Technische Universität Braunschweig

ISF

News Miner

Gruppe 1

Software-Entwicklungspraktikum (SEP)
Sommersemester 2013

Systemspezifikation

Auftraggeber
Technische Universität Braunschweig
Institut für Informationssysteme
Prof. Dr. Wolf-Tilo Balke
Mühlenpfordstraße 23
38106 Braunschweig

Betreuer: Philipp Wille

Auftragnehmer:

Name	E-Mail-Adresse
Maria Joanna Born	maria.born@tu-braunschweig.de
Arne Brüsch	a.bruesch@tu-braunschweig.de
Jana Sarah Riquel	j.riquel@tu-braunschweig.de
Jennifer Sieg	jennifer.sieg@tu-braunschweig.de
Viviane Werner	v.werner@tu-braunschweig.de

Braunschweig, 26. Juni 2013

Versionsübersicht

Version	Datum	Autor	Status	Kommentar
0.1	15.05.2013	Maria Born, Arne Brüsch, Jana Riquel, Jennifer Sieg, Viviane Werner	abgenommen	überarbeiteter Grobentwurf
0.2	16.05.2013	Jennifer Sieg	abgenommen	Kapitel 4.1 bearbeitet
0.3	19.05.2013	Maria Born	abgenommen	Kapitel 4.4 bearbeitet
0.4	20.05.2013	Viviane Werner	abgenommen	Kapitel 4.3 bearbeitet
0.5	21.05.2013	Arne Brüsch	abgenommen	Kapitel 4.5 bearbeitet
0.6	21.05.2013	Jana Riquel	abgenommen	Kapitel 4.2 bearbeitet
0.7	21.05.2013	Maria Born, Jana Riquel	abgenommen	Datenmodell bearbeitet
0.8	21.05.2013	Maria Born, Arne Brüsch, Jana Riquel, Jennifer Sieg, Viviane Werner	abgenommen	Fehler überarbeitet
0.9	21.05.2013	Maria Born, Arne Brüsch, Jana Riquel, Jennifer Sieg, Viviane Werner	abgenommen	Abgabe an Betreuer
1.0	26.05.2013	Maria Born, Arne Brüsch, Jana Riquel, Jennifer Sieg, Viviane Werner	abgeschlossen	fertig gestellt

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

News Miner wird als webbasierte Client-Server Anwendung realisiert, d.h. der Nutzer kann per Webbrowser auf das System zugreifen. Er hat dann die Möglichkeit, RSS-Feeds zu aktuellen Trends zu lesen. Der Server ist dabei für die Ausführung des Programms verantwortlich.

Außerdem verwenden wir eine 3-Schichten-Architektur auf der Serverseite. Die Benutzerschnittstelle ist dabei die grafische Oberfläche. Hierüber kann der Nutzer mit dem System kommunizieren, wenn er zum Beispiel eigene Nachrichtenquellen angibt. Zudem werden ihm hier die Ergebnisse, die der Kern von *News Miner* ausgibt, präsentiert.

Dieser Kern wird in der Systemschicht realisiert und teilt sich in verschiedene Komponenten auf. Ziel der Systemschicht ist es, die vom Nutzer gewählten Nachrichten nach ihrer Aktualität zu beurteilen. Grundlage dieser Bewertung ist die Trendextraktion aus Twitterdaten. So werden in der Systemschicht die Nachrichtenartikel ausgewählt, die sich mit aktuellen Trends befassen. Dazu werden außerdem RSS-Feeds und Tweets gesammelt und gespeichert, da diese die Grundlage bilden.

Auch die Persistenzschicht besteht im Wesentlichen aus zwei Komponenten. Zum einen werden die Tweets und Feeds auf dem Filesystem abgelegt. Hier werden sie durch Lucene indiziert, um eine performante Volltextsuche zu gewährleisten. Zum Anderen werden die Nutzerdaten in einer Derby-Datenbank abgelegt, welche vor allem den Datenschutz gewährleistet, da Derby die Verschlüsselung der Kommunikation mit der Datenbank und der Datenbank Files selbst unterstützt. Die Persistenzschicht sichert also sowohl den Zugriff auf die von Lucene indexierten Daten, als auch den Datenbankzugriff.

Die Applikation wird in Java, Scala und HTML/CSS programmiert. Des Weiteren verwenden wir Play als Webframework sowie zur Nutzerverwaltung.

Im folgenden Statechart in Abbildung 1.1 werden die vom Nutzer sichtbaren Seiten als Zustände modelliert. Daraus ist ersichtlich, wie sich der Nutzer innerhalb der Benutzeroberfläche von einer Seite zu einer anderen bewegen kann. Hierbei werden alle Funktionen mit ihrem deutschen Namen aus dem Pflichtenheft und alle Schaltflächen mit ihrem englischen Namen, also so wie sie dem Nutzer angezeigt werden, bezeichnet.

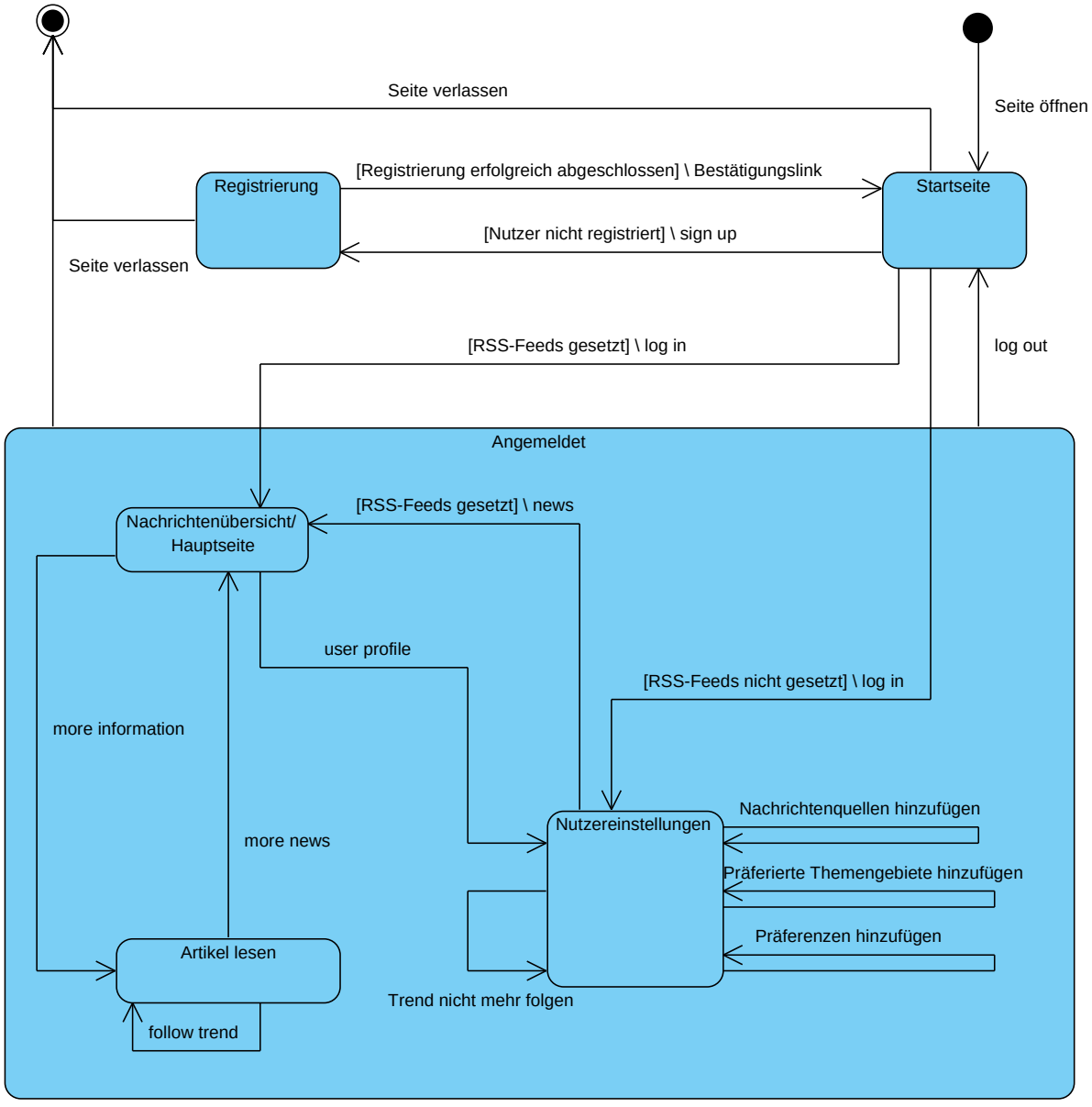

Abbildung 1.1: Übersicht über *News Miner*

1.1 Projektdetails

Um wichtige Aspekte der Anwendung *News Miner* genauer zu erläutern, werden in diesem Kapitel verschiedene Funktionen der Anwendung aufgeführt und beschrieben. In den folgenden Unterkapiteln sind die Funktionen des Registrierens, des Anmeldevorgangs und des Hinzufügens einer Nachrichtenquelle genauer erklärt.

1.1.1 Registrieren und Anmelden

Das folgende Aktivitätsdiagramm in der Abbildung 1.2 beschreibt den Workflow bei der Registrierung und Anmeldung eines Nutzers. Will sich ein Nutzer registrieren, so muss er zunächst auf der Startseite den “create an account”-Button anklicken. Dadurch wird ihm vom System die Seite angezeigt, auf der er seine Daten in Textfelder eintragen kann. Anschließend muss er diese durch einen Klick auf “sign up” bestätigen. Dies bewirkt, dass das System seine Daten in der Datenbank hinterlegt und eine Bestätigungsmail versendet. In dieser ist ein Bestätigungslink enthalten. Sobald der Nutzer diesem gefolgt ist, ist seine Registrierung abgeschlossen. Nun kann er sich anmelden. Dabei wird zunächst überprüft, ob er bereits Nachrichtenquellen angegeben hat. Wenn er sich das erste Mal angemeldet hat, ist dies natürlich nicht der Fall. Dann wird ihm vom System zunächst das Nutzerprofil angezeigt, verbunden mit einer Aufforderung, mindestens eine Nachrichtenquelle anzugeben. Hat er dies getan, so kann er durch einen Klick auf “News” in der Navigationsleiste auf die Nachrichtenübersichtsseite gelangen. Hier werden ihm die Nachrichten seiner Nachrichtenquellen zu den aktuellen Trends angezeigt.

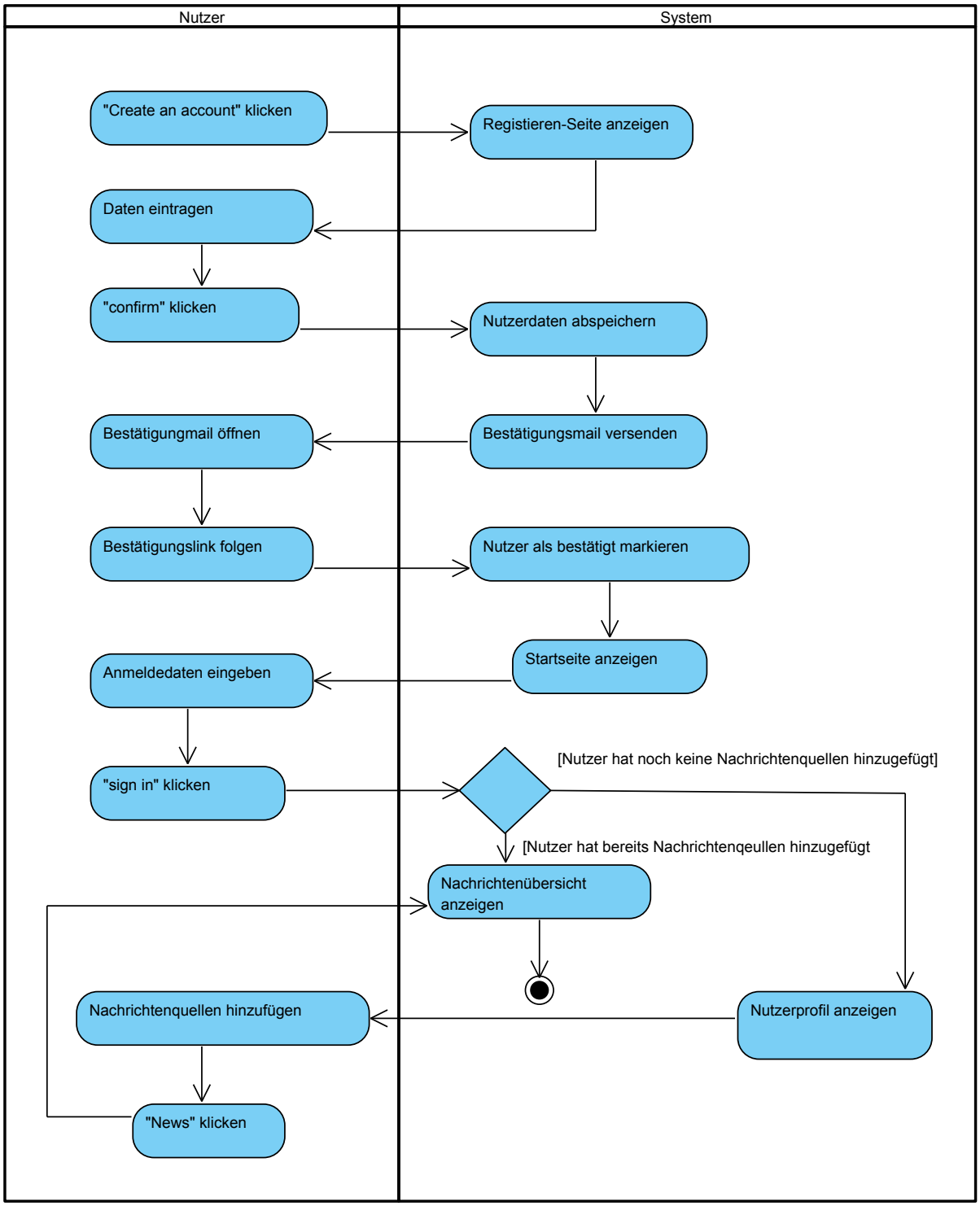

Abbildung 1.2: Übersicht über das Registrieren und Anmelden

1.1.2 Nachrichtenquellen hinzufügen

Das Aktivitätsdiagramm in Abbildung 1.3 wird der Workflow beim Hinzufügen einer Nachrichtenquelle beschrieben. Hierbei wird vorausgesetzt, dass sich der Nutzer bereits registriert und angemeldet hat. Will der Nutzer Nachrichtenquellen hinzufügen, muss er zunächst auf das Nutzerprofil gehen. Das Aktivitätsdiagramm nimmt an, dass ihm diese Seite bereits angezeigt wird. Hier sind unter der Überschrift “RSS-Feeds” mehrere Textfelder zu sehen. In eines dieser Felder muss der Nutzer die URL seines gewünschten RSS-Feeds eintragen und anschließend den “confirm”-Button klicken. Hat er dies getan, wird der Feed vom System überprüft. Findet das System unter der angegebenen URL keinen RSS-Feed, so ist dies auf einen Eingabefehler zurückzuführen. Deshalb wird dem Nutzer eine Fehlermeldung ausgegeben und er muss die Eingabe erneut bearbeiten. Findet das System an der angegebenen URL einen RSS-Feed, so extrahiert es den Titel. Danach speichert er die URL und den Titel des RSS-Feeds ab und ordnet sie dem Nutzer zu. Nun wird dem Nutzer der Titel anstatt des Textfeldes angezeigt. Zudem befindet sich nun anstelle des “confirm”-Buttons ein “change title”-Button. Sollte der Nutzer mit dem Titel nicht zufrieden sein, kann er diesen Button anklicken und den Titel ändern. Dieser wird vom System ebenfalls abgespeichert. Des Weiteren hat der Nutzer die Möglichkeit, weitere Nachrichtenquellen hinzuzufügen. Falls dies sein Wunsch ist, so wird der gesamte Workflow erneut durchlaufen, ansonsten ist der Workflow beendet und der Nutzer kann andere Funktionen von *News Miner* nutzen.

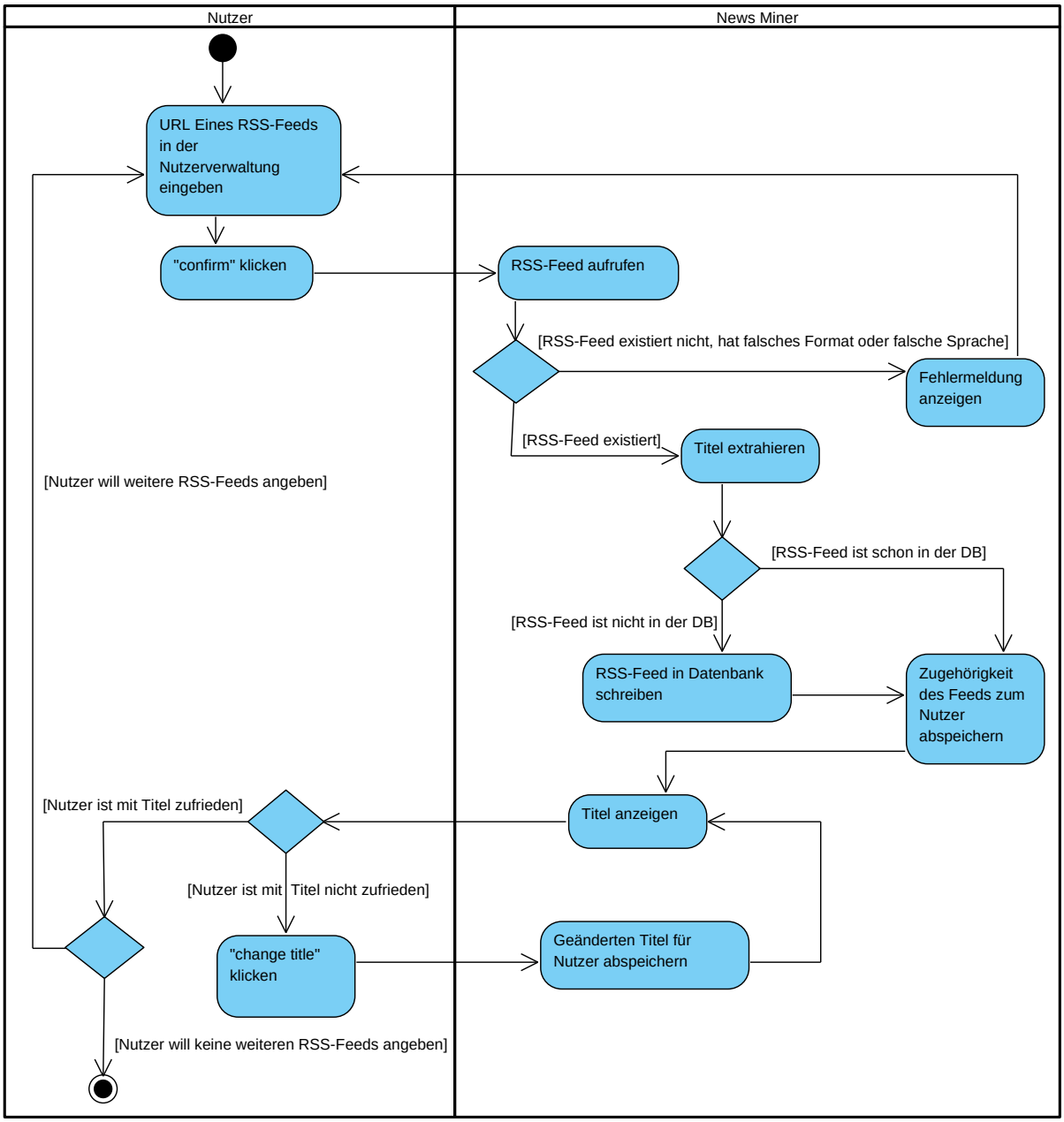

Abbildung 1.3: Übersicht über das Hinzufügen einer Nachrichtenquelle

2 Analyse der Produktfunktionen

Im folgenden Kapitel werden die Produktfunktionen sowie ggf. die nicht-funktionalen Anforderungen aus dem Pflichtenheft mittels Sequenzdiagrammen modelliert. Dadurch sollen die Interaktionen der jeweiligen Funktionen mit den verschiedenen Komponenten von *News Miner* verdeutlicht werden. Zusätzlich werden die Vorgänge in den Abbildungen näher erläutert.
Dies stellt insgesamt die Basis dar, um später eine geeignete Architektur festlegen zu können.

2.1 Analyse von Funktionalität /F100/: Registrieren

Die Funktion "Registrieren" ist essentiell, um die Webanwendung *News Miner* verwenden zu können, muss dafür aber auch nur einmal pro Benutzer durchgeführt werden. Denn mit dieser Funktion erstellt der Benutzer ein Nutzerkonto. Dieses wiederum wird benötigt, um die individuellen Angaben und die persönlichen Daten zu speichern und dem Nutzer zu einem späteren Zeitpunkt jederzeit wieder zuzuordnen, worauf letztlich weitere Funktionen aufbauen. Im folgenden Sequenzdiagramm in Abbildung 2.1 werden nun die Komponenten dargestellt, die an diesem Prozess beteiligt sind.

Zunächst verdeutlicht die Abbildung, wie der Nutzer seine persönlichen Daten, also eine E-mail-Adresse, ein Passwort und einen Nutzernamen, in der *NewsMinerGUI* angibt. Diese wiederum gibt sie an das *System* weiter, von wo aus sie dann an den *NewsMinerStorage* übergeben und dort letztlich abgespeichert werden.
Nachdem dieser Prozess erfolgreich war, wird eine Bestätigungsmail von der *NewsMinerGUI* an die angegebene E-mail-Adresse versendet. Durch Anklicken des Links in der versandten E-mail seitens des Nutzers, wird dieser von der *NewsMinerGUI* auf die Startseite von *News Miner* weitergeleitet. Damit ist die Registrierung erfolgreich abgeschlossen.

2.2 Analyse von Funktionalität /F200/: Anmelden

In dem Sequenzdiagramm in Abbildung 2.2 wird die Funktionalität "Anmelden" näher dargestellt. Diese Funktion wird benötigt, damit der Nutzer auf sein vorher erstelltes Nutzerkonto

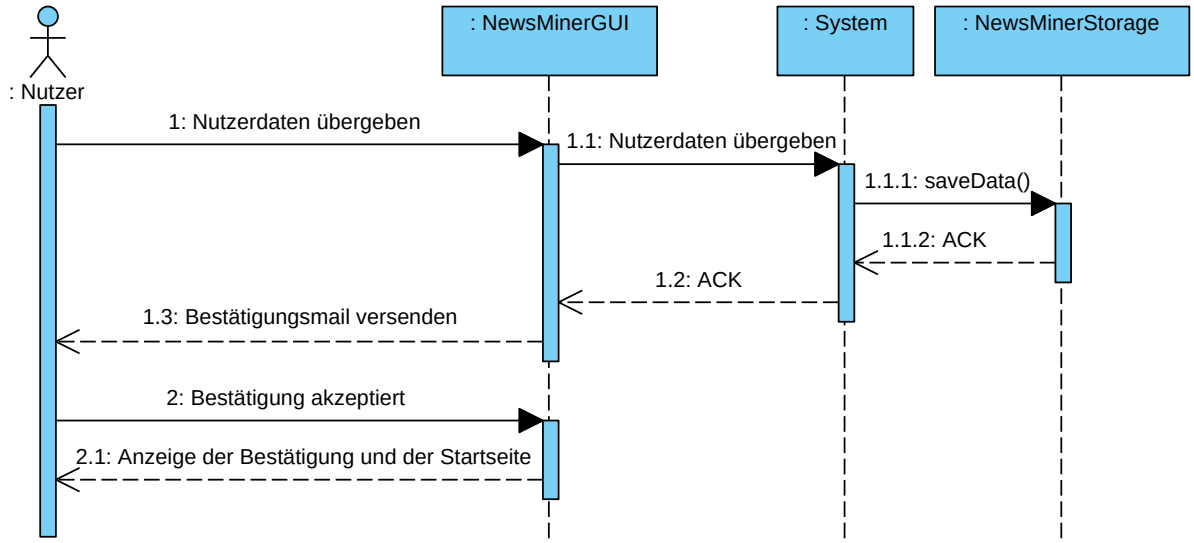

Abbildung 2.1: /F100/ Registrieren

zugreifen kann, um die Funktionen von *News Miner* verwenden zu können.

In der Abbildung gibt der Benutzer als Erstes die benötigten Daten, also die E-mail-Adresse und das Passwort, in der *NewsMinerGUI* an. Diese wiederum werden über das *System* an den *NewsMinerStorage* übergeben, wo die eingegebenen Angaben mit den dort abgespeicherten Daten abgeglichen werden. Wenn dies erfolgreich war, leitet der *NewsMinerStorage* eine positive Rückmeldung an das System weiter, welche dann an die *NewsMinerGUI* weitergegeben wird. Anschließend wird dem Benutzer die Hauptseite von *News Miner* angezeigt.

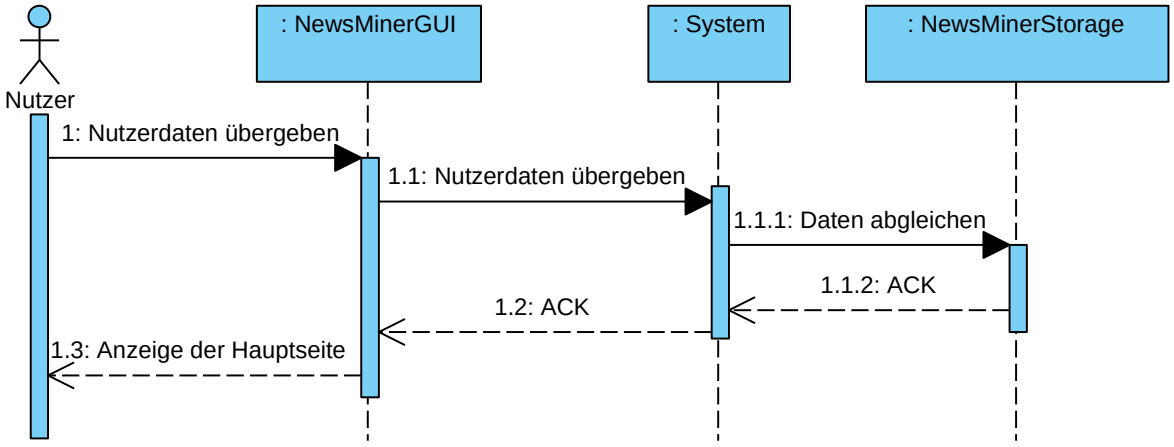

Abbildung 2.2: /F200/ Anmelden

2.3 Analyse von Funktionalität /F300/: Abmelden

Im Sequenzdiagramm in Abbildung 2.3 wird die Funktionalität "Abmelden" genauer dargestellt. Diese Funktion wird benötigt, damit sich der Nutzer aus seinem Nutzerkonto wieder abmelden kann, wenn er die Seite verlassen möchte.

In der Abbildung wird dargestellt, dass der Benutzer auf den "Logout"-Button in der *NewsMinerGUI* klickt, welche anschließend dem Nutzer eine Bestätigung anzeigt, dass er sich erfolgreich abgemeldet hat.

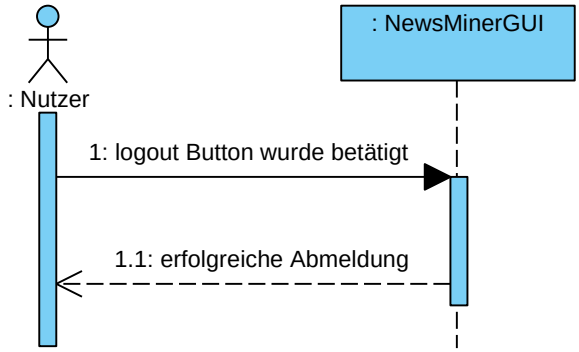

Abbildung 2.3: /F300/ Abmelden

2.4 Analyse von Funktionalität /F400/: Themengebiete auswählen

Die Funktion "Themengebiete auswählen" ist ein Zusatz, um dem Nutzer eine individuelle, seinen persönlichen Interessen entsprechende Auswahl an Nachrichtenartikeln zu bieten. Dies wird in Abbildung 2.4 in dem Sequenzdiagramm verdeutlicht.

Als Erstes wählt der Nutzer ein Thema über die *NewsMinerGUI* aus. Dies wird über das *System* an den *NewsMinerStorage* übergeben und dort in der Datenbank abgespeichert, sodass die Daten dem Nutzer jederzeit zugeordnet werden können. Anschließend fordert das *System* Nachrichtenartikel zum jeweiligen ausgewählten Thema aus dem *NewsMinerStorage* an. Nun übergibt das *System* die ausgewählten Nachrichtenartikel an die *NewsMinerGUI*. Diese wiederum ist dafür zuständig, die Nachrichten dem Nutzer anzuzeigen.

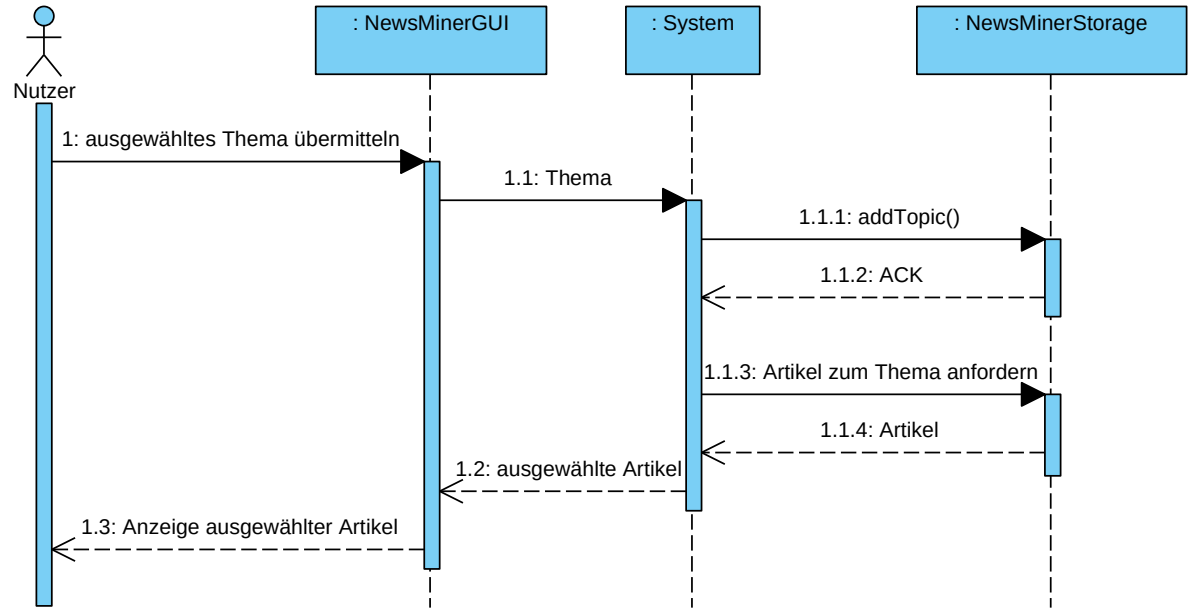

Abbildung 2.4: /F400/ Themengebiete auswählen

2.5 Analyse von Funktionalität /F500/: Präferenzen angeben

"Präferenzen angeben" ist eine Funktionalität, die die Hauptaufgabe von *News Miner* erweitert, um dem Nutzer mehr Vielfalt zu bieten. Es steht ihm vollkommen frei, diese zu verwenden. In dem folgenden Sequenzdiagramm in Abbildung 2.5 wird die Funktion bildlich dargestellt.

Zunächst ist in der Abbildung dargestellt, wie der Nutzer in der *NewsMinerGUI* eine persönliche Präferenz angibt. Diese wird dann über das *System* im *NewsMinerStorage* abgespeichert, damit sie dem Nutzer jederzeit wieder zugeordnet werden kann. Danach fordert dann das *System* vom *NewsMinerStorage* die abgespeicherten Nachrichtenartikel, die einen inhaltlichen Bezug zu der angegebenen Präferenz aufweisen, an. Die ausgewählten Nachrichten werden anschließend wieder von dem *System* an die *NewsMinerGUI* übergeben und dem Nutzer angezeigt.

2.6 Analyse von Funktionalität /F600/: Trend folgen

"Trend folgen" ist eine Funktion, die *News Miner* erweitert. Sie soll dem Nutzer die Möglichkeit geben, favorisierten Trends zu folgen, um weiterhin darüber informiert zu werden. Im Sequenzdiagramm in Abbildung 2.6 wird diese Funktion bildlich dargestellt.

Um diese Funktion nutzen zu können, muss der Benutzer als erstes auf den "Trend folgen"-Button in der *NewsMinerGUI* klicken. Die Informationen über den ausgewählten Nachrichtenartikel werden anschließend an das *System* weitergegeben und im *NewsMinerStorage* abgespeichert. Im

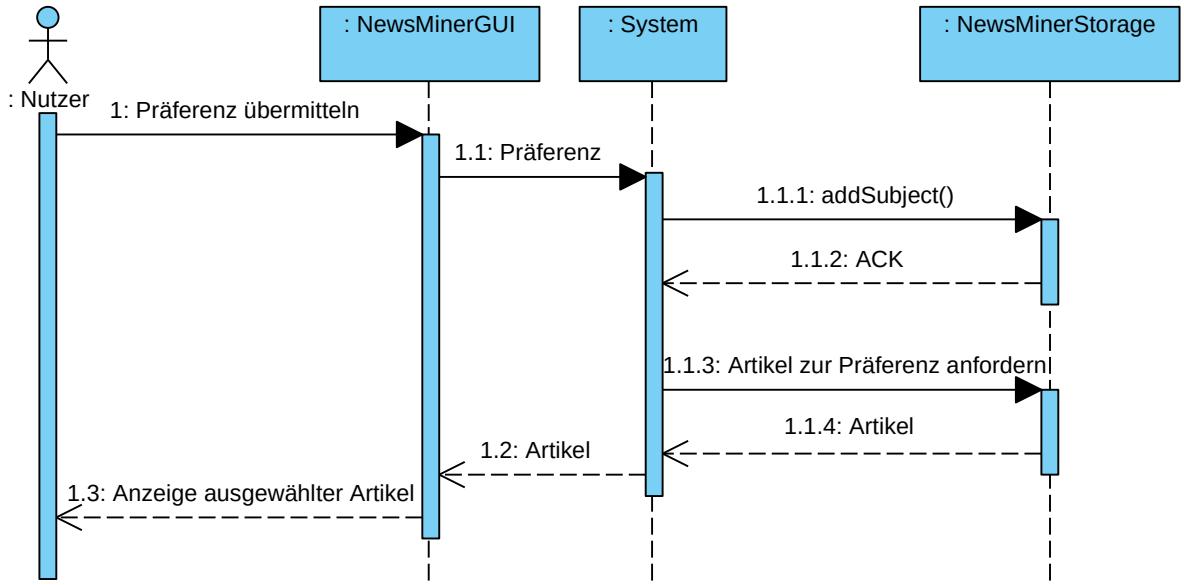

Abbildung 2.5: /F500/ Präferenzen angeben

nächsten Schritt erfolgt von dort ausgehend eine Bestätigung an das *System* und dann an die *NewsMinerGUI*, welche dem Nutzer den ausgewählten Trend anzeigt.

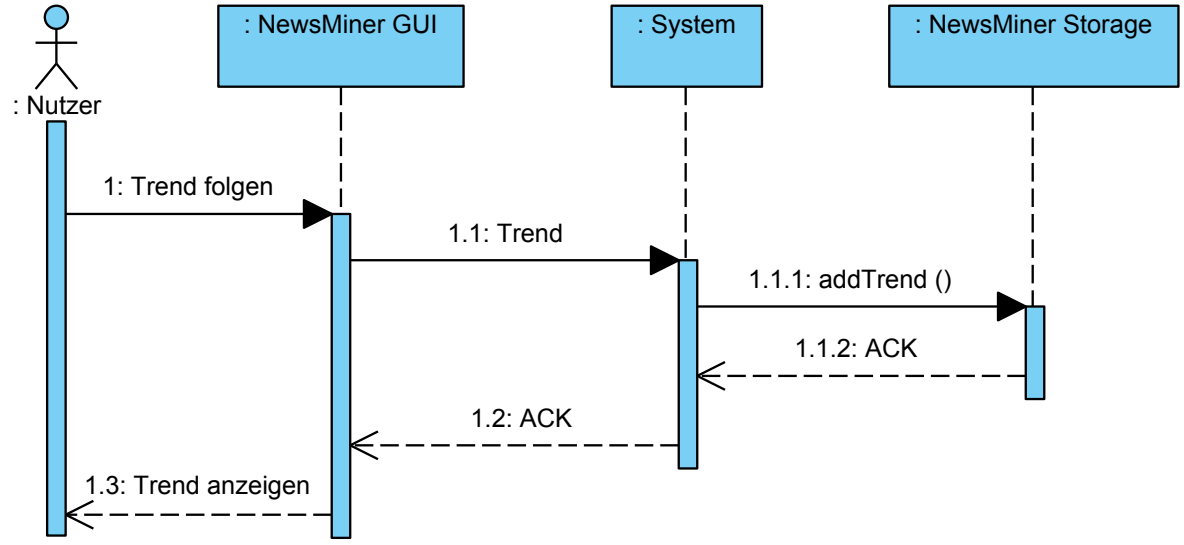

Abbildung 2.6: /F600/ Trend folgen

2.7 Analyse von Funktionalität /F700/: Trend nicht mehr folgen

Die Funktion “Trend nicht mehr folgen”, ist ebenfalls eine Erweiterung von *News Miner*. Dem Nutzer wird es mit dieser Funktion ermöglicht, einem Trend, dem er bisher gefolgt ist, wieder aus seinem Nutzerprofil zu löschen. Dies wird in dem Sequenzdiagramm in Abbildung 2.7 bildlich veranschaulicht.

Der Nutzer kann in der *NewsMinerGUI* einen Trend abwählen, dem er bisher gefolgt ist. Diese Information wird an das *System* und von da aus an den *NewsMinerStorage* weitergeleitet und dort abgespeichert, also aus diesem gelöscht. Nach diesem Vorgang wird von dem *NewsMinerStorage* über das *System* eine Bestätigung an die *NewsMinerGUI* gegeben, welche dem Nutzer eine Bestätigung darüber anzeigt, dass dieser ab sofort dem ausgewählten Trend nicht mehr folgt.

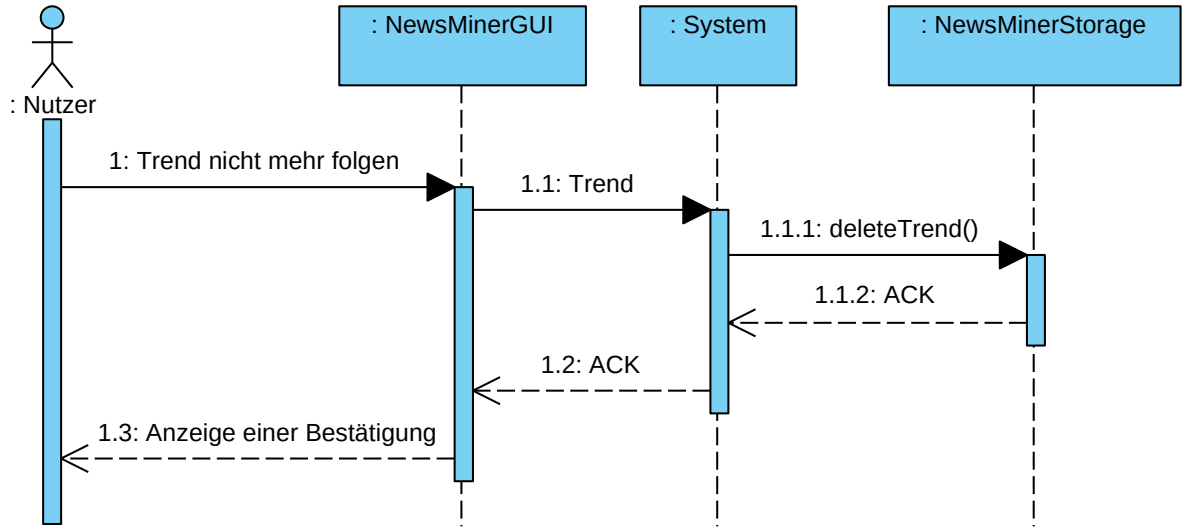

Abbildung 2.7: /F700/ Trend nicht mehr folgen

2.8 Analyse von Funktionalität /F800/: Nachrichtenquelle hinzufügen

“Nachrichtenquelle hinzufügen” ist eine Kernfunktion der Anwendung *News Miner*, die der Nutzer mindestens einmal ausführen muss, um Nachrichten zu erhalten. Diese Funktion und die beteiligten Komponenten werden im Sequenzdiagramm in Abbildung 2.8 dargestellt.

In der *NewsMinerGUI* muss zunächst eine Nachrichtenquelle ausgewählt werden. Diese Nachrichtenquelle wird anschließend über das *System* in den *NewsMinerStorage* eingepflegt. Falls die-

se Nachrichtenquelle schon im *System* eingetragen ist, wird sie nur im Nutzerprofil gespeichert. Nach der erfolgreichen Speicherung wird über das *System* eine Bestätigung an die *NewsMinerGUI* geleitet, welche dem Nutzer eine Meldung anzeigt, dass die Nachrichtenquelle gespeichert wurde.

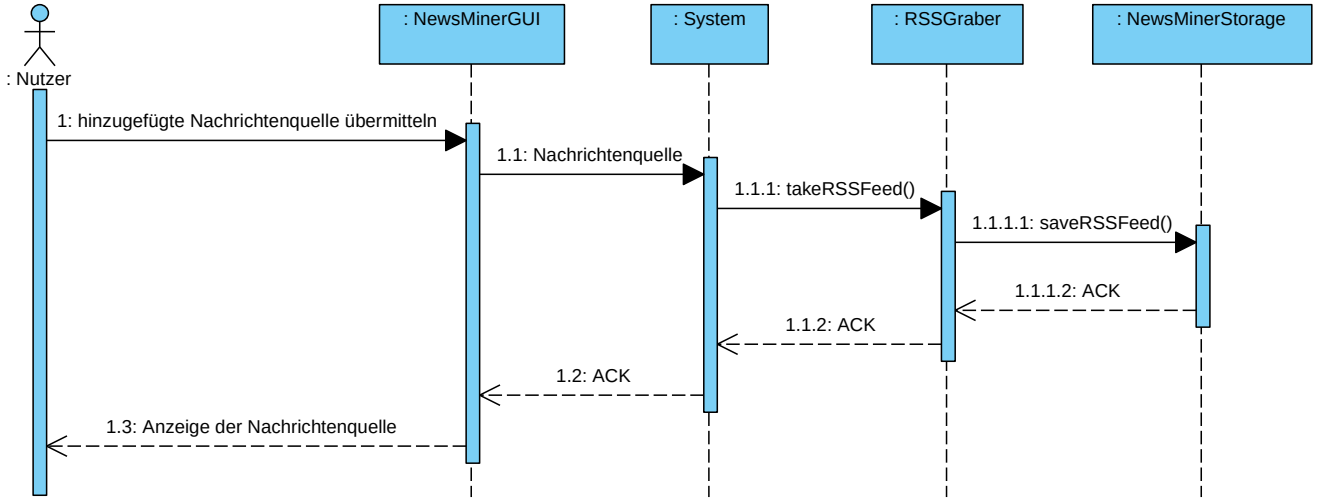

Abbildung 2.8: /F800/ Nachrichtenquelle hinzufügen

2.9 Analyse von Funktionalität /F900/: Nachrichtenquelle löschen

News Miner bietet dem Nutzer die Möglichkeit, ausgewählte Nachrichtenquellen wieder aus dem Nutzerprofil zu löschen, wenn er keine Nachrichtenartikel mehr von dieser Quelle erhalten möchte. Im folgenden Sequenzdiagramm in Abbildung 2.9 wird diese Funktion bildlich erläutert.

Um eine bisher im Nutzerprofil gespeicherte Nachrichtenquelle wieder zu löschen, muss der Nutzer in der *NewsMinerGUI* diese gewünschte Nachrichtenquelle auswählen. Diese Information wird daraufhin über das *System* an den *NewsMinerStorage* geleitet, in dem diese Nachrichtenquelle aus dem Nutzerprofil gelöscht wird. Nach der erfolgreichen Entfernung dieser Quelle wird eine Bestätigung über das *System* an die *NewsMinerGUI* geleitet, welche dem Nutzer mitteilt, dass die Nachrichtenquelle gelöscht wurde.

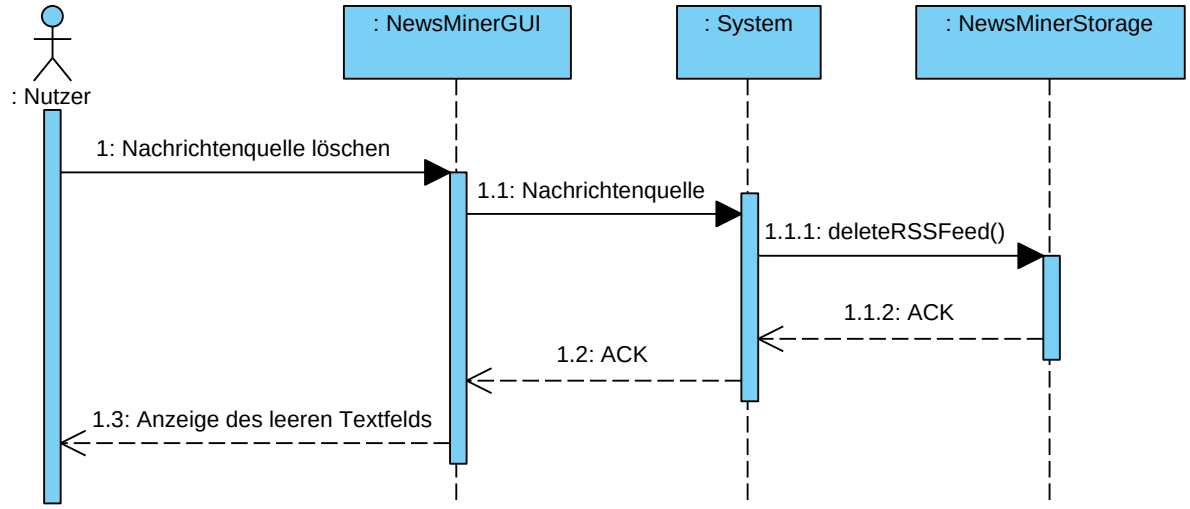

Abbildung 2.9: /F900/ Nachrichtenquelle löschen

3 Resultierende Softwarearchitektur

Der *News Miner* wird durch eine Vielzahl von Subsystemen, die durch einzelne Komponenten realisiert werden, implementiert. Dies wird in dem Komponentendiagramm in Abbildung 3.1 bildlich dargestellt. Die für den Nutzer sichtbare Komponente ist die *NewsMinerGUI*. Auf seiner grafischen Oberfläche ist es ihm möglich, sein persönliches Nutzerprofil zu erstellen. Im Gegenzug stellt die *NewsMinerGUI* ihm nun gefilterte Artikel dar. Dieser Vorgang wird durch die Komponenten des eigentlichen Systems realisiert. Dies sind der *TwitterCrawler*, der *RSS-Grabber*, der *TrendExtractor* und der *NewsAggregator*. Diese Subsysteme arbeiten alle zusammen, um aus den auf Twitter veröffentlichten Tweets und verschiedensten RSS-Feeds aktuelle Artikel zu extrahieren. Um diese Vorgänge umsetzen zu können, müssen verschiedene Informationen (zwischen-)gespeichert werden. Hierfür ist der *NewsMinerStorage* zuständig, welcher die gesamte Speicherverwaltung von *News Miner* übernimmt und somit auch sämtliche Nutzerprofile abspeichert. All diese Komponenten sind für sich abgeschlossene Subsysteme, die erst in ihrer Verknüpfung die vollständige Funktionalität von *NewsMiner* bilden.

3.1 Komponentenspezifikation

Komponente $\langle C10 \rangle$: NewsMinerGUI
Die *NewsMinerGUI* hat die Aufgabe, eine Verbindung zum Nutzer herzustellen. Der Nutzer kann auf der von der *NewsMinerGUI* zur Verfügung gestellten grafischen Oberfläche sämtliche persönlichen Nutzereinstellungen tätigen, wie die Registrierung bei *News Miner*, die Anmeldung und die Bearbeitung des Nutzerprofils. Die *NewsMinerGUI* übernimmt somit die Nutzerverwaltung und speichert alle relevanten Daten im *NewsMinerStorage* ab. Des Weiteren stellt die *NewsMinerGUI* dann individuell für jeden Nutzer passende aktuelle Nachrichtenartikel, welche sie aus dem *NewsMinerStorage* erhält, grafisch dar.

Komponente $\langle C20 \rangle$: TwitterCrawler
In dem Statechart in Abbildung 3.2 wird der *TwitterCrawler* näher dargestellt. Er ist dafür zuständig, dass auf Twitter neu veröffentlichte Tweets kontinuierlich gespeichert werden. Er liest somit ständig den Twitter-Stream mit und speichert in Intervallen Tweets im *NewsMinerStorage* ab. Hierbei filtert er zudem die Tweets nach ihrer verwendeten Sprache und wählt nur die

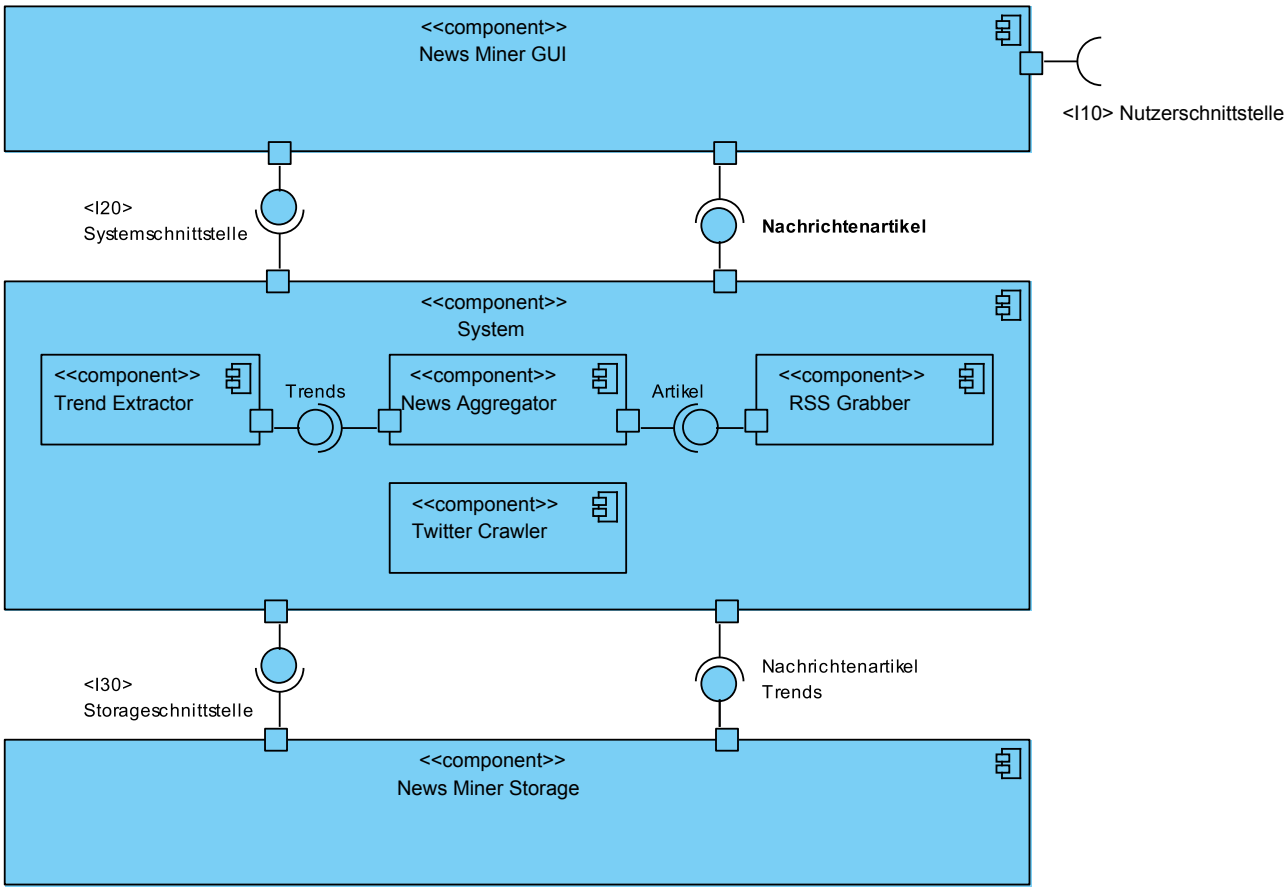

Abbildung 3.1: Aufbau von *NewsMiner*

englischen aus. Gespeichert werden dann diese Tweets mit all ihren Metadaten vom FileMaker in einer Textdatei, um ihre Herkunft auch zu einem späteren Zeitpunkt nachvollziehen zu können.

Komponente ⟨*C*30⟩**: RSSGrabber**
Der *RSSGrabber* wird in dem Statechart in Abbildung 3.3 verdeutlicht. Dieser holt regelmäßig RSS-Feeds und lädt neu dazugekommene Artikel nach. Die aus dem Internet geladenen Nachrichtenartikel werden nun mit Überschrift und Volltext im *NewsMinerStorage* gespeichert. Für den Fall, dass kein Volltext zu einem Artikel existiert, wird dieser nicht weiter verwendet.

Komponente ⟨*C*40⟩**: TrendExtractor**
Der *TrendExtractor* erhält die vom *TwitterCrawler* gesammelten Tweets und durchsucht diese mit Hilfe eines Suchalgorithmus auf auffällig häufig auftauchende Wörter. Diese gefunden Trends werden nun mit dem Zeitpunkt ihres Auftretens und den passenden Schlagwörtern im *NewsMinerStorage* abgespeichert.

Komponente ⟨*C*50⟩**: NewsAggregator**
Der *NewsAggregator* erhält die Nachrichtenartikel, die vom *RSSGrabber* geladen wurden, und die Trends, die der *TrendExtractor* identifiziert hat. Nun ist es die Aufgabe des *NewsAggregator*, die Artikel mit den Trends in Verbindung zu bringen. Hierfür werden die Artikel auf die Schlagwörter der Tweets, die die Trends behandeln, durchsucht. Diese Paarungen werden dann im *NewsMinerStorage* abgespeichert und können auf der Oberfläche der *NewsMinerGUI* dargestellt werden.

Komponente ⟨*C*60⟩**: NewsMinerStorage**
Der *NewsMinerStorage* übernimmt die gesamte Speicherverwaltung des *News Miner*. Er speichert sämtliche Daten, die in den einzelnen Komponenten des Systems entstanden sind, ab und stellt ihnen diese zu einem späteren Zeitpunkt zur Weiterverarbeitung zur Verfügung. Somit werden im *NewsMinerStorage* Nutzerdaten, Tweets, Trends und Nachrichtenartikel verwaltet.

3.2 Schnittstellenspezifikation

Schnittstelle ⟨*I*10⟩**: Nuterschnittstelle**

Operation	Beschreibung
input()	Der Nutzer gibt über die Tastatur Eingaben an den *News Miner*. Die *NewsMinerGUI* arbeitet dann mit diesen Daten weiter.

Schnittstelle $\langle I20 \rangle$: Systemschnittstelle

Operation	Beschreibung
addUser()	Der Nutzer registriert sich neu bei *News Miner*. Seine Daten werden an die *NewsMinerGUI* übergeben, ein neues Nutzerprofil wird angelegt und im *NewsMinerStorage* abgelegt.
logInUser()	Der Nutzer meldet sich nach erfolgreicher Registrierung bei *News Miner* an. Seine Angaben werden an die *NewsMinerGUI* übergeben und überprüft.
addRSSFeed()	Der Nutzer fügt neue Nachrichtenquellen hinzu. Hier wird die URL vom gewünschten RSS-Feed an die *NewsMinerGUI* übergeben, welche dann den jeweiligen RSS-Feed zum zugehörigen Nutzerprofil hinzufügt.
addTopic()	Der Nutzer fügt ein präferiertes Themengebiet zu seinem Nutzerprofil in der *NewsMinerGUI* hinzu. Dies wird im *NewsMinerStorage* abgelegt und es können dem Nutzer fortan Nachrichtenartikel zu diesem Themengebiet angezeigt werden.
saveTrend()	Der Nutzer folgt einem bestimmten Artikel. Dieses wird der *NewsMinerGUI* übergeben und im *NewsMinerStorage* abgelegt. Zu dieser Neuigkeit werden fortan präferiert Artikel angezeigt.

Schnittstelle $\langle I30 \rangle$: Storageschnittstelle

addSubject()	Der Nutzer fügt ein präferiertes Thema zu seinem Nutzerprofil hinzu. Dies wird im *NewsMinerStorage* abgelegt und es können dem Nutzer fortan Nachrichtenartikel zu diesem Thema angezeigt werden.
saveRSS(String)	Es werden die vom *RSSGrabber* geladenen Nachrichtenartikel im *NewsMinerStorage* abgespeichert.
saveTweet()	Hier werden die vom *TwitterCrawler* mitgelesenen Tweets von Twitter im *NewsMinerStorage* abgelegt.
addTrends()	Die vom *TrendExtractor* gefilterten Trends werden im *NewsMinerStorage* abgespeichert.
saveNews()	Nachdem der *NewsAggregator* zu jedem Trend einen zugehörigen Nachrichtenartikel gefunden hat wird zu den beiden zugehörigen Quellen im *NewsMinerStorage* ein Verweis abgespeichert.

3.3 Protokolle für die Benutzung der Komponenten

Die für den *News Miner* verwendeten Komponenten *TwitterCrawler* und *RSSGrabber* sind auch in einem anderen Zusammenhang gut einsetzbar, also zur Wiederverwendung geeignet. Der *TwitterCrawler* sorgt dafür, dass permanent Tweets abgespeichert werden. Diese können auch für

andere Zwecke, wie zum Beispiel Forschungen in den Sozialwissenschaften, verwendet werden. Dasselbe gilt auch für den *RSSGrabber*, auch die von ihm gespeicherten Nachrichtenartikel können in einem anderen Zusammenhang sinnvoll genutzt werden. Sie sind zudem auch in andere Softwareprodukte einfach einzubinden, da sie eigenständig arbeiten und keine weiteren Abhängigkeiten haben, somit sind sie unabhängige Subsysteme.

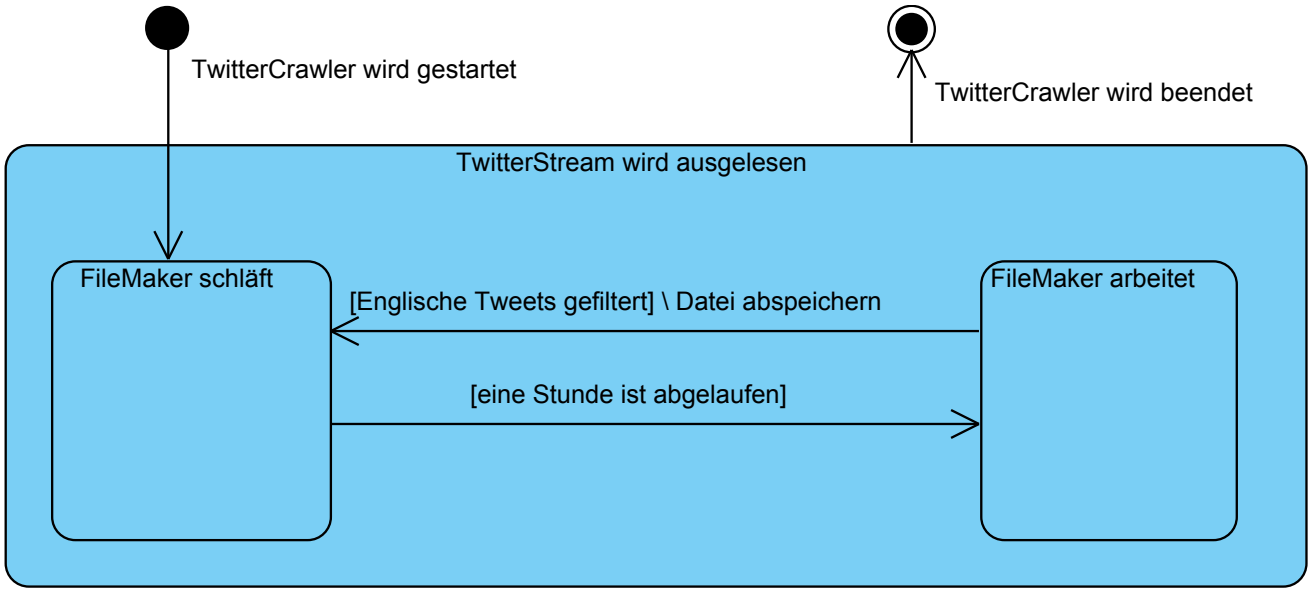

Abbildung 3.2: Ablauf des *Twittercrawlers* $\langle C20 \rangle$

Die anderen Komponenten des Systems - der *TrendExtractor* und der *NewsAggregator* - sind nicht zur Weiterverwendung geeignet. Sie sind sehr stark von den anderen Komponenten abhängig und genau auf den *News Miner* zugeschnitten. Somit ist die Einbindung in ein anderes Produkt sehr umständlich und somit wenig sinnvoll. Dasselbe gilt für die *NewsMinerGUI*, also ihr grafische Oberfläche, und den *NewsMinerStorage*, welches für die Speicherverwaltung zuständig ist. Sie sind explizit für das System von *News Miner* entwickelt worden und nur schwer auf andere Systeme zu übertragen.

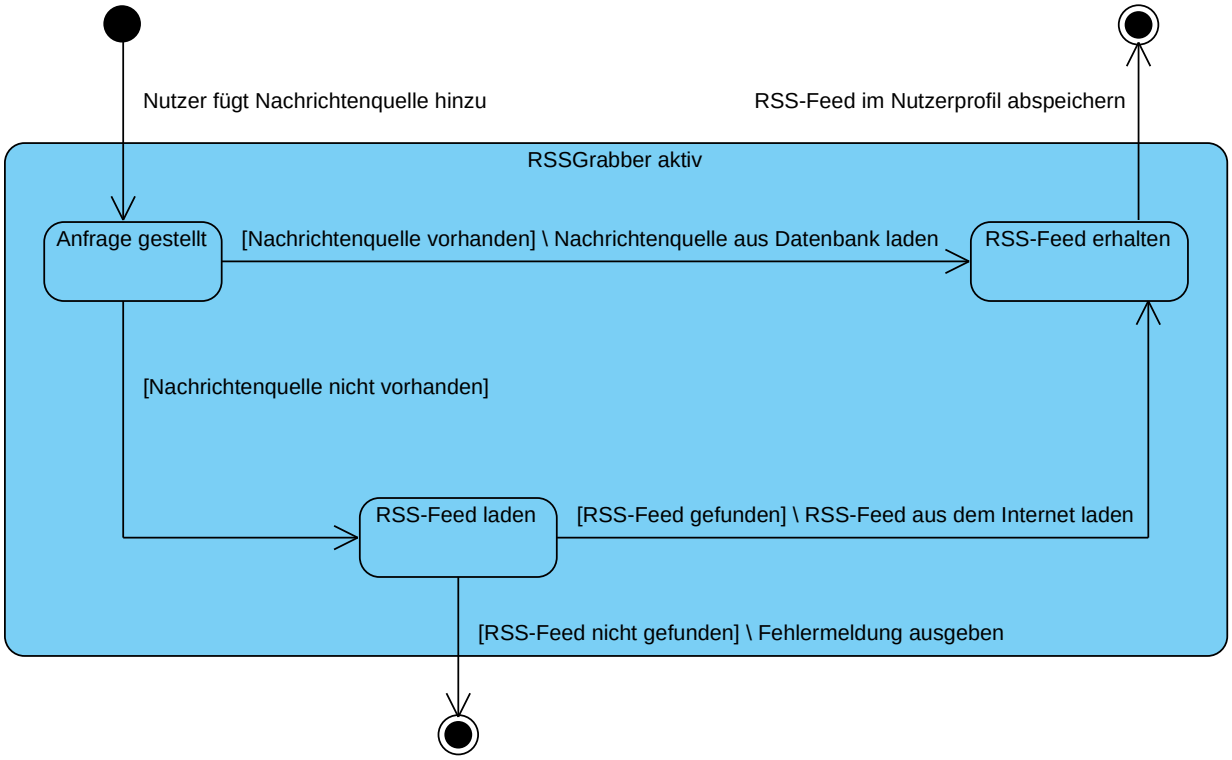

Abbildung 3.3: Darstellung des *RSSGrabbers* $\langle C30 \rangle$

4 Implementierungsentwurf

In den folgenden Abschnitten werden alle verwendeten Klassen der Komponenten von *NewsMiner* und die dazugehörigen Bibliotheken dokumentiert und ihre Implementierung erläutert.

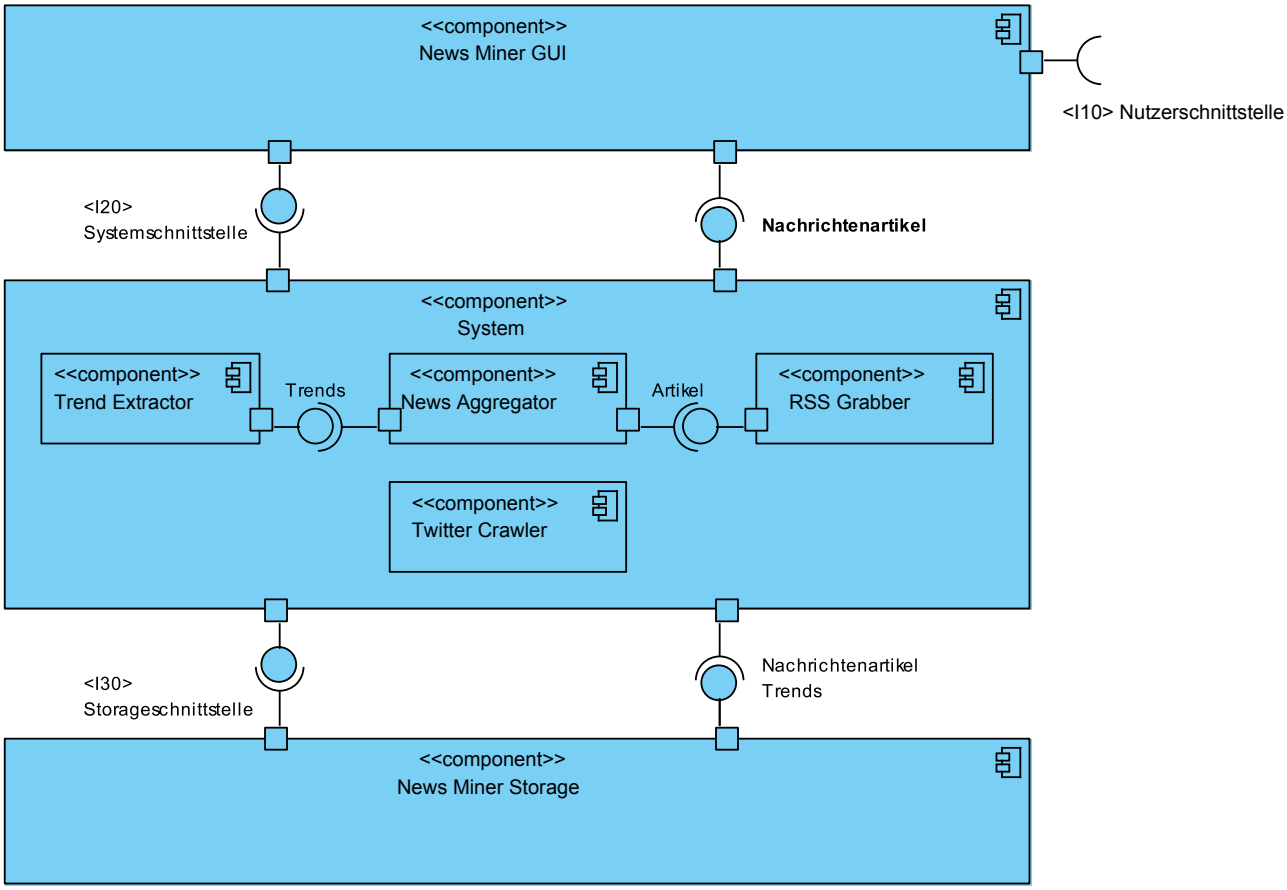

Abbildung 4.1: Komponentendiagramm

4.1 Implementierung von Komponente $\langle C10 \rangle$: *NewsMinerGUI*

Diese Komponente implementiert die grafische Benutzeroberfläche der Webanwendung *NewsMiner*. Um dies zu realisieren, wird das Framework Play verwendet, da dies viele benötigte Elemente bereits zur Verfügung stellt.

Die *NewsMinerGUI* verwaltet die gesamte Kommunikation zwischen dem Nutzer und den übrigen Komponenten. So ist sie dafür zuständig, die Nutzereingaben zu erfassen und an das *System* zu übergeben, damit die Daten weiter verarbeitet werden können. Zudem gibt die Benutzeroberfläche Rückmeldungen wie Bestätigungen und Fehlermeldungen sowie erfolgreich ermittelte Ergebnisse von den anderen Komponenten an den Benutzer weiter.

4.1.1 Klassendiagramm

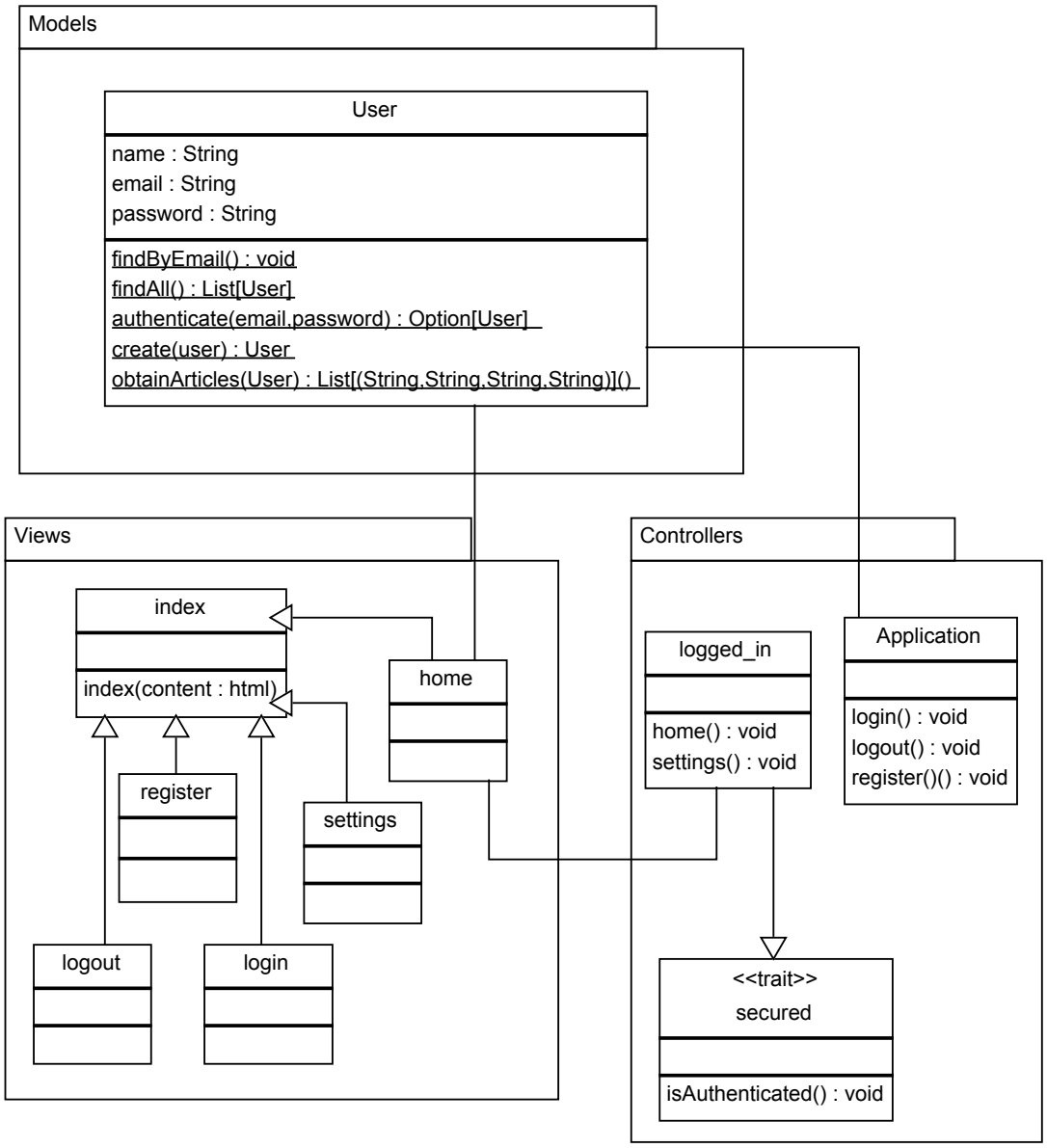

Abbildung 4.2: Klassendiagramm für *NewsMinerGUI* ⟨*C*10⟩

4.1.2 Erläuterung

User⟨*CL*10⟩

Aufgabe

Verwaltung der persönlichen Nutzereinstellungen

Attribute

- `username` Nutzername des Benutzers
- `e-mail` E-mail-Adresse des Nutzers
- `password` Passwort des Nutzers

Operationen

- `findMyEmail()` Diese Methode ermittelt die E-mail-Adresse zu dem jeweiligen Benutzer.
- `findAll()` Es werden alle Daten des Nutzers ermittelt.
- `authenticate(String email, String password)` Die Operation verifiziert den Nutzer, indem überprüft wird, ob die eingegebenen Daten mit denen in der Nutzerverwaltung übereinstimmen.
- `create(User user)` Ein neues Nutzerkonto wird angelegt.
- `obtainArticles(User user)` Die Nachrichtenartikel für einen bestimmten Nutzer werden angefordert.

Kommunikationspartner

Application, Home, *NewsAggregator*, *NewsMinerStorage*

LoggedIn⟨*CL*20⟩

Aufgabe

Durchführung der Anmeldung

Operationen

- `home()` Die Hauptseite wird konfiguriert.
- `settings()` Diese Methode lädt die (benutzerdefinierten) Einstellungen.

Kommunikationspartner

Home

Secured⟨*CL*30⟩

Aufgabe

Sicherstellung der Verbindung

Operationen

`isAuthenticated()` Es wird überprüft, ob der Nutzer angemeldet ist.

Kommunikationspartner

LoggedIn

Application⟨*CL*40⟩

Aufgabe

Konfiguration des An- und Abmeldevorgangs sowie der Registrierung

Operationen

- `logIn()` Diese Methode gewährt dem Nutzer bei erfolgreicher Authentifizierung Zugriffsrechte auf das jeweilige Nutzerkonto.
- `logOut()` Der Nutzer meldet sich von *NewsMiner* ab, d.h. er kann erst nach erneuter Anmeldung auf sein Nutzerkonto wieder zugreifen.
- `register()` Die Methode überprüft, ob die Eingabe vollständig ist und fügt die Daten anschließend zu einem neuen Nutzerkonto hinzu.

Kommunikationspartner

User

Index⟨*CL*50⟩

Aufgabe

Definition des Layouts von *NewsMiner*

Operationen

`index(content: html)` Der entsprechende html-Text wird eingefügt.

Kommunikationspartner

LogIn, Settings, Home, Register, LogOut

LogIn⟨*CL*60⟩

Aufgabe

Start- und Anmeldeseite der Webanwendung sowie Erklärung der Funktionalität von *NewsMiner*

Kommunikationspartner

Index

Settings⟨*CL*70⟩

Aufgabe

Festlegung der einzelnen Einstellungen

Kommunikationspartner

Index

Home⟨*CL*80⟩

Aufgabe

Hauptseite der Webanwendung

Kommunikationspartner

Index

Register⟨*CL*90⟩

Aufgabe

Seite zur Registrierung des Nutzers

Kommunikationspartner

Index

LogOut⟨*CL*100⟩

Aufgabe

Bestätigt dem Nutzer die erfolgreiche Abmeldung.

Kommunikationspartner

Index

4.2 Implementierung von Komponente $\langle C20 \rangle$: *TwitterCrawler*

Der TwitterCrawler hat die Aufgabe dem Sytstem ständig neue und aktuelle Tweets zur Verfügung zu stellen. Hierfür wird in erster Linie mit der Bibliothek Twitter4j gearbeitet, welche eine Reihe an Java-Methoden zur Arbeit mit dem von Twitter zur Verfügung gestellten TwitterStream bereit stellt. So liest die Klasse StreamGrabber mit Hilfe eines RawStreamListener laufend den TwitterStream mit und speichert sämtliche Tweets mit ihren Metadaten in einer CopyOnWriteArrayList zwischen. Hier werden zudem nur die englischen Tweets heraus gefiltert, in dem in den Metadaten nach den Beschreibung "lang:en"gesucht wird, welches die Sprache in der der Tweet verfasst wurde beschreiben soll. Mit der Speicherung dieser Daten ist dann die Methode writeToFile() beauftragt, welche mit Hilfe eines FileWriter's den Inhalt des erstellten Arrays in eine Textdatei schreibt und diese zur Identifikation mit einem eindeutigen Zeitpunkt der Speicherung versieht, dies geschieht über einen GregorianCalender. Zudem leert er das array zur weiteren Bearbeitung. Diese beiden Arbeitsschritte werden von der Methode saveToDisk() kontrolliert, während der SreamGrabber ständig und fortlaufend im Hintergrund arbeitet, kommt die Speicherung nur alle 2 Stunden zum Einsatz und stellt dann die neusten Daten dem TrendExtractor zur Verfügung.

4.2.1 Paket-/Klassendiagramm

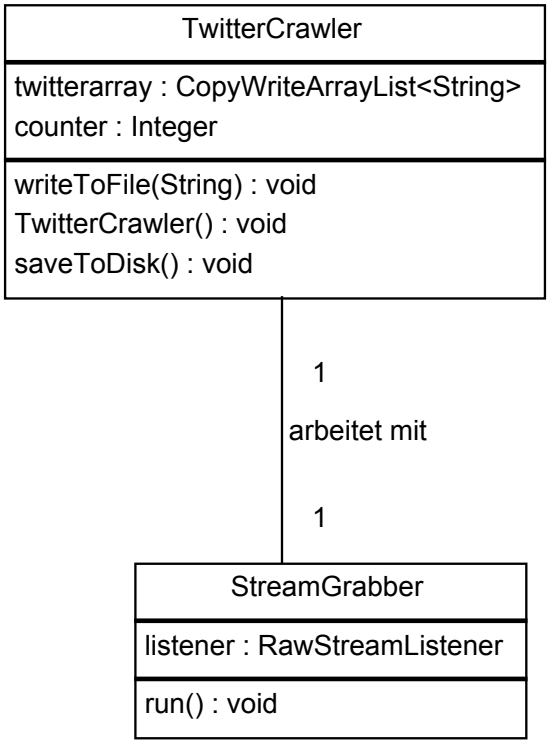

Abbildung 4.3: Klassendiagramm für *TwitterCrawler* ⟨*C*20⟩

4.2.2 Erläuterung

TwitterCrawler⟨*CL*110⟩

Aufgabe

Aktuellste Tweets in Form einer Textdatei anderen Komponenten zur Verfügung stellen.

Attribute

- `int counter` Gibt die Anzahl der bereits gespeicherten Dateien an.
- `CopyWriteArrayList<String> twitterarray` Speichert alle aktuellen Tweets mit ihren Metadaten zwischen.

Operationen

- `writeToFile(String filename)` Diese Operation speichert die Inhalt eines Arrays in eine Textdatei.
- `TwitterCrawler()` Diese Operation startet den TwitterCrawler.

- `saveToDisk()` Diese Operation speichert Textdatei in bestimmten Abständen ab.

Kommunikationspartner

StreamGrabber, *NewsMinerStorage*

StreamGrabber$\langle CL120 \rangle$

Aufgabe

Aktuellste Tweets aus dem TwitterStream auslesen.

Attribute

`RawStreamListener listener` Liest aktuelle Tweets ein.

Operationen

`run()` Diese Operation lässt den StreamGrabber dauerhaft laufen.

Kommunikationspartner

TwitterStream

4.3 Implementierung von Komponente $\langle C30 \rangle$: *RSSGrabber*

Als Erstes wird für jede URL und jeden Artikel ein Dokument erstellt. In dieses Dokument wird aus der angegebenen URL, der Artikeltitel, die Beschreibung, der Link und die URL hineingeschrieben. Außerdem wird ein Name für den RSS-Feed generiert, bei dem aus allen Titeln die Sonderzeichen durch ein Leerzeichen oder bestimmte Zahlen ersetzt werden. Die Feeds werden dann in der Datenbank für jeden User abgespeichert.

4.3.1 Paket-/Klassendiagramm

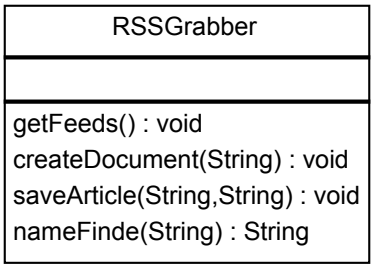

Abbildung 4.4: Klassendiagramm für *RSSGrabber* $\langle C30 \rangle$

4.3.2 Erläuterung

RSSGrabber$\langle CL130 \rangle$

Aufgabe

Holt regelmäßig RSS-Feeds und lädt neu dazugekommene Artikel nach.

Operationen

- `getFeeds()` Diese Methode holt für alle User die URLs ihrer favorisierten RSS-Feeds aus der Datenbank und ruft für jede einzelne die Methode `createDocumtent()` auf.
- `nameFinder(String attr)` Diese Methode generiert einen Namen für den RSS-Feed und zieht aus den Titeln alle Sonderzeichen heraus, um sie durch ein Leerzeichen oder bestimmte Zahlen zu ersetzen.
- `createDocument(String URL)` Diese Methode erstellt ein Dokument aus der angegebenen URL und schreibt in dieses den Artikeltitel, die Beschreibung, den Link und die URL rein.
- `saveArticle(String article, String feed)` Diese Methode speichert den Artikel in der Datenbank ab.

Kommunikationspartner

NewsAggregator

4.4 Implementierung von Komponente $\langle C40 \rangle$: *TrendExtractor*

Die Implementierung des *TrendExtractors* orientiert sich zu großen Teilen an den Algorithmen aus "See What's enBlogue – Real-time Emergent Topic Identification in Social Media"[1] - einem Paper von Foteini Alvanaki, Sebastian Michel, Krithi Ramamritham und Gerhard Weikum.

Die Funktionsweise ist dabei folgendermaßen: Die Tweets werden aus einer Datei ausgelesen, die der *TwitterCrawler* zur Verfügung stellt. Diese Datei enthält sehr viele unbenötigte Metadaten, weshalb der *TrendExtractor* nur den Volltext der Tweets und deren HashTags behält. Nach diesen HashTags wird dann sortiert. Kommt ein HashTag vor, der sich bereits in der Liste aller HashTags befindet, wird dessen Popularität erhöht und der zugehörige Tweet zu einer weiteren Liste hinzugefügt.

Daraus resultiert eine Liste von TagTupeln. Dieses ist eine Datenstruktur mit drei Attributen: Einem Hashtag, einem Popularitätswert und einer Liste von zugehörigen TweetsAndHashTags,

[1] http://dl.acm.org/citation.cfm?doid=2247596.2247636

was wiederum eine Datenstruktur ist. In einem TweetsAndHashTags-Objekt wird eine Liste von Hashtags zu jedem Tweet festgehalten, um später die verschiedenen Hashtags aufgrund eines gemeinsamen Vorkommens in einem Tweet einander zuordnen zu können.

Die entstandene TagTupel-Liste wird nach Popularitätswert sortiert und anschließend gekürzt. Das Ergebnis ist eine Liste der sogenannten seedTags. Dies sind alle Wörter, die in einem Zeitfenster besonders wichtig sind. Diese seedTags werden nun mit den Ursprungs-HashTags verglichen. Dabei wird geprüft, ob es gemeinsame Tweets gibt. Es entsteht eine Liste mit CorrelatedTags. Diese beinhalten zwei TagTupel und ihren Korrelationswert, welcher aufgrund folgender Formel [2] berechnet wurde: $corr(t_1, t_2) := \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} \cdot \frac{|S_1 \cap S_2|}{N}$ Dabei sind t_i hashTags, S_i die Zugehörigen Tweetmengen und N die Gesamtanzahl aller Hashtags im aktuellen Zeitfenster. Nun wird aufgrund einer Datei, die die Korrelationswerte der letzten 48 Stunden beinhaltet, ein Korrelationswert für jedes korrelierende TagTupel vorausgesagt und wiederum nach folgender Formel[3] ein Score ausgerechnet:

dt = Anzahl der Evaluationen die durchgeführt wurden, seitdem dieser Wert ausgerechnet wurde - in der Implementierung sein Alter
$error_{dt} = \frac{\text{(tatsächlicher Korrelationswert - vorhergesagter Korrelationswert)}}{\text{tatsächlicher Korrelationswert}}$
$reversePopularity_{dt} = \frac{1}{|log(popularity)|}$
$scoreTemp_{dt} = error \cdot reversePopularity$
$score_{dt} = scoreTemp \cdot e^{-0,2 \cdot dt}$Alle Werte werden für alle vorkommen dieses Paares in der History berechnet.x_{dt} ist also immer der Wert entsprechenden Alters
$score = max\{score_{dt}\}$

In dieser Formel ist von einem vorhergesagten Korrelationswert die Rede. Dieser wird aus alten Werten nach der Formel[4] $v'_t = a \cdot v_{t-1} + (1 - a) \cdot v'_{t-1}$ berechnet. Dabei ist v'_t der vorausgesagte Wert zum Zeitpunkt t und $0 < a \leq 1$

Dabei werden außerdem die korrelierenden Paare der letzten Stunde berücksichtigt, sodass diese nicht plötzlich verschwinden, sondern lediglich einen geringeren Score bekommen.

Die korrelierenden Trends werden dann mitsamt ihres Scores in einer Liste von CorrelatedTags abgelegt. Da bei der Scoreberechnung der vorhergesagte Wert mit eingeflossen ist, legt der Score fest, wie unerwartet und damit wie "trendy" die Kombination aus diesen beiden Tags ist. Mit anderen Worten: In dieser Liste stehen alle Trends mit einer Bewertung ihrer "Trendyness".

[2] http://dl.acm.org/citation.cfm?doid=2247596.2247636
[3] http://dl.acm.org/citation.cfm?doid=2247596.2247636
[4] http://dl.acm.org/citation.cfm?doid=2247596.2247636

4.4.1 Paket-/Klassendiagramm

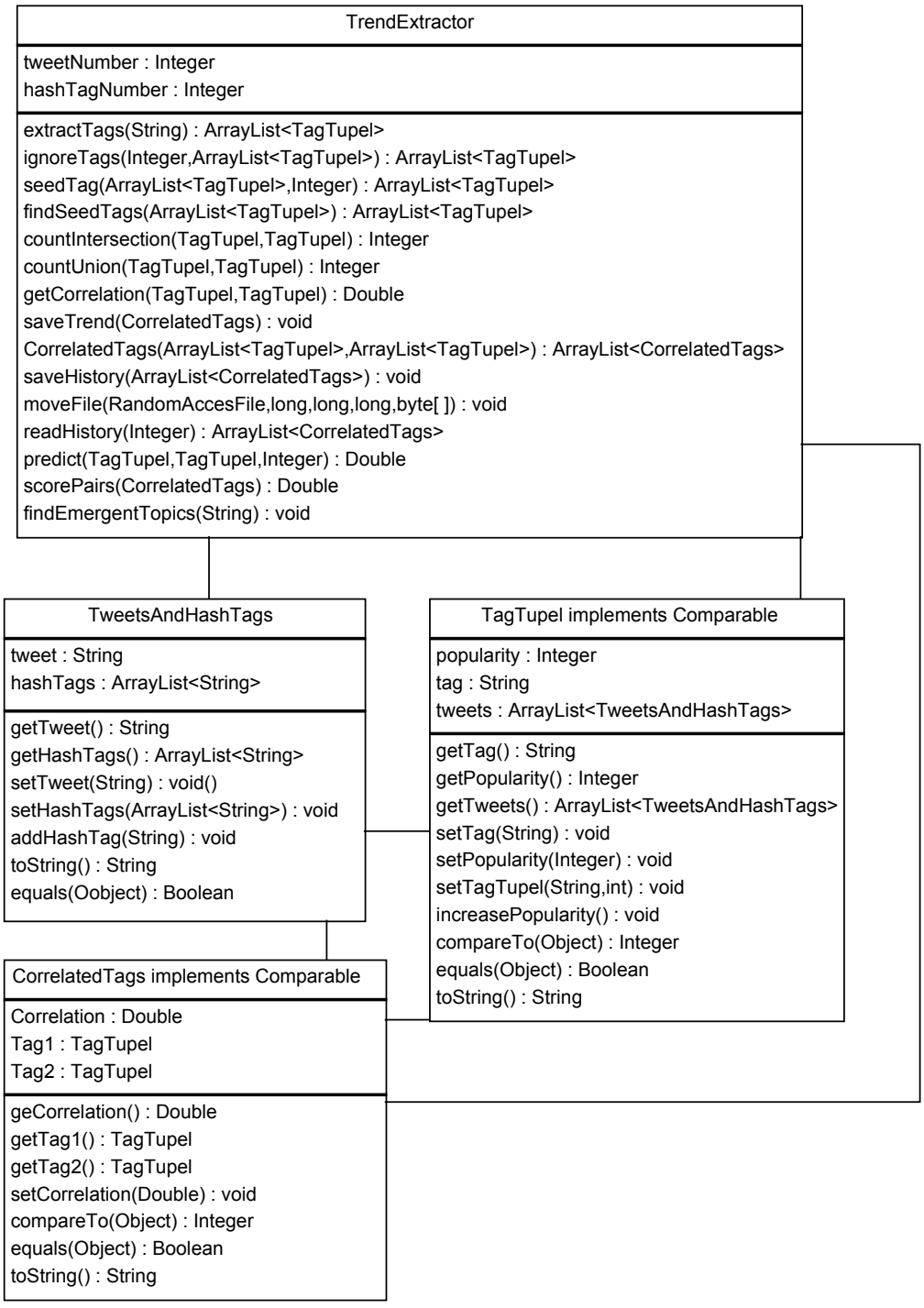

Abbildung 4.5: Klassendiagramm für *TrendExtractor* ⟨*C*40⟩

4.4.2 Erläuterung

TweetsAndHashTags⟨$CL140$⟩

Aufgabe

Datenstruktur zur Repräsentation von Tweets und zugehörigen Hashtags

Attribute

- `String tweet` Text des Tweets
- `ArrayList<String> hashTags` Liste aller zugehörigen Hashtags

Operationen

- `void addHashTags(String)` Fügt einen neuen Hashtag mit zugehörigen `hashTags` hinzu.
- `boolean equals(TweetsAndHashTags)` Vergleichsmethode. Gibt wahr zurück, wenn sowohl der Tweet-String, als auch die Hashtag-Liste übereinstimmen.
- `String toString()` Methode zur Konvertierung des Objektes zu einem String-Objekt

Kommunikationspartner

TagTupel, *TrendExtractor*

TagTupel⟨$CL150$⟩

Aufgabe

Datenstruktur zur Repräsentation von HashTags

Attribute

- `int popularity` Anzahl der Tweets in denen dieser HashTag in diesem Zeitfenster vorkam
- `String tag` Text des HashTags
- `ArrayList<TweetsAndHashTags> tweets` Liste aller Tweets in denen dieser HashTag in diesem Zeitfenster vorkommt. Zu jedem Tweet befinden sich in dieser Liste außerdem alle zugehörigen HashTags.

Operationen

- `void addTweet(TweetsAndHashTags)` Fügt einen neuen Tweet mit zugehörigen HashTags zu `tweets` hinzu.
- `void increasePopularity()` Erhöht die Popularität des HashTags um 1

- `int compareTo(Object)` Vergleichsmethode. Gibt 1 zurück, wenn das Object-Objekt vom Typ TweetsAndHashTags ist und dessen Popularität höher ist.
- `boolean equals(Object)` Vergleichsmethode. Gibt wahr zurück, wenn der Tag-String übereinstimmt.
- `String toString()` Methode zur Konvertierung des Objektes zu einem String-Objekt.

Kommunikationspartner

TweetsAndHashTags, *TrendExtractor*, CorrelatedTags

CorrelatedTags⟨*CL*160⟩

Aufgabe

Datenstruktur zur Repräsentation von zwei zusammengehörigen Hashtag

Attribute

- `TagTupel tag1` Erster HashTag
- `TagTupel tag2` Zweiter HashTag
- `double correlation` Wert, der angibt, wie sehr die beiden Hashtags zusammengehören. Wird in der Klasse *TrendExtractor*, in der Methode `getCorrelation` berechnet.

Operationen

- `int compareTo(Object)` Vergleichsmethode. Gibt 1 zurück, wenn Object-Objekt vom Typ CorrelatedTags ist und dessen `correlation` höher ist.
- `int equals(TagTupel,TagTupel)` Vergleichsmethode. Gibt wahr zurück, wenn die TagTupel übereinstimmen.
- `boolean equals(Object)` Vergleichsmethode. Gibt wahr zurück, wenn
- `String toString()` Methode zur Konvertierung des Objektes zu einem String-Objekt

Kommunikationspartner

TagTupel, *TrendExtractor*

TrendExtractor⟨*CL*170⟩

Aufgabe

Extrahieren von Tweets und Berechnung von Trendscores.

Attribute

- `tweetNumber` Anzahl der Tweets im aktuellen Zeitfenster
- `hashTag` Anzahl der Hashtags im aktuellen Zeitfenster

Operationen

- `extractTags(String filename)` Diese Methode liest die Datei "filename" ein. Diese Datei beinhaltet den Teil des Twitterstreams, den *TwitterCrawler* ausgelesen hat, also Tweets mit vielen zugehören Metadaten. Dabei stehen alle Informationen zu einem Tweet in zwei Zeilen der Datei. Die Methode extractTags() liest also immer zwei Zeilen aus der Datei und liest dann aus diesem String den Volltext des Tweets und seine zugehörigen HashTags. Daraus wird für jeden HashTag ein TagTupel gebaut. Dieses besteht aus dem Hashtag selbst und einem TweetsAndHashTags Objekt, das eine Liste von zugehörigen Tweets mit deren zugehörigen HashTags enthält. Wird ein HashTag erneut gefunden, so wird sein Popularitätswert erhöht und der Tweet und seine HashTags der TweetsAndHashTags Liste hinzugefügt.
- `ignoreTags(int k, ArrayList<TagTupel>, tagList)` Diese Methode löscht alle HashTags aus der Liste, die einen Popularitätswert haben der kleiner ist als l.
- `sortList(ArrayList<TagTupel> tagList)` Diese Methode sortiert die Liste der TagTupel absteigend nach ihrem Popularitätswert.
- `seedTag(ArrayList<TagTupel> tagList, int l)` Diese Methode schneidet die Liste nach l Elementen ab.
- `findSeedTags(ArrayList<TagTupel> seedList)` Diese Methode führt die Methoden seedTag, sortList und ignoreTags aus, sodass sie eine Liste der populärsten HashTags enthält. Diese heißen SeedTags.
- `countIntersection(TagTupel a, TagTupel b)` Diese Methode berechnet die Anzahl der gleichen Tweets zweier HashTags, also deren Durchschnitt.
- `countUnion(TagTupel a, TagTupel b)` Diese Methode berechnet die Anzahl der Tweets, die zwei HashTags zusammen haben, also deren Vereinigung.
- `getCorrelation(TagTupel a, TagTupel b)` Diese Methode berechnet beruhend auf der Anzahl der HashTags im aktuellen Zeitfenster und den Ergebnissen von countUnion() sowie countIntersection() den Korrelationswert von zwei HashTags, von denen mindestens einer ein seedTag sein muss.
- `correlatedTags(ArrayList<TagTupel> tagList, ArrayList<TagTupel> seedList)` Diese Methode iteriert durch die Liste von seedList und berechnet zu jedem Element aus der tagList den Korrelationswert. Dabei werden Tags ignoriert, die mit keinem der Seedtags gemeinsam in einem Tweet vorkommen. Es entstehen also Paare von Tags die populär sind und in den gleichen Tweets vorkamen.

- `saveTrend(CorrelatedTags tag)` Diese Methode speichert die fertig berechneten Trends in der Datenbank.
- `saveHistory(ArrayList<CorrelatedTags> corList)` Diese Methode speichert die Tagpaare in einer History Datei, welche benötigt wird in der Methode predict(). Dabei werden nur die Tweets der letzten 48 Stunden gespeichert, d. h. die History Datei hat 24 Einträge und jedes Mal, wenn ein neuer Eintrag geschrieben wird, wird der älteste gelöscht.
- `readHistory(int time)` Diese Methode liest die vorher gespeicherten Trends wieder aus der History Datei. Der Faktor k gibt dabei an, um wie viele Stunden in der History zurückgegangen wird. Wenn k beispielsweise 3 ist, liest readHistory() aus der History den Eintrag von vor sechs Stunden aus. Das hängt vor allem damit zusammen, dass der *TwitterCrawler* genau alle zwei Stunden eine Datei erstellt, weshalb auch der *TrendExtractor* nur alle zwei Stunden einmal läuft.
- `predict(TagTupel tag1, TagTupel tag2, int time)` Diese Methode berechnet rekursiv aufgrund der History-Datei den erwarteten Korrelationswert von zwei Hash-Tags. Sie wird zu Beginn mit time = 0 aufgerufen, wodurch genau der neueste Eintrag in der History untersucht wird. Anschließend ruft sie sich selbst mit größer werdendem time-Wert auf, bis dieser 24 beträgt. So werden alle Tagpaare der letzten 48 Stunden mitsamt ihres Korrelationswertes berücksichtigt.
- `scorePairs(CorrelatedTags corel)` Diese Methode berechnet anhand des Korrelationswerts des Ergebnisses von predict() und einiger andere Faktoren den Score von zwei HashTags. Dieser gibt dann an, wie "trendy" diese sind. D.h. also, alle Hashtags mit einem hohen Score werden von *NewsMiner* als Trends erkannt.
- `findEmergentTopics(String filename)` Diese Methode führt alle Methoden aus. Sie ruft also zunächst extractTags() mit dem filename auf. Dieser String wird dem *TrendExtractor* vom *TwitterCrawler* übergeben und ist der Name der Datei, in der der Twitterstream des aktuellen Zeitfensters steht. Mit der entstandenen Liste der Hashtags wird dann findSeedTags() aufgerufen. Das liefert eine Liste von Seedtags, mit der dann correlatedTags() aufgerufen wird. Diese Liste von Tagpaaren wird am Schluss in der History gespeichert. Anschließend wird für alle korrelierenden Tagpaare der Score berechnet. Das Ergebnis ist eine Liste von korrelierenden Tagpaaren mit einem Score, welche als Liste von CorrelatedTags Objekten aufgefasst wird. Jedes dieser Paare wird dann in der Datenbank gespeichert. Außerdem wird mit der entstandenen Liste der *NewsAggregator* aufgerufen.

Kommunikationspartner

NewsAggregator, *TwitterCrawler*, TagTupel, CorrelatedTags, TweetsAndHashTags

4.5 Implementierung von Komponente $\langle C50 \rangle$: *NewsAggregator*

Der *NewsAggregator* wählt basierend auf vom *TrendExtractor* gelieferten Trends und Userdaten für den Benutzer interessante Newsartikel, die vom *RSSGrabber* geliefert werden, aus. Diese werden dem Benutzer dann von der *NewsMinerGUI* angezeigt. Technisch gesehen bestimmt der NewsAggregator als Maß zum Filtern die Kosinus-Ähnlichkeit

$$\forall t \in \text{Trends},\ a \in \text{Artikel}:\ \frac{t * a}{\|t\| * \|a\|}$$

der Artikel als Wortmengen in einen hochdimensionalen Raum abgebildet.

Damit diese Abbildung erfolgen kann, müssen zunächst unwichtige Worte, die nichts zur Bedeutung der Texte beitragen entfernt werden (sogenannte Stopwords). Dies wird durch Hashing effizient erreicht. Außerdem können Wörter natürlich konjugiert oder dekliniert vorliegen. Um trotzdem Vergleichbarkeit zu gewährleisten, müssen sie auf einen gleichen Wortstamm zurückgeführt werden (engl. stemming). Für die englische Sprache (auf die wir uns beschränken) existiert hierfür der Porter2-Algorithmus[5].

[5] http://snowball.tartarus.org/algorithms/english/stemmer.html

4.5.1 Klassendiagramm

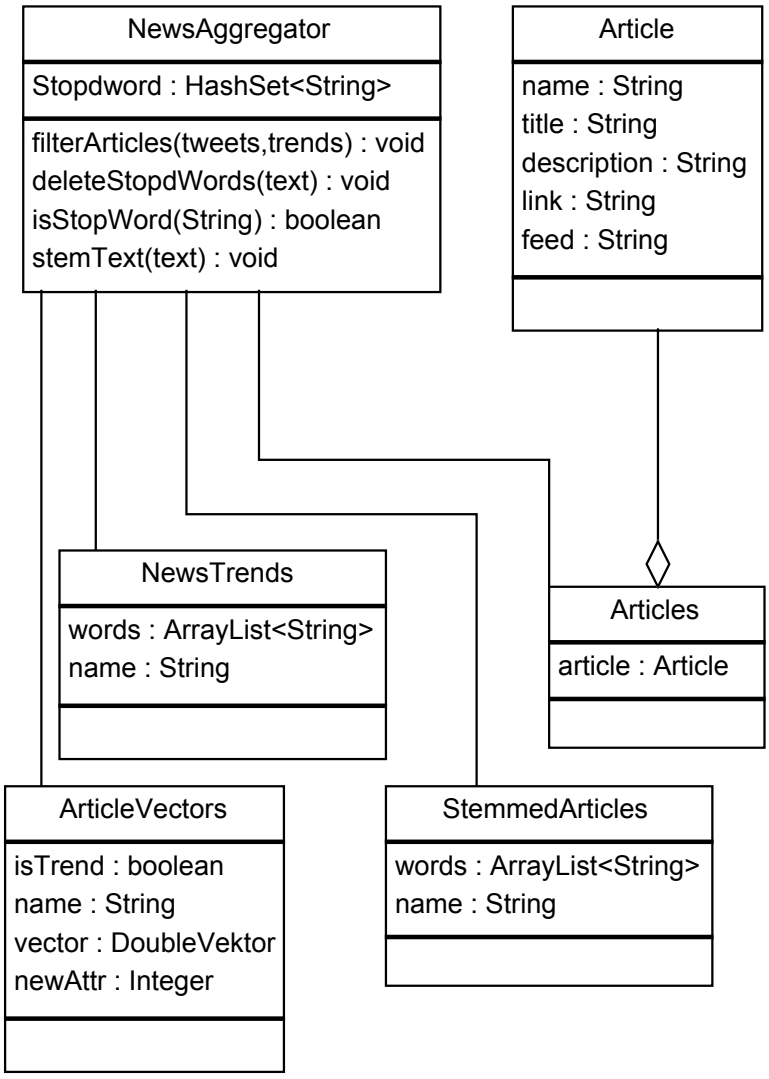

Abbildung 4.6: Klassendiagramm für den *NewsAggregator* $\langle C50 \rangle$

4.5.2 Erläuterung

NewsAggregator$\langle CL180 \rangle$

Aufgabe

Hauptklasse des NewsAggregator (Singleton)

Attribute

`stopwords` zum Entfernen von Füllworten

Operationen

- `filterArticles(trends, tweets)`
- `deleteStopWords(tbfiltered)` Füllworte werden aus "tbfiltered" entfernt
- `isStopWord(s)` entscheidet, ob ein Wort s ein Füllwort ist. Servicemethode für deleteStopWords()
- `stemText(tbstemmed)` Wörter eines Textes werden mittels des Porter2-Algorithmus auf einen Stamm zurückgeführt (nur englische Wörter)
- `stemWord(s)` Servicemethode analog zu isStopWord()

Article⟨$CL190$⟩

Aufgabe

Speicherklasse für Nachrichtenartikel aus RSS-Feeds

Attribute

- `name` Eindeutiger Name des Artikels (filename)
- `title` Titel des Artikels
- `description` Kurzfassung des Artikels
- `link` Link zum vollständigen Artikel (auf Ursprungsseite)
- `feed` Name des Newsfeed, aus dem der Artikel stammt

Articles⟨$CL200$⟩

Aufgabe

Singleton, über den Article eingesehen werden können.

Attribute

`articles` Liste von Article-Objekten

ArticleVectors⟨$CL210$⟩

Aufgabe

Speicherklasse für Artikel- und Trendvektoren, zusammen mit ihrem Namen als Metainformation.

Attribute

- `isTrend` Ist ein Trend oder ein Artikel gespeichert?
- `name` name des Trends/Artikels (um zuordnen zu können, woher dieser Vektor kommt)
- `vector` der eigentliche Vektor in einem hochdimensionalen Raum.

NewsTrends⟨$CL220$⟩

Aufgabe

Speicherklasse für Trends, bestehend aus ihrem Namen und ihrem Wortschatz

Attribute

- `words` Wortschatz aller Tweets, die zu dem Trend führen (Wortmenge)
- `name` name, um zuordnen zu können

Operationen

StemmedArticle⟨$CL230$⟩

Aufgabe

Speicherklasse für Article, deren Worte auf einen Stamm zurückgeführt wurden. Füllworte (stopwords) sind hier ebenfalls nicht enthalten.

Attribute

- `name` name, um zuordnen zu können
- `words` Wortmenge nunmehr von Füllwörtern befreiter, auf einen Stamm zurückgeführter Wörter.

5 Datenmodell

In dem folgenden Kapitel werden alle dauerhaft gespeicherten Daten aufgeführt und näher erläutert.

5.1 Diagramm

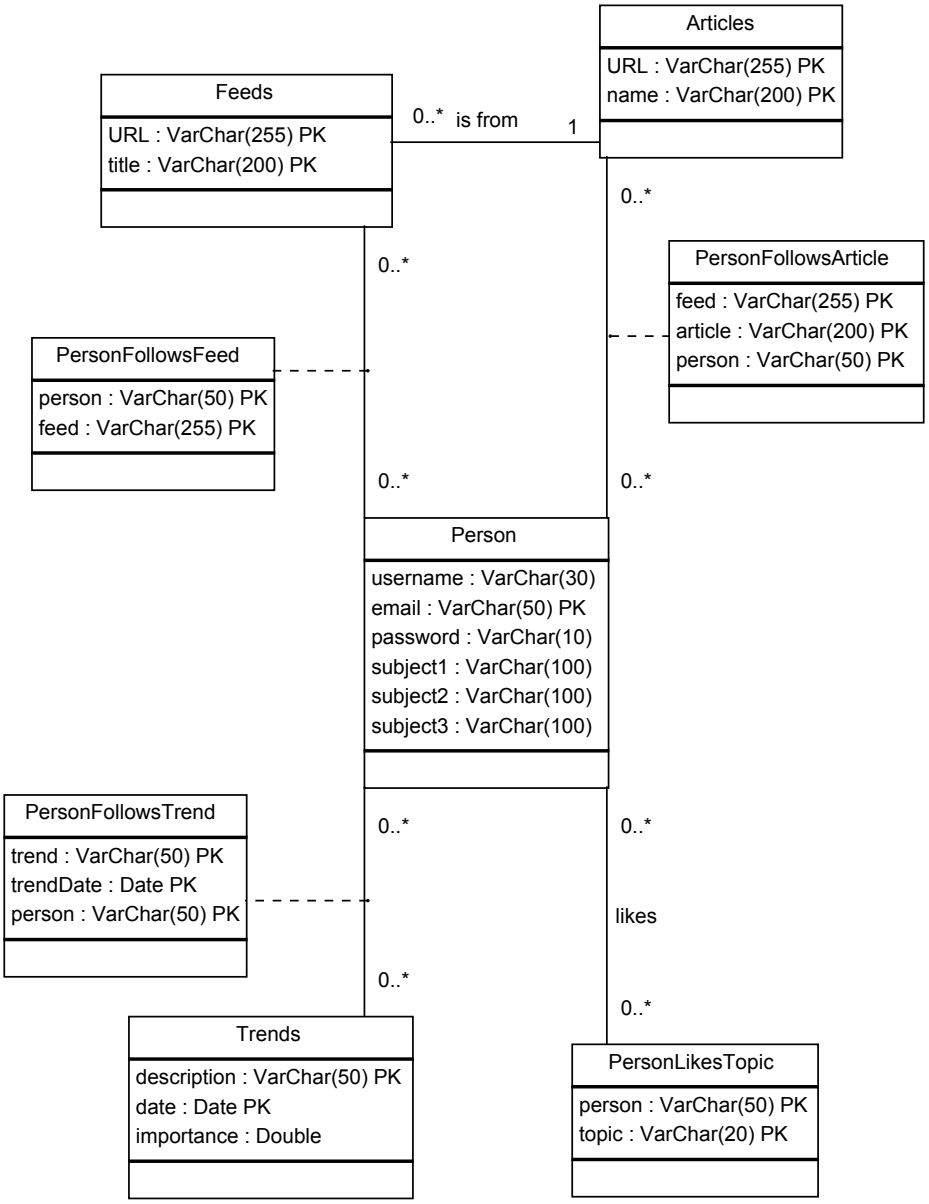

Abbildung 5.1: Klassendiagramm des Datenmodells

5.2 Erläuterung

Person ⟨*E*10⟩

Beziehung	Kardinalität
likesTopic	0..*
followsTrend	0..*
followsArticle	0..*
followsFeed	0..*

PersonLikesTopic ⟨*E*20⟩

Beziehung	Kardinalität
likesTopic	0..*

Trends ⟨*E*30⟩

Beziehung	Kardinalität
personFollows	0..*

PersonFollowsTrend ⟨*E*40⟩

Beziehung	Kardinalität
relationshipPersonTrend	0..*

Article ⟨*E*50⟩

Beziehung	Kardinalität
personFollows	0..*
isFrom	1

PersonFollowsArticle ⟨*E*60⟩

Beziehung	Kardinalität
relationshipPersonArticle	0..*

Feed ⟨*E*70⟩

Beziehung	Kardinalität
isFrom	0..*
personFollows	0..*

PersonFollowsFeed ⟨*E*70⟩

Beziehung	Kardinalität
relationshipPersonFeed	0..*

6 Serverkonfiguration

Der *NewsMiner* läuft auf einem Server. Der Benutzer erreicht ihn mit einem Webbrowser.
Der Server benötigt lediglich eine installierte Java Virtual Machine, auf der eine *.war-Datei ausgeführt werden kann.

7 Erfüllung der Kriterien

In diesem Kapitel wird darauf eingegangen, inwiefern die einzelnen Kriterien des Pflichtenheftes erfüllt werden.

7.1 Musskriterien

⟨*RM*1⟩ RSS-Feeds empfangen

Die *RSSGrabber*-Komponente ⟨*C*30⟩ von *NewsMiner* empfängt laufend RSS-Feeds und speichert diese auf dem Kundensystem zwischen. Es muss dabei genügend Speicherplatz vorhanden sein. Außerdem müssen alte Artikel wieder gelöscht werden, damit sie nicht versehentlich als neu gewertet werden.

⟨*RM*2⟩ Twitter Stream lesen

Analog zum *RSSGrabber* ⟨*C*30⟩ liest der *TwitterCrawler* ⟨*C*20⟩ fortlaufend den Sample-Stream der Twitter-API mit und puffert diesen auf dem Kundensystem. Es ist dabei wichtig, dass alte Twitter-Daten (=Tweets, die länger als 48 Stunden gespeichert sind) gelöscht werden, da sonst sehr viel Speicherplatz verschwendet wird.

⟨*RM*3⟩ Trends extrahieren

Das Extrahieren von Trends entfällt auf den *TrendExtractor* ⟨*C*40⟩. Dieser benötigt den *TwitterCrawler* ⟨*C*20⟩, um aus dessen gepufferten Daten Trends extrahieren zu können. Eine zentrale Speicherverwaltung über den *NewsMinerStorage* ⟨*C*60⟩ ist hier notwendig, um Konflikte zu vermeiden.

⟨*RM*4⟩ Unter Verwendung der Trends RSS-Nachrichten filtern

Der *NewsAggregator* ⟨*C*50⟩ filtert RSS-Nachrichten mit Trends, die er aus dem *NewsMinerStorage* ⟨*C*60⟩ bezieht. Er benötigt den *RSSGrabber* ⟨*C*30⟩ und *TrendExtractor* ⟨*C*40⟩, um arbeiten zu können.

⟨*RM*5⟩ gefilterte Nachrichten in ansprechender Form darstellen

Diese Aufgabe entfällt auf die *NewsMinerGUI* ⟨*C*10⟩, ein Webserver, der auf dem Play-Framework basiert.

⟨*RM*6⟩ Nutzerverwaltung

Die notwendige Nutzerverwaltung übernimmt die *NewsMinerGUI* ⟨*C*10⟩. Hierbei ist natürlich besonders auf Datensicherheit zu achten.

7.2 Sollkriterien

⟨*RS*1⟩ Die grafische Benutzeroberfläche soll übersichtlich gestaltet und intuitiv bedienbar sein.

Die *NewsMinerGUI* ⟨*C*10⟩ wird mittels eines Evaluationsbogens von 20 Testpersonen beurteilt, um eine hohe Intuitivität und Benutzerfreundlichkeit zu überprüfen und zu gewährleisten.

7.3 Kannkriterien

⟨*RC*1⟩ Trends folgen

⟨*RC*2⟩ Präferenzen angeben

⟨*RC*3⟩ Themen angeben

All diese Informationen sind nutzerbezogen und müssen entsprechend in der Nutzerverwaltung nach ⟨*RM*4⟩ abgespeichert werden. Ebenso muss der *NewsAggregator* ⟨*C*50⟩ diese Nutzerbezogenheit widerspiegeln und individuelle Nachrichten aggregieren.

7.4 Abgrenzungskriterien

⟨*RW*1⟩Twitter

Der Nutzer soll nichts vom Filterungsprozess via Twitter wissen müssen. Dies wird durch die konsequente Trennung von Front- und Backend in *NewsMiner* (Frontend: *NewsMinerGUI*, Backend: alle anderen Komponenten) und eine damit einhergehende größtmögliche Kapselung erreicht, wodurch die Berechnungen vollständig vor dem Nutzer verborgen sind.

⟨*RW*2⟩ Verlässlichkeit

Aufgrund des modularen Aufbaus und der Nutzung von bewährten Server-Betriebssystemen wird automatisch eine hohe Verlässlichkeit erreicht.

⟨*RW*3⟩ Trends ohne Feed

Der *NewsAggregator* filtert zunächst die vorhandenen Nachrichtenartikel mit allen Trends. Gibt es Trends ohne zugeordneten Artikel, werden diese ignoriert.

8 Glossar

3-Schichten-Architektur Die 3-Schichten-Referenz-Architektur ist ein weit verbreitetes Strukturierungsprinzip für Software. Hierbei werden die Bestandteile des Systems in Schichten aufgeteilt. Zwischen diesen Schichten kann der Nachrichtenaustausch nur über Schnittstellen stattfinden. Die Anwendung solcher Architekturen führt zu einer geringeren Kopplung verschiedener Systembestandteile untereinander.

API (Application Programming Interface) bzw. Programmierschnittstelle, die eine Anbindung von Anwendungsprogrammen auf Funktionen eines Systems oder eines Programms ermöglicht.

Applikation bzw. Anwendungssoftware, womit der Benutzer nicht systemtechnische Funktionalitäten unterstützen oder bearbeiten kann.

Backend ist der Teil der Anwendung, der näher am Betriebssystem ist. Im Gegensatz dazu steht das Frontend.

Framework ist eine Rahmenstruktur, die Regeln definiert, wie ein Programmierer eine Anwendung realisieren soll. Zudem stellt es Bibliotheken und Werkzeuge bereit, um die Programmierung zu erleichtern.

Frontend ist der Teil der Anwendung, der näher am Nutzer ist. Im Gegensatz dazu steht das Backend.

Hauptseite ist die Seite, auf die der Nutzer kommt, wenn er sich erfolgreich am System angemeldet hat.

indexieren etwas zu einem Index hinzufügen.

Index eine Art Register oder Nachschlagewerk für Daten, welches die Suchzeit verringert.

internetfähiges Endgerät ist z.B. ein PC, Tablet PC, Smartphone oder Fernseher.

Kapselung schützt Informationen vor dem Zugriff von außen. Es werden klare Schnittstellen für den Zugriff vorgegeben, wodurch unkontrollierter Zugriff vermieden wird.

Persistenzschicht ist in einer 3-Schichten-Architektur für die Speicherverwaltung zuständig.

Puffer, puffern ein Puffer ist ein Speicher für die Zwischenlagerung von Daten. Daten zu puffern bedeutet also, sie in einem dafür vorgesehenen Zwischenspeicher abzulegen, um nach einem kurzen Zeitraum darauf zuzugreifen.

RSS-Feed eine simple und strukturierte Veröffentlichung in einem XML-Format, die bei Veränderung einer Website automatisch über die jeweiligen Änderungen informiert.

Server ist ein zentraler und leistungsstarker Netzwerkrechner, der Daten oder Ressourcen bereit stellt.

Startseite ist die Seite, auf der sich der Nutzer anmelden bzw. registrieren kann, und die angezeigt wird, wenn ein beliebiger Internetnutzer auf den Internetdienst gelangt.

Trend ist ein steigender oder fallender Wandlungsprozess.

Tweet ist eine kurze Nachricht mit maximal 140 Zeichen, die über den Internetdienst Twitter veröffentlicht wird.

Workflow (dt. Arbeitsablauf) ist eine Beschreibung für eine Folge von Arbeitsschritte.