DHBW Mannheim

Team „Norbert“

Norbert - Your StudyBuddy

Softwareentwurf

5. 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.

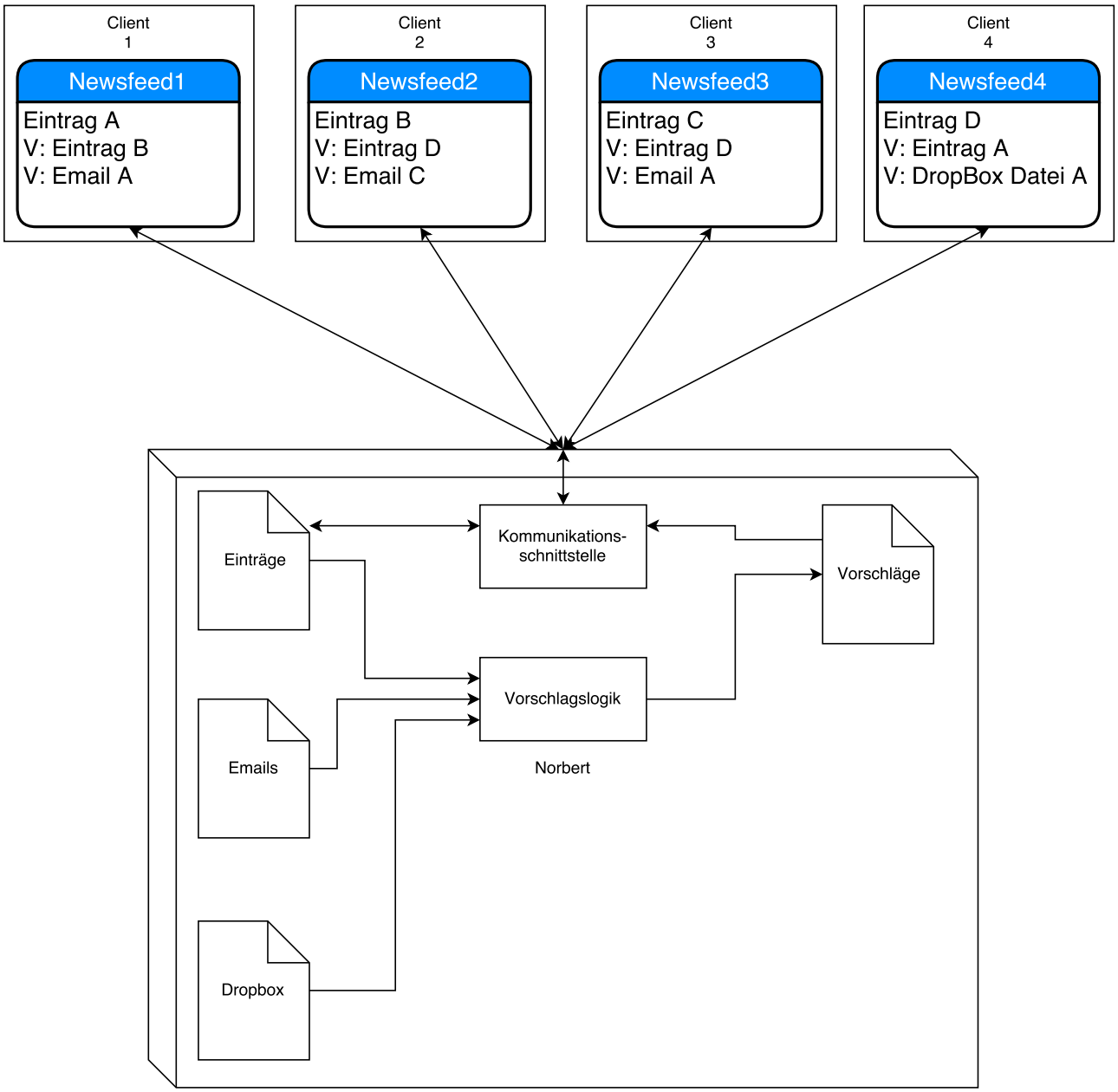

Abbildung 1.1: Grundlegende Funktionalität

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 Ablauf interner Prozesse ist nicht definiert und es ist unklar, wie dieser gesteuert wird

1.2 Datenhaltung

Das erste Problem kann durch die Verwendung eines Datenbanksystems gelöst werden. Durch eine Datenbank können die Daten zentral verfügbar gemacht werden. Außerdem ist der Zugriff über ein Datenbanksystem weniger fehleranfällig und in der Regel auch effizienter, da die meisten Datenbanksysteme bereits von Haus aus Performanceoptimierungen wie Puffer etc. mitbringen.

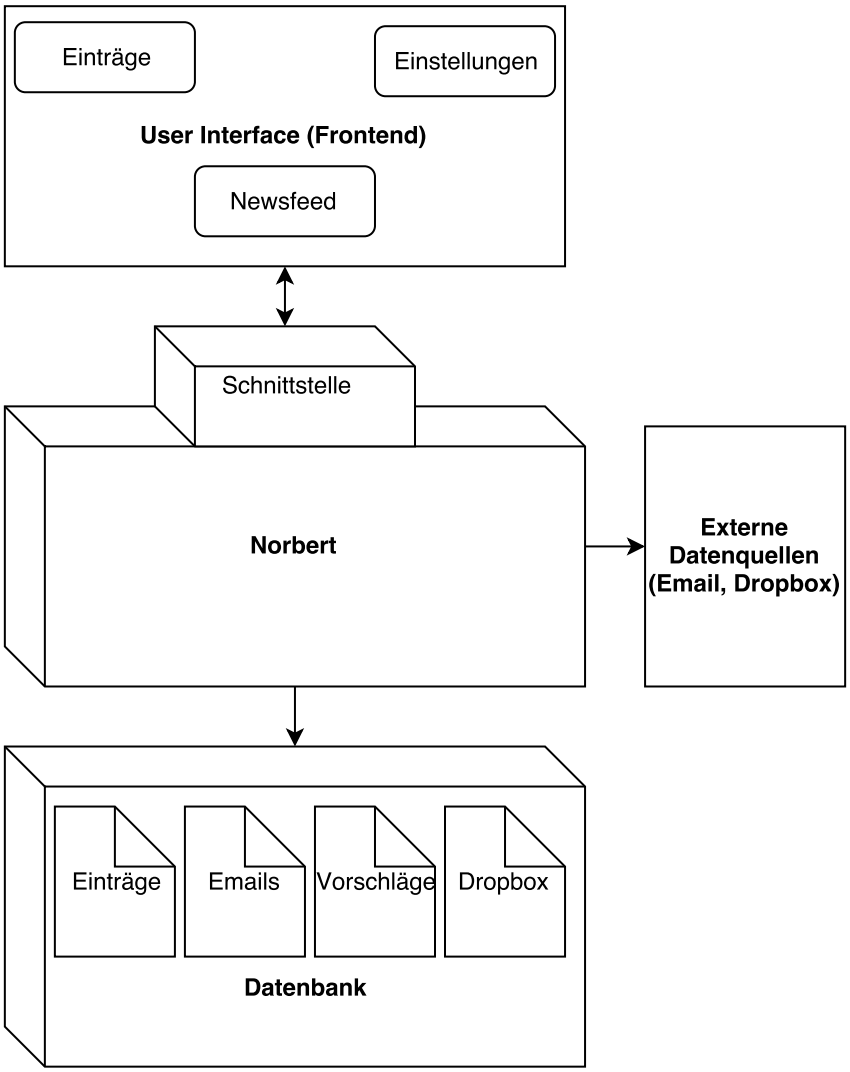

Abbildung 1.2: Kapselung der Daten in einer Datenbank

1.3 Interne Prozesse

Das zweite Problem kann durch eine Komponente zur Aufgabenverwaltung gelöst werden. Anstatt Aufgaben wahllos in Echtzeit zu bearbeiten, werden diese zunächst durch die Aufgabenverwaltung in eine Warteschlange eingereiht. Diese Liste an Aufgaben wird dann in regelmäßigen Abständen durch den Core, also die zentrale Logik- und Rechenkomponente, abgearbeitet. Ausnahme hierbei ist das Liefern von Daten an den Client, da dies möglichst schnell erfolgen sollte. Durch das Sammeln von Aufgaben und zyklisches Verarbeiten kann unter anderem die Anzahl seperater Schreibzugriffe auf die Datenbank reduziert werden, wodurch weniger Ressourcen verbraucht werden. Die in Abbildung 1.3 gezeigte Architektur erfüllt die genannten Anforderungen und löst die in Abbildung 1.1 auftretenden Probleme. Die genauere Funktionsweise einzelner Komponenten wird in den folgenden Kapiteln beschrieben.

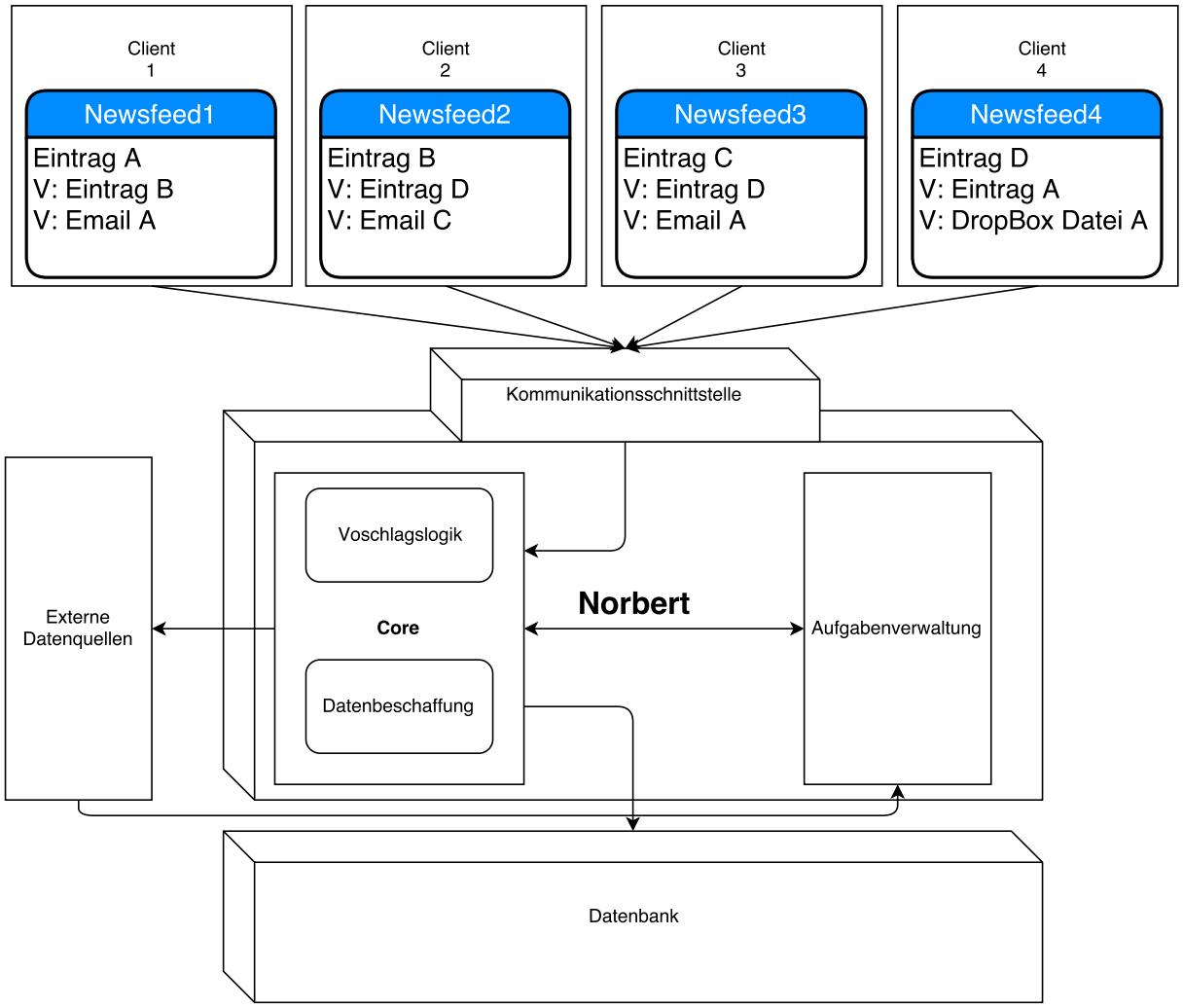

Abbildung 1.3: Kapselung der internen Prozesse

2 Frontend

2.1 Ziele

ID-Kürzel	Beschreibung
FZ-10	Einzelne Teile des Frontends können getrennt voneinander entwickelt werden.
FZ-20	Es gibt eine Trennung von Daten- und Anzeigelogik geben.
FZ-30	Das Frontend ist leicht um neue Funktionen erweiterbar.
FZ-40	Das Frontend ist responsiv gestaltet.

Tabelle 2.1: Ziele der Frontend-Architektur

2.2 Views

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	Route
F-10	Newsfeed	`/`
F-20	User Login	`/login`
F-30	User Registrieren	`/register`
F-40	Profilverwaltung und Einstellungen	`/profile`
F-50	Detailansicht von Einträgen	`/`
F-60	Adminansicht	`/administration`

Tabelle 2.2: Einsatzbereiche

2.3 Newsfeed

Der Newsfeed ist eine Kollektion aus Einträgen und Vorschlägen (Informationen). Diese werden nach Relevanz soritert. Dabei kann das Frontend die Einträge abfragen. Der Server ist verantwortlich für die Sortierung der Einträge.

2.3.1 Bearbeiten des Newsfeeds

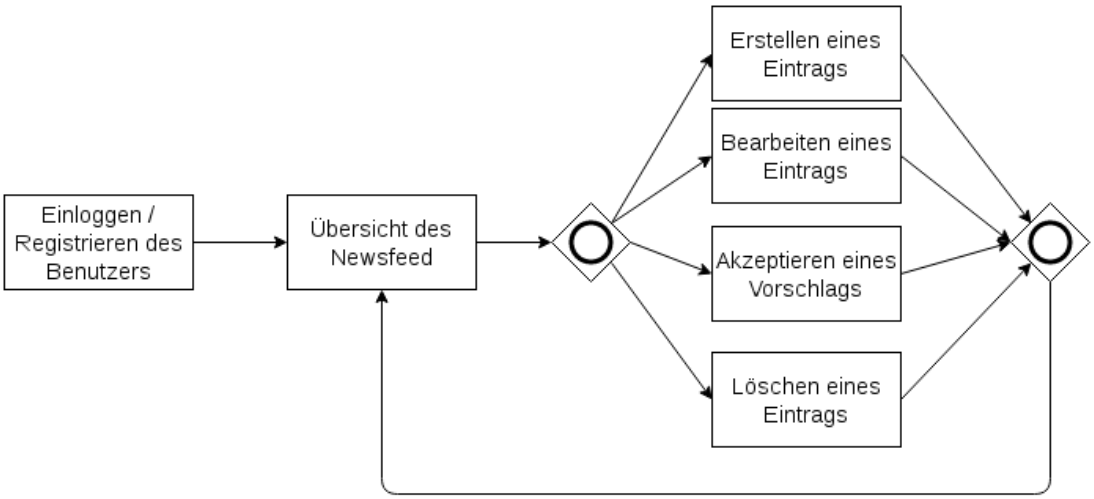

Abbildung 2.1: Ablauf von Dialogen

Ein Benutzer muss sich immer Authentifizieren bevor er seinen Newsfeed sieht. Anschließend erhält er eine Übersicht seiner Einträge. Jetzt hat er die Möglichkeit Einträge anzulege, verändern, löschen oder einen Vorschlag in seinen Newsfeed aufzunehmen. Wenn ein Eintrag erstellt oder bearbeitet wird öffnet sich ein Dialog in dem der Entsprechende Eintrag bearbeitet werden kann. Anschließend wird der neue Newsfeed dargestellt.

2.4 Profilverwaltung und Einstellungen

In der Profilverwaltung und den Einstellungen kann der Benutzer sein Passwort, Username oder Email ändern. Außerdem kann eingestellt werden, ob der Benutzer an Einträge erinnert werden möchte oder nicht (Notifikationen).

2.5 Komponenten der Anwendung

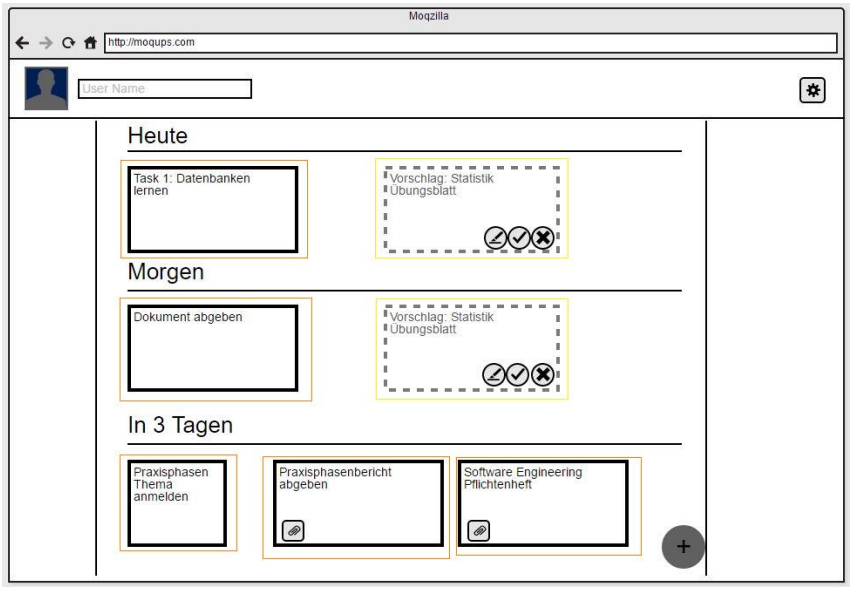

Abbildung 2.2: Komponenten des Newsfeeds

Wie in Abbildung 2.5 zu sehen ist besteht die Anwendung aus mehreren Einträgen und Vorschlägen. Diese UI-Elemente können als Komponenten angesehen werden. Deshalb soll das UI auf React.js basieren, da dort jedes UI-Element eine Komponente ist, die einen Zustand hat und mehrere Aktionen ausführen kann. Der zustand beinhaltet immer die anzuzeigenden Daten. Wird eine Aktion wie z.B. ein Klick-Event ausgelöst ändert sich der Zustand und die Komponente wird neu gerendert.[1] Dies erleichter die Entwicklung der Anwendung im Team und erhöht die Wiederverwendung von UI-Komponenten.

2.6 Trennung von Daten und Anzeigelogik

Eines der Ziele ist es Daten und Anzeigelogik getrennt von einander zu entwickeln. Deshalb können nicht alle Daten in den UI-Komponenten gespeichert werden. Möchten z.B. mehrere Komponenten auf die selben Daten zugreifen wird ein extra Lager (Store) für diese Daten benötigt.

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.

[1] Siehe https://facebook.github.io/react/

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. [2]

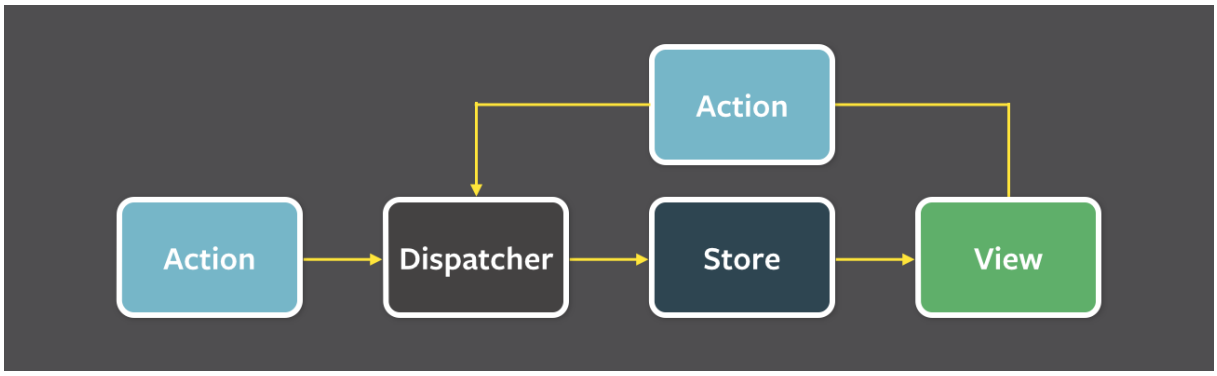

Abbildung 2.3: Flux
https://facebook.github.io/flux/docs/actions-and-the-dispatcher.html#content

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.

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

3 RESTful API

Die RESTful API basiert auf JSON und ist die zentrale Schnittstelle zwischen den Client Anwendungen (frontend) und dem Backend. Sie basiert auf http, welshalb alle Anfragen vom Client gesteuert werden.

3.1 Übersicht der Funktionen

Im folgenden werden alle Funktionen der RESTful Schnittstelle aufgelisted.

ID	Methode	Pfad	Beschreibung
R-10 Userverwaltung			
R-10.1	POST	/users/	Erstelle einen Benutzer. Es müssen ein „username“ und ein „password“ übergeben werden.
R-10.2	PUT	/users/:userId	Bearbeiten eines users.
R-10.3	DELETE	/users/:userId	Lösche einen User.
R-10.4	GET	/users/login	Authentifizierung durch session. „username“ und „password“ müssen übergeben werden.
R-10.5	GET	/users/logout	Beendigung der Session.
R-20 Newsfeed			
R-20.1	GET	/newsfeed/:userid	Hole alle Einträge für den Nutzer basierten Newsfeed. „top“ definiert den Anfang der Einträge relativ zum aktuellen Zeitpunkt. „skip“ definiert wie viele Einträge zurückgegeben werden. Zurückgegeben wird ein Array aus Einträgen.

Tabelle 3.1: RESTful Pfade

ID	Methode	Pfad	Beschreibung
R-20.2	POST	/entry/	Füge einen Eintrag zum Newsfeed hinzu. Der Parameter heißt „entry“.
R-20.3	GET	/entry/:id	Gibt alle Daten zu dem Eintrag mit der id wieder.
R-20.4	DELETE	/entry/:id	Löschen eines Eintrags.
R-20.5	PUT	/entry/:id	Updaten eines Eintrags.
R-30 Notifikationen			
R-30.1	GET	/notifications/:userid/	Holen aller Notifikationen.

Tabelle 3.1: RESTful Pfade

3.2 Authentifizierung

Ein Benutzer darf lediglich die zu ihm gehörenden Einträge bearbeiten. Bei allen Funktionen außer dem Login und der Registrierung muss der Benutzer authentifiziert sein.

3.3 Einträge

Einträge werden durch die RESTful API als JSON-Objekt zurückgegeben.

Sie können folgende Struktur haben:

```
1 {
2     "title": "Eintrag",
3     "tags": ["tag1", "tag2"],
4     "components": [...(Objects)]
5 }
```

3.4 Notifikationen

A notification is a JSON object with the following structure:

```
{
    "date": int (UNIX timestamp),
    "entryId": int
}
```

Mit dieser information kann das Frontend die Notification zu einem bestimmten Zeitpunkt aufrufen. Über die „entryId“ kann der passende Eintrag angefordert werden.

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. **Recommendations**: Speichert die generierten Vorschläge.

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.

  "dirty": false,            # Das dirty-Flag gibt an, ob sich
                             # das Entry geändert hat.
                             # In diesem Fall muss es beim nächsten
                             # Durchlauf des Schedulers neu
                             # verarbeitet werden (Suchindex, Vorschläge!).

  "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.

  "hidden_for": [42]         # Eine Teilmenge des "owned-by" Feldes,
                             # die angibt, welche Benutzer den Entry
                             # in ihrem Newsfeed ausgeblendet haben.
                             # Über die Suche ist er dann immernoch
                             # auffindbar.

  "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": ["C", "Übung"],   # Ein Array mit den Tags, die der
                            # Benutzer für den Entry vergeben hat.

  "components": [           # Ein Array mit den Komponenten des Entry
    { /* Component 1 */ },
    { /* Component 2 */ },
    { /* Component 3 */ }
  ]
}
```

Jedes Objekt im components-Array ist dabei nach dem folgenden Schema aufgebaut:

```
{
  "type": "text",           # Der Typ der Komponente

  "generated": false        # Gibt an, ob die Komponente von Norbert
                            # aus dem Inhalt extrahiert wurde.

  "data": {                 # Der eigentliche Inhalt der Komponente.
    /* ... */               # der Aufbau des data-Objekts ist abhängig
  }                         # von dem verwendeten Komponenten-Typ.
}
```

Informations Jede Information wird als ein Objekt in dieser Collection gespeichert. Informations sind grundsätzlich sehr ähnlich zu Entries aufgebaut. Ein Information-Objekt ist nach dem folgenden Schema aufgebaut:

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

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

  "dirty": false,           # Das dirty-flag gibt an, ob sich
                            # die Information vom externen Service
                            # geändert hat.
                            # In diesem Fall muss sie beim nächsten
                            # Durchlauf des Schedulers neu
```

```
                            # verarbeitet werden (Suchindex!).

  "hidden_for": [42, 12]    # Ein array mit den IDs der Benutzer, die
                            # die Information aus ihrem Newsfeed
                            # "gelöscht" haben. Bei diesen wird
                            # die Information nicht mehr angezeigt.
                            # Sie kann jedoch trotzdem noch über die
                            # Suche gefunden werden.

  "title": "Überschrift",   # Der Titel der Information

  "components": [           # Ein Array mit den Komponenten der Information
    { /* Component 1 */ },
    { /* Component 2 */ },
    { /* Component 3 */ }
  ]
}
```

Die Komponen-Liste der Informations hat das selbe Format, wie die Komponenten-Liste eines Entries.

Recommendations Für jede erzeugte Recommendation wird ein Objekt in der Recommendation-Collection angelegt, das dem folgenden Schema entspricht:

```
{
  "user_id": 42,            # Id des Benutzers, dem der Vorschlag
                            # angezeigt werden soll.

  "entry_id": 512           # Id des Eintrags, auf den sich der
                            # Vorschlag bezieht.

  "hidden": false           # Gibt an, ob der Vorschlag gelöscht wurde.
                            # Dann wird er nicht mehr angezeigt. Es ist
                            # nicht möglich, beim Löschen einfach das
                            # Objekt aus der Collection zu löschen, da
                            # der Vorschlag sonst neu
                            # generiert werden würde.
}
```

5 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.

5.1 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 5.1) 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.

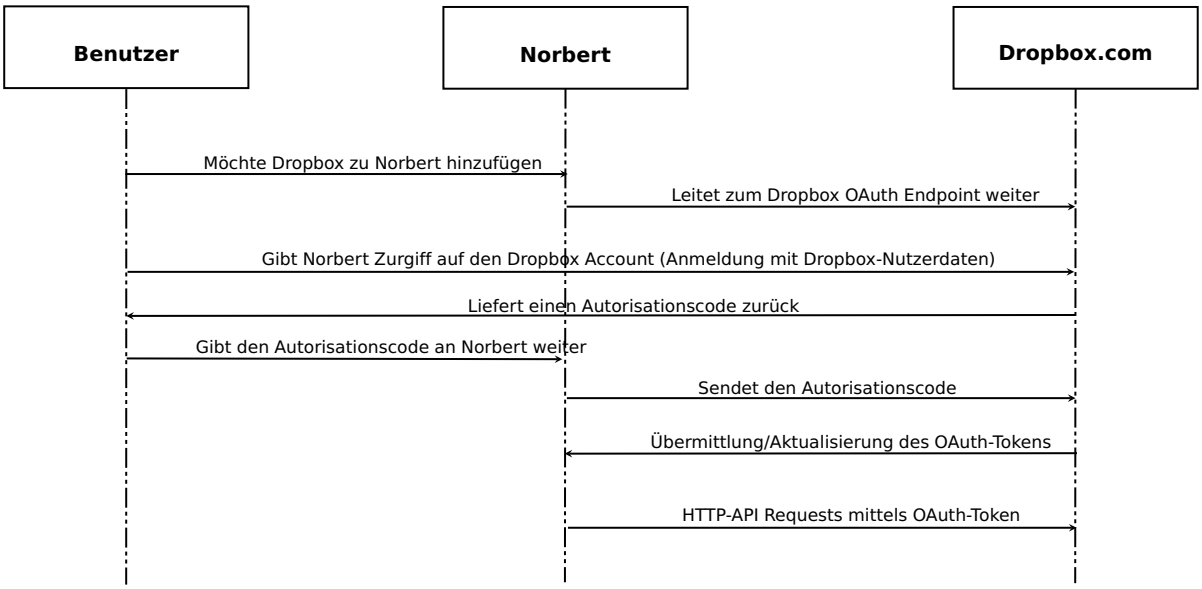

Abbildung 5.1: OAuth-Autorisierungsprozess zwischen Norbert und Dropbox.com

5.2 Dropbox Endpoints

Die Dropbox HTTP-API besteht aus mehreren Endpoints (Abbildung 5.2), 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.

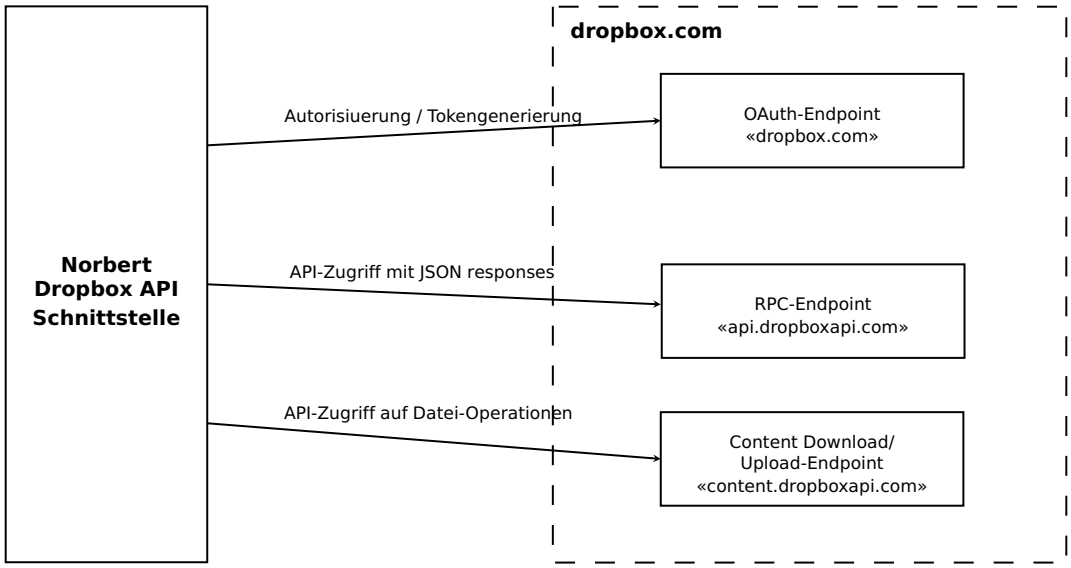

Abbildung 5.2: API-Zugriffe über die Dropbox-Endpoints