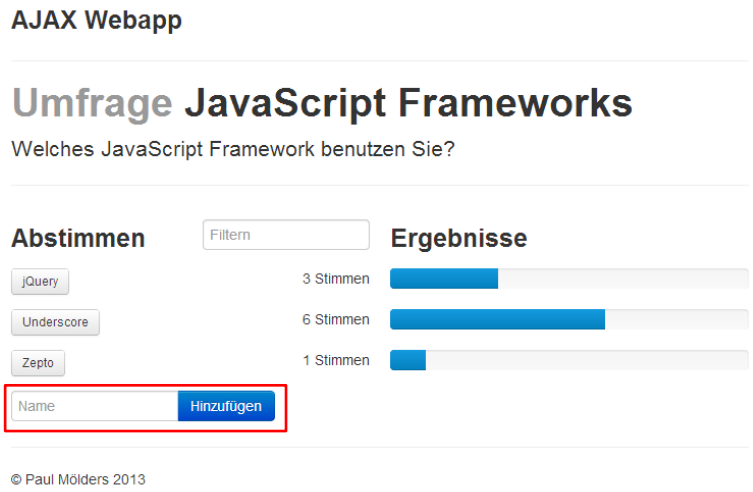

ABB. 5 Option hinzufügen

2.6.2.3 Use Case 3 - Filtern

„Als Benutzer möchte ich die Liste der Optionen durchsuchen können, indem ich den Namen oder einen Teil des Namens der gesuchten Option in das Suchfeld eingebe. Ich möchte jederzeit nur die Optionen sehen, welche zu meinem Suchtext passen."

Der Benutzer sucht die Option, für die er abstimmen möchte. Er gibt dazu einen Teil oder den ganzen Namen in das Filter Feld ein. Sobald mindestens ein Buchstabe eingegeben wurde, enthält die Liste der Optionen nur noch Resultate, die den Eingabestring beinhalten.

ABB. 6 Filtern

2.6.3 Umsetzung ohne JavaScript Framework

In *Use Case 1 – Abstimmen* entscheidet sich der Benutzer für eine Option und möchte seine Stimme für diese abgeben. Mit dem Klick auf die Option wird ein Request ausgelöst, welcher die Aktion des Benutzers an den Server weitergibt. Der Server verarbeitet den Klick und erstellt die Seite neu. Die Seite wird vollständig an den Client übertragen, damit die veränderte Stimmenanzahl sichtbar ist.

Bei *Use Case 2 – Option hinzufügen* möchte der Benutzer eine weitere Option hinzufügen. Dazu sind zwei Requests nötig. Der erste Request wird ausgelöst, wenn der Benutzer auf *Sonstiges* klickt. Der Server bekommt die Anfrage und sendet die gesamte Seite mit veränderter Darstellung (Eingabefeld für die neue Option) zurück. Sobald der Benutzer den Text der neuen Option eingegeben hat und auf *Hinzufügen* klickt wird dieser per Request übermittelt und der Server verarbeitet die Eingabe. Die komplette Seite wird mit der neuen Option an den Client übermittelt.

Bei *Use Case 3 – Filtern* möchte der Benutzer nur noch aktuelle Suchergebnisse sehen. Dafür wird für jeden Tastendruck ein Request benötigt. Sobald der Benutzer einen Buchstaben eingibt, wird der Request gesendet. Der Server interpretiert diesen und sendet die komplette Seite zurück, welche nur noch zum Eingabestring passende Optionen beinhaltet.

2.6.4 Umsetzung mit JavaScript Framework und AJAX

Die Umsetzung der Beispielapplikation mit einem JavaScript Framework und AJAX zeigt, dass die Benutzeraktionen oftmals ohne Kommunikation mit dem Server abgehandelt werden können. Dafür wird im JavaScript Code die Benutzerinteraktion behandelt und die Darstellung aktualisiert. Zur Aktualisierung der Daten auf dem Server werden Requests benötigt, jedoch ist ein Neuladen der vollständigen Seite nicht nötig.

Für *Use Case 1* ist ein Request notwendig, der die Erhöhung der Anzahl der Stimmen für die gewählte Option anweist.

Für *Use Case 2* ist ein Request notwendig, wenn der Name der Option eingegeben wurde. Es ist keine Verbindung zum Server notwendig um die Darstellung zu verändern und das Eingabefeld einzublenden. Der Request stellt lediglich sicher, dass der Server bei anderen Benutzern die Option ebenfalls darstellen kann. Wenn die Daten nicht auf dem Server gespeichert werden sollen, ist gar kein Request notwendig, da die Darstellung und Verarbeitung im Browser erfolgen kann.

Für *Use Case 3* ist kein Request notwendig, da die vorliegenden Daten per JavaScript gefiltert werden können. Die Darstellung wird aktualisiert, ohne dass der Server benötigt wird. Der Server merkt somit gar nicht, dass der Benutzer den Filter benutzt. Dies ist nur möglich, wenn alle Datensätze auf dem Client verfügbar sind.

2.6.5 Umsetzung mit BackboneJS

Die Umsetzung mit BackboneJS teilt die Anforderungen an die Anwendungen gemäß der von BackboneJS vorgegebenen Struktur in verschiedene Bereiche. Diese Bereiche sind miteinander verknüpft, jeder Bereich hat jedoch sein eigenes Aufgabengebiet.

2.6.5.1 Model

Für die beschriebene Beispielapplikation wird ein Backbone Model benötigt, das eine Option der Umfrage repräsentiert. Eine Option hat die Eigenschaft *text*, welche den Namen der Option beinhaltet und die Eigenschaft *votes*, die die Anzahl der Stimmen wiedergibt.

Zusätzlich werden Methoden zum Erhöhen der Stimmen (*vote*), für die Berechnung der Ergebnisdarstellung (*barValue*) und für die Darstellung in der HTML-Seite (*forTemplate*) implementiert.

Der JavaScript-Quellcode sieht wie folgt aus:

```
// Backbone Model
app.Choice = Backbone.Model.extend({
    defaults: {
        'text': '',
        'votes': 1
    },
    // Anzahl der Stimmen erhöhen.
    vote: function () {
        this.set('votes',Number(this.get('votes')) + 1);
    },
    // Berechnung des Balkens
    barValue: function () {
        return (Number(this.get('votes')) / Number(app.Choices.totalVotes()) *100);
    },
    forTemplate: function() {
        var j = this.toJSON();
        j.barValue = this.barValue();
        return j;
    }
});
```

ABB. 7 Backbone Model

Für jede Option der Umfrage wird eine Instanz dieses Models erstellt.

2.6.5.2 Collection

Alle Optionen werden in einer Backbone Collection verwaltet.

```
// Backbone Collection
var ChoiceList = Backbone.Collection.extend({
    // Model für diese Collection festlegen
    model: app.Choice,
    url: "localhost",
    // Gesamtanzahl der Stimmen berechnen
    totalVotes: function () {
        var sum = 0;
        this.each(function (x) {
            sum += Number(x.get('votes'));
        });
        return sum;
    },

    comparator: function (m) {
        return m.get('text').toLowerCase();
    },

    // Filterfunktionalität
    search: function(letters){
        if (letters === "") { return this; }

        return _(this.filter(function(d) {
            var pattern = new RegExp(letters,"gi");
            return pattern.test(d.get("text"));
        }));
    }
});
// Instanz erstellen
app.Choices = new ChoiceList();
```

ABB. 8 Backbone Collection

Die URL gibt das REST Interface an, mit dem die Collection synchronisiert werden soll, falls dies gewünscht ist. Die Methode *totalVotes* summiert die Stimmen aller Optionen. Dieser Wert wird in der Ergebnisdarstellung für die prozentuale Darstellung der Balken verwendet.

Die Filterfunktionalität prüft über einen regulären Ausdruck, ob die eingegebenen Buchstaben in den Optionen enthalten sind.

Die Instanziierung einer Option erfolgt über die Methode *create* Methode der *Backbone.Collection*, welche ein Objekt mit den Eigenschaften als Parameter erwartet. Um eine Option zu erstellen, wird folgender Code ausgeführt:

```
1   app.Choices.create({text: 'jQuery', votes: '3'});
```

ABB. 9 Backbone Model Instanziierung

Mit dieser Zeile wird eine neue Instanz des Models erstellt und der Collection hinzugefügt.

2.6.5.3 Views

Für die Beispielapplikation werden mehrere Views benötigt.

2.6.5.3.1 Choice View

Der *ChoiceView* repräsentiert eine Option der Umfrage und reguliert die Darstellung. Er reagiert auf ein Event: Klick des Benutzers auf die Option um seine Stimme abzugeben (*Use Case 1 – Abstimmen*).

```
1  app.ChoiceView = Backbone.View.extend({
2      tagName: 'div',
3      className: 'vote-row',
4      template: _.template($('#choice-template').html()),
5      events: {
6          'click button.btn': 'addVote'
7      },
8      initialize: function () {
9          this.listenTo(this.model, 'change', this.render);
10     },
11
12     render: function (model) {
13         this.$el.html( this.template( this.model.toJSON() ) );
14         return this;
15     },
16
17     addVote: function () {
18         // Increase vote count
19         this.model.vote();
20     }
21 });
```

ABB. 10 Choice View

In Zeile 6 wird dieses Event an den Button des Templates gebunden. Wenn dieser Button angeklickt wird, wird die *addVote* Methode aufgerufen. Diese verändert nicht direkt die Anzahl der Stimmen, sondern ruft die zugehörige Methode des Models auf.

Mit Zeile 9 wird definiert, dass der View aktualisiert wird, wenn sich das zugehörige Model ändert. Die *render* Methode bindet die Daten des Models an das Template. Das Template ist im HTML-Code definiert:

```
<script type="text/template" id="choice-template">
    <button class="btn btn-small"><%= text %></button>
    <div class="pull-right">   <%= votes %> Stimmen</div>
    <br /><br />
</script>
```

ABB. 11 Choice View Template

In diesem HTML-Code werden beim Aufruf der *Render* Methode die Daten aus dem Model für die Felder (*<%= text %>, <%= votes %>*) eingefügt.

Die Eigenschaften *tagName* und *className* des Views legen fest, welcher HTML-Tag mit welcher CSS-Klasse für den View erstellt werden soll.

Das Resultat der *render* Methode für eine die in ABB. 9 Backbone Model Instanziierung erstellte Option sieht im HTML Resultat wie folgt aus:

```
<div class="vote-row">
    <button class="btn btn-small">jQuery</button>
    <div class="pull-right">   3 Stimmen</div>
    <br><br>
</div>
```

ABB. 12 Choice View HTML

Im Quellcode des Users sind somit nach der Ausführung des JavaScripts keine zusätzlichen Attribute oder sonstige Zeichen vorhanden, die auf BackboneJS schließen lassen.

2.6.5.3.2 Result View

Für die Ergebnisdarstellung wird ein *Result View* erstellt, welcher den Balken darstellt.

```
app.ResultView = Backbone.View.extend({
    tagName: 'div',
    className: 'progress',
    template: _.template($('#result-template').html()),
    initialize: function () {
    },

    render: function (model) {
        this.$el.html( this.template( this.model.forTemplate() ) );
        return this;
    }
});
```

ABB. 13 Result View

Der Result View benötigt nicht den Text und die Anzahl der Stimmen des Models, sondern den relativen Ergebniswert, daher wird in Zeile 9 die *forTemplate* Methode aufgerufen.

Das Template für den Result View sieht wie folgt aus:

```
<script type="text/template" id="result-template">
    <div class="bar" style="width: <%= barValue %>%"></div>
</script>
```

ABB. 14 Result View Template

In der *render* Methode wird die Prozentzahl eingesetzt. Die Berechnung der Prozentzahl erfolgt in dem Model selbst.

```
<div class="progress">
    <div class="bar" style="width: 30%"></div>
</div>
```

ABB. 15 Result View HTML

2.6.5.3.3 AppView

Im AppView werden alle Elemente vereint und in die Seite eingebunden. Implementation des AppViews:

```
// AppView
app.AppView = Backbone.View.extend({

    // View
    el: '#votingApp',

    // Definition der Events
    events: {
        // Use Case 3 - Filtern
        'keyup #filter': 'filterByName',

        // Use Case 2 - Option hinzufügen
        'click #btnToggle': 'btnToggle',
        'keypress #txtAdd': 'submitOnEnter',
        'click #btnAdd': 'btnAdd'
    },

    // Event Binding
    initialize: function () {
        // Use Case 2 - Option hinzufügen: Filter zurücksetzen
        this.listenTo(app.Choices, 'add', this.resetFilter);
        // Use Case 3 - Filtern: Bei jeglichem Event Filter anwenden
        this.listenTo(app.Choices, 'all', this.filterByName);
    },

    // Use Case 2 - Option hinzufügen: Button
    btnToggle: function () {
        $('#btnToggle').hide();
        $('#divToggle').show();
    },

    // Usability: Enter anstatt klick auf den Hinzufügen Knopf zulassen.
    submitOnEnter: function (e) {
        if ( e.which !== 13) {
            return;
        }
        else
        {
            this.btnAdd();
        }
    },

    // Use Case 2 - Option hinzufügen
    btnAdd: function () {
        var val = $('#txtAdd').val().trim();
        if (val !== "") {
            // Neues Model erstellen
            app.Choices.create({text: val});
            // Darstellung anpassen
            $('#btnToggle').show();
            $('#divToggle').hide();
            $('#txtAdd').val('');
        }
    },

    // Use Case 3 - Filtern
    filterByName: function(e){
        // Zurücksetzen der Darstellung
        this.$('#backbone-votes').html('');
        this.$('#backbone-results').html('');

        var letters = $(e.currentTarget || '#filter').val();
        var results = app.Choices.search(letters);
        results.each(this.renderChoice);
        results.each(this.renderResult);
    },

    // Filter zurücksetzen
    resetFilter: function () {
        $('#filter').val('');
    },

    // Eine Option darstellen
    renderChoice: function (c) {
        var vote = new app.ChoiceView({ model: c });
        $('#backbone-votes').append( vote.render().el );
    },

    // Das Resultat zu einer Option darstellen
    renderResult: function (c) {
        var result = new app.ResultView({ model: c });
        $('#backbone-results').append( result.render().el );
    },

    // Alle Resultate zurücksetzen und neu darstellen
    renderResults: function ()
    {
        this.$('#backbone-results').html('');
        app.Choices.each(this.renderResult);
    }
});
```

ABB. 16 App View

Die Events werden im JavaScript Code an die ID des HTML Elements gebunden. Backbone sucht hierbei nur innerhalb des aktuellen Kontexts. Bei einem Event wird angegeben, welche Methode aufgerufen werden soll, sobald das Event ausgelöst wird.

Die *initialize* Methode entspricht einem Konstruktor für den AppView. Hier wird definiert, auf welche Events aus der Collection der Optionen der AppView reagieren muss. Es werden zwei Reaktionen definiert:

- Beim Hinzufügen einer Option wird der Filter zurückgesetzt, damit die neue Option angezeigt wird, auch wenn zuvor gefiltert wurde.
- Jegliche Änderung an der Collection aktualisiert die Darstellung der Optionen und der Ergebnisse. Somit ist sichergestellt, dass die Darstellung stets aktuell ist. Dies ist nötig, da der Ergebnisbalken einer Option von den anderen direkt abhängt.

Hier zeigt sich, dass Backbone als Verbindung zwischen Daten und Präsentationsschicht dient. Im AppView wird auf Events reagiert, die durch Veränderung an den Daten (also der Collection) auftreten. Ebenfalls wird auf Benutzerinteraktionen, wie Texteingaben oder Mausklicks reagiert.

Die Render Methoden (*renderChoice*, *renderResult*) bekommen als Parameter ein Model und führen dieses mit einem View zusammen. Der View bindet die Werte in das Template ein und gibt über die *render* Methode des Views den HTML-Code mit den eingebetteten Daten zurück. Der AppView koordiniert also, welcher View mit welchen Daten an welche Stelle im HTML-Code eingebunden wird.

2.6.5.4 Start der BackboneJS Applikation

Um die oben genannten Komponenten zu aktivieren, wird folgender JavaScript Code in die Seite eingebunden.

```
1  // Backbone App
2  $(function() {
3
4      // AppView instanziieren - ruft initialize Methode auf
5      new app.AppView();
6
7      // Daten
8      app.Choices.create({text: 'jQuery', votes: '3'});
9      app.Choices.create({text: 'Underscore', votes: '6'});
10     app.Choices.create({text: 'Zepto', votes: '1'});
11 });
```

ABB. 17 Backbone Applikation starten

Beim Laden der Seite wird somit eine Instanz des AppView erstellt. Hiermit wird die *initialize* Methode des AppView aufgerufen. Durch die Instanz des AppView sind die Eventhandler an den HTML-Code gebunden und einsatzbereit.

Es werden 3 Optionen angelegt, welche nach dem erfolgreichen Laden der Seite dargestellt werden. Diese Optionen könnten ebenfalls vom Server geladen oder mitgeliefert werden.

Die BackboneJS Implementierung der Beispielapplikation ist hiermit abgeschlossen.

2.6.6 Umsetzung mit AngularJS

Da AngularJS einen deklarativen Ansatz verfolgt, ist die Logik der Applikation auf zwei Stellen verteilt. Im HTML-Gründgerüst wird definiert, wie Daten angezeigt werden sollen und welche Events ausgelöst werden. Im JavaScript wird definiert, wie die Events verarbeitet werden sollen.

2.6.6.1 HTML

Das HTML-Gerüst für die AngularJS Applikation sieht wiefolgt aus:

```
1  <div class="row-fluid" ng-controller="VoteCtrl">
2    <div class="span6">
3      <h3>Abstimmen
4        <div class="pull-right">
5          <!-- Filter -->
6          <input type="text" ng-model="search" placeholder="Filtern" class="input-medium">
7        </div>
8      </h3>
9
10     <!-- Liste der Frameworks -->
11     <div ng-repeat="framework in frameworks | filter:search | orderBy:'text'" class="vote-row">
12       <button class="btn btn-small" ng-click="framework.votes = framework.votes*1.0+1 ">{{framework.text}}</button>
13       <div class="pull-right">   {{framework.votes}} Stimmen</div><br /><br />
14     </div>
15
16     <!-- Hinzufügen Button & Input -->
17     <button ng-hide="newchoice" ng-click="newchoice = true" ng-init="newchoice = false" class="btn btn-inverse">Sonstiges</button>
18     <div ng-show="newchoice" class="input-append">
19       <form ng-submit="addChoice()">
20         <input ng-model="choiceName" type="text" class="input-medium" placeholder="Name"></input>  
21         <input type="submit" class="btn btn-primary" value="Hinzufügen"></input>
22       </form>
23     </div>
24
25   </div>
26
27   <div class="span6">
28     <h3>Ergebnisse</h3>
29     <div ng-repeat="framework in frameworks | filter:search | orderBy:'text'" class="vote-row">
30       <div class="progress">
31         <div class="bar" style="width: {{framework.votes / totalVotes() * 100}}%"></div>
32       </div>
33     </div>
34   </div>
35 </div>
```

ABB. 18 Angular HTML

Die Namen der AngularJS spezifischen HTML-Attribute beginnen mit dem Präfix *ng.* In dieser Applikation werden die Attribute *ng-repeat, ng-click, ng-show, ng-model, ng-hide, ng-submit, ng-controller* und *ng-init* verwendet.

Mit *ng-controller* wird ein Abschnitt begonnen, der einem AngularJS Controller unterliegt. Dieser Controller ist nur innerhalb des Tags gültig, dem er zugewiesen ist. Im Beispiel gilt der Controller also von Z.1 bis Z.39.

Das Attribut *ng-model* deklariert die Datenbindung eines HTML Tags an eine Variable. In Z.6 wird somit der Wert des Textfeldes an die JavaScript Variable *search* gebunden. Sobald sich der Inhalt des Textfeldes ändert, wird der Wert der Variable automatisch von AngularJS aktualisiert.

Mit dem Attribut *ng-repeat* durchläuft AngularJS die Instanzen einer angegebenen Liste und rendert die beinhalteten HTML-Tags für jedes Element der Liste. Diese deklarative Schleife kann durch weitere Anweisungen erweitert werden. So wird im Wert des *ng-repeat* Attributs festgelegt, dass die Liste auf den eingegebenen Suchbegriff eingeschränkt werden soll

(Z.11: *filter: search*) und die Elemente in alphabetischer Reihenfolge ausgegeben werden sollen (Z.11: *orderBy: ‚text'*).

Das Attribut *ng-init* ermöglicht eine Anweisung bei der Evaluation auszuführen. So wird in Zeile 17 definiert, dass der Bereich zum Hinzufügen einer weiteren Option zunächst ausgeblendet ist.

Für Events stehen Attribute für viele Interaktionen bereit. Hierzu zählen die verwendeten Attribute *ng-click* und *ng-submit*. Es existieren weitere Attribute wie *ng-dblclick* für einen Doppelklick.
Mit diesen Attributen wird definiert, wie auf das Event reagiert werden soll. Beim Klick auf den *Hinzufügen* Knopf wird die *addChoice()* Funktion aufgerufen, welche im JavaScript definiert wird (Z.19).

In Zeile 12 ist definiert, dass beim Klick auf eine Option die Anzahl der Stimmen für diese Option erhöht wird. Diese Definition erfolgt somit deklarativ im HTML-Code.

2.6.6.2 Angular JavaScript

Im JavaScript wird ein Controller definiert, der die Daten bereitstellt, eine Methode zum Hinzufügen bereitstellt und die Gesamtanzahl der Stimmen ausgeben kann. Dieser Controller wird im HTML-Code an einen HTML-Tag gebunden und automatisch von Angular instanziiert.

```
1  function VoteCtrl($scope, $http) {
2      $scope.frameworks = [
3          {text: 'jQuery', votes: '3'},
4          {text: 'Zepto', votes: '1'},
5          {text: 'Underscore', votes: '6'}
6      ];
7
8      // Add Choice Function
9      $scope.addChoice = function () {
10         $scope.frameworks.push({text: $scope.choiceName, votes: '1'});
11         // Input zurücksetzen
12         $scope.choiceName = '';
13         // Layout zurücksetzen
14         $scope.newchoice = false;
15         // Filter zurücksetzen
16         $scope.search = '';
17     };
18
19     $scope.totalVotes = function () {
20         var total = 0;
21         angular.forEach($scope.frameworks, function (value, key) {
22             total += 1.0*value.votes;
23         });
24         return total;
25     };
26 }
```

ABB. 19 Angular JavaScript

Hier ist ersichtlich, dass die Verbindung zwischen JavaScript und HTML bei AngularJS sehr eng ist. So werden die im HTML definierten Namen für Attribute (z.B. choiceName, ABB. 18 Z. 20) als JavaScript Variablen bereitgestellt.

Dies wird möglich durch die *$scope* Variable von AngularJS, die im Kontext des aktuellen Controllers alle definierten Attribute bereitstellt und somit die Verbindung zwischen View und Model innerhalb des Controllers darstellt.

Ein im Scope definierter Wert kann somit im View dargestellt werden. Wenn der Wert im View geändert wird (z.B. bei einem Eingabefeld), wird diese Änderung von AngularJS an den Controller weitergegeben, sodass der Scope stets aktuell ist.

Der beschriebene Quellcode reicht aus, um die gewünschte Funktionalität der Beispielapplikation bereitzustellen.

3 Diskussion

Der in dieser Seminararbeit beschriebene Ansatz zur Entwicklung von Web-Applikationen, in denen die Applikationslogik mithilfe von JavaScript abgebildet wird, bietet verschiedene Vorteile gegenüber Webanwendungen, bei denen die Logik auf dem Webserver abgebildet und ausgewertet wird. Die in Kapitel 2 beschriebenen Vorteile zeigen sich bei der Implementierung mit AngularJS und mit BackboneJS.

Im vorgestellten Beispiel ist die Größe der übertragenen Daten sowie der Aufwand für den Server zur Bereitstellung dieser Daten minimal. Für den Benutzer ist die Anwendung stets sichtbar und benutzbar, da kein erneutes Laden der Webseite nötig ist, um die Änderungen anzuzeigen.

Beide betrachteten Frameworks bieten viele Hilfsmechanismen zur Unterstützung von Neuerungen in der Webentwicklung. Dabei unterscheiden sich AngularJS und BackboneJS erheblich.

3.1 Unterschiede zwischen AngularJS und BackboneJS

Die Implementation der Beispielanwendung zeigt die Unterschiede deutlich auf. Diese Unterschiede werden in verschiedenen Kategorien betrachtet.

3.1.1 Trennung von JavaScript und HTML

BackboneJS setzt auf strikte Trennung zwischen JavaScript und HTML Code. Die Verbindung zwischen HTML und JavaScript ist lediglich die Bindung von Events an HTML Elemente, sowie die Einbindung von Daten in HTML-Templates. AngularJS setzt hingegen auf die deklarative Definition von Verhalten im HTML Code. Im JavaScript Code werden Helfer definiert, die die Darstellung unterstützen oder die Verbindung zum Server realisieren.

Für beide Vorgehensweisen gibt es Pro- und Kontra-Argumente. Für die strikte Trennung von JavaScript und HTML bei Backbone spricht die klare Trennung von Code und Darstellung, die die Zusammenarbeit von Designern und Entwicklern vereinfacht. Zudem ist die Wiederverwendbarkeit von Quellcode bei diesem Modell höher, da der JavaScript Code bei verschiedenen Webseiten eingesetzt werden kann. Der Ansatz von AngularJS hingegen erzeugt höhere Verständlichkeit beim Entwickeln und Verändern von Webseiten. Die Logik ist stets sichtbar und nicht in verschiedenen Dateien versteckt ist. Zur Gewährleistung der Wiederverwendbarkeit von Code bei AngularJS lassen sich Module definieren, die per Dependency Injection eingebunden werden können.

3.1.2 Datenbindung

Die Umsetzung der Datenbindung unterscheidet sich sehr stark. AngularJS bietet Two-Way-Databinding, wogegen BackboneJS im Standard lediglich One-Way-Databinding unterstützt.

Two-Way-Databinding bedeutet, dass der Wert einer Variablen sowohl dargestellt, als auch aktualisiert wird, falls sich das gebundene Element verändert. Bei AngularJS ist der Wert einer Variablen an das Textfeld gebunden. Sobald sich das Textfeld ändert, wird die Variable automatisch aktualisiert. Somit kann der Entwickler sicher sein, den aktuellen Wert abrufen zu können, ohne diesen explizit aus der Webseite auslesen zu müssen.

Im Gegensatz dazu wird beim One-Way-Databinding der Wert der Variable in den HTML-Code eingebettet. Falls sich der HTML-Code ändert, wird die Variable nicht automatisch aktualisiert. Bei BackboneJS muss diese Verbindung um den Umweg eines Events und Eventhandlers realisiert werden. Im Eventhandler kann dann die gewünschte Variable aktualisiert werden.

Die verschiedenen Modelle zur Datenbindung bieten Vor- und Nachteile. Das automatisierte Two-Way Modell von AngularJS nimmt dem Entwickler viel Arbeit ab. Die Darstellung von Werten sowie die Aktualisierung von Werten nach Änderungen (durch den User oder durch den Server) ist nicht notwendig, sobald die deklarative Verbindung zwischen Element und Wert hergestellt wurde.

Bei BackboneJS kann der Entwickler im Gegensatz dazu selbständig definieren, wie die Applikation auf Änderungen reagieren soll. So ist es oftmals nicht notwendig, die Werte des Textfeldes nach jedem Tastendruck zu erhalten, sondern lediglich beim Klick auf einen Knopf. Dieses Beispiel zeigt die größte Gefahr bei AngularJS: Durch die beidseitige Bindung kann es zu Performanceproblemen kommen, wenn die Reaktion auf ein simples, häufig hintereinander durchgeführtes Event viel Rechenzeit in Anspruch nimmt, sodass der Browser nicht mit der Verarbeitung nachkommt.

3.1.3 Umfang der Frameworks

Der Kern von BackboneJS ist sehr schlicht. Der Quellcode besteht nahezu nur aus den Basiselementen (Model, Collection, View). Dem Entwickler wird damit ermöglicht, in kurzer Zeit einen umfassenden Überblick über das Framework zu erhalten. Es ist jedoch schnell ersichtlich, dass durch diese schlichte Implementierung viele Basisaufgaben (beispielsweise die Filterfunktion in der Beispielapplikation) eigenständig umgesetzt werden müssen.

Im Gegensatz dazu ist die Bereitstellung von Implementationen für Basisaufgaben einer der Vorteile von AngularJS. Dieses Framework bietet sehr viel Unterstützung, wie das Auflisten von Elementen in einer Menge von Elementen mithilfe von *ng-repeat* oder eben das Einschränken von Werten mit der Filterfunktion. Gerade für Standardaufgaben finden sich viele Hilfsmethoden im AngularJS Framework.

3.1.4 Erweiterbarkeit und Popularität

Google Trends („Google Trends AngularJS, Backbone.JS“) zeigt bei der Eingabe von „AngularJS“ und „Backbone.JS“ den Verlauf von Suchanfragen seit Juli 2010. AngularJS hat BackboneJS im Jahre 2012 nicht nur eingeholt, sondern überholt.

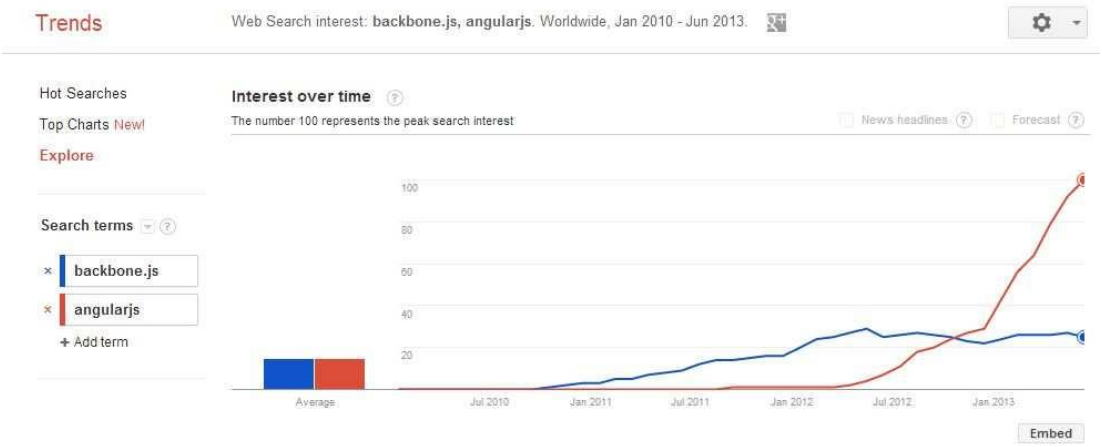

ABB. 20 Google Trends AngularJS Backbone.JS

Der Trend von AngularJS zeigt stark nach oben, wobei BackboneJS seit Mitte 2012 zu stagnieren scheint.

Auf Github wurde BackboneJS von ca. 15.000 Benutzern mit einem Star vermerkt. Hiermit kennzeichnet ein Benutzer, dass er das Projekt interessant findet. AngularJS liegt bei 10.000 Stars auf Github.

Beide Frameworks haben die Version 1.0.0 überschritten (AngularJS: 13.06.2012, BackboneJS: 20.03.2013). Beide Frameworks setzen auf Open Source und werden somit von verschiedenen Programmierern weiterentwickelt. Jeder kann über Github an der Entwicklung teilnehmen.

Die Erweiterbarkeit durch Module liegt sowohl bei AngularJS als auch bei BackboneJS im Fokus. Für BackboneJS gibt es beispielsweise eine Erweiterung zur Umsetzung von um Two-Way-Databinding („Backbone.DataBinding“). Für AngularJS sind mehr als 100 Erweiterungen auf „AngularJS Modules“ verfügbar.

Bei Erweiterungen variiert der Grad der Stabilität und Qualität sehr stark. Viele Erweiterungen sind zudem nur mangelhaft bis gar nicht dokumentiert.

Dennoch gibt es viele (auch kommerzielle) Projekte, in denen die beschriebenen Frameworks eingesetzt werden. Zu den bekannten Beispielen für BackboneJS zählen Foursquare, Bitbucket und Airbnb. Bei AngularJS ist Youtube für die Playstation 3 zu nennen.

3.2 Erkenntnisse aus der Umsetzung

Die Umsetzung der Beispielapplikation verdeutlicht die beschriebenen Vor- und Nachteile von AngularJS und BackboneJS.

Die Umsetzung mit AngularJS war sehr schnell und intuitiv möglich, da Teile der Applikation wie die Suche mit AngularJS sehr einfach umzusetzen sind. Mit dem Two-Way-Databinding sind Events und Auslesen von Eingabefeldern nicht nötig. Der HTML-Code bleibt übersichtlich und die Funktionen von Elementen sind durch die AngularJS Attribute verständlich. Der JavaScript Anteil bei der Beispielapplikation ist sehr gering und leicht verständlich.

Im Gegensatz dazu benötigt die Umsetzung mit BackboneJS erheblich mehr JavaScript-Code. Dadurch bleibt der HTML-Code zwar ebenfalls übersichtlich, allerdings muss der Entwickler wissen oder nachvollziehen, welche Elemente im HTML-Code welche Aufgabe haben, da die Events im JavaScript definiert werden. Hier liegt andererseits aber die große Stärke BackboneJS, da der Entwickler das Verhalten der Applikation exakt definieren kann. BackboneJS hilft dem Entwickler auch bei der Umsetzung, der größte Vorteil bei BackboneJS ist jedoch die klare Struktur des Codes. Durch die vorgegebenen Elementen (*Model*, *Collection*, …) bleibt die Übersichtlichkeit erhalten. Solange der Entwickler sich an die vorgegebene Struktur hält, bleibt der Quellcode verständlich.

Dennoch ist es ratsam bei BackboneJS Erweiterungen einzusetzen, da der Kern für viele Anwendungsfälle stark erweitert werden muss. Viele Erweiterungen setzen auf der klaren Struktur von BackboneJS auf und fügen anwendungsfallspezifische Hilfen hinzu, welche den Entwicklungsaufwand erheblich verringern können.

Entwickler, die vor der Wahl des JavaScript Frameworks für eine Webapplikation stehen, sollten die Vor- und Nachteile der vorgestellten Lösungen vergleichen und auf Basis der für ihr Projekt relevanten Faktoren entscheiden. Eine grundsätzliche Empfehlung kann nicht für eines der beiden Frameworks ausgesprochen werden, wohl aber eine Empfehlung für den generellen Einsatz eines JavaScript Frameworks bei der Entwicklung von Webapplikationen. Mit dem Einsatz eines solchen Frameworks kann Entwicklungsaufwand in erheblichem Umfang eingespart werden. Der Quellcode der Webapplikation bleibt wartbar, erweiterbar und verständlich. Die Trennung von Server und Client bereits in der Entwicklung bietet neben dem Performancegewinn weitere Möglichkeiten bei der Trennung von UI und Backend Teams, sowie die Anpassung einer der beiden Schichten, ohne bestehende Funktionalität oder Darstellungsmöglichkeiten zu verlieren.

Abbildungsverzeichnis

Literaturverzeichnis

Calero, C., Ruiz, J., & Piattini, M. (2005). Classifying web metrics using the web quality model. *Online Information Review*, *29*(3), 229-232.

Fielding, R. T. (2000*). Architectural styles and the design of network-based software architectures* (Doctoral dissertation, University of California).

Garrett, J. J. (2005). Ajax: A New Approach to Web Applications. Retrieved from http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications. Stand: 16.06.2013.

Leff, A., & Rayfield, J. T. (2001). Web-application development using the model/view/controller design pattern. In *Enterprise Distributed Object Computing Conference, 2001. EDOC'01. Proceedings. Fifth IEEE International*(pp. 118-127). IEEE.

Richardson, L., & Ruby, S. (2007). *Web Services mit REST*. O'Reilly Germany.

Runeberg, J. (2013). To-Do with JavaScript MV*: A study into the differences between Backbone. js and AngularJS. (Unpublished Thesis). Arcada University of Applied Science, Helsinki.

Internetquellen

AngularJS License, https://github.com/angular/angular.js/blob/master/LICENSE
Stand: 16.06.2013

AngularJS Modules, http://ngmodules.org/
Stand: 22.06.2013

Backbone.DataBinding, https://github.com/DreamTheater/Backbone.DataBinding
Stand: 22.06.2013

BackboneJS License, https://github.com/documentcloud/backbone/blob/master/LICENSE Stand: 16.06.2013

Google Trends: AngularJS, Backbone.JS,
http://www.google.com/trends/explore?q=AngularJS%2C+Backbone.JS
Stand: 22.06.2013

Anhang

Auf der beigefügten CD-ROM befinden sich folgende Inhalte:

Anhang 1: Dokumente

- Die vorliegende Arbeit in elektronischer Form (PDF-Dokument)
- Die zur Arbeit erstellte Präsentation (PowerPoint-Dokument)

Anhang 2: Beispielapplikation

Die erstellte Beispielapplikation liegt in 3 Fassungen vor:

2.1 Umsetzung ohne Framework und Funktionalität

index.html

2.2 Umsetzung mit BackboneJS

backbone.html
js/app-backbone.js

2.3 Umsetzung mit AngularJS

angular.html
js/app-angular.js

Die weiteren Dateien beinhalten die Frameworks, sowie Hilfsmittel, welche bei der Implementierung benutzt wurden.

Liste aller Dateien im Ordner Beispielapplikation

angular.html
backbone.html
index.html
css\bootstrap-responsive.css
css\bootstrap.css
img\glyphicons-halflings-white.png
img\glyphicons-halflings.png
js\angular.js
js\app-angular.js
js\app-backbone.js
js\backbone.js
js\bootstrap.js
js\jquery-1.8.3.min.js
js\underscore.js

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Hausarbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die anderen Quellen im Wortlaut oder dem Sinn nach entnommen wurden, sind durch Angaben der Herkunft kenntlich gemacht. Dies gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen sowie für Quellen aus dem Internet.

Witten, den 28.06.2013 (Paul Mölders)