

Does it feel like you're wading through honey to get your news?

Save time - Use InComb



Detailkonzept

Version	1.0
Autoren	Marc Sieber, Milan Bharanya, Josef Weibel
Letzte Änderung	22. Januar 2015



1 Änderungsübersicht

Version	Datum	Autor	Beschreibung der Änderung	Betroffene Kapitel
1.0	22.01.2015	Alle	Initiale Dokumentation	alle

2 Referenzierte Dokumente

Dokumentbezeichnung	Dokumentname	Version	Kurzbeschreibung
Pflichtenheft	Pflichtenheft.docx	1.0	Enthält die Ausgangssituation, alle Anforderungen und Ziele.
Konzept	Konzept.docx	1.0	Enthält eine grobe Variante des Konzepts

Inhaltsverzeichnis

1 Änderungsübersicht.....	2
2 Referenzierte Dokumente.....	2
3 Einleitung.....	5
3.1 Inhalt und Zweck des Dokuments.....	5
3.2 Abkürzungen und Definitionen.....	5
4 Fachliche Spezifikation.....	6
4.1 Logische Datenbeschreibung (ERD)	6
4.1.1 Inhalte	6
4.1.2 Benutzer	10
4.2 Spezifische Benutzeranforderungen.....	14
4.2.1 Use Cases Hauptseite	14
4.2.2 Use Cases Nachrichten.....	16
4.2.3 Use Cases Profil-Ansicht.....	18
4.2.4 Use Cases Lese-Ansicht	19
4.2.5 Use Cases Registrieren Ansicht	20
4.2.6 Use Cases Profil Einstellungen Ansicht	20
4.2.7 Use Cases Meine Comb Ansicht.....	21
4.3 Benutzerschnittstellen	22
4.3.1 Grundansicht.....	22
4.3.2 Nachrichten.....	22
4.3.3 Filterung/Sortierung von Nachrichten (Formulare).....	23
4.4 Anforderungen an externe Schnittstellen	26
4.5 Anforderungen an die Sicherheit.....	26
5 Technische Spezifikation.....	26
5.1 Strukturierung des Systems	26
5.2 Feinentwurf der Systemkomponenten (-elemente): Java-Teil	26
5.2.1 Datenbank.....	26
5.2.2 Hilfs-Klassen	35
5.2.3 Content Loading.....	38
5.2.4 Content Parsing.....	40
5.2.5 Indexierung und Suche	41
5.2.6 News Grouping.....	46
5.2.7 Validation	47
5.2.8 Translations.....	49
5.2.9 Konfiguration	50
5.2.10 Services	51
5.3 Feinentwurf der Systemkomponenten (-elemente): JavaScript-Teil.....	66
5.3.1 categoryPreference.....	67
5.3.2 category	67
5.3.3 comb	67

5.3.4	errors.....	67
5.3.5	filter.....	67
5.3.6	forms	68
5.3.7	home	68
5.3.8	incomb	68
5.3.9	nav.....	69
5.3.10	news.....	69
5.3.11	profile.....	69
5.3.12	provider.....	69
5.3.13	read	70
5.3.14	resources.....	70
5.3.15	router	70
5.3.16	search.....	70
5.3.17	settings.....	70
5.3.18	Slider	71
5.3.19	Styles	71
5.3.20	tagPreference.....	71
5.3.21	userLocale	71
5.3.22	userSettings.....	71
5.3.23	user	71
5.4	Physikalische Datenbeschreibung (Datenstrukturtypen)	72
5.5	Sicherheit	72
5.6	Anforderungen an die Entwicklungsumgebung.....	72

3 Einleitung

3.1 Inhalt und Zweck des Dokuments

Dieses Dokument zeigt Fachliche Informationen, sowie alle Benutzerbedürfnisse in einer detaillierten Art dar. Ausserdem werden alle Schnittstellen, Teilsysteme und genaue Definitionen der Klassenmodelle aufgezeigt.

3.2 Abkürzungen und Definitionen

Definitionen:

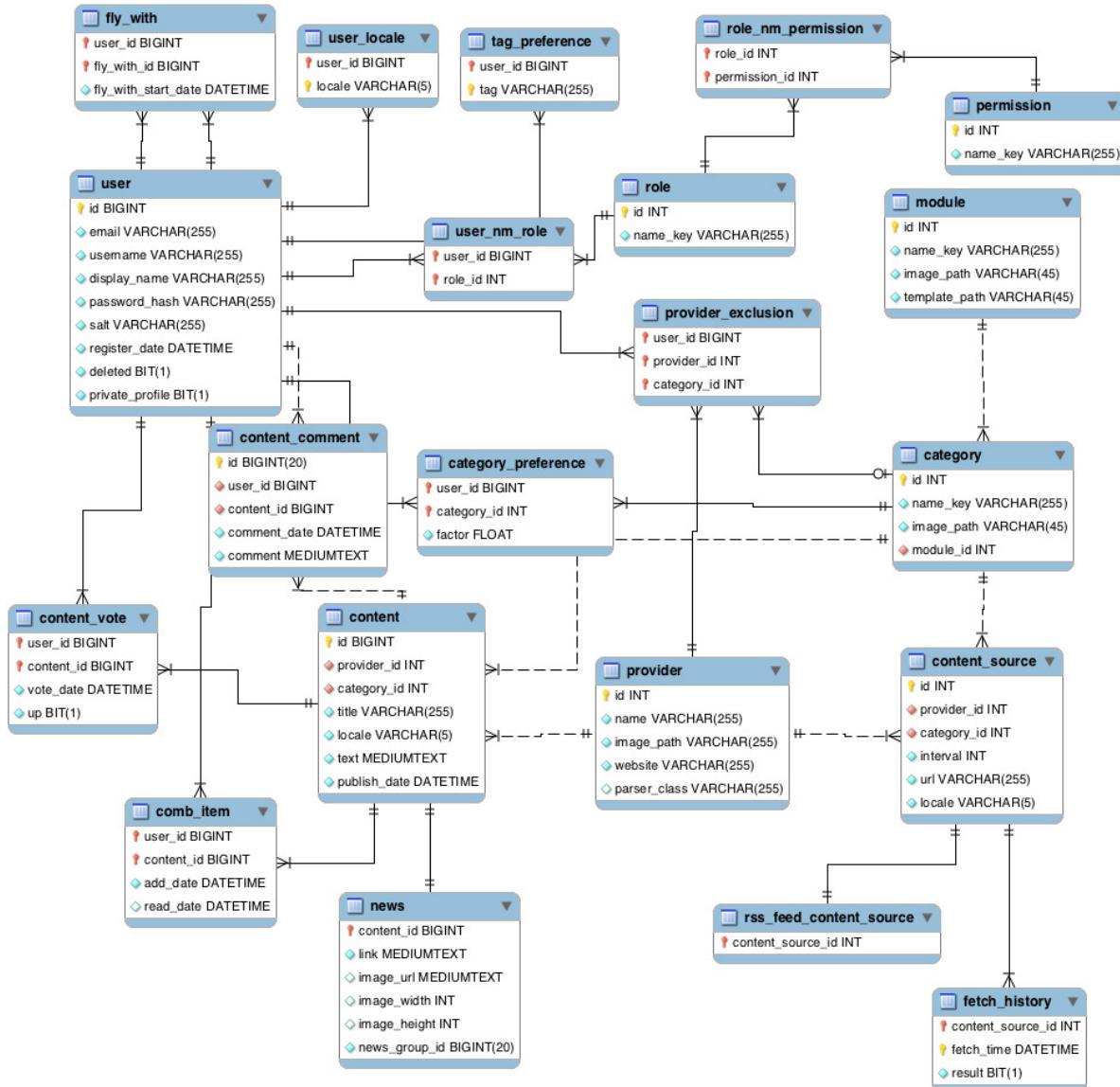
In	Positive Bewertung
Comb	Bienenwabe / Kamm (Negative Bewertung)
FlyWith	Einem anderen Nutzer folgen
WithFlyings	Nutzer die anderen Nutzern folgen
Gravatar	Global verfügbarer Avatar
Bee	Ein Benutzer von InComb
Tag	Ein Begriff der in Nachrichtenartikeln gefunden werden kann.
Icon	Symbol
Commit	In die Versionsverwaltung aufnehmen
Rollback	Änderungen rückgängig machen
jooq	Datenbank Abstraktionsframework
POJO	Plain Old Java Objects – Java Klassen die nur als Datenhalter fungieren
Votes	Bewertungen
JSON	Javascript Object Notation – Datenaustauschformat
Salts	Für Passwortsicherheit
Parser	Einleser und Übersetzer von Daten
Content	Inhalt
Enum	Verbundsdatentyp
Locale	Sprache
api	Schnittstelle für Kommunikation innerhalb von InComb
Scope	ViewModel – Daten und Funktionen vom Controller, die in der View verwendet werden können.
Directive	AngularJS Controller der auf einem HTML Element ausgeführt wird.
Router / Route	AngularJS Der Router verwaltet Routes, Eine Route ist ein Typ von HTML-Seite mit URL.

Abkürzungen:

HW	Hardware
SW	Software
GUI	Graphical User Interface / Grafische Benutzerschnittstelle

4 Fachliche Spezifikation

4.1 Logische Datenbeschreibung (ERD)



4.1.1 Inhalte

Es ist geplant, dass später noch andere Inhalte neben den Nachrichten angezeigt werden. Dies wurde im Datenbankschema berücksichtigt. Pro Inhaltstyp wird ein Eintrag in der Tabelle module erstellt.

4.1.1.1 module

Jedes module hat eine eindeutige ID. Der name_key ist ein Key, der verwendet wird um die Übersetzung laden zu können. Außerdem kann ein Template (template_path) und ein Bildpfad (image_path) angegeben werden, was aktuell aber noch nicht benötigt wird.

Spalte	Datentyp	NULL	Beschreibung
id	INT	Nein	Eindeutige ID des Moduls.
name_key	VARCHAR(255)	Nein	Key um den lokalisierten Namen abzurufen.
image_path	VARCHAR(45)	Nein	Absoluter Pfad zu einem Bild.
template_path	VARCHAR(45)	Nein	Absoluter Pfad zu einem Template.

4.1.1.2 category

Jedes Modul kann mehrere Kategorien haben. Diese werden in der Tabelle category gespeichert und das dazugehörige Modul wird über module_id referenziert. Auch hier kann ein name_key angegeben werden um den sprachabhängigen Text zu laden. Es ist ebenfalls möglich einen Bildpfad (image_path) anzugeben. Dieser wird bei den Nachrichten aber nicht verwendet, kann dann aber für Kategorien bei einem anderen Modul verwendet werden.

Spalte	Datentyp	NULL	Beschreibung
id	INT	Nein	Eindeutige ID des Themenbereichs.
name_key	VARCHAR(255)	Nein	Key um den lokalisierten Namen abzurufen.
image_path	VARCHAR(45)	Nein	Absoluter Pfad zu einem Bild.
module_id FK(module.id)	INT	Nein	ID des Moduls, welches dieser Themenbereich zugeordnet ist.

4.1.1.3 provider

Jeder Nachrichten-Anbieter wird in der Tabelle provider mit seinem Namen (name), einer eindeutigen ID (id), seiner Webseite (website) und einem Pfad zu seinem Logo gespeichert.

Spalte	Datentyp	NULL	Beschreibung
id	INT	Nein	Eindeutige ID des Nachrichten Anbieters.
name	VARCHAR(255)	Nein	Name des Anbieters.
image_path	VARCHAR(255)	Nein	Absoluter Pfad zu einem Logo.
website	VARCHAR(255)	Nein	Webseite des Nachrichten-Anbieters. Mit Protokoll.
parser_class	VARCHAR(255)	Ja	Optionale Java-Klasse, um für diesen Anbieter spezifisch Inhalte einlesen zu können.

4.1.1.4 content_source, rss_feed_content_source

Jede Datenquelle (RSS-Feed) eines Providers wird einer Kategorie zugeordnet. Diese Informationen wird in der Tabelle content_source gespeichert. Jeder Eintrag hat eine eindeutige ID (id), die Sprache, in der die Daten geliefert werden (locale), der Intervall (interval) in Sekunden wann die Inhalte neu geladen werden sollen und eine URL (fetch_url) wo die Daten abgerufen werden können.

Spalte	Datentyp	NULL	Beschreibung
id FK(provider.id)	INT	Nein	Eindeutige ID der Inhaltsquelle.
provider_id FK(provider.id)	INT	Nein	Nachrichten-Anbieter von welchem die Inhalte stammen.
category_id FK(category.id)	INT	Nein	Themenbereich, zu welchem die Inhalte zugeordnet werden sollen.
interval	INT	Nein	Intervall in Sekunden, in welchem neue Inhalte abgerufen werden sollen.
fetch_url	VARCHAR(255)	Nein	URL, von welcher die Inhalte abgerufen werden können.
locale	VARCHAR(5)	Nein	Sprache, in welcher die Inhalte verfasst sind.

Da in Zukunft andere Quellen neben RSS Feeds unterstützt werden sollen, gibt es eine weitere Tabelle rss_feed_content_source, die markiert, dass die Daten als RSS-Feed geladen werden sollen. Diese Tabelle enthält ausser der content_source_id, die auf die content_source Tabelle referenziert keine weiteren Informationen im Moment.

Spalte	Datentyp	NULL	Beschreibung
content_source_id FK(content_source.id)	INT	Nein	Verweis auf die Inhaltsquelle.

4.1.1.5 fetch_history

Nach jedem Abrufen einer content_source wird ein Eintrag in die Tabelle fetch_history erstellt. Diese protokolliert das Datum und die Zeit des Abrufs (fetch_time) und ob der Abruf erfolgreich war (result). Anhand von diesen Einträgen wird dann der Zeitpunkt für den nächsten Abruf bestimmt.

Spalte	Datentyp	NULL	Beschreibung
content_source_id FK(content_source.id)	INT	Nein	Verweis auf die Inhaltsquelle, von welcher Daten geladen wurden.
fetch_time	DATETIME	Nein	Zeitpunkt des Abrufs.
result	BIT(1)	Nein	Wenn gesetzt, dann konnten die Inhalte erfolgreich abgerufen werden.

4.1.1.6 content, news

Die Inhalte, die dann dem Benutzer angezeigt werden, werden in der Tabelle content gespeichert. Jeder Inhalt hat eine eindeutige ID (id), einen Titel (title), einen Text (text), ein Veröffentlichungsdatum (publish_date) und ist einer Sprache (locale), einem Nachrichten-Anbieter (provider_id) und einem Themenbereich (category_id) zugeordnet.

Spalte	Datentyp	NULL	Beschreibung
id FK(provider.id)	BIGINT	Nein	Eindeutige ID des Inhalts.
provider_id FK(provider.id)	INT	Nein	Anbieter, von welchem der Inhalt stammt.
category_id FK(category.id)	INT	Nein	Kategorie, zu welcher der Inhalt zugeordnet ist.
title	VARCHAR(255)	Nein	Titel des Inhalts.
locale	VARCHAR(5)	Nein	Sprache des Inhalts im ISO-Format.
text	MEDIUMTEXT	Nein	Der eigentliche Inhalt.
publish_date	DATETIME	Nein	Veröffentlichungsdatum oder Datum, wann der Inhalt in die Datenbank gespeichert wurde (wenn kein Veröffentlichungsdatum bekannt).

Da in Zukunft neben Nachrichten andere Inhalte gespeichert werden sollen, gibt es für Nachrichten eine weitere Tabelle, die nur Daten beinhaltet, die für Nachrichten Sinn machen. Für andere Inhalte kann dann später auch eine separate Tabelle erstellt werden. Die Einträge referenzieren aber immer einen Eintrag in der content-Tabelle.

In der news-Tabelle kann man noch einen Link zum Artikel (link), einen Pfad zu einem Artikelbild (image_url), dessen Höhe (image_height) und Breite (image_width) und eine Zugehörigkeit zu einer Nachrichten Gruppierung (news_group_id) angeben.

Spalte	Datentyp	NULL	Beschreibung
content_id FK(content.id)	BIGINT	Nein	Referenz zum Haupt-Inhaltseintrag.
link	MEDIUMTEXT	Nein	Kompletter Link zum Artikel auf der Webseite des Anbieters.
image_url	MEDIUMTEXT	Ja	URL zu einem Bild für die Nachricht.
image_width	INT	Ja	Breite des Bilds bei image_url.
image_height	INT	Ja	Höhe des Bilds bei image_url.
news_group_id	BIGINT(20)	Nein	Eine ID zu einer Nachrichten-Gruppe. Wenn zu keiner zugewiesen wird der Wert 0 gespeichert. Wenn die Nachricht noch nicht gruppiert wurde wird der Wert -1 gespeichert. Alle Nachrichten mit derselben news_group_id bilden eine Gruppe.

4.1.1.7 content_vote

Wenn ein Benutzer ein In oder ein Comb für einen Inhalt gibt, wird dies in der Tabelle content_vote gespeichert. Dazu wird der Benutzer (content_id) und der Inhalt (content_id) zusammen mit dem Zeitpunkt (vote_date) und der Information, ob es ein In oder ein Comb war (up) gespeichert.

Spalte	Datentyp	NULL	Beschreibung
<u>user_id</u> FK(user.id)	BIGINT	Nein	Der Benutzer, der ein In oder ein Comb gegeben hat.
<u>content_id</u> FK(content.id)	BIGINT	Nein	Der Inhalt, für welchen ein In oder ein Comb gegeben wurde.
<u>vote_date</u>	DATETIME	Nein	Zeitpunkt, als die Aktion ausgeführt wurde.
<u>up</u>	BIT(1)	Nein	Wenn gesetzt, wurde ein In gegeben, ansonsten ein Comb.

4.1.1.8 content_comment

Jeder Kommentar zu einem Inhalt wird in der Tabelle content_comment gespeichert. Jeder Kommentar hat eine eindeutige ID (id), die automatisch bei jedem Eintrag hochgezählt wird. Außerdem ist jeder Kommentar mit einem Benutzer (user_id) und dem kommentierten Inhalt (content_id) verknüpft. Zu jedem Kommentar wird der Zeitpunkt des Kommentierens (comment_date) und natürlich der eigentliche Kommentar (comment) gespeichert.

Spalte	Datentyp	NULL	Beschreibung
<u>id</u>	BIGINT	Nein	Eindeutige ID des Kommentars.
<u>user_id</u> FK(user.id)	BIGINT	Nein	Der Benutzer, der den Kommentar verfasst hat.
<u>content_id</u> FK(content.id)	BIGINT	Nein	Der Inhalt, der kommentiert wurde.
<u>comment_date</u>	DATETIME	Nein	Zeitpunkt des Kommentierens.
<u>comment</u>	MEDIUMTEXT	Nein	Der eigentliche Kommentartext.

4.1.2 Benutzer

4.1.2.1 user

Für jeden registrierten Benutzer wird ein Eintrag in der user Tabelle erstellt. Es wird dort der Benutzername, welchen eindeutig sein muss, die E-Mail Adresse, der Name, der anderen Benutzern angezeigt werden soll und das Registrierungsdatum gespeichert. Außerdem wird das Passwort mit dem sicheren PBKDF2WithHmacSHA1 Verfahren zusammen mit einem pro Benutzer erstellten Salt verschlüsselt und in der Datenbank gespeichert. Jeder Benutzer hat eine eindeutige id, welche nicht verändert werden kann. Auf der Tabelle gibt es auch eine Spalte private_profile, welche aussagt, ob das Profil des Benutzers öffentlich zugänglich sein soll oder nicht. Es gibt auch eine Spalte deleted, welche einen Benutzer markiert, der sein Konto gelöscht hat. Diese Funktion ist aber noch nicht implementiert (nicht im Pflichtenheft).

Spalte	Datentyp	NULL	Beschreibung
id	BIGINT	Nein	Eindeutige ID des Benutzers.
email	VARCHAR(255)	Nein	E-Mail Adresse des Benutzers um diesen zu kontaktieren.
username	VARCHAR(255)	Nein	Benutzername des Benutzers, um sich anzumelden. Eindeutig in der ganzen Tabelle.
display_name	VARCHAR(255)	Nein	Name, der anderen Benutzern angezeigt wird. Beispielsweise Vor- und Nachname.
password_hash	VARCHAR(255)	Nein	Mit dem Salt verschlüsseltes Passwort.
salt	VARCHAR(255)	Nein	Generiertes Salt für den Benutzer.
register_date	DATETIME	Nein	Zeitpunkt der Registrierung des Benutzers.
deleted	BIT(1)	Nein	Wenn gesetzt gilt der Benutzer als gelöscht.
private_profile	BIT(1)	Nein	Wenn gesetzt, wird das Profil des Benutzers nicht anderen Benutzern angezeigt.

4.1.2.2 role, permission

Ebenfalls noch nicht implementiert, da auch nicht im Pflichtenheft vorhanden, sind Zugriffsbeschränkungen für Benutzer. Jedoch wurde dies schon im Datenbank-Schema berücksichtigt. So können bestimmte Benutzergruppen erstellt werden (zum Beispiel Administratoren, Bezahlkunden), welche dann bestimmte Rechte haben. Die Rechte würden in der Tabelle permission gespeichert und die Benutzergruppen in der Tabelle role. Ein Benutzer kann mehreren Benutzergruppen zugeordnet sein (user_nm_role) und ein Recht kann mehreren Benutzergruppen erteilt werden (role_nm_permission).

4.1.2.3 user_locale

Der Benutzer kann angeben, in welchen Sprachen er Nachrichten lesen will. Dazu gibt es die Tabelle user_locale. Für jede vom Benutzer gewünschte Sprache (locale) wird ein Eintrag erstellt.

Spalte	Datentyp	NULL	Beschreibung
user_id FK(user.id)	BIGINT	Nein	Eindeutige ID des Benutzers.
locale	VARCHAR(5)	Nein	Sprache im ISO-Format, in der Inhalte auch angezeigt werden sollen.

4.1.2.4 tag_preference

Der Benutzer kann Tags angeben. Diese werden in der Tabelle tag_preference gespeichert.

Spalte	Datentyp	NULL	Beschreibung
user_id FK(user.id)	BIGINT	Nein	Eindeutige ID des Benutzers, für welchen das Tag gilt.
tag	VARCHAR(255)	Nein	Tag, für welches sich der Benutzer interessiert.

4.1.2.5 category_preference

Der Benutzer kann pro Themenbereich einen Faktor angeben, wie stark ihn diese Kategorie interessiert. Dieser Wert wird in der Tabelle category_preference gespeichert.

Spalte	Datentyp	NULL	Beschreibung
<u>user_id</u> FK(user.id)	BIGINT	Nein	Eindeutige ID des Benutzers für welchen der Wert gilt.
<u>category_id</u> FK(category.id)	INT	Nein	ID der Kategorie, für welche der Wert gilt.
factor	FLOAT	Nein	Der Wert zwischen 0.0 (nicht interessant) und 1.0 (sehr interessant).

4.1.2.6 provider_exclusion

Eine Funktionalität, die nicht im Pflichtenheft erwähnt wurde und deswegen auch nicht vollständig implementiert wurde, ist die Möglichkeit Nachrichten-Anbieter für bestimmte Kategorien auszuschliessen. So werden dann dem Benutzer keine Nachrichten dieses Anbieters in der jeweiligen Kategorie angezeigt. Mit unserer Datenstruktur lässt sich auch ein kompletter Ausschluss eines Anbieters oder einer Kategorie realisieren.

Spalte	Datentyp	NULL	Beschreibung
<u>user_id</u> FK(user.id)	BIGINT	Nein	Eindeutige ID des Benutzers für welchen der Ausschluss gilt.
<u>provider_id</u> FK(provider.id)	INT	Nein	ID des Nachrichten-Anbieters, der ausgeschlossen wurde.
<u>category_id</u> FK(category.id)	INT	Nein	ID des Themenbereichs, der ausgeschlossen wurde.

4.1.2.7 fly_with

Wenn der Benutzer einem anderen Benutzer folgen will (fly_with) dann wird ein Eintrag in die Tabelle fly_with erstellt. Der Benutzer der verfolgt wird, wird in der Spalte fly_with_id eingetragen. Der Benutzer, der dem anderen Benutzer folgt in der Spalte user_id. Die Spalte fly_with_start_dates speichert das Datum und die Zeit, wann der Eintrag erstellt wurde.

Spalte	Datentyp	NULL	Beschreibung
<u>user_id</u> FK(user.id)	BIGINT	Nein	Eindeutige ID des Benutzers, welcher einem anderen Benutzer folgt.
<u>fly_with_id</u> FK(user.id)	BIGINT	Nein	Eindeutige ID des Benutzers, welcher verfolgt wird.
fly_with_start_date	DATETIME	Nein	Zeitpunkt, als der Benutzer begann dem andern zu folgen.

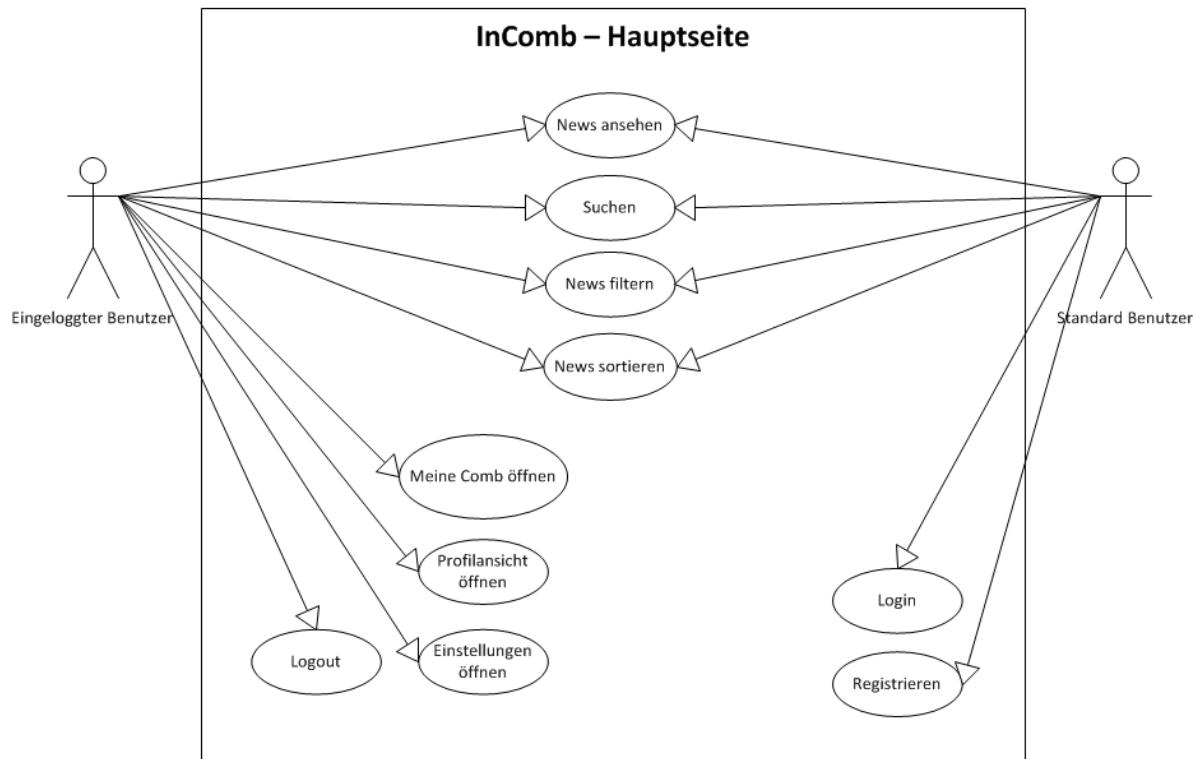
4.1.2.8 comb_item

Wenn der Benutzer einen Inhalt in seine „Comb“ legt (Später lesen), dann wird ein Eintrag in der Tabelle comb_item erstellt. Dabei wird der Benutzer (user_id) und der gespeicherte Inhalt (content_id) zusammen mit dem Datum, als er den Inhalt zur Comb hinzufügte (add_date) und dem Zeitpunkt, als er den Inhalt gelesen hat (read_date), gespeichert.

Spalte	Datentyp	NULL	Beschreibung
<u>user_id</u> FK(user.id)	BIGINT	Nein	Der Benutzer, der den Inhalt in seine Comb gelegt hat.
<u>content_id</u> FK(content.id)	BIGINT	Nein	Der Inhalt, der in die Comb gelegt wurde.
<u>add_date</u>	DATETIME	Nein	Zeitpunkt, als der Inhalt in die Comb gelegt wurde.
<u>read_date</u>	DATETIME	Ja	Zeitpunkt, wann er den Inhalt das erste Mal gelesen hat, nachdem er ihn in die Comb gelegt hat. Wenn null gespeichert wird, dann wurde er noch nicht gelesen.

4.2 Spezifische Benutzeranforderungen

4.2.1 Use Cases Hauptseite



Use Case: Suchen

Der USE CASE Startet wenn der Benutzer in das Suchfenster klickt. Der Nutzer muss für diese Aktion nicht eingeloggt sein. Während der Nutzer am Schreiben ist, werden ihm verschiedene, mögliche Ergebnisse angezeigt, welche auf die Momentane Suchanfrage zutreffen. Zu den möglichen Ergebnissen zählen, Nutzer (Bees), Nachrichtenanbieter (Provider), Themenbereiche (Kategorien) und Nachrichten. Der USE CASE endet, wenn der Benutzer auf ein Suchresultat klickt, oder die Suche mit dem „x“ im Suchfenster beendet.

Use Case: News ansehen

Der USE CASE Startet wenn der Benutzer die Hauptseite von InComb aufruft. Der Nutzer muss für diese Aktion nicht eingeloggt sein. Es wird eine Gewisse, noch nicht definierte Anzahl von News vorgeladen. Die Nachrichten werden nach einer definierten Sortier- und Filterreihenfolge dargestellt, Ähnliche Nachrichten werden gruppiert. Standardmässig werden die Nachrichten bei einem nicht eingeloggten Benutzer nach Veröffentlichungsdatum sortiert. Bei einem eingeloggten Benutzer, werden die Nachrichten nach Relevanz sortiert. Die Relevanz wird aus Kategorie Vorlieben, benutzerspezifischen Tags, Kommentaren, Ins und Combs von Flywiths, und Veröffentlichungsdatum errechnet. Mit dem Scrollen nach unten werden weitere Nachrichten nachgeladen bis keine weiteren vorhanden sind. Der USE CASE endet, wenn der Benutzer auf eine andere Seite navigiert oder die Webseite schliesst.

Use Case: News Filtern

Der USE CASE Startet wenn der Benutzer auf das Filter Icon klickt. Der Benutzer muss nicht eingeloggt sein. Der nicht eingeloggte Benutzer kann nach Newsanbieter, Thema, Alter (Veröffentlichungsdatum der Nachricht) filtern. Eingeloggte Benutzer können weiter wählen, ob die Filterresultate personalisiert werden sollen. Der USE CASE endet, wenn der Benutzer den Filter wieder schliesst.

Use Case: Sortieren

Der USE CASE Startet wenn der Benutzer auf das Filter Icon klickt. Der Benutzer muss nicht eingeloggt sein. Es erscheint ein Drop down Auswahl Menü unter dem Filter. Der Benutzer kann zwischen mehreren Sortierreihenfolgen wählen. Zum Beispiel Titel, Datum, Relevanz, Kategorie etc. Der Benutzer kann mehre Sortierungen nacheinander durchführen, ohne dass der Filter wieder zuklappt. Der USE CASE endet, wenn der Benutzer den Filter wieder schliesst.

Use Case: Registrieren

Der USE CASE Startet wenn der Benutzer auf den dafür vorgesehenen Registrieren-Button klickt. Für diese Aktion darf der Benutzer nicht eingeloggt sein. Durch das Drücken des Knopfes wird der Benutzer automatisch auf die Registrierungs-Seite weitergeleitet. Der USE CASE endet nachdem sich der Benutzer auf der Registrierungs-Seite befindet.

Use Case: Login

Der USE CASE Startet wenn der Benutzer auf den dafür vorgesehenen Login Button klickt. Für diese Aktion muss der Benutzer ausgeloggt sein. Der Button darf nicht vorhanden sein, wenn der Benutzer nicht eingeloggt ist. Durch Drücken des Knopfes wird der Benutzer zu einer Login Seite weitergeleitet, wo er aufgefordert wird Benutzernamen und Passwort einzugeben. Stimmen die Login Daten überein, so wird der Benutzer auf die Startseite weitergeleitet. Andernfalls wird eine Fehlermeldung angezeigt, welche eine Fehlermeldung signalisiert. Der USE CASE endet, wenn der Benutzer sich erfolgreich eingeloggt hat, oder aber die InComb verlässt.

Use Case: Logout / Abmelden

Der USE CASE Startet wenn der Benutzer auf den dafür vorgesehenen Logout / Abmelden Button klickt. Für diese Aktion muss der Benutzer eingeloggt sein. Durch Drücken des Knopfes wird der Benutzer automatisch abgemeldet und auf die Login Seite weitergeleitet. Der USE CASE endet nachdem sich der Benutzer auf der Login Seite befindet.

Use Case: Einstellungen öffnen

Der USE CASE Startet wenn der Benutzer auf den dafür vorgesehenen Einstellungen Button klickt. Der Benutzer muss eingeloggt sein, um dies tun zu können. Durch Klicken wird der Benutzer auf die Einstellungen Seite weitergeleitet. Der USE CASE endet, wenn der Benutzer sich auf der Einstellungen Seite befindet.

Use Case: Profilansicht öffnen

Der USE CASE Startet wenn der Benutzer auf den dafür vorgesehenen Profilansicht Button klickt, welcher mit dem Displaynamen des Benutzers versehen ist. Der Benutzer muss eingeloggt sein, um

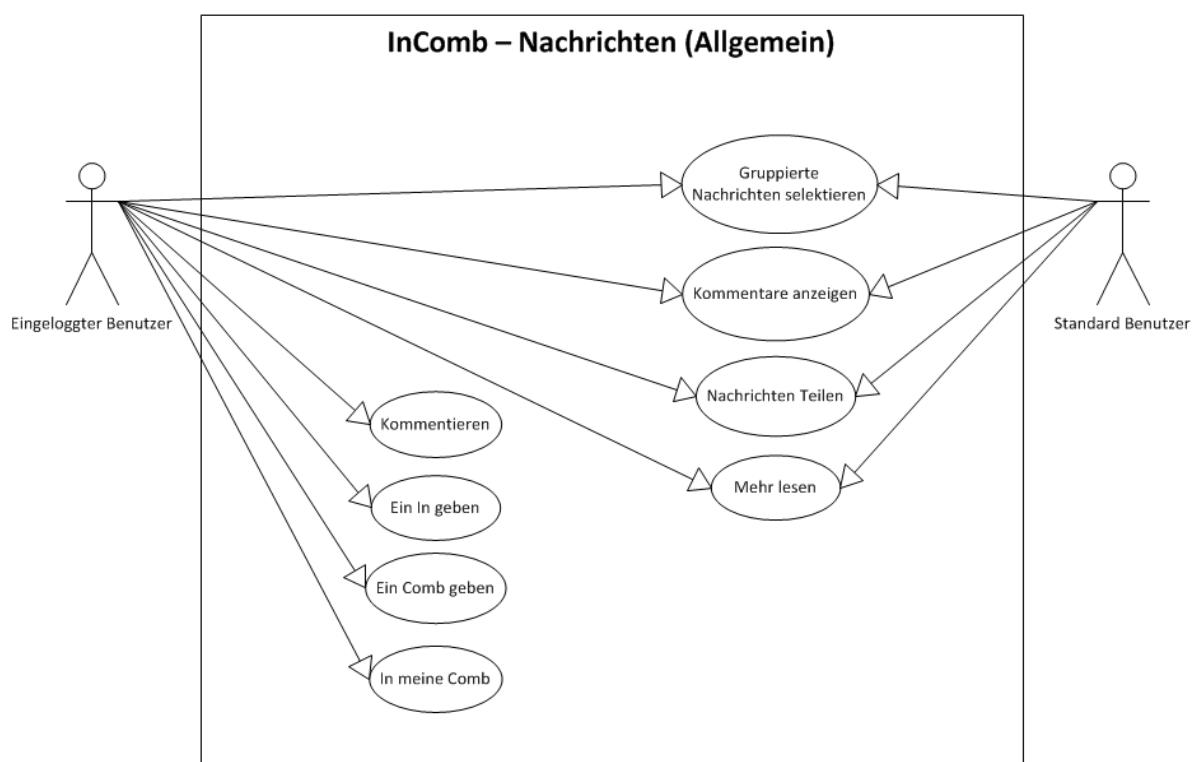
dies tun zu können. Durch Klicken wird der Benutzer auf seine Profilansicht-Seite weitergeleitet. Der USE CASE endet, wenn der Benutzer sich auf seiner Profilansicht befindet.

Use Case: Meine Comb öffnen

Der USE CASE Startet wenn der Benutzer auf den dafür vorgesehenen Meine Comb Button klickt. Der Benutzer muss eingeloggt sein, um dies tun zu können. Durch Klicken wird der Benutzer auf die Seite seiner Comb weitergeleitet. Der USE CASE endet, wenn der Benutzer sich auf der Comb Ansicht befindet.

4.2.2 Use Cases Nachrichten

Alle folgenden Use Cases beziehen sich auf die Interaktion mit einem Nachrichten Element. Es ist egal, wo ein Nachrichten Element angezeigt wird. Alle Aktionen müssen überall verfügbar sein. Ausgeschlossen ist jedoch die Profilansicht.



Use Case: Gruppierte Nachrichten selektieren

Der USE CASE startet wenn der Benutzer einen Gruppierten Nachrichtenblock vorfindet und einen Artikel anklickt. Der Benutzer muss für diese Aktion nicht eingeloggt sein. Der Benutzer kann auf eine Nachricht in der Gruppe klicken, sodass die Nachricht hervorgehoben und in normaler Größe dargestellt wird. Nun soll er alle Möglichkeiten zur Verfügung haben, mit dem Artikel zu interagieren. Der USE CASE endet sobald der Benutzer eine neue Nachricht selektiert hat. Dieser USE CASE kann selbstverständlich mehrmals ausgeführt werden.

Use Case: Kommentare anzeigen

Der USE CASE startet wenn der Benutzer auf den dafür vorgesehenen Kommentare Button klickt. Der Benutzer muss für diese Aktion nicht eingeloggt sein. Jeder Benutzer darf jeden Kommentar ansehen. Der USE CASE endet, wenn die Kommentare wieder über den Button geschlossen werden oder der Benutzer InComb verlässt.

Use Case: Kommentieren

Der USE CASE Startet wenn der Benutzer auf den dafür vorgesehenen „Add my honey“ Button klickt. Der Benutzer muss für diese Aktion eingeloggt sein und die Kommentare müssen angezeigt werden. Der Benutzer kann Text in die dafür vorgesehene Textbox eingeben. Mit dem Knopf „add my honey“ wird der Beitrag veröffentlicht. Es wird ein neuer Kommentar mit einer relativen Zeitanzeige, dem Benutzernamen und dem Kommentar selbst hinzugefügt. Zudem wird der Zähler der Kommentare um eins hochgezählt. Es steht dem Benutzer frei, weitere Kommentare zu erfassen. Der USE CASE endet, wenn die Kommentare wieder über den Kommentare Button geschlossen werden oder der Benutzer InComb verlässt.

Use Case: Ein In geben

Der USE CASE Startet wenn der Benutzer auf den dafür vorgesehenen In Button klickt. Der Benutzer muss für diese Aktion eingeloggt sein. Durch das Klicken des Buttons wird der In-Zähler der Nachricht um eins erhöht. Zudem wird die Nachricht zu dem Benutzerprofil auf der In Sektion sichtbar. Der USE CASE endet sobald der Knopf gedrückt und die Daten aktualisiert wurden.

Use Case: Ein Comb geben

Der USE CASE Startet wenn der Benutzer auf den dafür vorgesehenen Comb Button klickt. Der Benutzer muss für diese Aktion eingeloggt sein. Durch das Klicken des Buttons wird der In Zähler der Nachricht um eins erhöht. Zudem wird die Nachricht zu dem Benutzerprofil auf der Comb Sektion sichtbar. Der USE CASE endet sobald der Knopf gedrückt und die Daten aktualisiert wurden.

Use Case: Nachrichten Teilen

Der USE CASE Startet wenn der Benutzer auf den dafür vorgesehenen Teilen Button klickt. Dies kann der Twitter tweet, Facebook like oder der Google plus Button sein. Der Benutzer muss für diese Aktion nicht eingeloggt sein. Durch das Klicken des Buttons wird der Benutzer, mit dem entsprechenden Link um die Nachricht zu teilen, zu der entsprechenden Social Plattform weitergeleitet. Es ist wichtig, dass die Seite in einem neuen Tab oder Fenster aufgemacht wird. Der USE CASE endet sobald der Knopf gedrückt wurde und ein neues Tab mit der Social Plattform aufgegangen ist.

Use Case: In meine Comb

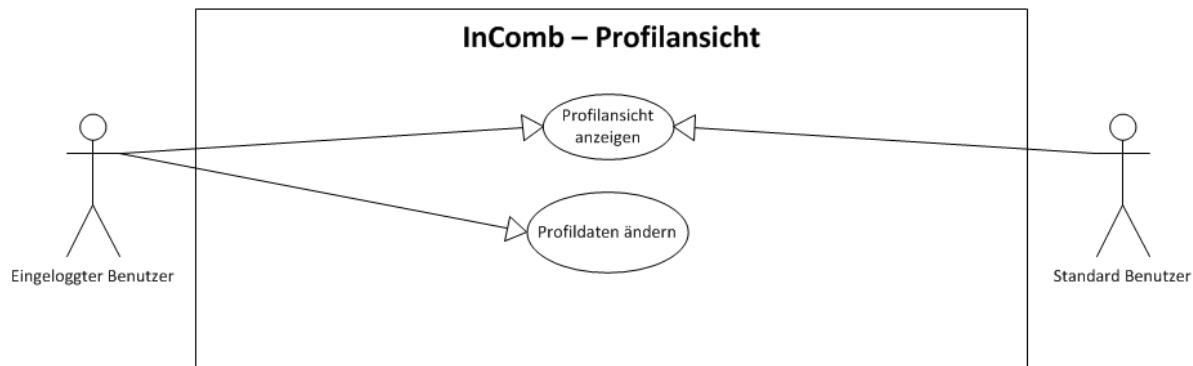
Der USE CASE Startet wenn der Benutzer auf den dafür vorgesehenen „In meine Comb“ Button klickt. Der Benutzer muss für diese Aktion eingeloggt sein. Durch Klicken des Buttons wird der Nachrichten Artikel in die Comb des eingeloggten Benutzers gelegt. Der Button wechselt das Icon, welches symbolisieren soll, den Artikel wieder zu entfernen. Durch erneutes Drücken wird der Artikel wieder aus der Comb des Benutzers entfernt. Der USE CASE endet sobald der Knopf gedrückt wurde und alle Daten Aktualisiert sind.

Use Case: Mehr lesen

Der USE CASE startet wenn der Benutzer auf den Mehr lesen Link klickt. Der Benutzer muss dazu nicht eingeloggt sein. Durch Klicken des Links wird der Benutzer auf die Lese-Ansicht weitergeleitet. Der USE CASE endet, wenn die Lese-Ansicht geladen ist.

4.2.3 Use Cases Profil-Ansicht

Alle folgenden Use Cases beziehen sich auf die Profilansicht eines Benutzers. Ein Profil kann auf Privat gestellt werden. Ist ein Profil auf privat, hat kein Benutzer, ausser dem Besitzer selbst, Zugriff auf das Profil - Sprich die Seite wird als Privat angezeigt.

**Use Case: Profilansicht anzeigen**

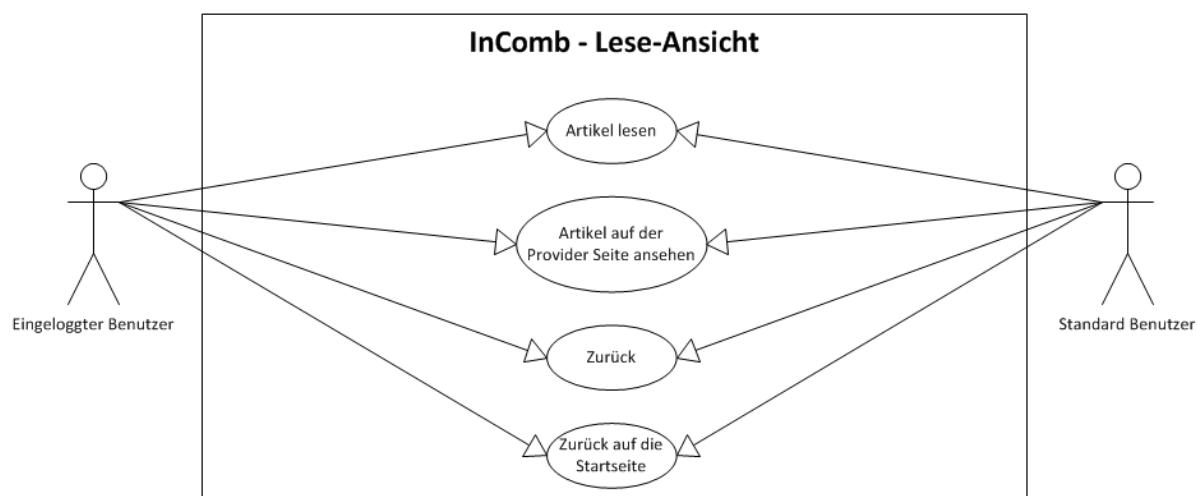
Der USE CASE startet wenn der Benutzer auf ein Profil navigiert. Dies kann das eigene sein, oder auch das eines anderen Benutzers. Der Benutzer muss dazu nicht eingeloggt sein, kann dann aber auch sein Profil nicht modifizieren. Es werden Informationen, Bees denen der Benutzer folgt und welche dem Benutzer folgen, Ins, Combs und Kommentare des entsprechenden Nutzers angezeigt. Ist das Profil jedoch Privat, wird dies auf der Seite vermerkt und keines der genannten Dinge ist sichtbar. Das Profilbild des Benutzers wird anhand der E-Mail Adresse mit Gravatar verknüpft. Besteht keine Gravatar Account zu der E-Mail Adresse, so wird ein Standard User Bild angezeigt. Die Bees werden in einer Liste dargestellt. Durch das Klicken auf ein Bee wird man direkt auf dessen Profil weitergeleitet. Die Ins und Combs werden durch eine abgespeckte Version eines Nachrichten Artikels dargestellt. Durch Klicken auf ein Element wird der Benutzer zu dem entsprechenden Artikel weitergeleitet. Die Kommentare werden in einer weiteren Liste dargestellt. Auch hier kann man durch Klicken wieder auf den entsprechenden Kommentar navigieren. Im Allgemeinen ist es so, dass die verschiedenen Bees, Ins, Combs und Kommentare nicht alle auf einmal angezeigt werden und somit beschränkt sind. Jedoch können mit dem Scrollen auf der entsprechenden Liste weitere Elemente geladen werden. Der USE CASE endet, wenn von der Profilansicht wegnavigiert wird, oder der InComb geschlossen wird.

Use Case: Profildaten ändern

Der USE CASE Startet wenn der Benutzer auf den dafür vorgesehenen Bearbeiten Button klickt. Der Benutzer muss für diese Aktion eingeloggt sein. Auch muss es sich um das eigene Profil handeln. Es wird geprüft, ob das Passwort mit dem wiederholten Passwort übereinstimmen. Danach wird eine Anfrage an den Server gesendet und es wird geprüft, dass die E-Mail Adresse und Benutzernamen noch nicht im System vorhanden sind. Ist die Validation ohne Probleme vonstattengegangen, so wird der Benutzer auf die Startseite weitergeleitet. Ansonsten wird dem Benutzer eine Fehlermeldung eingeblendet welche ihn auf einen Fehler aufmerksam machen soll. Der Benutzer kann hier auch einstellen, ob sein Profil privat sein soll oder nicht. Der USE CASE endet, wenn der Benutzer mit dem speichern Button betätigt und alle Daten erfolgreich angepasst wurden. Alternativ endet der Use Case auch, wenn die Bearbeiten Ansicht wieder geschlossen wird.

4.2.4 Use Cases Lese-Ansicht

Alle folgenden Use Cases beziehen sich auf die Leseansicht. Für alle Aktionen auf dieser Ansicht muss der Benutzer nicht angemeldet sein. Falls es eine Ausnahme geben sollte wird dies explizit erwähnt. Die in der Leseliste (Nicht die Comb) dargestellten Artikel verfügen über die normalen Eigenschaften. Für gewisse Aktionen auf dem Nachrichten Artikel muss der Benutzer jedoch eingeloggt sein. Es können alle Aktionen durchgeführt werden, welche normalerweise verfügbar sind.



Use Case: Artikel lesen

Der USE CASE startet wenn der Benutzer auf den Mehr lesen Link klickt. Durch klicken des Links wird der Benutzer auf die Lese-Ansicht weitergeleitet. Der ausgewählte Artikel wird in einem Hauptfenster dargestellt. Auf der Seite wird eine Navigation eingeblendet, in welcher weitere Artikel ausgewählt werden können, Der USE CASE endet, wenn der Benutzer die Leseansicht verlässt oder InComb schliesst.

Use Case: Artikel auf der Provider Seite ansehen

Der USE CASE startet wenn der Benutzer auf das Icon des Providers klickt. Durch klicken des Icons wird der Benutzer auf denselben Artikel zum Provider weitergeleitet. Der USE CASE endet, wenn sich der Benutzer auf der Seite des Providers befindet. Falls der gewünscht Artikel beim Provider nicht existiert gilt der USE CASE jedoch auch als abgeschlossen.

Use Case: Zurück

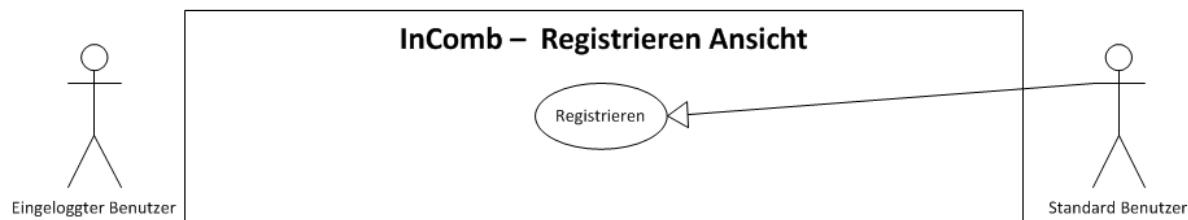
Der USE CASE startet wenn der Benutzer auf das Zurück Icon klickt. Durch das zurück klicken wird der Benutzer auf die Letzte besuchte Seite zurück geleitet. Der USE CASE endet sobald die vorherige Seite geladen ist.

Use Case: Zurück auf die Startseite

Der USE CASE startet wenn der Benutzer auf das InComb Logo klickt. Der Benutzer wird dann auf die Startseite weitergeleitet. Der USE CASE endet, wenn sich der Benutzer auf der Startseite befindet.

4.2.5 Use Cases Registrieren Ansicht

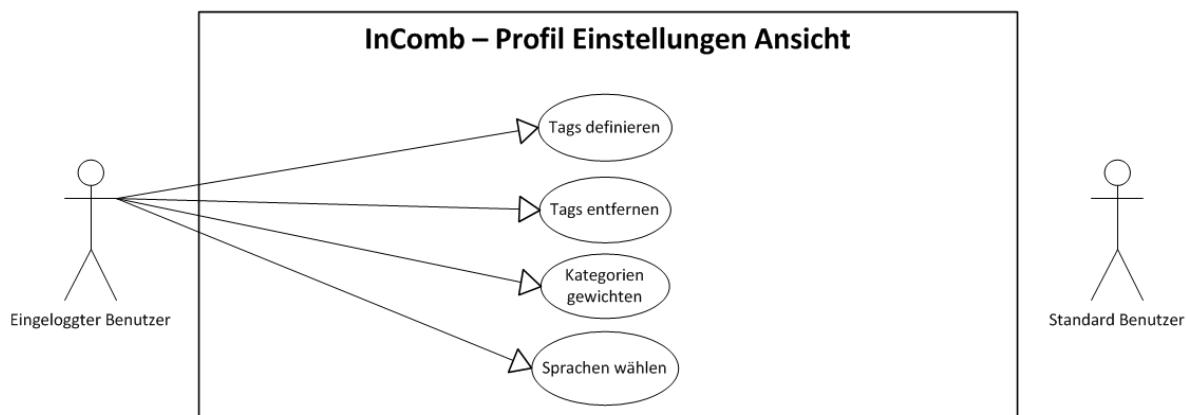
Für den Folgenden Use Case muss sich der Benutzer auf der Registrierungsseite befinden.

**Use Case: Registrieren**

Der USE CASE Startet wenn sich der Benutzer auf der Registrieren Seite befindet und auf den Registrieren Button klickt. Es wird geprüft, ob das Passwort mit dem wiederholten Passwort übereinstimmen. Danach wird eine Anfrage an den Server gesendet und es wird geprüft, dass die E-Mail Adresse und Benutzernamen noch nicht im System vorhanden sind. Ist die Validation ohne Probleme vonstattengegangen, so wird der Benutzer auf die Startseite weitergeleitet. Ansonsten wird dem Benutzer eine Fehlermeldung eingeblendet welche ihn auf einen Fehler aufmerksam machen soll. Der USE CASE endet, wenn die Registration erfolgreich ist und der Benutzer auf die Startseite weitergeleitet wurde, oder dem Benutzer eine Fehlermeldung einen Fehler signalisiert.

4.2.6 Use Cases Profil Einstellungen Ansicht

Für die folgenden Use Case muss sich der Benutzer auf der Einstellungen Ansicht / Seite befinden. Um diese Ansicht überhaupt sehen zu können muss der Benutzer zwingend eingeloggt sein. Aus diesem Grund werden wir es auch nicht mehr explizit erwähnen. Alle getätigten Aktionen werden sofort an den Server übermittelt – Es ist kein speichern Button von Nöten.

**Use Case: Tags definieren**

Der USE CASE startet wenn der Benutzer in das dafür vorgesehene Feld Text eingibt und die Eingabe mit Enter Bestätigt. Die Eingabe wird verarbeitet und an den Server gesendet. Ab nun sollen alle Nachrichten welche unter diesem Tag gefunden werden eine grössere Relevanz erhalten. Der USE CASE endet, wenn der Tag erfolgreich hinzugefügt wurde.

Use Case: Tags entfernen

Der USE CASE startet wenn der Benutzer auf das Löschen Icon des Tags klickt. Das Tag soll aus dem GUI entfernt werden und auf der Datenbank gelöscht werden. Ab nun sollen die durch das Tag höheren Kategorisierten Artikel eine tiefere Relevanz erhalten als vorhin. Der USE CASE endet, wenn das Tag aus der Datenbank gelöscht und das GUI aktualisiert wurde.

Use Case: Kategorien gewichten

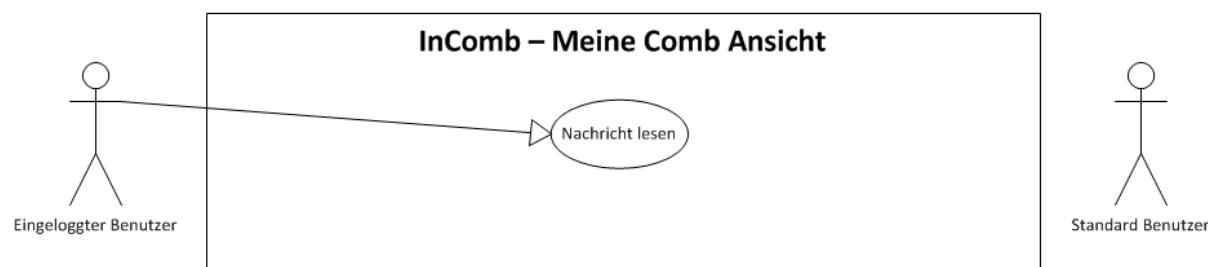
Der USE CASE startet wenn der Benutzer auf den Gewichtung-Schieber der Kategorien klickt / zieht. Durch das Schieben der Regler wird die Relevanz für die jeweilige Kategorie geändert. Die Anzahl der Artikeln pro Kategorie wird Prozentual anhand diesen Einstellungen errechnet. Der USE CASE endet, wenn sich der Benutzer die Konfiguration der Kategorien abgeschlossen hat.

Use Case: Sprachen wählen

Der USE CASE Startet wenn der Benutzer seine gesprochenen Sprachen verwalten will. Der Benutzer kann in einer Liste aller Angeboteten Sprachen auswählen in welcher Sprache er die Nachrichten erhalten wird. Alle nicht angekreuzten Sprache werden dann bei der Anzeige der Artikeln nicht berücksichtigt. Der USE CASE endet, wenn der Benutzer die Konfiguration der Sprachen abgeschlossen hat.

4.2.7 Use Cases Meine Comb Ansicht

Auf der Meine Comb Ansicht kann der Benutzer alle Artikel sehen welche er für das später lesen aufbewahrt hat. Um auf diese Ansicht zu gelangen muss der Benutzer eingeloggt sein. In den folgenden Use Cases wird dies deshalb nicht mehr explizit erwähnt. Alle Artikel Elemente werden normal dargestellt. Alle Aktionen sind verfügbar.



Use Case: Nachricht lesen

Der USE CASE Startet wenn der Benutzer einen Artikel lesen will und auf den „Mehr lesen“ Link klickt. Der Link wird dann in der Lese Ansicht geöffnet und der Link wird aus der Liste entfernt. Der USE CASE endet, wenn sich der Benutzer auf der Leseansicht befindet.

4.3 Benutzerschnittstellen

Die Entwürfe sind aus Platzgründen auf den nächsten Seiten zu finden.

4.3.1 Grundansicht

Der Aufbau der Webseite ist grundsätzlich gleich. Oben links befindet sich das InComb-Logo, über welches man mit einem Klick zurück auf die Startseite gelangt. Oben rechts befindet sich das Suchfeld. Sobald Text eingegeben wird, wird die Suche gestartet und es klappert ein Overlay mit den kategorisierten Resultaten (Nachrichten, Benutzer, Themenbereiche, Anbieter) auf.

Es besteht die Möglichkeit einen Text oder ein Bild über dem eigentlichen Inhalt anzeigen zu können. Beispielsweise für einen Willkommenstext oder das Logo eines Nachrichten-Anbieters.

Auf den meisten Seiten (ausser Login, Registrierung, Lese-Ansicht, Profil) befindet sich links die Navigation. Sofern die Browser-Höhe genügend hoch ist, wird diese beim Scrollen dort fixiert. Zuerst befindet sich der Navigationspunkt für die Startseite. Gleich anschliessend werden die Themenbereiche aufgelistet. Durch einen Klick auf einen Bereich gelangt man auf dessen Nachrichten-Seite.

Wenn der Benutzer sich noch nicht angemeldet hat, befindet sich in der Navigation unten je ein Link zur Anmelde- und Registrierungs-Seite. Wenn der Benutzer angemeldet ist, wird dort sein Profil, die Einstellungen, seine Comb und die Abmelde-Seite verlinkt.

Jede Seite hat einen Seitentitel. Dieser wird über dem Inhalt im teilweise transparenten Bereich angezeigt. Danach folgen zum Beispiel die Nachrichten. Rechts vom Titel besteht je nach Seite die Möglichkeit einen Button rechts vom Titel anzuzeigen. Beispielsweise wird für die Filterung und Sortierung dort ein Button angezeigt, um die Filter- und Sortieroptionen ein- und auszuklappen.

4.3.2 Nachrichten

Es gibt zwei Darstellungsarten für Nachrichten. Wenn das Artikelbild breiter als 300px ist, wird dieses gross über dem Artikel präsentiert. Der Titel wird dann über das Bild gelegt. Ansonsten wird das Bild links vom Titel und vom Text angezeigt. Über dem Titel wird jeweils noch die Kategorie und der Anbieter angezeigt und auf dessen Seiten verlinkt.

Der Artikeltext wird ohne HTML-Tags auf dem weissen Hintergrund angezeigt. Breaks (
) werden behalten, um den Text zu gliedern. Nach dem Text wird ein Link mit dem Titel „Mehr lesen“ angezeigt, der auf die Lese-Ansicht weiterleitet.

Über jeder Nachricht wird ein Balken in der Farbe der Kategorie, in welcher der Artikel veröffentlicht wurde angezeigt. Dadurch kann der Benutzer den Themenbereich der Nachricht beim Überfliegen schnell erkennen.

Sobald man mit der Maus über den Artikel fährt werden die sozialen Interaktionsmöglichkeiten eingebettet. Dann wird die Anzahl Ins und Combs angezeigt und durch einen Klick auf das jeweilige Icon kann der Benutzer ein solches vergeben. Ebenfalls lässt sich der Artikel in die Comb des Benutzers legen. Diese Aktionen erfordern jedoch, dass sich der Benutzer angemeldet hat. Jeder

Anwender kann den Artikel auf den sozialen Netzwerken Facebook, Twitter und Google+ teilen, indem er auf den jeweiligen Button klickt. Ausserdem sieht der Benutzer die Menge an Kommentaren und kann diese durch einen Klick auf den Kommentar-Button anzeigen lassen. Wenn der Benutzer sich bereits angemeldet hat, kann er auch einen neuen erfassen. Das Eingabefeld mit dem Absende-Button wird am Ende der existierenden Kommentare angezeigt.

4.3.3 Filterung/Sortierung von Nachrichten (Formulare)

Auf der Startseite, bei einer Kategorie-Seite oder Anbieter-Seite besteht die Möglichkeit die angezeigten Nachrichten zu sortieren und zu gruppieren (siehe Pflichtenheft). Diese Optionen werden standardmässig dem Benutzer noch nicht angezeigt, um den Fokus auf die Nachrichten zu legen. Durch einen Klick auf den Button rechts des Seitentitels werden diese aber eingeblendet. Ein Absende-Button ist nicht nötig, da die Änderungen sofort übernommen werden.

Wie die Eingabefelder allgemein angezeigt werden sollen, sieht man beim Formular für die Filterung und die Sortierung. Einzelne Formularfelder können gruppiert werden und erhalten einen weissen Hintergrund.

Nachrichten-Ansicht

Hey Josef!

Es gibt neue Nachrichten für dich. Wir wünschen dir viel Spaß beim Lesen bei InComb.
Du willst noch mehr? Dann schau dir doch unsere [Upgrade-Möglichkeiten](#) an.

[Home](#)[Schweiz](#)[Ausland](#)[Sport](#)[Wirtschaft](#)[Digital](#)[Gesundheit](#)[Josef](#)[Einstellungen](#)[Logout](#)

Schweiz · Nachrichten



Watson, Weiland

Rückschlag für die touristische Raumfahrt
«Spaceship Two» abgestürzt

Das private Raumflugzeug «SpaceShipTwo» ist bei einem Testflug im US-Bundesstaat Kalifornien abgestürzt. Das teilte das Unternehmen Virgin Galactic am Freitag mit. Laut dem amerikanischen Sender ABC News ist ein Pilot beim Crash ums Leben gekommen. Der zweite Pilot wurde schwer verletzt.

Der TV-Sender CNN zeigte Bilder mehrerer Wrackteile in der Mojave-Wüste. Das «SpaceShipTwo» wurde den Informationen zufolge zunächst von einem Trägerflugzeug in die Höhe getragen und dann ausgeklinkt. Das Trägerflugzeug sei sicher gelandet, so Virgin Galactic. [Mehr Lesen](#) →



20 Minuten · Ausland

Weltraumtourismus bereits vor dem Aus?

Mit dem privaten Raumflugzeug hätten dereinst Privatpersonen ins All fliegen sollen. Doch der Absturz von «SpaceShipTwo» könnte dem Weltraumtourismus langfristig schaden. Es war der Traum von Prominenten wie Stephen Hawking und Justin Bieber, aber auch von wohlhabenden Privatpersonen. [Mehr Lesen](#) →



The Verge · Digital

Virgin Galactic's SpaceShipTwo crashes during test flight

Today one test pilot died when Virgin Galactic's SpaceShipTwo crashed in the Mojave desert. A second pilot was evacuated to a hospital. It's not clear why the crash occurred. SpaceShipTwo had been undergoing testing ahead of commercial flights. The spacecraft launches from the belly of the larger WhiteKnightTwo, once that cargo aircraft has reached cruising altitude similar to a normal airplane. Once SpaceShipTwo separates, it functions as a glider with a rocket motor. WhiteKnightTwo's takeoff occurred at 9:20 a.m. local time; SpaceShipTwo was released at 10:10. At 10:12, the ground crew became aware of an "inflight anomaly." [Mehr Lesen](#) →



Watson, Weiland

Rückschlag für die touristische Raumfahrt
«Spaceship Two» abgestürzt

Das private Raumflugzeug «SpaceShipTwo» ist bei einem Testflug im US-Bundesstaat Kalifornien abgestürzt. Das teilte das Unternehmen Virgin Galactic am Freitag mit. Laut dem amerikanischen Sender ABC News ist ein Pilot beim Crash ums Leben gekommen. Der zweite Pilot wurde schwer verletzt.

Der TV-Sender CNN zeigte Bilder mehrerer Wrackteile in der Mojave-Wüste. Das

Filter/Sortierung (Formulare)

 Nach Nachrichten, Bees, ... suchen

Hey Josef!

Es gibt neue Nachrichten für dich. Wir wünschen dir viel Spass beim Lesen bei InComb.
Du willst noch mehr? Dann schau dir doch unsere [Upgrade-Möglichkeiten](#) an.

Sortierung Nachrichten-Anbieter ▾

Anbieter Tages-Anzeiger ▾

Thema Alle Themen ▾

Zeitraum Benutzerdefiniert ▾ 01.01.2014 31.01.2014

Quellen ✓ Von Freunden ✓ Personalisiert

Schweiz · Nachrichten



Watson · Ausland in
Rückschlag für die touristische Raumfahrt «Spaceship Two» abgestürzt

Suche

 Schwei

Kategorien

 Schweiz

Nachrichten

Das Schöne an der Schweiz

Frankfurter Allgemeine Zeitung · Schweiz

Lore ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla.



4.4 Anforderungen an externe Schnittstellen

Die grösste Schnittstelle der Anwendung sind die RSS-Feeds. Damit die Nachrichten korrekt angezeigt werden können, müssen diese die Daten im RSS-Standard zurückliefern und auch erreichbar sein. Alle Schnittstellen wurden bereits im Konzept beschrieben.

4.5 Anforderungen an die Sicherheit

Folgende Punkte müssen bzgl. der Sicherheit bei der Realisierung beachtet werden.

- Keine SQL-Injection Möglichkeiten (Manipulation von SQL-Abfragen durch Benutzer-Eingaben)
- Benutzer-Passwörter sicher speichern
- Cross-Site-Scripting verunmöglichen (Ausführen von Code in der Webseite)

5 Technische Spezifikation

5.1 Strukturierung des Systems

Der Java-Teil wurde in Packages aufgeteilt. Das Hauptpackage ist com.incomb.server. Darin können nun für die einzelnen Teile eigene Packages erstellt werden.

com.incomb.server.services

In diesem und dessen Subpackages befinden sich die Controller, die die REST-Schnittstelle bilden.

com.incomb.server.model

In diesem Package sind die Datenhalter-Klassen vorhanden. Die meisten Strukturen existieren auch in der Datenbank und werden mit jooq in Objekte oder Datenbankeinträge umgewandelt. Jooq generiert dazu Klassen in den Packages com.incomb.server.model.tables, com.incomb.server.model.records und com.incomb.server.model.dao.internal.

5.2 Feinentwurf der Systemkomponenten (-elemente): Java-Teil

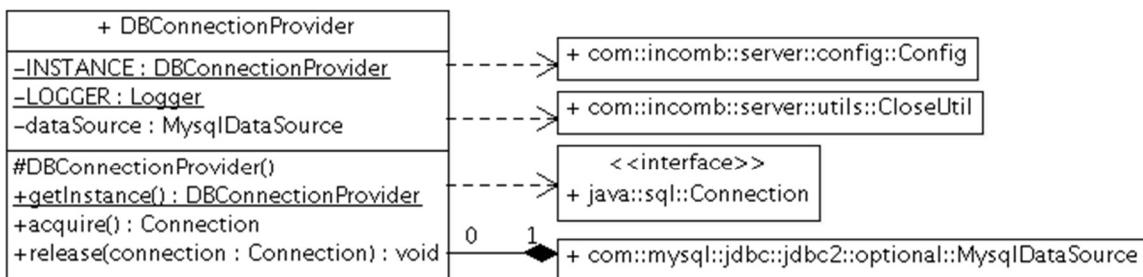
Die UML-Klassendiagramme zeigen nur diejenigen Klassen und Interfaces mit allen Methoden und Membervariablen an, wen diese für den jeweiligen Bereich relevant sind. Klassen und Interfaces, die nur Abhängigkeiten anzeigen sollen, werden nur mit dem Namen und dem Package angezeigt und werden dann in ihrem Bereich komplett angezeigt. So können Redundanzen vermieden werden.

Das System wurde in die folgenden Teile aufgeteilt:

5.2.1 Datenbank

Package: com.incomb.server.db

Über den DBConnectionProvider können Connections zur Datenbank abgerufen und wieder zurückgegeben werden. Intern werden diese über eine MysqDataSource bezogen. Standardmäßig haben die Connections autocommit auf false gestellt, was bedeutet dass man manuell einen Commit oder ein Rollback durchführen muss. So können im Fehlerfall alle Änderungen rückgängig gemacht werden und die Datenbank bleibt konsistent.



Daten in die Datenbank oder Index können über DAOs gelesen oder geschrieben werden. Sie sind der einzige Punkt um solche Operationen durchzuführen. So kann im DAO jeweils entschieden werden, ob die Datenbank oder der Index für die gewünschte Operation verwendet werden soll und eine zukünftige Änderung dieser Entscheidung kann an einem Ort angepasst werden.

Für die meisten Tabellen existiert ein DAO, welches die Operationen für diese vornimmt. Diese leiten von `ADao` ab. Dies ist eine abstrakte Klasse, welche im Konstruktor eine Datenbank-Connection erwartet und eine jooq-Konfiguration erstellt. Diese kann dann in den Subklassen verwendet werden um auf die Datenbank zuzugreifen. Wenn keine Connection übergeben wird, wird automatisch eine vom `DBConnectionProvider` bezogen. Die DAO-Klassen verwenden jeweils die von jooq generierten Hilfsklassen, welche nur von diesen DAO-Klassen verwendet werden dürfen.

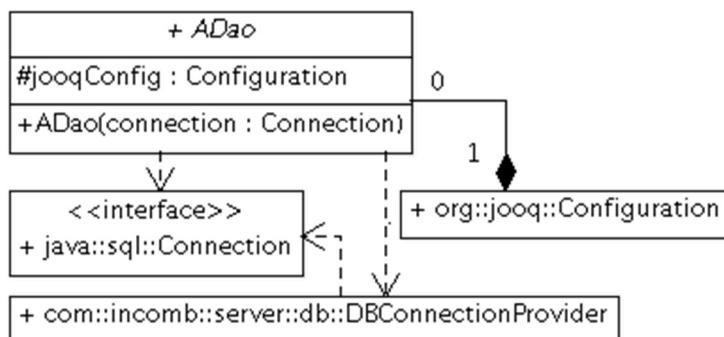
Jooq wandelt die Records einer Tabelle automatisch in Java-Objekte um. Diese werden POJOs genannt. Diese werden von den DAO-Klassen zurückgegeben und als Methoden-Argumente akzeptiert.

Die DAO-Klassen befinden sich im Package `com.incomb.server.model.dao`.

Die POJOs befinden sich im Package `com.incomb.server.model`.

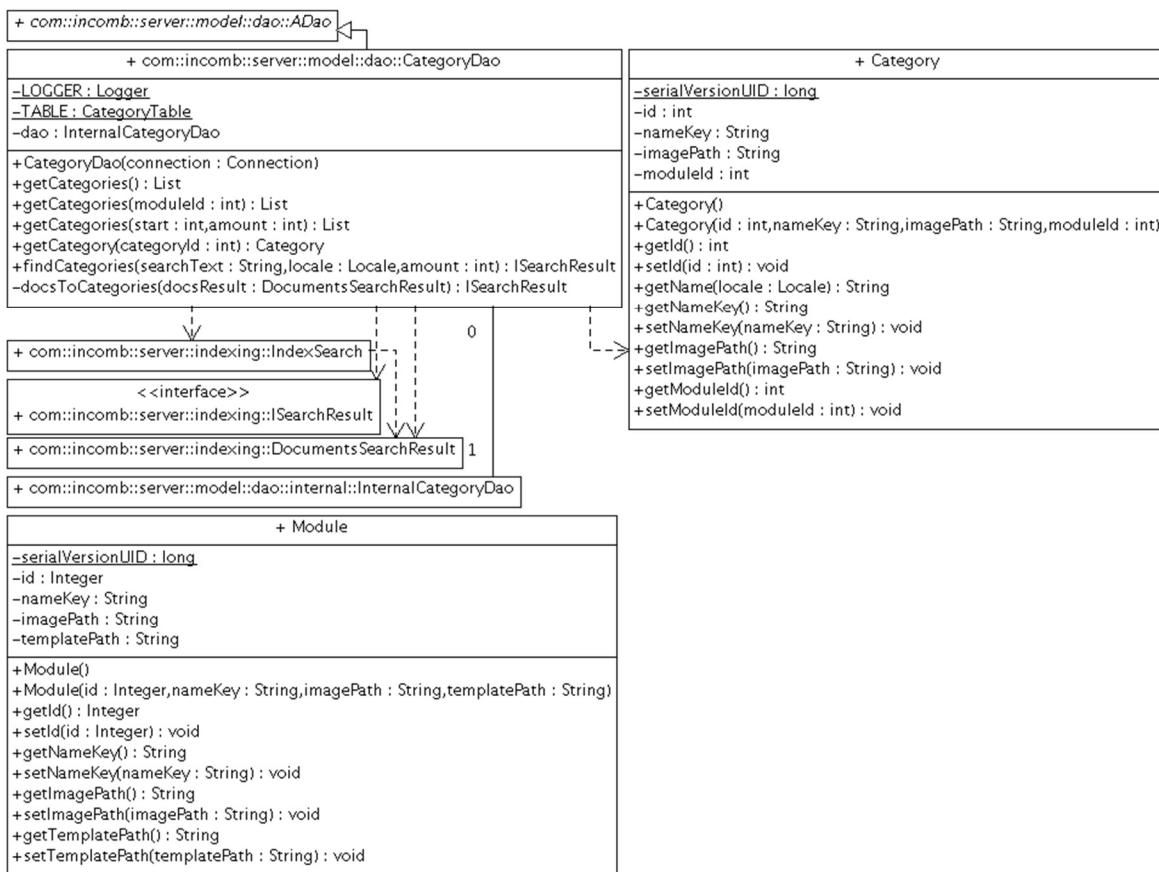
Die generierten jooq-Klassen befinden sich in den Packages

- `com.incomb.server.model.dao.internal`
- `com.incomb.server.model.records`
- `com.incomb.server.model.tables`

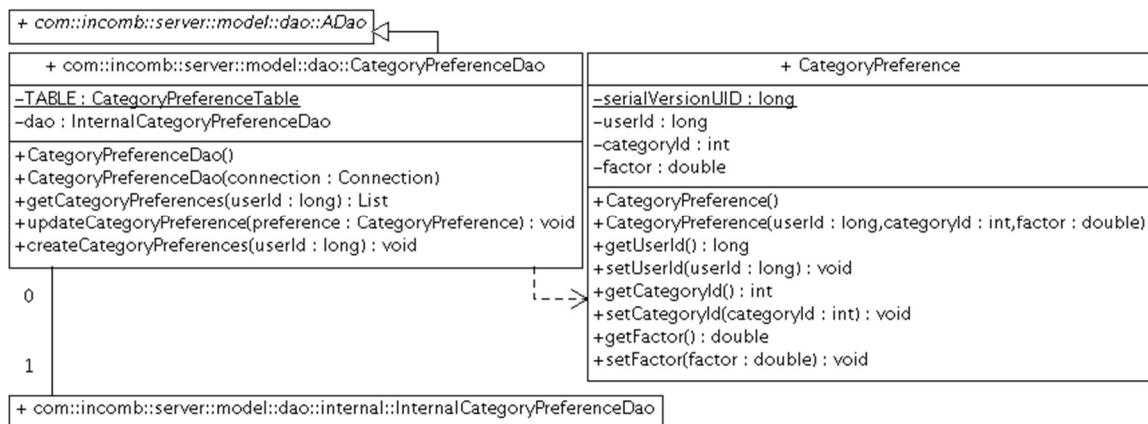


Nachfolgend sind die UML-Klassendiagramme für die Datenbank-Tabellen.

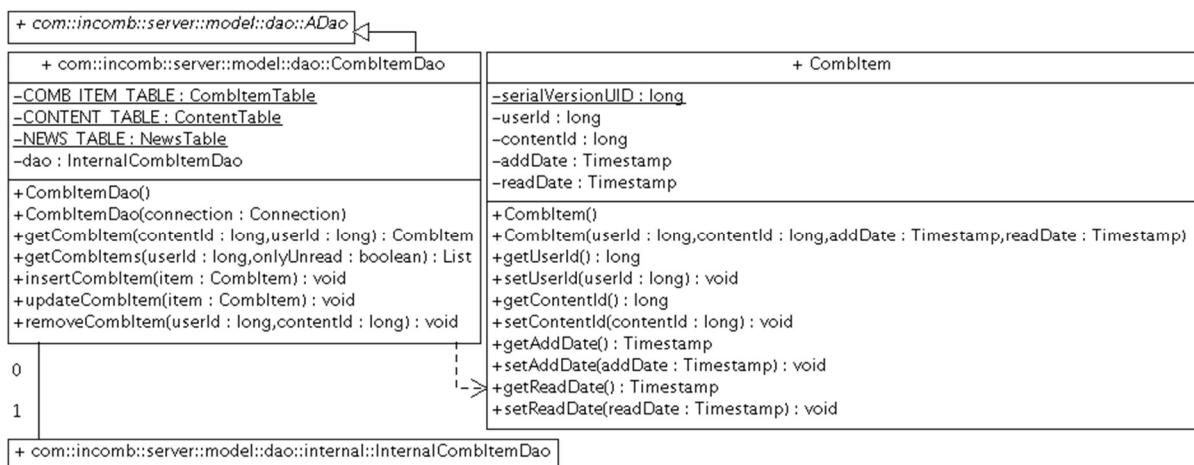
5.2.1.1 Kategorien



5.2.1.2 Kategorien Präferenzen

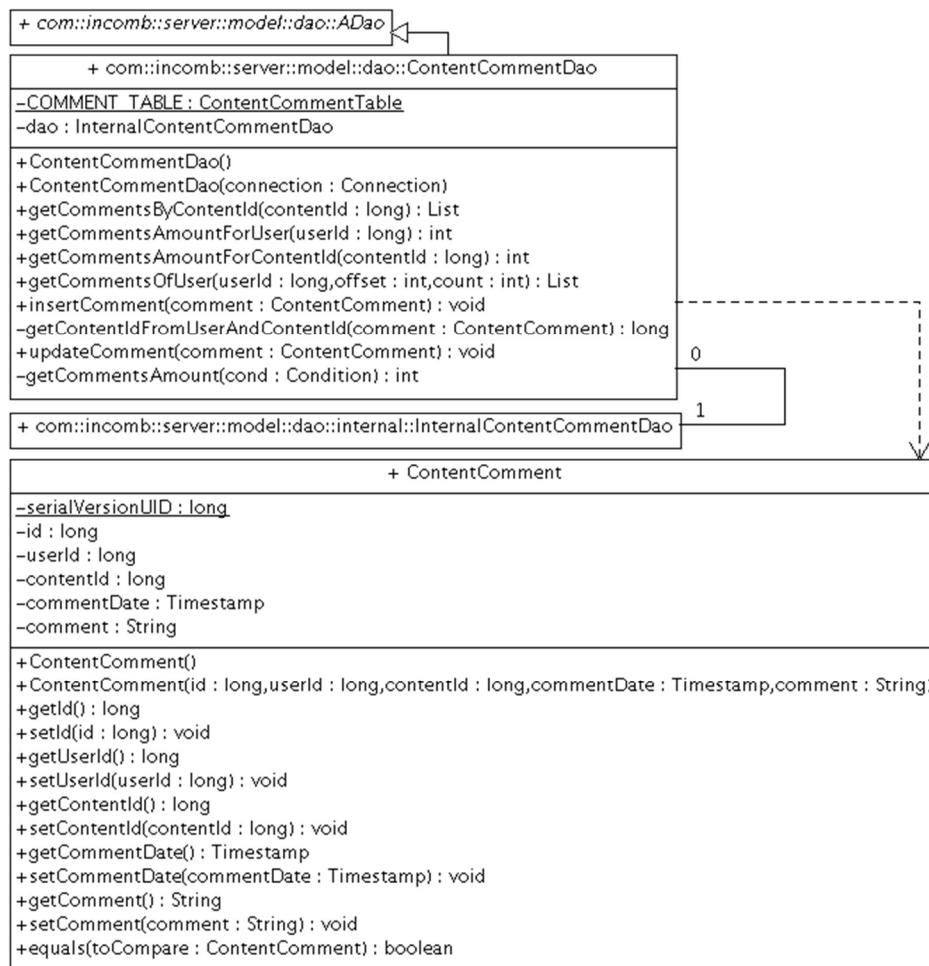


5.2.1.3 Comb Items



5.2.1.4 Kommentare

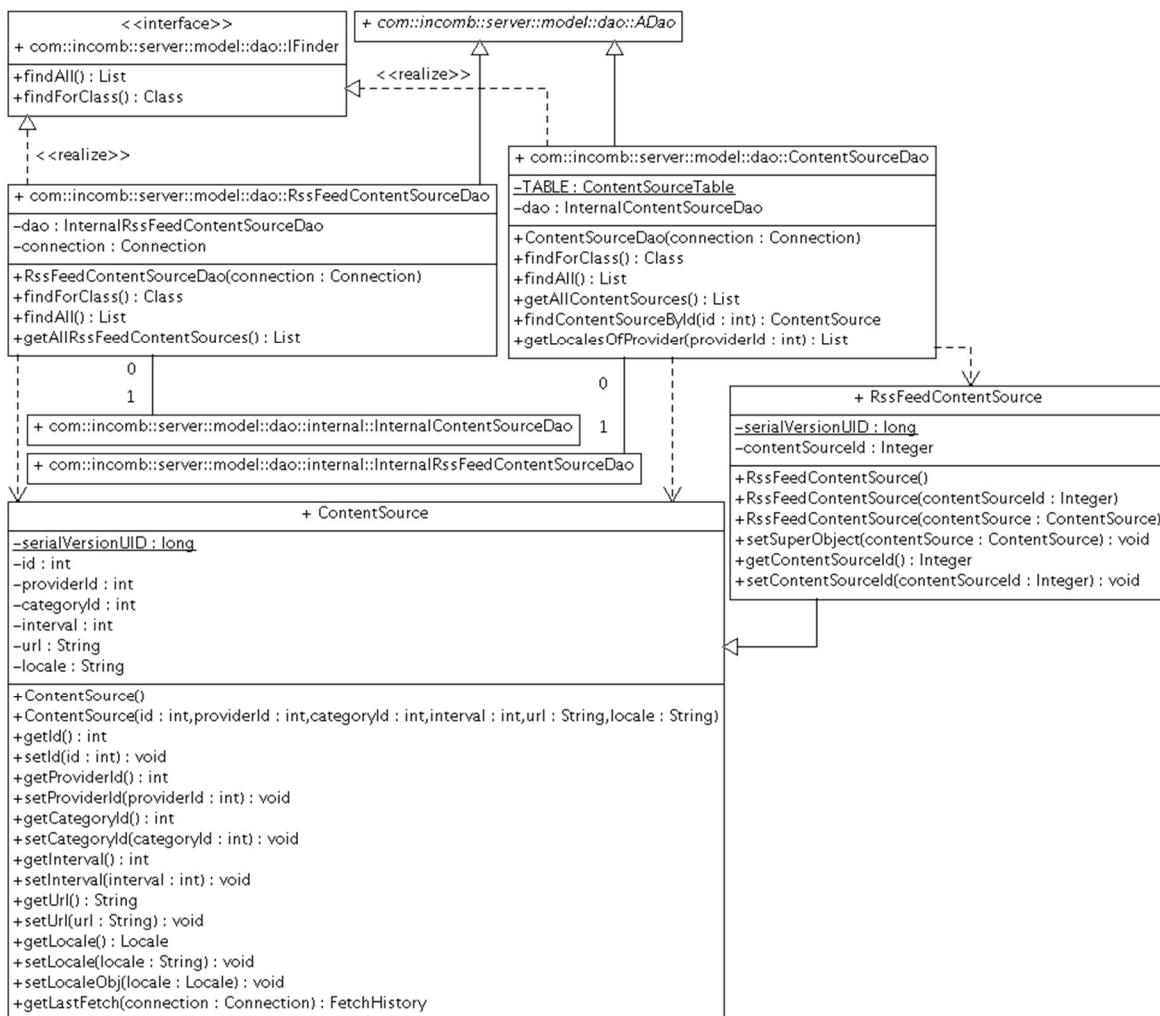
Package: com.incomb.server.model.dao



Die Klasse `ContentCommentDao` ermöglicht es, Kommentare mit verschiedenen Bedingungen von der Datenbank zu holen. Wie alle Dao Klassen leitet `ContentCommentDao` von `ADao` ab und muss mit einer Connection im Konstruktor instanziiert werden.

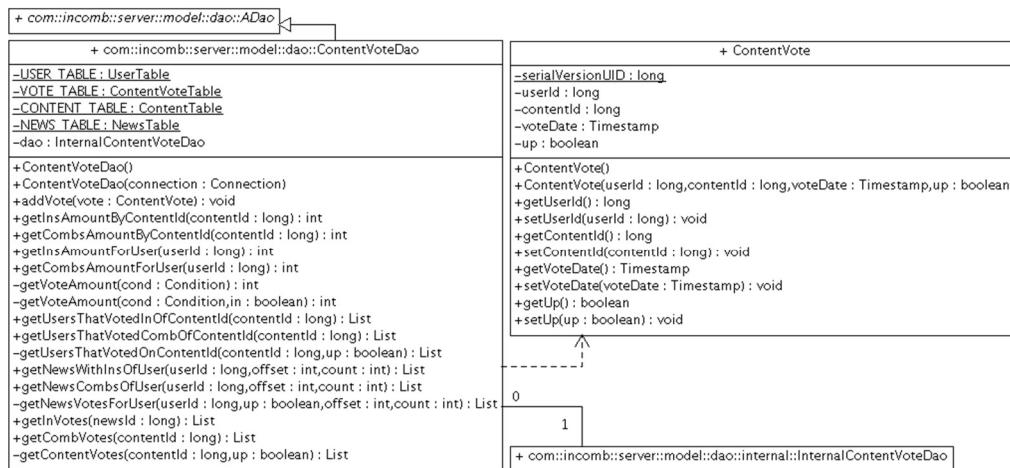
Die meisten Methoden geben entweder eine Liste von ContentComment Models zurück oder geben eine Anzahl der Kommentare zurück.

5.2.1.5 Content Sources



5.2.1.6 Ins, Combs

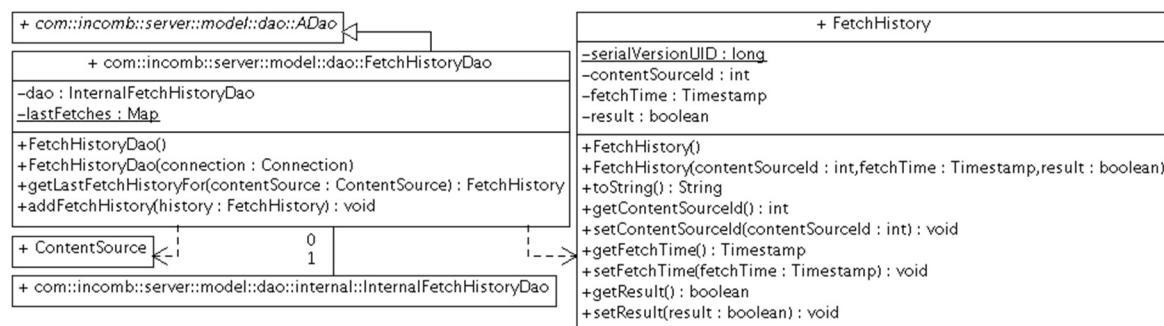
Package: com.incomb.server.model.dao



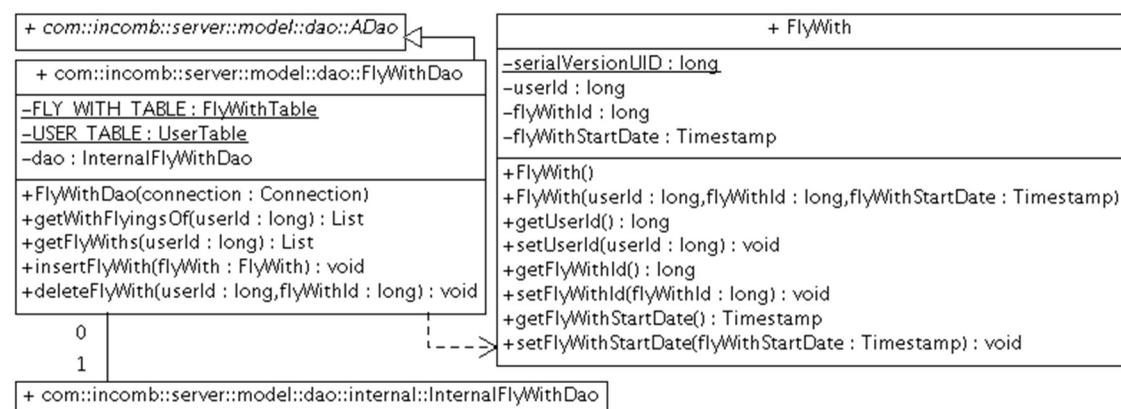
Die Klasse ContentVoteDao ermöglicht es, Votes (Ins/Combs) mit verschiedenen Bedingungen von der Datenbank zu holen. Wie alle Dao Klassen leitet ContentVoteDao von ADao ab und muss mit einer Connection im Konstruktor instanziiert werden.

Die meisten Methoden geben entweder eine Liste von ContentVote Models zurück oder geben eine Anzahl der Ins/Combs zurück. Diese Votes sind meistens auf einen NewsArtikel (contentId) bezogen.

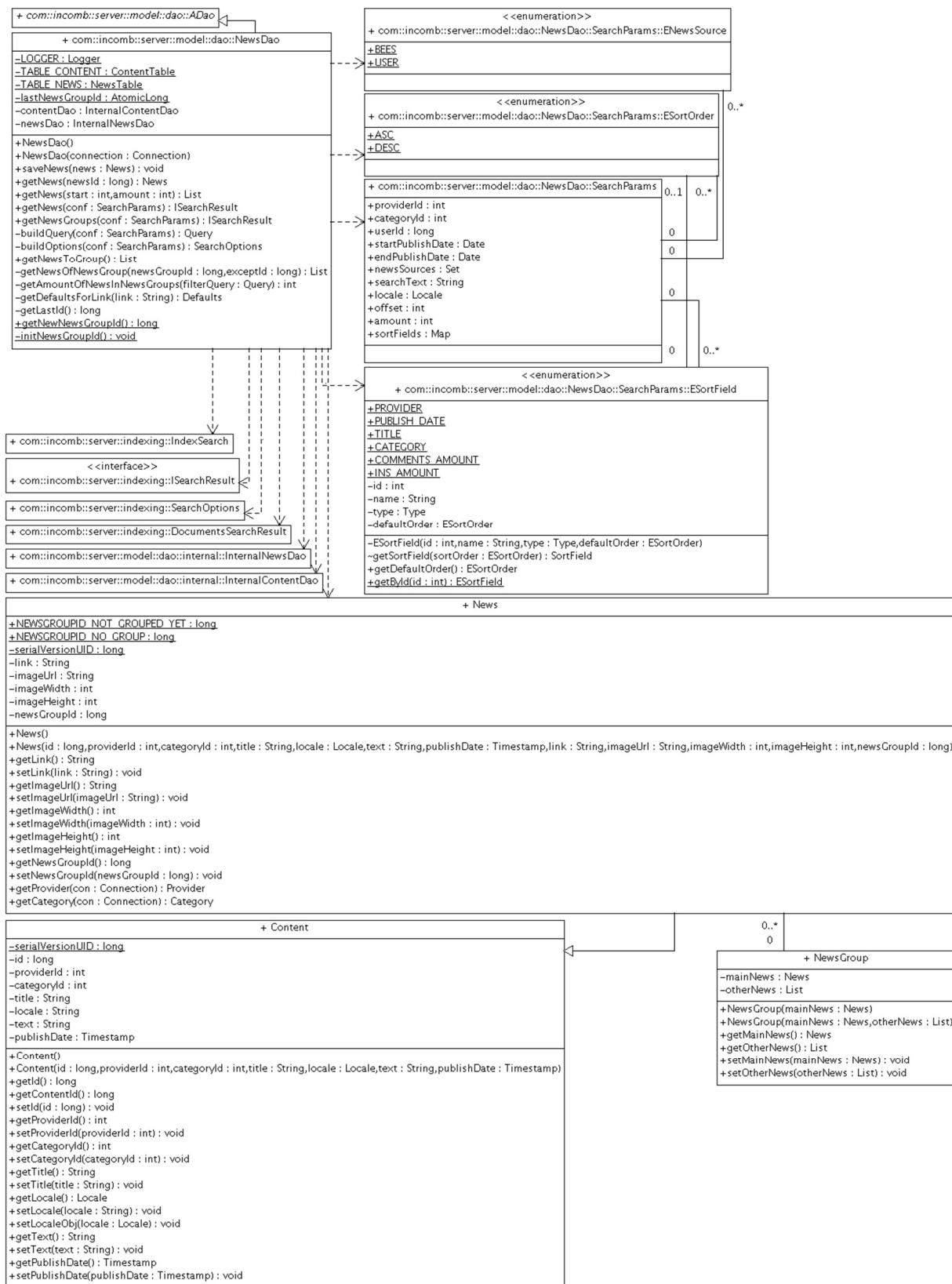
5.2.1.7 Fetch History



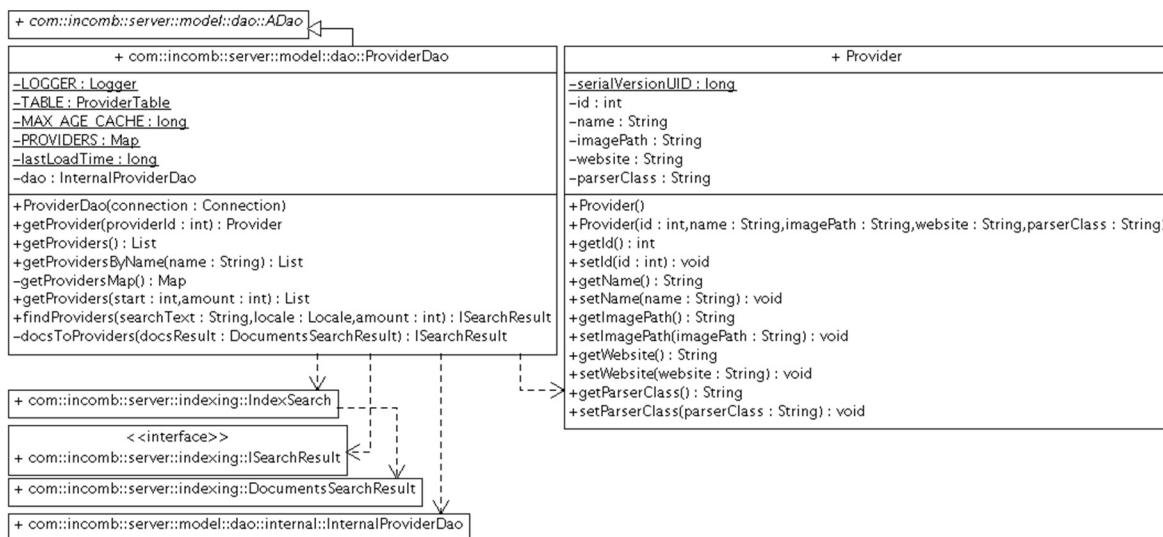
5.2.1.8 Fly Withs



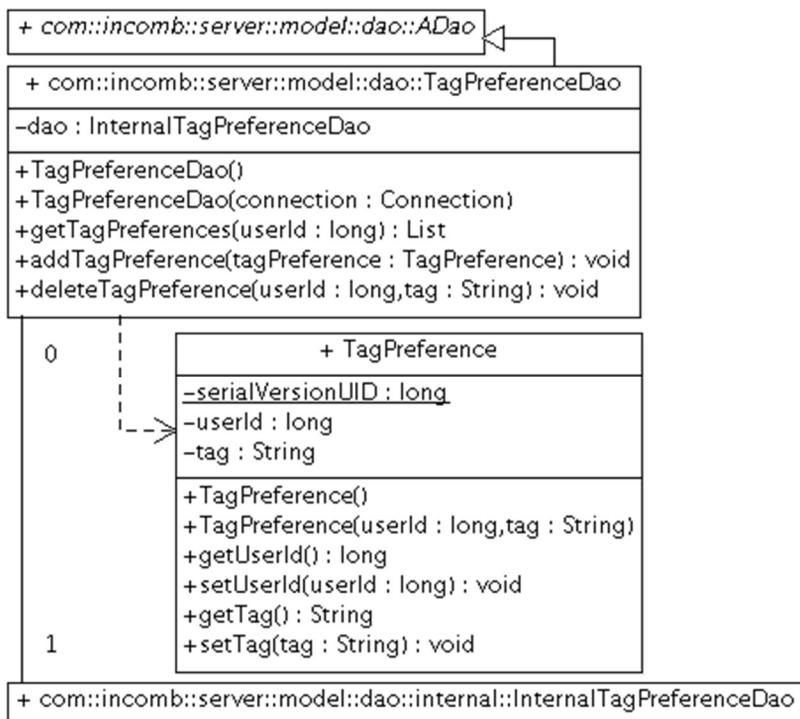
5.2.1.9 Nachrichten



5.2.1.10 Anbieter



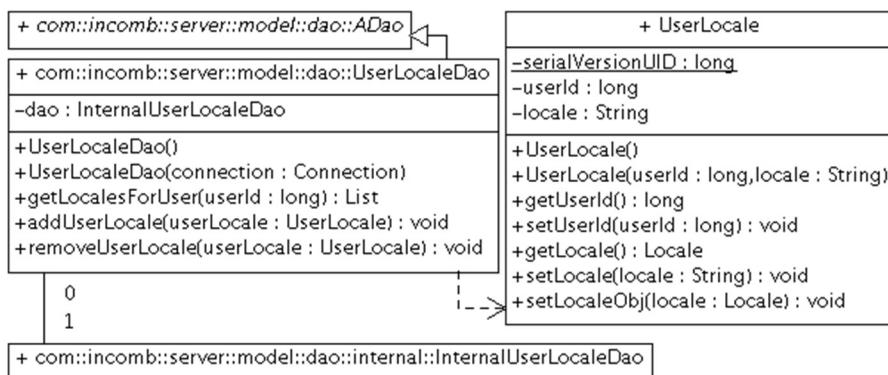
5.2.1.11 Tag Präferenzen



5.2.1.12 Benutzer



5.2.1.13 Benutzer Sprachen



5.2.2 Hilfs-Klassen

Die Klassen im Package com.incomb.server.utils sind Hilfsklassen, welche nur statische Methoden enthalten und nicht instanzierbar sind. Sie bieten kleine Code-Schnipsel an.

5.2.2.1 CloseUtil

Eine Hilfsklasse um Closeable und AutoCloseable-Implementationen zu schliessen ohne Exceptions behandeln zu müssen.

+ com::incomb::server::utils::CloseUtil
-LOG : Logger
+close(close : Closeable) : void
+close(close : AutoCloseable) : void

5.2.2.2 CollectionUtil

Hilfsklasse um Operationen auf Collections durchzuführen.

+ com::incomb::server::utils::CollectionUtil
-CollectionUtil()
+subList(list : List,offset : int,amount : int) : List

5.2.2.3 ConfigUtil

Hilfsklasse um Systemteile zu Konfigurieren. Zum Beispiel den Logger.

+ com::incomb::server::utils::ConfigUtil
+LOGBACK_CONFIG : String
-docBase : String
+getDocBase() : String
+setDocBase(docBase : String) : void
+initLogger() : void

5.2.2.4 GravatarUtil

Generiert für einen User einen Link um dessen Gravatar-Bild abrufen zu können. Dieses wird als Profilbild verwendet.

+ com::incomb::server::utils::GravatarUtil
-LOGGER : Logger
-URL_PREFIX : String
-URL_POSTFIX : String
-GravatarUtil()
+getGravatarImg(user : User) : String

```

graph LR
    GU[+ com::incomb::server::utils::GravatarUtil] --> User[+ com::incomb::server::model::User]
  
```

5.2.2.5 HtmlUtil

Hilfsklasse für Modifikationen im HTML-Code. Zum Beispiel Tags entfernen.

+ com::incomb::server::utils::HtmlUtil
+BREAK TAGS PATTERN : Pattern
+REMOVE TAGS PATTERN : Pattern
-HtmlUtil()
+removeTags(content : String,preserveBreaks : boolean) : String

5.2.2.6 JsonUtil

Hilfsklasse um JSON zu generieren.

+ com::incomb::server::utils::JsonUtil
-LOGGER : Logger
+getJson(bean : Object,formatOutput : boolean) : String
+getJson(bean : Object) : String

5.2.2.7 LocaleUtil

Klasse, um alle verfügbaren Locales abzurufen und Strings in Locales umzuwandeln.

+ com::incomb::server::utils::LocaleUtil
+DEFAULT_LOCALE : Locale
-LOCALES : List
-LocaleUtil()
+getAllLocales() : List
+toLocale(locale : String) : Locale

A dashed arrow points from the LocaleUtil class to the org::apache::commons::lang3::LocaleUtils class, indicating a dependency.

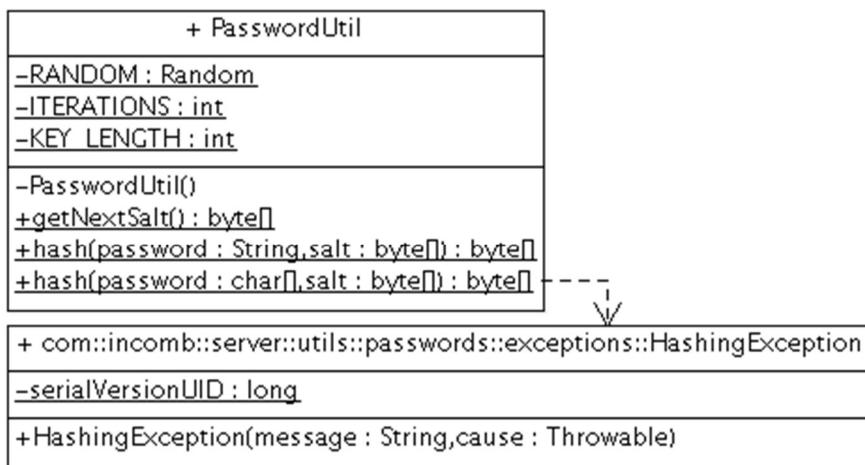
5.2.2.8 ObjectUtil

Hilfsklasse, um Validierung von Objekten durchzuführen.

+ com::incomb::server::utils::ObjectUtil
+assertNotNull(object : T,message : String) : T

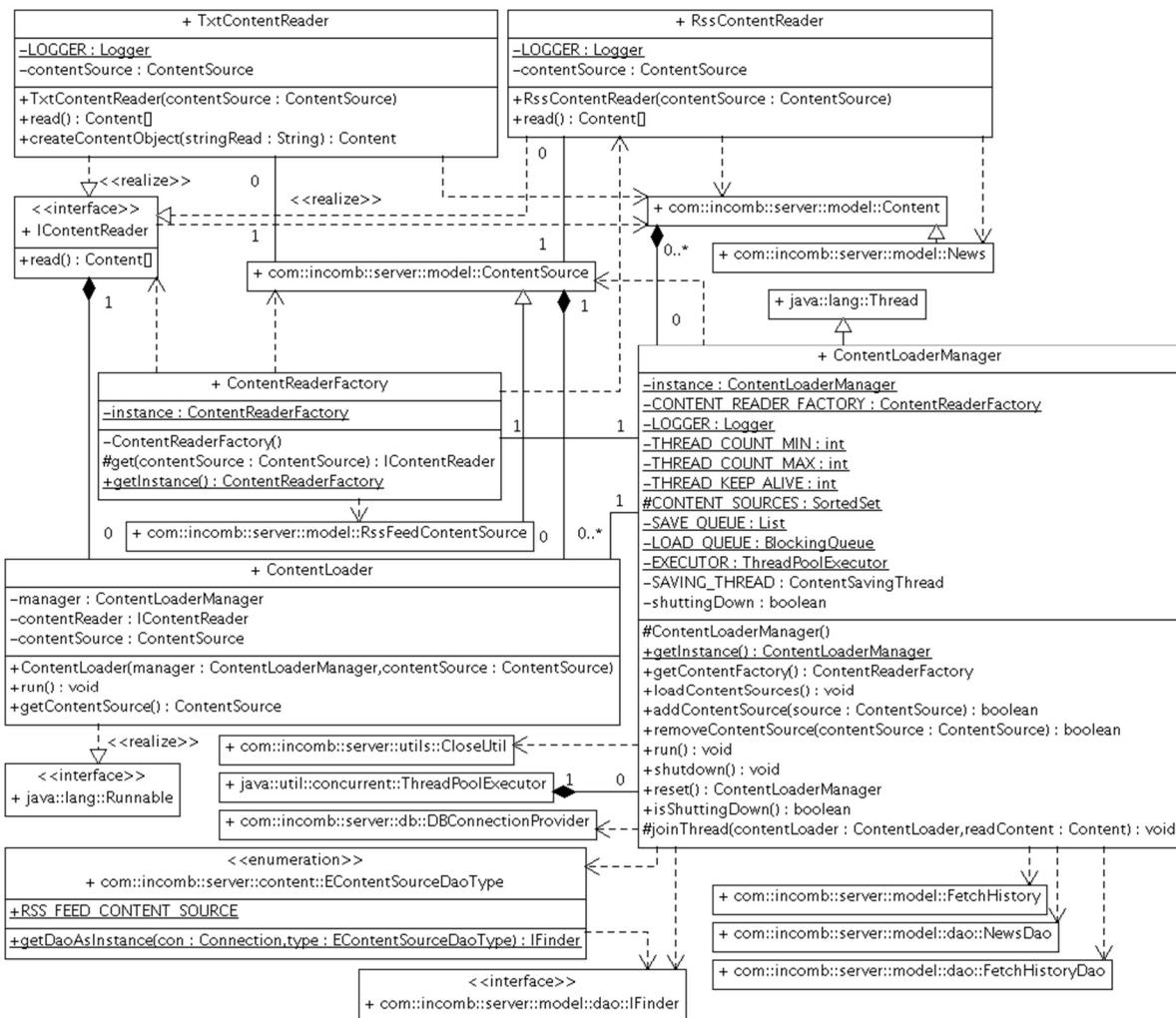
5.2.2.9 PasswordUtil

Klasse, um Passwörter zu verschlüsseln und Salts zu generieren. Eine HashingException wird geworfen, wenn ein Fehler dabei auftritt.



5.2.3 Content Loading

Bei Content Loading verstehen wir das laden von verschiedenen ContentSourcen und das abschliessende Speichern der jeweiligen geladenen Content Objekten. Dies geschieht über unser NewsDao. Unser Content Loading befindet sich im com.incomb.server.content.loader Package. Der Parser welcher für das Auslesen und generieren der Content Objekten zuständig ist, wird weiter unten erklärt.



5.2.3.1 ContentLoaderManager

Beim ContentLoaderManager handelt es sich um ein Thread mit einem ThreadPoolExecutor welcher dafür zuständig ist, ContentSourcen in einem auf der ContentSource definierten Zeitabstand neu zu laden. Das laden findet in der run Methode statt. Über die loadContentSources() Methode können alle ContentSourcen initial geladen werden. In der Methode joinThread() wird von dem ContentLoader aufgerufen. Es wird der geladene Content übergeben, welcher dann in die Datenbank gespeichert wird. Auch werden hier neue FetchHistory Objekte erstellt und ebenfalls gespeichert. Die Shutdown Methode wird benötigt um alle Threads bei dem Herunterfahren des Servers zu beenden. Mit reset() kann eine neue Instanz des ContentLoaderManager erstellt werden wobei die alte überschrieben wird.

5.2.3.2 ContentLoader

Im ContentLoader werden die Content Objekte in der run() Methode geladen und dem ContentLoaderManager übergeben. Er besitzt einen IContentReader welchen er von der ContentReaderFactory über die get() Methode erhält.

5.2.3.3 ContentReaderFactory

Die ContentReaderFactory ist ein Singleton. Ihre get() Methode analysiert die übergebene ContentSource und gibt den entsprechenden IContentReader zurück.

5.2.3.4 IContentReader

Das Interface ContentReader stellt eine read() Methode zur Verfügung welche geladene Content Objekte zurückgibt. Das Interface haben wir erstellt, da wir später mehr als nur Rss Feeds laden wollen.

5.2.3.4.1 TxtContentReader

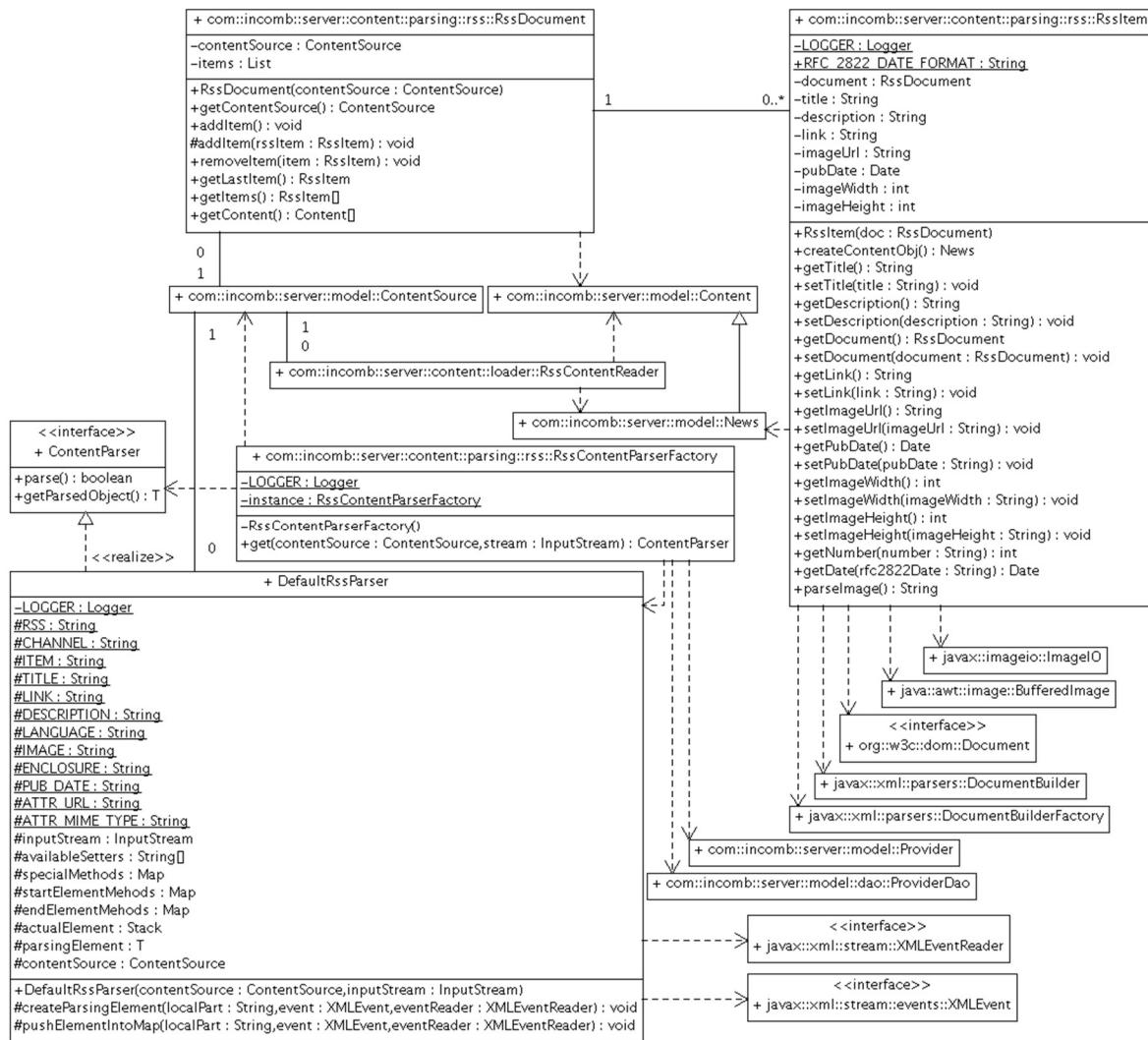
Der TxtContentReader kann verwendet werden um einfache Text Dateien aus dem Internet herunterzuladen und zu Parsen.

5.2.3.4.2 RssContentReader

Der RssContentReader wird verwendet um eine Verbindung in das Internet zu einem News Anbieter zu erstellen. Ist die Verbindung erstellt wird der Content des Rss Feeds geladen und geparsst.

5.2.4 Content Parsing

Unter Content Parsing verstehen wir das Generieren von Content Objekten aus gelesenem Content.



5.2.4.1 RssContentParserFactory

Die RssContentParserFactory ist ein Singleton. In der get() Methode wird kontrolliert ob die ContentSource eine Parser Klasse definiert hat. Ist dies der Fall, so wird diese Instanziert und zurückgegeben. Andernfalls wird eine Instanz des DefaultRssParser zurückgegeben.

5.2.4.2 ContentParser

Der Contentparser ist ein Interface welches Methoden zum parsen bereitstellt. Mit der getParsedObject() Methode kann das geparsste Objekt zurückgegeben werden.

5.2.4.3 DefaultRssParser

Der DefaultRssParser ist eine Default Implementation um ein Rss Dokument zu parsen. In den Maps specialMethods, startElementMethods, endElementMethods werden die Methoden gespeichert, welche aufgerufen werden sollen wenn man bei Parsen auf spezielle Operationen angewiesen ist, oder man Start oder Endtags findet. Trifft man auf Start Tags, so wird die Methode in der Map gesucht und über Reflection aufgerufen. Das gleiche gilt für End Tags.

5.2.4.4 RssDocument

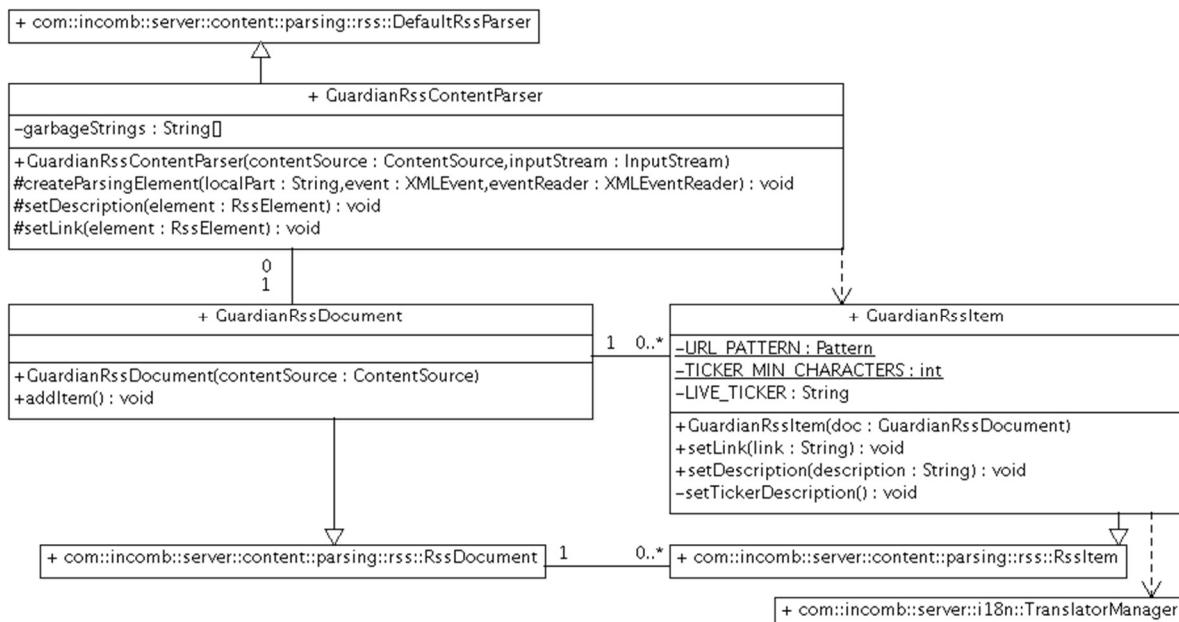
Das RssDocument repräsentiert ein Rss Dokument. Es besitze eine Liste von RssItems. Diese Klasse ist so gebaut, dass sie gut subklassbar ist. Mit der Methode getContent() können alle Content Objekte zurückgegeben werden. Die addItem() Methode wird verwendet, um neue RssItems hinzuzufügen.

5.2.4.5 RssItem

Dass RssItem repräsentiert das Rss Tag welches sich im Channel Tag befindet. Es verfügt über Getter und Setter um alle Eigenschaften zu setzen.

5.2.4.6 Guardian Parser Klassen

Der GuardianParser ist eine Subklasse des DefaultRssParsers. Dieser wird erstellt, da der Nachrichtenanbieter *The Guardian* einen Live-Ticker in das Rss Dokument einfügt. Es wird zudem ein GuardianRssItem eingeführt, welches erkennen soll ob es sich um einen Ticker handelt.



5.2.5 Indexierung und Suche

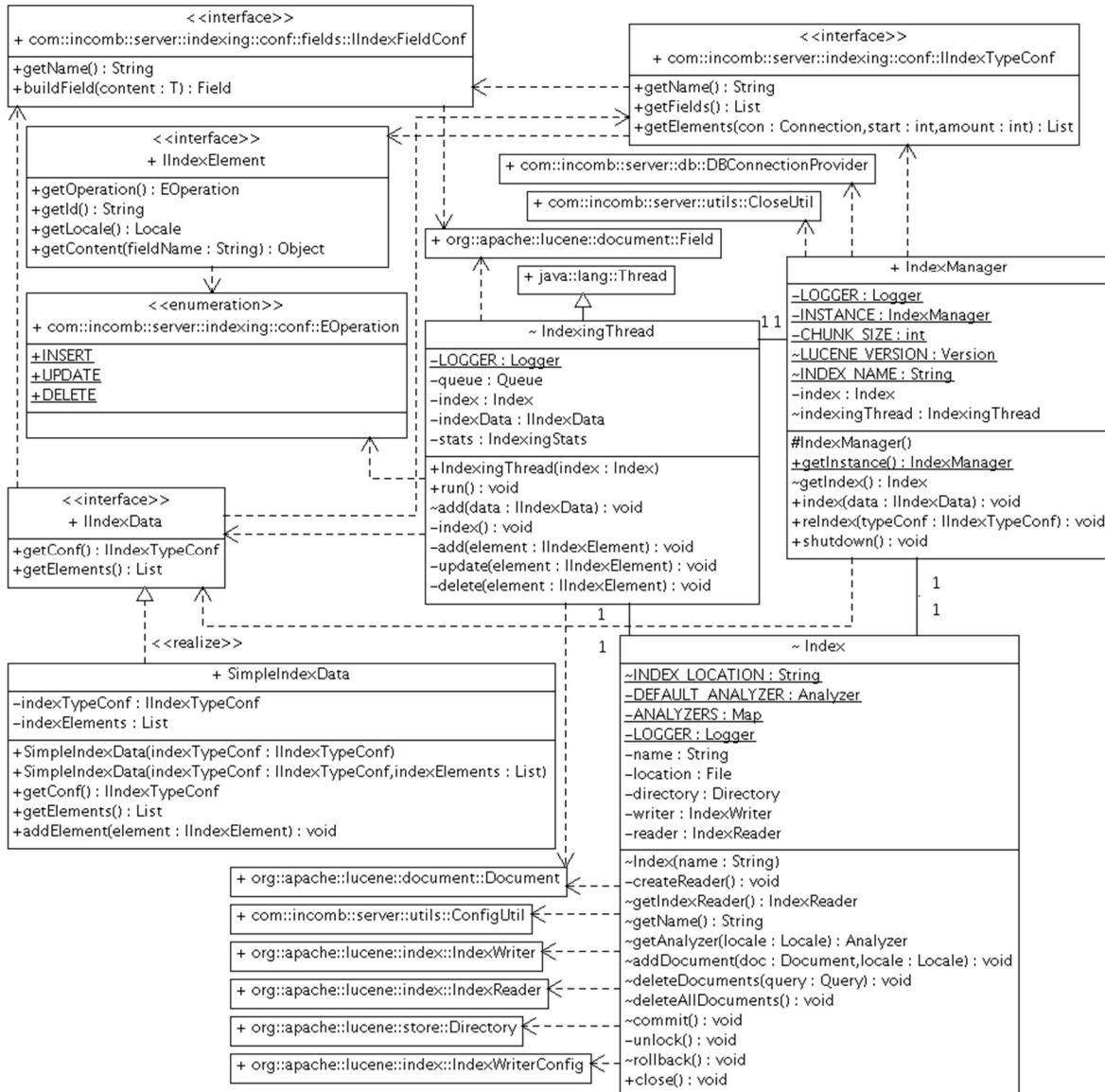
Im Moment werden Nachrichten, Themenbereiche, Anbieter und Benutzer indexiert um diese durchsuchbar zu machen und generell schneller abrufen zu können. Dazu wurde das Framework Lucene verwendet und im Projekt ein eigenes Framework um dieses herum gebaut um Lucene grösstenteils zu abstrahieren.

5.2.5.1 Framework

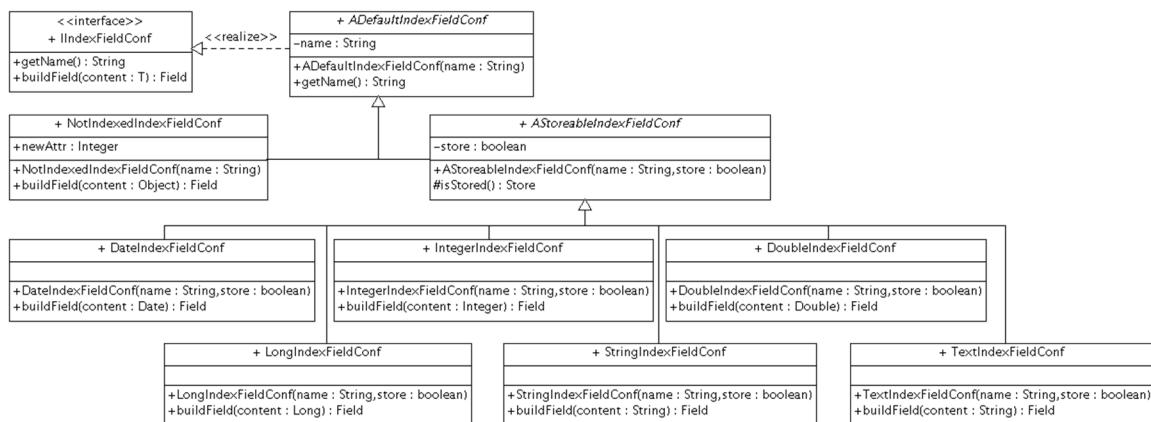
Für jeden Typ von Daten, der indexiert werden soll (Benutzer, Nachrichten, ...) wird eine Implementation von `IIndexTypeConf` geschrieben, welche die Konfiguration der Werte, die indexiert werden sollen zurückgibt. Wenn nun eine Änderung im Index gemacht werden soll (Daten einfügen, ändern oder löschen) müssen diese als `IIndexElement` in einem `IIndexData` dem `IndexManager` übergeben werden. Das `IIndexElement` gibt die Werte zurück, die indexiert werden sollen. Das `IIndexData` fasst mehrere `IIndexElement` zusammen. Eine einfache Standardimplementation ist das `SimpleIndexData`.

Der IndexManager gibt das IIndexData in die Queue des IndexingThreads. Dieser hat die Aufgabe die Änderungen im Index vorzunehmen. Er arbeitet ein IIndexData nach dem anderen in seiner Queue ab. Dies macht er in einem eigenen Thread. Er wandelt die IIndexElement in Lucenes Documents und Fields um und speichert die dann mit dem Index-Objekt. Über den Enum EOperation Wert auf IIndexElement wird bestimmt, wie die Änderung im Index stattfinden soll (INSERT, UPDATE, DELETE).

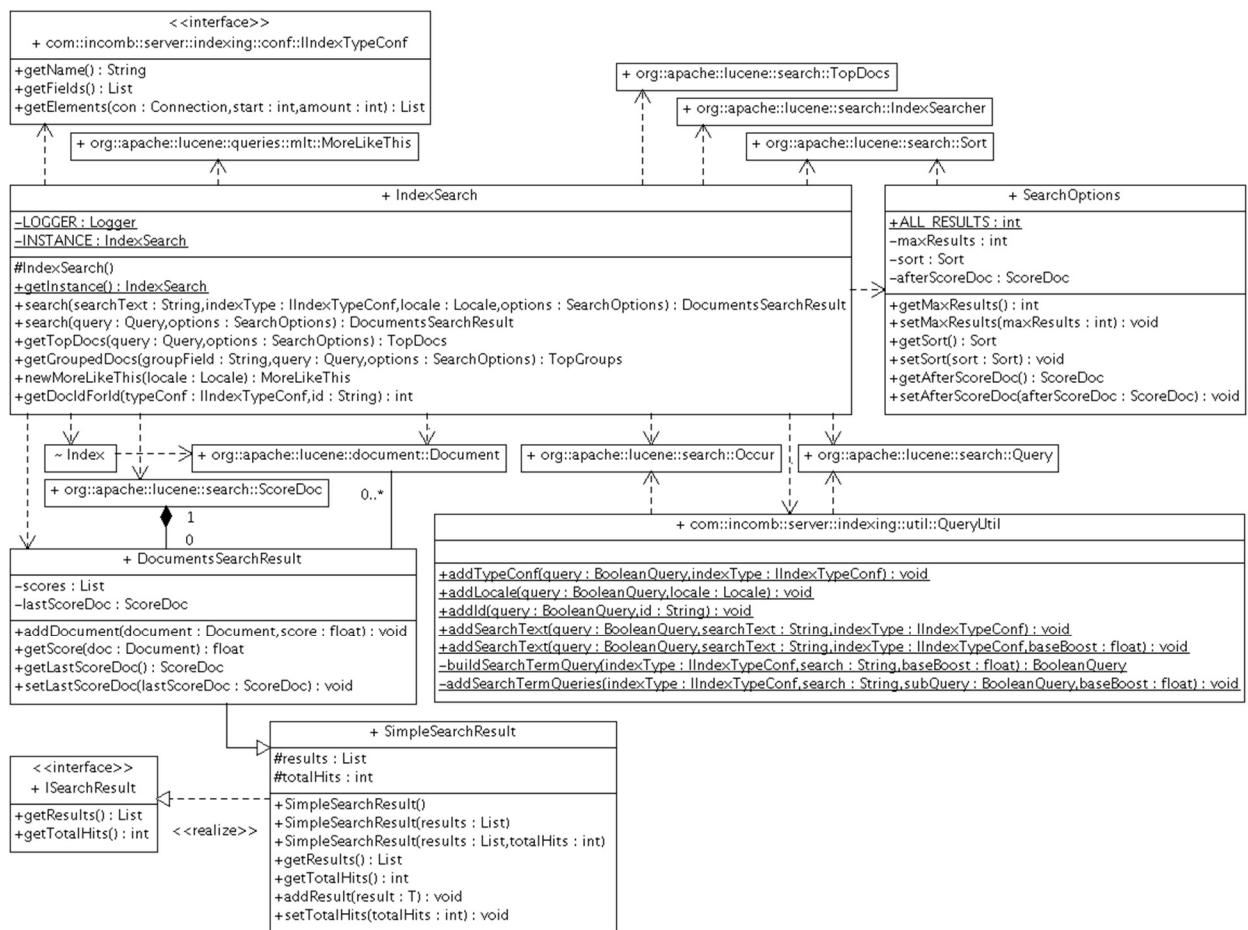
Die Index-Dateien werden unter /WebContent/WEB-INF/indexes/main abgelegt.



Die IIndexFieldConf verpacken einen Wert in ein Lucene Field. Dies tun sie je nach Datentyp und Konfiguration.



Alle Suchabfragen laufen über die IndexSearch-Instanz. Es kann ein Suchtext zusammen mit einem IIndexTypeConf übergeben werden. Dann wird in allen Feldern dieses Typs nach dem Suchtext gesucht. Es kann auch direkt eine Lucene Query übergeben werden, die ausgeführt wird. Zurückgegeben wird immer ein DocumentsSearchResult mit den gefundenen Documents, der Gesamtzahl an Treffern und die Scores, welche die einzelnen Documents erzielt haben. Über die SearchOptions kann die Suche eingeschränkt werden. So kann die maximale Anzahl an Treffern definiert werden oder wie die Resultate sortiert werden sollen. Das QueryUtil unterstützt beim Erstellen von Query-Objekten.

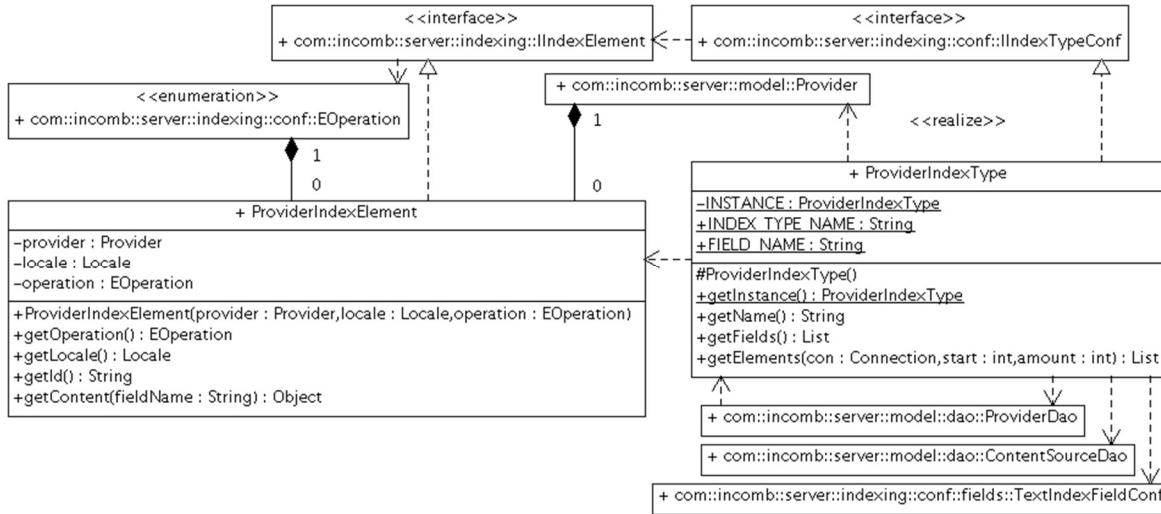


Nachfolgend die Implementationen der einzelnen Typen.

5.2.5.2 Nachrichten-Anbieter

Package com.incomb.server.providers.indexing

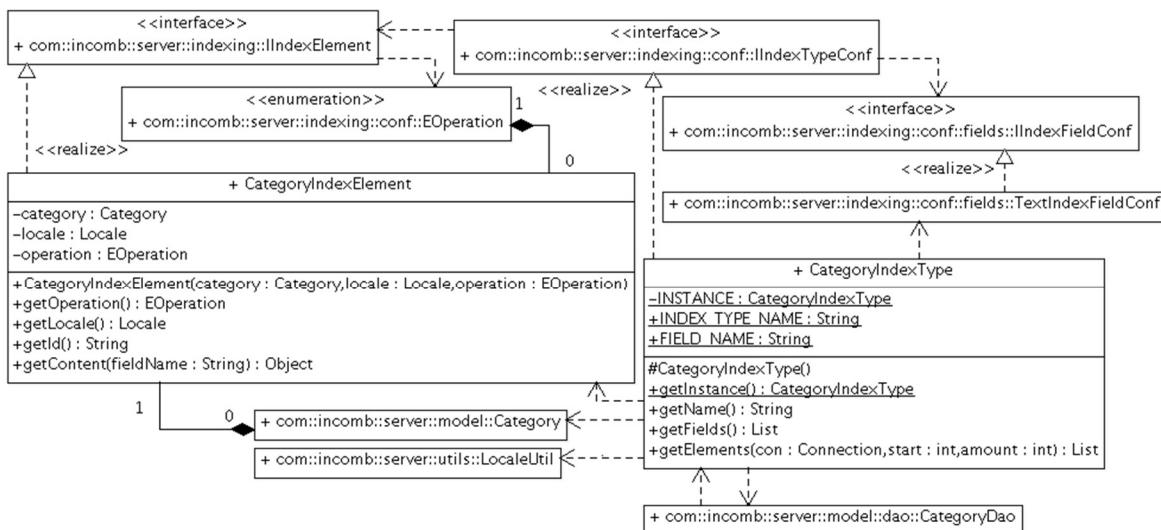
Indexiert wird lediglich der Name des Anbieters. Über das ProviderDao kann nach einem Text gesucht werden. Dazu wird der ProviderIndexType und das ProviderIndexElement verwendet. Das ProviderIndexElement hält den Provider der indexiert werden soll und die Locale für die sprachabhängigen Texte.



5.2.5.3 Themenbereiche

Package com.incomb.server.categories.indexing.

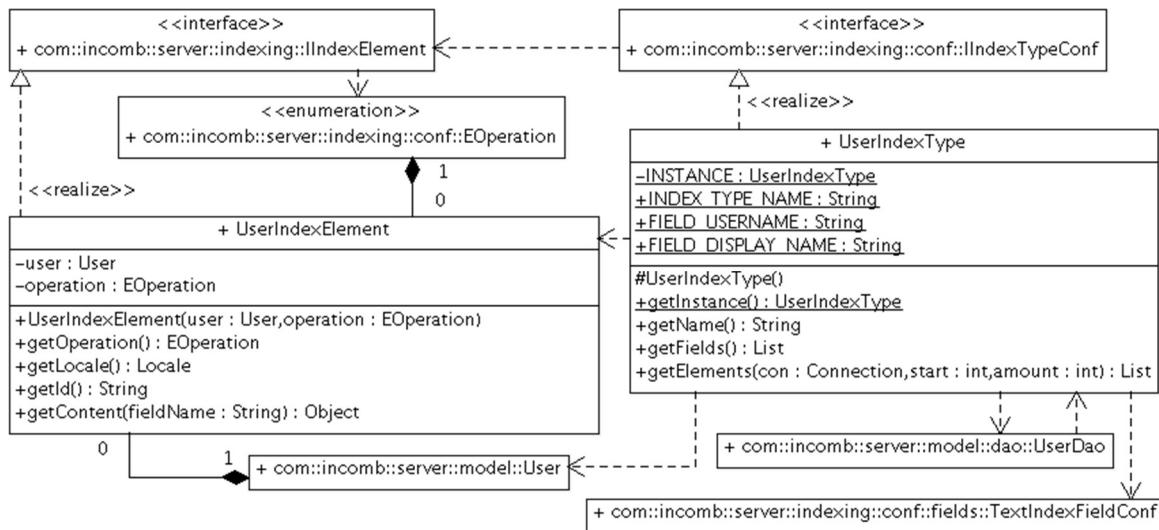
Bei den Kategorien wird der sprachabhängige Name indexiert, weswegen die Locale auf dem CategoryIndexElement gebraucht wird. Außerdem hält es die Category, die indexiert wird. Das CategoryDao kann verwendet werden um nach einem Text zu suchen.



5.2.5.4 Benutzer

Package com.incomb.server.users.indexing.

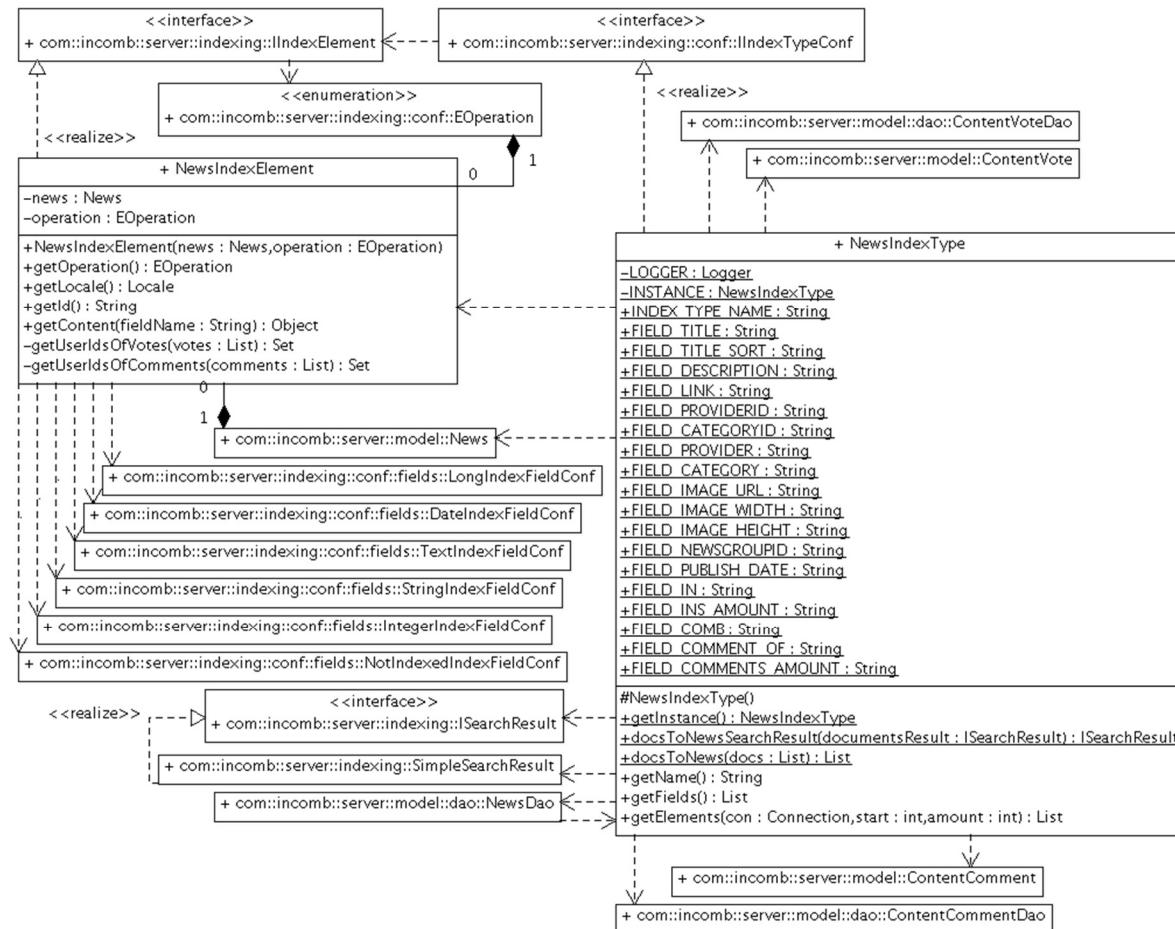
Wenn sich ein neuer Benutzer registriert oder ein bestehender seine Daten aktualisiert, wird vom UserDao eine Änderung im Index ausgelöst. Es lässt sich auch über das UserDao nach einem Text suchen. Dazu wird der UserIndexType, welcher als Felder den Benutzernamen und den Anzeigenamen zum Indexieren zurückgibt. Das UserIndexElement hält den zu indexierenden User und gibt dessen Werte in #getContent zurück.



5.2.5.5 Nachrichten

Package com.incomb.server.content.indexing.

Wenn eine Nachricht über das NewsDao erstellt oder verändert wird, löst dieses eine Indexierung dieser Nachricht aus. Indexiert werden alle Felder (zurückgegeben von NewsIndexType), die auch auf dem News-Objekt vorhanden sind. Wenn nach Nachrichten gesucht wird kann so aus den Dokumenten das News-Objekt schnell zurückgebaut werden, ohne langsame Datenbankabfragen machen zu müssen. Die Felder werden je nach Datentyp konfiguriert und das NewsIndexElement gibt die entsprechenden Werte im richtigen Datentyp zurück. Es werden auch die Menge an Kommentaren und Ins indexiert, damit bei einer Suche nach diesen sortiert werden kann. Ausserdem werden die IDs der User mit der Nachricht indexiert, die einen Kommentare verfasst, ein In oder ein Comb gegeben haben, um so diese Nachricht für Nachfliegende bei einer Suche als relevant zu markieren.



5.2.6 News Grouping

Package com.incomb.server.content

Der SimilarNewsFinder liefert für die übergebene Nachricht ähnliche zurück. Damit eine Nachricht eine ähnliche ist, müssen diese Kriterien erfüllt sein:

- Gleicher Themenbereich.
- Gleiche Sprache.
- Nicht älter und nicht jünger als drei Stunden.
- Möglichst gleiche Wörter in Titel und Text.
- Lucene Score muss grösser als 1.2 sein.

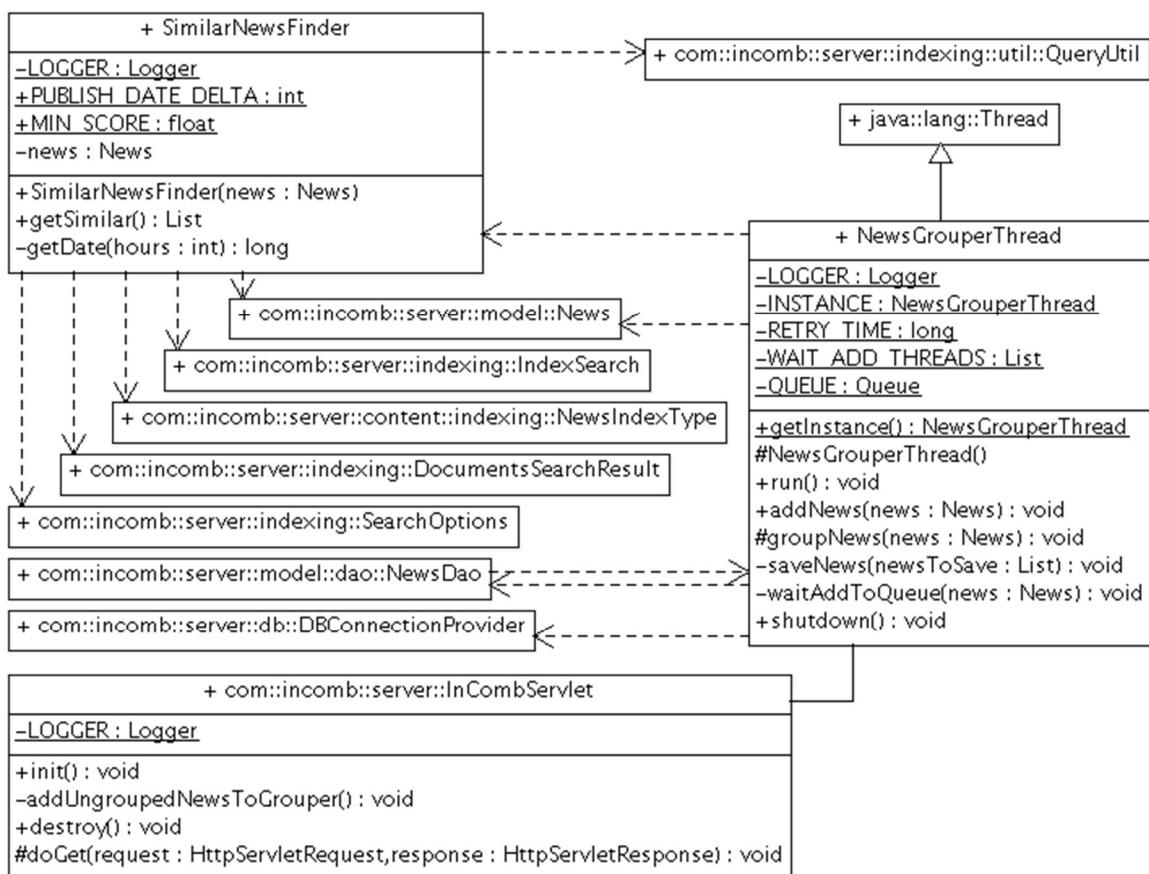
Es wird unter anderem die MoreLikeThis-Funktion von Lucene verwendet, um gleiche Wörter in Titel und Text zu finden.

Der NewsGrouperThread hält eine Queue mit Nachrichten, für welche noch passende Nachrichten gefunden werden sollen. Diese arbeitet er ab und setzt auf der Nachricht die newsGroupId. Alle Nachrichten mit derselben newsGroupId bilden eine Gruppe mit ähnlichen Nachrichten. Wenn die newsGroupId = 0 ist, dann wurden keine ähnlichen Nachrichten gefunden und wenn newsGroupId = -1 ist, dann wurde noch nicht nach ähnlichen Nachrichten gesucht. Beim Aufstarten des Servers werden alle Nachrichten, für welche noch nicht nach ähnlichen gesucht wurden in die Queue gelegt

(InCombServlet). Wenn eine neue Nachricht im NewsDao gespeichert wird, wird diese ebenfalls in die Queue gelegt.

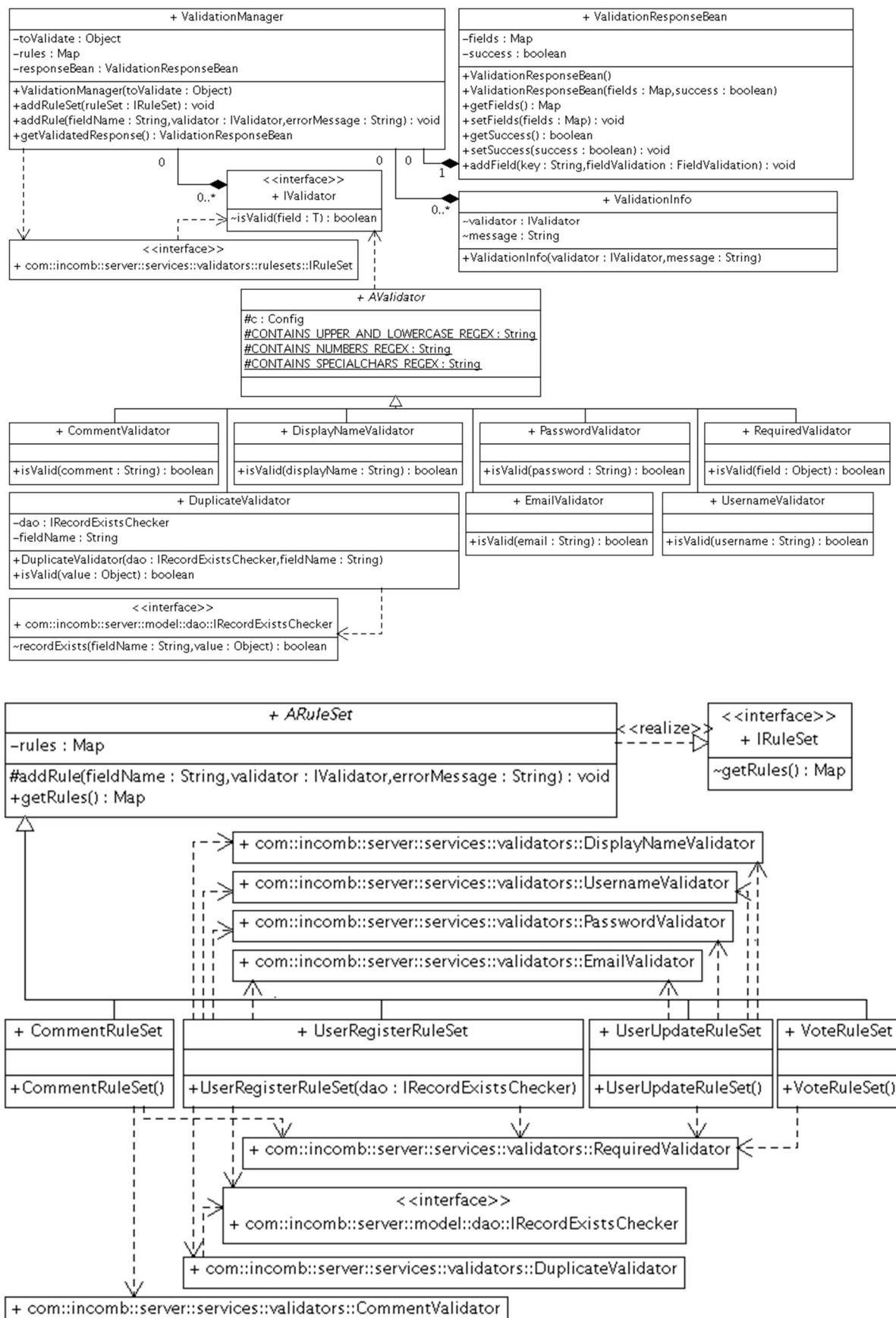
Falls während der Suche nach ähnlichen Nachrichten ein Fehler auftritt, wird die Nachricht nach 60 Sekunden erneut in die Queue gelegt und es wird erneut versucht ähnliche Nachrichten zu finden.

Wenn im NewsGrouperThread passende Nachrichten gefunden wurden, wird bei den gefundenen Nachrichten nach einer schon existierenden Gruppe gesucht. Die erste Nachricht, bei der die newsGroupId > 0 ist wird als newsGroupId für die anderen verwendet. Falls bei keiner eine newsGroupId > 0 vorhanden ist, wird auf dem NewsDao eine neue generiert. Die gefundene oder neu generierte newsGroupId wird dann auf allen gefundenen Nachrichten und der „Basis-Nachricht“ gespeichert, die eine newsGroupId <= 0 haben. Gespeichert wird wie immer über das NewsDao.



5.2.7 Validation

Package: com.incomb.server.services.validators



Formulare auf InComb müssen validiert werden. Grundsätzlich sollte man nur Model-Instanzen

validieren. (Also Java Beans, welche direkt vom Client stammen). Wenn man ein Formular validieren möchte, gibt es zwei Möglichkeiten:

1. Bei kleinen Formularen mit 2-3 Feldern direkt auf dem ValidationManager mit addRule eine neue Regel hinzufügen. addRule braucht den Namen des zu validierenden Feldes, sowie eine Instanz die IValidator implementiert (z.B. CommentValidator). Zuletzt wird noch eine lokalisierte Fehlermeldung benötigt („Getrennt wie bei allen Lang-Strings, z.B. „validation.comments.invalid“)
2. Bei grösseren Formularen sollte ein RuleSet erstellt werden. Rulesets leiten alle von ARuleSet ab. ARuleSet wiederum implementiert das Interface IRuleSet. Auf dem RuleSet können nun gleich wie in der ersten Variante Rules gesetzt werden (addRule).

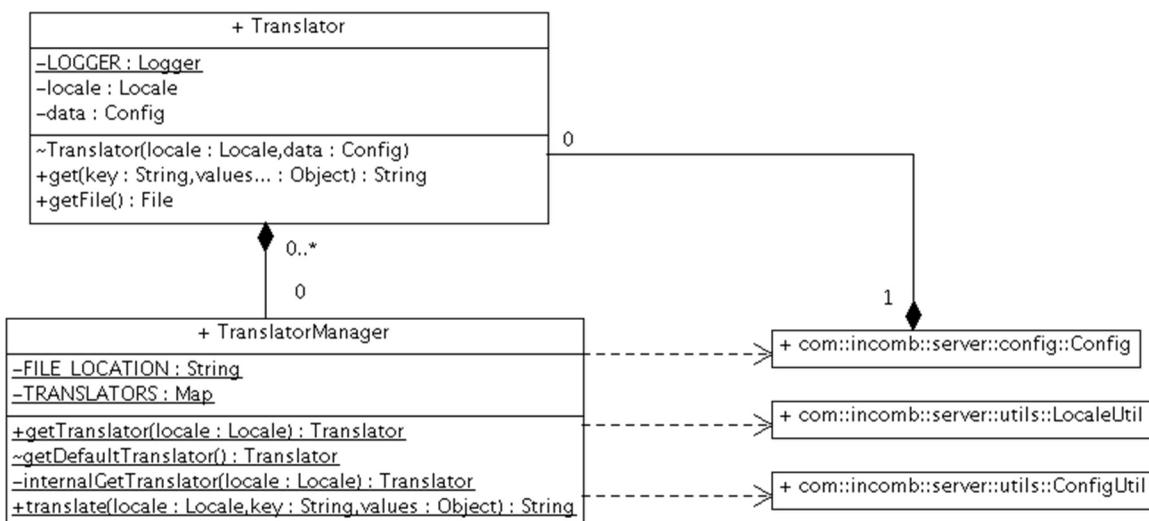
Der ValidationManager muss jeweils instanziert werden, dann Rules oder RuleSets hinzugefügt werden und dann getValidatedResponse aufgerufen werden. Sobald getValidatedResponse() aufgerufen wird, werden Intern alle Feldnamen mit Reflection aufgerufen mit dem jeweiligen, angegebenen Validator validiert. Falls irgendeine der Validationen fehlschlägt wird auf der ValidationResponseBean success auf false gesetzt. Jedes fehlgeschlagene Feld hat dann auch die mitgelieferte lokale Fehlermeldung, welche bei addRule mitgegeben wurde. Falls alle Felder valid sind, wird success auf true gesetzt.

Wichtiger Hinweis: Alle Validatoren geben bei isValid() true zurück falls das Feld null, oder leer ist. Falls ein Feld nicht leer sein sollte, muss dem Feld ein RequiredValidator hinzugefügt werden.

Falls der Wert eines Feldes noch nicht in der Datenbank vorhanden sein darf, kann man dies mit dem DuplicateValidator validieren. Teilweise greifen Validatoren auf die Config Klasse zu, wenn sie verschiedene Optionen prüfen sollten (z.B. PasswordValidator, hierbei wird die Länge und Komplexität angegeben).

5.2.8 Translations

Package com.incomb.server.i18n



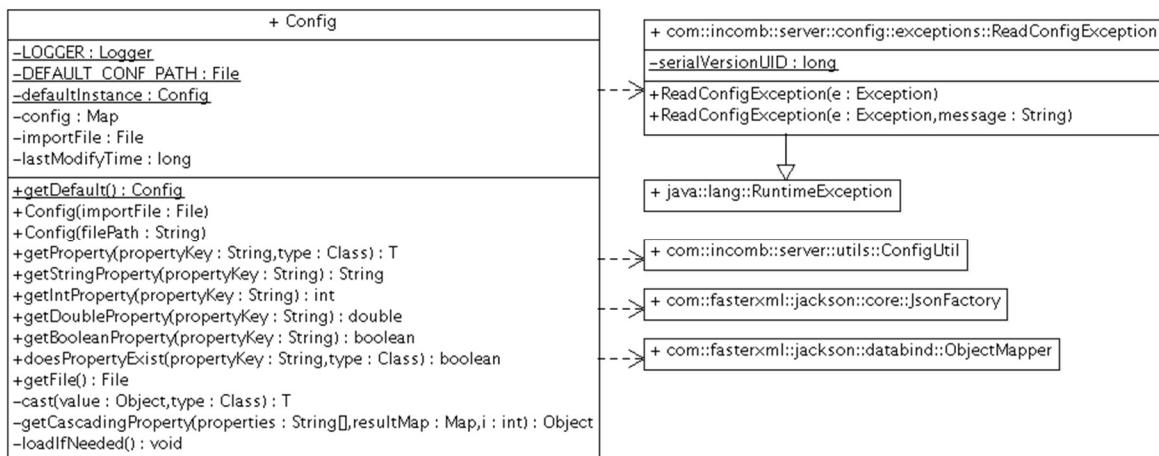
Es existiert für jede verfügbare Sprache eine Translator-Instanz. Diese Instanzen werden vom TranslatorManager verwaltet, welcher nur statische Methoden enthält. Über TranslatorManager#getTranslator erhält man für die übergebene Locale den Translator zurück oder falls diese Sprache nicht unterstützt wird, wird der Translator für die Standardsprache (Englisch) zurückgegeben.

Auf einem Translator-Objekt kann man #get aufrufen um eine Übersetzung zu erhalten. Dazu gibt man den Schlüssel der Übersetzung und falls benötigt Platzhalter-Werte an, die im übersetzten Text ersetzt werden sollen, an. Zurückgegeben wird dann der übersetzte Text. Um diesen Prozess zu vereinfachen, kann man direkt TranslatorManager#translate aufrufen, der dann mit der Locale wieder den Translator holt und auf diesem dann #get aufruft.

Die Übersetzungen werden in einem JSON-File im Ordner WEB-INF/translations gespeichert, welches als Name den ISO-Code der Sprache hat und mit .json endet. Beispielsweise en.json, de.json. Die Inhalte werden mit Config-Instanzen gehalten.

5.2.9 Konfiguration

Package: com.incomb.server.config



Die Config Klasse ist die zentrale Anlaufstelle für alle Konfigurationen. Es wird grundsätzlich „incomb_config.json“ ausgelesen. Das JSON-File kann auch während dem Betrieb angepasst werden, da mit loadIfNeeded() jeweils das Änderungsdatum der Datei geprüft wird.

Die ConfigDatei muss grundsätzlich im Constructor mitgegeben werden, allerdings kann mit getDefault den Standardpfad laden.

Grundsätzlich muss man spezifizieren, welchen Datentyp man zurückhaben möchte, da JSON nicht Java-Kompatible Definitionen der Datentypen besitzt. Somit muss bei getProperty das .class Attribut des gewünschten Datentyps mitgegeben werden. Alternativ gibt es zu den meist gebrauchten Datentypen verschiedene Wrapper-Methoden wie zum Beispiel getStringProperty(liefert einen String zurück) etc.

Ausserdem kann geprüft werden, ob ein Wert existiert.

Um Werte zu finden muss ein „.“ separierter String bei getProperty mitgegeben werden.

```
{  
    "validation":{  
        "displayname": {  
            "minLength": 3,  
            "maxLength": 255,  
            "allowSpecial": true  
        }  
    }  
}
```

Will man bei dem Beispiel oben allowSpecial auslesen müsste also folgender Befehl verwendet werden:

```
Config c = new Config().getDefaults();  
Boolean allowSpecial = c.getBooleanProperty("validation.displayname.allowSpecial");
```

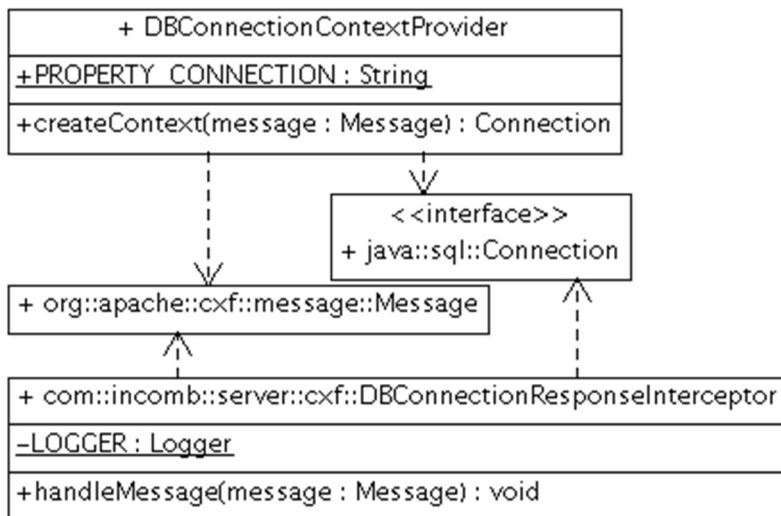
5.2.10 Services

Package com.incomb.server.services

Die REST-Schnittstelle, die im JavaScript verwendet wird um Daten abzurufen und zu speichern wird mit dem Framework CXF realisiert. Für jeden Datentyp, der über die Schnittstelle abgerufen oder verändert werden kann gibt es ein eigenes Package mit maximal zwei Klassen, welche dann die Schnittstelle bilden. Eine Klasse, welche mehrere Records behandelt und eine, welche sich auf einen einzelnen Record bezieht.

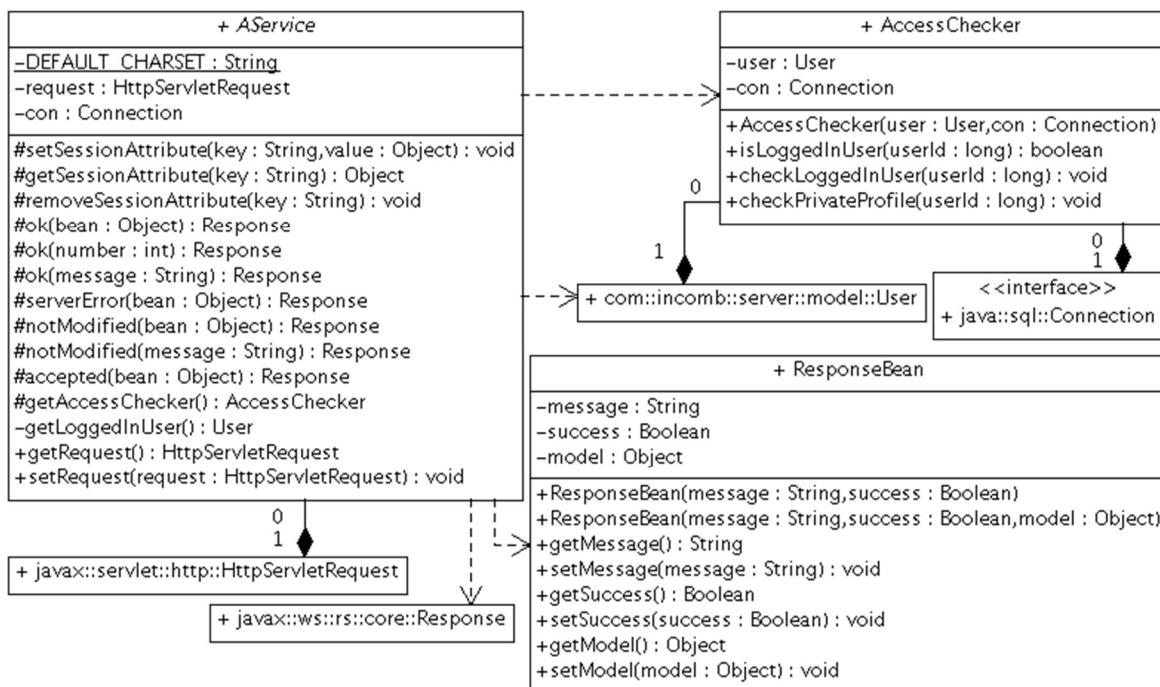
Alle URLs beginnen jeweils mit /api/. Die URL für die Schnittstelle, welche mehrere Records behandelt enthält den Namen des Datentyps in der Mehrzahl. Zum Beispiel die für Benutzer: /api/users. Bei der Schnittstelle, die einen einzelnen Record behandelt, wird ein eindeutiger Schlüssel angehängt. Für den Benutzer mit der id = 10 wäre das demnach: /api/users/10. Je nach HTTP-Methode werden Daten zurückgegeben oder verändert. Für jede erlaubte HTTP-Methode wird in der Klasse ein Methode erstellt. Diese gibt eine Response-Instanz zurück, welche dann durch CXF an den Client gesendet wird. Als Parameter werden GET-Parameter, Pfad-Parameter (bspw. die userId) und die Daten im Request-Body übergeben.

Es besteht ebenfalls die Möglichkeit als Parameter eine Datenbank-Connection zu erhalten. So wird während der gesamten Request-Verarbeitung auf dem Server dieselbe Connection verwendet. Der DBConnectionContextProvider liefert diese CXF. Falls am Ende der Verarbeitung der Response-Statuscode im Bereich 500-599 liegt, was einen Server-Fehler darstellt, wird automatisch ein Rollback der Connection durchgeführt. Ansonsten wird ein commit ausgeführt und die Daten sind erst dann tatsächlich in der Datenbank. Dies passiert im DBConnectionResponseInterceptor. So kann vermieden werden, dass durch einen Serverfehler (Exception, ...) eine Inkonsistenz in der Datenbank entsteht.

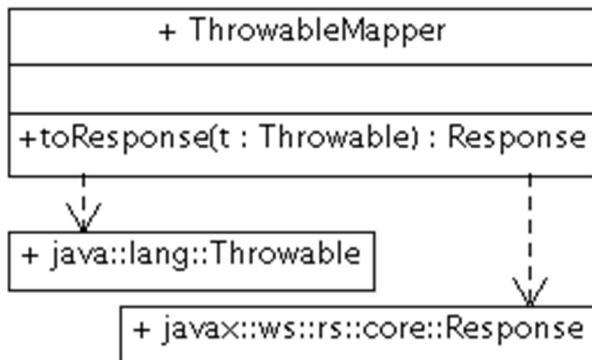


Um das Verarbeiten von Requests zu vereinfachen wurde ein kleines Framework dazu erstellt. Alle Services können von AService ableiten, mit welchem man schnell Response-Objekte bauen und die Session abrufen oder bearbeiten kann. Ausserdem lässt sich ein AccessChecker abrufen. Mit diesem kann überprüft werden, welcher Benutzer gerade angemeldet ist oder ob ein bestimmter Benutzer sein Profil auf privat gestellt hat.

Das ResponseBean wird vom AService verwendet. Es wird als JSON in der Response an den Client gesendet und enthält einen Status, eine Message und ein Objekt. Dies wird meistens nach einer Änderung in der Datenbank verwendet um dem Client den Status mitzuteilen.



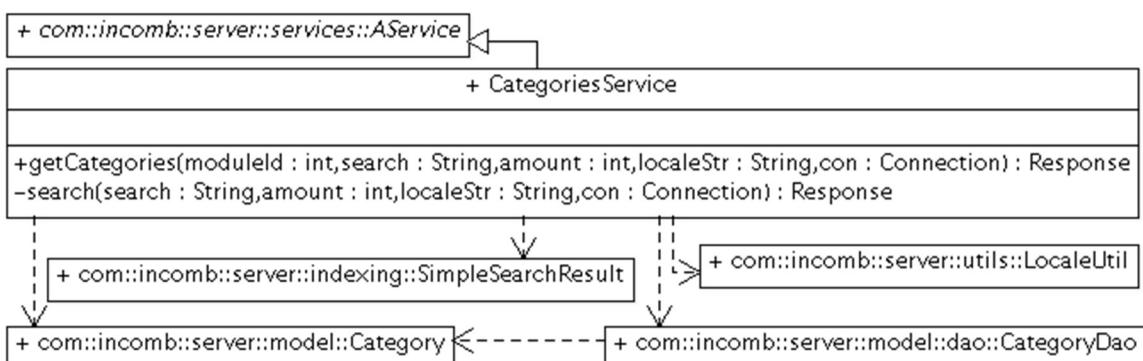
Falls während der Request-Verarbeitung eine Exception oder ein Error geworfen wird, der nicht abgefangen wurde, wird CXF diese an den ThrowabeMapper weiterleiten, welcher eine Response mit dem Statuscode 500 baut. So kann verhindert werden, dass CXF Informationen zum Code an den Client sendet.



5.2.10.1 Kategorien

Package com.incomb.server.services.categories.

/api/categories			
Methode	Parameter	Statuscodes	Beschreibung
GET	Query <ul style="list-style-type: none"> • moduleId • locale • search • amount 	200 – ok.	Gibt alle Categories zurück, die dem Modul zugewiesen sind oder sucht nach dem Suchtext. Es werden maximal so viele Kategorien zurückgegeben, wie bei amount angegeben. Wenn amount nicht angegeben wurde, werden alle gefundenen zurückgegeben.

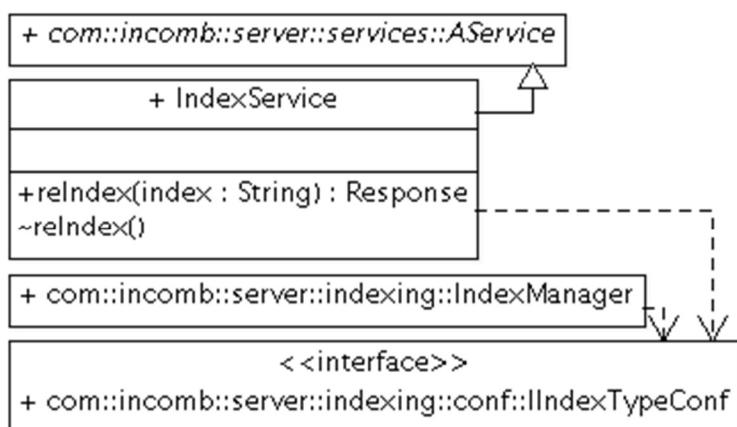


5.2.10.2 Index

Package com.incomb.server.services.indexes.

/api/indexes/{indexClass}

Methode	Parameter	Statuscodes	Beschreibung
PUT	Path • indexClass	204 – ok. 404 – Klasse nicht gefunden.	Löscht den Index für die übergebene IIndexTypeConf-Implementation und legt diesen anschliessend neu an. (Reindexierung).

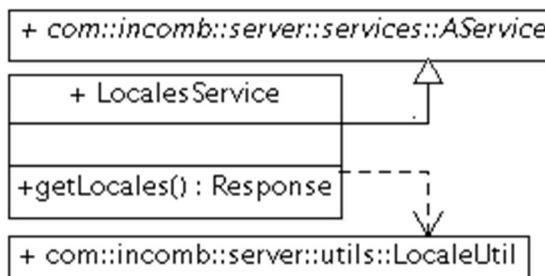


5.2.10.3 Locales

Package com.incomb.server.services.locales.

/api/locales

Methode	Parameter	Statuscodes	Beschreibung
GET		200 – ok.	Gibt alle unterstützten Sprachen zurück.

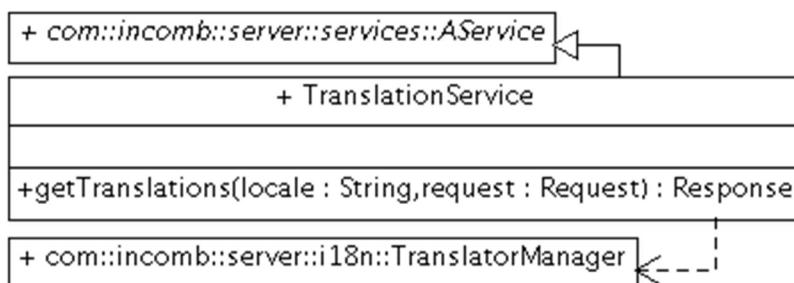


5.2.10.4 Translations

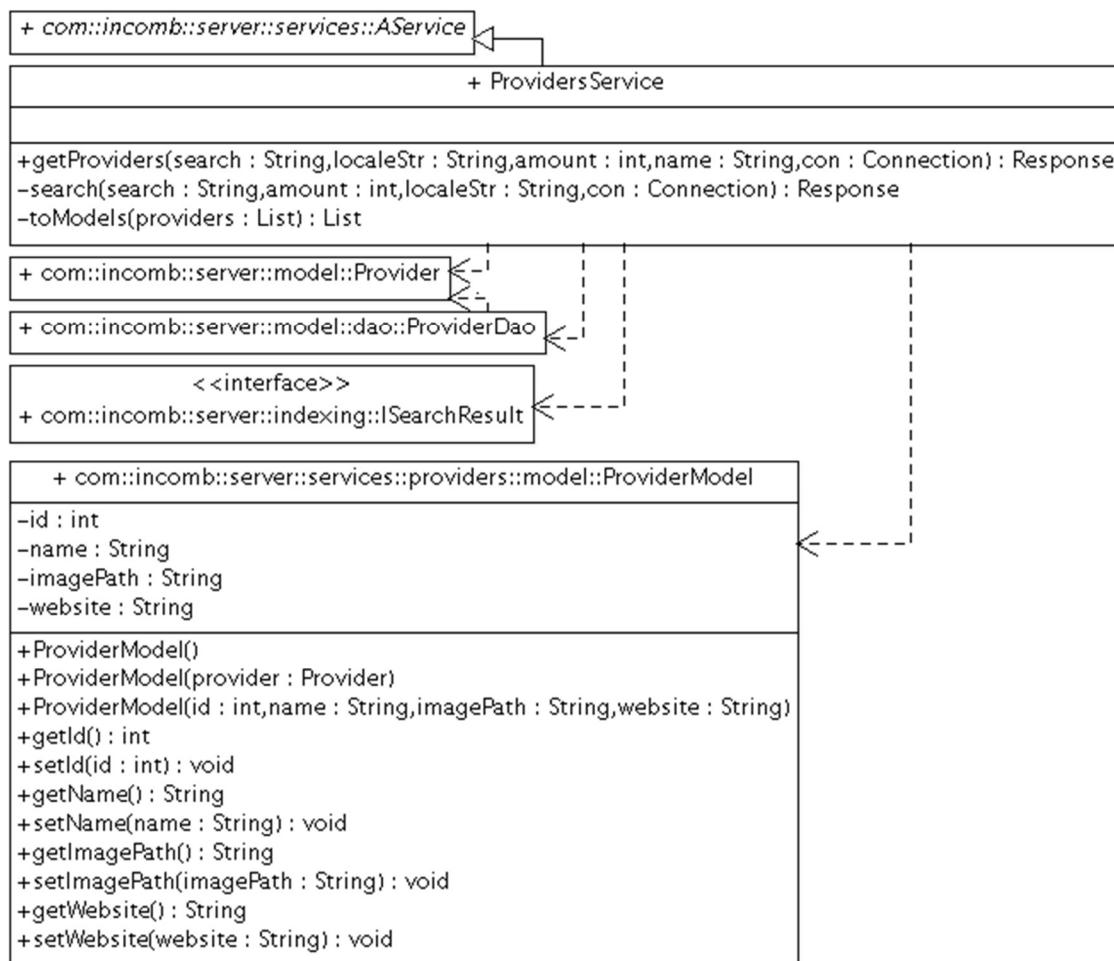
Package com.incomb.server.services.translations.

/api/translations/{locale}

Methode	Parameter	Statuscodes	Beschreibung
GET	Path • locale	200 – ok.	Gibt das JSON-File mit den Übersetzungen für die angeforderte Sprache zurück. Wenn diese Sprache nicht unterstützt wird, wird das File für die Standardsprache (en) zurückgegeben.



5.2.10.5 Anbieter

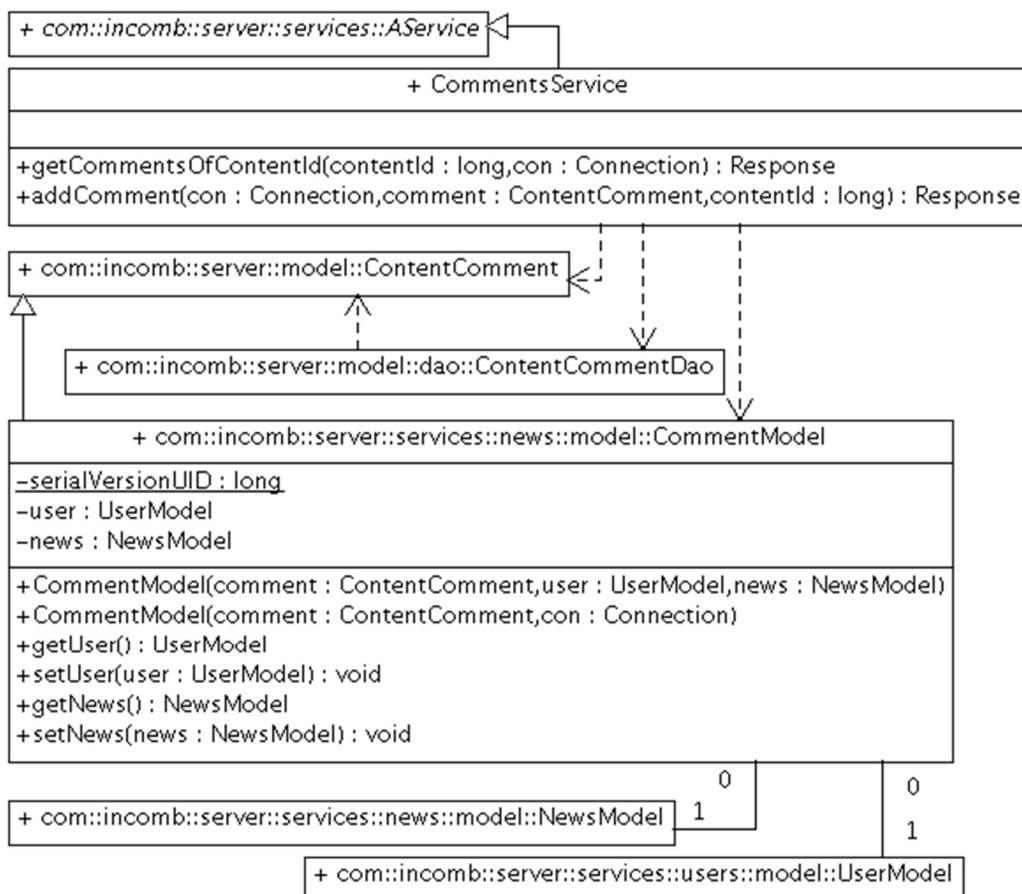


5.2.10.6 Nachrichten

5.2.10.6.1 Kommentare

Package com.incomb.server.services.news

/api/news/{contentId}/comments			
Methode	Parameter	Statuscodes	Beschreibung
GET	contentId	200 – ok.	Gibt eine Liste von Kommentaren für den Newsartikel mit der ContentId zurück
POST	contentId, ContentComment mit userId, contentId und der Kommentar (comment)	200 –ok. 400 – bad Request	Speichert einen neuen Kommentar auf dem Newsartikel.

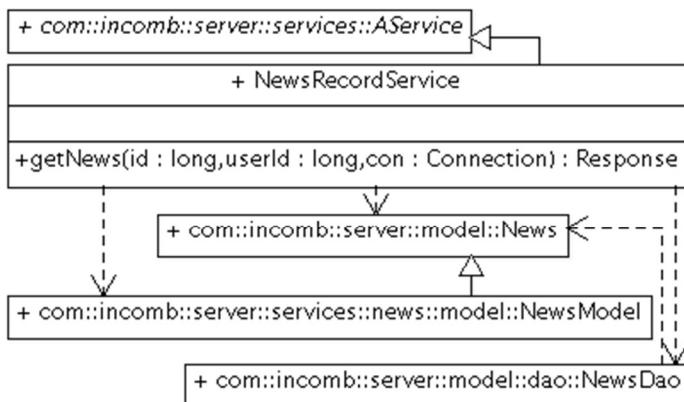


5.2.10.6.2 Nachrichten

Package com.incomb.server.services.news

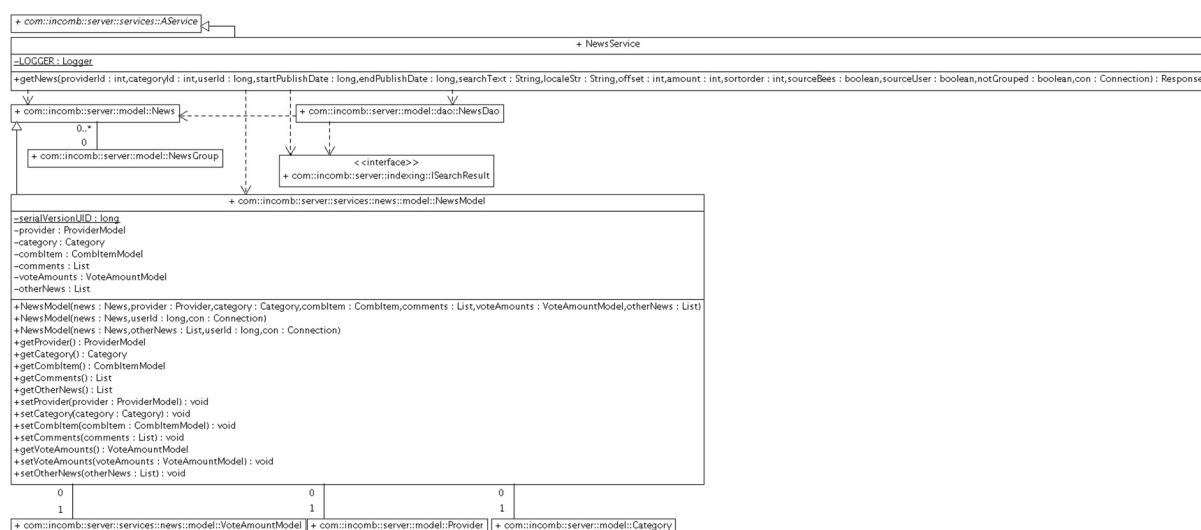
/api/news/{contentId}

Methode	Parameter	Statuscodes	Beschreibung
GET	contentId	200 – ok.	Gibt den Newsartikel mit der id contentId zurück



/api/news

Methode	Parameter	Statuscodes	Beschreibung
GET	<ul style="list-style-type: none">• providerId• categoryId• userId• startPublishDate• endPublishDate• searchText• locale• offset• amount• sortorder• sourceBees• sourceUser• groupNews	200 – ok.	Gibt eine Liste der gefundenen Nachrichten zurück



5.2.10.6.3 Ins und Combs

Package com.incomb.server.services.news

/api/news/{contentId}/ins

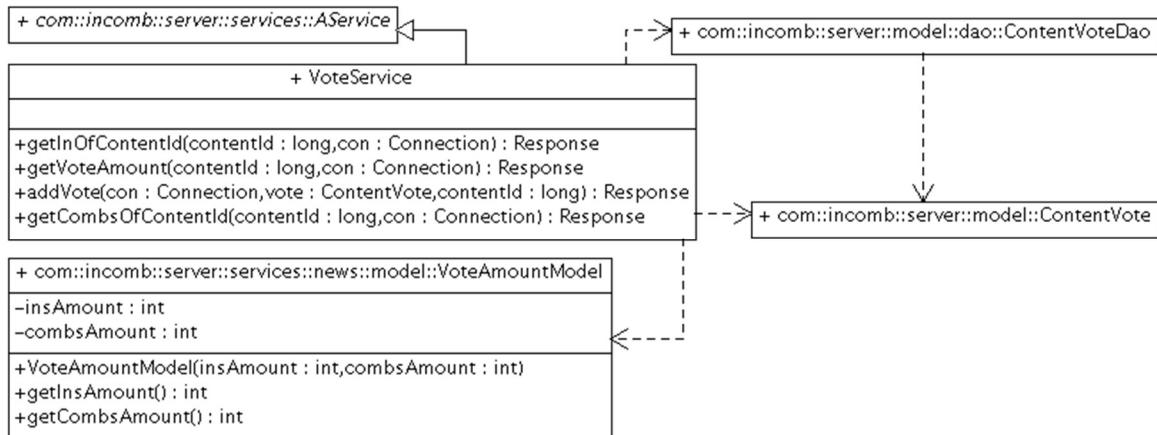
Methode	Parameter	Statuscodes	Beschreibung
GET	contentId	200 – ok.	Gibt eine Liste von Bees zurück, die ein „in“ bei Newsartikel mit id <code>contentId</code> gegeben haben

`/api/news/{contentId}/votes`

Methode	Parameter	Statuscodes	Beschreibung
GET	contentId	200 – ok.	Gibt die Anzahl der ins uns combs für den Newsartikel mit id contentId.
POST	<ul style="list-style-type: none"> • userId • contentId • voteDate • up 	200 –ok 400 – Bad Request	Fügt ein In oder ein Comb für den Newsartikel mit id contentId.

/api/news/{contentId}/combs

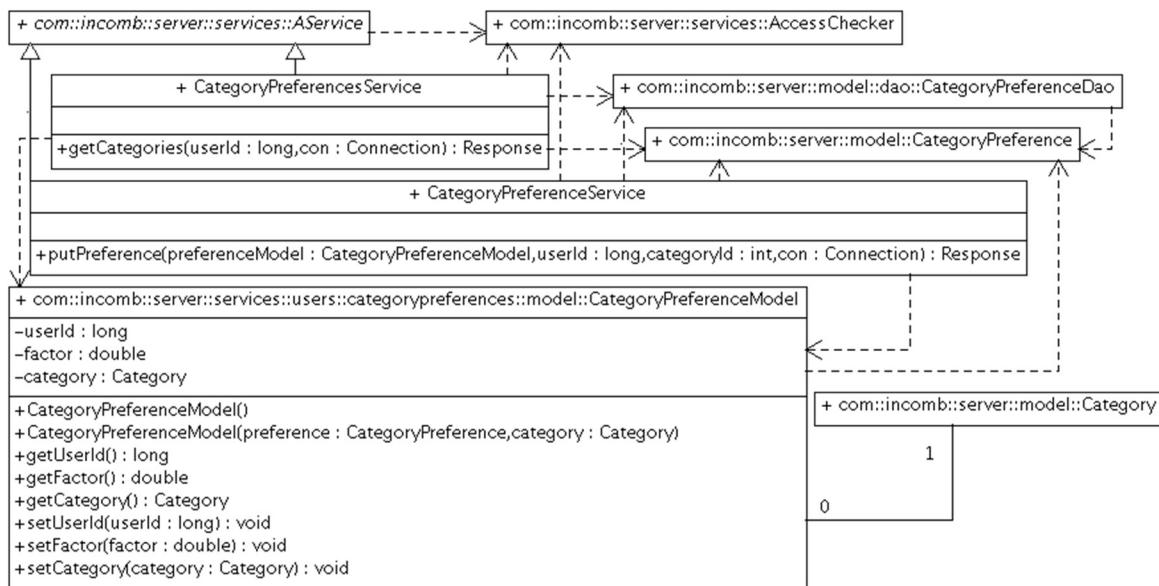
Methode	Parameter	Statuscodes	Beschreibung
GET	contentId	200 – ok.	Gibt eine Liste von Bees zurück, die ein „comb“ bei Newsartikel mit id <code>contentId</code> gegeben haben

**5.2.10.7 Benutzer****5.2.10.7.1 Kategorienpräferenzen**Package `com.incomb.server.services.users.categorypreferences`**/api /users/{userId}/categories**

Methode	Parameter	Statuscodes	Beschreibung
GET	userId	200 – ok. 401 – Not Authorized	Gibt die Kategoriepräferenzen für den gewünschten Benutzer mit id <code>userId</code> zurück. Die <code>userId</code> muss hierbei mit dem aktuell eingeloggten Benutzer übereinstimmen.

/api /users/{userId}/categories/{categoryId}

Methode	Parameter	Statuscodes	Beschreibung
PUT	userId categoryId userId factor category	200 – ok. 401 – Not Authorized	Updated die aktuellen Kategorie Präferenzen mit den neuen. Die <code>userId</code> muss hierbei mit dem aktuell eingeloggten Benutzer übereinstimmen.



5.2.10.7.2 Meine Comb Einträge

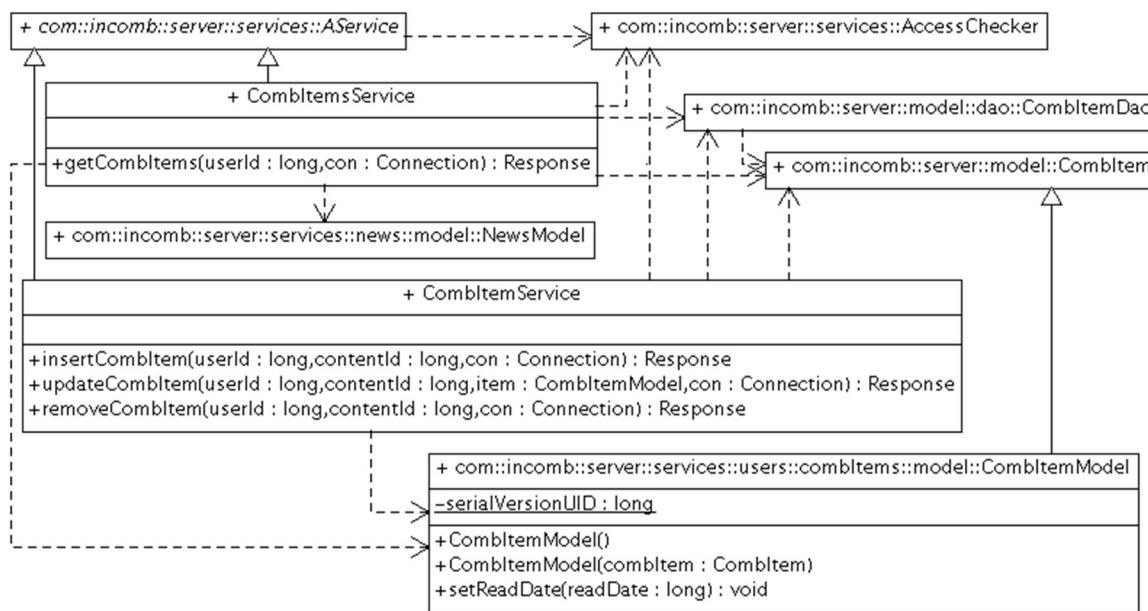
Package com.incomb.server.services.users.combItems

/api /users/{userId}/combItems/{contentId}

Methode	Parameter	Statuscodes	Beschreibung
POST	userId, contentId	200 – ok. 401 – Not Authorized	Fügt den Newsartikel mit der id contentId in die Comb des Benutzers mit der id userId. Die userId muss hierbei mit dem aktuell eingeloggten Benutzer übereinstimmen.
PUT	userId contentId addDate readDate	200 – ok. 400- Bad Request 401 – Not Authorized	Aktualisiert den Newsartikel mit der id contentId in der Comb des Benutzers mit der id userId. Die userId muss hierbei mit dem aktuell eingeloggten Benutzer übereinstimmen.
DELETE	userId contentId addDate readDate	200 – ok. 401 – Not Authorized	Löscht den Newsartikel mit der id contentId in der Comb des Benutzers mit der id userId. Die userId muss hierbei mit dem aktuell eingeloggten Benutzer übereinstimmen.

/api /users/{userId}/combItems

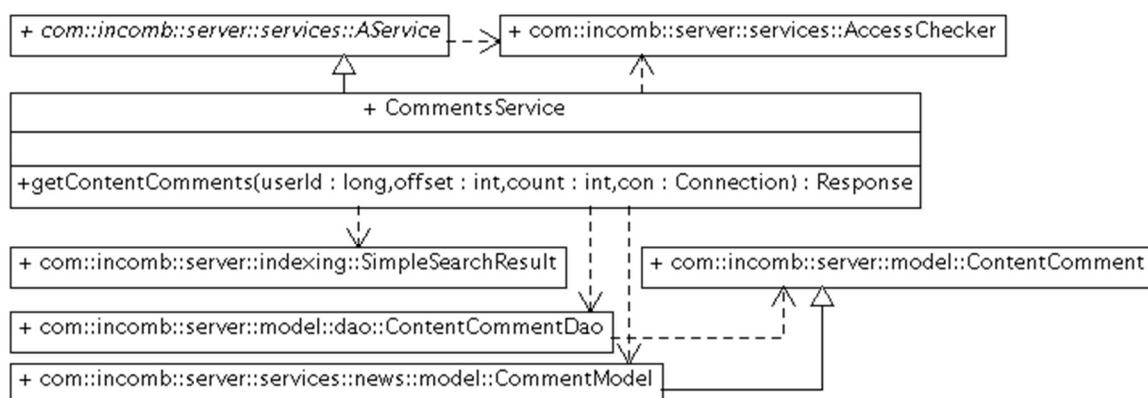
Methode	Parameter	Statuscodes	Beschreibung
GET	userId	200 – ok. 401 – Not Authorized	Gibt eine Liste aller Newsartikel in der Comb des Benutzers mit der id userid Die userId muss hierbei mit dem aktuell eingeloggten Benutzer übereinstimmen.



5.2.10.7.3 Kommentare

Package com.incomb.server.services.users.comments

/api /users/{userId}/comments			
Methode	Parameter	Statuscodes	Beschreibung
GET	userId, offset, count	200 – ok. 401 – Not Authorized	Gibt eine Liste von Suchresultaten(Kommentare des Benutzers) für die angegebenen Parameter zurück. Das Profil des Benutzers mit id userId darf dabei nicht privat sein.



5.2.10.7.4 Ins und Combs

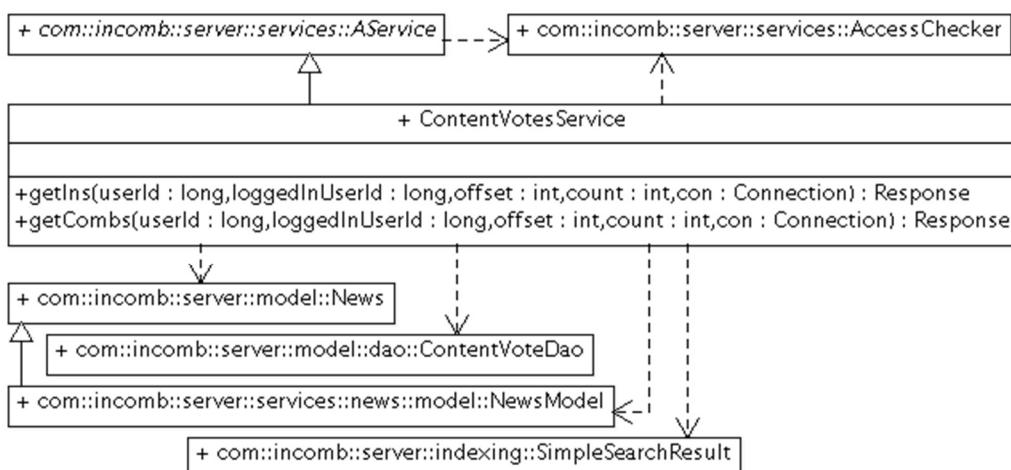
Package com.incomb.server.services.users.contentVotes

/api /users/{userId}/ins

Methode	Parameter	Statuscodes	Beschreibung
GET	userId loggedInUserId offset count	200 – ok. 401 – Not Authorized	Gibt eine Liste von Suchresultaten(Ins des Benutzers) für die angegebenen Parameter zurück. Das Profil des Benutzers mit id userId darf dabei nicht privat sein.

/api /users/{userId}/combs

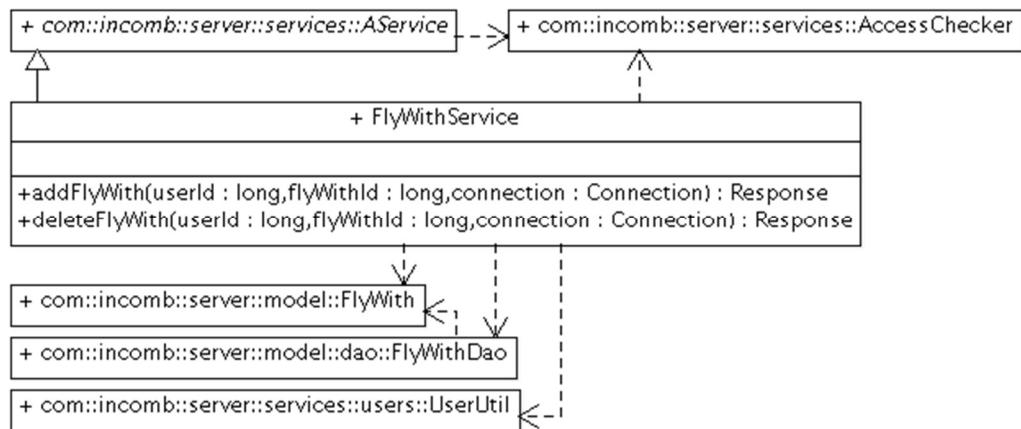
Methode	Parameter	Statuscodes	Beschreibung
GET	userId loggedInUserId offset count	200 – ok. 401 – Not Authorized	Gibt eine Liste von Suchresultaten(Combs des Benutzers) für die angegebenen Parameter zurück. Das Profil des Benutzers mit id userId darf dabei nicht privat sein.

**5.2.10.7.5 Fly-Withs**

Package com.incomb.server.services.users.flyWiths

/api /users/{userId}/flyWiths/{flyWithId}

Methode	Parameter	Statuscodes	Beschreibung
POST	userId flyWithId	200 – ok. 401 – Not Authorized	Fügt einen neuen FlyWith für den Benutzer mit der id userId hinzu. Die userId muss hierbei mit dem aktuell eingeloggten Benutzer übereinstimmen.
DELETE	userId flyWithId	200 – ok. 401 – Not Authorized	Löscht einen FlyWith auf einem Benutzer mit der Id userId. Die userId muss hierbei mit dem aktuell eingeloggten Benutzer übereinstimmen.



5.2.10.7.6 Locales

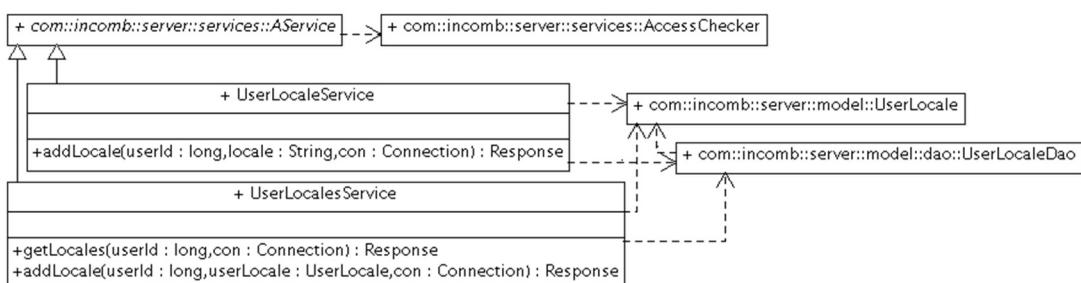
Package com.incomb.server.services.users.locales

/api /users/{userId}/locales/{locale}

Methode	Parameter	Statuscodes	Beschreibung
DELETE	userId locale	200 – ok. 401 – Not Authorized	Löscht eine Sprache des Benutzers mit der ID userId. Die userId muss hierbei mit dem aktuell eingeloggten Benutzer übereinstimmen.

/api /users/{userId}/locales

Methode	Parameter	Statuscodes	Beschreibung
GET	userId	200 – ok. 401 – Not Authorized	Gibt die Sprache des Benutzers mit der ID userId zurück. Die userId muss hierbei mit dem aktuell eingeloggten Benutzer übereinstimmen.
POST	userId userLocale	200 – ok. 401 – Not Authorized	Fügt dem Benutzer mit der ID userId eine neue Sprache hinzu. Die userId muss hierbei mit dem aktuell eingeloggten Benutzer übereinstimmen.



5.2.10.7.7 Tags

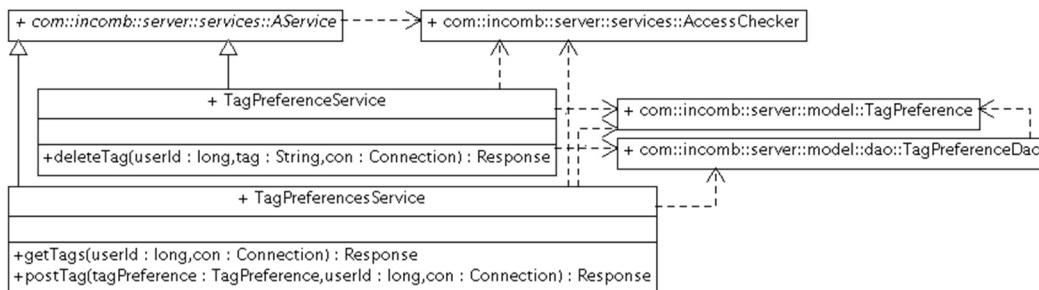
Package com.incomb.server.services.users.tagpreferences

/api /users/{userId}/tags/{tag}

Methode	Parameter	Statuscodes	Beschreibung
DELETE	userId tag	200 – ok. 401 – Not Authorized	Löscht eine Tag des Benutzers mit der ID userId. Die userId muss hierbei mit dem aktuell eingeloggten Benutzer übereinstimmen.

/api /users/{userId}/tags

Methode	Parameter	Statuscodes	Beschreibung
GET	userId	200 – ok. 401 – Not Authorized	Gibt alle Tags des Benutzers mit der ID userId zurück. Die userId muss hierbei mit dem aktuell eingeloggten Benutzer übereinstimmen.
POST	userId tag	200 – ok. 401 – Not Authorized	Fügt dem Benutzer mit der ID userId einen neuen Tag mit dem Wert von „tag“ hinzu. Die userId muss hierbei mit dem aktuell eingeloggten Benutzer übereinstimmen.

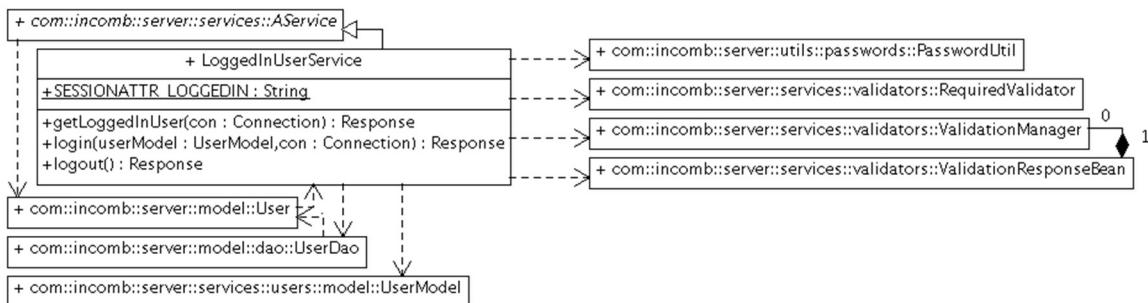


5.2.10.7.8 Login

Package com.incomb.server.services.users

/api/users/loggedIn

Methode	Parameter	Statuscodes	Beschreibung
GET		200 – ok. 404 – Not Found	Gibt den aktuell eingeloggten Benutzer zurück.
POST	Username Password	200 – ok 401 – Not Authorized	Loggt den Benutzer ein und falls erfolgreich, die Benutzerdaten
DELETE		200 – ok	Loggt den Benutzer aus



5.2.10.7.9 Benutzer

Package com.incomb.server.services.users

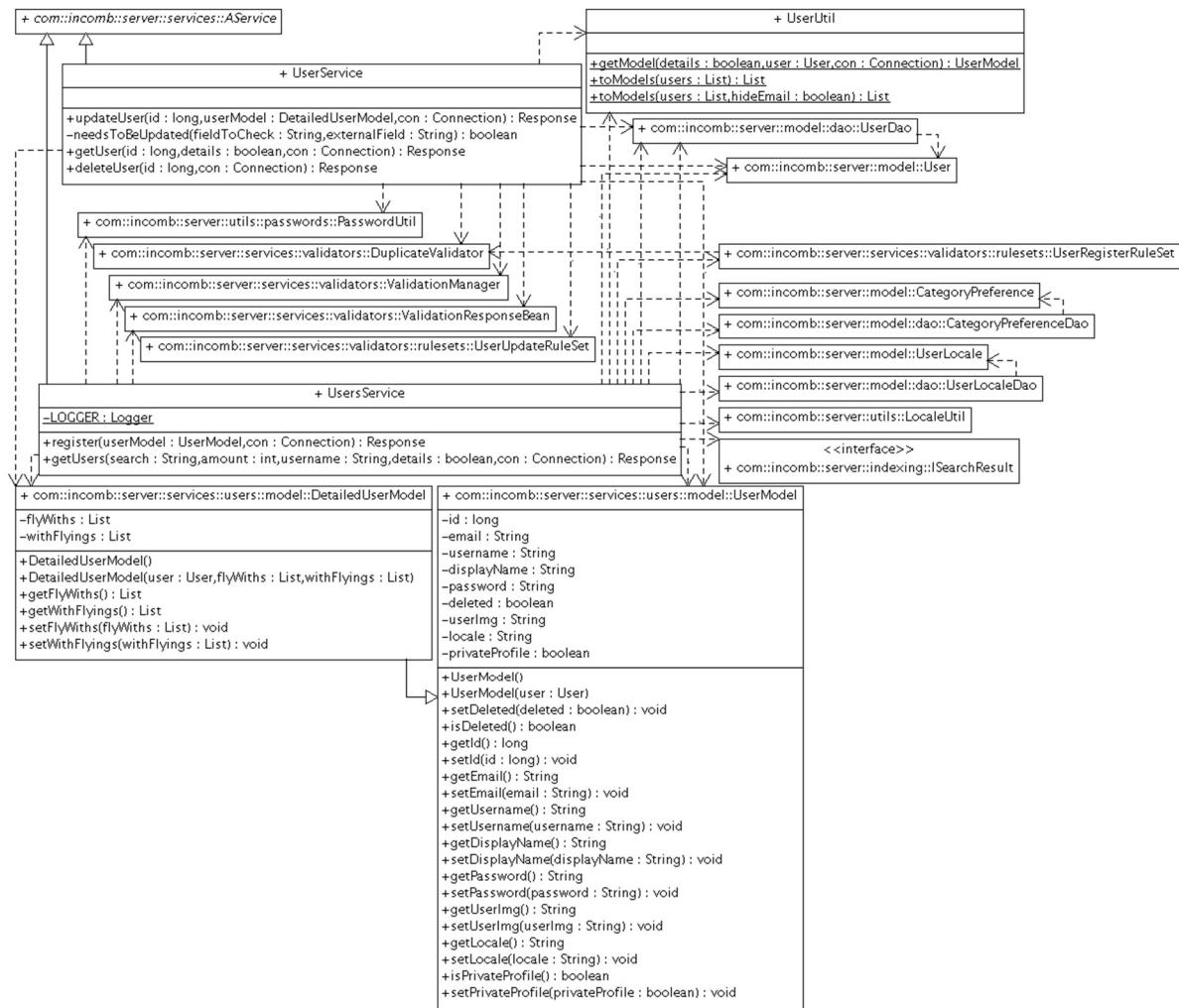
/api/users/{id}

Methode	Parameter	Statuscodes	Beschreibung
PUT	userId <ul style="list-style-type: none"> • List<UserModel> flyWiths • List<UserModel> withFlyings • id • email • username • displayName • password • deleted = false • userImg • locale • privateProfile 	200 – ok. 400 – Bad Request 401 – Not Authorized	Ändert die Benutzerdaten eines Benutzers mit der ID userId. Die userId muss hierbei mit dem aktuell eingeloggten Benutzer übereinstimmen.
GET	userId details	200 – ok. 400 – Bad Request 404 – Not Found	Gibt die Daten eines Benutzers zurück, falls der Benutzer eingeloggt ist, wird zusätzlich die E-Mail zurückgegeben. Falls der Benutzer gelöscht wurde, oder nicht existiert wird ein HTTP 404 zurückgegeben.
DELETE	userId	200 – ok. 400 – Bad Request 404 – Not Found	Löscht den Benutzer mit der id userId oder gibt 404 zurück falls dieser bereits gelöscht wurde, oder nicht existiert.

/api/users

Methode	Parameter	Statuscodes	Beschreibung
POST	<ul style="list-style-type: none"> • id • email • username • displayName • password • deleted 	200 – ok. 400 – Bad Request	Hiermit wird ein neuer Benutzer registriert. Es wird ein HTTP 400 Fehler geschickt, falls die Validierung der Daten fehlschlägt.

	<ul style="list-style-type: none"> • userImg • locale • privateProfile 		
GET	<ul style="list-style-type: none"> • search • amount • username • details 	200 – ok. 400 – Bad Request	Hiermit können Benutzer gesucht werden. Es wird eine Liste von Benutzern zurückgegeben.



5.3 Feinentwurf der Systemkomponenten (-elemente): JavaScript-Teil

Das Frontend wurde mit dem Framework Angular.js umgesetzt. Es ruft die HTML-Templates vom Server ab und rendert diese dann mit den Daten, welche es über die REST-Schnittstelle auf dem Server abruft.

Die REST-Schnittstelle ist im Modul resources (resources.js) mit verschiedenen Services abgebildet. Diese Services werden in den anderen Modulen verwendet um die Schnittstelle zu verwenden.

Der Code wurde in Module aufgeteilt. Jedes Modul enthält Funktionalität für einen bestimmten Themenbereich. Die Module wurden so gestaltet, dass sie in anderen Projekten wiederverwertet werden können. Nachfolgend die Beschreibung der Module.

5.3.1 categoryPreference

Datei: category-preference.js

5.3.1.1 categoryPreferencesCtrl

Dieser Controller ruft für den angemeldeten Benutzer seine Kategorie Präferenzen ab und legt diese in den Scope. Ausserdem stellt es auf jeder Präferenz eine save-Funktion zur Verfügung um diese auf dem Server zu speichern.

5.3.2 category

Datei: category.js

5.3.2.1 Route

Beim Aufstarten wird die Route /categories/{name} mit dem categoryCtrl auf dem router registriert.

5.3.2.2 categoryCtrl

Der Controller wird benötigt, um auf der Themenbereich-Ansicht die Nachrichten darzustellen. Es wird anhand des Namens in der URL die Kategorie vom Server abgerufen und in den Scope gelegt. Danach wird der newsFilterService initialisiert, um die Nachrichten auch filtern zu können. Bei einer Änderung im Filter und zu Beginn werden die Nachrichten über den newsLoader abgerufen.

5.3.3 comb

Datei: comb.js

5.3.3.1 Route

Beim Aufstarten wird die Route /comb mit dem combCtrl auf dem router registriert.

5.3.3.2 combCtrl

Der Controller wird benötigt, um die Nachrichten anzuzeigen, welche der Benutzer in seiner Comb hat. Diese werden dazu über die combItemsResource abgerufen und durch den newsLoader mit allen nötigen Funktionen abgefüllt. Zuletzt werden diese in den Scope gelegt.

5.3.4 errors

Datei: errors.js

5.3.4.1 Route

Beim Aufstarten werden die Routes /404 mit dem error404Ctrl und /500 mit dem error500Ctrl auf dem router registriert.

5.3.4.2 error404Ctrl

Der Controller ändert den Fenstertitel mit den pageTitle-Service.

5.3.4.3 error500Ctrl

Der Controller ändert den Fenstertitel mit den pageTitle-Service.

5.3.4.4 500checkInterceptor

Dieser fängt alle Responses vom Server ab und prüft den Statuscode, ob dieser 500 ist. Wenn ja, wird der Benutzer auf einen Fehlerseite (/500) geleitet.

5.3.5 filter

Datei: filter.js

5.3.5.1 newsFilterCtrl

Dieser Controller wird benötigt um das Filter-Formular vorzubereiten und die Dropdown-Boxen abzufüllen. Es werden alle Kategorien und Anbieter abgerufen und in den Scope gelegt.

Ausserdem speichert er bei jeder Filter-Veränderung die Filter-Parameter in die URL, damit beim Neu laden der Seite automatisch wieder vorausgewählt sind. Er wirft zusätzlich jeweils den filterChanged-Event, damit die Filter angewendet werden können.

5.3.5.2 newsFilterService

Er bietet eine init-Funktion an um den Filter zu initialisieren. Er lädt die Filter-Parameter aus der URL in den Scope und blendet einzelne Filter nach Bedarf aus.

5.3.6 forms

Datei: forms.js

5.3.6.1 validateError

Diese Directive kann als Attribut verwendet werden, damit die Fehlermeldung für das im Attribut angegebene Feld im Element dargestellt wird.

5.3.6.2 validateRetype

Diese Directive kann als Attribut auf einem Eingabefeld definiert werden. Es wird dann der Inhalt dieses Eingabefeldes mit dem Inhalt des Eingabefeldes, welches im Attribut angegeben ist verglichen.

5.3.7 home

Datei: home.js

5.3.7.1 Route

Beim Aufstarten wird die Route / mit dem homeCtrl beim router registriert.

5.3.7.2 homeCtrl

Dieser Controller lädt bei einer Änderung des Filters und zu Beginn über den newsLoader die Nachrichten.

5.3.8 incomb

Datei: incomb.js

Dies ist das Hauptmodul der Webseite. Es hat Abhängigkeiten auf alle verwendeten Module.

5.3.8.1 pageTitle

Dieser Service kann verwendet werden um den Seitentitel zu ändern.

5.3.8.2 localeUtil

Mit diesem Service kann die im Browser eingestellte Sprache abgefragt werden.

5.3.8.3 plain

Mit diesem Filter werden alle Tags aus einem Text entfernt.

5.3.8.4 urlencode

Mit diesem Filter kann ein Teil einer URL enkodiert werden, so dass es eine valide URL ergibt.

5.3.8.5 afterRender

Der JavaScript-Ausdruck in diesem Attribut wird ausgeführt, sobald Angular.js das Tag gerendert hat.

5.3.9 nav

Datei: nav.js

Dieses Modul wird für die Navigation benötigt.

5.3.9.1 navCtrl

Dieser Controller lädt alle Themenbereiche und legt diese in den Scope, damit die Navigation aufgebaut werden kann.

5.3.10 news

Datei: news.js

Dieses Modul enthält Funktionalität um Nachrichten darzustellen.

5.3.10.1 Route

Beim Aufstarten wird die Route /news/{id} mit dem newsCtrl beim router registriert.

5.3.10.2 newsCtrl

Dieser Controller lädt die Nachricht vom Server, welche in der URL angegeben wurde und legt diese in den Scope, damit diese angezeigt werden kann.

5.3.10.3 newsLoader

Nachrichten, welche über die normale newsResource bezogen werden können, werden mit Vorteil über diesen Service abgerufen. Er bietet die Funktionalität an weitere Nachrichten einfach nachzuladen.

Der Service kann auch verwendet werden um Funktionen auf einer Nachricht bereitzustellen, um Ins und Combs zu geben, Kommentare zu schreiben, in die Comb zu legen und weitere Nachrichten spezifische Aktionen durchzuführen.

5.3.11 profile

Datei: profile.js

Dieses Modul wird verwendet, um eine Profilansicht darzustellen.

5.3.11.1 Route

Beim Aufstarten wird die Route /{username} mit dem profileCtrl beim router registriert.

5.3.11.2 profileCtrl

Dieser Controller lädt den Benutzer, der in der URL angegeben wurde und legt diesen in den Scope. Außerdem lädt er dessen Ins, Combs und Kommentare und legt diese ebenfalls in den Scope und bietet Funktionen an um weitere zu laden. Ebenfalls stellt er Funktionen zur Verfügung um mit dem Benutzer mitzufliegen oder um von ihm wegzufliegen.

5.3.12 provider

Datei: provider.js

5.3.12.1 Route

Beim Aufstarten wird die Route /providers/{name} mit dem providerCtrl auf dem router registriert.

5.3.12.2 providerCtrl

Der Controller wird benötigt, um auf der Anbieter-Ansicht die Nachrichten darzustellen. Es wird anhand des Namens in der URL der Anbieter vom Server abgerufen und in den Scope gelegt.

Danach wird der newsFilterService initialisiert, um die Nachrichten auch filtern zu können. Bei einer Änderung im Filter und zu Beginn werden die Nachrichten über den newsLoader abgerufen.

5.3.13 read

Datei: read.js

Dieses Modul wird für die Lese-Ansicht benötigt.

5.3.13.1 Route

Beim Aufstarten wird die Route /read mit dem readCtrl auf dem router registriert.

5.3.13.2 readCtrl

Dieser Controller ruft anhand der Query-Parameter die Nachrichten mit dem newsLoader ab und markiert die Nachricht mit der ID, die als Query-Parameter selectedId hat als aktiv.

5.3.14 resources

Datei: resources.js

Diese Datei abstrahiert die REST-Schnittstelle auf dem Server. Für jede URL wird ein Service geschaffen, der eine \$resource-Instanz zurückgibt. Dieses Objekt enthält Funktionen mit dem Namen der HTTP-Methode, welche dann aufgerufen werden können um einen Aufruf an die Schnittstelle zu machen.

5.3.15 router

Datei: router.js

Dieses Modul ist eine Abstraktion des route-Moduls von Angular.js. Es bietet die Möglichkeit an URLs zu generieren. Dazu werden alle Routes mit einem zusätzlichen Namen registriert, der für die Generierung angegeben werden muss. Für die Generierung steht ein Service und ein Filter zur Verfügung. Außerdem lässt sich mit einem Filter herausfinden, ob die aktuelle Route die ist, die dem Filter übergeben wurde. So kann zum Beispiel der aktive Navigationspunkt markiert werden.

5.3.16 search

Datei: search.js

Dieses Modul implementiert die Suche.

5.3.16.1 searchCtrl

Dieser Controller übernimmt die Suchfunktionalität. Sobald etwas in das Suchfeld eingegeben wird, wird die Schnittstelle aufgerufen, um die Suche auszuführen. Jeder Inhaltstyp (Anbieter, Themenbereich, Benutzer, Nachricht) hat eine eigene Resource. Deshalb gibt es für jeden Typ einen eigenen Service, der die Schnittstelle mit den benötigten Parametern aufruft und abstrahiert. Diese Services werden vom searchCtrl verwendet und die Resultate in den Scope gelegt.

5.3.17 settings

Datei: settings.js

Dieses Modul ist für die Einstellungsseite verantwortlich, und stellt dort zum Beispiel ein User Objekt zur Verfügung

5.3.18 Slider

Datei: slider.js

Dieses Modul sorgt für den Slider mit den With-Flyings, der auf dem Profil angezeigt wird.

5.3.19 Styles

Datei: styles.js

Dieses Modul enthält Logik für das Styling von InComb. Wie zum Beispiel die fixierte Kategorienbar auf der Hauptseite.

5.3.20 tagPreference

Datei: tag-preference.js

5.3.20.1 tagPreferenceCtrl

Mit diesem Controller können auf dem Benutzerprofil Tags hinzugefügt und entfernt werden.

5.3.21 userLocale

Datei: user-locale.js

5.3.21.1 userLocalesCtrl

Mit diesem Controller können auf dem Benutzerprofil Sprachen hinzugefügt und entfernt werden.

5.3.22 userSettings

Datei: user-settings.js

5.3.22.1 userSettingsCtrl

Dieser Controller kann Einstellungen des Benutzers, wie zum Beispiel der Benutzername aktualisieren und anpassen.

5.3.23 user

Datei: user.js

Dieses Modul bearbeitet alles, was mit Benutzern zu tun hat.

5.3.23.1 getUser

Dieser Provider wird für das Login verwendet. Nachher hat man überall das user Objekt verfügbar.

5.3.23.2 userInitializer

Wird initial beim eingeloggten User ausgeführt. Der Service sorgt dafür, dass flyWith und flyAway Anfragen gesendet werden können.

5.3.23.3 loginCtrl

Dieser Controller sorgt dafür, dass Login-Eingaben vom Server validiert werden, und dann im Frontend dargestellt werden. Ausserdem werden auch Benutzer mit diesem Controller auch eingeloggt.

5.3.23.4 registerCtrl

Dieser Controller sorgt dafür, dass Registrier-Eingaben vom Server validiert werden, und dann im Frontend dargestellt werden. Ausserdem werden auch Benutzer mit diesem Controller auch registriert.

5.3.23.5 logoutCtrl

Dieser Controller sorgt dafür, den Benutzer auszuloggen.

5.3.23.6 authCheckInterceptor

Fängt alle HTTP 401 Fehler ab, welche bedeuten, dass der Benutzer nicht mehr angemeldet ist und leitet dann auf die Login Seite weiter.

5.4 Physikalische Datenbeschreibung (Datenstrukturtypen)

Die Datenstrukturen wurden bereits im Datenbankschema beschrieben.

5.5 Sicherheit

Die Passwörter der Benutzer werden in der Datenbank zusammen mit einem für jeden Benutzer neu generiertem Salt verschlüsselt gespeichert. Es wird das nicht entschlüsselbare Verfahren PBKDF2WithHmacSHA1 verwendet. Aus dem generierten Passwort-Hash lassen sich die Passwörter nicht mehr zurückbilden.

Das Datenbankframework jooq verwendet Prepared Statements für die Abfragen auf die Datenbank. Durch dieses Verfahren kann SQL Injection ausgeschlossen werden, da die eigentlichen (möglicherweise gefährlichen) Werte separat an die Datenbank übermittelt und nicht direkt in das SQL-Skript eingefügt werden.

Beim Einlesen der RSS-Feeds werden sämtliche HTML-Tags (bis auf
-Tags) entfernt, damit nicht allfällige <script>-Tags vom Browser bei der Anzeige der Nachrichten ausgeführt werden.

Ausserdem werden sämtliche HTML-Tags durch Angular.js nicht ausgeführt, sondern als HTML-Code angezeigt. Soll HTML-Code ausgeführt werden, so muss dies explizit konfiguriert werden.

5.6 Anforderungen an die Entwicklungsumgebung

Zur Umsetzung des Projekts wird eine Java-Entwicklungsumgebung benötigt, die das Ausführen als Tomcat-Anwendung unterstützt. Ebenfalls sollte diese auf das SVN-Repository zugreifen und auch dort Änderungen speichern können. Mit Eclipse konnte eine passende Lösung gefunden werden, mit der das gesamte Team bereits Erfahrungen hatte sammeln können. InComb wurde als Dynamic Web Project erstellt und ins SVN-Repository aufgenommen.