

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

Geschrieben von Tom Nick  
14. August 2015

---

### Abstrakt

Das DAI-Labor (Distributed Artificial Intelligence) ist eine der führenden Forschungseinrichtungen im Bereich verteilte Suchmaschinen - welche nun nicht mehr reine Forschung sind, sondern wie deren PIA-System (Personal Information Assistant) von zahlreichen Verwaltungen als Alternative zu privaten Suchmaschinen benutzt wird. Eine interessante Aufgabe besteht darin, die Suchergebnisse von verschiedenen Suchmaschinen zusammenzuführen und für den Suchenden in einem visuell einheitlichen Format darzustellen. In dieser Arbeit wird eine Visualisierung in der Art einer *Social Media Wall* untersucht und realisiert. Ein weiteres Themengebiet des DAI-Labors ist die Gamification im Enterprise Bereich. Auch diese Mechanismen werden untersucht und benutzt, um die Benutzung der Applikation zu fördern. Ausführlich wird auf die neuen Entwicklungsmethoden im Bereich der Webapplikationen eingegangen und aufgezeigt, wie diese die Entwicklung der Applikation vereinfachen und verbessern können. Das Ergebnis dieser Arbeit ist eine funktionsfähige und mit neuesten Methoden entwickelte Webapplikation, die ein neuartiges Interface für Suchergebnisse unterschiedlicher Herkunft besitzt und deren Benutzung durch nahtlos eingebundene Spielelemente gefördert wird.

# 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.<sup>1</sup>

Berlin, 14. August 2015

Place, date

Signature

---

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

# Inhaltsverzeichnis

Eidesstattliche Erklärung	i
<b>1 Einleitung</b>	<b>1</b>
1.1 Beschreibung des Infoboards . . . . .	2
1.2 Anforderungen . . . . .	2
1.3 Relevante Arbeiten . . . . .	3
1.3.1 Social Media Walls . . . . .	3
1.3.2 Gamification . . . . .	3
<b>2 Konzeptuelle Umsetzung</b>	<b>6</b>
2.1 Visualisierung der Informationen . . . . .	6
2.1.1 Layout . . . . .	6
2.1.2 Probleme beim Flussverhalten . . . . .	9
2.1.3 Animationen . . . . .	9
2.1.4 Kacheln . . . . .	10
2.1.5 Eindeutige Zuordnung von Farben zu Themen . . . . .	12
2.2 Gamification . . . . .	13
2.2.1 Aktionen und deren Belohnung . . . . .	14
2.2.2 Anerkennung der Leistung . . . . .	14
2.2.3 Abzeichen . . . . .	16
<b>3 Technische Umsetzung</b>	<b>20</b>
3.1 Wahl der Technologie . . . . .	20
3.1.1 Programmiersprache . . . . .	20
3.1.2 Ladezeiten . . . . .	22
3.1.3 Frameworks und Bibliotheken . . . . .	23
3.1.4 Tooling . . . . .	24
3.1.5 Facebook React und die Flux Architektur . . . . .	25
3.2 Architektur der Applikation . . . . .	29
3.2.1 Überblick . . . . .	29
3.2.2 Stores . . . . .	31
3.2.3 Views . . . . .	33
3.3 Backend . . . . .	35
3.3.1 HTTP-API . . . . .	35
3.3.2 Websocket-API . . . . .	36

## **INHALTSVERZEICHNIS**

---

<b>4 Fazit und Ausblick</b>	<b>37</b>
<b>Literaturverzeichnis</b>	<b>iv</b>
<b>Abbildungsverzeichnis</b>	<b>x</b>
<b>Abkürzungsverzeichnis</b>	<b>xii</b>
<b>Anlagen – Inhalte der beiliegenden CD</b>	<b>xiii</b>
<b>Appendix</b>	<b>xiii</b>
Abbildungen . . . . .	xiv
Beispiel einer interaktiven Komponente . . . . .	xiv

# Kapitel 1

## Einleitung

Die gemeinsame Darstellung von Inhalten aus verschiedenen Quellen ist ein interessantes und weit erforschtes Thema. Die Idee ist, dem Benutzer die Benutzung bzw. die Konsumierung von Inhalten angenehmer und effizienter zu gestalten, indem ein einheitliches Interface für Inhalte geschaffen wird. Vor der *digitalen Revolution* war das ein äußerst aufwendiges und kostspieliges Unterfangen<sup>1</sup>. Als Beispiel seien die frühen Enzyklopädien genannt, welche versuchten, sämtliches Wissen an einem Ort in homogener Sprache und Darstellung bereitzustellen. Durch die momentan verfügbaren Technologien ist das Erstellen einer einheitlichen Datenquelle für verschiedene Inhalte deutlich einfacher geworden. Die Schwierigkeit einer qualifizierten Zusammenführung besteht dennoch weiter. Produkte die diese Aufgabe entsprechend lösen sind sehr erfolgreich. Wie zum Beispiel Facebook mit der Aggregation von sozialen Aktivitäten.

Wenn man sich jedoch auf eine Teilmenge der verfügbaren Informationen beschränkt, wie Nachrichten die als RSS (Rich Site Summary) verfügbar sind, ist es deutlich einfacher eine angemessene Zusammenstellung der Inhalte zu gestalten. Prominentes Beispiel für eine Anwendung dieser Art ist die Nachrichten-Applikation pulse [2], die auf Basis von RSS ein angenehmes und attraktives Interface für die gleichzeitige Konsumierung mehrerer Nachrichtenseiten anbietet. Der Erfolg von *pulse* spricht mit ca. 22 Millionen Downloads auf dem Android-System für sich.

Eine der komplexesten Zusammenführungen von Informationen unterschiedlichster Art stellen Suchmaschinen da, da hier eine Reduzierung der benutzten Informationen, eine sofortige Verschlechterung der Suchergebnisse bewirkt - die ideale Suchmaschine würde jede mögliche Datenquelle und jedes existierende Dateiformat indizieren. Suchmaschinen der Gegenwart versuchen sich diesem Optimum immer weiter zu nähern und können mittlerweile eine Vielzahl an Datenformaten und Datenquellen nutzen und verarbeiten. Trotz der beinahe unendlichen Diversität der indizierten Daten werden die Suchergebnisse meistens über eine einfache Auflistung in Textform dargestellt, weil jedes Dateiformat mehr oder minder auf Text reduziert werden kann.

Neue Visualisierungsformen für Suchergebnisse sind bisher überraschenderweise wenig untersucht worden. Andere Bereiche sind auf diesem Gebiet experimentierfreudiger, wie zum Beispiel die *Social Media Walls* mit der Zusammenführung von Social-Media-Kanälen. Sie sind eine Form der Darstellung für Inhalte mit kurzen Texten und Bildern aus verschiedenen sozialen Kanälen. Die Besonderheit liegt in der Homogenisierung der Inhalte und in der gleichzeitigen Darstellung von verschiedenen Kanälen. Suchergebnisse besitzen ganz ähnliche Attribute wie die Datenpunkte von Social-Media-Kanälen, womit die Frage nahe liegt, ob die Visualisierung von Suchergebnissen verbessert werden kann, indem eine ähnliche Darstellung wie der von *Social Media Walls* benutzt wird.

---

<sup>1</sup>Vergleich hier zu der Encyclopædia Britannica und Wikipedia, letztere konnte in wenigen Jahren mehr Inhalte mit gleicher Qualität aggregieren, also das Traditionssunternehmen in 200 Jahren [1].

Diese Arbeit bemüht sich diese Frage zu beantworten und eine neue Visualisierungsform für Suchergebnisse zu finden mit der Inspiration von *Social Media Walls*.

Der Einsatz von Gamification hat in den letzten Jahren stetig zugenommen, da es ein effektives Mittel ist, um das Erlebnis des Nutzers zu erhöhen und seine Interaktionen mit der Applikation zu fördern. Die Benutzung von Gamification im Enterprise-Bereich bzw. in kleinen sozialen Kreisen ist bisher noch nicht allzu populär. Die prominentesten Beispiele sind Sport-Apps, wie Strava [3] oder Nike+ [4], bei denen Freunde sich miteinander vergleichen können. Eine Integration von Gamification stellt damit in diesem Bereich eine interessante Herausforderung dar, da deren Potential bisher noch nicht vollständig geklärt ist.

## 1.1 Beschreibung des Infoboards

Das Infoboard ermöglicht die Abrufung von Informationen aus heterogenen Unternehmensquellen. Hierzu kann der Benutzer nach Informationen mittels verschiedener Themen suchen. Es können jederzeit Themen entfernt oder hinzugefügt werden. Die gleichzeitige Anzeige mehrerer Themen hat die Motivation, dass der Benutzer bei der Suche von Themenübergreifenden Informationen diese schneller und komfortabler finden kann. Die Informationen können binär bewertet werden, womit sie global in der Relevanz steigen. Durch die gezielte Benutzung von Gamification-Elementen wird versucht die Anzahl an Bewertungen durch den Benutzer zu erhöhen. Durch die Bewertungen soll die Qualität der Relevanz der Information verbessert werden, womit eine Verbesserung des unternehmensinternen Informationsaustausches angestrebt wird.

## 1.2 Anforderungen

Im Folgenden werden die Anforderungen zur Entwicklung des Infoboards aufgelistet. Diese waren jedoch anfangs größtenteils noch offen. Die genauen Spezifikationen haben sich erst während der Ausarbeitung ergeben.

**Darstellung von Daten** Eine der Hauptanforderungen ist die Darstellung von heterogenen Informationen (im zukünftigen Verlauf des Textes wird sie als Kacheln bezeichnet) im Stil einer *Social Media Wall*. Daraus folgen weitere Ansprüche, zum Beispiel dass es unterschiedliche Kacheln gibt - also für verschiedene Informationen verschiedene Darstellungen. Kacheln für Informationen wie Kontakte und Webseiten können ganz unterschiedlich aufgebaut sein. Die dargestellten Inhalte sind weiterhin dynamisch, während des Betriebes können Kacheln hinzukommen, entfernt oder anderen Themen zugeordnet werden. Die Kacheln bieten zudem Interaktionsmöglichkeiten, da sie bewertet und favorisiert werden können.

**Bewertungssystem** Der Benutzer soll in der Lage sein, angezeigte Suchergebnisse zu bewerten und dadurch die Priorisierung von Suchergebnissen zukünftiger Anfragen für alle Benutzer zu verändern.

**Benutzung mehrerer heterogener Datenquellen** Die angezeigten Informationen können aus mehreren Datenquellen kommen, das heißt die gleichzeitige Darstellung von Informationen von zum Beispiel Facebook mit denen von Twitter. Auch hier ist wichtig, dass zusätzliche Datenquellen hinzufügt werden können.

**Authentifizierungssystem** Zur Benutzung der Gamification-Elemente und der Suchanfrage bedarf es einer Authentifizierung. Die Suchergebnisse können unternehmensinterne Dokumente enthalten, deswegen 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* erstellt werden, das heißt eine Webapplikation bei der JavaScript abseits des initialen Seitenaufrufes alles rendernt. Im Gegensatz zur klassischen Methodik, bei der jede Interaktion in einer komplett neuen Seite, die vom Server gerendert wird, resultiert [5].

**Gamification** Um zur Benutzung der Seite zu animieren sollen Gamification-Methoden eingeführt werden wie zum Beispiel Punkte für das tägliche einloggen bis hin zu erweiterten Methoden wie der Erstellung eines Compulsion Loop. “Der Compulsion Loop ist eine modellierte Kette an Aktivitäten, die gewohnheitsmäßig wiederholt werden, um eine neurochemische Belohnung zu erhalten” ([6], übersetzt aus dem Englischen).

**Konfiguration** Es muss möglich sein eine Vielzahl an Parameter anzupassen, einige davon dienen auch als Benutzeroberfläche für den Benutzer andere wiederum als simple Konfigurationsdatei für verwendete URLs und ähnliches.

## 1.3 Relevante Arbeiten

Da es Bestandteil dieser Arbeit ist, Erkenntnisse aus verschiedenen Gebieten zu Aggregieren, gibt es keine direkt zu Grunde liegenden Arbeiten. Vielmehr werden Erkenntnisse verschiedener Bereiche untersucht und zusammengeführt.

### 1.3.1 Social Media Walls

Im folgenden werden verschiedene Angebote von *Social Media Walls* untersucht um Unterschiede und Gemeinsamkeiten dieser festzustellen.

Eine sehr gute Auflistung verschiedener *Social Media Wall*-Applikationen bietet [7]. Die Homogenität dieser Angebote ist jedoch erstaunlich. Würde man die Produkte zweier verschiedener Anbieter nebeneinander sehen, wären die größten Unterschiede stilistischer Natur wie etwa die benutzten Farbschemata und Schriftarten. Leider sind die wenigsten Produkte in einer Demo zu testen, da die Lösungen üblicherweise auf den Kunden zugeschnitten werden. Primärer Use-Case sind große Veranstaltungen wie Konzerte, weswegen auch keine im Produktiveinsatz eingesehen werden konnten. In Abbildung 1.1 kann eine der Implementierungen gesehen werden.

Da die dargebotenen Angebote sehr ähnlich sind, wird in dieser Arbeit nicht weiter auf alle Anbieter eingegangen. Die Darstellungsform ist aber dennoch weiterhin interessant und wird als grobe Inspiration die Darstellung der Suchergebnisse dienen.

### 1.3.2 Gamification

Neben der Visualisierung von Informationen aus heterogenen Unternehmensquellen, ist Gamification der zweite große Aspekt der Applikation. Wie in der Einleitung erwähnt,

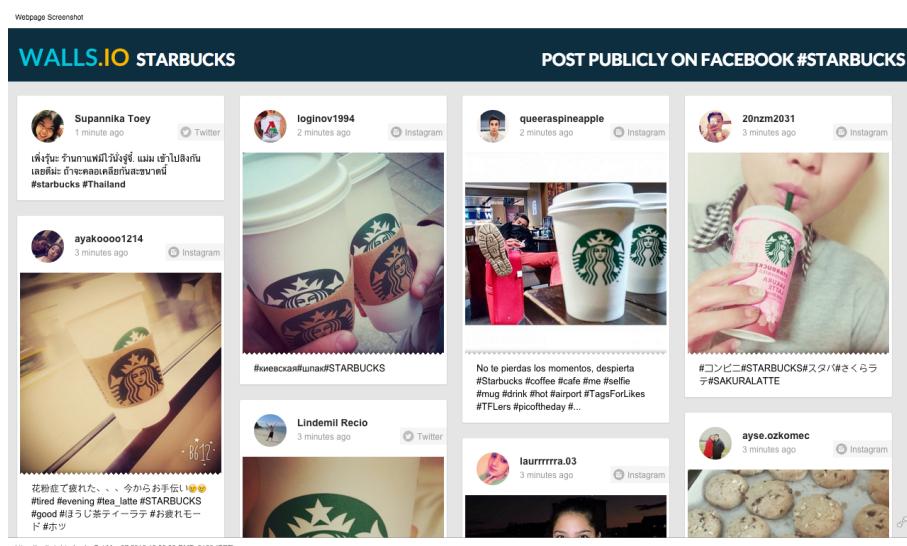


Abbildung 1.1: Eine *Social Media Wall* vom Anbieter walls.io [8]

ist Gamification eine mittlerweile weitverbreitete und äußerst effektive Technik, um das sogenannte *User-Engagement* zu erhöhen. Allein durch Gamification konnte die Seite DevHub ihre *Engagement*-Rate um 20% steigern [9, pp. 17]. Zahlreiche StartUps feiern große Erfolge mit dem Hinzufügen von Gamification zu anstrengenden oder langweiligen Aufgaben, wie das Lernen einer Sprache (Duolingo<sup>2</sup>) oder Programmieren (Codecademy<sup>3</sup>).

Der Begriff Gamification entstand erst 2008 und hat seitdem viele Bedeutungen erlangt. Deterding [10] bemühte 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” [10].

Das Infoboard dient als Interface für den Benutzer die heterogenen Unternehmensquellen komfortabel zu benutzen, welches zu besserem Wissensaustausch führen soll. Dies ist offensichtlich ein nicht-Spiele-Kontext der durch erhöhte Benutzerbeteiligung 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 bzw. ein konstanteres *User-Engagement* erzielt werden, andererseits sollen die Mechaniken zur Bewertung der Suchergebnisse gefördert werden, wodurch diese noch besser werden.

Im Folgenden werden bisherige Arbeiten analysiert, die ähnliche Inhalte anzeigen, und Gamification implementiert haben.

Einen schwachen aber effektiven Ansatz verfolgt die Social-News-Seite reddit [11]. Jegliche Inhalte (eingereichte Links, Kommentare) der Seite können binär bewertet werden (im reddit-jargon wird hier von *upvotes* und *downvotes* gesprochen). Reicht man nun selbst etwas ein und andere Benutzer bewerten den eingereichten Inhalt, wird die Differenz aus *upvotes* und *downvotes* einem als Karma gutgeschrieben. Wie in klassischen Spielen hat Karma einen endogenen Wert [12, pp. 21 - 22], das heißt der Wert besitzt keinerlei Wert außerhalb des Systems, in diesem Falle außerhalb von reddit.

Eine Analyse reddis im Spielekontext führte [13] durch und zeigte auf, dass viele Muster reddis Teil eines Spiels aufbauend auf Karma sind, die die Nutzung und Nutzer reddis

<sup>2</sup><https://de.duolingo.com/>

<sup>3</sup><http://www.codecademy.com/>

weitreichend formten. Dies zeigt, dass Gamification ein großer Teil reddis, und damit auch dessen Erfolgs ist. Laut der Webanalyseplattform Alexa ist reddit die 32 meist besuchte Webseite der Welt, in den USA ist sie sogar auf Platz 10 [14]. Doch besitzt die Gamification reddis auch negative Seiten, so legt [15] dar, das eine Anhäufung Karmas zum Zweck der Selbstbestätigung der eigenen Meinung betrieben wird. Benutzer reddis werden darauf gepolt ihre eingereichten Beiträge anhand der Höhe des Karmas, welches der Beitrag voraussichtlich liefern wird, auszusuchen - und nicht anhand Qualität oder Relevanz.

DAIKnow [16] ist eine Bookmarking Seite wie auch delicious.com [17], bei der Links mit Beschreibungen und Keywords eingereicht werden können. Durch den gezielten Einsatz von Punkten, Abzeichen und Bestenlisten wird die Benutzung der Seite gesteigert. Im Gegensatz zu reddit ist dieses System jedoch weitaus komplexer, so bekommt der Benutzer unter anderem Punkte für den täglichen Aufruf, Punkte für das Einrechen eines Links oder das ein andere Benutzer seinen Link kopiert hat. Ein Problem des Konzeptes von DAIKnow ist, dass häufige Benutzer alle verfügbaren Abzeichen freischalteten und damit der Gamification-Aspekt abgeschwächt wurde, da keine neuen Abzeichen hinzugefügt wurden.

Die Untersuchung zeigt, dass Gamification bei den Beispielen erfolgreich eingesetzt wird, und bei reddit ein so integraler Bestandteil der Seite ist, dass viele Nutzungsmuster auf reddit davon komplett abhängig sind. Daraus leiten wir ab, dass Gamification einen positiven Nutzen für das Infoboard erzielen kann. Allerdings ist damit nicht endgültig geklärt wie die Gamification im Detail implementiert wird, da das Infoboard im Gegensatz zu reddit oder DAIKnow, keine Möglichkeit besitzt eigenen Inhalt einzureichen. Die genaue Spezifikation der Gamification wird in 2.2 erarbeitet.

# Kapitel 2

## Konzeptuelle Umsetzung

Einer der wichtigsten Aspekte von Interface-getriebenen Applikationen ist das Nutzererlebnis [18, S. 8 - 17]. In diesem Kapitel werden die zwei Hauptmerkmale dieser Arbeit, die für das Nutzererlebnis wichtig sind analysiert: Gamification und die Visualisierung der Informationen. Auf dieser Basis wird ein Konzept vorgestellt, welches in Kapitel 3 umgesetzt wird.

### 2.1 Visualisierung der Informationen

Wie [19] gezeigt hat, erlauben heutige Webtechnologien nahezu uneingeschränkte Möglichkeiten Daten zu visualisieren. Abseits von der initialen Idee, eine Darstellung ähnlich einer *Social Media Wall* zu benutzen, gibt es eine Vielzahl von Parametern, die betrachtet werden müssen: Das Layout, die Animationen und das Design der individuellen Information.

#### 2.1.1 Layout

Wie Pinterest [20] oder das Metro-Design von Microsoft zeigen, ist ein auf Kacheln basierendes Layout aktuell eine der populärsten Darstellungen für Medieninhalte verschiedener Art, siehe Abbildung 2.1.



Abbildung 2.1: Links das Rasterbasierte Layout von Windows 8 [21], rechts das spaltenbasierte Layout von Pinterest.

Oft existiert eine Regel, nach welchem Muster die Kacheln angeordnet werden. Zum Beispiel sind die neuesten Kacheln an oberster Stelle positioniert, oder *wichtigere* Kacheln werden prominenter dargestellt. Die Kacheln für das Infoboard unterliegen der Regel, dass sie anhand ihrer Wertung (Relevanz + Nutzerbewertung) angeordnet werden müssen. Die folgende Analyse wird deshalb auf das Verhaltensmuster des Layouts fokussiert

sein, für den Fall dass Kacheln hinzukommen/wegfallen und damit neu sortiert werden müssen. Dies ist ein normaler Anwendungsfall des Infoboards und tritt immer dann auf wenn der Benutzer Informationen zu einem Thema abfragt oder diese Abfrage wieder löscht. Im folgenden wird die rasterbasierte mit der spaltenbasierten Darstellung der Kacheln verglichen.

- **Spaltenbasierte Anordnung**

Bei dieser Form werden die einzelnen Kacheln mit einheitlicher Breite und variabler Höhe in Spalten in der Größenordnung von 3-5 dargestellt, siehe Abbildung fig:column.

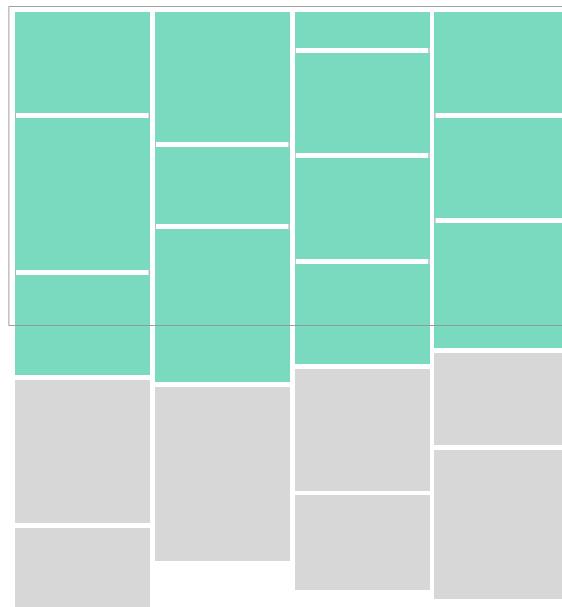


Abbildung 2.2: Spaltenbasierte Anordnung der Inhalte

### Vorteile

Die Vorteile liegen in der Simplizität der Implementierung und der Intuitivität des Inhaltflusses. Neuer Inhalt wird oben eingefügt und älterer Inhalt rutscht nach unten, wobei je neu hinzugekommener Kachel jeweils nur eine Spalte verrutscht. Weiterhin ermöglicht die variable Höhe viel Flexibilität beim Anzeigen des Inhalts - zum Beispiel können lange Texte angemessen angezeigt werden, ohne die Zeichenanzahl zu begrenzen oder ähnliche Limitierungen einzuführen.

### 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 Kacheln einer Sortierung unterliegen, kann der Inhaltsfluss nicht optimal funktionieren. Siehe Abbildung 2.3 für eine detaillierte visuelle Erklärung.

Man kann den optimalen Fluss nur beibehalten indem man die Sortierung Spaltenweise gliedert, dazu müssen jedoch die Kacheln auf eine Spalte festgesetzt werden zum Beispiel mittels einer Hash-Funktion. Dadurch kann aber eine korrekte Sortierung nicht beibehalten werden, wodurch die Güte dieser vom Zufall abhängig und den Benutzer irritieren kann.

- **Rasterbasierte Anordnung** Bei dieser Form werden die einzelnen Datenpunkte in einem einheitlichen Raster dargestellt, das heißt die darstellende Fläche wird in

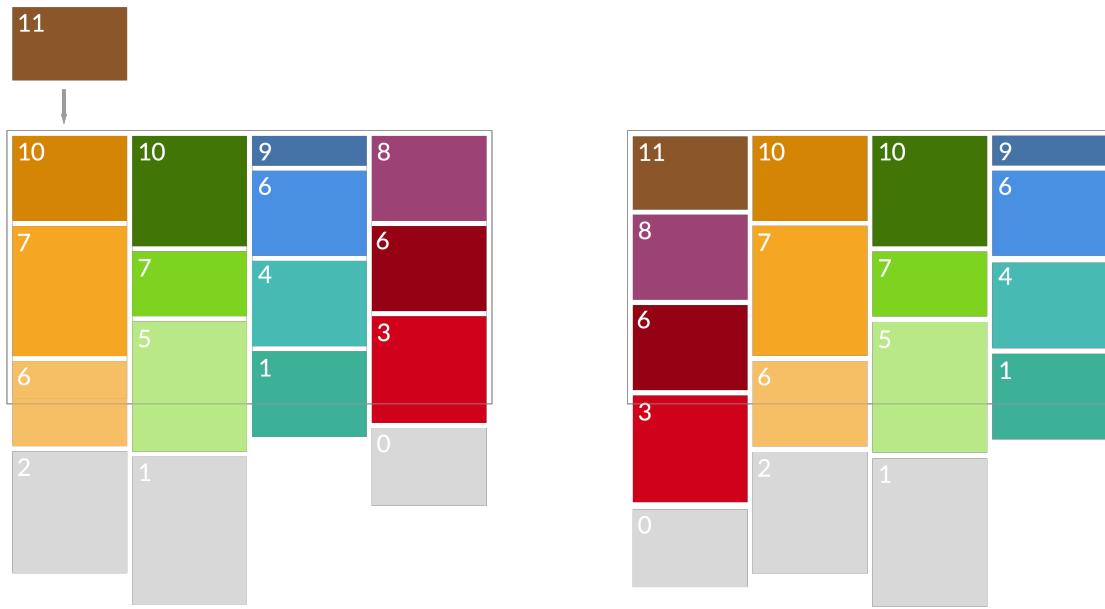


Abbildung 2.3: Links wird eine neue Kachel eingefügt die den höchsten Wert besitzt von allen Kacheln, rechts ist das Ergebnis bei Beibehaltung der Sortierung von links nach rechts.

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, siehe Abbildung 2.4.

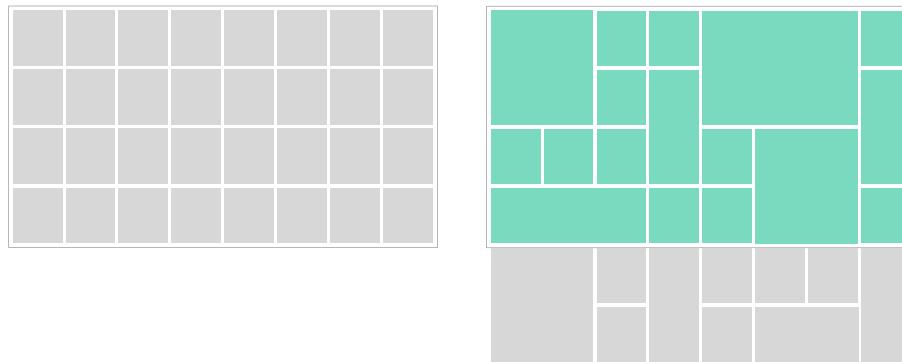


Abbildung 2.4: links: Das Raster des Grids, rechts: eine zufällige Benutzung des Grids

### Vorteile

- Ansprechendes Aussehen
- Bedeutung 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 beim Spaltendesign

### Nachteile

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

### 2.1.2 Probleme beim Flussverhalten

Das Flussverhalten beim Einfügen neuer Inhalte ist bei einem Grid-basierten Layout mit unterschiedlichen Kachelgrößen äußerst komplex. Zunächst müsste das Behälterproblem gelöst werden<sup>1</sup>, und im zweiten Schritt müssten die Kacheln anhand ihrer Bewertung nochmals sortiert werden. Die Lösungen des Behälterproblems so zu optimieren, dass der optische Fluss der Kacheln minimiert wird, ist von derartiger Komplexität, dass dies nicht in Betracht gezogen werden kann. Die JavaScript-Bibliothek packery [22] implementiert den ersten Schritt und ordnet die Kacheln anhand einer Lösung des Behälterproblems. Anhand der packery-Methode prepend kann sehr gut erahnt werden, dass der Fluss nicht einfach nachvollziehbar ist, siehe Abbildung 2.5.

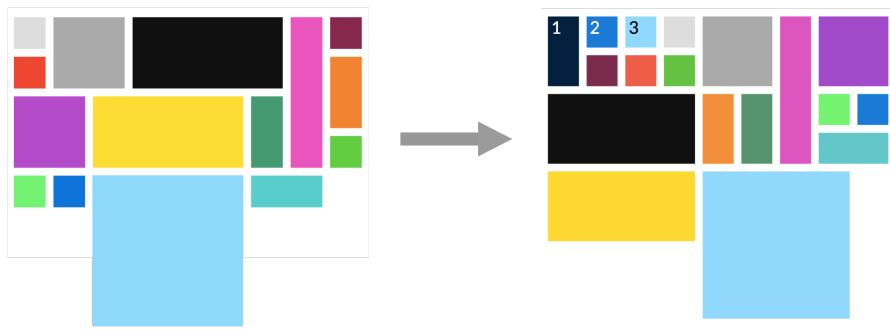


Abbildung 2.5: Zu dem Grid links wurden die Elemente 1, 2 und 3 hinzugefügt. Das resultierende neue Layout ist rechts zu sehen. Dies ist ein praktisches Beispiel welches mithilfe von packery durchgeführt wurde.

### 2.1.3 Animationen

Die Nutzung von funktionalen Animationen wurde in den letzten Jahren zu einem populären Mittel, um die Nutzererfahrung zu verbessern [23]. Vergleich hier zu Googles Material Design [24], bei dem eines der genannten Designziele “Motion provides meaning” ist. Bei diesem Design wird viel Wert darauf gelegt, mittels Animationen die Nutzererfahrung zu erhöhen. Genauer wird bei den Animationen geschrieben: “In the world of material design, motion describes spatial relationships, functionality, and intention with beauty and fluidity” (Google 2014, [24]). So war es auch hier eines der gesetzten Ziele, Animationen zu entwickeln, die, neben dem rein ästhetischen Aspekt, einen Vorteil im Verständnis des Datenflusses der Applikation ermöglichen.

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 von oben kommen. Beispiele im echten Leben sind Börsenticker, Nachrichtenseiten oder die Abfahrttafeln an Bahnhöfen.

<sup>1</sup>Dies ist nötig damit die minimale Anzahl an Lücken entstehen.

2. Die neuen Kacheln ordnen sich an ihrem Platz ein, 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 Beispiele gezeigt haben, dass eine 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 ihrer Anordnung die wichtigste Design-Entscheidung. Zusätzlich zu dem rein funktionalen Aspekt ist auch das Aussehen wichtig, da gewährleistet sein muss, dass Inhalte leicht voneinander unterschieden werden können und die Fülle an Information den Benutzer nicht überfordert.

Rein funktional besitzt eine Kachel folgende Funktionalitäten:

- Anzeige des 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, etc) angezeigt.
- Kacheln können schnell zu dem Thema zugeordnet werden, dem sie assoziiert sind.
- 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 Interaktionen zu bieten. Auch sollten die Metainformationen gleich angezeigt werden. Mit diesen Anforderungen kann eine funktionale Kachel konzipiert werden, siehe Abbildung 2.6.

Im Normalbetrieb des Infoboards werden gleichzeitig die Informationen mehrerer Themen angezeigt. Um diese Informationsfülle für den Benutzer verständlicher zu machen, sollte es für den Benutzer möglich sein, schnell zu sehen, welche Informationen zu welchem Thema gehören. Menschen sind äußerst gut darin, Farben und unterschiedliche Formen schnell zu gruppieren. Nach aktuellem Kenntnisstand sind dies Teile der ersten Verarbeitungsstufe von visuellen Informationen [25]. Eine schnelle Assoziation von Thema und Kachel ist elementar wichtig, deswegen wird 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 mehr möglich sind. Man könnte zum Beispiel das Symbol zum Favorisieren nicht überall rot anzeigen. Doch war das dadurch erzielte Design so überzeugend, dass dies in Kauf genommen wurde. Ein erstes Design kann in Abbildung 2.7 betrachtet werden.

## KAPITEL 2. KONZEPTUELLE UMSETZUNG

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

Abbildung 2.6: Ein Wireframe mit rein funktionalen Aspekten.

The figure displays three wireframes representing different document types:

- PDF:** Shows a document titled "Machine Learning" with a rating of 30. The content area contains a file icon and the text: "The title of the document. Some titles are quite long, or even longer. Just some more words to simulate a long title." The footer shows the date "20.12.2014" and domain "domain.com".
- Web:** Shows a document titled "Some other query" with a rating of 51. The content area contains a globe icon and the same descriptive text as the PDF version. The footer shows the date "20.12.2014" and domain "domain.com".
- Contact:** Shows a document titled "Hans Müller" with a rating of 39. The content area contains a portrait of a man (Tom Hardy) and the text: "Hans Müller Abteilung NORD". The footer shows the date "20.12.2014" and domain "domain.com".

Abbildung 2.7: Ein erstes Design, dass verschiedene Darstellungsformen für unterschiedliche Typen von Informationen benutzt.

### 2.1.5 Eindeutige Zuordnung von Farben zu Themen

Da in Abschnitt 2.1.4 entschieden wurde, die Hintergrundfarbe der Kacheln als Hauptunterscheidungsmerkmal der Themen zu verwenden, muss nun festgelegt werden wie eine Farbe zu einem Thema ausgewählt werden soll. Es wird eine eindeutige Zuordnung von Farbe zu Thema benutzt, damit der Benutzer bei wiederholten gesuchten Themen, allein die Farbe benutzen kann um zu wissen, welches Thema eine Kachel hat. Alle angezeigten Farben sollen optisch zueinander gehören und miteinander harmonieren. Um das zu erreichen, gibt es verschiedene Ansätze. Bei allen ist die Konvertierung des Themas zu einer Zahl, der *erste* Schritt. Diese Funktion sollte die typischen Eigenschaften einer *guten* Hash-Funktion mitbringen: gleichmäßige Verteilung der Funktionswerte auf den Zielraum, minimale Kollisionen und Determinismus. Die hier Verwendete stammt von Dan Bernstein und ist bekannt unter der Bezeichnung djb2 [26].

Der erste Ansatz wäre es, eine Liste mit zueinander harmonierenden Farben zu verwenden und jeweils eine auszusuchen. Das kann jedoch nur mit einer langen Liste von Farben funktionieren. Außerdem kann es bei dieser Methode häufiger vorkommen, dass Themen dieselbe Farbe zugewiesen bekommen, was äußerst unerwünscht ist. Man könnte sich nun überlegen, ob komplexe Konfliktresolution, also wenn Themen miteinander kollidieren und einer davon eine andere Farbe bekommt, betrieben werden sollte. Dadurch wird aber verhindert, dass jedes Thema 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 zufällig und würden nur per Zufall harmonieren. Ein weiteres Problem ist, dass zu viele dunkle und helle Farben erstellt werden.

Der RGB-Farbraum ist nicht dazu geeignet die Helligkeit der entstehenden Farben zu kontrollieren, dafür sind Wahrnehmungsorientierte Modelle, die Farben durch Helligkeit, Sättigung und Farbton beschreiben, deutlich geeigneter [27]. Die populärsten Wahrnehmungsorientierten Modelle sind die HSx-Räume wie HSV oder HSL. Probleme von den HSx Farträumen sind die für den Menschen ungünstige Verteilung der Farben und die Helligkeit der Farben, die bei Änderung des Farbtons stark variiert [28]. Abhilfe schaffen die dafür erstellten Farträume Lab (CIELAB) bzw. die Weiterentwicklung HCL (CIELUV). Sie sind jedoch nicht einfach zu benutzen, weil der valide Wertebereich der Eingabeparameter je nach Kombination der Werte unterschiedlich ist. Das macht es schwer, mit diesen Farträumen Farben prozedural zu erstellen.

Die Wahl eines idealen Farbraums ist damit nicht eindeutig möglich, mittels der HSx-Räume ist es einfach, eine gute Farbskala prozedural zu erstellen, der Farbbereich ist jedoch ungünstig für den Menschen. LAB und HCL bieten die beste Farbverteilung für die menschliche Wahrnehmung, sind jedoch schwierig einzurichten. Eine mögliche Kombination der Vorteile beider Farträume zeigt [28]. Hier wird ein Farbraum mit der Bezeichnung *human-friendly HSL* (kurz HUSL) vorgestellt. HUSL bietet einen einfach zu bedienenden Farbraum, welcher versucht die Helligkeitsschwankungen bei Farbtonänderung zu minimieren und die Farbverteilung für das menschliche Auge zu optimieren.

Doch keiner der Farträume konnte eine *ideale* Farbskala erstellen. Es konnten in allen Farträumen sehr anspruchsvolle Farbskalen mit wenig Kollisionen erstellt werden. Da Farbskalen ein subjektives Thema sind, wurde die Möglichkeit eingebaut, sie in der Applikation zu verändern - dies wurde gleich mit dem Gamification-Aspekt verbunden, wodurch man nach und nach neue Farbskalen freischalten kann und der Nutzer sich diejenigen aussuchen kann, die ihm am meisten zusagen.

Zum Testen der Farbskalen wurde eine Liste von möglichen Themen benutzt. Die Ergebnisse sind in Abbildung 2.8 zu sehen.

### HSL



### RGB



### HUSL



### HCL



Abbildung 2.8: Farbskalen, die mit verschiedenen Farbräumen erstellt wurden.

## 2.2 Gamification

In Abschnitt 1.3.2 wurde Gamification definiert, sowie anhand von Anwendungen gezeigt, dass Gamification einen Mehrwert für das Infoboard erzeugen kann. Im Folgenden werden die für das Infoboard eingesetzten Gamification-Mechanismen erläutert.

Die Grundlage unseres Spieledesigns ist der *Compulsion Loop*, der “eine modellierte Kette an Aktivitäten darstellt, die gewohnheitsmäßig wiederholt werden, um eine neurochemische Belohnung zu erhalten” ([6], übersetzt aus dem Englischen)..

Nach [6] besteht diese modellierte Kette aus 3 Schritten:

### 1. Anerkennung der Leistung

Die Ausführung der Aktionen wird belohnt, der Benutzer bekommt etwas das einen Wert für ihn besitzt. Die Herausforderung hierbei ist, etwas zu erschaffen, dass der Benutzer möchte und die Belohnung so einzuteilen, dass er gerade soviel bekommt, dass es eine Belohnung darstellt. Am Beispiel des Rollenspiels wäre eine Belohnung 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 zu gering, um sofortige Glücksgefühle auszulösen. Erst die Anhäufung vieler 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. Im Rollenspiel könnte das Gold sein, dass benötigt wird um eine neue Rüstung zu kaufen.

### 3. Aktion

Die Aktion ist die Grundlage der Schleife. Aktionen sind das was der Benutzer ausführen soll. Im Rollenspiel könnte dies das Lösen von Aufträgen sein.

Im Folgenden wird die Gamification des Infoboard anhand dieser Schritten modelliert.

#### 2.2.1 Aktionen und deren Belohnung

Das Infoboard besitzt ausgewählte Interaktionsmöglichkeiten, die alle Teil der Aktionen sind, die eine Belohnung geben:

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

Durch einige dieser Aktionen bekommt der Benutzer Punkte. Wichtig für uns ist es, dass die Bewertung der Suchergebnisse besonders gefördert wird, weswegen für diese Aktionen 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 analysiert. Anhand dieser wird abschließend ein passendes Konzept für das Infoboard entwickelt.

#### Punkte

Punkte sind unerlässlich im Spielekontext. Auch wenn die Punkte für den Benutzer nicht sichtbar gemacht werden, sind sie wichtig um dem Spieldesigner die Möglichkeit zu geben, das System zu evaluieren und daraufhin zu verändern [9, pp. 36]. Nach Zicherman [9] 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 aufbauen.

Erfahrungspunkte stellen die wichtigste Kategorie von Punkten dar, jede Interaktion des Benutzers wird mittels Erfahrungspunkten festgehalten [9, S. 38 - 39] und stellen bei einer bedachten Punktevergabe eine gute Quantifizierung der Leistung des Benutzers da. Dadurch kann man Erfahrungspunkte gleichzeitig für eine Bestenliste benutzen - und damit als eine Art Ansehenspunkte. Um den Punktemechanismus noch etwas spannender zu gestalten, wurde entschieden, dass sie gegen *Booster* eingetauscht werden können. Mittels Boostern wird jeder Erhalt von Punkten für einen gewissen Zeitraum mit einem Multiplikator erhöht. So kann eine Kategorie von Boostern bewirken, dass der Benutzer für einen Tag die doppelte Anzahl an Punkten erhält.

Die Idee hinter den Boostern ist es, den Compulsion-Loop daraufhin zu unterstützen, dass der Nutzer sie kaufen wird, um seine Punktzahl noch weiter zu erhöhen. Durch die kurze Verweildauer des Boosters und dem Ziel des Benutzers seine Punkte zu erhöhen, wird dieser zwangsläufig seine Nutzung mit dem System erhöhen, da sich der Booster sonst nicht gelohnt hätte.

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 dargestellt, wobei ein Balkendiagramm eine Kategorisierung anzeigt, wie der Benutzer diese Punkte genau erlangt hat, siehe Abbildung 2.9.

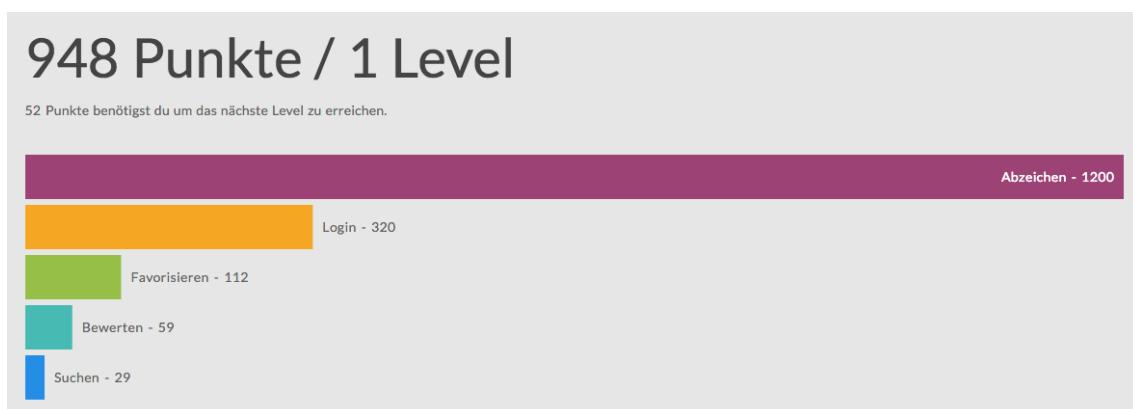


Abbildung 2.9: Die Punkte des Benutzers aufgeteilt in Kategorien.

Zusammengefasst benutzen wir **Erfahrungspunkte**, die für alle Aktionen vergeben werden und für *Booster* ausgegeben werden können. Bestenlisten ermöglichen einen Vergleich der Punkte mit anderen Benutzern.

### Bestenliste

Damit der Benutzer sich mit anderen vergleichen kann, werden Bestenlisten benutzt. Nach [9, pp. 49 - 50] sind Bestenlisten zu einem integralen Bestandteil unseres alltäglichen Lebens geworden. So sehr, dass wir sofort eine erkennen, wenn wir sie sehen. Beispiele für Bestenlisten im echten Leben umfassen die Ergebnisse der Stiftung Warentest, Klausurnotenhaushänge oder die Bundesligatabellen.

Nach [9, pp. 50 - 51] kann zwischen zwei Arten von Bestenlisten unterschieden werden, 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 der Mitte angezeigt. Hierbei ist es irrelevant ob er Platz 43 oder 40000 ist, er sieht sich selbst immer in der Mitte der Liste. Um sich herum sieht er seine direkte Konkurrenz. 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 *unendlichen* 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 [9, pp. 51].

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 viele 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 zu beschränken, zum Beispiel auf die letzten 7 Tage. Dies hat dann aber einen ähnlich negativen Effekt auf die dauerhaften Nutzer: Sie können ihre Vormachtstellung nicht demonstrieren.

Es wurde deswegen entschieden immer 2 Bestenlisten anzuzeigen: Eine mit dem kompletten Zeitraum und eine mit den letzten 30 Tage. Dadurch ist es möglich beide Parteien zufriedenzustellen, 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 globalen Statistiken wird eine *unendliche* Bestenliste benutzt. Siehe Abbildung 2.10 für die *lass-dich-nicht-entmutigen*-Bestenliste.

### 2.2.3 Abzeichen

Abzeichen bzw. Badges sind ein etabliertes Element der Gamification. Nach [9, pp. 55] nutzen sie zahlreiche Eigenschaften der menschlichen Psyche, um begehrwert 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 mit speziellen Abzeichen ersetzen, als Beispiel sei Foursquare gennant, bei der der Benutzer das Abzeichen für das Erreichen einer bestimmten Anzahl an *check-ins*<sup>2</sup> erhält. Dies funktioniert bei Foursquare deswegen gut, weil *check-ins* der wichtigste Bestandteil der Applikation sind [9, pp. 57]. Auch haben *check-ins* keine negative Konnotation, da es zumeist als überaus positiv angesehen wird, wenn man häufig ausgeht und neue Orte erkundet.

Trotz aller positiven Aspekte von Abzeichen sind sie nicht einfach zu verwenden, so benutzt [9, pp. 56] den Begriff “badgenfreude” für ein Konzept, bei dem zu viele langweilige und sinnlose Abzeichen den Sinn von Abzeichen zunichte machen und sogar negative Folgen für das *User Engagement* haben können.

Es ist sehr wichtig, dass die Abzeichen den Benutzer nicht stören. Er soll es mögen, sie zu bekommen. Aber wann mag er es Abzeichen zu bekommen, und wann stören sie? Zunächst kann die Frequenz des Erhalts betrachten werden. Um anfangs den Benutzer zu motivieren gibt man ihm in der ersten Zeit meistens relativ viele Abzeichen, so ist es typisch ein Abzeichen für die erste Anmeldung oder die erste geschaffte Herausforderung zu geben. Die Frequenz muss mit der Zeit nachlassen, damit sie den Benutzer

<sup>2</sup>Der Begriff *check-in* wird von vielen sozialen Diensten benutzt um an sich an einem realen Ort “einzutragen” - um dies z.B. seinen Freunden mitzuteilen

Bestenliste			
Platz	Name	Punkte	Abzeichen
#47	Joye	1002	6
#48	Micki	995	6
#49	Sarina	985	6
#50	Ashly	956	6
#51	Nick	948	8
#52	Sarina	833	5
#53	Jenae	826	5
#54	Bettina	812	5
#55	Daisy	797	5
#56	Marvin	793	5

Abbildung 2.10: Die Benutzerzentrische *lass-dich-nicht-entmutigen* Bestenliste, hier sieht sich der Benutzer immer in der Mitte der Liste.

nicht von der eigentlichen Applikation ablenken, und damit die Abzeichen weiterhin als wertvoll anerkannt werden. Es sollte für den Benutzer immer offensichtlich sein wofür, er ein Abzeichen bekommen hat und welchen Wert es besitzt. Der Wert des Abzeichens sollte graphisch repräsentiert werden, zum Beispiel werden sehr oft die Farben Bronze/Silber/Gold als Analogie zu Medaillen benutzt, um den Wert zu symbolisieren. Damit der Benutzer weiß, welche Abzeichen es noch gibt, und wie er sie erhalten kann, sollte es möglich sein, alle erhältlichen Abzeichen mit einer jeweiligen Beschreibung, wie man es denn erhält, einzusehen.

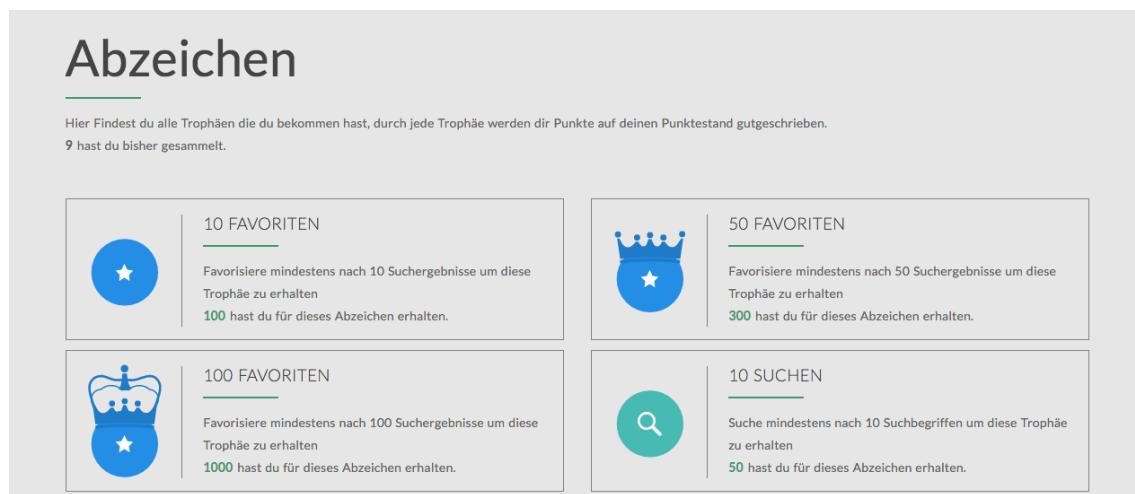


Abbildung 2.11: Die Benutzeroberfläche für die erhaltenen Abzeichen. Der hier gezeigte Benutzer besitzt die Favoriten-Abzeichen in allen Ausführungen.

So hat sich schlussendlich folgendes Schema für die Abzeichen des Infoboards ergeben. Es gibt 3 Kategorien von Abzeichen, die jeweils einen anderen Wert darstellen. Die erste Kategorie sind leicht zu erhaltene Abzeichen, die zweite sind mittelmäßig schwer zu erhaltene Abzeichen, d.h. der Benutzer es haben möchte, kann dieser es auch in angemessener Zeit erhalten. Die Abzeichen der letzten Kategorie benötigen ca. 50x mehr Zeit als die der ersten. Dadurch sollen auch Vielbenutzer weiterhin durch das System motiviert werden. Visuell wird der Wert durch eine Krone dargestellt, wobei die der letzten Kategorie am imposantesten aussieht, siehe dazu Abbildung 2.11. Der Erhalt von neuen Abzeichen wird so schnell wie möglich dem Benutzer mitgeteilt - 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 Abbildung 2.12. Dieser Vorgang dauert weniger als eine halbe Sekunde. Damit das Abzeichensammeln nicht langweilig und obsolet wird, sollten regelmäßig Neue eingeführt werden. Der Quelltext wurde daraufhin optimiert, dass dies keinen größeren Aufwand darstellt.



Du hast ein Abzeichen erhalten!  
100 Suchergebnisse wurden favorisiert.  
**1000** Punkte hast du dafür bekommen.



Abbildung 2.12: Um den Benutzer über das Erreichen eines neuen Levels oder den Erhalt eines Abzeichens zu informieren, werden Flashmessages benutzt.

# Kapitel 3

## Technische Umsetzung

In Kapitel 2 wurde die Applikation soweit geplant, dass die technische Umsetzung erfolgen kann. Viel Wert wurde darauf gelegt, die getroffenen Technologieentscheidungen, wie auch die gewählte Architektur, zu begründen. Die technische Umsetzung wird sich größtenteils mit dem Frontend beschäftigen, das Backend wird kurz in Kapitel 3.3 behandelt.

### 3.1 Wahl der Technologie

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

Die Wahl derer ist ein wichtiger Punkt, der die darauffolgende Entwicklung und spätere Wartung beeinflusst [29]. Es wird versucht auf folgende Punkte einzugehen:

- Die Technologie ist *erwachsen*

Die verwendete Bibliothek oder ähnliches ist nicht allzu neu und hat sich in vielen Applikationen bewährt. Sie ist soweit ausgereift, dass Umgehungslösungen oder Fehlerbehebungen 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, dass sie einfach zu lernen sind und intuitiv in der Benutzung sind.

- Die Technologie ist zukunftssicher

Es ist immer schwer abzuschätzen, 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 Programmiersprache

Die wichtigste Wahl ist die Wahl der Programmiersprache [29]. Im Frontend wird diese Entscheidung größtenteils dadurch abgenommen, dass aktuelle Browser exklusiv die Ausführung von JavaScript beherrschen, andere Sprachen sind nur über Erweiterungen (Flash, Java, Silverlight) verfügbar, deren Relevanz in den letzten Jahrzehnten soweit abgenommen hat, dass sie für uns keine Option mehr darstellen [30].

JavaScript ist allerdings nicht die beliebteste Sprache. Die Tatsache das die Sprache in ca. 10 Tagen geschrieben wurde mag dazu beigetragen haben [31]. So kam es, dass JavaScript in den letzten Jahren immer mehr zu einem Kompilierungsziel geworden ist, d.h.

andere Programmiersprachen kompilieren zu validem JavaScript-Quelltext. Diese Praxis ist mittlerweile so populär geworden, dass JavaScript auch als *Assembly of the Web* bezeichnet wird [32]. Die populärsten Programmiersprachen, die zu JavaScript kompilieren, sind vermutlich CoffeeScript, TypeScript und Java (mittels GWT). Die Liste der Sprachen ist endlos mit über 300-Sprachen, die zu JavaScript kompilieren. Eine Auflistung jener ist auf dem Github-repository von CoffeeScript zu finden [33].

Zakai [34] 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 zum Beispiel der TeX-Compiler im Browser ausgeführt werden konnten [35].

Die Nutzung dieser Sprachen, vor allem der populärsten, hat viele Vorteile: von vereinfachter Syntax (CoffeeScript) bis hin zu statischer Typisierung (Google Web Toolkit - GWT). So bleibt die Frage, ob es überhaupt noch zukunftssicher bzw. praktikabel ist reines JavaScript zu benutzen. Die Frage kann mit einem klaren Ja beantworten werden, wenn eine Prämissen an die Ausführungsumgebung gesetzt wird: Sie unterstützt JavaScript in der Version 6, bzw. ECMAScript 6 (Im nachfolgenden mit ES6 abgekürzt). Die für JavaScript zuständige Organisation ECMA investierte viel Arbeit, um die Tatsache, dass JavaScript nur in 10 Tagen geschrieben wurde, etwas weniger offensichtlich zu machen. Mit ECMAScript 6 [36] trägt diese Arbeit am deutlichsten Früchte. Funktionalitäten wie Destructuring, Generators, Modules, Template Strings oder Promises lassen ES6 zu einer modernen Sprache werden.

Die aktuellen Browser unterstützen bisher nur einen Bruchteil der ES6 Spezifikation, eine komplette Übersicht ist unter [37] zu finden. Es gibt keine offizielle Stellungnahme seitens der Browser-Hersteller wann ES6 komplett unterstützt werden wird, doch kann davon ausgegangen werden, dass dies in den nächsten Jahren geschieht, wenn die ES5-Unterstützung als Vergleich benutzt wird. Etwa zwei Jahre nach offizieller Festlegung von ES5, unterstützten die populärsten Browser ES5 vollständig [38].

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. Die Tatsache, dass JavaScript heutzutage immer im Zusammenhang mit Build-Tools benutzt wird, um zum Beispiel den Quelltext verkleinern oder ihn unlesbarer zu 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 die 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 schon kennen: ES6 ist abwärtskompatibel und die neuen Funktionalitäten können in wenigen Stunden gelernt werden, wenn das überhaupt nötig ist.

Neben reinem ES6 benutzen wir jedoch auch noch eine von Facebook entwickelte Erweiterung namens JSX [39]. Diese erweitert JavaScript mit der Möglichkeit, UI-Komponenten mit einer HTML-Syntax zu beschreiben. Dies wäre z.B. valides JSX:

```
var header = <h1>Überschrift</h1>;
```

Die direkte Einbettung von HTML-Markup in JavaScript wird derzeit kontrovers diskutiert [40], doch wird dadurch die Benutzung von der hier verwendeten Interface-Bibliothek React.js deutlich vereinfacht. Bisher ist JSX ein Vorschlag von Facebook an

ECMA, doch ist die Benutzung von JSX mittlerweile sehr verbreitet. So unterstützt eslint [41] wie auch der ES6-ES5-Kompilierer Babel JSX [42], dass eine Integration etwas JSX-ähnlichem ziemlich wahrscheinlich ist. Mittels ES6-templatestrings ist es möglich *Domain Specific Languages* zu schreiben, so gibt es mittlerweile auch ein proof-of-concept JSX rein in ES6 zu schreiben [43].

Neben Javascript, muss, um die Anwendung zu gestalten, CSS (Cascading Style Sheets) geschrieben werden. CSS besitzt einige Schwächen, deshalb versuchen Projekte wie less [44] oder sass [45] diese zu beheben, indem sie CSS mit Funktionalitäten wie Variablen, Mixins und Verschachtelung erweitern. Allgemein werden diese Projekte CSS-preprocessors gennant, und funktionieren, ähnlich wie den Kompilier-Zu-JavaScript-Sprachen, indem sie die in ihrer Sprache geschriebenen Dateien in CSS übersetzen [46]. Es hat sich bewährt less oder sass zu benutzen, da damit die Effizienz der Entwicklung gesteigert werden kann [46]. 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 sich nicht lohnt, die Kosten/Nutzen genauer zu betrachten.

### 3.1.2 Ladezeiten

Ein großer Faktor der in anderen Umgebungen schon lange keine Rolle mehr spielt, ist die Größe des Quelltextes. Bevor in JavaScript irgendeine Bibliothek benutzt werden kann, muss sie zuerst beim Benutzer heruntergeladen und ausgeführt werden, was bei langsamem Computern mit schlechter Internetverbindung einen großen Zeitaufwand bedeutet. Um den Punkt nochmal zu verdeutlichen, wurde ein kleiner Test durchgeführt. Dazu wurde eine EmberJS-Demoapplikation ausgeführt,<sup>1</sup> 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 finden [47]. Die Ergebnisse

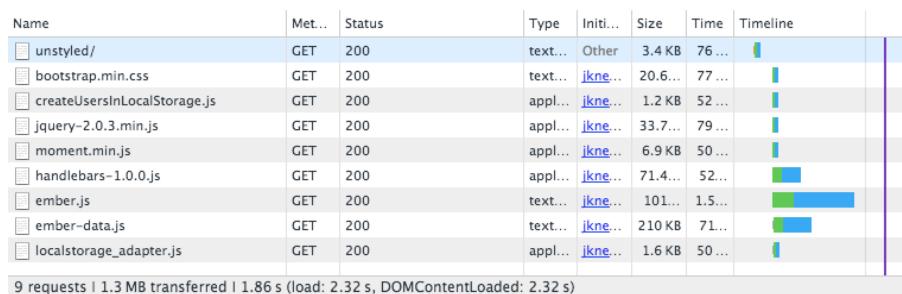


Abbildung 3.1: Übertragungszeit der benötigten Dateien

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.

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

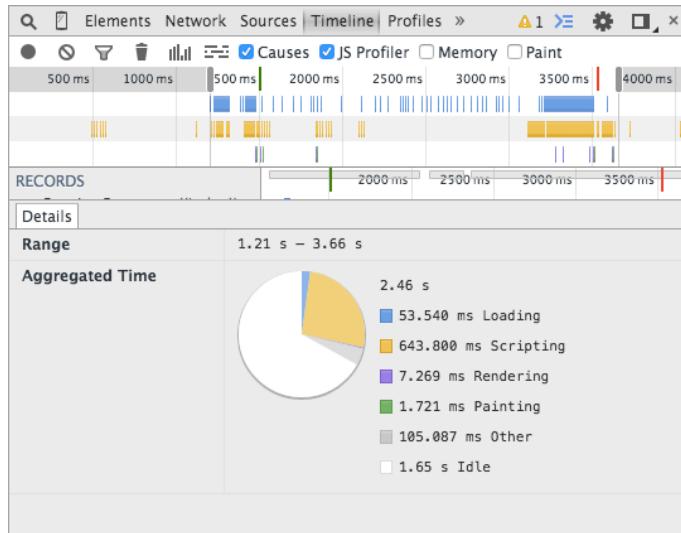


Abbildung 3.2: Benötigte Ausführungszeit der Skriptdatein

### 3.1.3 Frameworks und Bibliotheken

Es wurden schon etliche Webanwendungen entwickelt, somit gibt es kaum ein Problem, das noch nicht gelöst wurde. Es gibt tausende JavaScript-Bibliotheken und Frameworks, die einem das Leben einfacher machen sollen. Ein klarer Platzhirsch ist jedoch nur für Subkategorien wie Datenvisualisierung zu finden.

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 der Entscheidungsfindung in diesem Bereich und zeigt auf, welche Aspekte alles beachtet werden müssen.

Die einflussreichste Entscheidung hierbei ist diejenige für ein Framework wie Dojo[48], AngularJS[49] oder EmberJS[50]. Nach [51] gibt viele Vorteile die für die Benutzung eines Frameworks sprechen: Große Benutzergemeinde, getestete Komponenten, Cross-Browser Unterstützung, eine immense Anzahl an vorgefertigten Lösungen für häufige Probleme/Aufgaben, eine klare Projektstruktur und viele vorgefertigte Komponenten.

Die Benutzung eines Frameworks spart also Zeit und beschleunigt die Entwicklung, doch gibt es auch Gründe, warum immer mehr Leute sich von den großen Frameworks abwenden [52, 53]. Alle Frameworks besitzen eine große Lernkurve, jeder Entwickler ist gezwungen, das Framework zu lernen, um produktiv mitzuarbeiten, da ein Framework jeden Aspekt der Anwendung berührt. Frameworks veralten schnell, 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, das 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, siehe dazu auch Abschnitt 3.1.2, indem gezeigt wird, dass die Größe des Quelltextes einen erheblichen Einfluss auf Ladezeiten hat.

Allen Vorteilen zum Trotz wurde für diese Arbeit entschieden, dass die Benutzung eines Frameworks nicht die optimale Lösung ist, weil zum Beispiel die Visualisierung so aufwändig ist, dass unsere Erfahrung mit großen Frameworks es deutlich machten, dass die gegebenen Abstraktionen derer nicht ausreichen würden.

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

Es wurden am Ende folgende Bibliotheken verwendet

**React.js [54]** wird für die komplette Darstellung der Applikation benutzt. Im nächsten Abschnitt wird auf React.js noch genauer eingegangen, da es viele Innovationen in diesen Bereich bringt.

**react-router [55]** ist eine routing-Bibliothek für react und macht es sehr einfach URLs zu einer View zuzuordnen.

**refluxjs [56]** ist eine Bibliothek für eine unidirektionelle-Datenfluss Architektur, mehr dazu im nächsten Abschnitt.

**bootstrap [57]** 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-bootstrap [55]** bietet für alle verfügbaren bootstrap-Komponenten die entsprechenden react-Komponenten.

**d3.js [19]** ist die berühmteste Bibliothek für Visualisierungen. Hiermit wurde das animierte Säulendiagramm geschrieben, aber auch vieles der Farberzeugung entstand mithilfe von d3.js.

**immutable.js [58]** 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.

**lodash [59]** ist eine utility-Bibliothek mit einem starken Fokus auf funktionaler Programmierung. Sie wurde so ziemlich überall in der Applikation benutzt.

**cookies-js [60] / store.js [61]** 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.js [62]** vereinfacht den Umgang mit Datums-Objekten und bietet Funktionen wie add(1, 'day') und erweiterte Formattierungsmöglichkeiten.

### 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 dass ein sogenanntes Build-Tool diese Dateien nochmals verändert.

Die bekanntesten Build-Tools sind Gulp[63], Grunt[64], Webpack[65] und Brunch[66].

Webpack stellt eine Ausnahme dar, 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. Da die meisten Build-Systeme eben diese Ausgabe haben, kann Webpack die meisten Aufgaben von Build-Tools ersetzen.

Zwar ist Webpack noch ein junges Projekt, aber es hat sich schon für große Projekte wie Instagram bewährt [67].

Für dieses Projekt viel die Wahl auf Webpack, weil es von allen Build-Tools die folgenden Dinge am besten realisieren konnte:

1. Es muss möglich sein JavaScript-Bibliotheken mittels dem npm-Packetmanager zu benutzen.
2. ES6-Quelltext muss zu ES5 Quelltext kompiliert und gebündelt werden.
3. Wenn der Quelltext *minified* wird, müssen SourceMaps erstellt werden.
4. \*.less-stylesheets müssen in CSS umgewandelt und gebündelt werden.
5. Bei Dateiänderungen in der lokalen Entwicklungsumgebung sollte erneut kompiliert werden und die Seite aktualisiert werden.

Die Konfigurationsdatei von Webpack dafür ist keine 50-Zeilen lang und einfach zu lesen<sup>2</sup>. Neben den aufgelisteten Anforderungen besitzt Webpack als einziges Build-Tool die Möglichkeit, bei Dateiänderungen, die Änderungen ohne Neuladen der Applikation einzufügen<sup>3</sup>. Durch den dadurch schnelleren Iterationszyklus, wird die Entwicklung beschleunigt. Dies allein ist eine so nützliche Funktionalität, dass Webpack den anderen Möglichkeiten vorgezogen werden sollte.

### 3.1.5 Facebook React und die Flux Architektur

Facebooks React.js ist eine JavaScript-Bibliothek, die seit ihrem erscheinen im Jahr 2013 viel Beifall geerntet hat. Zum Zeitpunkt dieser Arbeit wird sie produktiv von zahlreichen Firmen benutzt wie z.B. Khan Academy[68], Netflix[69] und AirBnB[70]. Facebook benutzt sie für seine größten Produkte: Facebook und Instagram (Google benutzt das hauseigene Framework AngularJS nicht für das Flaggschiff Google Mail).

React.js ist jedoch kein Framework wie AngularJS oder Ember, es gibt keine Directives, Controllers, Templates oder Models. Das Einzige, was React.js bietet, sind Components, welche ähnlich wie der kommende Standard der *Web Components* [71] wiederverwendbare Interface-Widgets darstellen, indem Sie HTML/CSS und JavaScript koppeln und vom Rest des Systems separieren.

Dabei folgt React.js nicht den *best-practices*, die in den letzten Jahren in Webentwicklung entstanden sind und sich in MVC-Frameworks widerspiegeln. Klassisch gibt es ein Template, das in HTML oder einer ähnlichen Sprache wie YAML geschrieben ist und Platzhalter enthält, die später gefüllt werden - dies ist das V in MVC. Dazu kommt ein

---

<sup>2</sup>Die Datei befindet sich in frontend/webpack.config.js.

<sup>3</sup>hotmodule

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 dar, 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 allerdings nicht heißt, dass es keine Kritik gibt.

## Virtual DOM

Einer der innovativsten Aspekte von React ist die virtuelle Repräsentation des DOMs (Document Object Model [72]). Die normale Interaktion mit dem DOM ist langsam und kompliziert. Es ist schwer den aktuellen Status der Applikation darüber abzubilden - so geschieht das häufig über CSS-Klassen oder spezielle Attribute. 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 (weil der Browser das rendern übernimmt) und Dinge, wie der Fokus auf Eingabe-Feldern, würde verloren gehen, was nicht besonders benutzerfreundlich wäre.

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 überführen. Dies macht Frameworks, die einen virtuellen DOM benutzen deutlich schneller, als solche die die Hauptarbeit auf dem richtigen DOM verrichten. Das ganze Konzept ist 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 [73], skalierbar. Der von React.js verwendete Algorithmus besitzt eine Komplexität von  $O(n)$ , wobei  $n$  die Anzahl der DOM-Elemente ist [74]. Ein Geschwindigkeitsvergleich von Frameworks, die einen virtuellen DOM benutzen, zeigt Abbildung 3.3, bei welcher die verschiedenen Frameworks anhand ihrer TodoMVC Implementierung verglichen werden.<sup>4</sup>.

Es ist zu sehen, dass Ember und AngularJS am langsamsten sind. Backbone ist in diesem Benchmark schneller als React.js, wobei die verwendete Implementierung der React.js-App nicht optimiert ist - Om[76] kann als optimierte React-Version betrachtet werden, da Om im Grunde React.js mit nicht veränderbaren Datenstrukturen<sup>5</sup> (immutable data-structures) ist. Mittels Backbone wie aber auch JQuery oder Dojo können theoretisch Applikationen geschrieben werden, die optimal in Geschwindigkeit sind. Dafür müsste der Programmierer bei jeder Zustandsänderung wissen, welche Teile des DOMs genau upgedatet werden müssen. Dieser Ansatz ist Erfahrungsmäßig nicht skalierbar, da bei solchen Optimierungen oft Grenzfälle vergessen werden und die Größe und Komplexität des Quelltextes stark zunimmt.

Die beiden schnellsten Frameworks, benutzen ebenso eine virtual-DOM Implementierung und wie Om, auch nicht veränderbare Datenstrukturen. Dies ist auch der Grund, weswegen immutable.js[58] verwendet wurde, denn dies konnte die render-Geschwindigkeit in den aufwändigsten Teilen verbessern.

---

<sup>4</sup>Der TodoMVC ist ein Versuch die schiere Anzahl an Webframeworks anhand eines realen Beispiels zu vergleichen. Siehe <http://todomvc.com/>

<sup>5</sup>Durch nicht veränderbare Datenstrukturen kann das diffing noch effizienter gestaltet werden, da viele Teilbäume nicht betrachtet werden müssen.

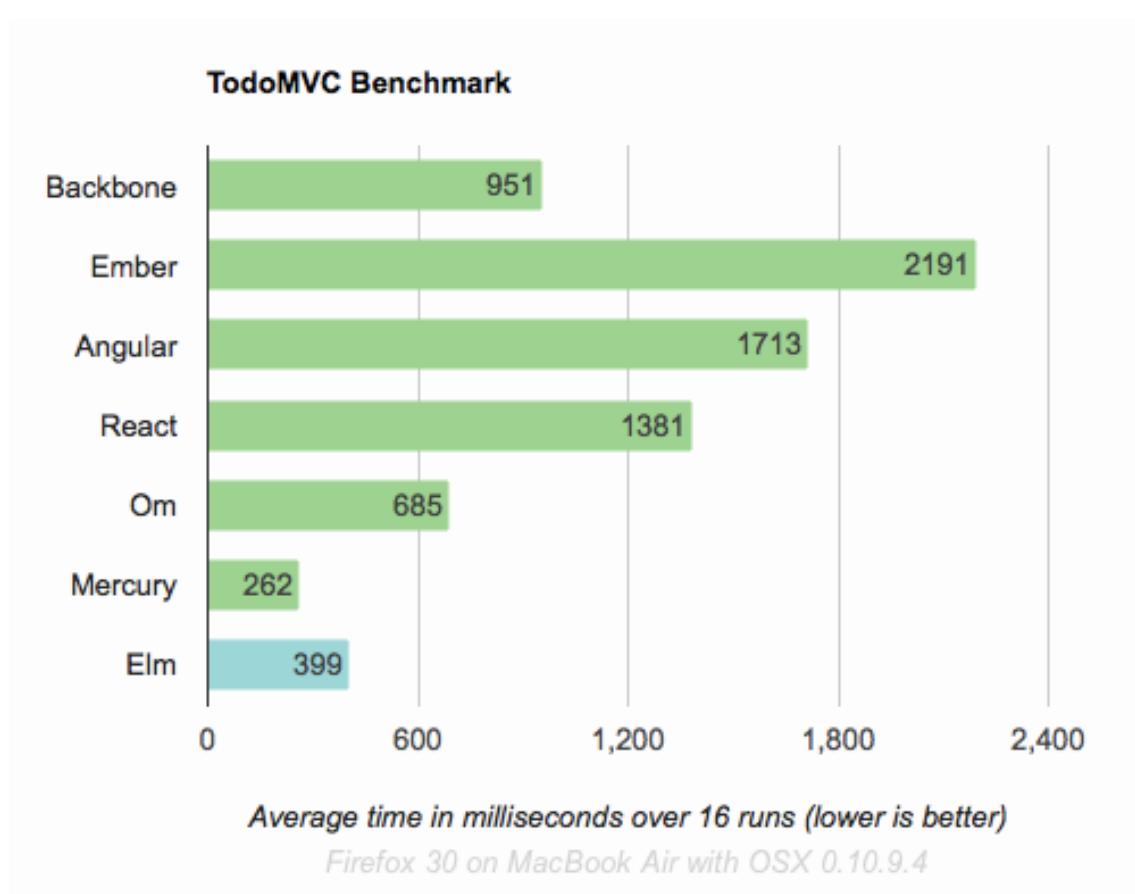


Abbildung 3.3: Geschwindigkeitsvergleich von TodoMVC-Implentierungen mit verschiedenen Frameworks [75].

Ein weiterer Vorteil der virtuellen Darstellung ist die Unabhängigkeit des Browsers, um die Applikation zu rendern. Das 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 rendern, oder in einen HTML-String zu konvertieren. Durch das simple rendern zum HTML-String, was ganz ohne Browser möglich ist, beherrscht React.js isomorphes Rendering, d.h. Inhalte können auf dem Server und im Clienten gerendert werden. Erste Versuche zeigen, dass man dies komplett agnostisch machen kann und alles auf dem Server/Client gerendert werden kann, ohne dass der Benutzer einen Unterschied merkt [77]. So ist es denkbar das schwache Geräte mehr auf dem Server rendern und somit komplexe Inhalte vergleichsweise schnell darstellen können.

### Flux Architektur

Flux ist die Applikations-Architektur, die Facebook für ihre Frontend-Applikationen benutzt. Es ergänzt React.js insoweit, dass es einen unidirektionalen Datenfluss ermöglicht [78]. Es ist wie eine Alternative zu MVC zu verstehen und ist eher ein Muster um Applikationen zu schreiben als ein Framework, das einem rigoros die Struktur vorschreibt. Im Allgemeinen benutzt Flux das Paradigma der Datenfluss Programmierung[79], bei der ein Programm als gerichteter Graph modelliert wird. Typisch für das Paradigma ist die Benutzung eines Beobachter-Entwurfsmuster, wie eines pub/sub-System. Eine Auflistung und Bewertung typischer Muster ist unter [80] zu finden. Bei Flux wird für letzteres eine Abwandlung eines pub/sub-Systems benutzt.

Flux besitzt drei große Bestandteile: Einen Dispatcher, Stores und Views (React.js components). Des Weiteren gibt es Actions, sie sind Hilfsmethoden vom Dispatcher und werden benutzt, um eine semantische API zu unterstützen, die alle möglichen Veränderungen in der Applikation beschreibt.

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

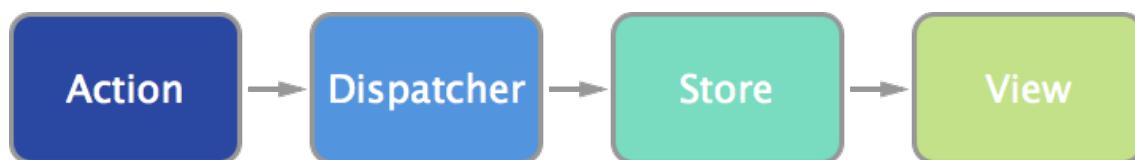


Abbildung 3.4: Dispatcher, Stores und Views sind unabhängige Knoten mit unterschiedlichen Ein und Ausgaben. Die Aktionen sind einfache Objekte die den Typ und die Daten selbst enthalten.

Abbildung 3.4 zufolge ist es 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 verwendet werden, um neue Daten an den Dispatcher zu schicken, der sie an den Store weiterleitet. Mittels des Stores kann dann der Zustand verändert werden, siehe Abbildung 3.5.

Allem in Allem ist die Facebook-Flux Architektur in Kombination mit React.js ein äußerst fortschrittlicher und zukunftssicherer Weg, eine Webapplikation zu schreiben, womit deren Anwendung für die Erstellung des Infoboards als eine gute Wahl erscheint.

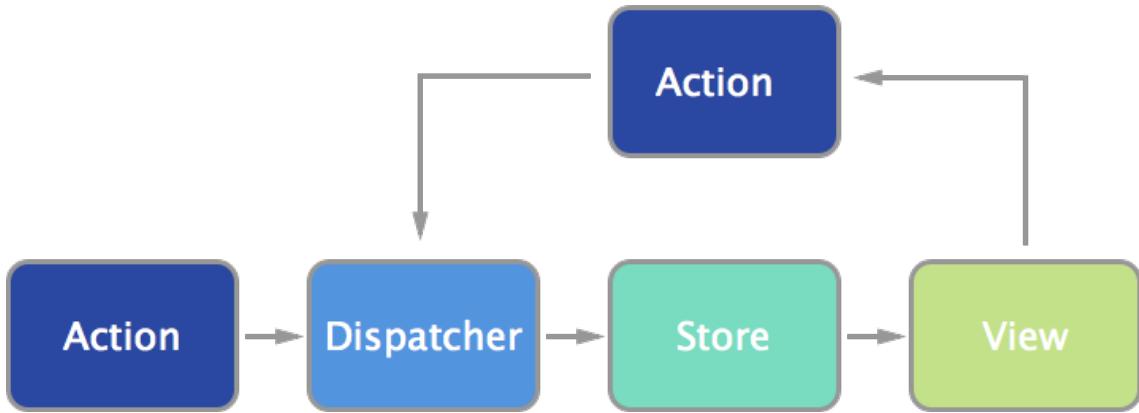


Abbildung 3.5: Mithilfe von Actions ist die View in der Lage, den Zustand der Applikation, zu verändern.

## 3.2 Architektur der Applikation

Wie im vorherigen Abschnitt beschrieben, wurde die Applikation mittels React.js und der Flux-Architektur realisiert. Anstelle der Flux-Implementierung von Facebook, wurde allerdings *reflux* benutzt. Reflux vereinfacht Flux insofern, als dass es keinen einzelnen Dispatcher benutzt. Jede Action ist ihr eigener Dispatcher, mit dem Ziel Flux noch mehr zu vereinfachen und unnötigen Quelltext (sogenannten Boilerplate) zu reduzieren<sup>6</sup>. Eine Vollständige Erklärung für reflux ist auf dem Blog des Autors zu finden [56]. Reflux ist nicht der einzige Versuch, eine Alternative bereitzustellen. Mittlerweile gibt es über 10 verschiedene Flux-Bibliotheken<sup>7</sup> - mit teilweise großen Gemeinden dahinter oder Firmen wie z.B. *Yahoo* mit *Flexible*. 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 [81] ihren Höhepunkt gefunden, wo die Stores gar keinen State mehr besitzen und nur noch als Action-Aggregatoren dienen. Dadurch ergibt sich die Möglichkeit, dass der Zustand der Applikation vor- und zurückgespult werden kann, da er rein von den gesendeten Actions abhängig ist[82].

### 3.2.1 Überblick

Die Abbildung 3.6 zeigt einen Überblick über die gesamte Architektur der Infoboard Applikation. 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 die Abbildung, um sich im Quelltext zurecht zu finden.

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

<sup>6</sup>Keinen zentralen Dispatcher zu benutzen hat auch Nachteile, für dieses Projekt waren die jedoch nicht von Relevanz.

<sup>7</sup>Ein Vergleich dieser ist hier zu finden <https://github.com/voronianski/flux-comparison>

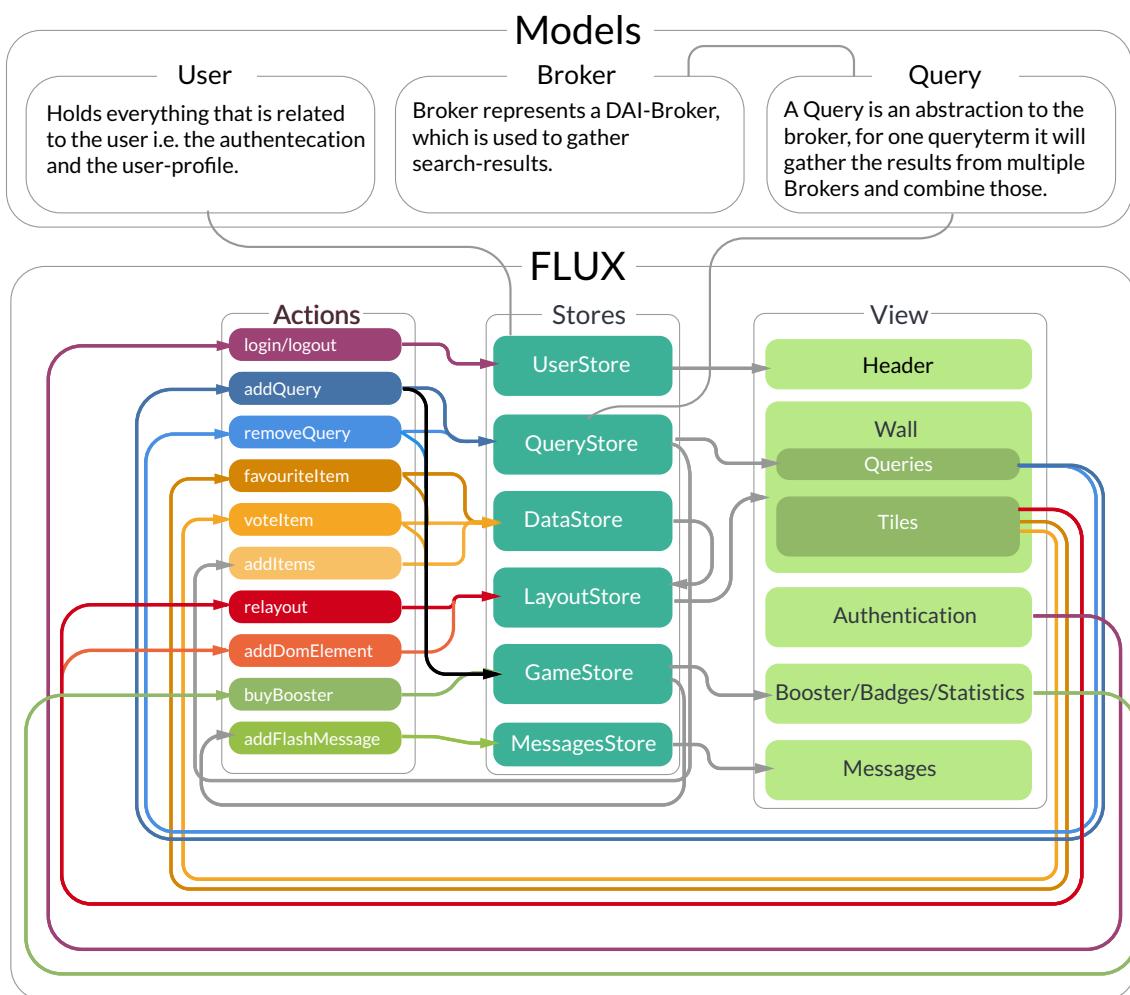


Abbildung 3.6: Die wichtigsten Bestandteile der Applikation.

Zusätzlich 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 aber auch dass alle Komponenten, mit denen der Benutzer nicht direkt interagiert, nicht Teil des Flux-Datenflusses sein müssen.

### 3.2.2 Stores

Im folgenden werden die Stores genauer beschrieben.

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 vier, diese werden nun im Detail behandelt.

#### **QueryStore**

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

#### **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 die 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.

#### **DataStore**

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

## LayoutStore

Im Design-Abschnitt wurde ausführlich darauf eingegangen, was die Layouting-Engine alles leisten muss, leider konnte keine der OpenSource-Bibliotheken allen Anforderungen gerecht werden. Deswegen wurde 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 durch 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 zu sein und hat den Vorteil, dass subpixel-animationen möglich sind, was im Allgemeinen flüssigere Animationen erlaubt [83].

```
.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 der schnellste Ansatz [84]. Hierbei werden die Elemente zu Texturen gerendert und mit Einsatz des Grafikprozessors animiert. Dadurch ist es sogar möglich, auf mobilen Endgeräten flüssige Animationen zu gewährleisten, da diese gewöhnlich dedizierte Grafikprozessoren besitzen.

```
.position-translate3D {
    position: absolute; /* or relative/fixed */
    transform: translate3D(50px, 50px, 0);
    -webkit-transform: translate3D(50px, 50px, 0);
}
```

transform3D wird von allen aktuellen Browsern soweit unterstützt, dass unser Szenario möglich ist[85]. Was weiterhin zu beachten ist, dass am Ende immer gerundete Pixelwerte innerhalb von translate3D(x, y, z) benutzt werden sollten, weil die Elemente ansonsten unscharf sind, siehe Abbildung 3.7.

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:

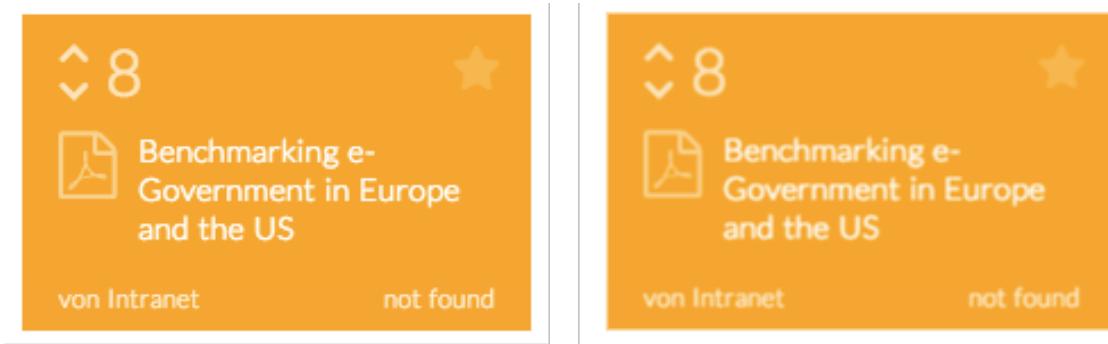


Abbildung 3.7: Links eine Kachel mit geraden Werten bei translate3D, rechts mit jeweils .5 bei  $x/y$ .

1. Berechne die Breite mittels Javascript bevor das Element in den DOM eingefügt wird und setze die Breite mittels style-Attribut.
2. Durch die Benutzung von *Media Queries* können direkt 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. Von uns durchgeführte 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 das 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, muss dieser Wert auch nicht ein zweites Mal ausgelesen werden. Wenn es zu allen Suchergebnissen eine Kachel gibt, wird die `layout`-Funktion aufgerufen, womit die korrekte Position der Kacheln ausgerechnet wird und auf diese weitergegeben wird.

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. Das 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 nicht direkt, welches Element sich verändert hat, es muss durch alle Elemente des `DataStores` mit den Eigenen abgleichen. Dies ist nicht unbedingt optimal, allerdings stellt sich die Frage, ob eine andere Architektur das Problem einfacher lösen würde.

### 3.2.3 Views

Als View wird das Interface bezeichnet, mit dem der Benutzer schlussendlich interagiert. Für jedes View wird React.js und React-Bootstrap verwendet. Die Gründe, warum Re-

act.js verwendet wird, sind in Abschnitt 3.1.5 genauer ausgeführt.

## Responsives Design

Responsives Webdesign bezeichnet ein Paradigma, bei dem Webseiten daraufhin geschrieben werden, dass die Darstellung der Inhalte auf unterschiedlichen Geräten, ohne Einbußen in der Benutzerfreundlichkeit möglich ist. Praktikabel bedeutet das, dass die Darstellung abhängig von der Bildschirmbreite des Gerätes ist. [86].

Aktuell wird das so umgesetzt, dass jeglicher Inhalt mit der Benutzung eines *Grid*-Systems dargestellt wird. Grid-Systeme werden benutzt, um "... Seitenlayouts mittels Zeilen und Spalten, die Inhalte aufnehmen, zu kreieren." ([57], übersetzt aus dem Englischen).

Eine genaue Erklärung von Grid-Systemen ist nicht im Rahmen der Arbeit, für mehr Informationen zu Grid-Systemen wird auf [57] verwiesen. In dieser Applikation wird das von Bootstrap bereitgestellte Grid intensiv benutzt, um auf jedem Endgerät eine möglichst gute Benutzererfahrung zu garantieren.

## Erstellte React.js-Komponenten

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

An dieser Stelle wurde überlegt, eine Einführung in React.js zu schreiben, jedoch würde auch dies die Ansprüche dieser Arbeit übersteigen. Stattdessen wird auf das offizielle Tutorial von Facebook verwiesen [87].

Es wurden 30 React-Komponenten erstellt, viele davon sind nicht anwendungsspezifisch und wurden an mehreren Stellen der Applikation verwendet. Unterteilt sind sie semantisch nach Aufgabe/Seiten, wie zum Beispiel *settings*, *stats* oder *utility*. Eine Minimalübersicht findet sich in Abbildung 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.

Die erstellten Seiten sind in den Abbildungen 4.1, 4.3, 4.4 und 2.11 zu sehen.

Die wichtigste Seite ist die mit der eigentlichen Applikation, nämlich das Infoboard bzw. das Interface für das Suchen von Informationen mittels Themen, zu sehen in Abbildung 4.5. Der Benutzer wird sich voraussichtlich die meiste Zeit auf dieser Seite befinden, deshalb sind auch alle relevanten Aktionen, neben dem Einloggen und dem Kaufen von Boostern, nur auf dieser Seite möglich. Für weitere Details zu dem Konzept dieser Seite, siehe Abschnitt 2.1.

Neben der Seite mit dem Infoboard, sind noch 4 weitere Seiten verfügbar. Diese dienen hauptsächlich der Gamification. Für den Nutzer am interessantesten ist die Seite mit den Benutzerstatistiken, zu sehen in Abbildung 4.2. Weiterhin gibt es noch die Seite, auf dem der Benutzer den derzeitigen Status seines aktuellen Boosters sehen kann, und falls er keinen hat, die Möglichkeit besitzt, einen neuen Booster zu erwerben, siehe Abbildung 4.1. Einstellungen, wie das Auswählen des Farbschemas, kann der Nutzer ebenfalls auf einer dedizierten Seite tätigen, siehe Abbildung 4.3.

Eine allgemeine Übersicht der Aktivität aller Nutzer ist auf der Statistik-Seite zu sehen. Diese Seite ist dafür gedacht, auf einem Fernseher, der in einem öffentlichen Raum des Betriebes aufgehängt ist, angezeigt zu werden. Die Seite unterteilt sich in drei Abschnitte. Der erste Teil zeigt zwei Bestenlisten an, eine für den kompletten Zeitraum und eine

für den letzten Monat. Im zweiten Teil ist die Gesamtpunktzahl der Mitarbeiter zu sehen. Wie auf der Seite mit den Benutzerstatistiken, ist sie aufgeteilt in die einzelnen Kategorien. Der Dritte Teil besteht aus einem Livestream der Nutzeraktivitäten, jedes Mal wenn eine Aktion ausgeführt wurde, dort anzeigen. Die Seite ist zu sehen in Abbildung 4.4.

Der Aufbau der eigentlichen Komponenten, wird Beispielhaft in Appendix 4 anhand der Query-Komponente gezeigt.

### 3.3 Backend

Im Verlaufe der Arbeit wurde erkannt, dass eine dedizierte Backend-Applikation nötig ist, um den Grad der Interaktion und der Dynamik zu ermöglichen, die als Ziele gesetzt wurden. Es gibt einige Frameworks, die zur Frage standen, aber schließlich fiel die Entscheidung auf node.js, da dadurch die Möglichkeit bestand, einiges an Quelltext mit dem Frontend zu teilen. Das node.js *non-blocking* ist und *event-driven IO* benutzt, macht es vorteilhaft für Applikationen, die viele Anfragen verarbeiten müssen [88]. Anders als in klassischen Frameworks wie rails oder django, wo bei jeder Anfrage ein neuer Thread aufgemacht wird, der solange blockiert ist, bis er abgehendelt wurde, wird jede Anfrage von einem *event-loop* beantwortet [88]. Alles in node.js ist asynchron, weswegen diese Anfragen nicht blockieren. Damit ist es mit node.js möglich, bis zu 1 Million Verbindungen zeitgleich zu halten [89].

Als Datenbank wurde mysql verwendet, die durch einen Object Relationship Mapper (kurz ORM) angesteuert wird. Mit einem ORM ist es möglich, Datenbanken mittels Klassen und Objekten zu abstrahieren. Ein ORM ist jedoch nur eine einfache Abstraktion und beherrscht nicht alle Funktionalitäten SQLs, jedoch reichen die Fähigkeiten des ORMs meistens aus und es kann immer auf direkte SQL-Anfragen zurückgegriffen werden. Das hier Verwendete ORM ist sequelize [90]. Als Datenbank wird mysql verwendet, da viel internes Knowhow mit dieser Datenbank bestand. Es können aber ohne Einschränkungen andere von sequelize unterstützte Datenbanken benutzt werden.

Das verwendete Datenmodell kann in Abbildung 4.6 betrachtet werden, es ist komplett mit sequelize beschrieben und ist damit ohne viel Aufwand veränderbar.

#### 3.3.1 HTTP-API

Die Kommunikation von Backend und Frontend geschieht per HTTP-Anfragen, das verwendete Framework ist express.js, es wurde überlegt ein API-Framework wie Swagger [91] zu benutzen. Diese erstellen eine einheitliche, sich selbst-aktualisierende Dokumentation wie auch einfache Tests [92]. Damit wäre jedoch eine weitere Technologie eingeführt worden und da sich die Anzahl der Endpunkte nur auf 8 beläuft, haben wir uns bewusst dagegen entschieden. Im folgenden wird Schemenhaft der Ablauf der Anfragen aller Endpunkte illustriert.

**POST /user/action** Jede Aktion des Benutzers wird an diesen Endpunkt gesendet der folgenden Ablauf besitzt:

1. Finde den angegebenen Benutzer in der Datenbank
2. Suche alle Booster des Benutzers und finde heraus ob einer aktiv ist
3. Füge die Aktion in die Datenbank ein, wenn ein Booster aktiv ist, multipliziere die Punkte der Aktion mit dessen Multiplikator

4. Falls die Aktion auf einem Suchergebnis statt fand, verknüpfe dieses mit der Aktion
5. Berechne die Abzeichen neu, wenn sich diese von den bisherigen Abzeichen unterscheiden, füge die neuen in die Datenbank ein und füge sie der Antwort hinzu
6. Antworte mit der erstellten Aktion und ggf. mit den erstellten Abzeichen

**POST /user/vote** Sobald der Benutzer ein Suchergebnis bewertet, wird diese Anfrage gesendet. Der folgende Ablauf findet statt:

1. Finde den angegebenen Benutzer in der Datenbank
2. Finde das angegebene Suchergebnis in der Datenbank und erstelle es neu falls nicht vorhanden.
3. Finde die bisherige Bewertung des Suchergebnissen oder erstelle diese neu falls nicht vorhanden.
4. Setze den neuen Wert für die Bewertung und speichere diese
5. Antworte mit der erstellten Antwort

**POST /user/booster** Sobald der Benutzer einen Booster erwirbt wird mittels dieser Anfrage die Transaktion in der Datenbank festgehalten. Dazu wird der angegebene Benutzer gesucht und der gekaufte Booster erstellt.

**GET /points** Diese Anfrage nimmt zwei Parameter, Anfangsdatum und Enddatum. Das Ergebnis ist eine Liste aller Benutzer mit ihren jeweiligen Punkten und Abzeichen, zusätzlich werden die Punkte aller Nutzer Kategorieunterteilt und komplett aufsummiert zurückgeliefert.

**GET /items** Diese Anfrage nimmt als Parameter eine Liste an Suchergebnissen-URLs und findet alle Bewertungen und Aktionen auf das Suchergebnis zeigen. Die Bewertungen werden aufsummiert.

**GET /actions** Diese Anfrage liefert die letzten 50 Aktionen zurück.

### 3.3.2 Websocket-API

Um die Dynamik des Infoboards zu maximieren wurden Websockets verwendet die auf alle ausgeführten Aktionen des Systems reagieren und dadurch z.B. die Gesamtpunktzahl erhöhen. Websockets wurden jedoch nicht in ihrer reinen Form benutzt, sondern mittels der Bibliothek socket.io [93]. Diese ermöglicht neben einer einfacheren Benutzung von Websockets, auch einen Alternativmodus mittels polling um Geräte zu unterstützen, die keine Websockets besitzen.

# Kapitel 4

## Fazit und Ausblick

Viele Technologien, die in dieser Arbeit genutzt wurden, sind vor wenigen Jahren noch nicht denkbar gewesen. Da der Maturitätsgrad dieser nicht sehr hoch ist, ist die Nutzung dieser auch mit Risiken verbunden. 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, ästhetisch anspruchsvoll 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 nach aktuellsten Praktiken im Bereich der Benutzererfahrung erstellt und besonders die Darstellung der Suchergebnisse entstand in einem langen iterativen 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 programmietechnisch 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 nie im finalen Entwicklungsstadium, sondern wird ständig weiterentwickelt. Es gibt viele Aspekte wie zum Beispiel die Darstellung der Statistiken, Optimierung der Gamification, Implementierung von Zugriffsrestriktionen oder die 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 Gebiet, dass noch viel Potential birgt.

# Literatur

- [1] **Jim Giles:** „Internet encyclopaedias go head to head“. In: *Nature* 438.7070 (2005), S. 900–901.
- [2] **Pulse:** *Pulse*. <https://www.pulse.me/>. [Online, accessed 07-19-2015]. 2013.
- [3] **Strava Inc:** *Running and Cycling GPS Tracker, Performance Analytics, Maps, Clubs and Competition*. <https://www.strava.com/>. [Online, accessed 07-19-2015]. 2009.
- [4] **Nike Inc:** *Nike+*. <http://www.nikeplus.com.br/>. [Online, accessed 07-19-2015]. 2006.
- [5] **Ali Mesbah und Arie Van Deursen:** „Migrating multi-page web applications to single-page Ajax interfaces“. In: *Software Maintenance and Reengineering, 2007. CSMR'07. 11th European Conference on*. IEEE. 2007, S. 181–190.
- [6] **Joseph Kim:** *The Compulsion Loop Explained*. [http://www.gamasutra.com/blogs/JosephKim/20140323/213728/The\\_Compulsion\\_Loop\\_Explained.php](http://www.gamasutra.com/blogs/JosephKim/20140323/213728/The_Compulsion_Loop_Explained.php). [Online, accessed 06-03-2015]. 2014.
- [7] **Clemens Maria Schuster:** *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]. 2014.
- [8] **walls.io:** *Starbucks*. <https://walls.io/starbucks>. [Online, accessed 07-19-2015]. 2013.
- [9] **Gabe Zichermann und Christopher Cunningham:** *Gamification by design: Implementing game mechanics in web and mobile apps.* ” O'Reilly Media, Inc.”, 2011.
- [10] **Sebastian Deterding u. a.:** „From game design elements to gamefulness: defining gamification“. In: *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*. ACM. 2011, S. 9–15.
- [11] **Reddit:** *The front page of the internet*. <http://reddit.com>. [Online, accessed 07-19-2015]. 2015.
- [12] **Greg Costikyan:** „I Have No Words 8: I Must Design“. In: *The game design reader: A rules of play anthology* (2005).
- [13] **Adrienne Massanari:** „Playful Participatory Culture: Learning from Reddit“. In: *Selected Papers of Internet Research* 3 (2013).
- [14] **Alexa Internet Inc:** *How popular is reddit.com?* <http://www.alexa.com/siteinfo/reddit.com>. [Online, accessed 07-19-2015]. 2015.
- [15] **Annika Richerich:** „‘KARMA, PRECIOUS KARMA!’ KARMAWHORING ON REDDIT AND THE FRONT PAGE’S ECONOMETRISATION“. In: *The Journal of Peer Production* 4 (2014). ISSN: 2213-5316. URL: <http://peerproduction.net/issues/>

- issue-4-value-and-currency/peer-reviewed-articles/karma-precious-karma/.
- [16] **Michael Meder, Till Plumbaum und Frank Hopfgartner**: „Daiknow: A gamified enterprise bookmarking system“. In: *Advances in Information Retrieval*. Springer, 2014, S. 759–762.
  - [17] **LLC Deliccious Science**: *Delicious*. <https://delicious.com/>. [Online, accessed 07-19-2015]. 1996.
  - [18] **Jesse James Garrett**: *Elements of user experience, the: user-centered design for the web and beyond*. Pearson Education, 2010.
  - [19] **Michael Bostock, Vadim Ogievetsky und Jeffrey Heer**: „D<sup>3</sup> data-driven documents“. In: *Visualization and Computer Graphics, IEEE Transactions on* 17.12 (2011), S. 2301–2309.
  - [20] **Pinterest**: *Kreative Ideen entdecken und speichern*. <http://pinterest.com>. [Online, accessed 07-19-2015]. 2015.
  - [21] **Connie Malamed**: *Create A Metro Style Design For eLearning*. <http://theelearningcoach.com/media/graphics/metro-style-for-elearning/>. [Online, accessed 07-19-2015]. 2013.
  - [22] **David DeSandro**: *The bin-packing layout library*. <http://packery.metafizzy.co/>. [Online, accessed 07-19-2015]. 2012.
  - [23] **Amit Daliot**: *Functional Animation In UX Design*. <http://www.smashingmagazine.com/2015/05/functional-ux-design-animations/>. [Online, accessed 07-19-2015]. 2015.
  - [24] **Google Inc**: *Material design*. <https://www.google.com/design/spec/material-design/introduction.html>. [Online, accessed 07-19-2015]. 2015.
  - [25] **Anne Treisman**: „Merkmale und Gegenstände in der visuellen Verarbeitung“. In: *Spektrum der Wissenschaft* 1.1987 (1987), S. 72–82.
  - [26] **Dan Bernstein**: *Hash Functions*. <http://www.cse.yorku.ca/~oz/hash.html>. [Online, accessed 07-19-2015]. 2015.
  - [27] **Barbara J Meier, Anne Morgan Spalter und David B Karelitz**: „Interactive color palette tools“. In: *Computer Graphics and Applications, IEEE* 24.3 (2004), S. 64–72.
  - [28] **Alexei Boronine**: *HUSL Color Space*. <http://www.husl-colors.org/>. [Online; accessed 06-07-2015]. 2015.
  - [29] **Paul Graham**: *Beating the averages*. <http://www.paulgraham.com/avg.html>. [Online, accessed 07-19-2015]. 2004.
  - [30] **Q-Success**: *Client-side programming languages market position report*. [http://w3techs.com/technologies/market/client\\_side\\_language](http://w3techs.com/technologies/market/client_side_language). [Online, accessed 07-19-2015]. 2015.
  - [31] **Charles Severance**: „JavaScript: Designing a language in 10 days“. In: *Computer* 2 (2012), S. 7–8.

- [32] **Brendan Eich**: *From ASM.js to WebAssembly*. <https://brendaneich.com/2015/06/from-asm-js-to-webassembly/>. [Online, accessed 07-19-2015]. 2015.
- [33] **Doug Holton Rory O’Kane Brian Horakh**: *List of languages that compile to JS*. <https://github.com/jashkenas/coffeescript/wiki/list-of-languages-that-compile-to-js>. [Online, accessed 07-19-2015]. 2015.
- [34] **Alon Zakai**: „Emscripten: an LLVM-to-JavaScript compiler“. In: *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*. ACM. 2011, S. 301–312.
- [35] *Compiling LaTeX (TeX live) in your browser*. <https://github.com/manuels/texlive.js/>. [Online, accessed 07-19-2015]. 2013.
- [36] **Ecma International**: <http://www.ecma-international.org/ecma-262/6.0/>. <http://www.ecma-international.org/ecma-262/6.0/>. [Online, accessed 07-19-2015]. 2015.
- [37] **Juriy “kangax” Zaytsev**: *The es6 compatibility table*. <http://kangax.github.io/compat-table/es5/>. [Online, accessed 07-19-2015]. 2015.
- [38] **Sebastian Damm**: *ECMAScript 5 – the current JavaScript standard*. <http://blog.oio.de/2013/04/16/ecmascript-5-the-current-javascript-standard/>. [Online, accessed 07-19-2015]. 2013.
- [39] **Facebook Inc**: *XML-like syntax extension to ECMAScript*. <https://facebook.github.io/jsx/>. [Online, accessed 07-19-2015]. 2014.
- [40] *JSX: XML-like syntax extension to ECMAScript – Draft specification*. <https://news.ycombinator.com/item?id=8265945>. [Online, accessed 07-19-2015]. 2014.
- [41] **Nicholas C. Zakas**: *The pluggable linting utility for JavaScript and JSX*. <http://eslint.org/>. [Online, accessed 07-19-2015]. 2015.
- [42] **James Kyle Sebastian McKenzie**: *Babel - The compiler for writing next generation JavaScript*. <https://babeljs.io/>. [Online, accessed 07-19-2015]. 2015.
- [43] **Dr. Axel Rauschmayer**: *React JSX via ECMAScript 6 template strings*. <http://www.2ality.com/2014/07/jsx-template-strings.html>. [Online, accessed 07-19-2015]. 2015.
- [44] *Getting started | Less.js*. <http://lesscss.org/>. [Online, accessed 07-19-2015]. 2015.
- [45] *Sass: Syntactically Awesome Style Sheets*. <http://sass-lang.com/>. [Online, accessed 07-19-2015]. 2015.
- [46] **Studio Mitte**: *CSS-Preprocessors revolutionieren die CSS-Entwicklung*. <http://www.thinkdesigndevelop.com/2012/06/css-preprocessors-compass-und-sass-revolutionieren-die-css-entwicklung/>. [Online, accessed 07-19-2015]. 2012.
- [47] **Julien Knebel**: *Simple CRUD for Ember.js 1.1.2 with Ember Data 1.0.0beta3*. <https://github.com/jkneb/ember-crud>. [Online, accessed 07-19-2015]. 2013.

- [48] **The Dojo Foundation**: *Dojo Toolkit*. <https://dojotoolkit.org/>. [Online, accessed 07-19-2015]. 2004.
- [49] **Misko Hevery u. a.**: *AngularJS - HTML enhanced for web apps!* <https://angularjs.org/>. [Online, accessed 07-19-2015]. 2010.
- [50] **Peter Wagenet u. a.**: *emberJS - a framework for creating ambitious web applications*. <https://angularjs.org/>. [Online, accessed 07-19-2015]. 2011.
- [51] **Addy Osmani**: *The Pros And Cons Of JavaScript Micro-Frameworks*. <http://addyosmani.com/blog/prosconsmicroframeworks/>. [Online, accessed 07-19-2015]. 2011.
- [52] **Joe Gregorio**: *No more JS frameworks*. [http://bitworking.org/news/2014/05/zero\\_framework\\_manifesto](http://bitworking.org/news/2014/05/zero_framework_manifesto). [Online, accessed 07-19-2015]. 2014.
- [53] **Tero Piirainen**: *Frameworkless JavaScript*. <https://muut.com/blog/technology/frameworkless-javascript.html>. [Online, accessed 07-19-2015]. 2013.
- [54] **Pete Hunt Paul O'Shannessy Ben Alpert u. a.**: *React - a JavaScript library for building user interfaces*. <https://facebook.github.io/react/>. [Online, accessed 07-19-2015]. 2013.
- [55] **Dan Abramov Michael Jackson Ryan Florence**: *The most popular front-end framework, rebuilt for React*. <http://react-bootstrap.github.io/>. [Online, accessed 07-19-2015]. 2014.
- [56] **Mikael Brässman**: *Deconstructing ReactJS's Flux*. <http://spooke.ghost.io/deconstructing-reactjss-flux/>. [Online, accessed 07-19-2015]. 2014.
- [57] **Twitter**: *The world's most popular mobile-first and responsive front-end framework*. <http://getbootstrap.com/>. [Online, accessed 07-19-2015]. 2011.
- [58] **Lee Byron**: *Immutable.js*. <http://facebook.github.io/immutable-js/>. [Online, accessed 07-19-2015]. 2014.
- [59] **Kit Cambridge Jeremy Ashkenas John-David Dalton**: *A JavaScript utility library delivering consistency, modularity, performance, and extras*. <https://lodash.com/>. [Online, accessed 07-19-2015]. 2009.
- [60] **Scott Hamper**: *JavaScript Client-Side Cookie Manipulation Library*. <https://github.com/ScottHamper/Cookies>. [Online, accessed 07-19-2015]. 2012.
- [61] **Marcus Westin**: *localStorage wrapper for all browsers without using cookies or flash*. <https://github.com/ScottHamper/Cookies>. [Online, accessed 07-19-2015]. 2010.
- [62] **Isaac Cambron Tim Wood Iskren Ivov Chernev**: *Parse, validate, manipulate, and display dates in JavaScript*. <http://momentjs.com/>. [Online, accessed 07-19-2015]. 2011.
- [63] **Rob Richardson Eric Schottstall**: *Gulp - the streaming build system*. <http://gulpjs.com/>. [Online, accessed 07-19-2015]. 2015.
- [64] **Kyle Robinson Young Ben Alman**: *Grunt - The JavaScript Task Runner*. <http://gruntjs.com/>. [Online, accessed 07-19-2015]. 2015.

- [65] **Tobias Koppers**: *webpack module bundler*. <http://webpack.github.io/>. [Online, accessed 07-19-2015]. 2012.
- [66] **Thomas Schranz Paul Miller Elan Shanker**: *Brunch - ultra fast HTML5 build tool*. <http://brunch.io/>. [Online, accessed 07-19-2015]. 2015.
- [67] **Pete Hunt**: *webpack-howto*. <https://github.com/petehunt/webpack-howto>. [Online, accessed 07-19-2015]. 2015.
- [68] **Joel Burget**: *Backbone to React*. <http://joelburget.com/backbone-to-react/>. [Online, accessed 07-19-2015]. 2014.
- [69] **Jafar Husain**: *Beyond the DOM: How Netflix plans to enhance your television experience*. <https://www.youtube.com/watch?v=eNC0mRYGWgc>. [Online, accessed 07-19-2015]. 2015.
- [70] **Chris Wren**: *Dev Chats: Spike Brehm of Airbnb*. <https://medium.com/codestories/dev-chats-spike-brehm-of-airbnb-87e155f3475d>. [Online, accessed 07-19-2015]. 2014.
- [71] **Mozilla Developer Network und individual contributors**: *Web Components*. [https://developer.mozilla.org/en-US/docs/Web/Web\\_Components](https://developer.mozilla.org/en-US/docs/Web/Web_Components). [Online, accessed 07-19-2015]. 2014.
- [72] **Mozilla Developer Network und individual contributors**: *Document Object Model (DOM)*. [https://developer.mozilla.org/en-US/docs/Web/API/Document\\_Object\\_Model](https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model). [Online, accessed 07-19-2015]. 2014.
- [73] **Josep M Sobrepera**: *What is the time complexity for AngularJS's dirty checking algorithm?* <http://stackoverflow.com/questions/27109429/what-is-the-time-complexity-for-angularjss-dirty-checking-algorithm>. [Online, accessed 07-19-2015]. 2014.
- [74] **Facebook Inc.**: *Reconciliation*. <https://facebook.github.io/react/docs/reconciliation.html>. [Online, accessed 07-19-2015]. 2014.
- [75] **Evan Czaplicki**: *Blazing Fast HTML*. <http://elm-lang.org/blog/blazing-fast-html>. [Online, accessed 07-19-2015]. 2014.
- [76] **David Nolen**: *ClojureScript interface to Facebook's React*. <https://github.com/omcljs/om>. [Online, accessed 07-19-2015]. 2014.
- [77] **James Long**: *Presenting The Most Over-Engineered Blog Ever*. <http://jlongster.com/Presenting-The-Most-Over-Engineered-Blog-Ever>. [Online, accessed 07-19-2015]. 2015.
- [78] **Facebook Inc.**: *Overview*. <https://facebook.github.io/flux/docs/overview.html>. [Online, accessed 07-19-2015]. 2014.
- [79] **Wesley M Johnston, JR Hanna und Richard J Millar**: „Advances in dataflow programming languages“. In: *ACM Computing Surveys (CSUR)* 36.1 (2004), S. 1–34.
- [80] **James M. Greene**: *Comparison between different Observer Pattern implementations*. <https://github.com/millermedeiros/js-signals/wiki/Comparison-between-different-Observer-Pattern-implementations>. [Online, accessed 07-19-2015]. 2014.

- [81] **Dan Abramov**: *Predictable state container for JavaScript apps*. <https://github.com/gaearon/redux>. [Online, accessed 07-19-2015]. 2015.
- [82] **Dan Abramov**: *The Evolution of Flux Frameworks*. [https://medium.com/@dan\\_abramov/the-evolution-of-flux-frameworks-6c16ad26bb31](https://medium.com/@dan_abramov/the-evolution-of-flux-frameworks-6c16ad26bb31). [Online; accessed 06-07-2015]. 2015.
- [83] **Paul Irish**: *Why Moving Elements With Translate() Is Better Than Pos:abs Top/left*. <http://www.paulirish.com/2012/why-moving-elements-with-translate-is-better-than-posabs-topleft/>. [Online, accessed 07-19-2015]. 2012.
- [84] *translate3d vs translate vs css left/top vs css margin*. <http://jsperf.com/translate3d-vs-xy/123>. [Online, accessed 07-19-2015]. 2015.
- [85] **Alexis Deveria**: *CSS3 3D Transforms*. <http://caniuse.com/#feat=transforms3d>. [Online, accessed 07-19-2015]. 2015.
- [86] **Kailashkumar V Natda**: „Responsive Web Design“. In: *Eduvantage* 1.1 (2013).
- [87] **Facebook Inc.**: *Tutorial*. <https://facebook.github.io/react/docs/tutorial.html>. [Online, accessed 07-19-2015]. 2014.
- [88] **Stefan Tilkov und Steve Vinoski**: „Node.js: Using JavaScript to build high-performance network programs“. In: *IEEE Internet Computing* 6 (2010), S. 80–83.
- [89] **Aaron “Caustik” Robinson**: *Node.js w/1M concurrent connections!* <http://blog.caustik.com/category/node-js/>. [Online, accessed 07-19-2015]. 2012.
- [90] **Sascha Depold u. a.**: *Sequelize - The Node.js / io.js ORM for PostgreSQL, MySQL, SQLite and MSSQL*. <http://docssequelizejs.com/en/latest/>. [Online, accessed 07-19-2015]. 2011.
- [91] **SWAGGER - The World’s Most Popular Framework for APIs**. <http://swagger.io/>. [Online, accessed 07-19-2015]. 2015.
- [92] **Florian Haupt u. a.**: „A model-driven approach for REST compliant services“. In: *Web Services (ICWS), 2014 IEEE International Conference on*. IEEE. 2014, S. 129–136.
- [93] **Guillermo Rauch**: *Socket.IO-the cross-browser WebSocket for realtime apps*. <http://socket.io/>. [Online, accessed 07-19-2015]. 2013.

# Abbildungsverzeichnis

1.1	Eine <i>Social Media Wall</i> vom Anbieter walls.io [8] . . . . .	4
2.1	Links das Rasterbasierte Layout von Windows 8 [21], rechts das spaltenbasierte Layout von Pinterest. . . . .	6
2.2	Spaltenbasierte Anordnung der Inhalte . . . . .	7
2.3	Links wird eine neue Kachel eingefügt die den höchsten Wert besitzt von allen Kacheln, rechts ist das Ergebnis bei Beibehaltung der Sortierung von links nach rechts. . . . .	8
2.4	links: Das Raster des Grids, rechts: eine zufällige Benutzung des Grids . .	8
2.5	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 durchgeführt wurde. . . . .	9
2.6	Ein Wireframe mit rein funktionalen Aspekten. . . . .	11
2.7	Ein erstes Design, dass verschiedene Darstellungsformen für unterschiedliche Typen von Informationen benutzt. . . . .	11
2.8	Farbskalen, die mit verschiedenen Farbräumen erstellt wurden. . . . .	13
2.9	Die Punkte des Benutzers aufgeteilt in Kategorien. . . . .	15
2.10	Die Benutzerzentrische <i>lass-dich-nicht-entmutigen</i> Bestenliste, hier sieht sich der Benutzer immer in der Mitte der Liste. . . . .	17
2.11	Die Benutzeransicht für die erhaltenen Abzeichen. Der hier gezeigte Benutzer besitzt die Favoriten-Abzeichen in allen Ausführungen. . . . .	18
2.12	Um den Benutzer über das Erreichen eines neuen Levels oder den Erhalt eines Abzeichens zu informieren, werden Flashmessages benutzt. . . . .	19
3.1	Übertragungszeit der benötigten Dateien . . . . .	22
3.2	Benötigte Ausführungszeit der Skriptdatein . . . . .	23
3.3	Geschwindigkeitsvergleich von TodoMVC-Implentierungen mit verschiedenen Frameworks [75]. . . . .	27
3.4	Dispatcher, Stores und Views sind unabhängige Knoten mit unterschiedlichen Ein und Ausgaben. Die Aktionen sind einfache Objekte die den Typ und die Daten selbst enthalten. . . . .	28
3.5	Mithilfe von Actions ist die View in der Lage, den Zustand der Applikation, zu verändern. . . . .	29
3.6	Die wichtigsten Bestandteile der Applikation. . . . .	30
3.7	Links eine Kachel mit geraden Werten bei translate3D, rechts mit jeweils .5 bei x/y. . . . .	33

## ABBILDUNGSVERZEICHNIS

---

4.1	Auf dieser Seite kann der Benutzer verschiedene Booster erwerben und sehen, wie lange der Aktuelle noch aktiv ist. . . . .	xiv
4.2	Auf dieser Seite kann der Benutzer einsehen, wie viele Punkte er hat, wie sie aufgeteilt sind, welche Abzeichen er besitzt und welche Personen direkt über oder unter ihm auf der Bestenliste stehen. . . . .	xv
4.3	Auf dieser Seite kann der Benutzer sämtliche Parameter für das Info-board einstellen. Im Moment beschränkt sich dies auf die Auswahl des Farbschemas, welche jeweils erst nach dem erreichen eines bestimmten Levels verfügbar sind. . . . .	xvi
4.4	Die Gesamtstatistikseite. Um den Effekt der Gamification zu erhöhen, sollte diese an einem öffentlichen Ort des Betriebs permanent gezeigt werden. . . . .	xvi
4.5	Dies ist die Hauptapplikation, hier kann der Benutzer nach verschiedenen Begriffen suchen, die Suchergebnisse bewerten/favorisieren und sehen, wer welche Aktion als letztes auf ein Suchergebnis ausgeführt hat. .	xvii
4.6	Das benutzte Datenmodell der Applikation. Ein Item ist ein Suchergebnis, ein Vote eine Bewertung, und eine Action alle vom Benutzer ausgeführten Aktionen. . . . .	xviii

# Abkürzungsverzeichnis

PDF	Portable Document Format
RGB	Rot Grün Blau (Farbraum)
HSL/HLS	Hue, Saturation, Luminance
HSV	Hue, Saturation, Value
Lab/CIELAB	Farbraum der Internationalen Beleuchtungskommission
HCL/CIELUV	Farbraum der Internationalen Beleuchtungskommission
HUSL	human-friendly HSL
GWT	Google Web Toolkit
ECMA	European Computer Manufacturers Association
ES	ECMAScript
CSS	Cascading Style Sheets
DOM	Document Object Model
MVC	Model View Control
ORM	Object Relationship Mapper
HTTP	Hypertext Transfer Protocol

# Anlagen – Inhalte der beiliegenden CD

- Quelltext

# Appendix

## Abbildungen

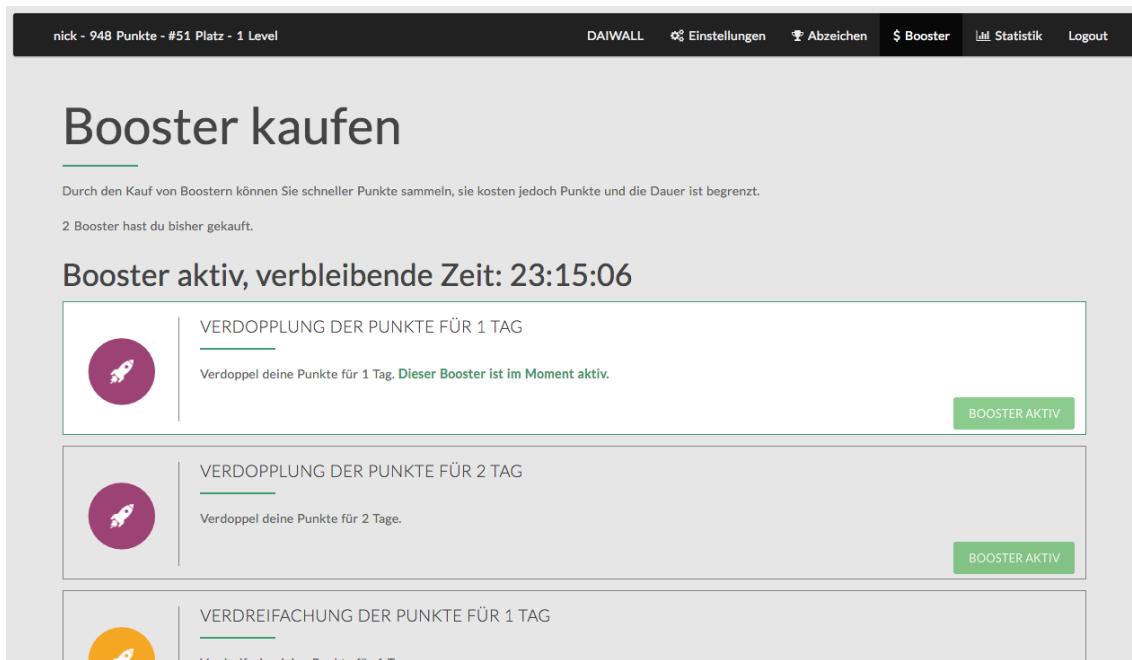


Abbildung 4.1: Auf dieser Seite kann der Benutzer verschiedene Booster erwerben und sehen, wie lange der Aktuelle noch aktiv ist.

## ABBILDUNGSVERZEICHNIS

nick - 948 Punkte - #51 Platz - 1 Level

DAIWALL Einstellungen Abzeichen \$ Booster Statistik Logout

# 948 Punkte / 1 Level

52 Punkte benötigst du um das nächste Level zu erreichen.

Abzeichen - 1200

Login - 320

Favorisieren - 112

Bewerten - 59

Suchen - 29

## Abzeichen

Hier Findest du alle Trophäen die du bekommen hast, durch jede Trophäe werden dir Punkte auf deinen Punktestand gutgeschrieben.  
8 hast du bisher gesammelt.

 <b>10 FAVORITEN</b> Favorisiere mindestens nach 10 Suchergebnisse um diese Trophäe zu erhalten <b>100</b> hast du für dieses Abzeichen erhalten.	 <b>50 FAVORITEN</b> Favorisiere mindestens nach 50 Suchergebnisse um diese Trophäe zu erhalten <b>300</b> hast du für dieses Abzeichen erhalten.
 <b>10 SUCHEN</b> Suche mindestens nach 10 Suchbegriffen um diese Trophäe zu erhalten <b>50</b> hast du für dieses Abzeichen erhalten.	 <b>10 UPVOTES</b> Bewerte mindestens 10 Suchergebnisse positiv um diese Trophäe zu erhalten. <b>50</b> hast du für dieses Abzeichen erhalten.
 <b>100 UPVOTES</b> Bewerte mindestens 100 Suchergebnisse positiv um diese Trophäe zu erhalten. <b>300</b> hast du für dieses Abzeichen erhalten.	 <b>1 ANMELDUNG</b> Melde dich einmal an der DAI-Wall an um diese Trophäe zu erhalten. <b>50</b> hast du für dieses Abzeichen erhalten.
 <b>10 DOWNVOTES</b> Bewerte mindestens 10 Suchergebnisse negativ um diese Trophäe zu erhalten. <b>50</b> hast du für dieses Abzeichen erhalten.	 <b>100 DOWNVOTES</b> Bewerte mindestens 100 Suchergebnisse negativ um diese Trophäe zu erhalten. <b>300</b> hast du für dieses Abzeichen erhalten.

## Bestenliste

### Aller Zeiten

Platz	Name	Punkte	Abzeichen
#47	Joye	1002	6
#48	Micki	995	6
#49	Sarina	985	6
#50	Ashly	956	6
<b>#51</b>	Nick	948	<b>8</b>
#52	Sarina	833	5
#53	Jenae	826	5

### Dieser Monat

Platz	Name	Punkte	Abzeichen
#47	Joye	1002	6
#48	Micki	995	6
#49	Sarina	985	6
#50	Ashly	956	6
<b>#51</b>	Nick	948	<b>8</b>
#52	Sarina	833	5
#53	Jenae	826	5

Abbildung 4.2: Auf dieser Seite kann der Benutzer einsehen, wie viele Punkte er hat, wie sie aufgeteilt sind, welche Abzeichen er besitzt und welche Personen direkt über oder unter ihm auf der Bestenliste stehen.

The screenshot shows the 'Einstellungen' (Settings) page of the DAI-Wall. At the top, there's a navigation bar with links for 'DAIWALL', 'Einstellungen', 'Abzeichen', 'Booster', 'Statistik', and 'Logout'. Below the navigation, the title 'Einstellungen' is displayed. A note below the title says: 'Hier können Sie permanente Einstellungen an der DAI-wall vornehmen. Klicken Sie auf speichern um die Änderungen zu übernehmen.' Under the title, there's a section titled 'Farbschemas' (Color Schemes). It contains three examples: 'Pastel' (with a grid of pastel-colored squares), 'Random RGB' (with a grid of random colored squares), and 'Pastel HUSL' (with a grid of pastel-colored squares). Each example has a 'BENUTZEN' (Use) button.

Abbildung 4.3: Auf dieser Seite kann der Benutzer sämtliche Parameter für das Infoboard einstellen. Im Moment beschränkt sich dies auf die Auswahl des Farbschemas, welche jeweils erst nach dem erreichen eines bestimmten Levels verfügbar sind.

The screenshot shows the main statistics page. At the top, there's a navigation bar with links for 'DAIWALL', 'Einstellungen', 'Abzeichen', 'Booster', 'Statistik', and 'Logout'. Below the navigation, the title 'Bestenliste' (Best List) is displayed. To its right, the total points '106653 Punkte' are shown. Further right, the section 'Aktionen' (Actions) is displayed, showing a list of recent actions with icons and counts: 'Abzeichen - 52800' (Search icon), 'Login - 30445' (Orange square), 'Favorisieren - 12006' (Green square), 'Bewerten - 6181' (Blue square), and 'Suchen - 3007' (Blue square). Below these sections, there are two tables: 'Aller Zeiten' (All Time) and 'Dieser Monat' (This Month), both listing user names, their scores, and the number of achievements.

Abbildung 4.4: Die Gesamtstatistikseite. Um den Effekt der Gamification zu erhöhen, sollte diese an einem öffentlichen Ort des Betriebs permanent gezeigt werden.

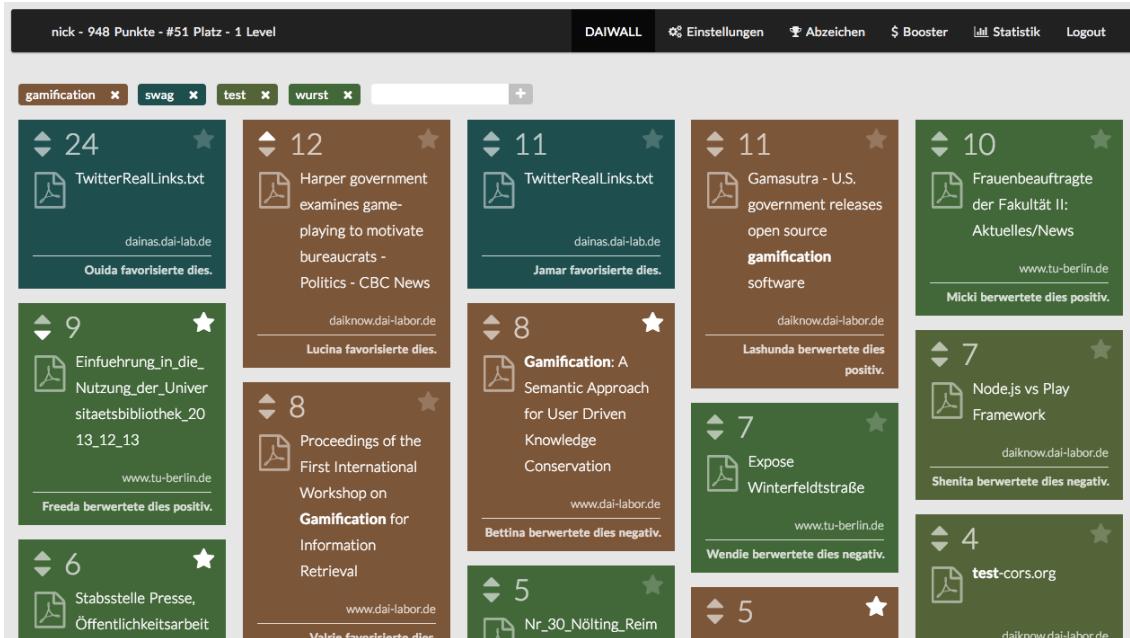


Abbildung 4.5: Dies ist die Hauptapplikation, hier kann der Benutzer nach verschiedenen Begriffen suchen, die Suchergebnisse bewerten/favorisieren und sehen, wer welche Aktion als letztes auf ein Suchergebnis ausgeführt hat.

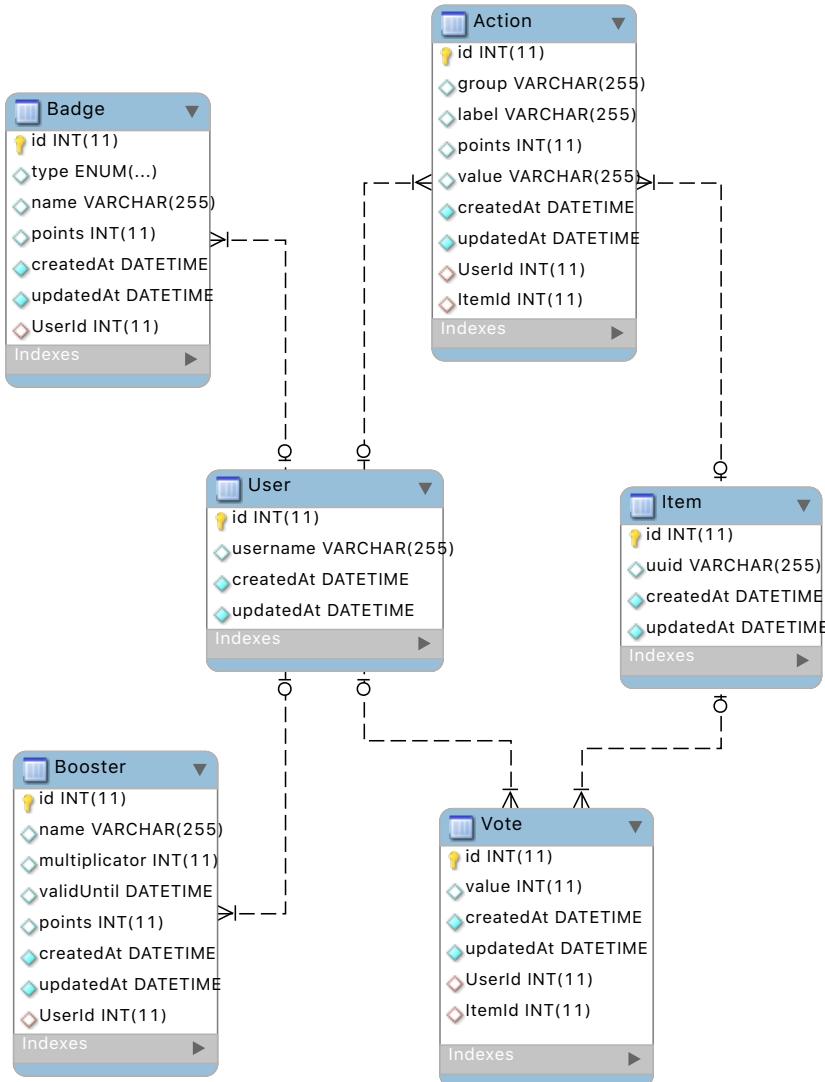


Abbildung 4.6: Das benutzte Datenmodell der Applikation. Ein Item ist ein Suchergebnis, ein Vote eine Bewertung, und eine Action alle vom Benutzer ausgeführten Aktionen.

## 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 Abbildung 4.5 gesehen werden, ich werde nun Schritt für Schritt deren Implementierung angehen.

1. Die `Queries` Komponente soll den aktuellen Stand der `Queries` anzeigen, dieser wird vom `QueryStore` verwaltet, weswegen wir seine Veränderungen abonnieren, 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 Themen und erstellt für jeden eine Query-Komponente. In der `render`-Methode wird sie dann aufgerufen.

4. Bleibt zuletzt noch das Löschen von Themen, 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>

```
);  
}  
});
```