ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

Abschlussbericht

für die Projektarbeit im Modul PIB-PA SS 16
an der Hochschule für Technik und Wirtschaft des Saarlandes
im Studiengang Praktische Informatik
der Fakultät für Ingenieurwissenschaften

Intelligente Benachrichtigungen für Twitter

vorgelegt von

Marius Backes
Martin Feick
Niko Kleer
Marek Kohn
Stefan Schlösser
Philipp Schäfer
Oliver Seibert

betreut und begutachtet von
Prof. Dr.-Ing. Klaus Berberich

Saarbrücken, 30.09.2016

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Gemäß dem Metcalfe'schen Gesetz[1] haben sich *Social Media* zu einer der wichtigsten Informationsquellen unserer Zeit etabliert. Nutzer agieren nicht nur als Konsumenten, sondern immer mehr als Produzenten von Information. Unterstützt durch die Allgegenwärtigkeit des Internets, gelangen auf diese Weise exorbitante Mengen unstrukturierter Information in Echtzeit an die Teilnehmer des Netzwerks. Eine Selektion dieser Daten, etwa nach persönlichen Interessen, bleibt dem Nutzer meist selbst überlassen. Als Inbegriff dieses Prinzips gilt der Mikroblogging-Dienst *Twitter*. Nutzer des Dienstes können auf 140 Zeichen beschränkte Nachrichten - sogenannte *Tweets* - verfassen und diese dann ungefiltert und ungeprüft veröffentlichen.

Ziel dieses Projektes ist es nun eine Software zu entwickeln, mit der die Selektion von Tweets automatisiert erfolgen kann. Grundlage dafür bietet die von Twitter bereitgestellte Streaming API, welche eine permanente Versorgung mit neuen Tweets sicherstellt. Diese sollen im weiteren Verlauf so strukturiert werden, dass eine individuelle Aufbereitung der Tweets ermöglicht wird. Im Vordergrund steht hierbei eine objektive Einschätzung der Tweets anhand subjektiv festgelegter Merkmale.

Die systematische Verarbeitung großer Datenmengen, im Allgemeinen unter dem Begriff *Data-Mining* bekannt, stellt dabei die zentrale Herausforderung dar. Aus dem vorhandenen Datenpool soll jene, für den Nutzer relevante Information extrahiert werden, sodass individuelles Wissen generiert werden kann. Da ständig neue Daten ankommen, muss dieser Prozess ereignisgesteuert erfolgen. Um den Datenüberschuss möglichst gering zu halten, sind schnelle Entscheidungen essentiell.

In den folgenden Abschnitten werden zunächst die technischen Rahmenbedingungen im Hinblick auf die Funktionsweise der Twitter API erläutert. Es folgt eine Konkretisierung der Anforderungen in Form einer Produktvision. Anschließend wird das Projekt erst aus organisatorischer, dann aus technischer Sicht beschrieben bevor die Implementierung der einzelnen Teilbereiche detailliert betrachtet wird. Im Rahmen einer Evaluation sollen abschließend einige Qualitätsmerkmale bezüglich Organisation und Umsetzung des Projektes untersucht werden.

[1] Das *Metcalfe'sche Gesetz* (Robert Metcalfe, 1980) besagt, dass der Nutzen eines Netzwerks proportional zu der Anzahl der Verbindungen innerhalb des Netzwerkes wächst, während die Kosten proportional zu der Anzahl der Teilnehmer wachsen.

1.2 TwitterStream API

Die *Twitter Streaming APIs* bieten eine öffentlich zugängliche Programmierschnittstelle für einen Echtzeit-Zugriff auf Daten der Social Mediaplattform *Twitter* auf der Grundlage von HTTP. Eine Webanwendung, die ihrem Endnutzer Daten eines Dritten zur Verfügung stellt, arbeitet stets einerseits als Server für die Clientanwendung des Nutzers und ist andererseits selbst Client eines Datenservers, in diesem Fall Twitter. Der hergebrachte Weg zu online verfügbaren Daten, sowohl für Twitter als auch bei anderen Diensten wie *Facebook* oder *GoogleEarth*, erfolgt über eine REST API. Dabei stellt eine auf REST basierende Webanwendung, meist auf Anfrage eines Benutzers mittels eines HTTP-Requests, mit dem bestimmte Suchparameter übermittelt werden, eine entsprechende Anfrage beim Server des Anbieters der Daten. Er erhält die gewünschten Daten in der Response und gibt sie an den Benutzer weiter. Die Anfrage vollzieht sich dabei zustandslos. Für jede Anfrage wird eine neue TCP-Verbindung zum Anbieter aufgebaut, die wie jede HTTP-Verbindung nach der Übermittlung der Daten sofort wieder durch den Anbieter-Server beendet wird. Das Prinzip einer Streaming API beruht dagegen gerade darauf, die Verbindung zum Da-

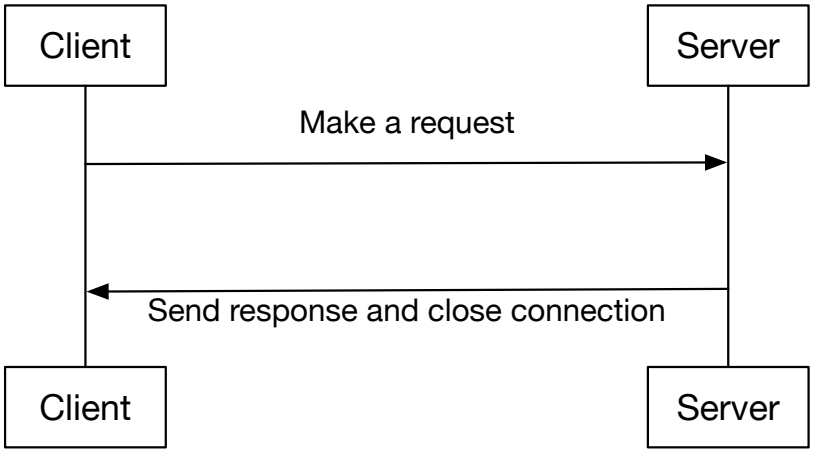

Abbildung 1.1: Grundprinzip einer REST-API [2]

tenserver über einen längeren Zeitraum aufrecht zu erhalten. Die Verbindung wird auch hier durch einen Request eingeleitet, in dem gewisse Such- oder vielmehr Filterparameter enthalten sind. Doch der Datenserver antwortet nicht unverzüglich mit bestimmten vorliegenden Daten und beendet die Verbindung, sondern hält die Verbindung aufrecht und sendet Informationen über zu den Suchparametern passende Ereignisse in Echtzeit. Auf der Client-Seite, also in der Webanwendung, können diese Informationen dann aus dem Socket der Verbindung gelesen werden. Dieser Unterschied führt auch zu einer anderen allgemeinen Architektur eines Anwendungsservers, der eine Streaming API verwendet. Bei Verwendung einer REST API wird eine Anfrage meist von einem Benutzer eingeleitet und vom Anwendungsserver an die Schnittstelle weitergereicht. Die erhaltenen Daten werden nicht langfristig gespeichert, sondern nach eventueller Bearbeitung direkt an den Benutzer geliefert. Eine permanente Verbindung kann hingegen nicht ständig durch den Benutzer überwacht werden und muss deshalb von der Serversoftware verwaltet werden. Die empfangenen Daten werden für den künftigen Zugriff gespeichert. Während die

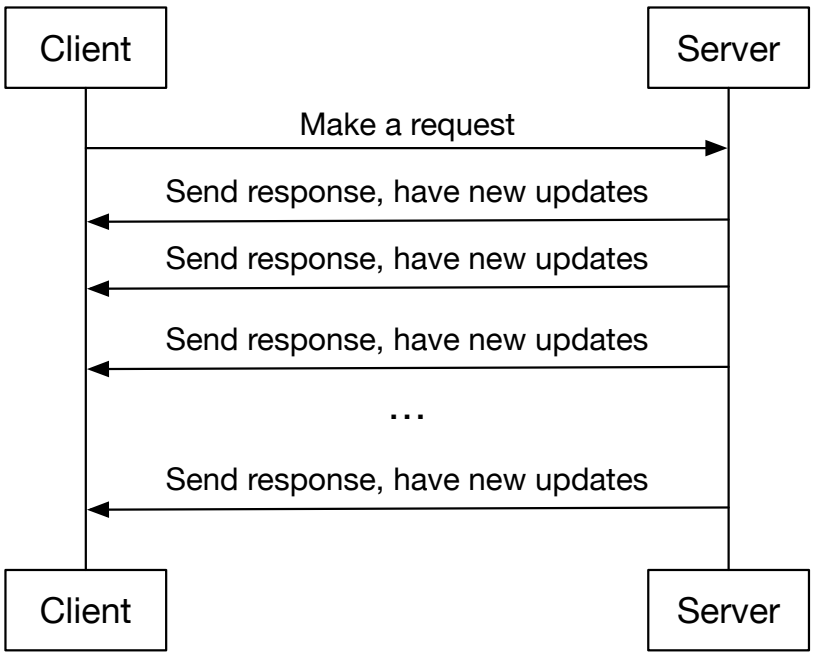

Abbildung 1.2: Grundprinzip einer Streaming-API [2]

Serversoftware im Falle einer auf REST basierenden Anwendung also aus einer Einheit besteht, die sowohl für die Interaktion mit dem Nutzer als auch mit dem Datenserver verantwortlich ist, lassen sich bei einer typischen Streaming API zwei Teile unterscheiden. Der eigentliche Stream wird von einem abgeschlossenen Modul unabhängig vom Nutzer aufgebaut und verwaltet. Dabei handelt es sich im Grunde genommen um eine Clientanwendung zum Twitter-Datenserver. Er erhält seine Suchparameter aus einer Datenbank, in der auch die empfangenen Daten gespeichert werden. Der Zugang zu diesen Daten durch den Nutzer wird etwa durch ein Webinterface hergestellt, für das der zweite Teil der Anwendung als Server zuständig ist.

Twitter bietet insgesamt vier Streaming-Schnittstellen an. Im Zentrum des Projekts steht der *Public Stream*.[2] Dieser wird neben anderen Suchparametern, wie Standort oder Sprache, vor allem mit einer Liste von bis zu 400 Schlüsselwörtern initialisiert. Jeder neue Tweet, der einem oder mehreren Keywords entspricht, wird durch den Stream empfangen. Das mögliche Datenvolumen ist also nur durch die Anzahl an Parametern begrenzt, während alle Tweets, die den Parametern entsprechen, erfasst werden. Allerdings ist es seitens Twitter vorgesehen, dass nur ein Public Stream pro Anwendung existiert. Daneben werden außerdem noch *User Streams* und *Site Streams* angeboten, die alle Tweets liefern, welche mit einem bestimmten Twitternutzer in Verbindung stehen.[3] Der *Firehose Stream* liefert schließlich bedingungs- und ausnahmslos alle öffentlich zugänglichen Tweets. Er ist jedoch nicht allgemein zugänglich, sondern bedarf einer Freischaltung durch Twitter.

[2] Für eine nähere Beschreibung der möglichen Parameter und ihrer Beschränkungen siehe https://dev.twitter.com/streaming/reference/post/statuses/filter

[3] *Site Streams* sind die Multi-User-Version der *User Streams*.

1.3 Produktvision

Soziale Medien haben ein gespaltenes Image. Einerseits gehören sie unumstritten zum Leben im 21. Jahrhundert. Kaum ein Ereignis wird nicht durch Twitter, Facebook und andere dokumentiert und kommentiert, wenn diese Plattformen nicht sogar als Katalysator wirken, durch den ein Ereignis erst weltweit bekannt wird. Andererseits besteht bei der Teilnahme an dieser neuen Form der Kommunikation auch immer die Gefahr von ihr vereinnahmt zu werden. Jede neue Form von sozialem Medium bildet einen eigenen Mikrokosmos, dessen Regeln befolgt werden müssen, um die angebotenen Informationen nutzen zu dürfen oder zu können.[4] Nicht zuletzt setzt man sich durch die Mitgliedschaft in einem sozialen Netzwerk immer auch zugleich einer höheren Wahrscheinlichkeit aus selbst zum Diskussionsstoff zu werden. Die Alternative zur aktiven Teilnahme an sozialen Medien, das Rezipieren einer Auswahl von Beiträgen zu einem Thema durch ein traditionelles Medium, etwa die Tagesschau, macht wiederum gerade den Vorteil einer ungefilterten Meinung im Internet zunichte. Es bedarf im Internet einer Möglichkeit für Außenstehende die Kommunikation in sozialen Netzwerken nach eigenem Ermessen zu beobachten. Durch Anwendungen nach dem REST-Prinzip sind nur gezielte Zugriffe auf die Vergangenheit möglich. Es fehlt jedoch eine Möglichkeit, um auf die aktuelle Kommunikation zugreifen zu können.
Das Ziel dieses Projekts ist es diese Lücke zu schließen. Es wird eine Webanwendung erstellt, in der sich Benutzer mit einem möglichst geringen Verwaltungsüberbau anmelden und beliebige Schlüsselwörter anlegen können, die ihren Interessen entsprechen. Diese werden zur Schlüsselwortliste eines Public Streams hinzugefügt, so dass sämtliche Tweets, die sie enthalten, während der Abwesenheit des Nutzers erfasst und in einer Datenbank gespeichert werden. Dem Nutzer stehen die seinen Schlüsselwörtern entsprechenden Tweets in einer Übersicht zur Verfügung, die er selbst noch einmal nach beliebigen Schlüsselwörtern und anderen Parametern filtern kann. Um den Vorteil des Echtzeitzugriffs auf aktuelle Ereignisse durch einen Stream voll auszunutzen, soll dem Benutzer außerdem die Möglichkeit gegeben werden, seine E-Mail-Adresse zu hinterlassen, so dass er automatisch über neu eingetroffene Tweets unterrichtet wird.

[4] In Hinsicht auf die Einhaltung von Verhaltensregeln bzw. dem Verständnis des Jargons

2 Projektmanagement

Als Vorgehensmodell für das Projekt wurde ein agiles Vorgehen nach dem Vorbild von Scrum und Extreme Programming gewählt. Dies ist hauptsächlich der Tatsache geschuldet, dass die Verwirklichung unseres Vorhabens die Einarbeitung in zahlreiche Einzelaspekte erforderte, die dem gesamten Team neu waren. Es war nicht abzusehen, welche Teile überhaupt umsetzbar sind und wie erfolgreich die Implementierung verlaufen wird. Deshalb sollte das Projekt zunächst nur den eigentlichen Stream als Kern verwirklichen und dann in kleinen Iterationen um immer mehr Funktionalität, vor allem einer Website als User Interface, erweitert werden.

2.1 Teamorganisation

Es wurde kein Projektleiter festgelegt. Entscheidungen wurden stets im Plenum getroffen. Die Gruppe repräsentierte sich nach außen stets gemeinsam. Die sonstige interne Organisation des Projektteams wurde ebenfalls durch die Unwägbarkeiten der Durchführung geprägt. Da zu Beginn noch nicht klar war, welche konkreten Aufgaben es geben wird und wie groß ihr Umfang sein wird, konnten keine klaren Zuständigkeiten verteilt werden. Die achtköpfige Projektgruppe teilte sich jedoch schnell in mehrere kleine Teams aus zwei oder drei Mitgliedern, die die einzelnen Aufgaben, die sich stellten, übernahmen. So stellte sich bald eine gewisse Spezialisierung der Teams ein. Ein „Datenbankteam“ kümmerte sich um die Erstellung und Pflege der Datenbank. Diese Gruppe war auch für die Installation und Wartung der neuesten Testversion auf dem Testserver im Softwarelabor zuständig. Ein „Backendteam“ sorgte für den Stream und ein Interface für Nutzerabfragen aus der Datenbank. Und ein „Frontendteam“ war für die Erstellung der Website verantwortlich. Übergreifende Aufgaben wurden von einem vierten Team übernommen. Dazu gehörten sowohl projektorganisatorische Erledigungen (Zeiterfassung) als auch übergreifende Aspekte der Implementierung wie etwa Sicherheitsfragen.

2.2 Tools

In dem nun folgenden Abschnitt wird auf die organisatorischen Aspekte des Projekts eingegangen. Insbesondere welche Kommunikationstools und unterstützende Systeme zur erfolgreichen Durchführung des Projekts benutzt wurden.

Als Versionskontrollsystem wurde vom Projektteam GitHub gewählt, da einige Projektteilnehmer bereits Erfahrung mit diesem System gesammelt hatten. Außerdem erfreut es sich großer Beliebtheit im Informatik spezifischen Umfeld. Das Projekt(Code), die Datenbank und die Erstellung des Berichts wurden unter Zuhilfenahme von GitHub realisiert. Alle zusätzlichen Dokumente wie zum Beispiel Klassendiagramme, ER-Diagramme usw. wurden in einer eigens dafür angelegten Dropbox gehalten.

Als Kommunikationsplattform wurde Slack festgelegt. Dieses bietet deutliche Vorteile gegenüber Whatsapp und andern Messenger. Slack besitzt ein umfangreiches Internetforum mit unterschiedlichen Channels. Es ist sowohl möglich einzelnen Personen, sowie ganzen Gruppen oder auch allen Projektteilnehmer zu schreiben, sodass die Informationen jeden Projektteilnehmer direkt und ohne Umwege über Dritte erreichen. Slack bietet

eine Integration von GitHub, Skype und Dropbox. Dadurch wird man stetig über Änderung(Commits bei GitHub) benachrichtigt. Die Benachrichtigungen sind direkt auf der Internetseite von Slack zu sehen, können aber auch über E-Mail oder die App Slack abgerufen werden. Dokumente kann man sofort via Drag and Drop versenden. Auch die Umfragefunktion wurde vom Projektteam zur Terminabstimmung häufig genutzt.

Um Aufgaben besser verwalten zu können wurde das Tool Trello verwendet. Bei Trello handelt es sich um ein Tool zur dynamischen Verwaltung und Planung von verschiedenen Aufgaben. Man hat die Möglichkeit diese Aufgaben den unterschiedlichen Mitgliedern zur Bearbeitung zuzuordnen. Des weiteren kann man die einzelnen Aufgabe mit weiteren Checklisten versehen, welche die einzelnen Schritte einer Aufgabe näher erläutern. Außerdem lassen sich die Aufgaben in sogenannte Labels einteilen. Solche Labels geben das Gebiet der Aufgabe an, wie z.B. Frontend oder Datenbank. Zuletzt werden die erstellten und personalisierten Aufgaben in einen Bereich eingeteilt, der angibt in welchem Stadium sich die Aufgabe befindet. Man unterscheidet hier zwischen den Bereichen „To Do“, „In Arbeit“ und „Fertig“. Ist eine Aufgabe in Ihrem Bereich fertig, so wird sie einfach in den weiterführenden Bereich verschoben.

Damit der Projektaufwand genau dokumentiert werden konnte wurde eine Excel-Vorlage erstellt, welche nach jeder Kalenderwoche ausgefüllt, an eine zuvor festgelegte Person geschickt wurde. Diese fertigte eine Wochenliste an und trug die Stunden in die Gesamtliste ein. Die Wochenliste sowie die Gesamtliste wurden, sobald alle Stundenzettel eingegangen waren, erstellt und via E-Mail an den betreuenden Professor geschickt. Dadurch war eine Lückenlose Dokumentation der Stunden jedes Projektteilnehmers gewährleistet.

Name	Datum	Von	Bis	Stunden	Thema
Dummy	30.09.2016	8:00	10:00	2:00	Projekt wurde erfolgreich beendet
	01.10.2016	12:00	13:00	1:00	Wintersemster 2016/2017
Gesamtstunden der KW:				**3:00**	

Abbildung 2.1: Beispieldatensatz Stundenzettel

Fazit GitHub bereitete dem Projektteam einige Probleme, da unvorhersehbare Konflikte beim Pullen, Pushen und Mergen des Projekts auftraten. Jedoch bekamen die Projektteilnehmer diese Probleme weitestgehend durch mündliche und schriftliche Abstimmung untereinander in der Griff.

Slack wurde zur gesamten Kommunikation innerhalb des Teams benutzt. Besonders aber die Umfragefunktion, welche zur Abstimmung von Terminen sehr nützlich war. Auch um über den Fortschritt des Projekts (Commits bei GitHub) ständig informiert zu sein, war Slack bzw. die Integration von GitHub in Slack eine gute Möglichkeit.

Trello wurde vorallem in der ersten Hälfte des Projekts benutzt. Da sich das Projektteam aber in Gruppen für die verschiedenen Anforderungsbereiche einteilte(Frontend, Architektur und Backend), die weitestgehend unabhängig voneinander sind, gab es wenig Konflikt beim Suchen, finden und bearbeiten von Tasks. Innerhalb der Gruppen fand der Austausch vor allem über Skype statt. Dadurch gab es keine Probleme mit beispielsweise „Doppelbearbeitung“ von Aufgaben. Somit war ein ergebnisorientiertes und effektives Arbeiten möglich.

3 Projektbeschreibung

Der Benutzer interagiert mit der Software über eine Webanwendung, die er in einem Browser startet. Im Zentrum arbeitet eine Springanwendung auf einem Tomcat-Server, die die Aufgaben übernimmt die Verbindung mit Twitter herzustellen und die empfangenen Daten dem Frontend zur Verfügung zu stellen. Die Funktionsweise der Anwendung orientiert sich am bereits beschriebenen allgemeinen Aufbau einer Streaming-Anwendung aus zwei Teilen, die unabhängig voneinander arbeiten und nur über eine gemeinsame Datenbank miteinander in Verbindung stehen. Das Stream-Modul der Anwendung speichert die Tweets in der Datenbank und ein Ausgabe-Modul liefert die nötigen Daten zur Erstellung einer Webpage.

Der Stream liefert sogenannte Statusmeldungen. Dabei handelt es sich im Grunde genommen um eine Zusammenstellung aller Informationen zu einem bestimmten Tweet. Dazu gehören unter anderem auch stets alle Informationen zum Autor des Tweets. Somit sind die zentralen Daten, die dem Stream entnommen und in der Datenbank gespeichert werden, Informationen zu einem Tweet und seinem Autor. Sowohl Tweets als auch Autoren werden von Twitter durch eindeutige IDs markiert. Diese werden auch in der Anwendung zur Identifikation verwendet. Von den sonstigen zahlreichen Daten werden für Tweets der Text, die Sprache, ein Ort, der Entstehungszeitpunkt, ein eventuell vorhandenes, angehängtes Bild[1], die Anzahl positiver Bewertungen[2] und die Häufigkeit der Verbreitung durch einen *Retweet* aufgenommen. Zu Autoren werden Name, ein Profilbild und die Anzahl der *Follower* gespeichert. Die meisten dieser Daten sind für die Ausgabe im Frontend bestimmt. Die *Favorite-*, *Retweet-* und *Follower*-Statistiken werden zur Bewertung von Tweets in Form einer Priorisierung verwendet.
Für die Ausgabe von einzelnen Tweets werden alle dafür erforderlichen Informationen in einem *OutputTweet* gesammelt. Dazu gehören neben den relevanten Daten zum Tweet und seinem Autor auch die aktuelle Priorität. Die Priorität eines Tweets ergibt sich sowohl aus den allgemeinen Statistiken, die sich laufend ändern, als auch aus den persönlichen Einstellungen des Benutzers, die ebenfalls ständigen Änderungen unterliegen. OutputTweets sind dadurch also immer auf bestimmte Benutzer und den aktuellen Zeitpunkt zugeschnitten.

Auch wenn es das Ziel des Projekts ist einen möglichst unbeteiligten Einblick in die Twitter-Kommunikation zu bieten, besteht auch eine gewisse Notwendigkeit, die Nutzer längerfristig zu erfassen, so dass sie Schlüsselwörter hinterlegen können. Das heißt, dass Benutzer sich mit einem Benutzernamen und einem Passwort registrieren und anmelden müssen. Für die automatische Benachrichtigung muss außerdem eine E-Mail-Adresse angegeben werden. Auch die Benutzer werden in der Datenbank gespeichert. Die Ein- und Ausgabe von Benutzerdaten und Schlüsselwörtern erfolgt über dasselbe Modul der Spring-Anwendung, das auch für die Ausgabe der Tweets zuständig ist.

Die zentralen Parameter, nach denen die Tweets durch den Stream erfasst und später

[1] Mehrere Bilder sind zwar mögich aber unwahrscheinlich
[2] Diese wird bei Twitter als *favoriteCount* bezeichnet

den Benutzern präsentiert werden, sind deren Schlüsselwörter. Für jeden Nutzer werden diese Schlüsselwörter dabei mit einer Priorität von eins bis fünf versehen, die angibt, wie wichtig dem Nutzer dieses Schlüsselwort ist. Die jeweils aktuelle effektive Priorität eines Tweets ergibt sich aus allgemeinen Faktoren und der Priorisierung durch einen Benutzer. So lassen sich die verfügbaren Tweets für einen bestimmten Benutzer immer in eindeutiger Reihenfolge anzeigen. Die sehr weite Verwendung der Schlüsselwörter lässt sich am ehesten mit der Suche in Suchmaschinen und anderen sozialen Medien vergleichen, so dass viele Benutzer damit vertraut sind. Groß- und Kleinschreibung werden sowohl am Anfang als auch mitten im Wort nicht beachtet. Ebenso werden die für Twitter typischen Präfixe „#“ und „@“ miteinbezogen. Sollen zwei oder mehr Suchbegriffe in einem Tweet enthalten sein, so können diese durch Leerzeichen getrennt in einem Keyword gesucht werden.[3]
Neben diesen positiven Keywords soll auch die Möglichkeit bestehen ein Schlüsselwort auf eine Blacklist zu setzen. Tweets, die diese Schlüsselwörter enthalten, werden dem Benutzer nicht angezeigt. Es wird hier die genaue Schreibweise beachtet. Beide Arten von Keywords können auch pausiert werden, so dass die entsprechenden Tweets nicht mehr angezeigt werden. So wird eine Sichtung der Tweets durch den Benutzer weiterhin erleichtert, da er seine Suche zeitweilig global auf bestimmte Keywords einschränken kann oder bestimmte gesperrte Schlüsselwörter zulassen.
Die Sammlung von Tweets aus dem Stream wird dabei weder durch die Blacklist noch die Pausierung von Keywords beeinflusst. Beide Faktoren bestimmen lediglich, welche Tweets dem Anwender potentiell angezeigt werden. Dabei wird ihm zunächst eine übersichtliche, zahlenmäßig begrenzte Auswahl präsentiert, die sich aus den interessantesten Tweets zusammensetzt. Die Liste kann nach der Priorität der Tweets oder ihres Entstehungszeitpunkt sortiert werden. Ein weiteres mögliches Anzeigekriterium ist die Sprache, in der die Tweets verfasst werden. So kann die Darstellung etwa nur auf deutschsprachige Tweets beschränkt werden. Schließlich ist eine weitere Filterung anhand eines beliebigen Schlüsselworts möglich. Die bloße Eingabe dieses zusätzlichen Schlüsselworts führt zunächst zu einer Filterung der angezeigten Tweets, so dass nur noch diejenigen davon angezeigt werden, die auch dieses Schlüsselwort enthalten. Der Suchbegriff lässt sich jedoch auch in einer Anfrage als weiteres Schlüsselwort bei der Filterung der gesammelten Tweets in der Datenbank verwenden. Eine solche Suche zieht alle für den Nutzer gesammelten Tweets in der aktuell eingestellten Sprache in Betracht. Ein Sonderfall ist die Suche nach einem leeren Suchbegriff, die alle Tweets des Nutzer zurückgibt. Das Ergebnis lässt sich stets wie die ursprüngliche Liste durch Sortierung und Filterung weiterverarbeiten.

Zu den größten Herausforderungen des Projekts gehört die Skalierung des Ausmaßes der Sammlung von Daten. Konkret geht es vor allem um die Frage, welche Daten wie lange gespeichert werden. Es muss ein Kompromiss erzielt werden, der sowohl die praktische Durchführbarkeit als auch die Relevanz und Verfügbarkeit der Daten für den Endnutzer gewährleistet. Das größte Problem, das bei einem unskalierten Umgang mit dem Twitterstream auftreten kann, ist eine zu große Menge an Daten in der Datenbank, so dass Abfragen sehr lange dauern. Eine weitere Schwierigkeit dabei ist die Möglichkeit, dass durch zu viele Keywords und ein hohes Tweet-Aufkommen die Abarbeitung der Tweets durch das Stream-Modul nicht schnell genug verläuft.
Die Anzahl der zu einem bestimmten Zeitpunkt gespeicherten Tweets und somit der Daten in der Datenbank lässt sich durch das regelmäßige Löschen von nicht mehr aktuellen Tweets begrenzen. Außerdem werden Tweets, die ein bestimmtes Alter überschritten haben, gar nicht erst gesammelt und aufbewahrt. Das Limit wurde auf 48 Stunden festgelegt.

[3] vgl. https://dev.twitter.com/streaming/overview/request-parameters#track

Es sollten sich also keine älteren Tweets im System befinden. Autoren müssen nur so lange gespeichert werden wie es auch Tweets von ihnen in der Datenbank gibt.
Da der Schwerpunkt des Interesses auf aktuellen Ereignissen liegt, genügt eine vollständige Aufnahme der Tweets der letzten zwei Tage. Eine Alternative bzw. Erweiterung zu dieser zentralen Einschränkung ist die unvollständige Sammlung der Tweets, über die etwa anhand der allgemeinen Priorität entschieden werden kann. Weniger wichtige Tweets werden außen vor gelassen. Andererseits können allgemein besonders interessante Tweets beim Löschen übergangen werden. Solche Verfeinerungen verkomplizieren die Anwendung jedoch unnötig und verfehlen außerdem den Zweck des Projekts. Einerseits ist eine vollständige Sammlung aller Tweets zu einem Thema erstrebenswert, um eine möglichst ungefilterte Meinung präsentieren zu können. Die zur allgemeinen Bewertung eines Tweets herangezogenen Faktoren geben nur bedingt Auskunft darüber, ob ein Tweet von Interesse ist. Andere Faktoren spielen ebenfalls eine Rolle, sind aber zu individuell um in eine allgemeine Bewertung einzufließen. Andererseits sollte die Menge an Tweets, die dem Nutzer zur Verfügung steht, immer möglichst übersichtlich und aktuell sein.
Neben dem Alter eines Tweets kann auch die Sprache, in der er verfasst ist, als Ausschlusskriterium dienen. Da zunächst an eine deutschsprachige Zielgruppe gedacht wird, liegt bei deutschen Tweets auch das Hauptinteresse. Außerdem werden auch Tweets in der *lingua franca* des Internets Englisch gesammelt. Eine Erweiterung wäre hier leicht möglich und für Sprecher weiterer Sprachen sicherlich wünschenswert, würde jedoch in der gegenwärtigen Anwendung zu einer zu hohen Auslastung des Streams und der Datenbank führen.
Eine weitere Möglichkeit die Anzahl der gesammelten Tweets zu verringern ist der Ausschluss von Retweets. Wenn es sich bei einem Tweet um einen Retweet handelt, enthält das *Status*-Objekt auch einen *Status* des ursprünglichen Tweets. Da Retweets meist nur der Verbreitung eines Tweets dienen, werden sie in der Anwendung ignoriert und lediglich zur Aktualisierung der Statistiken des ursprünglichen Tweets verwendet.
Eine weitere notwendige Maßnahme ist die Beschränkung der Anzahl von Schlüsselwörtern pro Benutzer auf maximal zehn Stück. Dadurch wird einerseits sichergestellt, dass auch bei größeren Nutzerzahlen die obere Grenze von 400 Keywords eines *public streams* nicht überschritten wird.[4] Andererseits begrenzt auch dies die Anzahl an vorhandenen Daten in der Datenbank und der Stream wird weniger belastet, was wiederum eine Performanzsteigerung zugunsten der Nutzererfahrung impliziert. Die Länge der Blacklist muss nicht beschränkt werden, da sie keinen Einfluss auf die Filterung des Streams und die Speicherung in der Datenbank hat, sondern nur die Ausgabe einschränkt.
Schließlich muss auch die Anzahl der im Frontend angezeigten Tweets kontrolliert werden. Eine zu große Anzahl an Tweets ließe sich durch den Benutzer dabei sowieso nicht überblicken. Andererseits kann die Datenbank-Anfrage und die Übertragung eines entsprechenden JSON-Dokuments bei genügend vorhandenen Tweets andernfalls mehrere Minuten dauern. In der Anwendung werden deshalb zunächst nur maximal 100 Tweets angezeigt, die nach ihrer Priorität ausgewählt werden. Sie bieten einen schnellen Überblick über die interessantesten Neuigkeiten. Diese Vorgehensweise verfälscht jedoch die Ergebnisse der Weiterverarbeitung der Tweets, also die Sortierung nach der Zeit und die Filterung nach einem Schlüsselwort, da nur noch die 100 angezeigten Tweets einbezogen werden. Deshalb besteht für den Benutzer auch immer die Möglichkeit alle für ihn gesammelten Tweets zu laden, anhand eines bestimmten Begriffs zu filtern und weiter zu bearbeiten. Dabei werden lange Ladezeiten und unübersichtlich viele Einträge in Kauf genommen, um dem Nutzer Zugriff auf alle seine Tweets bieten zu können.

[4] Damit ist eine maximal mögliche Nutzeranzahl von 40 Nutzern gegeben unter der Worst-Case-Bedingung, dass sich die Schlüsselwörter der Nutzer nicht überschneiden.

4 Implementierung

4.1 Das Spring-Framework

4.1.1 Warum Spring?

In dem nun folgenden Abschnitt wird die Frage erläutert warum das Spring Webframework verwendet wurde. Insbesondere welche Alternativen es gab und die jeweiligen Vor- und Nachteile sowie die Anforderungen, die an das Webframework gestellt wurden.
Die Projektteilnehmer legten zusammen folgende Anforderungen fest. Zum einen sollte es ein Webframework sein, welches auf Java basiert, da alle Teilnehmer mit dieser Programmiersprache vertraut sind. Desweiteren sollte ein standardisiertes Framework eingesetzt werden, da diese getestet sind, Code minimieren und somit weniger Fehler und potentielle Fehlerquellen für das Webprojekt bedeuten. Weitere Anforderungen waren die Popularität/Zukunftssicherheit, Toolunterstützung, Dokumentation und der Support.
Nach der Recherchezeit standen folgende Frameworks zur Auswahl:

- Spring (https://spring.io/)
- Vaadin (https://vaadin.com/home)
- Java Server Faces (https://javaserverfaces.java.net/)
- Spark (http://sparkjava.com/)

Im Folgenden werden zu jedem der obigen aufgelisteten Frameworks einige grundlegenden Informationen, sowie die Vor- und Nachteile im Bezug auf die oben genannten Anforderungen, für die Nutzung der Projektgruppe aufgelistet.

Spring ist ein auf J2EE basierendes Anwendungsframework und enthält ein eigenes MVC Framework. Es handelt sich dabei um ein Framework welches speziell für die Entwicklung mit Java und JavaEE gedacht ist. Die Vorteile sind, dass es ein quelloffen ist. Zudem ist es sehr verbreitet, was eine hohe Zukunftssicherheit mit sich bringt. Die Dokumentation und der Support sind durch die häufige Verwendung hervorragend. Bei Spring steht die Entkopplung der Applikationskomponenten im Vordergrund, was nach dem Model-View-Controller Prinzip realisiert wird. Ein Nachteil ist der hohe Einarbeitungsaufwand, da keiner der Projektteilnehmer Erfahrungen mit Webanwendungen hatte. Dieser Nachteil trifft jedoch auf alle Frameworks zu, insbesondere aber auf das Spring Framework, da es weitere Web Entwicklungstechnologien benutzt, wie Servlets, JavaServer Pages und Faces welche bis dahin auch weitestgehend unbekannt waren.

Vaadin ist eine neue Innovative Möglichkeit der Webentwicklung, welche ebenfalls speziell für Java gedacht ist. Vaadin bietet es eine serverseitige Architektur, was bedeutet, dass der Großteil der Programmlogik auf dem Server läuft. Dadurch hat das Framework eine andere Vorgehensweise als JavaScript Bibliotheken oder auf Browser Plugins basierenden Lösungen. Vaadin ist eine frei verfügbare Software. Der Einstieg in

die Webentwicklung fällt deutlich leichter als mit dem oben genannten Spring Framework, da einfache Anwendungen intuitiv entwickelt werden können. Es existiert ebenfalls eine umfangreiche Dokumentation und ein guter Support über das Vaadin Forum. Jedoch erfreut sich das Framework zum Zeitpunkt unseres Projekts keiner hohen Popularität, dadurch kann über die Zukunftssicherheit noch keine Prognose getroffen werden.

Java Server Faces ist ein Framework welches in JavaEE bereits integriert ist. Java Server Faces hat eine hohe Popularität. Dadurch besitzt es eine vergleichsweise hohe Zukunftssicherheit. Jedoch ist es wie das Spring Framework für Großprojekte gedacht und verursacht damit bei kleineren Projekten sehr viel Anfangsarbeit („Oberhead"). Zusätzlich benötigt es Einarbeitungszeit in weitere Basistechnologien, die das Framework nutzt. Die Strukturierung des späteren Projekts (wie auch bei Spring, durch das Model-View-Controller-Konzept realisiert) erhöht die Übersichtlichkeit und ermöglicht eine Wiederverwendung in anderen Projekten.

Spark ist ein „kleines" und einfache zu benutzendes Java Framework für Webanwendungen. Dabei handelt sich um ein Framework, welches für Echtzeitanalysen d.h. die schnelle Verarbeitung großer Datenmengen gedacht ist. Die Dokumentation ist für die Größe des Frameworks ausreichen. Die Popularität schätzt das Projektteam zum Projektstart relativ gering ein. Jedoch würde es für die Ansprüche im Twitter-Projekt, die geringste Einarbeitungszeit benötigen, da es sehr übersichtlich und intuitiv zu benutzten ist.

Auswahl: Vaadin schied als Innovativsoftware aus, da man keine Prognose für die Zukunftssicherheit machen kann. Somit wäre das Projekt nicht mehr erweiterbar, sollte sich das Framework nicht durchsetzen und vom Markt verschwinden. Spark wurde wegen des zu geringen Supports (Nutzergruppe zu klein) aus der Auswahl gestrichen. Java Server Faces und das Spring Framework befanden sich in der letzten Auswahl. Das Projektteam sah beide als gleichwertig im Bezug auf das gestellte Thema bzw. die gestellte Aufgabe an. Schlussendlich fiel die Entscheidung auf das Spring Framework, da es nicht nur in Web Anwendungen benutzt wird und das Team ihm eine größere Popularität/Zukunftssicherheit zutraut.

4.1.2 Spring als Grundlage für das Projekt

Die konkrete Implementierung ist maßgeblich von der Entscheidung für das Spring Framework geprägt. Dieses bietet einerseits eine einfache Möglichkeit zum Aufbau einer Servlet-basierten Webanwendung und andererseits eine flexible Datenbankschnittstelle. Dadurch ermöglicht es die Kapselung der eigentlichen Funktionalität in Java Servlets, während die Datenbank und die Website unabhängig davon aufgebaut werden können. Spring bietet einen modularen Aufbau, somit können verschiedene Spring Bibliotheken einzeln oder in Kombination verwendet werden. Für dieses Projekt kommen insbesondere folgende Module zum Einsatz:

- Die beiden Module `spring-core` und `spring-beans` bilden das Fundament des Spring Frameworks und sind somit unverzichtbar. Sie unterstützen das Konzept der Steuerungsumkehr (*Inversion of Control*) und die entsprechende Erzeugung von *Spring Beans*.
- Das `spring-jdbc` Modul unterstützt JDBC-basierte Anwendungen etwa bei dem Aufbau der Datenbankverbindung, der Konfiguration einer `Data Source` oder der

Erzeugung von *Prepared Statements*.

- Das `spring-webmvc` Modul wird für die Interaktion mit REST-basierten Webservices, sowie für die Implementierung einer MVC-Architektur benötigt.
- Das `spring-security` Paket stellt ein eigenständiges Spring Sub-Projekt dar und kann wiederum in diverse Module unterteilt werden. Es bietet in erster Linie verschiedene Methoden zur Authentifizierung und Autorisierung in J2EE-Anwendungen.

Spring Beans

Spring grenzt sich von anderen *Java Enterprise* Frameworks insofern ab, als dass keine standardisierten Komponenten, wie etwa EJBs, verwendet werden, sondern vielmehr eine lose Kopplung von POJOs[1] fokussiert wird. In diesem Zusammehang bezeichnet man die Objekte einer auf Spring basierten Anwendung auch als *Spring Beans*. Jede *Spring Bean* Instanz definiert sich über konfigurationsspezifische Metadaten, die entweder explizit (mittels Java oder XML) oder implizit (mittels *Autowiring*) festgelegt werden. In Tabelle 4.1 sind die wichtigsten Eigenschaften einer *Spring Bean* in Bezug auf Konstruktion, Lebenszyklus und Abhängigkeiten aufgelistet.

Tabelle 4.1: Konfiguration einer *Spring Bean* [3]

Eigenschaft	**Beschreibung**
`class`	Legt die Klasse einer *Bean* fest.
`name / id`	Indentifiziert jede *Spring Bean* eindeutig.
`scope`	Spezifiziert den Kompetenzbereich einer *Bean*. Unterschieden wird hierbei zwischen Singleton, Prototyp, Request und Session.
`constructor-arg`	Definiert die, bei einer Konstruktor-basierten *Dependency Injection* auftretenden Abhängigkeiten als Konstruktorargumente
`properties`	Definiert die, bei einer `Setter`-basierten *Dependency Injection* auftretenden Abhängigkeiten als Methodenparameter
`autowiring mode`	Aktiviert die Konfiguration von Abhängigkeiten mittels *Autowiring*.
`lazy-initialization mode`	Impliziert eine Erstellung der Instanz nach tatsächlichem Bedarf, im Gegensatz zu einer Konstruktion beim Starten des Programms.
`initialization method`	Definiert eine Methode, welche unmittelbar nach der Initialisierung einer *Bean* aufgerufen wird.
`destruction method`	Definiert eine Methode, welche unmittelbar nach der Destruktion einer *Bean* aufgerufen wird.

[1] Ein Plain Old Java Object (POJO) ist in diesem Kontext ein Objekt, das nicht mittels Oberklassen oder Interfaces spezialisiert ist. Es stellt somit die einfachste Form eines Objektes im Sinne der Objektorientierung dar.

Ein Anwendung für die explizite Angabe von Konfigurationsdetails via XML, findet sich am Beispiel dieses Projektes in der Definition des `ViewResolvers`, welcher in den Abschnitten 4.2.1 und 4.3 genauer betrachtet wird:

```
<bean id="ViewResolver"
        class="org.springframework.web.servlet.view.
            InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsps/"></property>
        <property name="suffix" value=".jsp"></property>
</bean>
```

Inversion of Control

Einhergehend mit der losen Kopplung von *Spring Beans*, werden deren Abhängigkeiten zu anderen Klassen zentral verwaltet. Man spricht hier von Steuerungsumkehr oder auch *Inversion of Control* (IoC). Darunter versteht man insbesondere die automatische Erstellung und Referenzierung von Objekten durch eine gemeinsame `ApplicationContext` Instanz. Abbildung 4.1 veranschaulicht dieses Konzept. Auf diese Art und Weise können

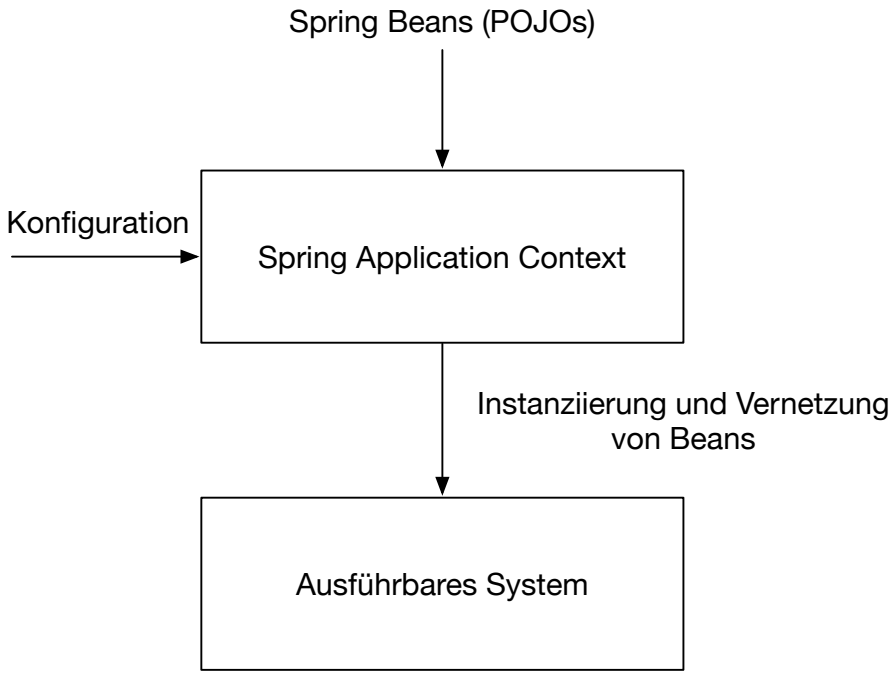

Abbildung 4.1: Grundprinzip einer Springanwendung [4]

Abhängigkeiten einer *Bean* unter Berücksichtigung der Konfigurationsdaten während der Laufzeit erzeugt werden. Dies wird im Allgemeinen auch als *Dependency Injection* bezeichnet. Man unterscheidet dabei zwischen einer Injizierung der Abhängigkeiten mittels Konstruktor (*Constructor Injection*) oder mittels einer eigens dafür vorgesehenen Methode (*Setter Injection*). Wie bereits erwähnt, kann die Konifguration der Beziehungen zwischen Beans automatisch erfolgen.

Autowiring

Autowiring bezeichnet im Wesentlichen die implizite Verknüpfung von *Spring Beans* durch das Spring-Framework selbst. Ziel dabei ist es den Konfigurationsaufwand zu minimieren. Um dies zu erreichen, muss der `ApplicationContext` Beans automatisch finden und als solche erkennen können. Dieser Prozess ist als *Component Scanning* in das Framework integriert, muss aber unter Angabe des Packages, in dem sich die zugehörigen Klassen befinden, explizit aktiviert werden:

```
<context:component-scan base-package="de.htwsaar.db">
            </context:component-scan>
<mvc:annotation-driven></mvc:annotation-driven>
```

Ebenso müssen Klassen, welche als *Bean* instanziiert werden sollen, mittels *Annotation* als solche gekennzeichnet werden. Das *spring-webmvc* Modul unterstützt hierbei verschiedene Annotationen, wie etwa `@Controller`, `@Service` oder `@Component`, welche die jeweilge Schicht der Architektur widerspiegeln. Darüber hinaus kann die Annotation durch einen Namen qualifiziert werden, sodass später eine eindeutige Zuordnung möglich ist.

```
@Component("keywordDao")
public class KeywordDao {
        ...
}

...

@Service
public class KeywordService {

        private KeywordDao keywordDao;

        @Autowired
        public KeywordService(KeywordDao keywordDao) {
                this.keywordDao = keywordDao;
        }
...
}
```

Im obigen Beispiel veranlasst die Annotation `@Autowired` nun ein Scannen der Komponenten. Wird hierbei die entsprechende Bean gefunden, kann diese instanziiert und referenziert werden.

4.2 Architektur

Open for extension, closed for modification [2] - Um einen hohen Grad an Flexibilität und Wartbarkeit zu erzielen, sowie eine Abhängigkeit von konkreten Klassen zu vermeiden, ist die Wahl eines geeigneten Architekturmusters essentiell. Besonders für Webanwendungen bietet sich eine Model-View-Controller (MVC) Architektur an, um die Interaktion zwischen Client und Server zu abstrahieren und die Kommunikation transparent zu gestalten.

Bei dem klassischen MVC unterscheidet man zwischen drei Komponenten:

- Das **Model** dient als Verarbeitungskomponente und enthält Daten sowie Kernfunktionalität.
- Die **View** stellt die Ausgabekomponente dar. Sie zeigt dem Anwender Informationen, welche vom Model bezogen werden.
- Der **Controller** steuert als Kontrollkomponente alle Eingaben und vermittelt diese zwischen Model und View.

Im folgenden Abschnitt wird die Umsetzung des MVC-Musters in Spring betrachtet. Insbesondere wird dabei klar, wie das Muster im Kontext *Web* funktioniert und wie dies die Anwendung hinsichtlich ihrer Erweiterbarkeit und Wartbarkeit verbessert.

4.2.1 Das Spring Web MVC Framework

Spring unterstützt die MVC Architektur als integralen Bestandteil des Frameworks. Basierend auf einigen Java-Konzepten wie beispielsweise *Annotations* oder *Dependency Injection* können somit komplexe Webanwendungen realisiert werden. Die Implementierung erfolgt dabei in leicht abgewandelter Struktur:

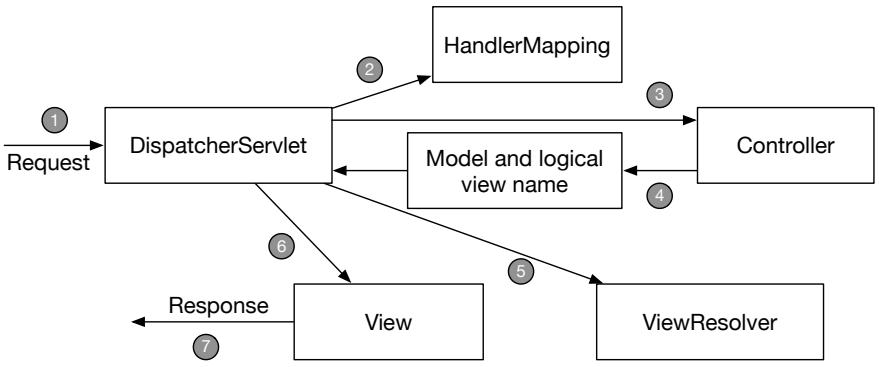

Abbildung 4.2: Komponenten in Spring MVC [1]

[2] Robert C.Martin, *The Open/ Closed Principle*

Das `Dispatcher Servlet` stellt hierbei das zentrale Element der Architektur dar. Dieses fungiert als Fassade, nimmt Anfragen vom Client entgegen und verteilt diese auf die jeweils zuständigen Controller. Die dabei notwendige Zuordnung zwischen Eingabe und Kontrollkomponente wird von einem `HandlerMapper` übernommen. Der `Controller` nimmt nun die Eingabe entgegen und verarbeitet die Daten. Meist werden dabei sogenannte `Service`-Klassen referenziert, welche die zur Berechnung notwendige Geschäftslogik enthalten. Werden bei der Datenverarbeitung Daten verändert oder erzeugt, so wird das zugehörige `Model` angepasst. Der `Controller` leitet dann dessen Referenz zusammen mit einem `View Name` an das `Dispatcher Servlet` weiter. Der `View Name` kann nun von einem `View Resolver` mit einer konkreten Implementierung identifiziert werden. Dies kann beispielsweise in Form einer JSP geschehen. Der `View` rendert schließlich die Daten und liefert ein `Response` Objekt zurück.

4.3 Frontend

Zur Steuerung der Anwendung durch den Benutzer dient eine Website. Die einzelnen Seiten werden durch ein gemeinsames, auch in gleichen Farben gehaltenes Design, das durch CSS organisiert wird, zusammengehalten. Alle Seiten besitzen eine Kopfsektion zur Verwaltung und eine Rumpfsektion für den eigentlichen Inhalt. Auf allen Seiten, mit Ausnahme der Startseite, befindet sich im Kopfteil ein Menü zur Navigation. Die eigentliche Funktionalität verteilt sich dabei auf die vier Seiten *Tweets*, *Keywords*, *Einstellungen* und *Anleitung*. Das Menü bietet außerdem eine *Logout*-Option.
Die Erstellung der Webpages erfolgt über Java Server Pages (JSP). Dabei handelt es sich um eine Programmiersprache zur dynamischen Erstellung von Webseiten durch einen Webserver, die auf HTML und Java basiert. JSPs erlauben die Einbettung von Java-Code in eine herkömmliche HTML-Webseite. In der Anwendung wird dies jedoch hauptsächlich zur Implementierung eines sicheren Logins verwendet. Ihr eigentlicher Verwendungszweck ergibt sich aus der Verwendung von Spring MVC. JSPs stellt eine Möglichkeit dar, eine View zu implementieren. Eine Client-Anfrage wird von einem Controller der Spring-Anwendung verarbeitet. Als Antwort erstellt der Server eine Webseite anhand einer JSP und einem Datenmodell. Auf die Daten des Models lässt sich dann im JSP etwa mit der sogenannten *Expression Language* zugreifen. Aber auch diese Funktionalität wird in der Anwendung kaum genutzt. Um zu häufiges Neuladen und Neukompilieren der Seiten zu verhindern, werden sie selbst durch Client-seitiges Javascript dynamisch gehalten. Eine Seite wird nur dann völlig neu geladen, wenn über das Menü auf sie gewechselt wird. Dabei werden lediglich die statischen Elemente, wie Menü und Schaltflächen, beim konkreten Seitenaufruf geladen. Die restlichen Elemente, die den eigentlichen Inhalt darstellen, werden im Nachhinein mit asynchronen Anfragen durch Javascript in JSON-Objekten nachgeladen.

In den folgenden Abschnitten werden die, für die unmittelbare Nutzerinteraktion relevanten Seiten hinsichtlich ihrer Funktionalität und Implementierung betrachtet.

4.3.1 Home

Der Nutzer betritt die Anwendung über die Startseite. Diese erfüllt die zwei Aufgaben Login und Registrierung. Die Seite ist ebenfalls in zwei Bereiche aufgeteilt, besitzt jedoch noch kein Menü, da die Navigation zu den meisten Seiten erst im Zusammenhang mit einem bestimmten angemeldeten Nutzer sinnvoll ist. Beim Aufruf der Seite befindet sich im Kopfbereich ein Login-Interface und es wird im unteren Bereich eine Animation angezeigt, die versucht den Umstand der täglichen Flut an Tweets, den die Anwendung adressiert, auf einen Nenner zu bringen. Bereits vor der Registrierung soll dem Benutzer so ohne große Erklärung klar werden, um was es sich bei der Anwendung handelt.
Im Login-Teil kann sich der Anwender mit einem Benutzernamen und einem Passwort anmelden. Dabei wird neben diesen beiden Werten außerdem ein CSRF-Token übergeben, um die Login-Sitzung vor entsprechenden Täuschungsversuchen zu schützen. Die Anmeldung der Anwendung nutzt den Login-Mechanismus von Spring Security. Nach dem Login ist der verwendete Benutzername für einen Controller aus dem Request einer Seite ersichtlich. Der Nutzer wird auf die Seite *Tweets* weitergeleitet, sobald seine Daten verifiziert sind.
Wenn noch kein Account existiert, kann der Nutzer ein neues Konto erstellen. Durch einen Klick auf die entsprechende Schaltfläche unter dem Login-Interface wird anstelle der Animation ein Registrierungsformular eingeblendet. Um die Absprungrate möglichst gering zu halten, werden vom Anwender nur wenige Daten erfragt. Außer einem Be-

nutzernamen und einem Passwort ist lediglich die Angabe einer Email-Adresse, an die eventuell Benachrichtigungen gesendet werden sollen, erforderlich. Diese Daten müssen jedoch auch bestimmten Richtlinien entsprechen. Der Benutzername muss etwa mindestens fünf Zeichen, das Passwort mindestens acht Zeichen lang sein. Und auch die Email-Adresse muss die richtige Form aufweisen. Registrierungsanfragen mit fehlerhaften Werten werden von der Anwendung ignoriert. Deshalb wird bereits im Frontend das Format überprüft und anhand von entsprechenden farblichen Markierungen und Meldungen auf Fehler aufmerksam gemacht. Nach der Registrierung wird die Startseite neu geladen, damit sich der Nutzer mit seinem neuen Account anmelden kann.

4.3.2 Tweets

Auf der Seite *Tweets* kann der Nutzer die für ihn gesammelten Tweets sichten. Der Hauptbereich der Seite ist dazu in einen Anzeigebereich, der die eigentlichen Tweets auflistet, und eine Sidebar, die verschiedene Möglichkeiten zur Bearbeitung der Anzeige bietet, eingeteilt.
Die Liste der Tweets wird dynamisch durch eine Javascript-Funktion generiert. Zur Darstellung eines Tweets gehören neben dem eigentlichen Content, also dem Text und einem eventuellen angehängten Bild, vor allem der Name und ein Bild des Autors sowie Entstehungszeit und -ort des Tweets. Zur Übertragung der Tweet-Daten stellt die Seite mittels Javascript eine Anfrage an den Webserver. Die Tweets werden in Form eines JSON-Objekts bereitgestellt, das einer entsprechenden Funktion übergeben wird. Diese Funktion erstellt für jeden Tweet einen Container, der an die angezeigte Liste angehängt wird. Zur weiteren Bearbeitung der aktuell geladenen Tweets werden diese außerdem lokal gespeichert. Die Seite *Tweets* lädt außerdem stets die Keywords des aktuellen Nutzers und hebt diese durch rote Farbe in den Tweet-Texten hervor.
Die Optionen in der Sidebar dienen teilweise dieser weiteren lokalen Bearbeitung von geladenen Tweets und teilweise dazu weitere Anfragen zu stellen. Die Seite lädt initial zunächst die hundert interessantesten Tweets des Benutzers. Mit dem Button „Tweets aktualisieren“ wird diese anfängliche Anfrage wiederholt, um die Anzeige auf den neuesten Stand zu bringen. Die aktuell geladenen Tweets werden durch die Eingabe eines Suchbegriffs in das entsprechende Eingabefeld gefiltert. Durch einen Klick auf den Button „Deep Search“ wird eine neue Anfrage gestellt, die alle Tweets des Benutzers mit diesem Suchbegriff liefert. Das Ergebnis kann weiter gefiltert werden. Die anfängliche Sortierung der Tweets nach ihrer Priorität kann auch gemäß ihres Alters erfolgen. Das Sortierkriterium wird in einem Drop-Down-Menü ausgewählt. In einem weiteren Drop-Down-Menü kann die Sprache der Tweets, die angezeigt werden, gewählt werden.
Eine weitere Funktionalität der Sidebar betrifft bereits die Festlegung von Keywords. Die beiden Buttons „Keyword“ und „Blacklist“ fügen den aktuell markierten Text in die entsprechende Liste hinzu. Dadurch kann der Nutzer bereits beim Sichten der Tweets weitere interessante Themen hinzufügen und vor allem unliebsame Themen, die sich mit interessanten Themen überschneiden, ausblenden.

4.3.3 Keywords

Die Seite *Keywords* ermöglicht die Verwaltung von Schlüsselwörtern und der Blacklist des Benutzers. Die Keywords werden links angezeigt. Die Blacklist steht rechts. Unter jeder der beiden Listen befindet sich ein Eingabefeld und ein Button zum Hinzufügen der Eingabe in die entsprechende Liste. Außer dem Keyword selbst gibt jede Zeile der Keyword-Liste die Priorität, die der Benutzer dem Begriff zugewiesen hat, in Form von

gelb ausgefüllten Sternen an. Die Begriffe auf der Blacklist müssen nicht priorisiert werden. In beiden Listen gibt es außerdem für jeden Begriff einen „Pause“-Button und einen „Delete“-Button. Wird ein Begriff auf der Keyword-Liste pausiert, so wird er in der Anzeige auf *Tweets* nicht mehr in Betracht gezogen. Ein pausierter Blacklist-Begriff wird nicht mehr herausgefiltert.

4.3.4 Einstellungen und Anleitung

Die *Einstellungen* dienen dazu die Daten des eigenen Benutzerkontos einzusehen und zu ändern. Dazu gehört vor allem die Möglichkeit sein Passwort und seine Email-Adresse zu ändern sowie die automatische Benachrichtigung von neuen Tweets zu aktivieren oder deaktivieren. Schließlich wird hier die Möglichkeit gegeben seinen Account unwiderruflich zu löschen. Bei allen Änderungen am Account wird durch einen Dialog sichergestellt, ob die Änderungen wirklich gewollt sind.
Obwohl darauf geachtet wurde den Zweck und die Funktionsweise der Anwendung möglichst intuitiv zu gestalten, kann nicht davon ausgegangen werden, dass jeder Nutzer die Möglichkeiten auf Anhieb erkennt. Deshalb werden auf der Seite *Anleitung* außerdem auch explizite Anweisungen gegeben, wie die Bedienung ablaufen sollte. Auch hier wird jedoch auf eine möglichst kurze Erklärung abgezielt.

4.4 Backend

Die Geschäftslogik einer Spring-Anwendung ist in den einzelnen, sogenannten Services, im weiteren Sinne des Begriffs, gekapselt. Dabei handelt es sich um herkömmliche Komponenten des `Application Contexts`. In der Anwendung lassen sich diese in drei Kategorien einteilen, die dem Aufbau der typischen 2-teiligen Architektur einer Streaming-Anwendung dienen. Die Klassen `StreamService` und `StatusService` sind dabei für den Aufbau und die Verwaltung des Streams zuständig. `TweetService` und `UserService` ermöglichen die Interaktion mit den Controllern und somit dem Benutzer am Frontend der Anwendung. Bei ihnen handelt es sich um die Services im engeren Sinne, die dementsprechend auch als solche annotiert werden. Die Schnittstelle zur Datenbank für beide Teile bieten eine Reihe von DAO-Services. In diesem Abschnitt sollen die konkreten Aufgaben dieser einzelnen Teile und verschiedene Alternativen zu ihrer Bewältigung mit ihren jeweiligen Problemen vorgestellt werden.

4.4.1 Controller und Services

In einer Spring-Webanwendung werden die ankommenden HTTP-Anfragen von den entsprechenden Controller-Methoden, die als RequestHandler annotiert sind, verarbeitet. Die Aufgabe eines Controllers beschränkt sich dabei darauf eine Darstellungsform, etwa in Form einer JSP-Datei, zusammen mit einem eventuell nötigen Datenmodell zu liefern. Gemäß der Spring-Dokumentation soll keine konkrete Funktionalität in den Controllern implementiert sein. Sie nutzen dafür die Services. Die Anfrageparameter für die Services, etwa bei der Erstellung eines neuen Keywords, erhalten die Controller-Methoden aus JSON-Objekten. In den meisten Fällen ist in der Anwendung jedoch keine Übergabe von Daten erforderlich. Um die Anfragen für einen bestimmten Benutzer durchzuführen, erhalten sie in der Regel lediglich den Benutzernamen der aktuellen Session.[3] Dieser kann von einem jederzeit verfügbaren Principal-Objekt angefordert werden [4].

Die Services einer Spring-Anwendung im engeren Sinne sind im Wesentlichen Fassaden, die die von den Controllern benötigten und von den anderen Services im weiteren Sinne des Wortes bereitgestellte Funktionalitäten zur Bearbeitung und Abfrage von Daten in einem übersichtlichen Interface bündeln. In der vorliegenden Anwendung handelt es sich dabei um TweetService und UserService. Da ihre Aufgabe sich darin erschöpft Daten von den Controllern in die Datenbank zu übertragen und umgekehrt, handelt es sich bei beiden Klassen lediglich um Wrapper, die Anfragen der Controller an die DAOs weiterreichen. Obwohl dieser Zwischenschritt über die Services somit im Grunde genommen überflüssig ist, wird er trotzdem genommen, um die Erweiterbarkeit der Anwendung sicherzustellen. Zusätzliche Bearbeitung von angefragten Daten durch das Backend könnte jederzeit eingefügt werden ohne die DAO-Klassen verändern zu müssen.Ein Beispiel einer solchen Erweiterung ist die Filterung von Tweets in `TweetService`. Tweets, die die übergebenen Begriffe nicht enthalten, werden aus dem Ergebnis entfernt.

[3] vgl. Abschnitt 4.8.1: Spring Security Module

[4] Die JSONs und das Principal-Objekt werden als Parameter des RequestHandlers angegeben und sind dann in dessen Kontext verfügbar

Die Aufgaben der Controller und Services im engeren Sinne werden vom *Spring Framework* bereits stark festgelegt. Da in der Interaktion mit dem Frontend keine besonderen Bearbeitungen der Daten durchgeführt werden, sind bei der Implementierung keine Schwierigkeiten aufgetreten.

4.4.2 Stream

Das Kernstück der Anwendung ist die Verbindung zu Twitter, über die sie mit neuen Tweets versorgt wird. Für den Stream selbst sind in der Anwendung die Komponenten `StreamService`, `TweetListener` und `StatusService` verantwortlich. Der `StreamService` stellt die eigentliche Verbindung zu Twitter her. Der `TweetListener` hört diese Verbindung ab und gibt empfangene Tweets an den `StatusService` zur Bearbeitung und Speicherung in der Datenbank weiter.

Twitter4J

Zum Aufbau der Verbindung zu Twitter wird die *Twitter4J* API verwendet, die auch das zugrundeliegende Beobachtermuster festlegt. Dabei handelt es sich um eine einfache Programmierschnittstelle, die den größten Teil der Funktionalität der Twitter REST und Streaming APIs in Form von Java-Klassen zur Verfügung stellt. Twitter nennt selbst zahlreiche alternative Bibliotheken in den unterschiedlichsten Programmiersprachen.[5] Neben Twitter4J wird hier als einzige weitere Java-basierte Bibliothek die Twitter-eigene *Hosebird* Client-Bibliothek *hbc* angeführt. Eine weitere Recherche führte außerdem zu den weiteren Alternativen *twitter-java* und *JTwitter*. Die letzteren beiden Möglichkeit stellten sich jedoch beide schnell als veraltet und nicht mehr aktuell heraus. Auch Twitter4J ist bereits etwas älter, aber offensichtlich noch auf einem relativ aktuellen Stand. Außerdem erfreut es sich einer gewissen Beliebtheit, so dass mit genügend Support gerechnet werden kann. Aber vor allem wird nur eine kurze Einarbeitungszeit benötigt. Die wesentlichen Funktionalitäten stehen schnell und einfach zur Verfügung. Twitters *hbc* wäre eine mögliche Alternative gewesen, doch die Entscheidung fiel auf Twitter4J.

StreamService

In der Anwendung wird der Stream in `StreamService` implementiert. Seine Aufgabe ist es sicherzustellen, dass zu jeder Zeit ein Stream existiert, der auf die aktuellen Keywords aller Benutzer ausgerichtet ist. Zu Beginn der Laufzeit wird `StreamService` als Komponente instantiiert. Im Konstruktor wird dabei der erste Stream erstellt und initialisiert. Die Konfiguration bei der Erstellung betrifft vor allem die *OAuth*-Authentifizierung, die die Anwendung gegenüber Twitter identifiziert [6]. Außerdem muss dem Stream ein Listener gegeben werden, der vor allem ankommende `Status`-Objekte weiterleitet. Für die eigentliche Initialisierung des Streams wird ihm schließlich ein `Filter` übergeben, der die notwendigen Suchparameter enthält. Hier sind das die Schlüsselwörter und die gewünschten Sprachen. Die Schlüsselwörter lädt der `StreamService` direkt mit einer entsprechenden Methode des `KeywordDao`. Von nun an empfängt der Listener alle Tweets, die den Keywords entsprechen.

[5] vgl. https://dev.twitter.com/overview/api/twitter-libraries

[6] *Oauth* ermöglicht eine API-Autorisierung mittels offenem Standard. Der Eigentümer einer Ressource kann den Zugriff auf diese mittels Zugangsberechtigung (typischerweise via tokenbasierter Multi-Faktor-Authentifizierung) kontrollieren

Obwohl der Aufbau der Verbindung selbst keine Probleme bereitet, gibt es jedoch gewisse Schwierigkeiten bei der Verwaltung der Verbindung über längere Zeit.

Aktualisierung der Keyword-Liste

Insbesondere die Aktualisierung der Keyword-Liste eines laufenden Streams stellt ein Problem dar. Wenn ein Benutzer ein neues Schlüsselwort zu seiner Liste hinzufügt, das noch nicht von einem anderen Nutzer gesucht wird, soll die Keyword-Liste des Streams möglichst zeitnah erweitert werden. Das Gleiche gilt bei einem bestehenden Keyword, für das sich kein Benutzer mehr interessiert, und somit nur unnötig die Bandbreite der Verbindung blockiert. Da es nicht möglich ist die Keyword-Liste eines laufenden Streams zu verändern, muss der Stream dazu neu gestartet werden. Der erste Lösungsansatz ist dementsprechend ein einfacher, regelmäßiger Neustart des Streams gewesen. Zu viele Neustarts in zu kurzer Zeit durch die gleiche Anwendung werden von Twitter jedoch mit einer Blockierung für eine bestimmte Zeit geahndet. Zudem besteht bei der großen Geschwindigkeit, mit der Tweets durch den Stream eintreffen, eine hohe Wahrscheinlichkeit, dass während eines Neustarts einige Tweets verpasst werden. Bei einem zu geringen Zeitintervall für den Neustart können so leichter Tweets verloren gehen und es besteht außerdem die Gefahr einer Blockierung durch Twitter. Wird das Zeitintervall hingegen zu hoch gesetzt, so werden die neuen Keywords der Benutzer nicht schnell genug auf den Stream angewendet. Auch dadurch gehen somit Tweets, die diesen Keywords entsprechen und in der Zwischenzeit entstehen, verloren.
Ein erster Schritt zur Lösung dieses Dilemmas liegt darin, den Neustart nur dann durchzuführen, wenn auch wirklich neue Keywords vorliegen. In einem weiteren Ansatz wurde ein expliziter Methodenaufruf durch die entsprechenden Methoden des `KeywordService`, die für Änderungen an den Keywords verantwortlich sind, durchgeführt. Diese Lösung ist jedoch nicht praktikabel, da bereits das mehrfache Hinzufügen von Keywords durch einen einzelnen Benutzer auch mehrfache Neustarts bedeuten und schnell eine Blockierung durch Twitter nach sich ziehen würde. Die Verknüpfung des Neustarts mit der Erstellung und dem Löschen von Keywords bedeutet also einen Kontrollverlust.
Stattdessen wird nun in regelmäßigen Abständen überprüft, ob ein Neustart aufgrund einer veränderten Liste notwendig ist. Dazu wird der `Scheduler` verwendet, der von *Spring-Core* bereitgestellt wird. Damit lässt sich eine bestimmte Methode in Intervallen oder zu bestimmten Zeitpunkten aufrufen.[7] Da das Hinzufügen und Entfernen von Keywords durch einen Benutzer eher zu den Ausnahmefällen gehört, lohnt sich die Prüfung, ob die Liste sich geändert hat. Die Prüfung kann ruhig öfter stattfinden, da ein Neustart nur selten erforderlich sein wird. Die Kosten der Prüfung selbst betragen vor allem eine Anfrage an die Datenbank. Diese liefert den eigentlichen Maßstab für die Wahl des Zeitintervalls. In der Anwendung werden die Keywords einmal in der Minute geprüft.

Unerwartete Abbrüche

Weitere Probleme im Zusammenhang mit der Aufrechterhaltung des Streams sind Ausnahmesituationen, in denen der Stream etwa nicht korrekt startet oder unerwartet beendet wird. Auch diese Möglichkeiten würden durch den regelmäßigen Neustart des Streams abgedeckt werden. Da dieser jedoch in der oben beschriebenen Lösung nicht mehr garantiert werden kann, wird auch ein unbedingter Neustart in einem größeren Zeitintervall von einem Tag durchgeführt. So wird sichergestellt, dass der Stream auch ohne Beobachtung über einen längeren Zeitraum erhalten bleibt.

[7] Die Methode muss lediglich mit einer entsprechenden Annotation gekennzeichnet werden.

TweetListener

Der `TweetListener` implementiert das `StatusListener-Interface` der Twitter4J-API. Er wird dem Stream des StreamService als Beobachter hinzugefügt. Jeder Tweet aus dem Stream führt zu einem Aufruf der Methode `onStatus()`, die dabei ein `Status`-Objekt erhält, das direkt weiter an den `StatusService` gereicht wird. Jegliche Bearbeitung wird dort durchgeführt.

Die weiteren Methoden des `TweetListener` reagieren auf Meldungen, die den Zustand des Streams betreffen. Auf dem aktuellen Stand des Projekts werden sie nicht verwendet. Sie könnten jedoch eingesetzt werden, um flexibler auf unerwartete Probleme, etwa mit einem Neustart des Streams, zu reagieren.
Während des Projekts kam die Vermutung auf, dass der Aufruf der `onStatus()`-Methode asynchron verläuft. Das hätte geheißen, dass jeder Tweet zur Erstellung eines neuen *Worker*-Threads führt. Entsprechend hätte die Möglichkeit bestanden, dass mehrere Threads zur gleichen Zeit einen `Status` an `StatusService` übergeben. Dies hätte zu Ressourcenkonflikten führen können. Die Annahme hat sich jedoch mittlerweile als unbegründet herausgestellt. Die einzigen asynchronen Nachrichten in der Anwendung werden durch `Scheduler`-Aufrufe gesendet.

StatusService

Die Aufgabe des `StatusService` ist die Erstellung von Instanzen der internen Datenmodelle für Tweets und Autoren aus den `Status`-Objekten, die er vom `TweetListener` erhält. Diese werden dann mit Hilfe der entsprechenden DAOs in die Datenbank übertragen.
Der `StatusStream` ist dabei zunächst dafür verantwortlich, dass im Falle eines Retweets der ursprüngliche Tweet aus dem `Status`-Objekt bearbeitet wird. Um die Anzahl der erfassten Tweets zu verringern, werden außerdem keine Retweets von älteren Nachrichten gespeichert. Tweets, die älter als zwei Tage sind, werden ebenfalls aussortiert. Dies betrifft insbesondere die Retweets.
Hinsichtlich seiner Aufgabe ähnelt der `StatusService` stark den Services im engeren Sinne mit der Unterschied, dass diese die Controller, also das Frontend, mit den DAOs verbinden. Aufgrund der erfahrungsgemäß zu erwartenden Geschwindigkeit von eintreffenden Tweets würde eine einfache Übergabe der Tweets und Autoren an das entsprechende DAO nach der Erstellung eine entsprechend hohe Anzahl von Datenbankverbindungen in kurzer Zeit bedeuten. Um diese zu reduzieren, werden Tweets und Autoren zunächst in einer HashMap zwischengespeichert, die regelmäßig mit einem Batch-Update in die Datenbank geladen werden. Besonders aktive Autoren mit vielen Tweets in kurzer Zeit und häufige Retweets eines bestimmten Tweets werden dabei auch nur einmal übertragen. Dies verbesserte die Performance beiläufig.

Obwohl mit einer sequentiellen Ankunft der Tweets gerechnet werden kann, ist die Klasse letztlich als Monitor implementiert. Dies liegt daran, dass auch hier der `Scheduler` eingesetzt wird, um in regelmäßigen, kurzen Abständen eine Methode aufzurufen, die die Tweets und Autoren aus den Buffern in die Datenbank lädt. Da diese Methode somit asynchron in einem eigenen Thread aufgerufen wird, kann es zu einem Resourcenkonflikt beim Aufruf kommen. Indem die beiden Methoden des `StatusService` als `synchronized` definiert werden, wird die gleichzeitige Ausführung beider Methoden und somit der gleichzeitige Schreib- und Lesezugriff auf die Buffer verhindert. Ohne den `Scheduler` könnte auch eine nicht-synchronisierte Implementierung verwendet werden, bei der die Buffer immer ab einer gewissen maximalen Größe geleert werden. Diese sequentielle und

somit einfachere Lösung hätte jedoch den Nachteil, dass nicht absehbar ist, wie schnell ein gesetztes Limit erreicht wird.[8] Es werden zwar Tweets erfasst, aber solange das Limit nicht erreicht ist, werden sie nicht in die Datenbank übertragen und sind auch nicht vom Benutzer einsehbar. Auch hier lässt sich durch den `Scheduler` die Kontrolle über die Ereignisse behalten. Die Anzahl der Datenbankzugriffe in einem bestimmten Zeitraum durch den `StatusService` kann so genau festgelegt werden.

4.4.3 Data Access Objects

Datenquelle mit Pooling

Die MySQL-Datenbank der Anwendung wird in der web.xml-Datei des Spring Frameworks als eine Datenquelle, ein `Datasource`-Objekt, konfiguriert, das damit für die Erstellung der DAOs als Komponente des `ApplicationContext` verfügbar ist. *Spring* übernimmt dabei die Verwaltung der Datenbankverbindungen in einem Verbindungspool.

Anfragen mit DAOs

Jedes DAO entspricht einer Klasse des Datenmodells. Ihre Aufgabe ist das Speichern, Laden und Bearbeiten der Daten von Objekten des jeweiligen Datentyps in der Datenbank. Spring bietet zur Durchführung von SQL-Anfragen JDBC-Template-Klassen an, die bei der Erstellung eines DAOs mit der in der web.xml-Datei definierten Datenquelle initialisiert werden. Für eine Anfrage an die Datenbank werden dem Template ein *Prepared Statement* [9] sowie eventuelle Parameter übergeben. Die Erstellung der angefragten Objekte aus den einzelnen Zeilen des Ergebnisses übernimmt ein `RowMapper`, der für eine bestimmte Anfrage-Methode jeweils als innere Klasse definiert wird. Die Trennung von Prepared Statement und Parametern verhindert eine *SQL Injection* durch eine User-Anfrage. Das JDBC-Template stellt sicher, dass die Parameter als solche eingefügt werden und nicht selbst als SQL-Code interpretiert werden.

DAOs im TwitterMonitor

In der Anwendung existieren vier DAOs für die Model-Klassen `User`, `Keyword`, `Author` und `Tweet`. Die DAOs der ersten drei Datentypen bieten keine größeren Besonderheiten. Die `TweetDAO` ist aufwändiger.

User Der `UserDAO-Service` wird nur dazu verwendet einen neuen Nutzer auf Anfrage des UserService in der Datenbank einzutragen oder zu löschen.[10] Die Abfrage nach vorhandenen Benutzern in der Datenbank beim Login wird automatisch von Spring-Security durchgeführt.[11] Das Einfügen eines Benutzers betrifft dabei zwei Tabellen, die (im Sinne des Spring Frameworks) einerseits den eigentlichen Nutzer und andererseits dessen unterschiedlichen Rollen festhalten. Da in der vorliegenden Anwendung nicht zwischen verschiedenen Rollen unterschieden wird, könnte darauf verzichtet werden. Das Spring Framework verlangt jedoch eine Rolle beim Login. Im Datenmodell User können die Daten der beiden Tabellen zusammengefasst werden, da es keinen Nutzer mit mehreren Rollen gibt.

[8] Bei der großen Menge an Tweets, die in einem realistischen Szenario vorliegt, ist dieser Punkt selbstverständlich releativ.

[9] Ein String mit einem SQL-Statement mit eventuellen Platzhaltern.

[10] Auf dem aktuellen Stand sind E-Mail-Benachrichtigung und E-Mail-Ändern noch nicht implementiert. Würde aber auch mit `insert on duplicate key` implementiert werden.

[11] vgl. Abschnitt 4.3

Keyword Das `KeywordDAO` liefert einerseits dem `StreamService` eine duplikatfreie Liste aller Schlüsselwörter, die in der Datenbank gespeichert sind.[12] Andererseits ist er auf Anfrage des `UserService` für das Einfügen, Entfernen und Laden von Schlüsselwörtern zuständig. Beim Einfügen von Duplikaten wird dabei stets ein Update der Priorität und des Aktiv-Status durchgeführt. Das Einfügen dient also auch der Änderung dieser Werte. Beim Laden der Liste eines Nutzers können entweder die positiven oder die negativen Schlüsselwörter angefragt werden.

Author Der `AuthorDAO` dient nur zum Einfügen von Autoren in die Datenbank, die zuvor anhand eines `Status`-Objekts erstellt werden. Um die Anzahl von Datenbank-Aufrufen zu reduzieren werden die erstellten Autoren dabei gepuffert und mit einem Batch-Update eingefügt. Wenn ein bereits vorhandener Autor seine Daten in der Zwischenzeit geändert hat, werden diese Änderungen übernommen. Da alle Ausgabeinformationen in `OutputTweet`-Objekten enthalten sind, die im `TweetDAO` erstellt werden, müssen die in der Datenbank vorhandenen Autorendaten nicht für sich abgefragt werden. Das Löschen der Autoren wird durch ein Event in der Datenbank sichergestellt.[13]

Tweet Mit dem `TweetDAO` werden zunächst die aus einem `Status`-Objekt erstellten Tweets in die Datenbank eingefügt. Insbesondere im Falle von Retweets sind schon entsprechende Tweet-Ids in der Datenbank vorhanden. Die Einträge werden dann stets auf den neuesten Stand gebracht.

Doch die eigentliche Herausforderung besteht in der Abfrage der Daten für die Ausgabe. Einerseits ist sowohl die Anzahl der Tweets an sich als auch die Anzahl der Tweets, die zu den Schlüsselwörtern eines Benutzers passen, so groß, dass eine entsprechende Datenbankanfrage sehr viel Zeit beansprucht. Andererseits enthält ein `OutputTweet` unter anderem auch eine Liste von Keywords, die nicht mit einer Anfrage erfasst werden können. In einer frühen Version wurde dafür eine gesonderte Abfrage für jeden einzelnen Tweet im `RowMapper` gemacht. In dieser Version wurde auch noch mit mehreren Bild-URLs pro Tweet gerechnet. Auch diese wurde für jeden Tweet in einer eigenen Anfrage geladen. Somit wurden für jeden Tweet zwei weitere Anfragen gemacht. Ab einer gewissen Anzahl von Tweets überlastete dies offensichtlich den Datenbankzugriff. Als Lösung wurde zunächst nur noch von höchstens einem Bild pro Tweet ausgegangen. Somit muss nur noch eine Liste für die Keywords erstellt werden. Dazu werden zunächst alle Tweets inklusive eines Schlüsselworts geladen. Bei mehreren Schlüsselwörtern pro Tweet ist dieser auch mehrfach in der Liste vertreten. Diese Liste wird durchlaufen. Dabei wird das jeweils erste Vorkommen eines Tweets mit seiner Tweet-Id als Schlüsselwert in einer HashMap hinterlegt und die Schlüsselwörter weiterer Vorkommen in die Keywordliste des Tweets in der HashMap eingefügt. Das Einfügen in die HashMap hat einen konstanten Aufwand. Somit ist der schlimmste Fall eine duplikatfreie Liste von Tweets mit einem linearen Aufwand.

Eine weitere Herausforderung besteht in der Einführung von auszuschließenden Schlüsselwörtern. Die dazu passenden Tweets werden in einer inneren SQL-Anfrage geladen und gelten für die eigentliche Anfrage als Ausschlusskriterium. Dies bedeutet immer auch eine doppelte Anfrage und somit doppelte Belastung der Datenbank.

Die Ausgabe der Tweets ist absteigend nach der Priorität geordnet und auf ein bestimmtes

[12] Ein TwitterStream der Twitter4J-API erfordert ein String-Array.
[13] vgl. Abschnitt 4.5.2

Limit beschränkt. So kann die maximale Anzahl der Tweets, die einem Benutzer angezeigt werden, festgelegt werden. Da die Such- und Sortiertfunktionen der Anzeige im Frontend jedoch allein in Javascript implementiert sind, ergibt sich damit das Problem, dass dem Nutzer nicht alle Daten zur Verfügung stehen.[14]

[14] Die Suchfunktion könnte etwa auch durch einen Controller-Aufruf in der Datenbank alle Tweets durchsuchen und so ein besseres Ergebnis liefert.

4.5 Datenbank

Als Datenbanksystem wird MySQL verwendet. Dies ist eine quelloffenes relationales Datenbankmanagementsystem (DBMS), welches schnell und verlässig arbeitet. Es bietet eine Menge Funktionalitäten, darunter eine Benutzerverwaltung, Unterstützung von Stored-Procedures/-Functions, Trigger, Events und eine Textindexierung. Letzteres ermöglicht ein schnelleres Mapping von Tweets und Keywords, was aufgrund der großen Datenmenge sehr nützlich ist.

Der Name der Datenbank lautet `twitter_monitor` und wird benutzt um Daten wie Tweets und Benutzerprofile des Projektes Twitter-Monitor zu speichern. Desweiteren hilft sie auch dabei Benachrichtigungen über interessante Tweets für die Nutzer anzulegen, welche dann über den Server per E-Mail versendet werden. Hilfreiche Funktionen zur Berechnung der Tweet-Priorität werden auch von der Datenbank angeboten.

4.5.1 Stored-Procedures und Stored-Functions

Stored-Procedures und -Functions werden direkt auf dem DBMS ausgeführt, bieten eine hohe Performance und ermöglichen auch komplexere Operationen einfach zu implementieren. Zusätzlich wird die Kommunikation mit der Serveranwendung verringert und damit auch der Netzwerkverkehr. Sie erhöhen außerdem die Wartbarkeit, weil der Code nur noch zentral an einer Stelle angepasst werden muss.

Tabelle 4.2: Stored-Procedures (P) und Stored-Functions (F)

Name	Beschreibung	Input	Output
`get_personal_prio` (F)	Errechnet die persönliche Priorität für einen Tweet passend zum Benutzer. Dafür wird die allgemeine Tweet-Prio und die vom Benutzer eingestellte Keyword-Prio mit einbezogen.	tweetId: bigint, username: varchar(60)	float: Errechnete Priorität
`calc_tweet_prio` (F)	Implementiert die Formel zur Priorisieren von Tweets	favoriteCount: int, retweetCount: int, followerCount: int	float: Priorität
`get_general_prio` (F)	Errechnet die Priorität für einen Tweet anhand der Tweet-ID	tweetId: bigint	float: Priorität
`user_exists` (F)	Überprüft, ob es bereits einen Nutzer mit dieser Email-Adresse bereits gibt.	username: varchar(60)	bool: 1:true / 0:false
`get_tweet_html` (F)	Hilfsfunktion, die ein Tweet mithilfe von HTML nachbaut wie auf der Webseite	tweetId: bigint	text: Tweet als HTML
`notify_users` (P)	Erstellt eine Benachrichtigung zu jedem Nutzer mit den Top-Tweets aus den letzten 12 Stunden, sofern der Nutzer dies aktiviert hat	username: varchar(60)	text: Top-Tweets als HTML
`get_top_tweets` (F)	Ermittelt zum übergebenen Nutzer die 5 besten Tweets aus den letzten 12 Stunden		
`check_preference` (F)	Überprüft ob ein Benutzer die entsprechende Einstellung gesetzt hat. Wenn ja gibt er den eingestellten Wert zurück, wenn nicht den Standard-Wert.	preferenceType: varchar(3), username: varchar(60)	int: Wert der Einstellung

Diese Funktionen kann man folgendermaßen in einem normalen SQL-Statement aufrufen:

```
select user_exists('oliverseibert'); -- Funktion
call notify_users(); -- Prozedur
```

Hinweis: Den Datentyp „bool“ gibt es zwar in MySQL, es wird jedoch als Ergebnis nach Aufruf der Funktion eine 0 oder 1 zurückgegeben.

4.5.2 Events

Damit der Event-Scheduler auch nach einem Neustart läuft wurde in der Konfigurationsdatei unter `mysqld` der Eintrag „event_scheduler=on“ hinzugefügt.

4.5.3 Trigger

Im Folgenden werden die Trigger im DBMS aufgelistet.

Tabelle 4.3: Events

Name	Beschreibung	Intervall
`delete_old_tweets`	Löscht Tweets, die älter sind als 5 Tage	Alle 720 Minuten
`notify_users_about_tweets`	Ruft die Prozedur notify_users() auf	Alle 720 Minuten, immer um 08:00 Uhr und um 20:00 Uhr

Tabelle 4.4: Trigger

Name	Beschreibung	Typ
`tweets_after_insert`	Trigger gleicht beim Einfügen eines neuen Tweets die Keywords mit dem neuen Tweet ab	after insert

4.5.4 Benutzer

Es wurden zwei Benutzer für die Benutzung der TwitterMonitor-Datenbank angelegt.

twitteruser: Wird von auf dem Server laufenden Programm Twitter-Monitor benutzt. Hat Select-Rechte auf alle Tabellen. Darf alle Stored-Procedures/Functions ausführen. Hat Update/Insert-Rechte auf die Entitäten users, authorities, keywords, tweetAuthors, tweets und tweetMedia. Jedoch beschränkt auf die Datenbank „TwitterMonitor“.

twitteradmin: Wird benutzt um die TwitterMonitor-Datenbank zu administrieren, wie etwa neue Benachrichtungstypen oder neue Benutzereinstellungen anzulegen. Oder auch um die Tweet-Tabelle zu leeren.
Hat Select/Update/Insert/Delete-Rechte auf alle Tabellen. Darf alle Stored- Procedures/Functions ausführen. Jedoch beschränkt auf die Datenbank „TwitterMonitor“.

4.5.5 Entitäten

Im Folgenden werden die Entitäten mit entsprechenden Informationen wie Primary Key (PK), Foreign Key (FK) und Not Null (NN) aufgelistet.

Tabelle 4.5: tweets: Speicherung der Tweet-Informationen aus Twitter.

Attribut	Beschreibung	Datentyp	PK	FK	NN
tweetId	Eindeutige ID, die von Twitter übernommen wird	bigint	X		X
autorId		bigint		X	X
text	Inhalt des Tweets	varchar(160)			X
favoriteCount	Anzahl der „Favourites“	int			X
retweetCount	Anzahl der „Retweets“	int			X
image	Url zu einem anhängenden Bild	varchar(100)			
place	Ort des Tweets	varchar(60)			
createdAt	Zeitpunkt, wann der Tweet auf Twitter veröffentlicht wurde	datetime			X
language	Sprache des Tweets	varchar(5)			

Tabelle 4.6: tweetAuthors: Informationen zu dem Benutzer, der den Tweet veröffentlicht hat.

Attribut	Beschreibung	Datentyp	PK	FK	NN
autorId	Eindeutige ID, die von Twitter übernommen wird	bigint	X		X
name	Benutzername von Twitter, wird mit „@“ angesprochen	varchar(20)			X
screenName	Name, wie er bei Twitter angezeigt wird	varchar(20)			
followerCount	Anzahl der Benutzer, die diesem Benutzer folgen	int			
pictureUrl	Url zum Profilbild	varchar(100)			

Tabelle 4.7: users: Stellt unsere Interessenprofile da. Notwendig für die Benutzerverwaltung.

Attribut	Beschreibung	Datentyp	PK	FK	NN
username	Eindeutiger Benutzername	varchar(60)	X		X
email	E-Mail Adresse	varchar(60)			X
password	Verschlüsselter Password-String	varchar(80)			X
enabled	Gibt an ob das Nutzerkonto aktiv ist oder nicht	tinyint			X
registeredAt	Zeitpunkt, wann das Interessenprofil angelegt wurde	datetime			

Tabelle 4.8: authorities: Stellt unsere Interessenprofile da. Notwendig für die Benutzerverwaltung.

Attribut	Beschreibung	Datentyp	PK	FK	NN
`username`	Der Benutzername	`varchar(60)`	X	X	X
`authority`	Die Autorität des Benutzers	`varchar(60)`			X

Tabelle 4.9: keywords: Schlagwörter, welche vom Benutzer festgelegt werden. Stellen die Interessen des Benutzers dar.

Attribut	Beschreibung	Datentyp	PK	FK	NN
`keyword`	Schlagwort	`varchar(50)`	X		X
`username`	Benutzername	`varchar(60)`	X	X	X
`active`	Gibt an, ob Tweets zu diesem Tweet dem Benutzer angezeigt werden sollen	`int`			X
`priority`	Priorisierung von 1 (niedrigste) bis 5 (höchste)	`int`			X
`positive`	Gibt an, ob das Keyword auf der Whitelist (1) oder Blacklist (0) steht	`int`			X
`registeredAt`	Zeitpunkt, wann das Keyword angelegt wurde	`datetime`			

Tabelle 4.10: tweets_x_keywords: Kreuztabelle zwischen benutzer und keywords. Hier steht drin, welche Tweets zu welchen Keywords passen.

Attribut	Beschreibung	Datentyp	PK	FK	NN
`tweetId`	Eindeutige ID eines Tweets	`bigint`	X	X	X
`keyword`	Schlagwort	`varchar(50)`	X	X	X

Tabelle 4.11: preferences: Liste von Einstellungen, die der Benutzer zur Verfügung hat

Attribut	Beschreibung	Datentyp	PK	FK	NN
`preferenceType`	Eindeutiges Kürzel mit dem Typ der Einstellung	`char(3)`	X		X
`descr`	Beschreibung der Einstellung	`varchar(60)`		X	
`defaultValue`	Standardwert zu einer Einstellung	`int`			X

Tabelle 4.12: user_x_preferences: Kreuztabelle zwischen Benutzer und Einstellungen. Hier erfolgt ein Eintrag, wenn der Eintrag vom Standardwert abweicht.

Attribut	Beschreibung	Datentyp	PK	FK	NN
`username`	Benutzername	`varchar(60)`	X	X	X
`preferenceType`	Kürzel mit dem Typ der Einstellung	`varchar(60)`	X	X	X
`value`	Benutzerspezifischer Wert zu einer Einstellung	`int`			

Tabelle 4.13: notificationType: Liste von Benachrichtigungsarten, die vom System unterstützt werden (z.B. E-Mail)

Attribut	Beschreibung	Datentyp	PK	FK	NN
`type`	Eindeutiges Kürzel mit der Benachrichtigungsart	`char(3)`	X		X
`descr`	Beschreibung zur Benachrichtigungsart	`varchar(60)`			X

Tabelle 4.14: notifications: Jeder Eintrag stellt eine Benachrichtigung für einen Nutzer dar

Attribut	Beschreibung	Datentyp	PK	FK	NN
`notificationId`	Fortlaufende eindeutige Benachrichtigungs-ID	`int`	X		X
`username`	Benutzername	`varchar(60)`		X	X
`type`	Kürzel mit der Benachrichtigungsart	`char(3)`		X	X
`subject`	Betreff	`varchar(60)`			X
`body`	Inhalt der Nachricht	`text`			X
`createdAt`	Zeitpunkt, an dem die Benachrichtigung erstellt wurde	`datetime`			
`sentAt`	Zeitpunkt, an dem die Benachrichtigung versendet wurde	`datetime`			

4.6 Tweet-Priorisierung

In diesem Abschnitt soll dargestellt werden, wie die Tweets bewertet bzw. priorisiert werden. Mithilfe eines selbst geschriebenen Algorithmus wird dies im Twitter-Monitor realisiert. Die Schwellwerte für die Berechnung der Priorität stammen aus Recherchen zu den Themen wie Twitter von den Usern benutzt wird (https://sysomos.com/inside-twitter/twitter-retweet-stats, Stand: 24.09.2016) und Erfahrungen von anderen Personen, die bereits versuchten einen Algorithmus zur Bewertung von Tweets aufzustellen (http://www.dasheroo.com/blog/whats-a-good-twitter-engagement-rate-for-your-tweets, Stand: 24.09.2016). Mit eigenen Erfahrungswerten wird das abstrahierte Verfahren Schritt für Schritt verfeinert. Dies wird im Folgenden aufgezeigt.

Ziele: Durch eine Priorisierung der Daten sollen dem Nutzer eine selektierte Auswahl von wichtigen bzw. interessanten Tweets angezeigt werden.

Kriterien: zur Priorisierung der Daten aufgelistet:

- Anzahl Retweets des Tweets
- Anzahl Likes des Tweets
- Anzahl Follower des Tweet-Autors
- Alter des Tweets
- Interesse des Benutzers

Schritt 1: Algorithmus aufstellen

Likes und Retweets werden ins Verhältnis zu der Anzahl Followern gestellt. Retweets sind unserer Meinung nach wichtiger als Likes, deswegen haben diese einen entsprechenden Faktor. Zuletzt wird mit 100 multipliziert, damit das Ergebnis nicht so viele Nachkommastellen hat und bei wichtigen Tweets etwa bei 1 liegt:

$$\frac{\frac{Likes}{2} + Retweets \cdot 2}{Follower} \cdot 100 \quad \textit{Priorisierung eines Tweets} \tag{4.1}$$

Das Ergebnis wird folgendermaßen in fünf Prioritätsstufen unterteilt (1 = höchste bis 5 = niedrigste):

- Prio 1: >= 1
- Prio 2: >= 0,5
- Prio 3: >= 0,1
- Prio 4: >= 0,05
- Prio 5: < 0,05

Je nach Alter des Tweets wird dieser anders bewertet. Dies geschieht mithilfe eines Faktors in folgender Staffelung:

Auffälligkeiten

- **Problem:** Anzahl Likes und Retweets sind am Anfang nicht vorhanden.
  Lösung: Tweets werden nach einer gewissen Zeit nochmals abgefragt. Zusätzlich wird, wenn ein Retweet eines vorhandenen Tweets über den Stream eingeht, die Daten des Original Tweets aktualisiert.
- **Problem:** Errechnete Priorität ist mit dieser Formel nicht optimal. Beispielsweise sind Tweets mit nur 2-3 Retweets und keinem Likes und dessen Autor nur 1 Follower hat, viel zu hoch priorisiert. Bsp.: Likes = 30, Retweets = 2, Follower = 1 führt zu einer Prio von 1900. Es handelt sich hierbei oft um Spam oder Werbung.
  Lösung: Die Statistik wird geglättet, indem die Anzahl Follower mit einer festen Konstante addiert werden. Eventuell kann auch die Anzahl der Follower in der Formel mithilfe einer Exponentialfunktion oder Wurzelfunktion unterstützt werden.
- **Problem:** Durch Unterteilung der errechneten Priorität in verschiedene Stufen, gehen Informationen verloren
  Lösung: Unterteilung in Prioritätsstufen wird nicht mehr durchgeführt.

Schritt 2: Algorithmus nach Erfahrungswerten verfeinern

Von der Anzahl der Follower wird nun die Wurzel gezogen. Dies hat zur Folge, dass Twitter- Nutzer mit wenig Followern abgeschwächt werden. Dasselbe wird auch durch die zusätzliche Addition mit 100 erreicht, zusätzlich ist hier eine statistische Glättung mit einbegriffen. Im Vergleich zur alten Formel sind die Retweets ein wenig abgeschwächt, da öfters Tweets aufgetaucht sind, die nur darauf abgezielt haben so viele Retweets wie möglich zu ernten.

$$\frac{\frac{Likes}{2} + Retweets}{\sqrt{Follower} + 100} \quad \textit{Priorisierung eines Tweets, zweiter Versuch} \tag{4.2}$$

Schritt 3: Tweets nach Interessen des Benutzers einstufen

Der Benutzer hat die Möglichkeit jeden seiner Keywords eine Wichtigkeit κ zuzuteilen. Dabei reicht die Skala von 1 (wenig interessant) bis 5 (sehr interessant). Wenn ein Tweet auf mehrere vom Benutzer eingestellten Keywords zutrifft, werden diese addiert. Anschließend wird mit 0,2 multipliziert. Dies ist ein ausgedachter Wert, damit bei einer

Alter	Faktor
< 1 Tag	1
< 2 Tage	0,9
< 3 Tage	0,6
< 5 Tage	0,3
> 5 Tage	0,1

Tabelle 4.15: Staffelung von Tweets nach Alter

Wichtigkeit von 5 der Tweet die gleiche Priorität wie vorher hat.

$$\frac{\frac{Likes}{2} + Retweets}{\sqrt{Follower} + 100} \cdot 0.2 \cdot \kappa \quad \textit{Individuelle Priorisierung eines Tweets} \tag{4.3}$$

Beispiel:

> Eingestellte Keywords: BVB (Wichtigkeit=5), Bundesliga (Wichtigkeit=3)
> Tweet: „BVB empfängt zum Auftakt in der Bundesliga Mainz 05“
> Allgemeine Tweet-Priorität: 10
> Benutzer-spezifische Priorität: $10 * ((5 + 3) * 0,2) = 16$

Die Priorität der Tweets wird somit erst berechnet, wenn der Benutzer diese braucht und nicht mehr beim Eintreffen eines jeden einzelnen Tweets. Dies soll die Performance erhöhen und Last vom Server nehmen.

4.7 Benachrichtigungen

Dieser Abschnitt handelt davon, wie Benutzer automatisiert über interessante Tweets informiert werden.

Um dies zu realisieren gibt es die Entitäten `notifications` und `notificationType`. Hier werden Informationen gespeichert wie Betreff, Inhalt, Empfänger, Benachrichtigungsart und ein Zeitstempel, der angibt, wann die Benachrichtigung versendet wurde. Letzteres dient zu gleich auch zur Überprüfung, ob die Benachrichtigung noch zum Versand ansteht.

Mit einem Datenbank-Event werden täglich um 08:00 Uhr und um 20:00 Uhr die fünf interessantesten Tweets für jeden Benutzer ermittelt und in die Entität `notifications` eingetragen. Der Inhalt ist im HTML-Format. Dies soll sicherstellen, dass die Tweets in der E-Mail dem Benutzer optisch genauso wie auf der Webseite präsentiert wird.

In der JAVA-Anwendung werden die noch ausstehenden Benachrichtigungen in der Klasse `NotficationDao` mithilfe der Funktion `getNotificationsNotSent()` ermittelt und als Liste mit Objekten vom Typ `Notification` zurückgegeben.

Nun wird noch eine Methodik benötigt, dass in einem gewissen Zeitintervall, die noch ausstehenden Benachrichtigungen ermittelt und anschließend versendet. Diese Funktionalität wird im `NotificationService` implementiert. Über die eingebundenen Klassen `Timer` und `TimerTask` aus dem `java-util`-Package lässt sich ein Dienst realisieren, der in einem angegebenen Zeitintervall immer wieder einen bestimmten Code ausführt.

Der Dienst lässt sich über die Methoden `start()` und `stop()`-Methode starten und beenden. Wenn der Dienst läuft und eine Benachrichtigung zum Versand ansteht, wird zuerst überprüft, welche Benachrichtigungsart gesetzt ist. Anschließend wird die entsprechende Strategie gestartet und der Benutzer erhält seine Neuigkeiten.

Sobald eine Benachrichtigung versendet wurde, wird sie entsprechend markiert. Hierzu wird der Zeitstempel mit der aktuellen Uhrzeit zur entsprechenden Benachrichtigung in der Datenbank gesetzt.

4.7.1 E-Mail Benachrichtigungen

Im Folgenden wird die Funktionsweise der Benachrichtigungen über E-Mail erläutert.

Jeder Benutzer hat eine E-Mail-Adresse hinterlegt. Diese wird als Empfänger-Adresse genutzt, wenn Benachrichtigungen mit der entsprechenden Benachrichtigungsart zum Versand anstehen.

Um den Versand per E-Mail zu realisieren wurde eine Google-Mail-Adresse angelegt und die Klasse `MailSender` implementiert. Diese meldet sich zuerst bei Google mit Benutzername und Passwort an und versendet dann die E-Mail.

In Java können Mails am einfachsten über das SMTP Protokoll versendet werden. Weltweit gibt es viele Mail-Dienste, die Mail mittels des SMTP Protokolls versenden. Einer der bekanntesten ist der Mail-Service "GMail"vom Internetkonzern Google.
Mit rund einer Milliarde Nutzer, ist GMail der meist genutzte E-Mail Dienst weltweit. Da

bei dieser Anzahl von Nutzern, sowohl die Zuverlässigkeit des E-Mail Transport, als auch die Sicherheit der E-Mail Übertragung gewährleistet sein muss, ist unsere Wahl auf diesen Service gefallen.

4.8 Sicherheitsaspekte

Jede Anwendung muss eine sichere Umgebung zur Verfügung stellen. Vor allem Webanwendungen bedürfen oft besonderer Sicherheit, da unsichere Implementierungen mit einigen Tricks umgangen werden können und Nutzer dadurch Zugriff auf Daten erhalten können, welche nicht für sie bestimmt sind. Einige Fragen müssen durchdacht werden, um eine nachhaltige Implementierung der Sicherheitsmechanismen zu gewährleisten:

- Wie werden Authentifizierung und Authorisierung sichergestellt?
- Müssen beim Austausch des / der Server die Sicherheitsmechanismen neu implementiert und / oder die Sicherheitseinstellungen neu konfiguriert werden?
- An welchen Stellen der Anwendung werden sensible Daten übertragen und bedürfen kryptografischer Verschlüsselung?

Dieser Abschnitt zeigt, wie wir das Spring Framework nutzen, um eine sichere Umgebung und damit auch alle damit in Verbindung stehenden Sicherheitsanliegen gewährleisten.

4.8.1 Spring Security Module

Das Spring Framework stellt den Service Spring Security zur Verfügung, welcher grundsätzlich genutzt werden kann, um alle oben genannten Problemstellungen zu lösen. Insgesamt stellt Spring Security folgende elf Module zur Implementierung von Sicherheitsmechanismen bereit:

Tabelle 4.16: Spring Security Module [1]

Modul	**Beschreibung**
ACL	Kapselt den Zugriff auf Geschäftsobjekte mittels Zugriffssteuerungslisten
Aspects	Untestützt die Verwendung einer AspectJ-basierten Implementierung anstelle des Spring AOP Standards im Rahmen von Spring Security.
CAS Client Support	Bietet eine Single Sign-on Authentifizierung (“ Einmalanmeldung“), die mittels der Central Authentication Service (CAS) Technik umgesetzt wird.
Configuration	Unterstützt die Konfiguration von Spring Security via Java und XML (Java wird seit Spring Security 3.2. unterstützt)
Core	Liefert die essentielle Spring Security Bibliothek
Cryptography	Bietet Methoden zur Verschlüsselung und Passwortcodierung
LDAP	Unterstützt eine LDAP-basierte Authetifizierung
OpenID	Unterstüzt eine zentrale Authentifizierung mittels OpenID.
Remoting	Sorgt für die Integration von verteilten, auf Spring basierenden Systemen.
Tag Library	Die Spring Security JSP Tag Bibliothek.
Web	Bietet eine, auf dem Spring Security *Filter* basierende Technik, zur Unterstützung von Web-Sicherheitsaspekten.

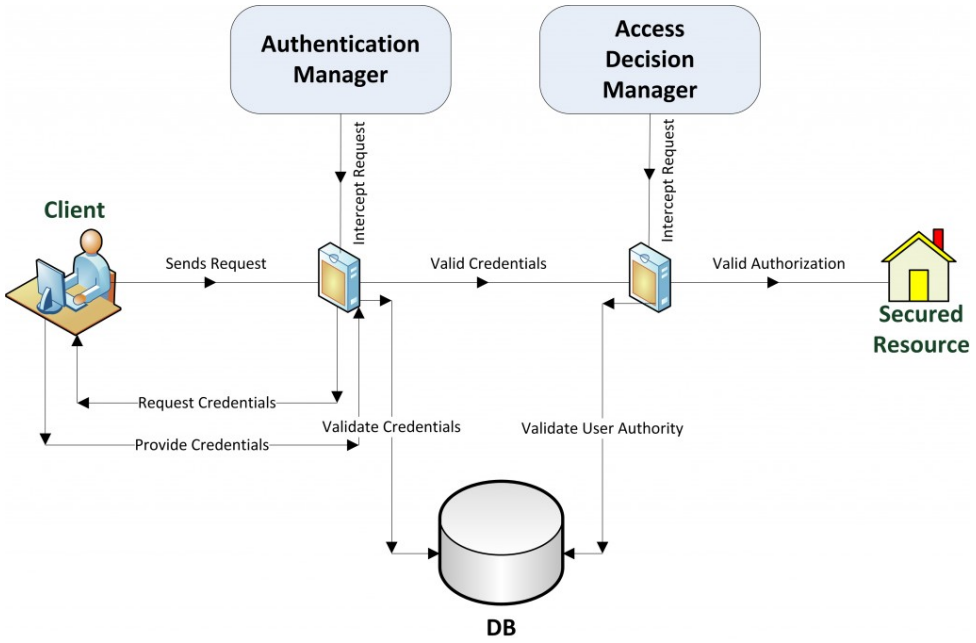

Abbildung 4.3: Authentifizierung und Validierung der Privilegien eines Nutzers [5]

Normalerweise werden nicht alle Module benötigt. Auch bei uns werden nur die Module Core, Configuration und Web eingesetzt. Der in einem vorherigen Kapitel vorgestellte Service Maven wird an dieser Stelle eingesetzt, um sämtliche benötigte Module, welche als .jar Dateien geliefert werden, zu importieren.

4.8.2 Implementierung und Funktionsweise

Um Spring Security aktiv in der Anwendung nutzen zu können muss als erstes eine springSecurityFilterChain Klasse implementiert werden. Wie der Name der Klasse schon sagt, handelt es sich dabei um einen Filter, der jegliche ankommende Webanfragen durch alle implementierten Sicherheitsmechanismen laufen lässt, sozusagen verkettet, und den Einsatz von Spring Security so erst ermöglicht. Abbildung 4.3 stellt den grudsätzlichen Ablauf einer Anfrage und deren Filterung durch Sicherheitsmechanismen dar.

Stellt ein Client nun eine Anfrage an eine Ressource, wird diese erst durch einen implementierten Authentication-Manager gefiltert. Die verschiedenen Unterfilter werden durch Authentication-Provider realisiert. Der Authentication-Provider prüft die beim Login eingegebenen Daten auf ihre Richtigkeit. Sind die Daten nicht im Code verankert und in einer Datenbank gespeichert, wird hier zusätzlich ein Jdbc-user-service benötigt, welcher die nötigen SQL-Anfragen zum Abgleichen der Nutzerdaten als Parameter erwartet. Hat der Abgleich mit der Datenbank stattgefunden und die Daten waren korrekt, wird die Anfrage weitergeleitet. In Abbildung 4.3 wird der folgende Prozess durch den Access Decision Manager dargestellt. Dieser ist in unserer konkreten Anwendung nicht implementiert, jedoch findet der Authorisierungsprozess hier -mittels Datenbankabfrage- trotzdem statt. An dieser Stelle muss überprüft werden, ob der Nutzer die nötigen Privilegien besitzt, um eine gewisse Seite aufrufen zu dürfen. Dieser Abgleich wird ebenfalls mit in der Datenbank gespeicherten Daten durchgeführt. An dieser Stelle gibt es folgende verschiedene Privilegien, die Spring Security zur Authorisierung zur Verfügung stellt:

Tabelle 4.17: Die verschiedenen von Spring zur Verfügung gestellten Privilegien [1]

Privileg	Beschreibung
`hasRole([role])`	Gibt an, ob der aktuelle Benutzer die spezifizierte Rolle innehat
`hasAnyRole([role1,role2])`	Gibt an, ob der aktuelle Benutzer eine der angegebenen Rollen innehat
`principal`	Erlaubt direkten Zugriff auf den aktuellen Benutzer
`authentication`	Erlaubt direkten Zugriff auf das aktuelle `Authentication` Objekt
`permitAll`	Erlaubt einen uneingeschränkten Zugriff auf eine Ressource
`denyAll`	Verweigert den Zugriff auf eine Ressource
`isAnonymous()`	Gibt an, ob es sich bei dem aktuellen Benutzer um einen anonymen Benutzer handelt
`isRememberMe()`	Gibt an, ob der aktuelle Benutzer ein sogenannter “ remember-me user„ist, also bereits zu einem früheren Zeitpunkt authentifiziert wurde.
`isAuthenticated()`	Gibt an, ob der aktuelle Benutzer authentifiziert ist.
`isFullyAuthenticated()`	Gibt, an ob der aktuelle Benutzer authentifziert ist und die “ remember-me“ aktiviert hat

Wird auch dieser Abgleich erfolgreich durchgeführt, erhält der Nutzer Zugriff auf die Ressource. Das folgende Codesegment veranschaulicht unsere Implementierung, dieser Sicherheitsmechanismen:

```
<security:authentication-manager>
    <security:authentication-provider>
        <security:jdbc-user-service
            data-source-ref="dataSource"
            authorities-by-username-query=
                'select * from authorities where binary username = ?'
            users-by-username-query=
                'select * from users where binary username = ?'
            id="jdbcUserService"/>
    </security:authentication-provider>
</security:authentication-manager>

<security:http use-expressions="true">
    <security:csrf disabled="true" />
    <security:intercept-url pattern="/" access="permitAll"/>
    <security:intercept-url pattern="/newAccount" access="permitAll"/>
    <security:intercept-url pattern="/showTweets"
                            access="isAuthenticated()" />
    <security:intercept-url pattern="/changePriority"
                            access="permitAll"/>
    <security:form-login login-page="/"
              login-processing url="/j_spring_security_check"
              password-parameter="j_password"
              username-parameter="j_username"
              default-target-url="/showTweets"/>
</security:http>
```

Das durch den `AuthenticationManager` gekennzeichnete Codesegment zeigt die Implementierung des Validierungsprozesses der Logindaten. Anhand des zweiten Abschnitts des Codesegments lässt sich erkennen, dass in unserer Anwendung z.B. die Startseite und die Seite zum Erstellen eines neuen Accounts von jedem Nutzer aufgerufen werden darf, da das `access` Attribut hier auf `permitAll` gesetzt wurde. Das Anzeigen der Tweets benötigt jedoch erst eine Authentifizierung, da der access Parameter den Wert `isAuthenticated()` annimmt. Natürlich könnten hier auch spezielle Bereiche für Administratoren eingerichtet werden. Das access Attribut müsste dort lediglich dementsprechend parametrisiert und die Autorität in der Datenbank hinterlegt werden.

4.8.3 Verschlüsselung sensibler Daten

Die vorherigen Abschnitte haben gezeigt, dass Spring Security in jeder Situation eine Lösung für die geforderte Problemstellung liefert. Auch im Bezug auf das Verschlüsseln von sensiblen Daten wie Passwörtern stellt das Modul eine einfache Lösung bereit. Da in unserer Anwendung das Anlegen von Accounts möglich sein soll und Passwörter aus Sicherheitsgründen nicht im Volltext in der Datenbank abgelegt werden dürfen, wird hier eine angemessene Verschlüsselungsmethode benötigt. Diese Verschlüsselung wird durch eine von Spring bereitgestellte passwordEncoder Klasse realisiert. Diese ist in der Lage, die eingegebenen Passwörter mithilfe folgender Algorithmen zu verschlüsseln:

1. *Bcrypt*: Eine auf dem Blowfish-Algorithmus basierende Funktion
2. *SHA*: Ein Verschlüsselungsalgorithmus der SHA-1 Familie
3. *SHA-256*: Ein Verschlüsselungsalgorithmus der SHA-2 Familie
4. *Md4*: Eine auf 32-Bit Systemen effektive Hashfunktion welche in 128-Bit verschlüsselt
5. *Md5*: Eine nicht mehr als sicher geltende Hashfunktion, die ebenfalls einen 128-Bit-Hashwert erzeugt

Wir haben uns an dieser Stelle für die Verschlüsselung der Passwörter mittels der SHA-256 Hashfunktion entschieden, da diese, im Gegensatz zu den meisten anderen oben genannten Funktionen, einen 256-Bit- Hashwert erzeugt, immer noch als stabil gilt und wir im Laufe des Studiums bisher nur mit den SHA- Hashfunktionen in Kontakt gekommen sind.
Das Einbinden der `passwordEncoder` Klasse findet ebenfalls im Authentication-manager Abschnitt statt und erfordert nur wenige Zeilen:

```
<security:authentication-provider>
            <security:jdbc-user-service
                data-source-ref="dataSource" />
            <security:password-encoder hash="sha-256"
                ref="passwordEncoder">
            </security:password-encoder>
</security:authentication-provider>
```

Hierzu wird ein weiterer Authentication-Provider Abschnitt erzeugt, in dem sowohl die Datenbank als auch der `passwordEncoder` und die Art des Hashes als Parameter übergeben werden. Des weiteren muss die Klasse selbst eingebunden werden:

```
<bean id="passwordEncoder"
    class="org.springframework.security.crypto
          .password.StandardPasswordEncoder">
</bean>
```

Die von einem Nutzer eingegebenen Passwörter werden dann mittels SHA-256 verschlüsselt und in der Datenbank hinterlegt.
Abschließend bleibt zu sagen, dass die meisten von uns genutzten Klassen und in diesem Kapitel erläuterten Methoden nur eine Option darstellen, um jeweilige Problemstellungen zu lösen. Spring Security ist ein mächtiger Service, der für jede Situation und Umgebung die nötigen Vorraussetzungen liefert, sichere und serverunabhängige Anwendungen zu schreiben. Des weiteren müssen von Spring- Security bereitgestellte Klassen nicht zwangsläufig verwendet werden. Zum Beispiel wäre es möglich, einen eigenen Filter zu implementieren und dabei auf die `springSecurityFilterChain` Klasse zu verzichten.

4.9 Deployment

Der Rechner auf dem die Anwendung läuft befindet sich im Softwarelabor der HTW-Saar. Im Folgenden wird die verbaute Hardware und die vorgenommenen Anpassungen etwas näher erläutert.

4.9.1 Hardwarespezifikationen

Betriebssystem: Windows 7 (64-bit Version)

Prozessor: Intel Core i7-4770

Arbeitsspeicher: 32 GB

Festplatten: Der Rechner bootet von einer 256 GB großen Solid State Drive. Auf dieser sind auch alle wichtigen Programme installiert die benötigt werden um die Anwendung starten zu können.
Des Weiteren sind zwei 2 TB große Hard Drive Disks verbaut. Diese dienen als Speicher für die My-SQL Datenbank, die dort alle registrierten User, die gesammelten Tweets und alle weitere Informationen des Tweets dort speichert.

4.9.2 Angepasste Einstellungen

Da der Arbeitsspeicher für die Anwendung auf 32 GB aufgestockt wurde, können einige Anpassungen vorgenommen werden um die Performance von Datenbankabfragen zu verbessern.

Folgende Datei wird dafür abgeändert:

C:\ProgrammData\MySQL\MySQL Server 5.7\my.ini

Im Original hat die Datei den folgenden Aufbau:
query_cache _size = 16M
key_buffer_size = 256M
innodb_buffer_pool_size = 8M

Die Änderungen an dieser Datei sehen wie folgt aus:
query_cache _size = 512M
key_buffer_size = 8G
innodb_buffer_pool_size = 8G

Erläuterung der einzelnen geänderten Parameter:

query_cache _size:
Die Größe des Speichers, die von MySQL für die kompletten Ergebnisse von häufig verwendeten Abfragen-Cache verwendet. Der Standardwert ist 0, was bedeutet, dass Query-Caching deaktiviert ist.

key_buffer_size:
Die Menge des verwendeten Speichers von häufig verwendeten Indizes. Dieser Cache ist global und von allen MySQL-Clients benutzt werden. Die maximal zulässige Größe beträgt 4 GB auf einem 32 -Bit-Maschine.

innodb_buffer_pool_size:
Gibt die Größe des Pufferpools an. Durch die Vergrößerung des Pufferpools, wird die Leistung gesteigert, indem die Menge an Festplattenzugriffen reduziert wird, weil die Abfragen auf die InnoDB-Tabellen zugreifen.

Folgen der Änderung:

Die Dauer von Abfragen wurde drastisch minimiert, z.B. benötigte vor der Änderung zwischen 35 und 40 Sekunden. Nach der manuellen Speicherzuweisung dauert die Abfrage nur noch zwischen 0,2 und 0,4 Sekunden, wenn man vom Localhost darauf zugreift. Über ein Netzwerk dauer die Abfrage dann rund 0,1 Sekunden länger.
Diese Beispielwerte wurden mit einer Abfrage gemacht, als die Tabelle eine Größe von ungefähr 1,6 Millionen Zeilen hatte!

5 Evaluation

5.1 Objektive Analyse der Priorisierung

Um die Effektivität der Priorisierung zu evaluieren, soll die Anwendung der Software hier an einem Beispiel veranschaulicht werden. Nachdem sich der Benutzer registriert und angemeldet hat, können über die Seite *Keywords* einige Schlüsselwörter hinzugefügt werden. Wie in Abbildung 5.1 zu sehen ist, entscheidet sich der Benutzer für die beiden

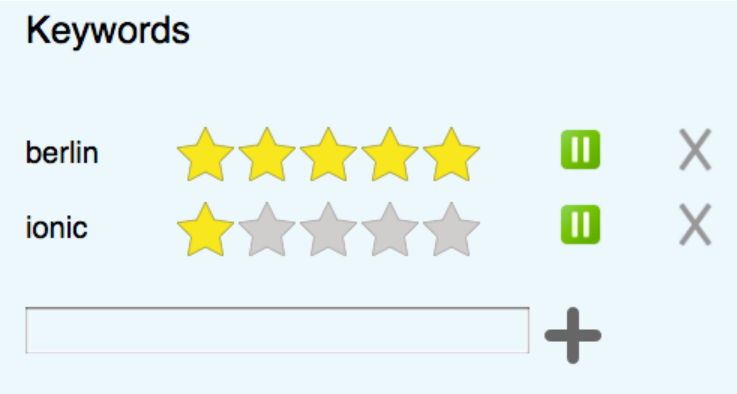

Abbildung 5.1: Konfiguration der *Keywords*

Begriffe „berlin“ und „ionic“ und vergibt eine individuelle Priorität von fünf Sternen für „berlin“ bzw. einem Stern für „ionic“. Tweets, welche dem ersten Begriff entsprechen sollten demanch eine höhere Gesamtpriorität erhalten.

Nachdem einige Tage vergangen sind, können nun die gefundenen Tweets analysiert werden. Es fällt auf, dass fast ausschließlich Tweets mit dem Keyword „berlin“ angezeigt werden. Auch der in Abbildung 5.2 dargestellte Tweet folgt dieser Tendenz. Dies soll nun im Folgenden analysiert werden.

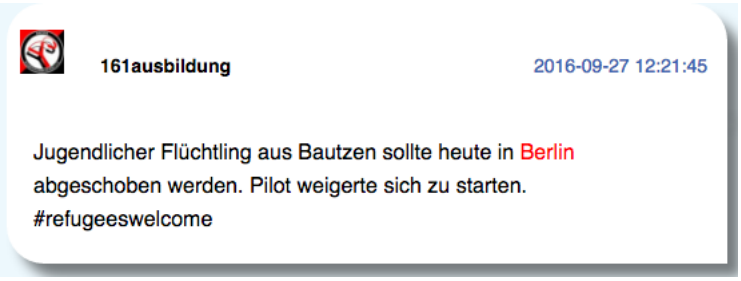

Abbildung 5.2: Dieser Tweet wird als erster angezeigt, hat demnach die höchste Priorität

Durch eine entsprechende Datenbankabfrage lässt sich herausfinden, dass der obige Tweet eine Basispriorität von 2,44 besitzt. Durch Hinzunahme der individuellen Einstufung (fünf Sterne) lässt sich diese auf 7,32 steigern. Zum Vergleich: Der am höchsten eingestufte Tweet mit dem Begriff „ionic“ (vgl. Abbildung 5.3) liegt mit einer Priorität von 1,42 deutlich darunter. Dieser Wert entspricht im Übrigen auch der Basispriorität, da die individuelle Einstufung durch den Faktor „1“ keine sichtbare Änderung hervorruft. Auffällig ist hier insbesondere der Unterschied bei den Basisprioritäten. Trotz einer

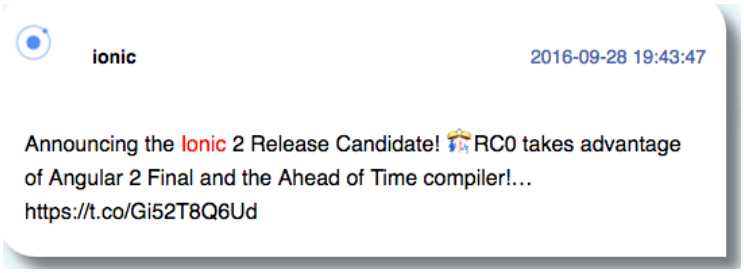

Abbildung 5.3: Der Tweet von „ionic“ wird vergleichsweise gering bewertet

deutlich höheren *Follower*-Anzahl des Autors, liegt der „ionic“ Tweet hinter dem „berlin“ Tweet. Dies ist auf die vergleichsweise geringe Gewichtung der *Follower* zurückzuführen und explizit so gewünscht, da die Tweets eher an der Anzahl von *Likes* und *Retweets* gemessen werden sollen. In dem konkreten Beispiel ist die Anzahl der *Retweets* (zum Zeitpunkt der Abfrage) in etwa gleich, die Anzahl der *Likes* des „berlin“ Tweets übertrifft seinem „Konkurrenten“ jedoch um das Doppelte.

Vertauscht man die individuelle Einstufung („berlin“ : ein Stern, „ionic“ : fünf Sterne), so erhöht sich die Priorität des „ionic“ Tweets auf 7,1. Er liegt nun vorne.

5.2 Subjektive Einschätzung der gefundenen Tweets

Ziel des Projektes ist die automatische Selektion von Tweets unter Berücksichtigung persönlicher Interessen. Zur Evaluation dessen wird eine Woche lang nach verschiedenen Schlüsselwörtern gesucht. Dabei sind unter anderem die Begriffe *Apple*, *BVB* und *Fender*. Bei dem Begriff *Fender*, welcher eigentlich Tweets über den gleichnamigen Instrumentenbauer finden sollte, ist aufgefallen, dass größtenteils Tweets gefunden wurden, bei denen eigentliche Keyword lediglich als Infix auftritt. Dieses Problem kann jedoch mithilfe der

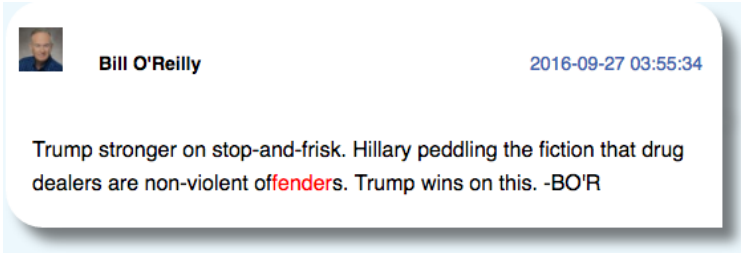

Abbildung 5.4: Das Keyword *Fender* tritt hier nur als Teilwort auf

Suchfunktion umgangen werden. Durch die erneute Filterung tauchen nun nur noch die Tweets auf, jene das entsprechende Keyword als eigeständiges Wort enthalten. Im konkreten Beispiel tauchen nun, neben den Tweets des Herstellers selbst, vermehrt Tweets mit

Bildern oder Tweets zu aktuelle Ereignissen in der Musikindustrie auf, die aber allesamt einen gewissen Informationsgehalt haben.

Eine weitere Funktion des Programms ist das Versenden von Benachrichtigungen bei besonders interessanten Tweets. Hierzu wurde ebenfalls eine Testreihe gestartet. Das Ergebnis wird am Beispiel des Keywords *BVB* verdeutlicht. In erster Linie will der Benutzer dabei über die neusten Tweets rund um den Fußballverein *Borussia Dortmund* informiert werden. Wie in Abbildung 5.5 zu sehen ist, wird der Benutzer kurzfristig über interessante

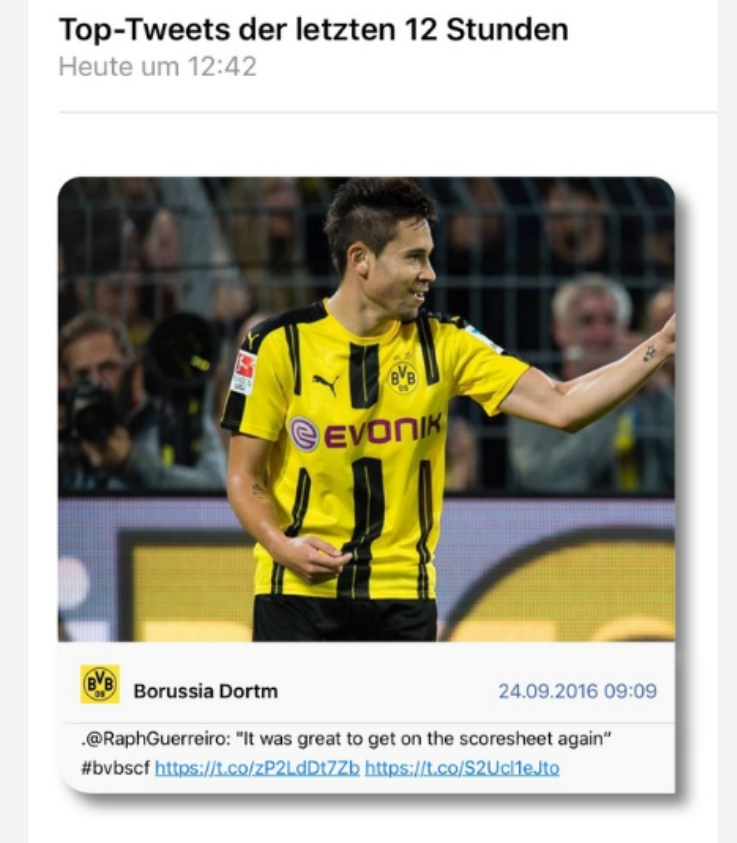

Abbildung 5.5: Nach dem Spiel wurde der Nutzer über interessante Tweets des Vereins benachrichtigt

Tweets benachrichtigt. Diese könne insgesamt als aktuell und informativ bewertet werden. Inhaltlich geht es meistens um den Fußballverein, selten ist auch ein Tweet dabei, bei dem das Keyword *BVB* etwa in URLs auftaucht. Die Zeitspanne für das Senden von Benachrichtigungen (zwölf Stunden) kann aus subjektiver Sicht als zu gering betrachtet werden, da nicht immer interessante Ereignisse stattfinden und dadurch eine gewisse Menge an irrelevanten Tweets zustande kommt.

Referenzen

Literatur

[1] Craig Walls. *Spring in action*. Shelter Island: Manning, 2011.

Internet

[2] Mohammed Jahangir. *What is meant by streaming API?* 2013. URL: https://www.quora.com/What-is-meant-by-streaming-API (besucht am 29. 09. 2016).

[3] Tutorialspoint (Hrsg.). *Spring - Bean Definition*. 2015. URL: https://www.tutorialspoint.com/spring/pdf/spring_bean_definition.htm (besucht am 29. 09. 2016).

[4] Tutorialspoint (Hrsg.). *Spring - IoC Containers*. 2015. URL: https://www.tutorialspoint.com/spring/spring_ioc_containers.htm (besucht am 29. 09. 2016).

[5] Java CodeBook (Hrsg.). *Web Application Security with Spring*. 2013. URL: http://www.javacodebook.com/2013/08/08/spring-book-chapter-15-web-application-security-with-spring/8/ (besucht am 29. 09. 2016).

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

REST	Representional State Transfer
TCP	Transmission Control Protocol
IoC	Inversion of Control
EJBs	Enterprise Java Beans
POJO	Plain Old Java Object
MVC	Model-View-Controller
JSP	Java Server Pages
DAO	Data Access Objekt
SQL	Structured Query Language
SMTP	Simple Mail Transfer Protocol
HTML	Hypertext Markup Language
DBMS	Datenbankmanagementsystem
PK	Primary Key
FK	Foreign Key
NN	Not Null

Kolophon

Dieses Dokument wurde mit der LaTeX-Vorlage für Abschlussarbeiten an der htw saar im Bereich Informatik/Mechatronik-Sensortechnik erstellt (Version 2.1). Die Vorlage wurde von Yves Hary und André Miede entwickelt (mit freundlicher Unterstützung von Thomas Kretschmer, Helmut G. Folz und Martina Lehser). Daten: (F)10.95 – (B)426.79135pt – (H)688.5567pt