

TECHNISCHE UNIVERSITÄT BERLIN

BACHELOR THESIS

DAI INFOBOARDENTERPRISE INFOBOARD - INTELLIGENTE VISUALISIERUNG UND AUSTAUSCH VON HETEROGENEN INFORMATIONEN IM UNTERNEHMEN

Supervisor: Prof. Albayrak

Advisors: -

Written by Tom Nick

6. Mai 2015

Abstract

Das DAI-Labor ist ein Innovationsleiter in Sachen 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 Kernherrausforderungen 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. Des Weiteren werden kollaborative Elemente untersucht d.h. Ergebnisse mehrerer Benutzer werden gleichzeitig angezeigt und eine Anzeige vergleichbar mit der *frontpage* von reddit, wo die Suchergebnisse aller Benutzer angezeigt werden. Das Ergebnis ist eine voll funktionsfähige Webapplikation die neben der reinen Visualisierung und Interaktion der Elemente, *Gamification* benutzt um die (kollaborative) Nutzung zu fördern indem Benutzer *belohnt* werden für die Benutzung sowie dafür das andere Benutzer ihre Inhalte *gut* finden.

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, 6. Mai 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
2 Umsetzung	5
2.1 Gamification	9
3 Technische Umsetzung	II
4 Ausblick	20
5 Appendix	21

Kapitel I

Einleitung

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 angenehmens und sehr 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.

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 und auch die Interaktion mit den Kacheln kann das Layout ändern.

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 Datenuellen 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 kann.

Authentifizierungssystem

Das DAI stellt aus gutem Grund ihre Datenquellen nur intern zu Verfügung bzw. nur nach einer Authentifizierung mittels Benutzername/Passwort, somit muss die Applikation in der Lage sein diese Authentifizierung auszuführen und sicherzustellen dass niemals *zuviel* unauthentifizierten Benutzern angezeigt wird.

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 Benutzer-interface für den Benutzer andere wiederum als simple Konfigurationsdatei für verwendete URLs und ähnliches.

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.

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 wäre der größte Unterschied kleinere stylistische 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.

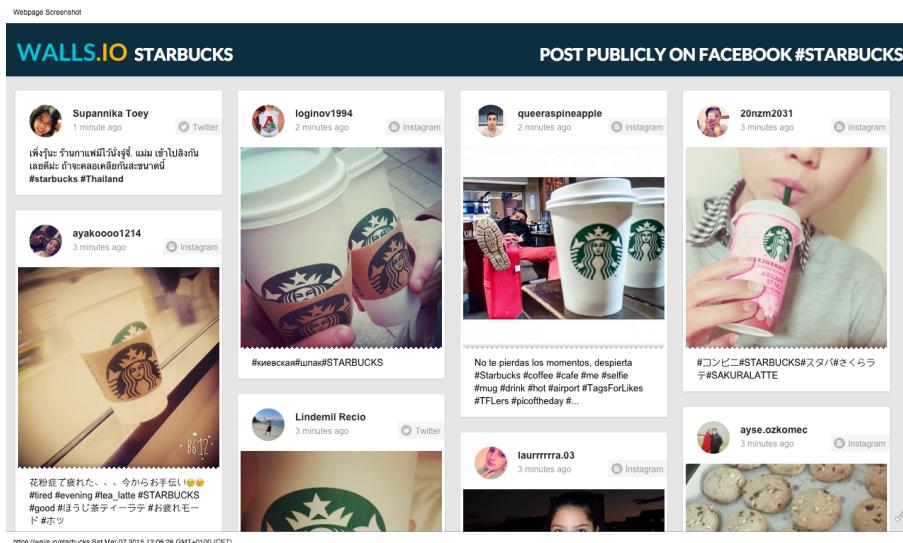


Abbildung 1.1: Eine *social media wall* vom Anbieter walls.io

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

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 erreichte in Videospielen keinen Wert in dem Sinne, dass mit diesem 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 für sich spricht⁷. 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.

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, Trophäen 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 einreichen eines Links oder das jemand anderes seinen Link kopiert hat. Ein Problem das sich auftat bei DAIKnow war dass häufige Benutzer alle verfügbaren Trophäen freischalteten und damit der Gamification-Aspekt damit kein Grund mehr ist die Applikation weiter zu benutzen.

⁶<http://reddit.com>

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

Kapitel 2

Umsetzung

Design

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.

Layout

Wie *social media walls* oder Websites wie [pinterest¹](#) oder gar Microsoft mit ihrem metro-Design es gezeigt haben ist die aktuell beste Darstellung für Medieninhalte verschiedener Art ein Layout basierend auf Kacheln. Bei dem Anordnen gibt es verschiedene Ansätze:

- **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.

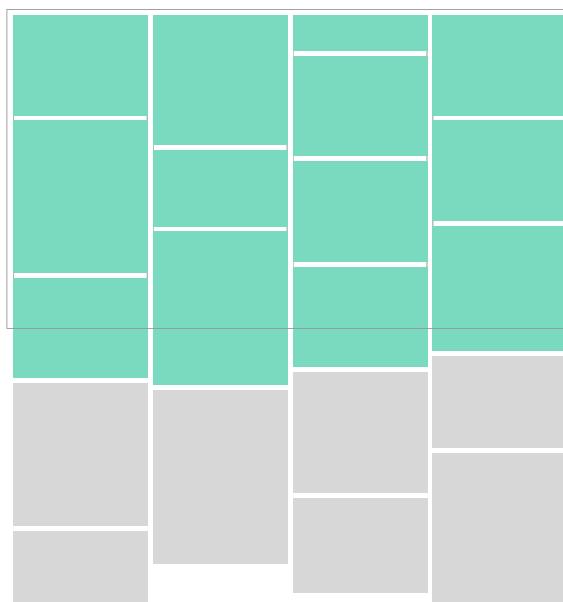


Abbildung 2.1: Spaltenbasierte Anordnung der Inhalte

¹<http://pinterest.com>

Vorteile

Die Vorteile liegen in der Simplizität der Implementierung und der Intuivität des *content-flows* 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 Größteile Nachteil ist die starre des Layouts, da alle Objekte die gleiche Breite haben ist man stark limitiert wie man die Inhalte darstellt.

- **Rasterbasierte Anordnung** Bei dieser Form werden die einzelnen Datenpunkte in einem einheitlichen Raster dargestellet, 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.

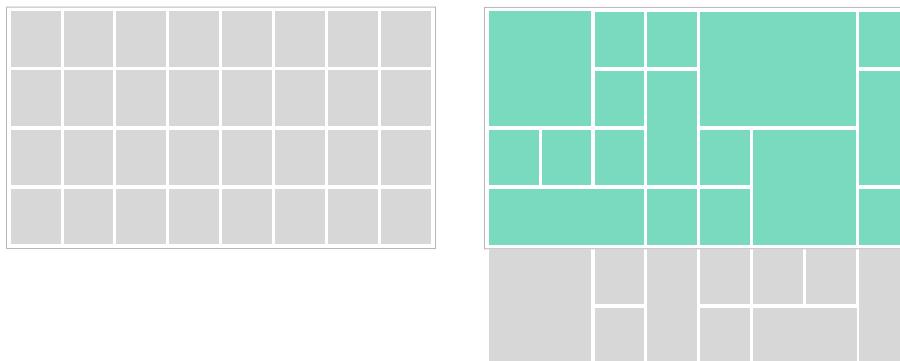


Abbildung 2.2: links: Das Raster des Grids, rechts: eine zufällige Benutzung des Grids

Vorteile

- Ansprechendes Aussehen
- Wichtigkeit kann durch 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 Design für die verschiedenen Kachelgrößen erstellt werden
- unterschiedliche Darstellungsformen können unübersichtlich wirken

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 Behälterproblem² gelöst werden und im zweiten Schritt müssten die Elemente mittels 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 einfache 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 wäre³.

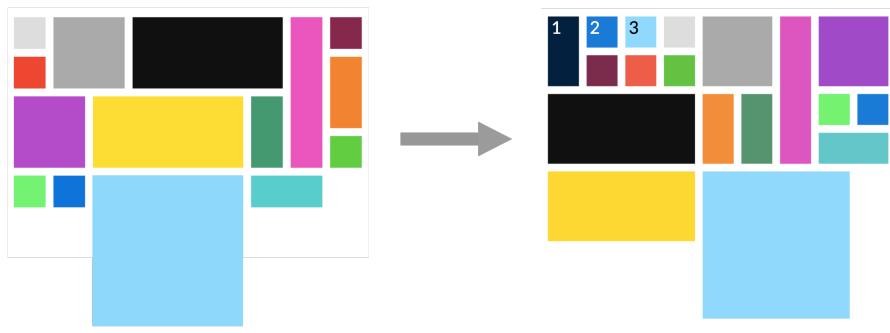


Abbildung 2.3: Zu dem Grid links wurden die Elemente 1, 2 und 3 hinzugefügt. Das resultierende neue Layout ist rechts zu sehen. Dies ist ein praktisches Beispiel welches mithilfe von packery gemacht wurde.

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 kommen⁴ und nicht überraschend irgendwo auftauchen.⁵
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 der *reflow* bei dem Kachelbasierten zu aufwändig ist, als dass nachverfolgt werden kann wie die Kacheln sich bewegen und welchen Gesetzmäßigkeiten sie unterliegen.

²

³<http://packery.metafizzy.co/methods.html#prepended>

⁴Siehe z.B. Börsenticker, Nachrichtenseiten oder die Abfahrttafeln am Bahnhöfen

⁵Vergleich zu Googles Material Design bei dem sehr viel Wert darauf gelegt wird sinnvolle Animationen zu machen, die den Benutzer unterstützen und nicht verwirren

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.
- 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.

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 konzipiert werden.

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

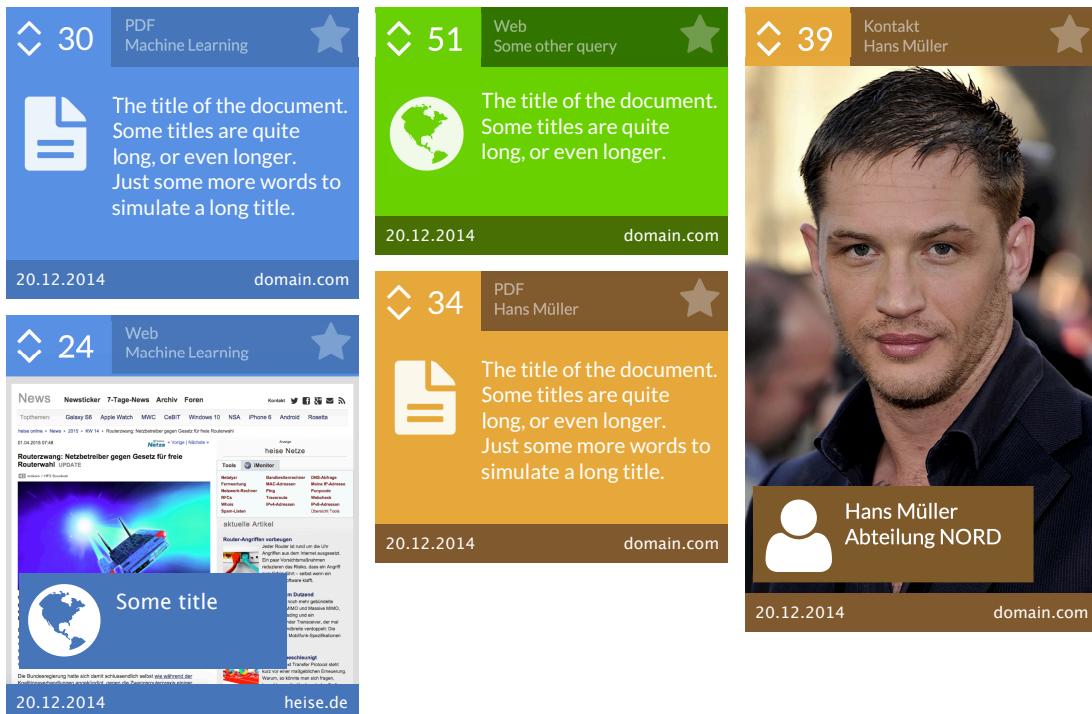


Abbildung 2.5: Ein erstes Design dass verschiedene Darstellungsformen für unterschiedliche Typen von Informationen benutzt.

2.1 Gamification

Wie sehr muss ich Gamification erklären?

Wie in der Einleitung erwähnt ist Gamification eine mittlerweile weitverbreitete und äußerst effektive Technik um Benutzer zu binden. Es ist schwer neue Technologien im Unternehmensfeld einzuführen, vor allem wenn sie freiwillig benutzt werden können - so wie das hier entwickelte Enterprise Dashboard. Ziel ist es also durch die Gamification der Benutzung die Benutzer dazu zu motivieren die Anwendung gewohnheitsmäßig zu benutzen.

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

Diese modellierte Kette besteht aus 3 Schritten:

- I. 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 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.

Das Enterprise Dashboard besitzt wenige Aktionen, sodass eigentlich alle Teil der Aktionen sein können die eine Belohnung geben.

- 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.

Kapitel 3

Technische Umsetzung

Im Kapitel Umsetzung wurde die Applikation soweit geplant, dass die Technische Umsetzung erfolgen kann - d.h. das geplante System wird soweit realisiert das es benutzt werden kann. Aspekte warum eine Technologie genutzt wurde und nicht eine andere, wie die Architektur der Applikation ist und warum sie so ist und alles was sonst noch anfällt wenn man eine Applikation schreibt. Die Technische Umsetzung wird sich fast ausschließlich mit dem Frontend beschäftigen da dies Teil der Bachelorarbeit war. Das Backend wird nur kurz wegen der Vollständigkeit behandelt.

Wahl der Technologie

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 workarounds oder bugfixes 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 zukunftsicher
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.

Eine sehr gepflegte Liste der derzeit verfügbaren Technologien im Bereich Frontend ist verfügbar auf [github¹](https://github.com/dypsi/foreground-dev-bookmarks).

Für das Frontend ist erstmal die Frage zu klären ob ein Framework a la AngularJS oder EmberJS verwendet werden soll. Dagegen spricht wie oben schon genannte die initiale Lernkurve, was es schwer machen könnte das System einfach wartbar zu machen, auch stellen Frameworks oft bei Applikationen die etwas ausgefeilteres machen als die

¹<https://github.com/dypsi/foreground-dev-bookmarks>

breite Masse ein Problem da, da die Abstraktion des Frameworks oft etwas verbirgt was benötigt wird.

Ein großer Faktor der in anderen Umgebungen lange Zeit keine Rolle mehr spielt ist die Größe des Quelltextes der verwendeten Bibliotheken/Frameworks. Bevor bei Javascript irgendeine Bibliothek benutzt werden kann muss sie zuerst beim Benutzer heruntergeladen und ausgeführt werden, was bei langsamem Computern mit nicht perfekter Internetverbindung ein enormer Bestandteil ist. Um den Punkt nochmal zu verdeutlichen wurde ein kleiner Test gemacht. Dazu wurde eine Ember-Demoapplikation ausgeführt² die sehr wenig *business-logic* besitzt, sodass sehr gut zu zeigen ist wieviel das herunterladen und ausführen nur der Bibliotheken ausmachen. Die Anwendung ist auf github zu finden³.

Name	Met...	Status	Type	Initi...	Size	Time	Timeline
unstyled/	GET	200	text...	Other	3.4 KB	76 ...	
bootstrap.min.css	GET	200	text...	jkne...	20.6 ...	77 ...	
createUsersInLocalStorage.js	GET	200	appl...	jkne...	1.2 KB	52 ...	
jquery-2.0.3.min.js	GET	200	appl...	jkne...	33.7 ...	79 ...	
moment.min.js	GET	200	appl...	jkne...	6.9 KB	50 ...	
handlebars-1.0.0.js	GET	200	appl...	jkne...	71.4 ...	52...	
ember.js	GET	200	text...	jkne...	101 ...	1.5...	
ember-data.js	GET	200	text...	jkne...	210 KB	71...	
localStorage_adapter.js	GET	200	appl...	jkne...	1.6 KB	50 ...	

9 requests | 1.3 MB transferred | 1.86 s (load: 2.32 s, DOMContentLoaded: 2.32 s)

Abbildung 3.1: Übertragungszeit der benötigten Dateien

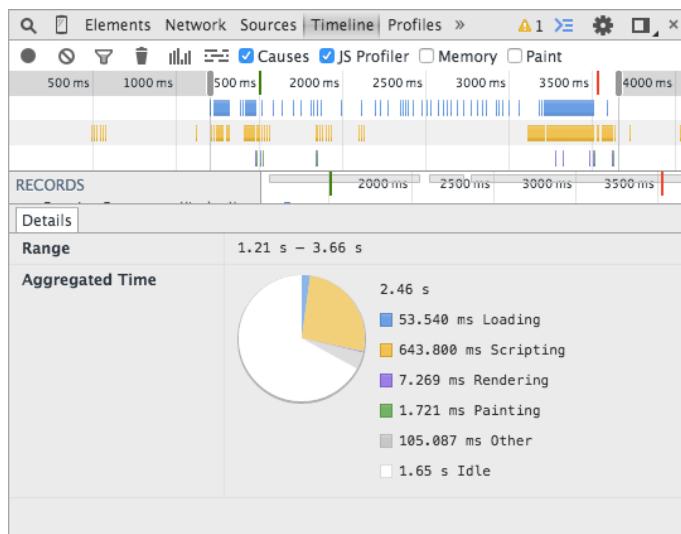


Abbildung 3.2: Benötigte Ausführungszeit der Skriptdatein

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.

²Auf einem Macbook Air mid 2013 in minimal Konfiguration und dem Chrome Browser in Version 41, die vorhandene Internetverbindung hatte eine Übertragungsraten von $50000 \frac{\text{bit}}{\text{s}}$

³<http://jkneb.github.io/ember-crud/unstyled/#/users>

Facebook React und die Flux Architektur

Facebooks React⁴ ist eine JavaScript-Bibliothek die seit ihrem erscheinen im Jahr 2013 viel Beifall und viele Anhänger finden konnte. Zum Zeitpunkt dieser Arbeit wird sie produktiv von vielen Firmen benutzt wie *Khan Academy*⁵, *Netflix*⁶, *Yahoo!*⁷, *Sony*⁸ 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*⁹ 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.

Virtual DOM

Wie weit muss ich hier den DOM beschreiben? Muss ich ihn beschreiben?

Einer der innovativsten Aspekte von React ist die virtuelle Repräsentation des DOMs¹⁰. 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)

⁴<https://github.com/facebook/react>

⁵<http://joelburget.com/backbone-to-react/>

⁶<http://conf.reactjs.com/schedule.html#beyond-the-dom-how-netflix-plans-to-enhance-your-television>

⁷<http://www.slideshare.net/mobile/rmsguhan/react-meetup-mailonreact>

⁸<https://medium.com/code-stories/dev-chats-spike-brehm-of-airbnb-87e155f3475d>

⁹https://developer.mozilla.org/en-US/docs/Web/Web_Components

¹⁰http://de.wikipedia.org/wiki/Document_Object_Model

besitzt, wobei n die Anzahl der veränderbaren Inhalte ist. Der von Facebook verwendete Algorithmus besitzt eine Komplexität von $O(n)^n$ wobei n die Anzahl der DOM-Element ist. Ein Geschwindigkeitsvergleich von Frameworks die einen virtuellen DOM benutzen zeigt diese Grafik vom Elm¹²-Entwickler bei der die verschiedenen Frameworks anhand ihrer TodoMVC¹³-Implementierung verglichen werden.

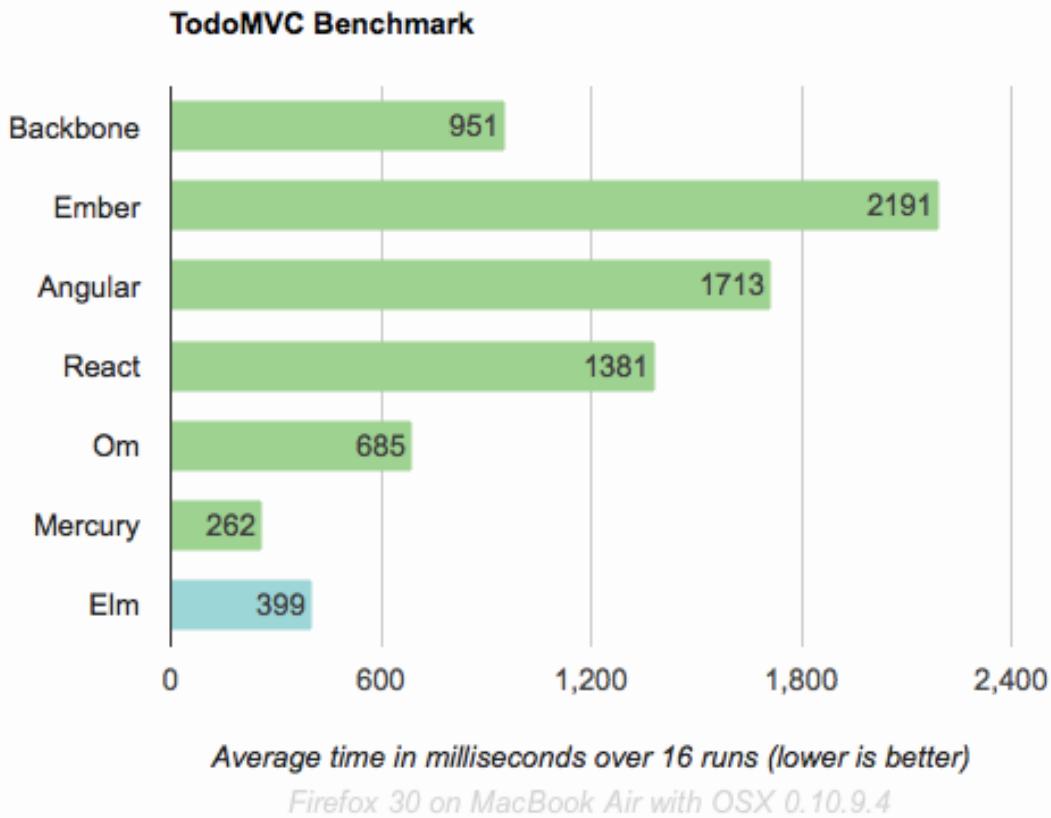


Abbildung 3.3: Geschwindigkeitsvergleich von TodoMVC-Implentierungen mit verschiedenen Frameworks.

Es ist zu sehen, dass Ember und AngularJS am langsamsten sind. Backbone ist in diesem Benchmark schneller als React, wobei die verwendete Implemtierung der React-App nicht optimiert ist - *Om*¹⁴ kann als optimierte React-Version betrachtet werden, da Om im Grunde React mit nicht veränderbaren Datenstrukturen¹⁵ (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.

¹¹<https://facebook.github.io/react/docs/reconciliation.html>

¹²http://en.wikipedia.org/wiki/Elm_programming_language

¹³Der TodoMVC ist ein Versuch die schiere Anzahl an Webframeworks anhand eines realen Beispiels zu vergleichen. <http://todomvc.com/>

¹⁴<https://github.com/omcljs/om>

¹⁵Durch nicht veränderbare Datenstrukturen kann das diffing noch effizienter gestaltet werden, da viele Teilbäume nicht betrachtet werden müssen.

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 rendern¹⁶ oder ganz einfach in einen String. Durch das simple 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.

Flux Architektur

Flux¹⁷ 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 Programmierung¹⁸, bei welchem ein Programm als gerichteter Graph modelliert wird. Typisch für das Paradigma ist die Benutzung eines Beobachter-Entwurfsmuster¹⁹ wie z.B. eines pub/sub-System²⁰. Bei Flux wird für letzteres eine Abwandlung eines pub/sub-Systems benutzt.

Flux besitzt drei große Bestandteile: *dispatcher*, *stores*, *views (React components)*. Des Weiteren 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.

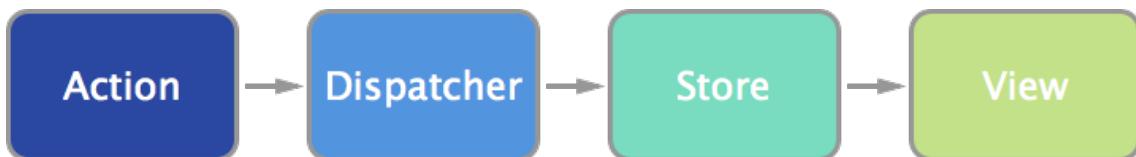


Abbildung 3.4: Dispatcher, Stores und Wiews sind unabhängige Knoten mit unterschiedlichen Ein und Ausgaben. Die Aktionen sind einfache Objekte die die neuen Daten enthalten und den Typ der Daten.

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* benutzt werden um neue

¹⁶<https://github.com/Flipboard/react-canvas>

¹⁷<https://facebook.github.io/flux/docs/overview.html>

¹⁸http://en.wikipedia.org/wiki/Dataflow_programming

¹⁹http://en.wikipedia.org/wiki/Observer_pattern

²⁰Eine kleine Auflistung und eine Bewertung typischer Muster ist hier zu finden <https://github.com/millermedeiros/js-signals/wiki/Comparison-between-different-Observer-Pattern-implementations>

Daten an den Dispatcher zu schicken, welcher diese dann an den Store weiterleitet - welche dann die Darstellung der *Component* beeinflussen kann.

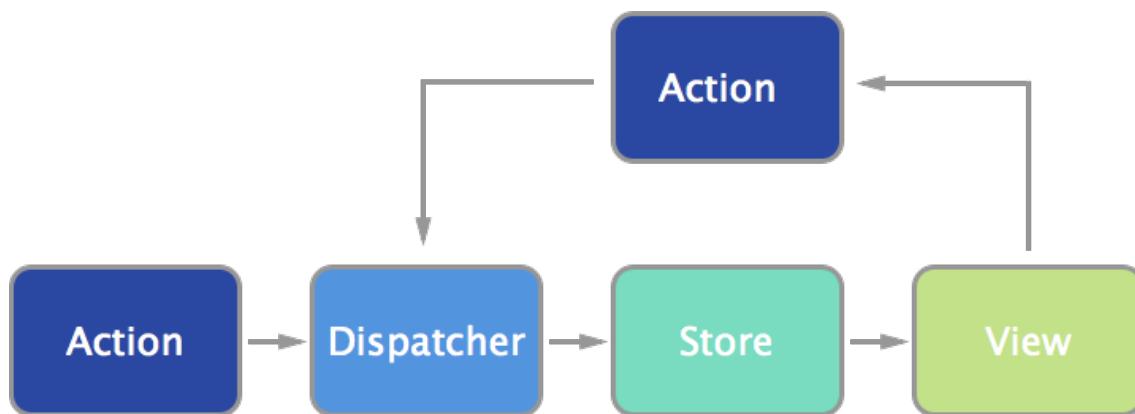


Abbildung 3.5: Mithilfe von *actions* ist die View in der Lage den Zustand der Applikation zu verändern.

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-Wall eine gute Wahl erscheint.

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 finden²¹. *reflux* ist nicht der einzige Versuch eine alternative bereitzustellen, mittlerweile gibt es über 10 verschiedene Flux-Bibliotheken²² - mit teilweise großen Communities dahinter oder Firmen wie z.B. *Flexible* 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.

²¹<http://spoke.ghost.io/deconstructing-reactjs-flux/>

²²Ein Vergleich dieser ist hier zu finden <https://github.com/voronianski/flux-comparison>

Überblick

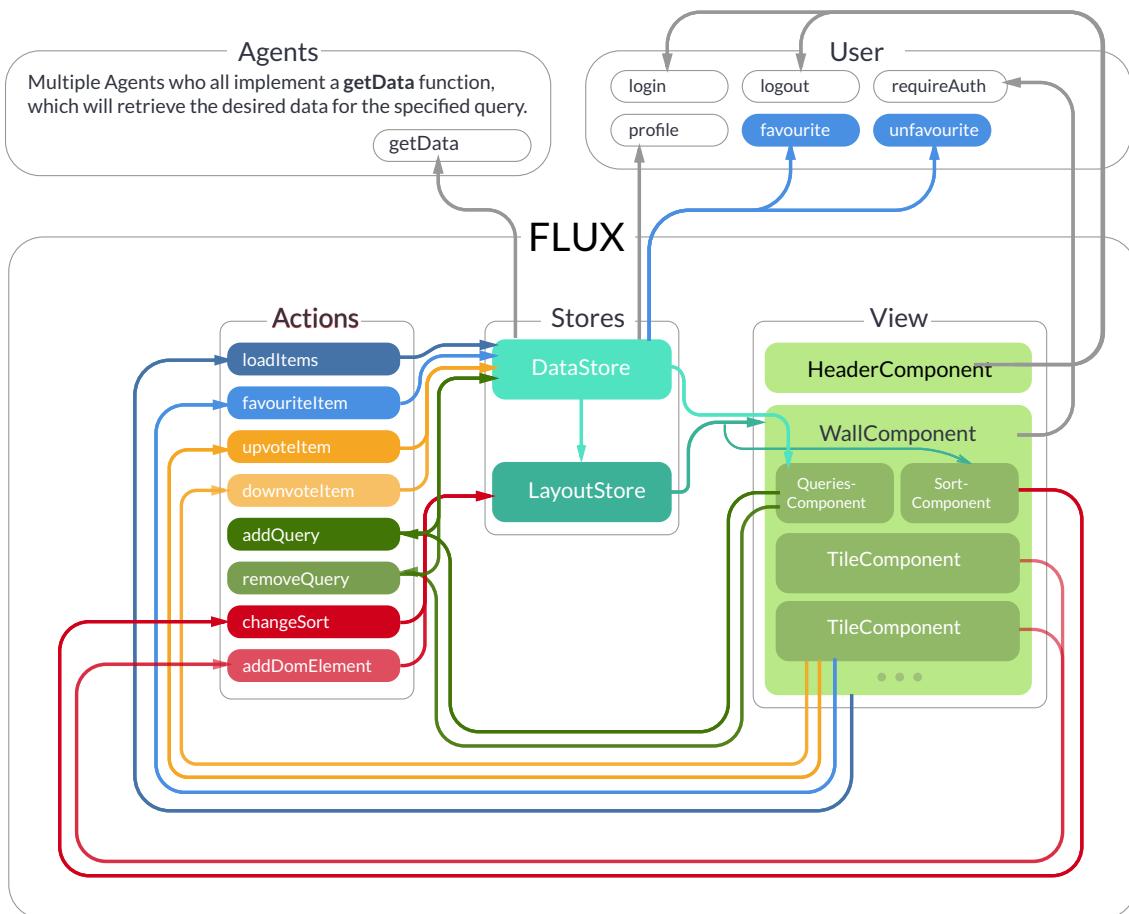


Abbildung 3.6: Die wichtigsten Bestandteile der Applikation.

Die Architektur wurde bewusst nicht in einem Klassendiagramm dargestellt da die Applikation keinen strengen objektorientierten Stil verfolgt und das Schaubild nicht sonderlich zum Verständnis beitragen würde. Es ist auch kein komplettes Schaubild der Applikation, jegliche Aspekte die zum Verständnis nicht sonderlich beitragen würden wurden rausgelassen.

Wie zu sehen kann die Applikation in drei Teile eingeteilt werden.

1. Agenten

Die Agenten stellen die Kommunikation mit den jeweiligen Brokern dar, sie implementieren alle eine Funktion `getData` die eine Suchanfrage nimmt und eine einheitliche Datenstruktur mit den Ergebnissen zurückliefert. Broker die eine Authentifizierung benötigen liefern einen Fehler sobald so Benutzer-Objekt mit gültigen Token benutzt werden.

2. **User - Authentifizierung** Der User beinhaltet neben seines Profils in welchem Favoriten, Punktestand und bewertete Suchergebnisse gespeichert werden die komplette Authentifizierung im System. Im aktuellen Zustand wird jeder Nutzer gezwungen sich zu authentifizieren um das System zu nutzen.

-
3. **Flux - Actions, Stores und View** Dies ist der wichtigste, komplizierteste und größte Teil der Applikation. Hier geschieht alles was mit dem Darstellen und der Interaktion mit den Inhalten zu tun hat. In den Stores werden die Daten der Agenten soweit verarbeitet dass sie an die View weitergegeben können wo sie als interaktive Kacheln dargestellt werden. Interaktionen geschehen über die Aktionen. Es ist gut zu sehen dass der Datenfluss Unidirektionell ist, wodurch das urteilen über Applikation vereinfacht wird.

Im folgenden werde ich genauer auf den Flux-Teil eingehen.

Stores - Data

Der DataStore stellt alle Daten bereit die für die Darstellung benötigt werden, dies sind die Suchergebnisse die mit dem Profil des Benutzers gemixt 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 auszuschließen, Anfragen an die Broker zu minimieren und die Interaktion mit den Suchergebnissen.

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 vorgebeugt werden da nur der DataStore selbst die Suchergebnisse verändern kann.

Stores - Layout

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 ist²³. Was weiterhin zu beachten ist, dass am Ende immer gerundete Pixelwerte innerhalb von TRANSLATE3D(x, y, z) benutzt werden sollten. Da Ansonsten die Elemente unscharf sind.

Es ist eine Grundanforderung Geräte mit unterschiedlich Großen Bildschirmen zu unterstützen, was dadurch erzielt wird, dass die 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 Bildschrimbreiten beschrieben werden.

Für die Positionierung müssen wir wissen wieviel 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.

Backend

Im Verlaufe der Arbeit wurde entschieden, dass eine weitere Server-Applikation ungünstig wäre und die Arbeiten im Frontend ausreichend waren für den Rahmen einer Bachelor-Arbeit. Zur Vollständigkeit halber werden die verwendeten Technologien aufgezählt.

²³<http://caniuse.com/#feat=transforms3d>

Kapitel 4

Ausblick

Kapitel 5

Appendix

Literatur

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 06-03-2014].

Treisman, Anne (1987): „Merkmale und Gegenstände in der visuellen Verarbeitung“. In: *Spektrum der Wissenschaft* 1.1987, S. 72–82.