DHBW Mannheim

Team „Norbert“

Norbert - Your StudyBuddy

Softwareentwurf

4. April 2016

Projektleitung: Arwed Mett

Projektmitglieder: Dominic Steinhauser, Tobias Dorra, Simon Oswald, Philipp Pütz

Inhaltsverzeichnis

1 Grundlegende Komponenten und deren Funktionsweise

Folgende Funktionalitäten müssen von der Architektur zwingend erfüllt werden, damit Norbert korrekt funktionieren kann:

1. Der Newsfeed muss auf dem Client angezeigt werden können.
2. Norbert muss die gesamten Daten (Einträge, Vorschläge, Emails. etc.) zentral speichern.
3. Die für den Newsfeeds relevanten Daten müssen von Norbert an die Clients geliefert werden können.
4. Der Client muss Einträge anlegen/bearbeiten können und auf Vorschläge reagieren können, bzw. seine Aktionen Norbert mitteilen können.
5. Norbert muss Vorschläge liefern können.

1.1 Grundlegender Aufbau aus Funktionssicht

Die in Grafik 1.1 gezeigte Architektur würde die oben genannten Kriterien erfüllen.

1. Über einen Webbrowser kann der Newsfeed auf den Clients angezeigt werden.
2. Norbert hat die gesamten Daten in getrennten Dateien/Ordnern zur Verfügung.
3. Über die Kommunikationsschnittstelle kann Norbert die relevanten Daten an den Client liefern.
4. Über die Kommunikationsschnittstelle können die Aktionen der Clients Norbert mitgeteilt werden.
5. Die Vorschlagslogik berechnet anhand der vorhandenen Daten Vorschläge, welche über die Kommunikationsschnittstelle an den Client geliefert werden.

Mit dieser Architektur treten aber mehrere Probleme auf:

1. Das manuelle Verwalten von Daten in separaten Dateien ist sehr aufwendig und extremst fehleranfällig.

2. Der genaue Ablauf

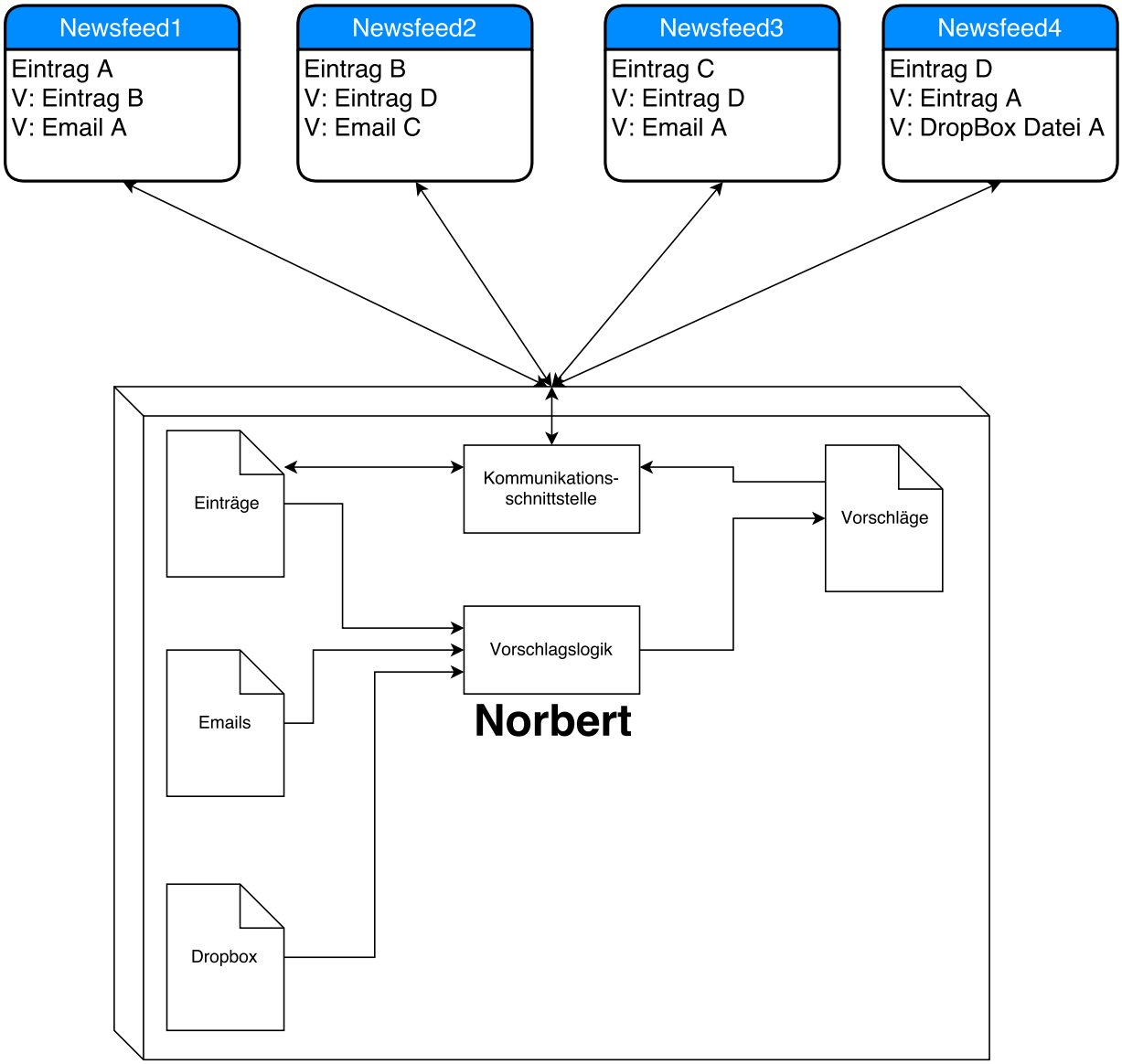

Abbildung 1.1: Grundlegende Funktionalität

1.2 Datenhaltung

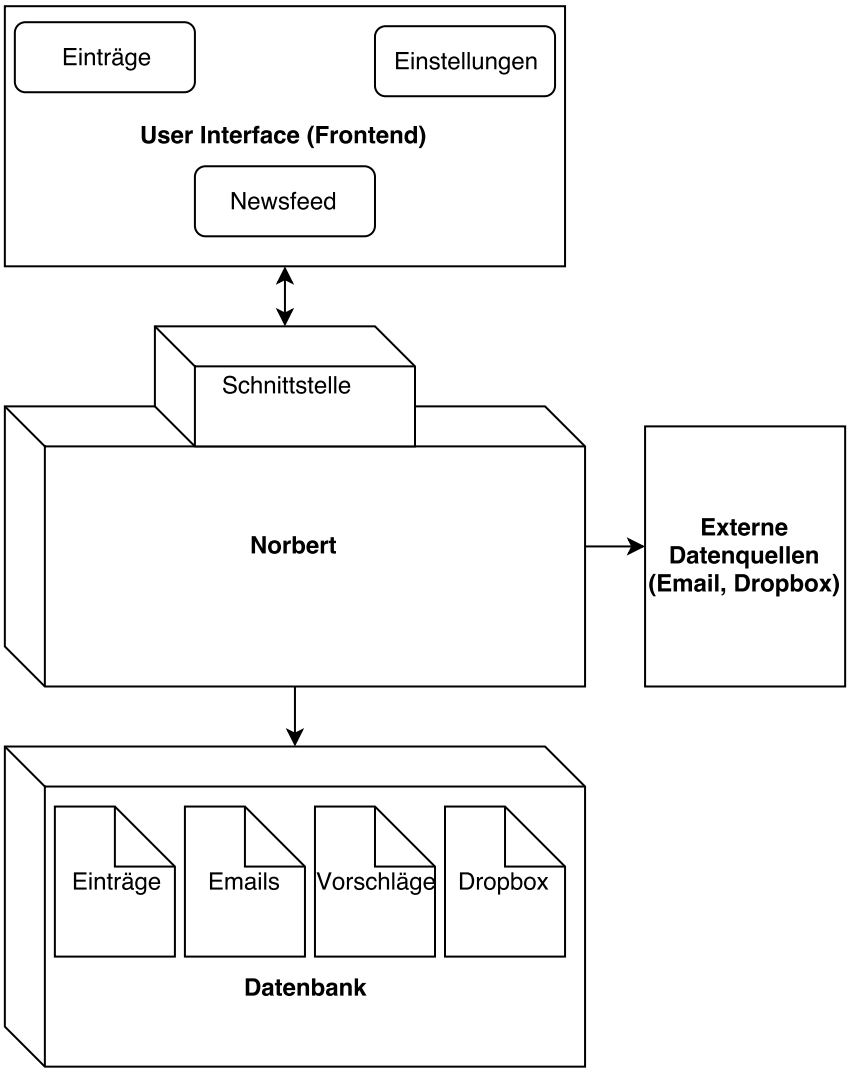

Abbildung 1.2: Kapselung der Daten in einer Datenbank

1.3 Interne Prozesse

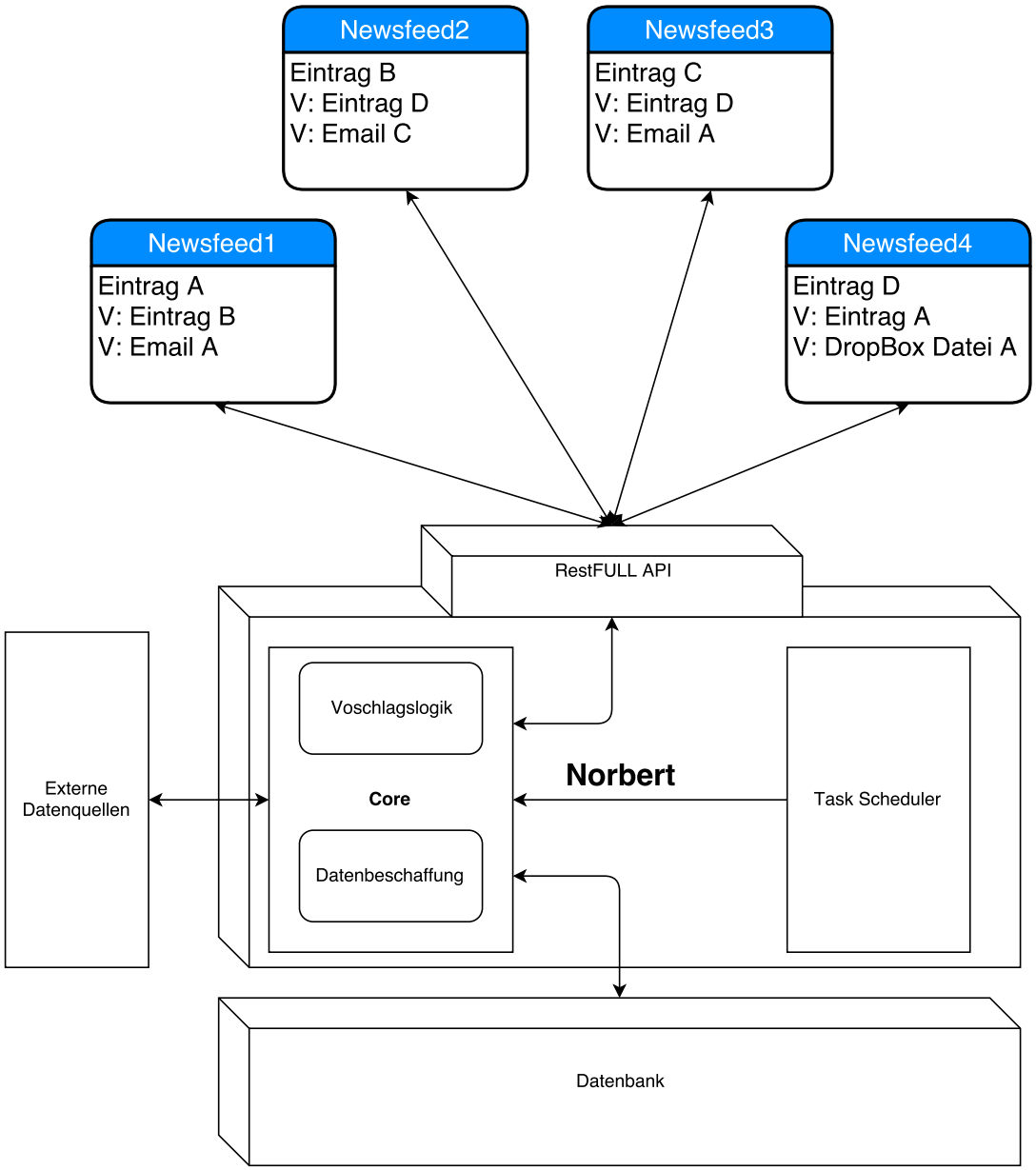

Abbildung 1.3: Kapselung der internen Prozesse

2 Anbindung externer Dienste - Dropbox

Eines der Ziele von Norbert - Your StudyBuddy ist es Wissensmanagement zu betreiben und somit Wissen zu sammeln und weiterzugeben. Da Informationen und Wissen von Studierenden meistens in Dropbox-Ordner abgelegt werden, wird in diesem Kapitel die technische Anbindung von Dropbox an Norbert beschrieben. Dabei wird zunächst erst allgemein auf den standardisierten Autorisiuerngsprozess „OAuth“ eingegangen und daraufhin die verschiedenen Dropbox Endpoints mit ihren Schnittstellen vorgestellt.

2.1 Autorisierungsprozess (OAuth)

OAuth ist ein offenes Protokoll, dass eine standardisierte und sichere API-Autorisierung für mobile Endgeräte, Webanwendungen und Desktop-Applikationen ermöglicht. Der Benutzer kann über den OAuth-Autorisierungsprozess (Abbildung 2.1) einer Anwendung (in diesem Fall Norbert) Zugriff auf gespeicherte Daten erlauben. Dabei loggt sich der Benutzer über die Applikation (Norbert) die Zugriff benötigt, auf dem Dienst, der die freizugebenden Daten speichert, ein (hier Dropbox) und erlaubt der Applikation den Zugriff. Da nicht jede Severinstanz von Norbert über eine Redirect-URL, die nach einem erfolgreichen Login aufgerufen wird, für den OAuth-Prozess verfügt und nicht jeder Severadministrator in der Dropbox Developer Console Norbert - Your StudyBuddy eintragen möchte, wird statt einem Redirect ein Autorisationscode angezeigt, den der Nutzer in Norbert eingeben kann. Norbert kann dann mit diesem Autorisationscode ein wiederverwendbares Access-Token anfragen. Dieses Access-Token dient zur Authentifizierung des Benutzers, sodass kein Benutzername und keine Passwörter ausgetauscht oder gespeichert werden müssen.

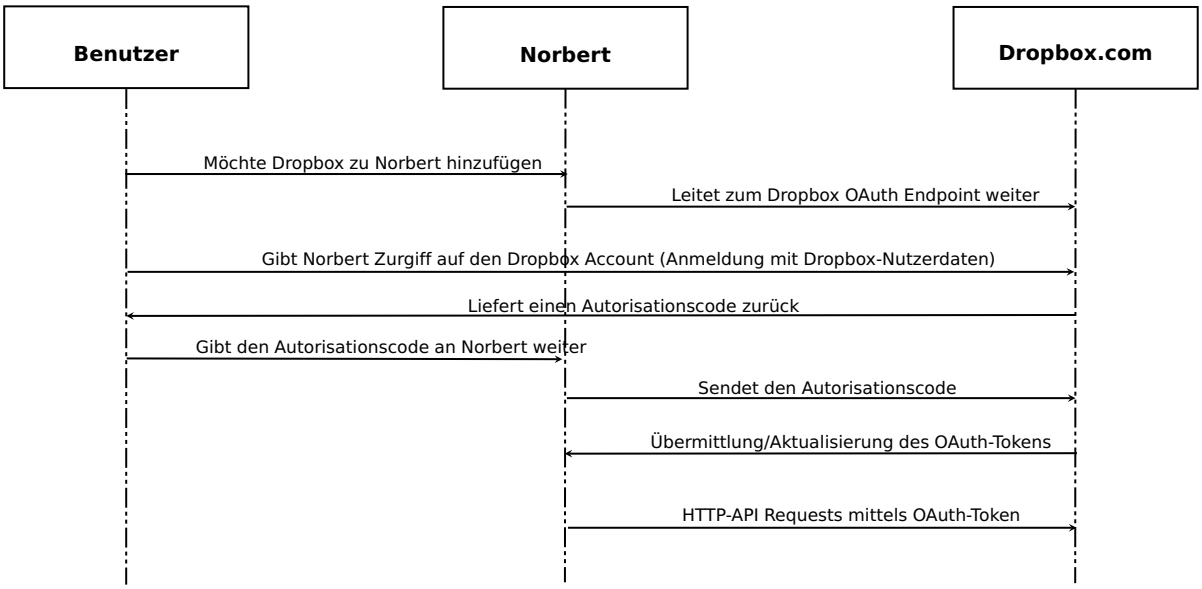

Abbildung 2.1: OAuth-Autorisierungsprozess zwischen Norbert und Dropbox.com

2.2 Dropbox Endpoints

Die Dropbox HTTP-API besteht aus mehreren Endpoints (Abbildung 2.2), die jeweils unterschiedliche Funktionen bereitstellen. Alle API-Aufrufe benötigen zu Autorisierung das individuelle Access-Token des Benutzers.

1. *dropbox.com:* Über diesen Endpoint/Webseite wird die OAuth-Autorisierung durchgeführt. Es wird nur einmalig zur Generierung des Access-Tokens ein Kontakt zu diesem Endpoint hergestellt.
2. *api.dropboxapi.com (RPC-Endpoint):* Über die Domain api.dropboxapi.com können RPCs (remote procedure calls) ausgeführt werden. Dabei nimmt dieser Endpoint JSON-Strings im HTTP-Body entgegen und führt je nach aufgerufener URL verschiedene Funktionen aus. Ein RPC-Funktion ist beispielsweise list_folder (*https://api.dropboxapi.com/2/files/list_folder*), die als Parameter einen Pfad erwartet und dann alle im entsprechenden Ordner sich befindenden Dateien auflistet. Das Ergebnis der Funktion wird daraufhin wieder im JSON-Format zurückgesendet.
3. *content.dropboxapi.com (Download/Upload-Endpoint):* Wie der Name schon sagt, können über den Download/Upload-Endpoint Dateien heruntergeladen oder hochgeladen werden. Die benötigten Meta-Daten (Pfad, Datei ID etc.) werden dabei im HTTP-Header als JSON oder URL-Argument übergeben. Funktionsergebnisse werden im JSON-Format im HTTP-Response Header übergeben und Dateien im HTTP-Body übersendet.

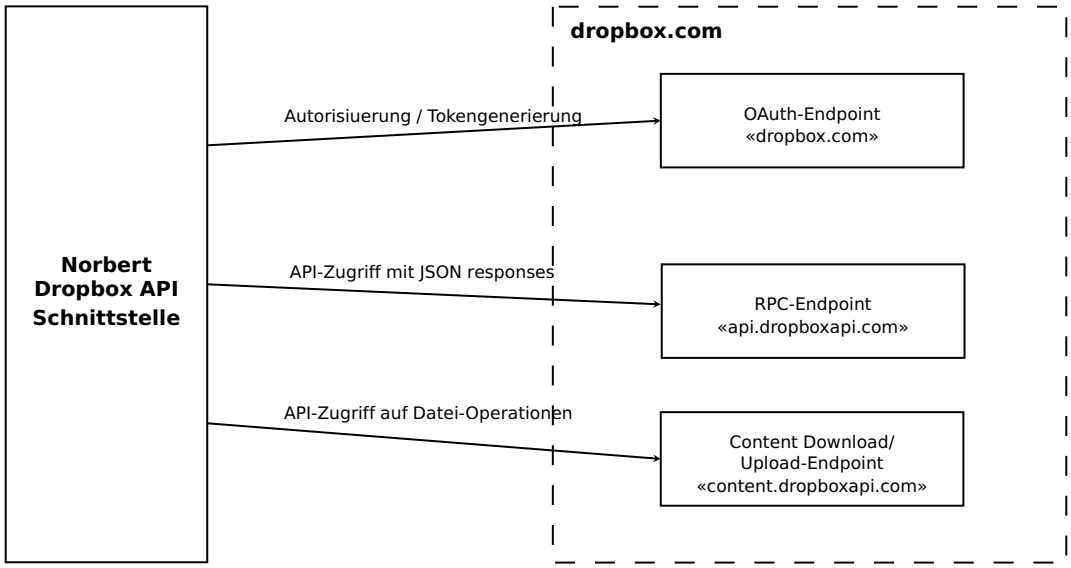

Abbildung 2.2: API-Zugriffe über die Dropbox-Endpoints

3 Frontend

Da die Anwendung auf mehreren Plattformen laufen soll wird die Applikation als HTML5 App programmiert. Dabei besteht das Frontend aus folgenden Views:

ID-Kürzel	Beschreibung
F-10	Newsfeed
F-20	User Login
F-30	User Registrieren
F-40	Profilverwaltung und Einstellungen
F-50	Detailansicht von Einträgen
F-60	Adminansicht

Tabelle 3.1: Einsatzbereiche

Wie zu sehen ist besteht das Frontend aus sehr wenigen „Views“ und der Benutzer muss immer im Newsfeed oder dem Login landen, wenn er die Seite neu lädt. Es muss also kein komplexes Routing System implementiert werden, welches Routen in der Url speichern kann. Allerdings ist es wichtig, dass sich die Anwendung einen „Globalen Zustand“ teilt, damit sobald sich z.B. etwas in den Einstellungen ändert diese direkt auf den Rest der Anwendung angewandt wird. Oder sobald der User einen Eintrag in der Detailansicht bearbeitet, soll dieser im Newsfeed geändert werden.

3.1 React.js als View der Anwendung

React.js soll in Kombination mit Flux für das Frontend eingesetzt werden. React.js wird dazu für die Erstellung der Komponenten oder der Views verwendet und Flux dient als Architektur der Anwendung.

Der Vorteil von React.js ist, dass die Anwendung in einzelne Komponenten unterteilt werden kann. Dies erleichter die Entwicklung der Anwendung im Team.

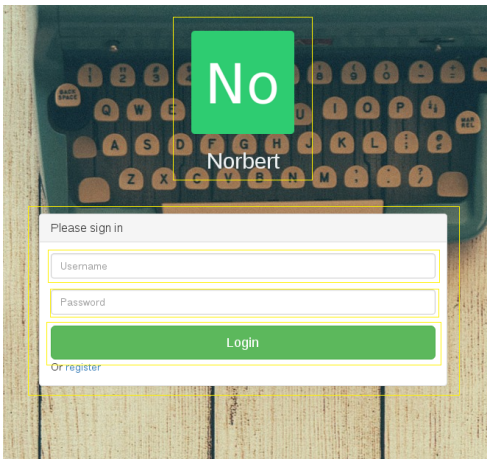

Abbildung 3.1: Beispiel Komponenten

In dem in Abbildung 3.1 gezeigten beispiel können alle gelb umramten Teile Komponenten sein. So kümmert sich die Formular Komponente um das absenden der Request sobald der User versucht sich einzuloggen. Andere Komponten wie z.B. Text oder Icons sind lediglich als View gedacht.

3.2 Flux als Architekturpattern

Ein großer Nachteil vieler Kleiner Komponenten in einer Anwendung ist es, dass diese sich einen Zustand könnten, welcher synchronisiert werden muss. Dazu wird Flux als ein Architektur Pattern eingesetzt. Die Grundbestandteile dieser Architektur sind Komponenten, Storages, Actions und ein Dispatcher. Dabei dienen die Komponenten zur Interaktion mit dem User, stellen somit also den „View“ der Anwendung dar. Die Komponenten können Actions auslösen, welche einfache Funktionen sind die ausgeführt werden. Diese Actions senden ein Objekt mit einer ID an den Dispatcher, welcher die Objekte an alle Stores verteilt. Ein Store ist wie der Name schon sagt ein Ort zum lagern von Daten. Jeder Store stellt eine Methode „handleAction“ zur Verfügung. Diese wird von dem Dispatcher aufgerufen und das Objekt der Action übergeben. Anschließend kann der Store aufgrund der in der Action definierten ID seine Daten aktualisieren. Jetzt können sich Komponenten bei dem Store registrieren um über Änderungen informiert zu werden. Geschieht dies wird der View der Komponente erneuert. [1]

[1] Vgl. https://facebook.github.io/flux/

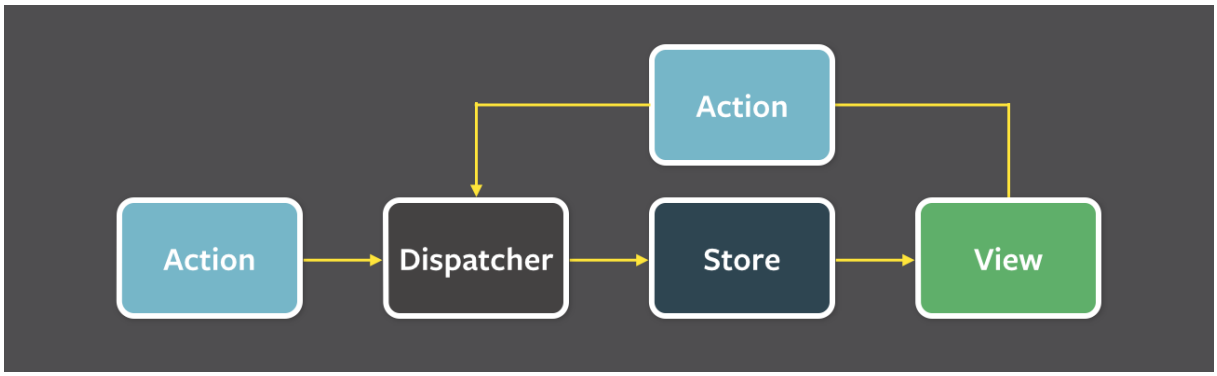

Abbildung 3.2: Flux

https://facebook.github.io/flux/docs/actions-and-the-dispatcher.html#content

Diese Architektur unterstütz das Teilen eines Zustandes mehrerer Komponenten, da sich mehrere Komponenten auf einen Storage registrieren und Änderungen an dem Zustand der Anwendung immer global über eine Action ausgeführt werden. Somit können alle Stores ihre Daten bei spezifischen Aktionen aktualisieren. Dadurch wird vermieden, dass sobald sich ein Eintrag ändert eine extra Methode geschrieben werden muss welche den Newsfeed ändert.

4 Datenbank

Um die anfallenden Daten zentral speichern zu können, wird eine Datenbank benötigt. Als Datenbankmanagementsystem wird MongoDB verwendet. Die Gründe dafür sind, dass der dokumnetenorientierte Ansatz von MongoDB sehr gut mit Javascript harmoniert und das Abspeichern von Entries sehr einfach macht.

MongoDB speichert die Daten in verschiedenen Collections. Diese nehmen eine ähnliche Rolle ein wie die Tabellen bei relationalen Datenbankmanagementsystemen. Eine Collection ist eine Sammlung von Objekten, die einen beliebigen Aufgau haben dürfen.

4.1 Datenbankschema

Die folgenden Collections werden von Norbert verwendet:

1. **Users**: Speichert die Benutzer von Norbert und ihre Zugangsdaten.
2. **Entries**: Speichert die Entries, die von den Benutzern erzeugt wurden.
3. **Informations**: Speichert die Informationen, die aus externen Systemen importiert wurden.
4. **Reccomendations**: Speichert die generierten Vorschläge.
5. **Tags**: Vom Nutzer vergebene Tags für Entries.

Users Jedes Objekt in der Users-Collection repräsentiert einen Benutzer von Norbert. Ein solches Objekt ist nach dem folgenden Schema aufgebaut:

```
{
  "id": 42,                 # Eindeutige ID

  "username": "benni",      # Der Benutzername,
                            # der zur Anmeldung genutzt wird.

  "name": "Ben Utzer",      # Der volle Name des Benutzers

  "password_hash": "xdjc", # Das gehashte Passwort
```

```
  "password_salt": "cjdx"  # und der Salt, der zum hashen des Passworts
                           # verwendet wurde.
}
```

Entries Jeder Entry wird als ein Objekt in der Entries-Collection repräsentiert. Ein solches Objekt besteht aus einem Titel, diversen Metadaten und einer Liste mit Components. Ein Entry-Objekt ist nach dem folgenden Schema aufgebaut:

```
{
  "id": 123,                # Eindeutige ID

  "created_at": 1234,       # Unix-Timestamp des Erstellzeitpunktes
                            # des Entries.

  "owned_by": [42, 43],     # Ein Array mit den IDs der Benutzer,
                            # denen der Eintrag gehört.
                            # Alle diese Benutzer sehen den Eintrag in
                            # ihrem Newsfeed und können ihn bearbeiten.

  "private": false,         # Wenn der Eintrag als privat markiert
                            # wurde, kann er von anderen Benutzern
                            # nicht über die Suche gefunden werden
                            # und wird niemandem vorgeschlagen.

  "equality_group": 123,    # Bei neuen Entries wird der Wert
                            # "equality_group" gleich der id des
                            # Entries gesetzt. Wird der Entry beim
                            # akzeptieren einer Reccomendation geklont,
                            # so hat der neue Eintrag zwar eine neue id,
                            # behält jedoch die equality_group.
                            # Die so entstehenden Äquivalenzklassen an
                            # ähnlichen Entries werden dafür genutzt,
                            # in der Suche und bei den Vorschlägen
                            # duplikate zu vermeiden.

  "title": "Überschrift",   # Der Titel des Entries

  "tags": [51, 32],         # Ein Array mit den IDs der Tags, die der
                            # Benutzer für den Entry vergeben hat.

  "components": [           # Ein Array mit den Komponenten des Entry
    { /* Component 1 */ },
```

```
35      { /* Component 2 */ },
36      { /* Component 3 */ }
37    ]
38  }
```