

TECHNISCHE UNIVERSITÄT BERLIN

BACHELOR THESIS

ENTERPRISE INFOBOARD - INTELLIGENTE VISUALISIERUNG UND AUSTAUSCH VON HETEROGENEN INFORMATIONEN IM UNTERNEHMEN

Supervisor: Prof. Dr. Dr. h.c. Albayrak
Advisors: Dipl.-Inform. Michael Meder

Written by Tom Nick
15. Juli 2015

Abstract

Das DAI-Labor ist ein Innovationsleiter im Bereich verteilte Suchmaschinen - welche nun nicht mehr reine Forschung sind, sondern wie deren PIA-System von vielen Verwaltungen als Alternative zu privaten Suchmaschinen benutzt wird. Einer der Kernherausforderungen bei Suchmaschinen mit vielen komplett separaten Suchindizes ist es die Ergebnisse dieser zusammen zu führen und für den Endbenutzer zu visualisieren. Diese Arbeit zielt darauf ab eine Visualisierungsform die ähnlich zu einer *social media wall* sein soll für diese Ergebnisse zu untersuchen und zu erstellen. Ein weiteres Themengebiet des DAI-Labors ist die Gamification im Enterprise Bereich, so werden auch dieser Mechanismen aus dem Bereich Gamification untersucht und benutzt um die Benutzung der Applikation zu fördern. Das Ergebnis ist eine voll funktionsfähige Webapplikation mit äußerst ansprechender Darstellung der Suchergebnisse und nahtlos eingebundenen Spielelementen.

Declaration of authorship

I hereby certify that the thesis I am submitting is entirely my own original work except where otherwise indicated. I am aware of the University's regulations concerning plagiarism, including those regulations concerning disciplinary actions that may result from plagiarism. Any use of the works of any other author, in any form, is properly acknowledged at their point of use.¹

Berlin, 15. Juli 2015

Place, date

Signature

¹The template for this declaration of authorship was taken from <https://www.wiwi.hu-berlin.de/international/mems/upload/authorship>.

Inhaltsverzeichnis

1 Einleitung	I
1.1 Motivation	I
1.2 Anforderungen	2
1.3 Relevante Arbeiten	2
1.3.1 Social Media Walls	3
1.3.2 Gamification	3
2 Konzeptuelle Umsetzung	5
2.1 Visualisierung der Suchergebnisse	5
2.1.1 Layout	5
2.1.2 Probleme beim Flow-Verhalten	7
2.1.3 Animationen	8
2.1.4 Kacheln	8
2.1.5 Eindeutige Zuordnung von Farben zu Suchbegriffen	10
2.2 Gamification	11
2.2.1 Aktionen und deren Belohnung	12
2.2.2 Anerkennung der Leistung	13
2.2.3 Abzeichen	14
3 Technische Umsetzung	16
3.1 Wahl der Technologie	16
3.1.1 Ladezeiten	16
3.1.2 Programmiersprache	17
3.1.3 Frameworks und Bibliotheken	19
3.1.4 Tooling	21
3.1.5 Facebook React und die Flux Architektur	21
3.2 Architektur der Applikation	25
3.2.1 Überblick	26
3.2.2 Stores	27
3.2.3 Views	30
3.3 Backend	33
4 Fazit und Ausblick	34
5 Appendix	35

Kapitel I

Einleitung

I.I Motivation

Die gemeinsame Darstellung von Inhalten verschiedener Quellen ist ein interessantes und weit erforschtes Thema. Die grundsätzliche Idee ist dem Benutzer die Benutzung bzw. die Konsumierung von Inhalten angenehmer zu gestalten indem ein einheitliches Interface für Inhalte geschaffen wird. Bevor der *digitalen Revolution* war dies ein immens aufwendiges und kostspieliges Unterfangen, als Beispiel sei die frühen Enzyklopädien genannt¹. Durch die momentan verfügbaren Technologien ist das erstellen einer einheitlichen Datenquelle für verschiedene Inhalte deutlich einfacher geworden, wobei die Schwierigkeit einer guten Zusammenführung dennoch weiterhin besteht und Produkte die dies gut machen sehr erfolgreich sind - wie z.B. die Suchmaschine von Google. Wenn man sich jedoch auf eine Teilmenge der verfügbaren Informationen beschränkt wie z.B. Nachrichten die per RSS² verfügbar sind ist es deutlich einfacher eine angemessene Zusammenführung der Inhalte zu gestalten, prominente Beispiele für eine Anwendung dieser Art ist z.B. die Nachrichten-Applikation *pulse*³ die auf Basis von RSS ein angenehmes und attraktives Interface für die gleichzeitige Konsumierung mehrerer Nachrichtenseiten anbietet, der Erfolg von *pulse* spricht für sich⁴. Diese Arbeit zielt auf einen Spezialfall der Darstellungsformen verschiedener Inhalte ab, die der *Social Media Wall* was eine populäre Form für Inhalte mit kurzen Texten und oder Bildern ist⁵.

Gamification hat in den letzten Jahren stetig an Zuwachs gewonnen da es ein äußerst effektives Mittel ist um Benutzer zu binden, was bei der immer größer werdenden Konkurrenz bei digitalen Produkten immer schwerer wird. Die Benutzung von Gamification im Enterprise-Bereich bzw. in kleinen sozialen Kreises ist jedoch bisher noch nicht allzu populär, die prominentesten Beispiele sind Sport-Apps bei denen Freunde sich miteinander vergleichen können. Eine Integration von Gamification stellt damit in diesem Bereich eine interessante Herausforderung da, dessen Potential bisher noch nicht geklärt ist.

¹Enzyklopädien sind insofern eine einheitliche Aggregierung des Inhalts, das sie versucht haben sämtliches Wissen in einer Quelle bereitzustellen die homogen in Sprache und Darstellung ist.

²Ein weit verbreitetes Dateiformat im Internet <http://de.wikipedia.org/wiki/RSS>

³<https://www.pulse.me/>

⁴ca. 22 Millionen Downloads auf dem Android-System

⁵Natürlich sind auch Inhalte anderer Art möglich, jedoch sind die Elemente solcher Wände in jedem Aspekt daraufhin optimiert.

I.2 Anforderungen

Die Anforderungen der Applikation waren von vornherein nicht vollständig geklärt, viel mehr gab es ein grobes Gerüst was die Anwendung liefern sollte. Die komplette Ausarbeitung ist während der Entwicklung geschehen.

Darstellung von Daten Einer der Hauptanforderungen ist die Darstellung von Datenpunkten (im zukünftigen Verlauf des Textes wird die Darstellung dieser Datenpunkte als *Kacheln* bezeichnet) im Stil einer *Social Media Wall*. Daraus folgen weiterhin die Anforderungen dass es unterschiedliche *Kacheln* gibt - also für verschiedene Daten verschiedene Darstellungen z.B. könnten Suchergebnisse für Kontakte und Webseiten ganz unterschiedlich aufgebaut sein. Die dargestellten Inhalte sind weiterhin dynamisch, während des Betriebes können Kacheln hinzukommen/entfernt werden. Die Kacheln bieten zudem Interaktionsmöglichkeiten die das komplette Layout ändern können.

Bewertungssystem Der Benutzer soll in der Lage sein angezeigte Suchergebnisse zu bewerten und dadurch zukünftige Anfragen für **alle** Benutzer zu verändern.

Benutzung mehrerer Datenquellen Die angezeigten Daten können aus mehreren Datenquellen kommen, d.h. die gleichzeitige Darstellung von Daten von z.B. Facebook mit denen von Twitter. Hier war es wieder besonders wichtig, dass man einfach neue Datenquellen hinzufügen kann. Auch das während des Betriebes vorhandene Quellen mit anderen Parametern (wie einer Suchanfrage) neu hinzugefügt werden können.

Authentifizierungssystem Zur Benutzung der Gamification-Elemente und der Suchanfrage bedarf es einer Authentifizierung. Es muss sichergestellt werden, dass kein Benutzer der nicht authentifiziert ist Zugriff auf die Applikation hat.

Single Page Application Es soll eine sogenannte *Single Page Application* werden, d.h. eine Webapplikation bei der JavaScript abseits des initialen Seitenaufrufes alles rendert, im Gegensatz zur klassischen Methodik bei der jede Interaktion zu einer komplett neuen Seite, die vom Server gerendert wird, resultiert.

Gamification Um die Benutzung der Seite zu animieren sollen Gamification-Methoden eingeführt werden wie z.B. Punkte für das tägliche einloggen bis hinzu erweiterten Methoden wie eines Compulsion Loop.

Konfiguration Es muss möglich sein viele Parameter anzupassen, einige davon auch als Benutzerinterface für den Benutzer andere wiederum als simple Konfigurationsdatei für verwendete URLs und ähnliches.

I.3 Relevante Arbeiten

Da die Applikation mehrere Gebiete vereint als etwas bestehendes weiterzuentwickeln, gibt es direkt keine relevanten Arbeiten. Jedoch ist es nützlich bestehendes in den einzelnen Gebieten zu untersuchen.

I.3.1 Social Media Walls

Eine sehr gute Auflistung verschiedener *Social Media Wall*-Applikationen bietet (Schuster, 2014). Die Homogenität dieser Angebote ist erstaunlich, würde man die Produkte zweier verschiedener Anbieter nebeneinander sehen wären die größten Unterschiede stilistische Unterschiede wie benutzte Farben oder ähnliches. Leider sind die wenigsten Produkte in einer Demo zu testen, da die Lösungen meistens zugeschnitten werden für den Kunden, welcher diese meistens auf großen Veranstaltungen wie Konzerten verwendet. In I.1 kann eine der Implementierungen gesehen werden.

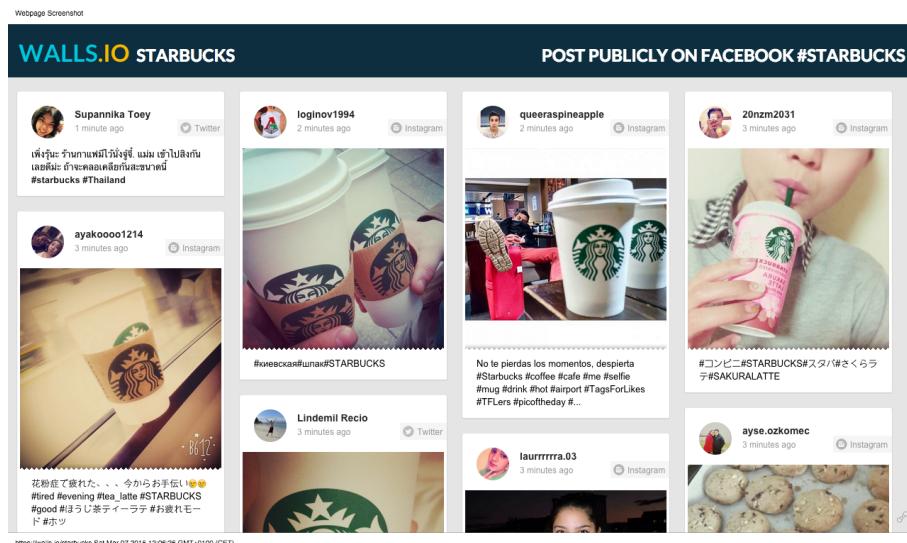


Abbildung I.1: Eine *Social Media Wall* vom Anbieter walls.io

Insofern sind diese Angebote für diese Arbeit nicht wirklich lohnenswert weiter zu analysieren und werden als grobe Inspiration für das Design benutzt.

I.3.2 Gamification

Es gibt einige Seiten die ähnliche Inhalte anzeigen und erfolgreich Gamification implementiert haben. Einen sehr schwachen aber effektiven Ansatz verfolgt die Social-News-Seite *reddit*⁶. Jegliche Inhalte (eingereichte Links, Kommentare) der Seite können binär bewertet werden (im reddit-jargon wird hier vom *upvote* und *downvote*) gesprochen. Reicht man nun selbst etwas ein, wird die Differenz aus *upvotes* und *downvotes* einem als *Karma* gutgeschrieben. Jedoch besitzt Karma ähnlich wie die in Videospielen erreichte Punktzahl keinen Wert in dem Sinne, dass mit ihr nichts weiter gemacht werden kann als den Wert mit den von anderen zu vergleichen. Dieser Gamification-Aspekt ist ein nicht zu unterschätzender Teil von *reddit* bzw. dem Erfolg von *reddit* welcher beachtlich ist⁷. Eine ausführliche Untersuchung von (**richterichkarma**) zeigt die Tragweite von *reddits* Karma-System. Es dient demnach vorzüglich als Bewertung des eingereichten Inhalts durch andere bzw. die daraus resultierende Selbstbestätigung wenn das eigene Karma erhöht wird. Benutzer *reddits* werden dadurch jedoch darauf gepolt ihre eingereichten Beiträge nach der Anzahl an Karma, welches der Beitrag liefern wird, auszusuchen.

⁶<http://reddit.com>

⁷laut Alexa ist *reddit* die 29 meist besuchte Webseite der Welt, in den USA ist sie sogar Platz 10.

DAIKnow (Meder, Plumbaum und Hopfgartner, 2014) ist eine Bookmarking Seite ähnlich zu *delicious.com* bei der Links mit Beschreibungen und Keywords eingereicht werden können. Durch den gezielten Einsatz von Punkten, Abzeichen und von Bestenlisten wird die Benutzung der Seite durch die Benutzer gesteigert. Im Gegensatz zu *reddit* ist dieses System jedoch weitaus komplexer, so bekommt man z.B. Punkte für den täglichen Aufruf, Punkte für das Einrechen eines Links oder das jemand anderes seinen Link kopiert hat. Ein Problem das sich auftat bei DAIKNow war dass häufige Benutzer alle verfügbaren Abzeichen freischalteten und damit der Gamification-Aspekt damit kein Grund mehr ist die Applikation weiter zu benutzen.

Kapitel 2

Konzeptuelle Umsetzung

2.1 Visualisierung der Suchergebnisse

Der wichtigste Aspekt für den Benutzer ist eine attraktive Darstellung und angenehme Bedienung der Applikation, natürlich neben dem Aspekt das es technisch an nichts mangelt d.h. es gibt keine Programmfehler und die nötigen Funktionen sind vorhanden. Im folgenden werden die einzelnen Aspekte der Darstellung dargestellt um die Findung der endgültigen Darstellung zu erklären.

2.1.1 Layout

Wie *Social Media Walls* oder Webseiten wie *pinterest*¹ oder gar Microsoft mit ihrem metro-Design es gezeigt haben ist aktuell eine der besten Darstellung für Medieninhalte verschiedener Art ein Layout basierend auf Kacheln. Es gibt oft eine Regel nach welchem Muster die Kacheln angeordnet werden sollen, z.B. sollten die neuesten Kacheln auch an oberster Stelle gezeigt werden oder auch *wichtigere* Kacheln prominenter dargestellt werden. Die Kacheln für das Infoboard unterliegen der Regel das sie anhand ihrer Wertung angeordnet werden müssen weswegen die folgende Analyse viel Wert darauf legt wie sich das Layout verhält wenn neue Kacheln hinzukommen und damit neu sortiert werden muss. Im folgenden werden zwei der bekanntesten Arten zur Darstellung näher untersucht.

- **Spaltenbasierte Anordnung**

Bei dieser Form werden die einzelnen Datenpunkte in Form von Kacheln mit einheitlicher Breite und variabler Höhe in Spalten in der Größenordnung von 3-5 dargestellt.

¹<http://pinterest.com>

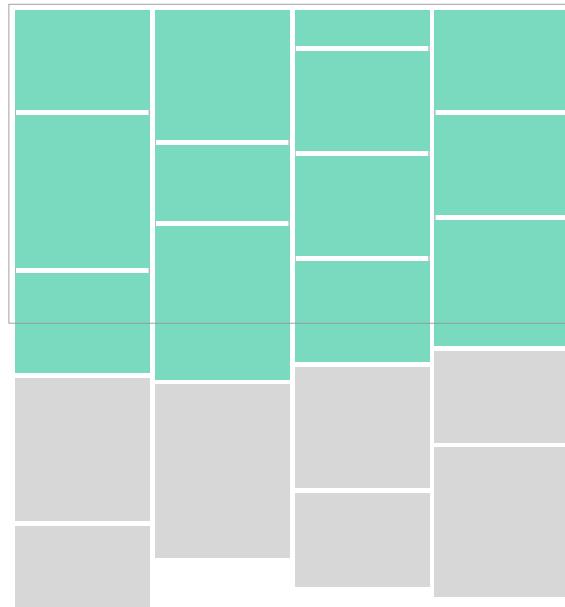


Abbildung 2.1: Spaltenbasierte Anordnung der Inhalte

Vorteile

Die Vorteile liegen in der Simplizität der Implementierung und der Intuitivität des Inhalt-Flusses d.h. neuer Inhalt wird oben eingefügt und alter rutscht nach unten, wobei jeweils nur eine Spalte *verrutscht* je neuem Datenpunkt. Weiterhin ermöglicht die Variable Höhe viel Flexibilität bei dem Anzeigen des Inhalts - z.B. könnten lange Texte angemessen gut angezeigt werden ohne die Zeichenanzahl zu limitieren oder ähnliches.

Nachteile

Der größte Nachteil ist die Starrheit des Layouts, da alle Objekte die gleiche Breite haben ist man stark limitiert wie man die Inhalte darstellt. Abstriche müssen auch gemacht werden bei der Sortierung der Inhalte: Sobald die angezeigten Inhalte sortiert werden müssen kann der Inhaltsfluss nicht optimal funktionieren (das der Fluss nur nach unten geht), man kann ihn nur behalten indem man die Sortierung nur Spaltenweise macht indem man die Kacheln mittels modulo oder anderem auf eine bestimmte Spalte festsetzt. Dies kann aber zu verwirrenden Ergebnissen führen, auch wenn es ästhetisch die beste Variante ist.

- **Rasterbasierte Anordnung** Bei dieser Form werden die einzelnen Datenpunkte in einem einheitlichen Raster dargestellt, d.h. die darstellende Fläche wird in einem Raster der Größe (a, b) unterteilt, die Kacheln können nun die Größe (x, y) mit $x \in \{1, \dots, a\}, y \in \{1, \dots, b\}$ besitzen.

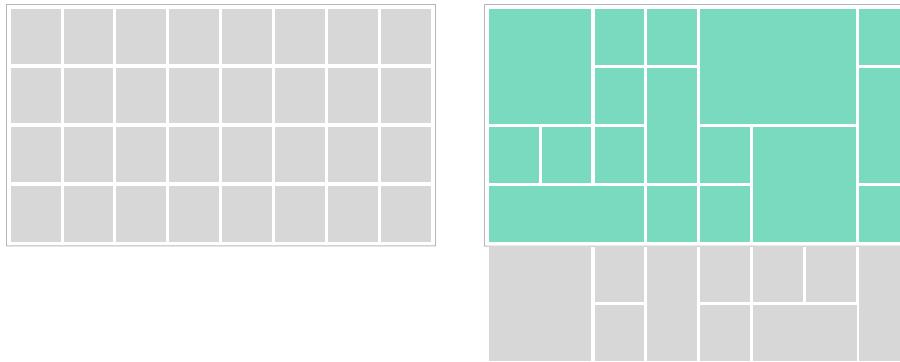


Abbildung 2.2: links: Das Raster des Grids, rechts: eine zufällige Benutzung des Grids

Vorteile

- Ansprechendes Aussehen
- Wichtigkeit kann durch die Größe der Kachel dargestellt werden
- Unterschiede in der Darstellung gleicher Datenpunkte durch unterschiedliche Größe
- Vorteilhaft bei der Anzeige ohne Interaktion, da es keine abgeschnittenen Inhalte gibt wie bei dem Spaltendesign

Nachteile

- komplexes flow-Verhalten bei neuem Inhalt
- Es müssen komplexere Methoden benutzt werden um Löcher zu verhindern
- Es müssen Designs für die verschiedenen Kachelgrößen erstellt werden
- unterschiedliche Darstellungsformen können unübersichtlich wirken

2.1.2 Probleme beim Flow-Verhalten

Das Flow-Verhalten beim einfügen neuer Inhalte ist bei einem grid-basierten Layout mit unterschiedlichen Kachelgrößen äußerst komplex, im ersten Schritt müsste das Behälterproblem gelöst werden und im zweiten Schritt müssten die Elemente anhand ihrer Bewertung nochmals sortiert werden. Die Lösungen des Behälterproblem darauf hin zu optimieren das der optische Fluss der Elemente minimiert wird scheint mir eine nicht triviale Aufgabe zu sein, weshalb man sich mit den *schlechten* Lösungen zufrieden geben muss. Die JavaScript-Bibliothek *packery* implementiert den ersten Schritt und ordnet die Elemente anhand einer Lösung des Behälterproblems, bei deren *prepend*-Methode kann sehr gut erahnt werden wie *wild* der Fluss bei jedem neuen Element wäre².

²<http://packery.metafizzy.co/methods.html#prepended>

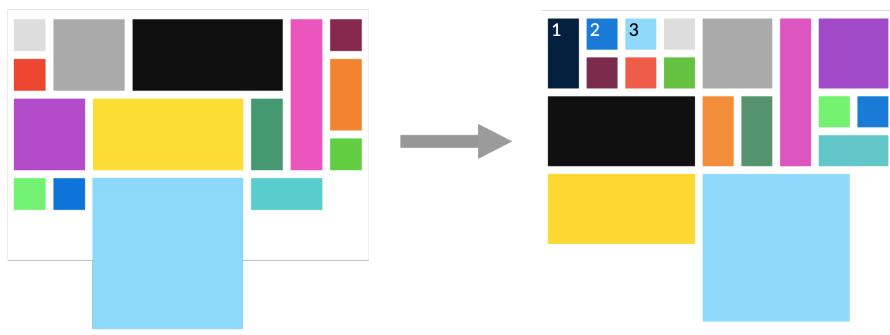


Abbildung 2.3: Zu dem Grid links wurden die Elemente 1, 2 und 3 hinzugefügt. Das resultierende neue Layout ist rechts zu sehen. Dies ist ein praktisches Beispiel welches mithilfe von packery gemacht wurde.

2.1.3 Animationen

Eines meiner Ziele waren Animationen die neben dem rein ästhetischen Aspekt einen Vorteil im Verständnis des Datenflusses der Applikation ermöglicht.

1. Neue Kacheln sollen von *oben* kommen um einerseits zu zeigen dass es neue Inhalte gibt und andererseits um das vertraute Konzept zu benutzen dass neue Dinge meistens von oben kommen³ und nicht überraschend irgendwo auftauchen.⁴
2. Die neuen Kacheln ordnen sich an ihrem Platz ein (ein Algorithmus benutzt verschiedene Werte um das zu bestimmen), alte Kacheln machen dementsprechend Platz.
3. Die Anzahl an Bewegungen soll minimal sein um nicht unnötig vom eigentlichen Inhalt abzulenken.

Nur das Spaltenbasierte Layout erfüllt diese Anforderungen für Animationen, da die Neuberechnung des Layouts bei dem Kachelbasierten zu aufwändig ist, als dass nachverfolgt werden kann wie die Kacheln sich bewegen und welchen Gesetzmäßigkeiten sie unterliegen.

2.1.4 Kacheln

Das Design der Kacheln ist neben derer Anordnung die wichtigste Design-Entscheidung. Neben dem rein funktionalen Aspekt ist auch das Aussehen wichtig um z.B. die einzelnen Inhalte gut voneinander zu unterscheiden und den Benutzer nicht zu überfordern.

Rein funktional besitzt eine Kachel folgende Anforderung:

- Anzeige des gefundenen Inhalts und Möglichkeit diesen zu öffnen. Die Anzeige des Inhalts kann variieren.

³Siehe z.B. Börsenticker, Nachrichtenseiten oder die Abfahrttafeln am Bahnhöfen

⁴Vergleich zu Googles Material Design bei dem viel Wert darauf gelegt wird sinnvolle Animationen zu machen, die den Benutzer unterstützen und nicht verwirren

- Benutzer können den Inhalt favorisieren (speichern) und bewerten.
- Es werden Metainformationen wie der Typ des Inhalts (PDF, Kontakt, ...) angezeigt.
- Kacheln können schnell zu dem Suchbegriff zugeordnet werden von dem Sie stammen.
- Anzeige der letzten Interaktion mit der Komponente um den sozialen Aspekt der Anwendung zu verdeutlichen.

Obwohl die Kacheln je nach dargestellten Inhalt stark variieren können ist es ratsam ein einheitliches Benutzerinterface für die Benutzerinteraktionen zu bieten. Auch sollten die Metainformationen gleich angezeigt werden. Mit diesen Anforderungen kann eine funktionale Kachel konzipiert wie unten gezeigt werden.

	<p>Bewertung Bedienelement zum bewerten des Dokuments. Die Zahl zeigt die derzeitige globale Bewertung an.</p>
<p>Header Bereich für Interaktionen mit dem Element.</p>	<p>51</p> <p></p>
<p>Content Darstellung des Inhaltes des Dokumentes mithilfe seines Titels sowie einem Bildelement.</p>	<p></p> <p>The title of the document. Some titles are quite long, or even longer.</p>
<p>Footer Anzeige von Information wie des Datums und der Domain des Dokumentes, allgemein Zusatzinformationen die nicht allzu relevant sind</p>	<p>20.12.2014 domain.com</p>

Abbildung 2.4: Eine prototyp-Kachel mit rein funktionalen Aspekten.

Menschen sind äußerst gut darin Farben und unterschiedliche Formen schnell zu gruppieren, da nach aktuellem Kenntnisstand, dies Teile der ersten Verarbeitungsstufe von visuellen Informationen ist (Treisman, 1987). Da für uns die schnelle Assoziation von Suchbegriff und Kachel äußerst wichtig ist, wurde die Hintergrundfarbe der Kachel dafür benutzt. Dies schränkt das Design aber insofern ein dass bis auf Grautöne und Abstufungen der verwendeten Hintergrundfarbe keine anderen Farben möglich sind (man könnte z.B. das Symbol zum Favorisieren nicht überall rot anzeigen), doch waren das dadurch erzielte Design so überzeugend, dass dies in Kauf genommen wurde.

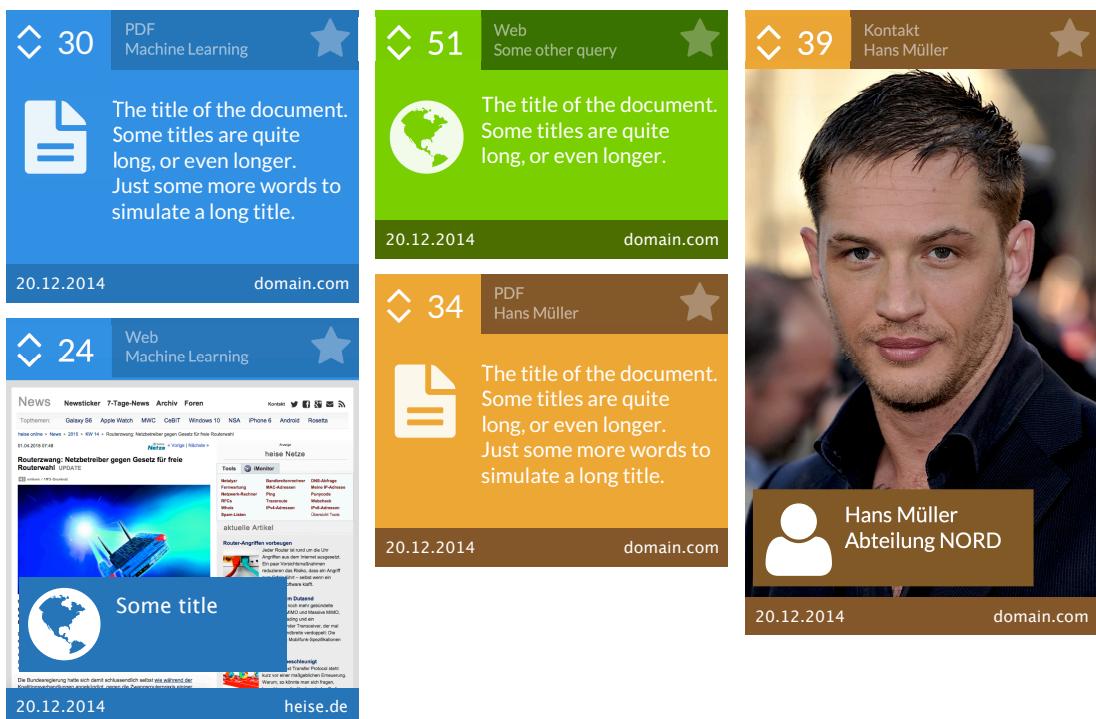


Abbildung 2.5: Ein erstes Design dass verschiedene Darstellungsformen für unterschiedliche Typen von Informationen benutzt.

2.1.5 Eindeutige Zuordnung von Farben zu Suchbegriffen

Da zuvor entschieden wurde die Hintergrundfarbe der Kacheln als Hauptunterscheidungsmerkmal der Suchbegriffe zu verwenden, bleibt zu klären wie eine Farbe zu einem Suchbegriff ausgewählt werden soll. Alle angezeigten Farben sollten optisch zueinander gehören und miteinander harmonieren, um dies zu erreichen gibt es verschiedene Ansätze, bei allen ist der *erste* Schritt die Konvertierung der Suchbegriffe zu einer Zahl. Diese Funktion sollte die typischen Eigenschaften einer guten Hash-Funktion mitbringen: gleichmässige Verteilung der Funktionswerte auf den Zielraum und Linkstotalität⁵. Die hier verwendete Funktion stammt von Dan Bernstein und ist bekannt unter der Bezeichnung *db2*.

Der erste Ansatz wäre es eine Liste mit zueinander harmonierenden Farben zu verwenden und jeweils eins auszusuchen. Dieser Ansatz kann jedoch nur mit einer sehr langen Liste von Farben funktionieren, auch passiert es bei disere Methode öfter das Suchbegriffe dieselbe Farbe zugewiesen bekommen, was äußerst unerwünscht ist. Man könnte sich nun überlegen ob komplexe Konfliktresolution betrieben werden sollte - dass wenn Suchbegriffe aufeinander kollidieren eine davon eine andere Farbe bekommt. Dadurch wird aber verhindert das jeder Suchbegriff Konsistent die gleiche Farbe bekommt.

Der zweite Ansatz besteht darin die Farben prozedural zu erstellen, der naivste Ansatz wäre es hierbei den Wertebereich der Hash-Funktion auf $0 - 2^{16}$) einzuschränken, das Ergebnis in Hexadezimal umzurechnen und als RGB-Farbe zu benutzen. Die dadurch entstehenden Farben sind jedoch alles andere als harmonierend oder könnten als *schön*

⁵Für jeden Suchbegriff gibt es ein Ergebnis der Funktion.

bezeichnet werden, da zu viele dunkle und zu viele helle Farben erstellt werden. Der RGB-Farbraum ist jedoch nicht dazu geeignet die Helligkeit der entstehenden Farben zu kontrollieren, dafür sind Wahrnehmungsorientierte Modelle die Farben durch Helligkeit, Sättigung und Farbton beschreiben deutlich geeigneter wie z.B. HSV oder HSL. Probleme von den HSx Farbräumen sind die für den Menschen ungünstige Verteilung der Farben, da wir z.B. besser im Unterscheiden von Blautönen sind als welche mit Grün/Rot-Anteilen, auch variiert z.B. die Helligkeit der Farben bei Änderung des Farbtone zu sehr. Abhilfe schaffen hier die dafür erstellen Farbräume LAB bzw. die Weiterentwicklung HCL. Diese sind jedoch nicht einfach zu benutzen, da der valide Wertebereich der Eingabeparameter je nach Kombination der Werte unterschiedlich ist. Dies macht es schwer mit diesen Farbräumen Farben zu erstellen. Es gibt im speziellen um den Chroma Wert, dieser kann grob mit der Helligkeit verglichen werden. Der Wertebereich der Sättigung und des Farbtone sind davon abhängig.

Die Wahl des idealen Farbraums ist damit nicht wirklich möglich, mittels HSL ist es einfach eine gute Farbskala prozedural zu erstellen, der Farbbereich ist jedoch ungünstig für den Menschen. LAB/HCL bieten die beste Farbverteilung für die menschliche Wahrnehmung, sind jedoch äußerst schwer zu benutzen. Eine mögliche Kombination der Vorteile beider Farbräume zeigt (Boronine, 2015), hier wird ein Farbraum mit der Bezeichnung *human-friendly HSL* vorgestellt.

Doch keiner der Farbräume konnte eine *ideale* Farbskala erstellen, es konnten in allen Farbräumen sehr anspruchsvolle Farbskalen mit wenig Kollision erstellt werden. Da Farbskalen ein subjektives Thema sind wurde die Möglichkeit eingebaut diese in der Applikation zu verändern - dies wurde gleich mit dem Gamification-Aspekt verbunden wodurch man nach und nach neue Freischalten kann und sich die aussuchen kann die einem am besten gefällt.

Zum testen der Farbskalen wurde eine Liste an möglichen Suchbegriffen benutzt, die Ergebnisse sind in ?? zu sehen.

TODO PICTURE

2.2 Gamification

Wie in der Einleitung erwähnt ist Gamification eine mittlerweile weitverbreitete und äußerst effektive Technik um das sogenannte *Engagement* von Benutzer zu erhöhen, allein durch Gamification konnte z.B. die Seite DevHub ihre *Engagement*-Rate um 20% steigern (Zichermann und Cunningham, 2011). Zahlreiche StartUps feiern große Erfolge mit dem hinzufügen von Gamification zu anstrengenden bzw. langweiligen Aufgaben, wie z.B. das Lernen einer Sprache (Duolingo⁶) oder Programmieren Lernen (Codecademy⁷).

Doch was ist Gamification eigentlich? Der Begriff entstand erst 2008 und hat seitdem viele Bedeutungen erlangt, (Deterding u. a., 2011) bemühten sich eine einheitliche Definition zu finden, die auch im folgenden Kontext verwendet wird.

“Gamification” is the use of game design elements in non-game contexts.

⁶<https://de.duolingo.com/>

⁷<http://www.codecademy.com/>

Das Infoboard dient als Interface für Unternehmensinterne Suchmaschinen, welches zu besserem Wissensaustausch führen soll. Dies ist offensichtlich ein nicht-Spieler-Kontext der durch erhöhtes *Engagement* positive Effekte für ein Unternehmen erzielen kann. Mit dem Einsatz von Gamification-Elementen werden von uns zwei Ziele verfolgt, einerseits soll durch die Verspielisierung der Applikationsmechaniken deren Benutzung erhöht werden bzw. soll ein konstanteres *Engagement* mit der Applikation vom Benutzer erzielt werden, andererseits sollen die Mechaniken zur Bewertung der Suchergebnisse gefördert werden wodurch diese noch besser werden.

Die Grundlage jedes Spieldesigns ist der *Compulsion Loop* welcher eine modellierte Kette an Aktivitäten darstellt, die gewohnheitsmäßig wiederholt werden um eine neurochemische Belohnung zu erhalten.

Nach (Kim, 2014) besteht diese modellierte Kette aus 3 Schritten:

1. **Anerkennung der Leistung** Die Ausführung der Aktionen wird belohnt, der Benutzer bekommt etwas was er gerne möchte. Die Herausforderung hierbei ist es etwas zu erschaffen, dass der Benutzer möchte und die Belohnung so einzuteilen das er gerade soviel bekommt dass es eine Belohnung darstellt. Am Beispiel des Rollenspiels wäre eine Belohnung z.B. die verbesserte Rüstung mit der gespielte Charakter widerstandsfähiger wird und neue bisher verschlossene Abenteuer bestreiten kann.
2. **Belohnung einer Aktion** Jede Aktion die der Benutzer ausführt hat eine direkte Belohnung zur Folge, diese ist jedoch so klein, dass sie direkt keine Glücksgefühle auslöst sondern erst eine Anhäufung der Belohnungen führt dazu dass der Benutzer die eigentliche **Belohnung der Leistung** erhält. Es ist zu beachten dass diese direkten Belohnungen nicht allzu schnell die Möglichkeit freischalten sie gegen dass einzutauschen wonach der Benutzer strebt. Dies könnte im Rollenspiel das Gold sein, dass benötigt wird um eine neue Rüstung zu kaufen.
3. **Aktion** Die Aktion ist die Grundlage, diese sind die Aspekte der Anwendung die der Benutzer ausführen soll. Im Rollenspiel könnte dies z.B. das lösen von Aufträgen sein.

Im folgenden wird für die Gamification des Infoboard anhand dieser Schritte modelliert.

2.2.1 Aktionen und deren Belohnung

Das Enterprise Infoboard besitzt wenige Aktionen, sodass alle Teil der Aktionen sind die eine Belohnung geben. Die Gewichtung ist doch auf die Bewertung der Ergebnisse sortiert, da wir darin den größten Wert sehen.

- Bewertung der Suchergebnisse
- Favorisieren von Suchergebnissen
- Eingabe und Löschung von Suchbegriffen
- Anmeldung und Abmeldung

Durch einige dieser Aktionen wird der User Punkte bekommen, z.B. ist es häufig üblich für das tägliche Besuchen der Seite dem Benutzer Punkte gutzuschreiben. Wichtig für uns ist es, dass die Bewertung der Suchergebnisse besonders gefördert wird, womit es denkbar ist hier eine größere Punktzahl im Vergleich zu den anderen Aktionen vergeben wird.

2.2.2 Anerkennung der Leistung

Die Leistung des Benutzers muss anerkannt werden, im folgenden werden verschiedene Ansätze für Punkte/Bestenlisten und Abzeichen gezeigt und erklärt warum wir uns für den einen oder anderen Ansatz entschieden haben.

2.2.2.1 Punkte

Punkte sind unerlässlich im Spielekontext, auch wenn die Punkte für den Benutzer nicht mal sichtbar gemacht werden sind sie wichtig um dem Spieldesigner die Möglichkeit zu geben das System zu evaluieren und daraufhin zu verändern ((Zichermann und Cunningham, 2011)). Punktesysteme können dementsprechend von nahezu unsichtbar zu der Hauptkomponente des Systems gemacht werden. Nach (Zichermann und Cunningham, 2011) unterscheidet man erhaltene Punkte in folgende Kategorien.

- Erfahrungspunkte
- Eintauschbare Punkte
- Fähigkeitspunkte
- Karmapunkte
- Ansehenspunkte

Anstelle eines komplexen Spielesystems wo verschiedene Punktesysteme benutzt werden, wurde hier entschieden nur eine Art von Punkten zu benutzen, welche größtenteils auf den Erfahrungspunkten aufbaut. Erfahrungspunkte stellen die wichtigste Kategorie von Punkten dar, jede Interaktion des Benutzers wird mittels Erfahrungspunkte festgehalten ((Zichermann und Cunningham, 2011)) und geben bei einer bedachten Punktevergabe eine gute Quantifizierung der Leistung des Benutzers womit man Erfahrungspunkte gleichzeitig für eine Bestenliste benutzen kann - und damit als eine Art Ansehenspunkte. Um den Punktemechanismus noch etwas spannender zu gestalten, kann man sie für sogenannte *Booster* eintauschen durch welche jeder Erhalt von Punkten für einen gewissen Zeitraum mit einem Multiplikator erhöht wird z.B. erhält der Benutzer für einen Tag die doppelte Anzahl an Punkten. Die Punkte sind für den Benutzer immer sichtbar und aktualisieren sich in Echtzeit sobald der Benutzer eine Aktion ausführt die Punkte bringt. Zudem werden sie auf der Abzeichenseite nochmals angezeigt wobei eine Balkendiagramm eine genauer Kategorisierung anzeigt also woher der Benutzer diese Punkte überhaupt hat, die Kategorien sind: Einloggen, Suchen, Favorisieren, Bewerten und Abzeichen.

TODo BILDER DER PUNKTE

Zusammengefasst benutzen wir **Erfahrungspunkte** die für alle Aktionen vergeben werden, für *Booster* ausgegeben werden können und für Bestenlisten benutzt wird.

2.2.2.2 Bestenliste

Der Sinn von Bestenlisten ist es einen einfachen Vergleich von Dingen (in unserem Fall Benutzern) zu bieten. Bestenlisten sind so zu einem integralen Bestandteil unseres alltäglichen Lebens geworden, dass wir sofort eine erkennen wenn wir eine sehen. Beispiele für Bestenlisten im echten Leben umfassen die Ergebnisse der Stiftung Warentest, Klausurnotenhaushänge oder die Bundesligatabellen.

Nach (Zichermann und Cunningham, 2011) kann man zwischen zwei Arten von Bestenlisten unterscheiden, die *lass-dich-nicht-entmutigen* und die *unendliche* Bestenliste. Bei ersterer wird der Benutzer, egal welchen Platz er in der Bestenliste inne hat, immer direkt in die Mitte getan. Hierbei ist es irrelevant ob er Platz 43 oder 40000 ist, er sieht sich selbst immer in der Mitte der Liste. Wenn der Benutzer auf einen der Top 10/20/30 Plätze kommt sollte sich dieses Verhalten jedoch ändern und er sollte die *korrekte* Liste sehen. Bei der *undendlichen* Bestenliste werden alle Benutzer angezeigt, es gibt jedoch meistens die Möglichkeit die Anzahl der Benutzer zu verringern, so hat z.B. das Spiel *Doodle Jump* 3 Arten des Leaderboards: Lokal, Freunde und Global.

Ein weiterer Aspekt ist der Zeitraum der angezeigt wird. Vor allem bei Applikationen bei denen die Punktzahl immer weiter akkumuliert werden kann und demnach unendlich Punkte zulassen, kann eine Bestenliste die auf dem kompletten verfügbaren Zeitraum aufbaut für Neuanfänger entmutigend wirken da sie es wohl nie schaffen werden einen höheren Platz zu erreichen wenn jeder doch gleich viele Punkte pro Tag erzielen kann. Eine offensichtliche Lösung wäre es den Zeitraum auf z.B. die letzte Woche zu beschränken, was dann aber einen ähnlich negativen Effekt auf die dauerhaften Nutzer haben würde: Sie könnten ihre Vormachtstellung nicht demonstrieren.

Wir haben uns deswegen entschieden immer 2 Bestenlisten anzuzeigen: Der komplette Zeitraum und die letzten 30 Tage. Dadurch ist es möglich beide Parteien zufrieden zu stellen ohne Kompromisse eingehen zu müssen. Es werden ebenfalls beide Arten der Bestenliste benutzt: Im persönlichen Dashboard wird die *lass-dich-nicht-entmutigen*-Bestenliste und in den globalem Statistiken wird eine *unendlichen* Bestenliste benutzt.

TOD BILDER DER BESTENLISTE

2.2.3 Abzeichen

Abzeichen bzw. Badges sind ein etabliertes Element der Gamification. Nach (Zichermann und Cunningham, 2011) nutzen sie viele Eigenschaften der menschlichen Natur um begehrswert zu sein: das menschliche Sammelverhalten, die plötzliche (positive) Überraschung wenn ein unerwartetes Abzeichen erhalten wurde, Sozialer Status (ähnlich zu Punkten), Pure Ästhetik der Abzeichen und das Aufgabenziel das hinter jedem Abzeichen steckt.

In manchen Fällen sind Abzeichen so effektiv eingesetzt dass sie sogar Level ersetzen - z.B. mit speziellen Level-Badges, als Beispiel sei Foursquare *Foursquare ist eine iPhone-Applikation mit der man an Orten einchecken kann* gennant bei der der Benutzer das Abzeichen für das erreichen einer bestimmten Anzahl an *check-ins*⁸ erhält. Dies funktioniert bei Foursquare darum gut, da *check-ins* der wichtigste Bestandteil der Applikation sind. Auch haben *check-ins* haben keine negative Konnotation da es zumeist

⁸Der Begriff *check-in* wird von vielen sozialen Diensten benutzt um an sich an einem realen Ort "einzuchecken" - um dies z.B. seinen Freunden mitzuteilen

als überaus positiv angesehen wird wenn man häufig ausgeht und neue Orte kennen lernt und man damit auch gerne zeigt wieviele *check-ins* man schon hatte.

Trotz aller positiven Aspekte von Abzeichen sind sie nicht einfach zu benutzen, so benutzt (Zichermann und Cunningham, 2011) den Begriff “*badgenfreude*” für ein Konzept, bei dem zu viele langweilige und sinnlose Abzeichen der ganze Sinn zunichte gemacht wird und negative Folgen für das *Engagement* haben können.

Für mich war es vor allem wichtig dass die Abzeichen den Benutzer nicht stören, er soll es mögen sie zu bekommen. Aber wann mag es ein Benutzer Abzeichen zu bekommen und wann fangen sie an zu stören? Als erstes kann man die Frequenz des Erhalts betrachten, um anfangs den Benutzer zu motivieren gibt man ihm meistens relativ viele Abzeichen so ist typisch ein Abzeichen für die erste Anmeldung, die erste geschaffene Herausforderung ... etc. zu geben. Die Frequenz muss jedoch mit der Zeit nachlassen, damit sie den Benutzer nicht von der eigentlichen Applikation ablenken und damit die Abzeichen weiterhin als wertvoll betrachtet. Es sollte für den Benutzer weiterhin immer offensichtlich sein wofür er ein Abzeichen bekommen hat und den Wert des Abzeichens sollte graphisch repräsentiert werden z.B. werden sehr oft Bronze/Silber/Gold benutzt um den Wert eines Abzeichens zu symbolisieren. Damit der Benutzer weiß welche Abzeichen es noch gibt und wie er sie erhalten kann, sollte es möglich sein alle erhältlichen Abzeichen einzusehen mit einer Beschreibung wie man denn eines erhält.

So hat sich schlussendlich folgendes Schema für unsere Abzeichen ergeben. Es gibt 3 Kategorien von Abzeichen, die jeweils eine anderen Wert darstellen. Die erste Kategorie sind leicht zu erhaltene Abzeichen, die zweite Kategorie ist schon schwerer, aber gut machbar wenn man dieses Abzeichen denn haben möchte. Die Abzeichen der letzten Kategorie benötigen ca. 50x mehr Zeit als die der ersten, dadurch sollen auch Vielbenutzer weiterhin durch das Abzeichensystem motiviert werden. Visuell wird der Wert durch eine Krone dargestellt, wobei die der letzten Kategorie am imposantesten aussieht, siehe dazu ???. Der Erhalt von neuen Abzeichen wird so schnell wie möglich dem Benutzer mitgeteilt werden - jede vom Benutzer ausgeführte Aktion wird an das Backend gesendet, welches **immer** berechnet ob dieser dafür ein neues Abzeichen bekommt. Ist dies der Fall wird das dem Frontend mitgeteilt und der Benutzer bekommt eine Nachricht angezeigt, siehe ???. Dieser Vorgang dauert weniger als eine halbe Sekunde. Damit das Abzeichensammeln nicht langweilig und obsolet wird, sollten immer mal wieder neue Abzeichen eingeführt werden, der Quelltext wurde daraufhin optimiert damit dies nicht länger als ein paar Minuten dauert.

Kapitel 3

Technische Umsetzung

Im Kapitel Umsetzung wurde die Applikation soweit geplant, dass die Technische Umsetzung erfolgen kann, dies geschieht in diesem Kapitel, dabei werden Themen behandelt wie warum eine Technologie genutzt wurde und nicht eine andere, wie die Architektur der Applikation ist und warum sie so ist und viele weitere Sachen die mir relevant erschienen. Die Technische Umsetzung wird sich größtenteils mit dem Frontend beschäftigen da dies den größten und auch interessantesten Teil der Applikation darstellt.

3.1 Wahl der Technologie

Die erste Frage die sich bei der Umsetzung stellt, ist welche Technologie verwendet werden soll.

Die Wahl der zu verwendeten Technologie ist ein wichtiger Punkt der die darauffolgende Entwicklung und spätere Wartung beeinflusst. Es wird versucht auf folgende Punkte einzugehen:

- Technologie ist *erwachsen*

Die verwendete Bibliothek oder ähnliches ist nicht allzu neu und hat sich in vielen Applikationen bewährt, sie ist soweit ausgereift das Umgehungslösungen oder Fehlerbehebung nur in den seltensten Fällen nötig sein sollten.

- Die Technologie hat keine große Einstiegsbarriere

Es wird versucht nicht allzu viele Frameworks zu benutzen und wenn, dann welche die weitestgehend bekannt sind und/oder in dem DAI-Labor viel benutzt werden. Wenn Bibliotheken verwendet werden, wird darauf geachtet das sie einfach zu lernen sind und intuitiv in der Benutzung.

- Die Technologie ist zukunftssicher

Es ist immer schwer ab zu schätzen welche Technologie länger überleben wird, doch wird versucht anhand von Faktoren wie Popularität, Aktivität der Entwicklung und Einsatz bei großen Firmen dies so gut wie möglich zu garantieren.

3.1.1 Ladezeiten

Ein großer Faktor der in anderen Umgebungen lange Zeit keine Rolle mehr spielt ist die Größe des Quelltextes. Bevor bei Javascript irgendeine Bibliothek benutzt werden kann muss sie zuerst beim Benutzer heruntergeladen und ausgeführt werden, was bei

langsamen Computern mit nicht perfekter Internetverbindung ein enormer Bestandteil ist. Um den Punkt nochmal zu verdeutlichen wurde ein kleiner Test gemacht. Dazu wurde eine EmberJS-Demoapplikation ausgeführt¹ die sehr wenig wirkliche Logik besitzt, sodass gut zu zeigen ist wie viel das Herunterladen und Ausführen nur der Bibliotheken ausmachen. Die Anwendung ist auf github zu finden².

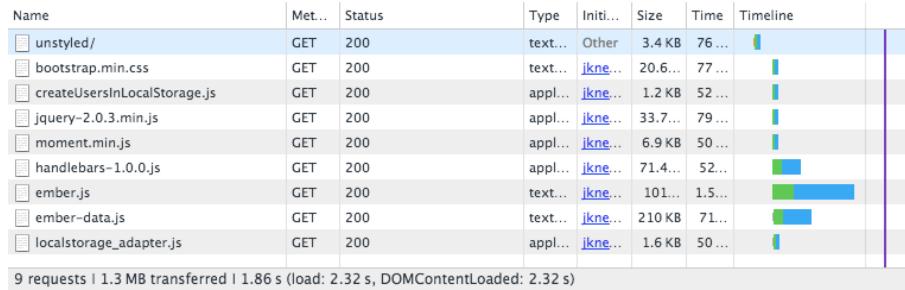


Abbildung 3.1: Übertragungszeit der benötigten Dateien

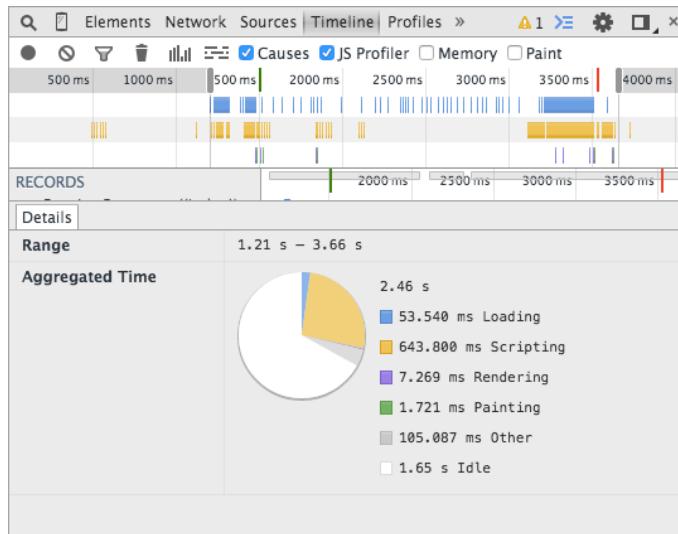


Abbildung 3.2: Benötigte Ausführungszeit der Skriptdatein

Die Ergebnisse zeigen, dass ganze 1.86s zum Übertragen und etwa 600ms zum ausführen den Anwendung benötigt wurden.

Aufgrund dessen wurde neben dem Versuch möglichst wenige Bibliotheken oder Frameworks zu benutzen (z.B. wenn nur eine Funktion einer großen Bibliothek benutzt wurde, wurde diese Funktion durch eine minimalere Bibliothek ersetzt oder selbst geschrieben bzw. kopiert.) auch diverse Techniken zum minimieren des Quelltextes genutzt.

3.1.2 Programmiersprache

Die wichtigste Wahl ist die Wahl der Programmiersprache, im Frontend wird diese Entscheidung größtenteils dadurch abgenommen das Browser die Ausführung von

¹Auf einem Macbook Air mid 2013 in minimal Konfiguration und dem Chrome Browser in Version 41, die vorhandene Internetverbindung hatte eine Übertragungsrate von $50000 \frac{\text{bit}}{\text{s}}$

²<http://jkneb.github.io/ember-crud/unstyled/#/users>

JavaScript beherrschen, andere Sprachen sind nur über Plugins (Flash, Java) verfügbar deren Relevanz in den letzten Jahrzehnten soweit abgenommen hat dass sie keine gute Wahl mehr darstellen.

JavaScript ist jedoch nicht die beliebteste Sprache, die Tatsache das die Sprache in ca. 10 Tagen geschrieben wurde mag dies erklären (Severance, 2012), so kam es dass JavaScript ist in den letzten Jahren immer mehr zu einem Kompilierungsziel geworden ist, d.h. andere Programmiersprachen kompilieren zu validem JavaScript-Code. Diese Praxis ist mittlerweile so populär geworden, dass JavaScript auch als *Assembly of the Web*³ bezeichnet wird. Die populärsten Programmiersprachen die zu JavaScript kompilieren sind vermutlich CoffeeScript, TypeScript und Java (mittels GWT). Die Liste der Sprachen ist jedoch endlos mit über 300-Sprachen die zu JavaScript kompilieren⁴.

(Zakai, 2011) veröffentlichten mit *emscripten* das mit Abstand aufwändigste Projekt in diesem Bereich. Dies ist ein LLVM-zu-JavaScript-Kompilierer mit dem es unter anderem möglich ist C++-Quelltext zu JavaScript zu kompilieren, wodurch z.B. TeX-Compiler im Browser ausgeführt werden konnten⁵.

Die Nutzung dieser Sprachen, vor allem der populärsten hat viele Vorteile von vereinfachter Syntax (CoffeeScript) hin zu statischer Typisierung (GWT), so bleibt die Frage ob es überhaupt noch zukunftsicher bzw. praktikabel ist reines JavaScript zu benutzen. Die Frage kann mit einem klaren Ja beantworten werden, wenn eine Prämissen für die Ausführungsumgebung gesetzt werden: Sie unterstützt JavaScript in der Version 6, bzw. formal bekannt als ECMAScript 6 (Im nachfolgenden mit ESx abgekürzt). Die für JavaScript zuständige Organisation ECMA investierte nämlich viel Arbeit darein die Tatsache dass JavaScript nur in 10 Tagen geschrieben wurde etwas weniger offensichtlich zu machen und mit ECMAScript 6 zeigt diese am deutlichsten Früchte. Kaum ein Merkmal dass man von High-Level-Sprachen wie Ruby oder Python kennt muss man hier missen.

Die aktuellen Browser unterstützen bisher jedoch nur einen Bruchteil der ES6 Spezifikation⁶ und vermutlich wird es noch bis in 2017 andauern bis alle aktuellen Browser ES6 ausreichend unterstützen.

Doch sind die Vorteile so überwiegend im Vergleich zu ES5, dass eine Nutzung von ES6 äußerst wünschenswert ist. Die Lösung des Problems wurde im Grunde schon erwähnt: Man kompiliert JavaScript zu JavaScript bzw. ES6 zu ES5. So absurd dies auch klingen mag, die Tatsache das JavaScript heutzutage immer im Zusammenhang mit build-tools benutzt wird die z.B. den Quelltext verkleinern oder ihn unlesbarer machen, bedeutet dass der Entwickler nur minimalen Aufwand betreiben muss damit er ES6 auch in ES5 Umgebungen benutzen kann.

So wurde für diese Arbeit entschieden, dass ES6 die beste Wahl für eine Programmiersprache ist. Alle Browser werden ES6 in Zukunft unterstützen womit ES6 keinesfalls eine obsolete Sprache werden wird. Außerdem ist im Vergleich zu den richtigen Kompilier-zu-JavaScript-Sprachen, ES6 die geringste Hürde für neue Entwickler die JavaScript kennen: ES6 ist abwärtskompatibel und die neuen Funktionalitäten können in ein paar wenigen Stunden gelernt werden, wenn dies überhaupt nötig ist.

³Hier sogar vom JavaScript-Erfinder persönlich: <https://brendaneich.com/2015/06/from-asm-js-to-webassembly/>

⁴Eine Auflistung dieser ist auf dem github-repository von CoffeeScript zu finden <https://github.com/jashkenas/coffeescript/wiki/list-of-languages-that-compile-to-js>

⁵<https://github.com/manuels/texlive.js>

⁶Eine komplette Übersicht ist hier zu finden <http://kangax.github.io/compat-table/es5/>

Neben reinem ES6 benutzen wir jedoch auch noch eine von Facebook entwickelte Erweiterung namens JSX, diese Erweitert JavaScript mit der Möglichkeit UI-Komponenten mit einer HTML-Syntax zu beschreiben. Dies wäre z.B. valides JSX:

```
~~Ivar header = <h1>Überschrift</h1>;
```

Es mag merwürdig erscheinen HTML-markup direkt in JavaScript zu schreiben, doch wird dadurch die Benutzung von der hier verwendeten Interface-Bibliothek *React* deutlich vereinfacht. Bisher ist JSX ein Vorschlag von Facebook an ECMA, doch ist die Benutzung von JSX mittlerweile so verbreitet⁷ dass eine Integration etwas JSX-ähnlichem ziemlich wahrscheinlich ist. Mittels ES6-templatestrings ist es möglich DSLs⁸ zu schreiben, so gibt es mittlerweile auch ein proof-of-concept JSX rein in ES6 zu schreiben⁹.

Neben JavaScript muss auch CSS geschrieben werden, da CSS ebenfalls viele Mängel hat gibt es auch Projekte wie *less*¹⁰ oder *sass*¹¹, die CSS z.B. mit Variablen erweitern. Es hat sich bewährt less oder sass zu benutzen, so wurde auch für diese Applikation entschieden *less* zu benutzen, da damit auch Bootstrap geschrieben wurde - das hier verwendete CSS-Framework. Das Ausmaß dieser Entscheidung ist jedoch kein Vergleich zu der Programmiersprachen-Entscheidung, weswegen es nicht lohnt die Kosten/Nutzen genauer zu betrachten.

3.1.3 Frameworks und Bibliotheken

Das Entwickeln von Webanwendungen wurde schon etliche male gemacht und somit gibt es kaum ein Problem das noch nicht gelöst wurde. Es gibt etliche Bibliotheken und Frameworks die einem das Leben einfacher machen sollen, ein klarer Platzhirsch ist jedoch nur für Subkategorien wie Datenvisualisierung zu finden, so ist die Wahl der Frameworks/Bibliotheken keinesfalls so trivial wie etwa in Java, wo Spring/Hibernate der Standard ist wenn man eine Webanwendung schreiben will.

Es ist schwer in der Welt der Webanwendungen mit der Geschwindigkeit ,in der neue (und auch praktikable) Bibliotheken und Frameworks veröffentlicht werden, mitzuhalten. Der nachfolgende Abschnitt befasst sich deshalb ausführlicher mit dem Entscheidungsfindungsprozess in diesem Bereich und zeigt auf welche Dinge alles geachtet werden muss.

Die einflussreichste Entscheidung hierbei ist die Entscheidung für ein Framework a la Dojo, AngularJS oder EmberJS. Es gibt viele Vorteile die für die Benutzung eines Frameworks sprechen: Große Benutzergemeinde, eine immense Anzahl an vorgefertigten Lösungen für häufige Probleme/Aufgaben, eine klare Projektstruktur und viel vorgefertigte Komponenten.

Die Benutzung eines Frameworks spart also Zeit und beschleunigt die Entwicklung, doch gibt es auch Gründe warum immer mehr Leute sich den großen Frameworks abwenden. Alle Frameworks besitzen eine große Lernkurve, jeder Entwickler ist gezwungen das Framework zu lernen um produktiv mitzuarbeiten da ein Framework

⁷eslint - <http://eslint.org/> unterstützt JSX wie auch der ES6-ES5-Kompilierer Babel - <https://babeljs.io/>

⁸Domain Specific Languages

⁹<http://www.2ality.com/2014/07/jsx-template-strings.html>

¹⁰<http://lesscss.org/>

¹¹<http://sass-lang.com/>

jeden Aspekt der Anwendung berührt. Frameworks veralten gerne und auch bei aktiver Entwicklung sind ständige API-Änderungen der Standard. Dem Entwickler bleibt nichts anderes übrig als sich mit dem Framework zu bewegen. Die vom Framework bereitgestellten Abstraktionen sind oft nicht optimal und verbergen zu viel oder sind auf einen bestimmten Benutzungszweck hin optimiert, womit sie mehr ein Hindernis darstellen können sobald etwas gemacht werden muss dass die Entwickler vergessen haben zu beachten. Weiterhin ist jede vom Framework erschaffene Abstraktion mehr ausgeführter Quelltext und wenn diese nicht gerade direkt der Geschwindigkeit der Applikation zu gute kommen, verlangsamen sie die Applikation. Dies ist normalerweise nicht so sonderlich wichtig, jedoch sind Webanwendungen Benutzerinterfaces bei der die Geschwindigkeit einen erheblichen Teil der Benutzererfahrung ausmacht, im ersten Abschnitt ging ich auf die Ladezeiten ein die durch die Benutzung eines Frameworks immer größer sind.

Allen Vorteilen zu trotz wurde für diese Arbeit entschieden das die Benutzung eines Frameworks nicht die optimale Lösung ist, da z.B. die Visualisierung so aufwändig ist, dass hier von vornherein klar war das die gegebenen Abstraktionen nicht ausreichen würden¹².

Es wurde entschieden eine Ansammlung von Bibliotheken zu benutzen, die sich für ihren Bereich bewährt haben. Dadurch wird garantiert dass die bestmögliche Benutzererfahrung möglich ist, da sobald eine Bibliothek nicht ausreicht ersetzt oder ergänzt werden kann.

Es wurden am Ende folgende Bibliotheken verwendet

react wird für die komplette Darstellung der Applikation benutzt. Im nächsten Abschnitt gehe ich auf react.js noch genauer ein, da es viele Innovationen in diesen Bereich bringt.

react-router¹³ ist eine routing-Bibliothek für react und macht es sehr einfach URLs zu einer View zuzuordnen.

reflux¹⁴ ist eine kleine Bibliothek für eine unidirektionelle-Datenfluss Architektur, mehr dazu im nächsten Abschnitt.

bootstrap¹⁵ ist das meist benutzte CSS-Framework und vereinfacht das erstellen von responsiven Applikationen. Weiterhin stellt es einige nützliche Komponenten wie z.B. eine Navigationsleiste bereit.

react-bootstrap¹⁶ bietet für alle verfügbaren bootstrap-Komponenten die entsprechenden react-Komponenten.

react-router-bootstrap¹⁷ verbindet react-bootstrap-Komponenten mit react-router.

d3.js¹⁸ ist die berühmteste Bibliothek für Visualisierungen. Hiermit wurde das animierte Säulendiagramm geschrieben, aber auch vieles der Farberzeugung entstand mithilfe von d3.js.

immutable.js¹⁹ bringt nicht veränderbare Datenstrukturen in die Welt von JavaScript. Sie wurde bei der Verarbeitung der Suchergebnisse bis hin zu deren Darstellung benutzt, der Vorteil wird unten noch genauer betrachtet.

¹²Ich habe viel Erfahrung mit der Benutzung einigen großen Frameworks wie AngularJS und EmberJS.

lodash²⁰ ist eine utility-Bibliothek mit einem starken Fokus auf funktionaler Programmierung. Sie wurde so ziemlich überall in der Applikation benutzt wo Daten verarbeitet werden mussten.

cookies.js²¹ / **store.js**²² sind mini-Bibliotheken die das arbeiten mit *cookies* und *localStorage* vereinfachen. *cookies* werden benutzt um den Benutzer eingeloggt zu lassen und *localStorage* um Suchergebnisse zu cachen.

moment.js²³ vereinfacht den Umgang mit Datums-Objekten und bietet Funktionen wie `add(1, day)` und erweiterte Formattierungsmöglichkeiten. In der aktuellen Version ist es mit `moment` nicht möglich deren eigene *duration* Objekte zu formatieren, dafür wurde das plugin `moment-duration-format`²⁴ benutzt welche womöglich auch bald zu `moment` gehören wird.

3.1.4 Tooling

Das Tooling im Frontend hat sich in den letzten Jahren rasant verändert, kaum jemand schreibt heutzutage noch JavaScript oder CSS ohne das ein Build-Werkzeug diese Dateien nochmals verändern.

Die bekanntesten Build-Werkzeuge sind Gulp²⁵, Grunt²⁶, Webpack²⁷ und Brunch²⁸.

Webpack stellt eine Ausnahme da, da es im Grunde gar kein Build-Tool ist, sondern ein *module bundler* d.h. es verarbeitet alle in der Entwicklung relevanten JavaScript/CSS/Bild-Dateien und bündelt diese. Durch die Architektur von Webpack ist es möglich so ziemlich jeden Datei-Typ zu verarbeiten den es gibt. Da die meisten Build-Systeme eben diesen Output haben, kann Webpack diese meistens komplett ersetzen.

Zwar ist Webpack noch ein junges Projekt es hat sich jedoch schon für große Projekte wie Instagram bewährt dessen Chefentwickler Pete Hunt darauf schwört²⁹.

Für dieses Projekt habe ich mich auch für Webpack entschieden, da es von allen Build-Systemen die folgenden Dinge am einfachsten realisieren konnte:

1. Es muss möglich sein JavaScript-Bibliotheken mittels npm zu benutzen.
2. ES6-Quelltext muss zu ES5 Quelltext kompiliert werden.
3. Less-Style

3.1.5 Facebook React und die Flux Architektur

Facebooks React³⁰ ist eine JavaScript-Bibliothek die seit ihrem erscheinen im Jahr 2013 viel Beifall geerntet hat. Zum Zeitpunkt dieser Arbeit wird sie produktiv von

²⁴<https://github.com/jsmreese/moment-duration-format>

²⁵<https://github.com/gulpjs/gulp>

²⁶<https://github.com/gruntjs/grunt>

²⁷<https://github.com/webpack/webpack>

²⁸<https://github.com/brunch/brunch>

²⁹<https://github.com/petehunt/webpack-howto>

³⁰<https://github.com/facebook/react>

vielen Firmen benutzt wie z.B. *Khan Academy*³¹, *Netflix*³², *Yahoo!*³³, *Sony*³⁴ und anderen. Facebook benutzt sie selbst für ihre größten Produkte Facebook und Instagram (*Google* benutzt ihr Framework *AngularJS* z.B. nicht für das Flaggschiff *Google Mail*).

React ist jedoch kein Framework wie AngularJS oder Ember, es gibt keine Directives, Controllers, Templates oder Models. Das einzige was React bietet sind *Components*, welche ähnlich wie der kommende Standard der *Web Components*³⁵ wiederverwendbare Interface-Widgets darstellen indem Sie HTML/CSS und JavaScript koppeln und vom Rest des System separieren.

Dabei folgt React nicht den *best-practices* die in den letzten Jahren in Webentwicklung entstanden sind und sich in MVC-Frameworks widerspiegeln. Klassisch gibt es ein Template, dass in HTML oder einem ähnlichen Sprache wie YAML geschrieben ist und Platzhalter enthält die später gefüllt werden - dies ist das V in MVC. Dazu kommt ein Controller der diese Templates mit Daten füllt, auf welche Art ist nicht relevant. Kommuniziert der Controller mit einem Server um Daten zu übertragen wie die eines Benutzers stellt dies das M da, also nicht die Kommunikation selbst sondern die Datenstrukturen auf beiden Seiten die den Benutzer darstellen. Applikationen auf diese Weise zu erstellen hat sich bewährt was jedoch nicht heißt es gibt keine Kritik.

3.1.5.1 Virtual DOM

Einer der innovativsten Aspekte von React ist die virtuelle Repräsentation des DOMs³⁶. Die normale Interaktion mit dem DOM ist langsam und kompliziert, es ist sehr schwer den aktuellen Status der Applikation darüber abzubilden - so geschieht das häufig über CSS-Klassen oder speziellen Attributen. Eine Lösung dieses Problems ist es die Applikation so zu schreiben, dass der ganze DOM bei jeder Veränderung des Zustands neu gerendert wird. (Vergleich hier zu Server-seitigem Rendern wo genau dies gemacht wird.) Natürlich wäre die Applikation bei so einem Vorgehen langsam (wegen des rendern durch den Browser) und Dinge wie Fokus auf Eingabe-Feldern würde verloren gehen was der Benutzer nicht gerade angenehm finden würde.

Mittels eines virtuellen DOMs kann dies deutlich effizienter und Benutzerfreundlicher gestaltet werden. Bei jeder Zustandsänderung wird ein neuer virtueller DOM erstellt und mit dem alten virtuellen verglichen wodurch die minimalen Änderungen erfasst werden können um den derzeitigen DOM zum Zustand des neuen zu bringen. Dies macht Frameworks die einen virtual DOM benutzen deutlich schneller als welche die die Hauptarbeit auf dem richtigen DOM verrichten. Das ganze Konzept ist skalierbar im Gegensatz zu Technologien wie dem *dirty checking* von AngularJS welches eine Komplexität von $O(n^2)$ bis $O(n^{10})$ (Ab 10 Iterationen wird abgebrochen) besitzt, wobei n die Anzahl der veränderbaren Inhalte ist. Der von Facebook verwendete Algorithmus besitzt eine Komplexität von $O(n)$ ³⁷ wobei n die Anzahl der DOM-Element ist. Ein Geschwindigkeitsvergleich von Frameworks die einen virtuellen DOM benutzen zeigt diese Grafik vom Elm³⁸-Entwickler bei der die verschiede-

³¹<http://joelburget.com/backbone-to-react/>

³²<http://conf.reactjs.com/schedule.html#beyond-the-dom-how-netflix-plans-to-enhance-your-television>

³³<http://www.slideshare.net/mobile/rmsguhan/react-meetup-mailonreact>

³⁴<https://medium.com/code-stories/dev-chats-spoke-brehm-of-airbnb-87e155f3475d>

³⁵https://developer.mozilla.org/en-US/docs/Web/Web_Components

³⁶http://de.wikipedia.org/wiki/Document_Object_Model

³⁷<https://facebook.github.io/react/docs/reconciliation.html>

³⁸http://en.wikipedia.org/wiki/Elm_programming_language

nen Frameworks anhand ihrer TodoMVC³⁹-Implementierung verglichen werden.

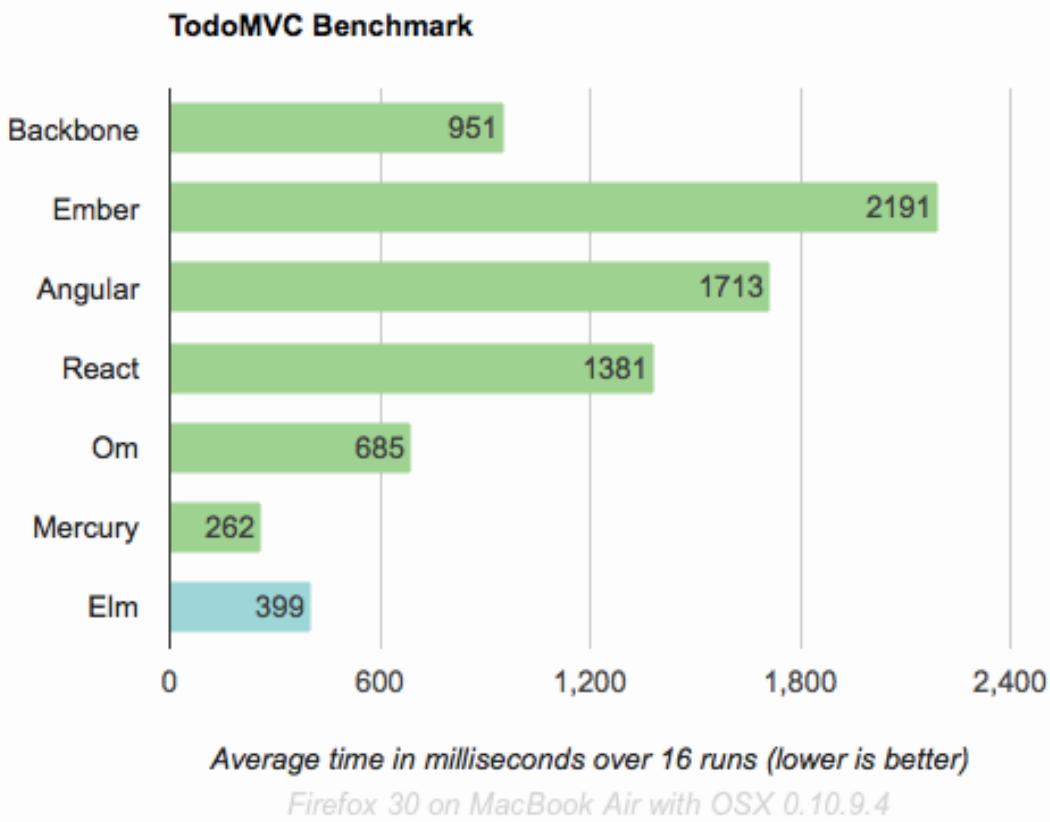


Abbildung 3.3: Geschwindigkeitsvergleich von TodoMVC-Implentierungen mit verschiedenen Frameworks.

Es ist zu sehen, dass Ember und AngularJS am langsamsten sind. Backbone ist in diesem Benchmark schneller als React, wobei die verwendete Implementierung der React-App nicht optimiert ist - *Om*⁴⁰ kann als optimierte React-Version betrachtet werden, da Om im Grunde React mit nicht veränderbaren Datenstrukturen⁴¹ (immutable data-structures) ist. Backbone wie aber auch JQuery oder Dojo können theoretisch so schnell gemacht werden wie es möglich ist, dies ist aber so komplex, dass es nie gemacht wird und Wartbarkeit und Fehlervermeidung über Performance gestellt wird. Die beiden schnellsten Frameworks benutzen ebenso eine virtual-DOM Implementierung und wie *Om* auch nicht veränderbare Datenstrukturen. Dies ist auch der Grund weswegen Ich `immutable.js` verwende da dies die render-Geschwindigkeit in den aufwändigsten Teilen verbessern konnte.

Ein weiterer Vorteil der virtuellen Darstellung ist die Unabhängigkeit des Browsers um die Applikation zu rendern, dass schlussendliche rendern des virtuellen DOMs in den richtigen DOM ist nicht zwingend. Es ist z.B. möglich den virtuellen DOM in ein Canvas-Element zu rendern⁴² oder ganz einfach in einen String. Durch das simple

³⁹Der TodoMVC ist ein Versuch die schiere Anzahl an Webframeworks anhand eines realen Beispiels zu vergleichen. <http://todomvc.com/>

⁴⁰<https://github.com/omcljs/om>

⁴¹Durch nicht veränderbare Datenstrukturen kann das diffing noch effizienter gestaltet werden, da viele Teilbäume nicht betrachtet werden müssen.

⁴²<https://github.com/Flipboard/react-canvas>

rendern zum String, was ganz ohne Browser möglich ist, beherrscht React isomorphes Rendering, d.h. die Inhalte können auf dem Server und im Client gerendert werden. So wird z.B. bei Instagram der erste Seitenaufruf auf dem Server alles weitere vom Clienten gerendert. Erste Versuche zeigen auch, dass man dies komplett agnostisch machen kann und alles kann auf Server/Client gerendert werden ohne das der Benutzer einen Unterschied merkt. So ist es denkbar das schwache Geräte mehr auf dem Server rendern und somit komplexe Inhalte vergleichsweise schnell darstellen können.

3.1.5.2 Flux Architektur

Flux⁴³ ist die Applikations-Architektur die Facebook für ihre Frontend-Applikationen benutzt, es ergänzt React insoweit dass es einen unidirektionellen Datenfluss ermöglicht. Es ist wie eine alternative zu MVC zu verstehen und ist mehr ein Muster um Applikationen zu schreiben als ein Framework das einem rigoros die Struktur vorschreibt. Im Allgemeinen benutzt Flux das Paradigma der Datenfluss Programmierung⁴⁴, bei welchem ein Programm als gerichter Graph modelliert wird. Typisch für das Paradigma ist die Benutzung eines Beobachter-Entwurfsmuster⁴⁵ wie z.B. eines pub/sub-System⁴⁶. Bei Flux wird für letzteres eine Abwandlung eines pub/sub-Systems benutzt.

Flux besitzt drei große Bestandteile: Einen *Dispatcher*, *Stores* und *Views (React components)*. Desweiteren gibt es *actions*, dies sind Hilfsmethoden vom *dispatcher* und werden benutzt um eine semantische API zu unterstützen die alle Veränderungen die möglich sind in der Applikation beschreibt.

Das wichtigste Designziel von Flux ist der unidirektionale Datenfluss, wodurch die Logik deutlich verständlicher und nachvollziehbar wird, siehe 3.4.

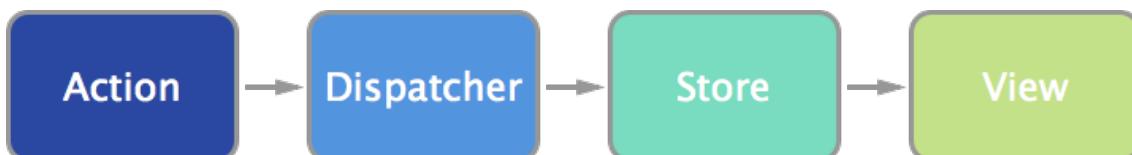


Abbildung 3.4: Dispatcher, Stores und Wiews sind unabhängige Knoten mit unterschiedlichen Ein und Ausgaben. Die Aktionen sind einfache Objekte die die neuen Daten enthalten und den Typ der Daten.

Diesem Schaubild zufolge ist es jedoch unmöglich dass eine *Component* den Zustand der Applikation verändern kann, was jedoch so gut wie immer nötig ist. Hierfür werden die *actions* benutzt, diese können von den *Components* benutzte werden um neue Daten an den Dispatcher zu schicken, welcher diese dann an den Store weiterleitet - welche dann die Darstellung der *Component* beeinflussen kann, siehe 3.5.

⁴³<https://facebook.github.io/flux/docs/overview.html>

⁴⁴http://en.wikipedia.org/wiki/Dataflow_programming

⁴⁵http://en.wikipedia.org/wiki/Observer_pattern

⁴⁶Eine kleine Auflistung und eine Bewertung typischer Muster ist hier zu finden <https://github.com/millermedeiros/js-signals/wiki/Comparison-between-different-Observer-Pattern-implementations>

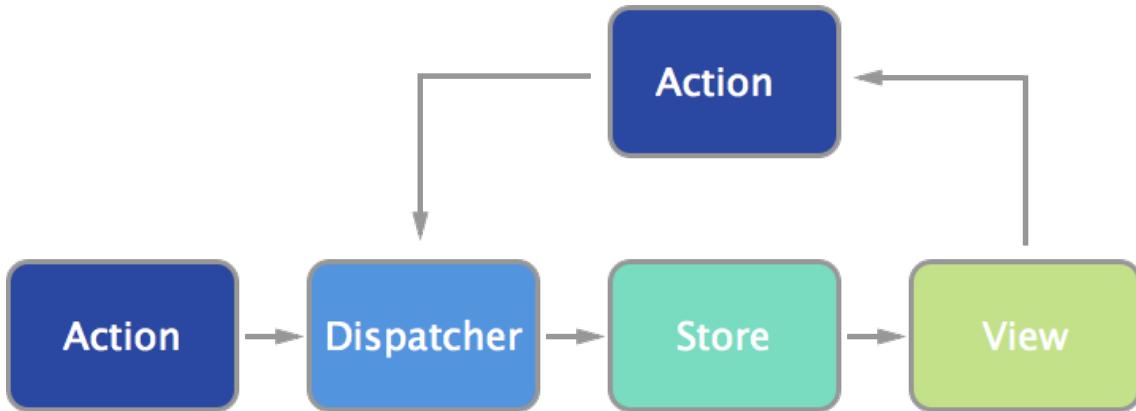


Abbildung 3.5: Mithilfe von *actions* ist die View in der Lage den Zustand der Applikation zu verändern.

Allem in allem ist die Facebook-Flux Architektur in Kombination mit React ein äußerst fortschrittlicher und zukunftssicherer Weg eine Webapplikation zu schreiben, womit die Anwendung dieser für die Erstellung der Enterprise-Infoboard eine gute Wahl erscheint.

3.2 Architektur der Applikation

Wie im vorherigen Abschnitt beschrieben wurde mittels React und der Flux-Architektur realisiert. Anstelle der Flux-Bibliothek von Facebook wurde jedoch *reflux* benutzt, *reflux* vereinfacht Flux insofern als das es keinen einzelnen Dispatcher benutzte, jede Action ist ihr eigener Dispatcher mit dem Ziel Flux eben mehr zu vereinfachen und unnötigen Quelltext (sogenannten Boilerplate) zu reduzieren. Eine Vollständige Erklärung für *reflux* ist auf dem Blog des Autors zu finden⁴⁷. *reflux* ist nicht der einzige Versuch eine alternative bereitzustellen, mittlerweile gibt es über 10 verschiedene Flux-Bibliotheken⁴⁸ - mit teilweise großen Communities dahinter oder Firmen wie z.B. *Fluxible* von *Yahoo*. Der Funktionsinhalt der verschiedenen Flux-Bibliotheken ist mehr oder minder gleich, somit könnte *reflux* mit Leichtigkeit ausgetauscht werden, wenn es denn nötig wäre. Die Entwicklung in diesem Bereich ist jedoch äußerst spannend und hat zum Zeitpunkt dieser Arbeit mit *redux* ihren Höhepunkt gefunden (Abramov, 2015), wo die Stores gar keinen State mehr besitzen und nur noch als action-Aggregatoren dienen. Dies ermöglicht das der Zustand der Applikation vor- und zurückgespult werden kann, er ist ja rein von den gesendeten actions abhängig.

⁴⁷<http://spoike.ghost.io/deconstructing-reactjs-flux/>

⁴⁸Ein Vergleich dieser ist hier zu finden <https://github.com/voronianski/flux-comparison>

3.2.1 Überblick

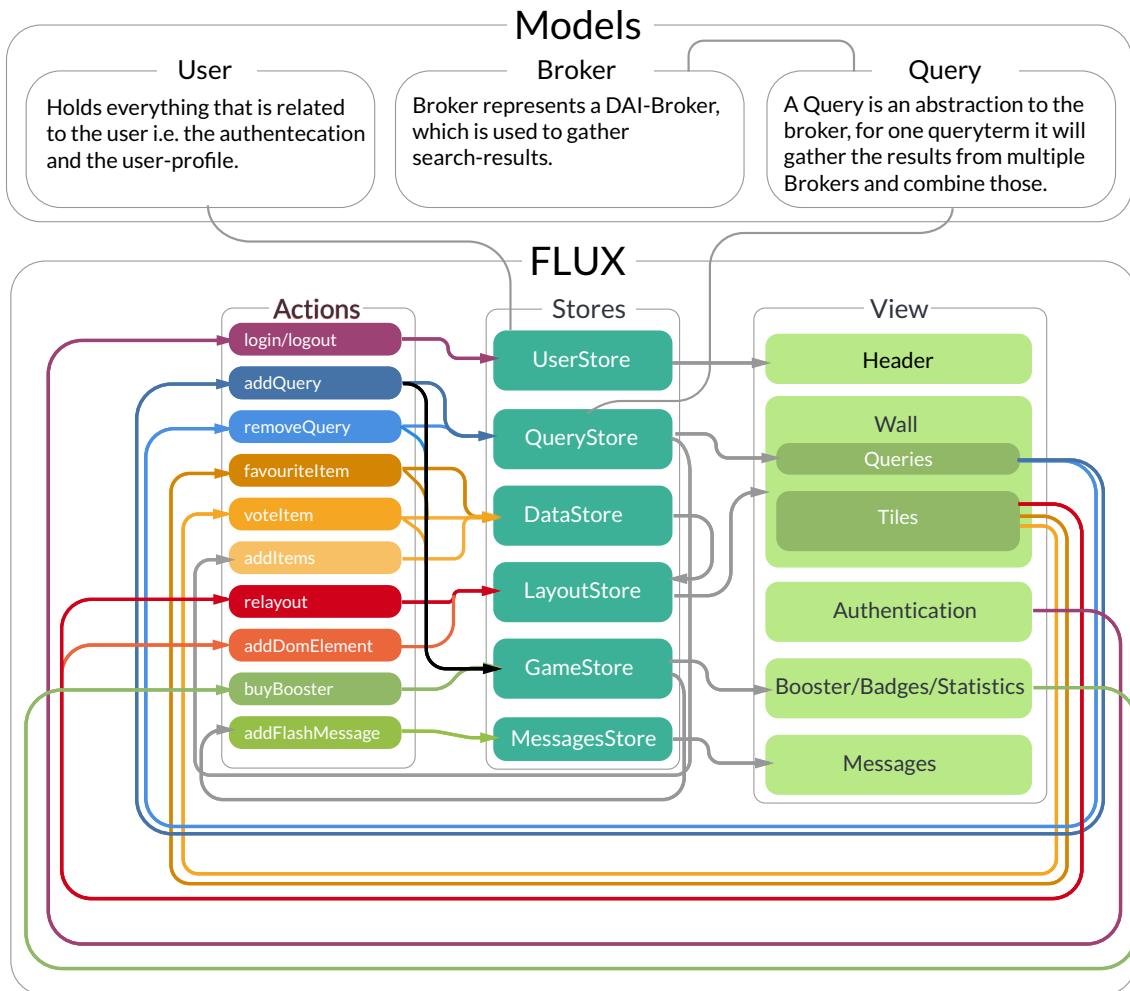


Abbildung 3.6: Die wichtigsten Bestandteile der Applikation.

Die Abbildung 3.6 zeigt einen Überblick über die gesamte Architektur, sie wurde bewusst nicht in einem Klassendiagramm dargestellt da die Applikation keinen strengen Objektorientierten Stil verfolgt und das Schaubild nicht sonderlich zum Verständnis beitragen würde. Dies ist eine grobe Abstraktion, doch reicht diese Abbildung um sich im Queltext zurecht zu finden.

Es ist deutlich zu sehen das der komplette Datenfluss bzw. Programmfluss mittels der Flux-Architektur modelliert wurde, was daran zu erkennen ist dass jegliche Interaktionen der View werden mittels *Actions* an die jeweiligen *Stores* weitergegeben wird die dann den neuen Zustand an die View abermals weitergeben.

Zu Flux gibt es noch drei Klassen die von den jeweiligen Stores benutzt werden, die Auslagerung dieser Logik in eigene Klassen dient der Übersichtlichkeit, verdeutlicht jedoch auch dass alle Komponenten mit denen der Benutzer nicht direkt interagiert nicht Teil des Flux-Datenflusses sein müssen.

Im folgenden werde ich genauer auf die Stores eingehen.

3.2.2 Stores

Die Stores sind zuständig den Datenfluss zu verwalten, insgesamt enthält die Applikation 6 Stores: UserStore, QueryStore, DataStore, LayoutStore, GameStore und MessagesStore. Der UserStore und MessagesStore sind jedoch äußerst trivial und sind nur wenige Zeilen lang. Die wichtigsten sind die anderen 4, diese werde ich nun im Detail behandeln.

3.2.2.1 QueryStore

Der QueryStore abonniert die Aktionen addQuery und removeQuery, er abstrahiert die Anfrage an die Broker d.h. für jeden Suchbegriff werden alle verfügbaren Broker angefragt, nach jeder Antwort eines Brokers wird die Nachricht addItems mit den Ergebnissen gesendet. Auf diese Nachricht hört der DataStore.

3.2.2.2 GameStore

Der GameStore liefert alles was für die Verspielisierung der Applikation relevant ist - wie die Bereitstellung der Punkte, der Bestenlisten und Trophäen aller Benutzer der Applikation. Weiterhin abonniert er alle Aktionen auf denen Punkte definiert sind wie das bewerten eines Suchergebnisses und sorgt dafür dass diese Aktionen vom Backend verarbeitet werden.

Um den sozialen Faktor zu erhöhen reagiert der GameStore ebenfalls auf alle Aktionen aller anderen Benutzer, dies geschieht quasi in Echtzeit und sorgt dadurch z.B. für eine äußerst lebendige Statistikseite (auf dieser werden die Daten aller Benutzer angezeigt). Damit das ganze noch effizient bleibt wurden WebSockets verwendet die deutlich weniger Overhead besitzen als normale HTTP-requests.

3.2.2.3 DataStore

Der DataStore stellt alle Daten bereit die für die Darstellung benötigt werden, dies sind die Suchergebnisse die mit dem Profil des Benutzers verbunden werden um anzuzeigen welche Ergebnisse bewertet/favorisiert wurden und die Suchanfragen, die zur Laufzeit verändert werden können. Herausforderungen hier waren es Duplikate in den Suchergebnissen zu verarbeiten und die Interaktion mit den Suchergebnissen.

Hierzu hört er auf die Nachricht addItems welche eine Liste mit Suchergebnissen enthält. Diese Daten sind noch ganz ohne die Bewertungsdaten, welche hier angefragt werden und in die Suchergebnisse integriert werden. Sobald das geschehen ist, wird allen Abonnenten (in dem Fall der LayoutStore) mitgeteilt dass der Datensatz sich verändert hat.

Eine Besonderheit ist die Benutzung einer nicht veränderbaren OrderedMap in welcher die Suchergebnisse mit ihrer URL als Schlüssel gespeichert sind. Die Suchergebnisse werden in der Applikation noch viel weitergereicht weswegen viele Fehler vorbeugegt werden da nur der DataStore selbst die Suchergebnisse verändern kann.

3.2.2.4 LayoutStore

Im Design-Abschnitt wurde lange darauf eingegangen was die Layouting-Engine alles können muss, leider konnte keine der OpenSource-Bibliotheken allen Anforderungen

gerecht werden. Deswegen wurde extra für die Darstellung eine eigene Layouting-Engine erstellt. Die Grundlage ist die absolute Positionierung innerhalb des Browsers, diese ist notwendig um Animationen zu ermöglichen. Es gibt verschiedene Arten DOM-Elemente mittels CSS zu positionieren:

1. Mithilfe von top/left.

```
.position-top-left {
    position: absolute; /* or relative/fixed */
    top: 50px;
    left: 50px;
}
```

2. Mithilfe von transform: translate(x, y). Dies ist allgemein anerkannt schneller als zu sein und hat den Vorteil das subpixel-animationen möglich sind was im Allgemeinen flüssigere Animationen erlaubt.

```
.position-translate {
    position: absolute; /* or relative/fixed */
    transform: translate(50px, 50px);
    -webkit-transform: translate(50px, 50px);
}
```

3. Mithilfe von transform: translate3D(x, y, z). Dies ist mit Abstand das schnellste. Hierbei werden die Elemente zu Texturen gerendert und mit Einsatz des Grafikprozessors animiert. Dadurch ist es sogar möglich auf mobilen Endgeräten flüssige Animationen zu haben, da diese meist dedizierte Grafikprozessoren haben.

```
.position-translate3D {
    position: absolute; /* or relative/fixed */
    transform: translate3D(50px, 50px);
    -webkit-transform: translate3D(50px, 50px);
}
```

transform3D wird von allen aktuellen Browsern soweit unterstützt das unser Szenario möglich ist⁴⁹. Was weiterhin zu beachten ist, dass am Ende immer gerundete Pixelwerte innerhalb von TRANSFORM3D(x, y, z) benutzt werden sollten. Da Ansonsten die Elemente unscharf sind, siehe 3.7.

⁴⁹<http://caniuse.com/#feat=transforms3d>



Abbildung 3.7: Links eine Kachel mit geraden Werten bei translate3D, rechts mit jeweils .5 bei x/y .

Es ist eine Grundanforderung Geräte mit unterschiedlich großen Bildschirmen zu unterstützen, was dadurch erzielt wird, dass die Anzahl der Spalten variabel ist. Es gibt zwei Möglichkeiten die Anzahl der Spalten variabel zu machen:

1. Berechne die Breite mittels Javascript bevor das Element in den DOM eingefügt wird und setze die Breite mittels css.
2. Durch die Benutzung von *Media Queries* können in css verschiedene Darstellungen für unterschiedliche Bildschirmbreiten beschrieben werden.

Für die Positionierung müssen wir wissen wie viele Spalten zu jedem Zeitpunkt dargestellt werden sollen. Dafür muss die Breite in Javascript berechnet werden, somit liegt 1. nahe. Versuche haben jedoch gezeigt, dass es schneller ist die Breite des Elementes nicht per Javascript zu setzen sondern diese über die Media-Queries zu bestimmen. Zwar bedeutet dies, dass die gleiche Logik an zwei Orten in 2 Sprachen steht, aber da dies nicht mehr als jeweils 10 Zeilen sind ist es für die resultierende bessere Geschwindigkeit hinnehmbar.

Die Breite aller Elemente ist zu jedem Zeitpunkt gleich, die Höhe ist jedoch variabel. Da es unmöglich ist die Höhe auszurechnen ohne eine komplette Browser-Engine zu implementieren, muss das Element in den DOM eingefügt werden bevor wir dessen korrekte y-Position berechnen können, hierfür existiert die Aktion addDOMElement, welche die gleichnamige Funktion im LayoutStore ausführt. Diese Funktion ist neben der relayout-Funktion der einzige Ort an dem der DOM direkt ausgelesen wird, nämlich die Höhe des Elementes mit `offsetHeight`. Solange kein `resize`-Event getriggert wird müssen wir diesen Wert auch nicht ein zweites mal auslesen. Wenn zu allen Suchergebnissen eine Kachel angelegt wurde, wird die `layout`-Funktion aufgerufen und die Kacheln bewegen sich auf dem Bildschirm.

Der vollständige Programmfluss ist komplex, da er sehr auf Geschwindigkeit getrimmt wurde, der Programmteil ist ausführlich kommentiert und beschrieben.

Allgemein hat die Flux-Architektur hier gut funktioniert, der größte Manko entstand beim Abgleichen des LayoutStores mit dem DataStore. Wenn nun ein Element positiv bewertet wird, muss das Layout neu berechnet werden. Der LayoutStore weiß inhärent jedoch nicht welches Element sich verändert hat, es muss durch alle Elemente des DataStores mit den eigenen abgleichen. Dies fühlt sich nicht optimal an, jedoch ist stellt sich die Frage auf ob mit einer anderen Architektur das Problem einfacher wäre.

3.2.3 Views

Als View wird das Interface bezeichnet mit dem der Benutzer schlussendlich interagieren kann. Für jede View wird React und React-Bootstrap verwendet, die Gründe warum React verwendet wird sind in 3.1.5 genauer ausgeführt.

3.2.3.1 Responsives Design

Responsives Webdesign bezeichnet ein Paradigma bei dem Webseiten daraufhin geschrieben werden, dass sie Inhalte auf unterschiedliche Endgeräte optimieren, was praktikabel bedeutet das Inhalte abhängig von der Bildschirmbreite des Gerätes angezeigt werden.

Aktuell wird dies so umgesetzt dass jeglicher Inhalt mit der Benutzung eines *Grid*-Systems dargestellt wird.

Grid systems are used for creating page layouts through a series of rows and columns that house your content.

Damit wird auf der Webseite⁵⁰ von Bootstrap das Grid-System erklärt, für mehr Informationen wie genau Grid-Systeme funktionieren siehe die Webseite. In dieser Applikation wird das von Bootstrap bereitgestellte Grid intensiv benutzt um auf jedem Endgerät eine möglichst gute Benutzererfahrung zu garantieren.

3.2.3.2 Erstellte React-Komponenten

Wie auch in anderen Interface-Bibliotheken wird in React das Interface in Komponenten (in anderen Bibliotheken auch *widgets* gennant) unterteilt.

Jede React-Komponente besitzt Lebenszyklus-Methoden⁵¹, welche eine Feingranule Benutzung der Komponenten ermöglicht. Eine Benutzung dieser ist jedoch komplett optional, die einfachste React-Komponente hat nur einer render-Methode. An dieser Stelle wurde überlegt eine Einführung für React zu schreiben, stattdessen verweise ich auf externe Tutorials wie das offizielle von Facebook⁵².

Es wurden ca. 30 React-Komponenten erstellt, viele davon wurden an mehreren Stellen der Applikation verwendet. Unterteilt sind sie semantisch nach Aufgabe/Seiten wie z.B. settings, stats oder utility. Eine Minimalübersicht findet sich in 3.6, in welchem Alle Komponenten gezeigt werden die Teil des Flux-Kreises sind, alle restlichen sind statisch und spielen für die Architektur keine Rolle.

3.2.3.3 Beispiel einer interaktiven Komponente

Ich werde exemplarisch eine Komponente behandeln um aufzuzeigen wie deren Verwendung ist und wie sie erstellt wurde. Ich habe dazu die Queries Komponente ausgewählt, da sie kurz ist jedoch viele Konzepte benutzt. Ich werde sie minimal vereinfachen, damit der Abschnitt nicht allzu lang wird.

Fertig implementiert kann die Queries Komponente in ?? gesehen werden, ich werde nun Schritt für Schritt deren Implementierung angehen.

⁵⁰<http://getbootstrap.com/css/#grid>

⁵¹<https://facebook.github.io/react/docs/component-specs.html>

⁵²<https://facebook.github.io/react/docs/tutorial.html>

1. Die Queries Komponente soll den aktuellen Stand der Queries aufzeigen, dieser wird vom QueryStore verwaltet, weswegen wir auf seine Veränderungen abonieren, dies wird in *reflux* mittels eines Mixin bewerkstelligt.

```
React.createClass({
  mixins: [Reflux.listenTo(queryStore, 'onStoreChange')],
  onStoreChange: function (queries) {
    this.setState({
      queries: queries
    })
  },
  getInitialState: function () {
    return {
      queries: queryStore.queries
    };
  },
  render: function () {
    return (
      <div className='queries'></div>
    );
  }
});
```

Im Detail heißt das, dass die ersten Daten die die Komponente anzeigen wird mittels `getInitialState` vom `QueryStore` bezogen werden, bei jeder Veränderung wird der Zustand der Komponente mittels `this.setState` auf den Zustand des `QueryStore` aktualisiert. `setState` bewirkt dass die Komponente neu gerendert wird.

2. Diese Art der Benutzung ist jedoch so häufig, dass *reflux* ihn mittels `connect` vereinfacht hat:

```
React.createClass({
  mixins: [Reflux.connect(queryStore)],
  render: function () {
    return (
      <div className='queries'></div>
    );
  }
});
```

3. Als nächstes rendern wir die Queries, dies ist wiederum eine eigenständige Komponente die wir hier nur verwenden werden.

```
React.createClass({
  mixins: [Reflux.connect(queryStore)],
  createQueries: function () {
    var queries = this.state.queries.map(s => {
      return <Query query={s}/>;
    });
    return (

```

```

        <ul className='queries--list'>
            {queries}
        </ul>
    );
},
render: function () {
    return (
        <div className='queries'>
            {this.createQueries()}
        </div>
    );
}
);

```

Die Funktion `createQueries` benutzt die Liste von Suchbegriffen und erstellt für jeden eine Query-Komponente. In der `render`-Methode wird sie dann aufgerufen.

4. Bleibt zuletzt noch das Löschen von Suchbegriffen, wir könnten diese Methode genauso gut auch in die Query-Komponente tun, hier übergeben wir sie jedoch mit einer weiteren property an die Query-Komponente.

```

React.createClass({
    mixins: [Reflux.connect(queryStore)],
    removeQuery: function (query) {
        actions.removeQuery(query);
    },
    createQueries: function () {
        var queries = this.state.queries.map(s => {
            return <Query
                query={s}
                removeQuery={() => this.removeQuery(k)}
            />;
        });
    };

    return (
        <ul className='queries--list'>
            {queries}
        </ul>
    );
},
render: function () {
    return (
        <div className='queries'>
            {this.createQueries()}
        </div>
    );
}
);

```

3.3 Backend

Im Verlaufe der Arbeit wurde erkannt das eine dedizierte Backend-Applikation nötig ist um den Grad der Interaktion und Dynamik zu ermöglichen die ich mir als Ziel gesetzt hatte. Es gibt einige Frameworks die zur Frage standen, aber ich habe mich schließlich für node.js entschieden, da ich dadurch die Möglichkeit hatte einiges an Code mit dem Frontend zu teilen. Das node.js *non-blocking* ist und *event-driven IO* benutzt macht es Verteilhaft für Applikation die viele requests verarbeiten werden müssen. Anders als in klassischen Frameworks wie rails oder django wo bei jedem request ein neuer Thread aufgemacht der solange blockiert ist bis er abgehandelt ist, wird jeder request von einem *event-loop* abgeandelt (Tilkov und Vinoski, 2010). Alles in node.js ist asynchron, weswegen diese requests nicht blockieren. Damit ist es mit node.js möglich bis zu 1 Million gleichzeitige Verbindungen zu halten⁵³.

Als Datenbank wurde *mysql* verwendet, die mittels einem ORM⁵⁴ namens *sequelize*⁵⁵ benutzt wird, so ist es auch möglich das System mit z.B. *sqlite* oder *Postgres* zu benutzen.

Das verwendete Datenmodell kann in ?? betrachtet werden, es ist komplett mittels *sequelize* beschrieben und ist damit ohne viel Aufwand veränderbar.

Die Kommunikation von Backend und Frontend geschieht per HTTP-Requests, das verwendete Framework ist *express*⁵⁶.

⁵³Wie hier zu sehen ist: <http://blog.caustik.com/category/node-js/>

⁵⁴Object Relationship Mapper, hiermit ist es möglich Datenbanken mittels Klassen und Objekten zu abstrahieren. ORM ist jedoch nur eine einfache Abstraktion und beherrscht nicht alles was SQL kann, jedoch reichen die Fähigkeiten des ORM meistens aus und es kann immer auf rohes-SQL zurückgriffen werden.

⁵⁵<http://docs.sequelizejs.com/en/latest/>

⁵⁶<http://expressjs.com/>

Kapitel 4

Fazit und Ausblick

In dieser Arbeit wurde vieles gemacht das vor wenigen Jahren noch nicht möglich gewesen wäre, die gewählten Technologien sind größtenteils jung und die Anwendung war, trotz großer Firmen die dahinter stehen, gewagt. Das Entwickeln der Applikation war damit zu einem gewissen Teil eine Fallstudie für neue Technologien in der Webentwicklung. Rein technologisch betrachten ist die Arbeit jedoch ein voller Erfolg, sie ist performant, ist ästhetisch anspruchsvoll¹ und die unidirektionale Architektur übersichtlich und erweiterbar. Wenn man nach den am Anfang der Arbeit gesetzten Anforderungen geht, erfüllt das Ergebnis diese vollständig.

Doch sind Softwaresysteme schwer nach dem Papier zu beurteilen, gerade wenn es sich um Interface-getriebene Systeme handelt. Das Interface wurde mit bestem Gewissen und viel Liebe zum Detail kreiert, besonders die Darstellung der Suchergebnisse entstand in einem langen Prozess und stellt etwas dar das so vorher noch nicht existiert hat, aber ob diese nun auch eine wirkliche Innovation darstellt muss erst noch herausgefunden werden. Mehr Funktionalität in einem Produkt bedeutet nicht dass es besser ist als das vorherige.

Die Implementierung der Gamification wurde nach aktueller Theorie betrieben und programmtechnisch besitzt sie keine offensichtlichen Mängel, jedoch muss eine wissenschaftliche Analyse betrieben werden um zu zeigen ob diese nun die gewünschten Effekte erzeugt.

Wie jedes Softwaresystem ist auch dieses nicht beendet. Es gibt viele Aspekte wie z.B. die Darstellung der Statistiken, Optimierung der Gamification oder Bereitstellung von neuen Darstellungsformen für die Suchergebnisse die noch behandelt werden können. Allgemein ist das Thema Gamification im Unternehmensbereich um Wissensaustausch zu Fördern ein interessantes Themengebiet das noch viel Potential birgt.

Abschließend sei gesagt, dass ich hoffe, dass meine Arbeit was das Testen von neuen Technologien angeht nicht unbeachtet bleibt und das hier geschriebene Projekt als Beispiel für zukünftige Projekt genommen wird.

¹Dies ist natürlich subjektiv.

Kapitel 5

Appendix

Literatur

- Abramov, Dan (2015):** *The Evolution of Flux Frameworks*. https://medium.com/@dan_abramov/the-evolution-of-flux-frameworks-6c16ad26bb31. [Online; accessed 06-07-2015].
- Boronine, Alexei (2015):** *HUSL Color Space*. <http://www.husl-colors.org/>. [Online; accessed 06-07-2015].
- Deterding, Sebastian u. a. (2011):** „From game design elements to gamefulness: defining gamification“. In: *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*. ACM, S. 9–15.
- Kim, Joseph (2014):** *The Compulsion Loop Explained*. http://www.gamasutra.com/blogs/JosephKim/20140323/213728/The_Compulsion_Loop_Explained.php. [Online, accessed 06-03-2015].
- Meder, Michael, Till Plumbaum und Frank Hopfgartner (2014):** „Daiknow: A gamified enterprise bookmarking system“. In: *Advances in Information Retrieval*. Springer, S. 759–762.
- Schuster, Clemens Maria (2014):** *Social Media Walls und Twitter Walls: Tools und Anbieter*. <http://hofrat.ch/2013/10/social-media-walls-und-twitter-walls-tools-und-anbieter/>. [Online; accessed 06-03-2015].
- Severance, Charles (2012):** „JavaScript: Designing a language in 10 days“. In: *Computer* 2, S. 7–8.
- Tilkov, Stefan und Steve Vinoski (2010):** „Node.js: Using JavaScript to build high-performance network programs“. In: *IEEE Internet Computing* 6, S. 80–83.
- Treisman, Anne (1987):** „Merkmale und Gegenstände in der visuellen Verarbeitung“. In: *Spektrum der Wissenschaft* 1.1987, S. 72–82.
- Zakai, Alon (2011):** „Emscripten: an LLVM-to-JavaScript compiler“. In: *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*. ACM, S. 301–312.
- Zichermann, Gabe und Christopher Cunningham (2011):** *Gamification by design: Implementing game mechanics in web and mobile apps*. ” O'Reilly Media, Inc.”