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
Moritz Grill
Niko Kleer
Marek Kohn
Stefan Schlösser
Philipp Schäfer
Oliver Seibert

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

Saarbrücken, Tag. Monat Jahr

Inhaltsverzeichnis

1 Einleitung

1.1 Aufgabenstellung

1.2 TwitterStream API

1.2.1 Grundkonzept

Die *Twitter Streaming APIs* bieten eine öffentlich zugängliche Programmierschnittstelle für einen EchtzeitZugriff auf Daten der SocialMediaPlattform Twitter auf der Grundlage von HTTP.

Vergleich zu REST APIs: (Twitter und andere) Der hergebrachte Weg zu online verfügbaren Daten, sowohl für Twitter als auch bei anderen Diensten (Facebook, GoogleEarth), erfolgt über eine REST API (Representational State Transfer). Dabei stellt der Server einer RESTWebAnwendung, meist auf Anfrage eines Benutzers per GET- oder POSTRequest eine entsprechende Anfrage mit den Suchparametern in der URI beim Server des Anbieter der Daten, erhält die gewünschten Daten in der Response und gibt diese an den Benutzer weiter. Die Anfrage vollzieht sich dabei zustandslos. Für jede Anfrage wird eine neue TCPVerbindung zum Anbieter aufgebaut, die (wie jede HTTPVerbindung) nach der Übermittlung der Daten sofort wieder durch den AnbieterServer beendet wird.

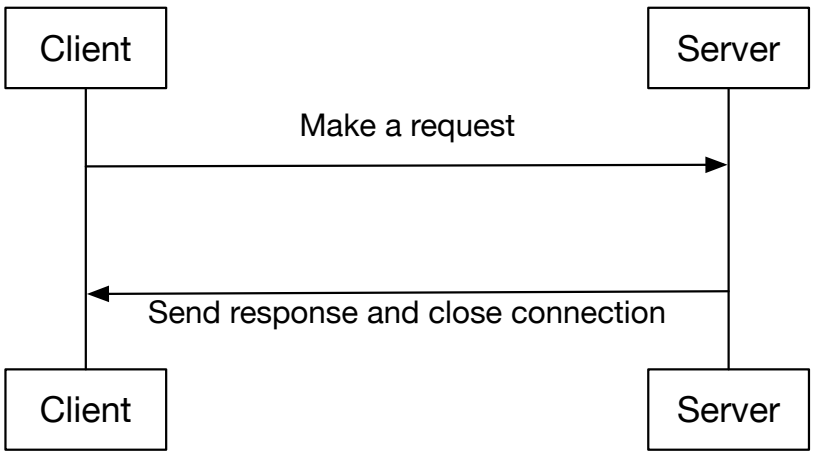

Abbildung 1.1: Grundprinzip einer REST-API

Das Prinzip einer Streaming API beruht dagegen gerade darauf, die Verbindung zum DatenServer über einen längeren Zeitraum aufrecht zu erhalten. Die Verbindung wird auch hier durch einen GET oder POSTRequest 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 ClientSeite, also beim Server der WebAnwendung können diese Informationen dann aus dem Socket der Verbindung gelesen werden.

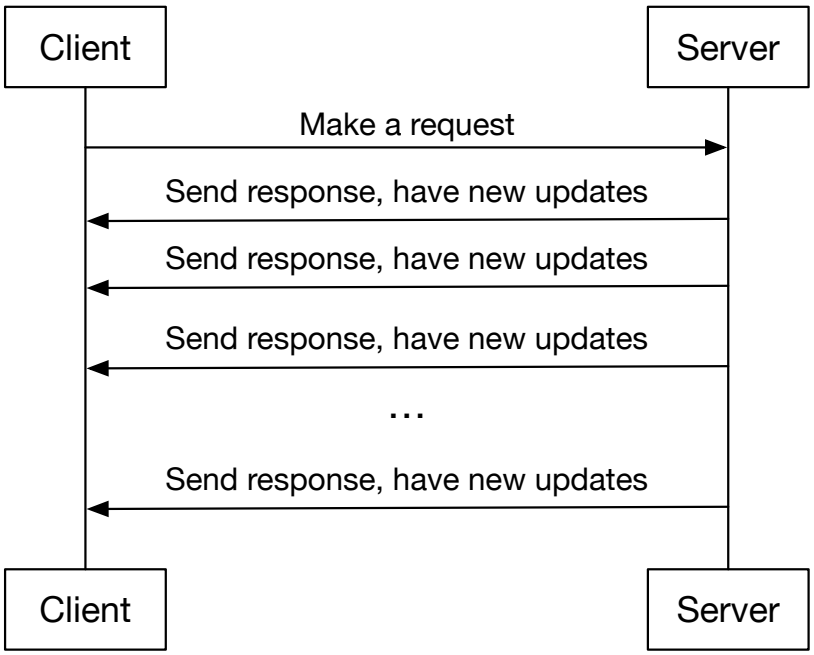

Abbildung 1.2: Grundprinzip einer Streaming-API

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 (evtl. bearbeitet) 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 Serversoftware im Falle einer REST APIAnwendung 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 ClientModul unabhängig vom Nutzer aufgebaut und verwaltet. 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 als Server zuständig ist.

Twitter bietet insgesamt vier Schnittstellen an. Im Zentrum des Projekts steht der Public Stream.[1] 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 Keyword entspricht, wird durch den Stream geschickt. 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.[2] Daneben werden außerdem noch User und Site Streams angeboten, die alle Tweets liefern, die 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.

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 Co. dokumentiert und kommentiert, wenn diese Plattformen nicht sogar als Katalysator wirken, der ein Ereignis erst weltweit bekannt macht. 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[4] oder zu können.[5] 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, die ... 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 RESTPrinzip 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, 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 auszunutzen, soll dem Benutzer außerdem die Möglichkeit gegeben werden, seine EmailAdresse zu hinterlassen, so dass er automatisch über neu eingetroffene Tweets unterrichtet wird.]

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.

Eclipse Mars/Neon Die Java-IDE unserer Wahl

Maven Ein Package/Buildmanager

GitHub Eine Online-Versionskontrollsystem

Trello Ein Taskboard zum Planen von Aufgaben

Slack Eine Online-Kommunikationsplatformm

3 Projektbeschreibung

3.1 Funktionsweise

Die Funktionsweise der Anwendung orientiert sich am bereits beschriebenen allgemeinen Aufbau einer StreamingAnwendung aus zwei Teilen, die unabhängig voneinander arbeiten und nur über die Datenbank miteinander in Verbindung stehen.

3.2 Datenmodell / Terminologie Twitter

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. Wenn es sich um einen Retweet handelt, enthält der Status außerdem auch den Status des ursprünglichen Tweets. Da Retweets jedoch meist nur der Verbreitung eines Tweets dienen, werden sie in der Anwendung jedoch mehr oder weniger ignoriert und lediglich für Aktualisierungen verwendet. Somit sind die zentralen Daten, die dem Stream entnommen und in der Datenbank gespeichert werden, Informationen zu einem Tweet und seinem Autor.

3.3 Mitgliederverwaltung

Auch wenn das Ziel des Projekt es 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.

3.4 Filter/Suchparameter (Keywords/Prioritäten)

Die zentralen Parameter, nach denen die Tweets durch den Stream erfasst werden und später dem Benutzer präsentiert werden, sind dessen Schlüsselwörter. Für jeden Nutzer werden diese Schlüsselwörter dabei mit einer Priorität (1–5) versehen, die angibt, wie wichtig dem Nutzer dieses Schlüsselwort ist. So lassen sich die verfügbaren Tweets für einen bestimmten Benutzer immer in einer eindeutigen Reihenfolge anzeigen, die durch weitere Filterung anhand eines Schlüsselworts verfeinert werden kann.

3.4.1 Priorisierung von Tweets

In diesem Abschnitt soll dargestellt werden, wie die Tweets bewertet bzw. priorisiert werden. Mithilfe eines selbst geschriebenen Algorithmus wird dies im Twitter-Monitor realisiert.

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 ausdenken

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 unter 1 liegt:

$$\frac{\frac{Likes}{2} + Retweets \cdot 2}{Follower} \cdot 100 \quad Priorisierung\ eines\ Tweets \tag{3.1}$$

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:

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

Tabelle 3.1: Staffelung von Tweets nach Alter

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.

$$\frac{\frac{Likes}{2} + Retweets}{\sqrt{Follower} + 100} \quad \textit{Priorisierung eines Tweets, zweiter Versuch} \tag{3.2}$$

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

$$\frac{\frac{Likes}{2} + Retweets}{\sqrt{Follower} + 100} \cdot 0.2 \cdot \kappa \quad \textit{Individuelle Priorisierung eines Tweets} \tag{3.3}$$

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.

3.5 Zeitplanung / Skalierung

Zu den größten Herausforderungen des Projekts gehört die Skalierung des Ausmaß 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. Da der Schwerpunkt des Interesses auf aktuellen Ereignissen liegt, werden zunächst nur Tweets gesammelt und aufbewahrt, die nicht bereits ein gewisses Alter erreicht haben. Das Limit wurde auf 48 Stunden festgelegt. Es sollten sich also keine älteren Tweets im System befinden.

4 Implementierung

4.1 Das Spring-Framework

4.1.1 Warum Spring?

In dem nun folgenden Abschnitt wird die Frage erläutert warum das Spring Web Framework verwendet wurde. Insbesondere welche Alternativen es gab und die jeweiligen Vor- und Nachteile sowie die Anforderungen, die an das Web Framework gestellt wurden.
Die Projektteilnehmer legten zusammen folgende Anforderungen fest. Zum einen sollte es ein Web- Framework 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
- Vaadin
- Java Server Faces
- Spark

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 quelloffenes Framework 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 Web Anwendungen 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 Web Entwicklung, welches ebenfalls speziell für die Entwicklung mit 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 Web-Entwicklung 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 Web Anwendungen. 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 JavaServerbasierten WebAnwendung und andererseits eine flexible DatenbankSchnittstelle. Dadurch ermöglicht es die Kapselung der eigentlichen Funktionalität im JavaCode, während die Datenbank und die Website unabhängig davon aufgebaut werden können.[6] Auch intern erleichtert Spring den Aufbau der zweiteiligen StreamingArchitektur (Wiring und DI zur Erstellung des Streams) und bietet daneben noch zahlreiche weitere hilfreiche Elemente (Autowiring, Scheduler, Login, Verschlüsselung).
Spring besteht aus einzelnen Komponenten, die (teilweise?) unabhängig voneinander je nach Bedarf verwendet werden können. Das Grundprinzip von Spring[7] besteht in der deklarativen (???) Art der Objekterstellung durch Wiring und Dependency Injection. Darunter versteht man die automatische Erstellung und Referenzierung von Objekten in einem gemeinsamen Application Context.[8] Explizite Konstruktoraufrufe (in einem Boiler PlateCode) sind nicht mehr notwendig. Da es sich bei allen Objekten in der Anwendung außerdem um Einzelstücke[9] selbsterstellter Klassen handelt, kann selbst die Deklaration an sich dem Autowiring überlassen werden. Alle Klassen, die als zu erstellende Beans/Komponenten[10] markiert sind, werden beim Anwendungsstart mit Refe-

renzen auf andere Beans/Komponenten aus dem Application Context, die anhand des benötigten Datentyps identifiziert werden, instantiiert. In der Anwendung wird diese grundlegende Architektur unter anderem auch verwendet, um zum Anwendungsstart ein Objekt zu erstellen, das für die Erstellung und Verwaltung des Streams zuständig ist. Eine SpringWebAnwendung verwendet außerdem zentral die mvcKomponente um Webpages zu erstellen und mit Daten aus der Anwendung zu versorgen. Dazu empfängt in einer laufenden Serveranwendung zunächst ein zentrales DispatcherServlet alle HTTP-Requests und verteilt diese dann an die passenden Controller. Dieser befragt die verschiedenen Services eventuell nach Daten und speichert diese in einem Model. Außerdem liefert der Controller ein JSP. Aus den Daten aus dem Model und der JSP wird schließlich die Webpage generiert und im Response versendet.

4.2 Architektur

Open for extension, closed for modification [1] - 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:

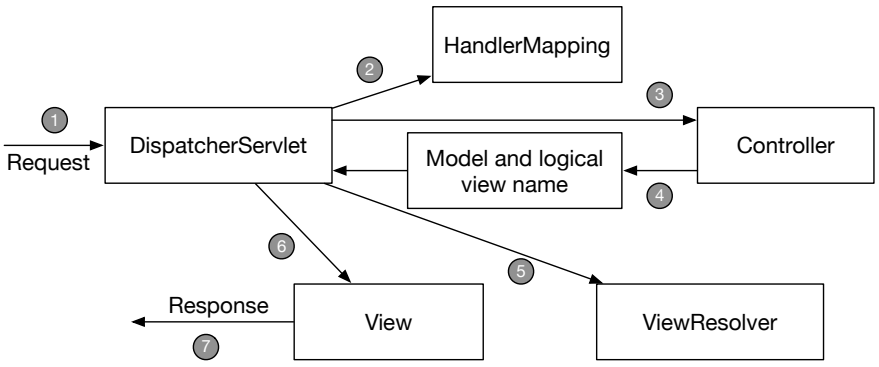

Abbildung 4.1: Komponenten in Spring MVC

[1] 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`. Der `View Name` kann nun von einem `View Resolver` mit einer konkrete Implementierung identifiziert werden. Dies kann beispielsweise in Form einer JSP geschehen. Der `View` rendert schließlich die Daten und liefert ein `Response Object` zurück.

TODO: Konzept von Java Beans erklären

4.3 Frontend

4.3.1 Programmierspezifische Mittel

Java Server Pages (JSP) erlauben es mittels eines Präprozessors eine HTML basierte Website zu erstellen, die das Hinzufügen von Java-Code inmitten des HTML-Codes erlaubt. Dazu bedient man sich bestimmter Tag-Bibliotheken.
Die Ausrichtung verschiedener Elemente auf einer Website sowie der Bestimmung farblicher Aspekte wird von CSS übernommen. Da CSS vom Browser des jeweiligen Clients interpretiert und umgesetzt wird, benötigt eine Änderung von CSS-Attributen keine Neukompilierung der JSP. DIese Eigenschaft ermöglicht die Neuausrichtung von Elementen bei Erfüllung einer Bedingung oder durch eine Javascript-Funktion.
Javascript ist eine höhere Skriptsprache die vom Browser des Clients interpretiert wird und ermöglicht zum einen eine funktions- sowie objektorientierte Programmierung. Sie kann aktiv auf HTML- Elemente zugreifen und sie verändern sowie neue HTML-Elemente dynamisch erstellen. Javascript hat auch die Eigenschaft CSS-Datein manipulieren zu können. Da auch Javascript keine Neukompilierung der JSP nach sich zieht, werden Veränderungen während der Laufzeit umgesetzt.
Mit dem Java-Framework Spring wird das Architekturmuster MVC umgesetzt. Das MVC definiert Controller, die als Bindeglied zwischen dem Model, welches hauptsächlich mit Java Umgesetzt wird und dem View gelten. Ein Controller kann den Datenaustausch zwischen dem Model und Browser des Clients steuern.

4.3.2 Konkreter Aufbau

Die Ziele die unser Team im Design verfolgen sind vor allem eine schnelle Orientierung zu ermöglichen, sowie eine daraus folgende unkomplizierte Anwendung für den Benutzer bereitzustellen. Heutzutage geht der Trend immer mehr zur One-Page-Website. Diese Art des Designs verbirgt Ihre Komplexität durch Links und Buttons innerhalb der durch Scroll erreichbaren Bereiche. Dies birgt jedoch die Gefahr, dass gesuchte Funktionen nicht gleich auf Anhieb gefunden werden können. Aus diesem Grund haben wir uns für ein Multi-Page-Layout entschieden, um die für den Benutzer relevanten Funktionen aufzuteilen und in folgenden Seiten widerzuspiegeln. Dies sind über ein Menü jederzeit erreichbar.

Home

Dem Benutzer soll es ermöglicht werden über eine Eingabemaske ein Konto anzulegen. Das Konto dient zur individuellen Zuweisung von Tweets und Keywords zu einem Benutzer. Die vom Benutzer geforderten Daten sollten nicht zu umfangreich sein, da sich eine schnelle Abwicklung des Prozesses positiv auf die Absprungrate auswirkt. Trotzdem sollten die Daten den Benutzer eindeutig identifizieren können. Zudem müssen Eingaben validiert werden, um Eingabefehlern vorzubeugen und Sicherheitsaspekte umsetzen zu können. Die Validierung sollte schon während der Laufzeit erfolgen, so dass dem Benutzer die Möglichkeit gegeben werden kann seine Eingaben unmittelbar zu korrigieren.
In folgendem Abschnitt wird darauf eingegangen wie wir diese Anforderungen umgesetzt haben.
Auf der Seite Home sind zwei Hauptbereiche dargestellt, die jeweils in zwei Unterbereiche untergliedert sind. Der erste Hauptbereich ist der `Header`. Er beinhaltet zum einen ein Logo und eine Login-Interface. Das Login-Interface beinhaltet ein Textfeld für den Benutzernamen, sowie ein besonderes Passworttextfeld, dass die Eingabe nicht als Klartext

wiedergibt. Zum anderen enthält die Eingabemaske einen Button, der Mittels einer Post-Message die Daten der beiden Textfelder an einen Controller des Servers weiterreicht. Natürlich muss hierbei die JSP neu geladen werden. Mit der Spring-Security- Bibliothek und einem CRF-Tokens ist es möglich diese Nachricht sicher abzufangen, an das Springframework weiterzureichen und eine Session für einen bestimmten Nutzer zu erstellen. Nachdem der Abgleich der Eingaben und der in der Datenbank gespeicherten Werte erfolgreich verlief, wird der Benutzer mittels eines an das Spring-Security-Bibliothek gebundenen Controllers ,auf die Seite *Tweets* weitergeleitet. Der zweite Hauptbereich enthält eine Animation die Mittels Javascript und CSS ermöglicht wurde. Dies geschieht indem mehrere, zeitlich versetzte Javascript-Funktionen den CSS-Code manipulieren, wobei die Animation selbst jedoch von CSS-Seite verwirklicht wird, da CSS auch Bewegungsabläufe darstellen kann. Unterhalb des Login-Interfaces befindet sich ein Hyperlink der die Darstellung der Animation unterbindet und eine neue Eingabemaske, die zuvor in der JSP deklariert wurde, sichtbar macht. Diese Eingabemaske dient dem Erstellen eines neuen Benutzerkontos. Jedes der Textfelder führt bei Eingabe eines Characters eine Javascript-Funktion aus, welche die Eingaben validiert und dynamisch eine entsprechende Ausgabe liefert. Auch bei dieser Eingabemaske wird eine POST-Message gesendet, die aber nicht von Spring-Security abgefangen, sondern mittels der JSTL-Tag-Library an einen `Controller` gebunden wird.

Tweets

Die durch ein Keyword gefilterten Tweets müssen übersichtlich dargestellt werden. Ein dargestellter Tweet sollte aus folgenden Teilen zusammengesetzt sein:

- Tweet-Text
- Tweet-Bild
- Tweet-Author
- Tweet-Zeit

Die Liste der dargestellten Tweets sollte sich während einer Session selbstständig aktualisieren.
Die Umsetzung dieser Kriterien wurde in auf der Seite Tweets umgesetzt.
Während die Seite Home hauptsächlich in der JSP selbst definiert wird, nutzt die Seite Tweets vor allem den Vorteil von Javascript, dynamisch HTML-Elemente zu erzeugen. Wird die Tweets-Seite geöffnet, erstellt ein bestimmter Controller eigenständig eine JSP, dessen Inhalt alle Tweets in Form von einem JSON-Objektes bereitstellt. Dieses Objekt wird von einer Javaskript-Funktion aufgegriffen. Die besagte Funktion erstellt für jeden Tweet im Objekt einen Tweet-Container, der alle Tweet-Attribute Formatiert darstellt. Zudem Verfügt die Seite Tweets über eine Sidebar mit Funktionen zur Suche von Schlagwörtern und Sortierung innerhalb der aktuell angezeigten Liste von Tweets. Diese Funktionen wurden ausschließlich mit Java-Skript realisiert, um häufigen Übertragungen zwischen Client und Server vorzubeugen und die Performance zu verbessern.

Keywords

Es soll eine Schnittstelle bereitgestellt werden, mit dem man Keywords erzeugen, bearbeiten und löschen kann. Zudem muss eine einfache Vergabe von Prioritäten möglich gemacht werden.
Die Seite Keywords dient zur Erstellung und Darstellung von Keywords und negativen

Keywords, die das Einschließen bzw. Ausschließen von Tweets ermöglicht. Wie auch bei der Seite Tweets sorgt ein Controller zur Übergabe eines JSON-Objektes. Diesmal werden keine Tweets sondern Keywords übergeben. Die durch Javascript erzeugten Keyworter bestehen aus drei Elementen. Dem Keyowrd selbst , fünf Sternen zur Symbolisierung der Priorität und einem Symbol zum Löschen des Kewords. Die Sterne wurden mit einem Hover-Effekt belegt, der die Anzahl der Prioritätspunkte wiedergibt. Bei einem Klick auf einen Stern wird mit einem Javascript-Framework namens AJAX eine Nachricht an den Server gesendet ohne dabei die Seite neu zu Laden. Diese Nachricht wird von einem Controller empfangen und speichert die Priorität eines Keywords in der Datenbank. Um ein neues Keyword zu erstellen wurde unter der Liste von bereits existierenden Keywords ein Textfeld mit einem Button implementiert. Auch hier gilt das AJAX-Prinzip, um die Seite nicht neu laden zu müssen. Negative Keywords werden nach dem selben Muster erstellt und angezeigt, verfügen jedoch nicht über eine Prioritätsvergabe.

Einstellungen

Es muss die Möglichkeit gegeben sein, sein Passwort zu ändern und gegebenenfalls eine Email-Benachrichtigung aktivieren oder deaktivieren zu können.

Logout

Desweiteren sollte es dem Benutzer möglich sein sich auszuloggen, um die Sicherheit von Mehrbenutzersystemen zu gewährleisten. Außerdem muss die Möglichkeit gegeben sein, sein Passwort zu ändern und gegebenenfalls eine Email-Benachrichtigung aktivieren oder deaktivieren zu können.

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 [2]. 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

Controller

In einer Spring-Web-Anwendung 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 jedoch in der Regel auch den Benutzernamen der aktuellen Session [3] Dieser kann von einem jederzeit verfügbaren Principal-Objekt angefordert werden [4].

Services

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.

[2] Data Access Objekt

[3] vgl. Abschnitt 4.6.1: 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. Neben den normalen Twitter Funktionen bietet die Twitter4j die Möglichkeit, Anfragen an Twitter zu stellen. Damit kann man Informationen zu Tweets (Name des Autors, Anzahl der Likes/Retweets, etc.) oder Autoren erhalten.

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 [5]. 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 Array von Schlüsselwörtern übergeben. Die Schlüsselwörter lädt der *StreamService* direkt mit einer entsprechenden Methode des *KeywordDao*. Von nun an empfängt der Listener alle Tweets, die den Keywords entsprechen. [6]

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, sollte die Keyword-Liste des Streams möglichst zeitnah erweitert werden. Das Gleiche gilt bei einem bestehendes 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,[7] muss der

[5] Oauth weiter ausführen oder auf Kap 1 verweisen (Twitter API Beschreibung)

[6] Vgl. Beschreibung der Funktionalität von Keywords in Kapitel 3 [muss noch geschrieben werden, aber da gehört es hin]

[7] Die naheliegende Lösung

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. Die Lösung dieses Dilemmas liegt darin, den Neustart nur dann durchzuführen, wenn auch wirklich neue Keywords vorliegen. In einem ersten Ansatz wurde eine expliziter Methodenaufruf durch die entsprechenden Methoden des UserService, die für Änderungen an den Keywords verantwortlich sind, zu machen. Diese Lösung ist jedoch nicht praktikabel, da bereits das mehrfache Hinzufügen von Keywords durch einen einzelnen Benutzer auch mehrfache Neustarts bedeuten würden. Die Verknüpfung des Neustarts mit der Erstellung und dem Löschen von Keywords bedeutet 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.[8] 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 nicht 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.

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 TweetListeners 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.

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

Eventuelle Probleme mit der Synchronisierung: 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 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. 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 bedeuten. Um diese zu reduzieren, werden Tweets und Autoren zunächst in einer HashMap zwischengespeichert und in einem Batch-Update gespeichert. Besonders aktive Autoren mit vielen Tweets in kurzer Zeit und häufige Retweets eines bestimmten Tweets werden dabei auch nur einmal übertragen. Dies bedeutet eine kleine Performance-Verbesserung.

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.[9] 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.

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

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 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 [10] 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 Modell-Klassen User, Keyword, Author und Tweet. Die DAOs der ersten drei Datentypen bieten keine größeren Besonderheiten. 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.[11] Die Abfrage nach vorhandenen Benutzern in der Datenbank beim Login wird automatisch von Spring-Security durchgeführt.[12] 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.

Keyword Das KeywordDao liefert einerseits dem StreamService eine duplikatfreie Liste aller Schlüsselwörter, die in der Datenbank gespeichert sind.[13] 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.

[10] Ein String mit einem SQL-Statement mit eventuellen Platzhaltern.
[11] Auf dem aktuellen Stand sind Email-Benachrichtigung und Email-Ändern noch nicht implementiert. Würde aber auch mit insert on duplicate key implementiert werden.
[12] Siehe entsprechenden Abschnitt.
[13] Ein TwitterStream der Twitter4J-API erfordert ein String-Array.

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.[14]

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 TweetIds 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.[15] 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 TweetId als Schlüsselwert in einer HashMap hinterlegt und die Schlüsselwörter weiterer Vorkommen in die Keyword-Liste 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 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.[16]

[14] Siehe entsprechenden Abschnitt.

[15] Es muss noch geprüft werden, ob dies nicht doch im SQL-Statement erledigt werden kann.

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

4.5 Datenbank

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 versendetwerden. Hilfreiche Funktionen zur Berechnung der Tweet-Priorität werden auch von der Datenbank angeboten.

4.5.1 Stored-Procedures und Stored-Functions

Tabelle 4.1: 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	tweet_text: text, author_name: varchar (15), author_picture: varchar(100), tweet_datum: datetime, tweet_image: varchar(100)	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.

Tabelle 4.2: 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

4.5.3 Trigger

Tabelle 4.3: Trigger

Name	Beschreibung	Typ
`tweets_after_insert`	Triggert 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

Tabelle 4.4: tweets: Speicherung der Tweet-Informationen aus Twitter.

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

Tabelle 4.5: 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`	ja		ja
`name`	Benutzername von Twitter, wird mit „@“ angesprochen	`varchar(15)`			ja
`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)`			

TODO: weitere Entitäten ergänzen. Wirklich an dieser Stelle ? Besser als Anhang ?

4.6 Sicherheitsaspekte

Jede Anwendung muss eine sichere Umgebung zur Verfügung stellen. Vor allem Webanwendungen bedürfen oft besonderer Sicherheit, da schlechtere Javascript- 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.

4.6.1 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 4.6: Spring Security Module

Module	**Description**
ACL	Provides support for domain object security through access control lists (ACLs)
Aspects	A small module providing support for AspectJ-based aspects instead of standard Spring AOP when using Spring Security annotations.
CAS Client Support	for single sign-on authentication using Jasig's Central Authentication Service (CAS).
Configuration	Contains support for configuring Spring Security with XML and Java. (Java configuration support introduced in Spring Security 3.2.)
Core	Provides the essential Spring Security library.
Cryptography	Provides support for encryption and password encoding.
LDAP	Provides support for LDAP-based authentication.
OpenID	Contains support for centralized authentication with OpenID.
Remoting	Provides integration with Spring Remoting.
Tag Library	Spring Security's JSP tag library.
Web	Provides Spring Security's filter-based web security support.

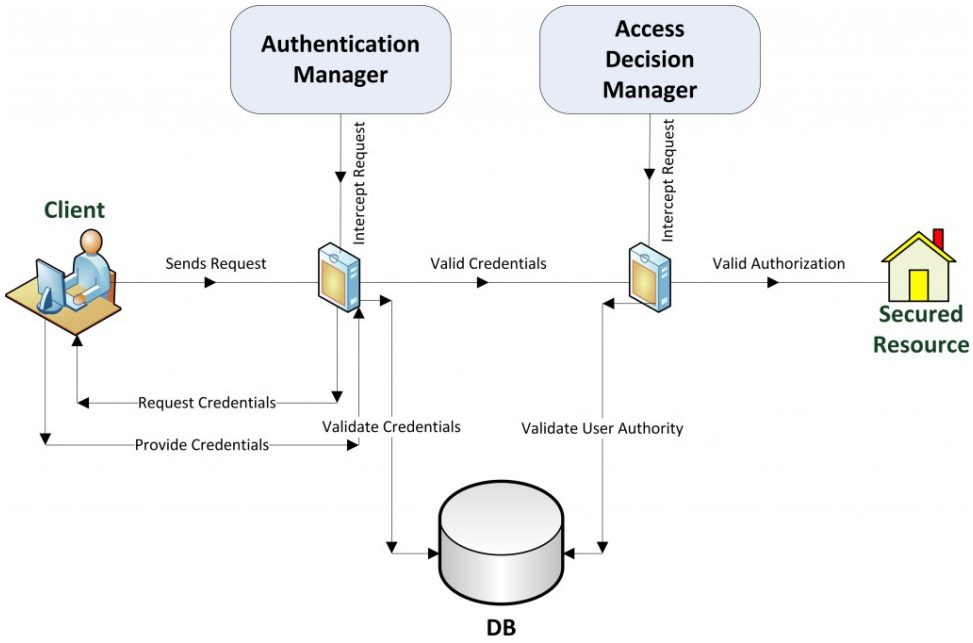

Abbildung 4.2: Authentifizierung und Validierung der Privilegien eines Nutzers

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.

4.6.2 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. Das folgende Diagramm 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. Im obigen Diagramm wird der folgende Prozess durch den Access Decision Manager dargestellt. Dieser ist in unserer konkreten Anwendung nicht implementiert, jedoch findet der Authorisierungsprozess hier 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 4.7: Die verschiedenen von Spring zur Verfügung gestellten Privilegien

Expression	Description
`hasRole([role])`	Returns `true` if the current principal has the specified role
`hasAnyRole([role1,role2])`	Returns `true` if the current principal has any of the supplied roles (given as a comma-separated list of strings)
`principal`	Allows direct access to the principal object representing the current user
`authentication`	Allows direct access to the current `Authentication` object obtained from the `SecurityContext`
`permitAll`	Always evaluates to `true`
`denyAll`	Always evaluates to `false`
`isAnonymous()`	Returns `true` if the current principal is an anonymous user
`isRememberMe()`	Returns `true` if the current principal is a remember-me user
`isAuthenticated()`	Returns `true` if the user is not anonymous
`isFullyAuthenticated()`	Returns `true` if the user is not an anonymous or a remember-me user

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 Authentication-manager 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 Authorität in der Datenbank hinterlegt werden.

4.6.3 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.

4.7 Deployment

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

MVC Model View Controller

JSP Java Server Pages

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