

# SOFTWAREENTWICKLUNGSPRAKTIKUM

## MOVIE GENIE

Software-Entwicklungspraktikum (SEP)  
Sommersemester 2012

### F e i n e n t w u r f



Auftraggeber  
Technische Universität Braunschweig  
Institut für Informationssysteme  
Mühlenpfordtstraße 23, 2.OG  
D-38106 Braunschweig

Betreuer: Silviu Homocanu, Dr. Christoph Lofi

Auftragnehmer:

Name	E-Mail-Adresse
André-Dominik Müller	andre-dominik.mueller@tu-braunschweig.de
Frederik Kanning	f.kanning@tu-braunschweig.de
Jasper Bergner	j.bergner@tu-braunschweig.de
Jewgeni Rose	j.rose@tu-braunschweig.de
Martyna Zurek	m.zurek@tu-braunschweig.de
Patrick Hill	p.hill@tu-braunschweig.de
Philipp Tresp	p.tresp@tu-braunschweig.de
Tatiana Reiß	tatiana.reiss@rocketmail.com

Braunschweig, 27.06.2012

## Versionsübersicht

Version	Datum	Autor	Status	Kommentar
0.1	14.06.12	siehe Auftragnehmer	in Bearbeitung	Einfügen der Kapitel
0.2	16.06.12	André-Dominik Müller, Tatiana Reiß	in Bearbeitung	Einleitung eingefügt
0.3	16.06.12	Jewgeni Rose	in Bearbeitung	Kapitel 3 bearbeitet
0.4	17.06.12	Frederik Kanning	in Bearbeitung	Kapitel 3, Kapitel 5 bearbeitet
0.5	18.06.12	Jasper Bergner, Patrick Hill	in Bearbeitung	Kapitel 2 bearbeitet
0.6	19.06.12	Philipp Tresp, Martina Zurek	in Bearbeitung	Kapitel 4 bearbeitet
0.7	21.06.12	siehe Auftragnehmer	in Bearbeitung	Abgabe Betreuer
1.0	21.06.12	siehe Auftragnehmer	in Bearbeitung	Abgabe SSE

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>5</b>
<b>2</b>	<b>Erfüllung der Kriterien</b>	<b>8</b>
2.1	Musskriterien . . . . .	8
2.2	Wunschkriterien . . . . .	10
2.3	Abgrenzungskriterien . . . . .	10
<b>3</b>	<b>Implementierungsentwurf</b>	<b>12</b>
3.1	Gesamtsystem . . . . .	12
3.2	Implementierung von Komponente /K10/: Movie Genie: . . . . .	14
3.2.1	Klassendiagramm . . . . .	14
3.2.2	Erläuterung . . . . .	15
3.3	Implementierung von Komponente /K20/: Movie Search: . . . . .	20
3.3.1	Klassendiagramm . . . . .	20
3.3.2	Erläuterung . . . . .	22
3.4	Implementierung von Komponente /K30/: Text Analysis: . . . . .	25
3.4.1	Klassendiagramm . . . . .	27
3.4.2	Erläuterung . . . . .	28
3.5	Implementierung von Komponente /K40/: User Profile: . . . . .	30
3.5.1	Klassendiagramm . . . . .	31
3.5.2	Erläuterung . . . . .	31
3.6	Implementierung von Komponente /K50/: Feature Index: . . . . .	32
3.6.1	Klassendiagramm . . . . .	33
3.6.2	Erläuterung . . . . .	33
<b>4</b>	<b>Datenmodell</b>	<b>36</b>
4.1	Diagramm . . . . .	37
4.2	Erläuterung . . . . .	39
<b>5</b>	<b>Serverkonfiguration</b>	<b>42</b>

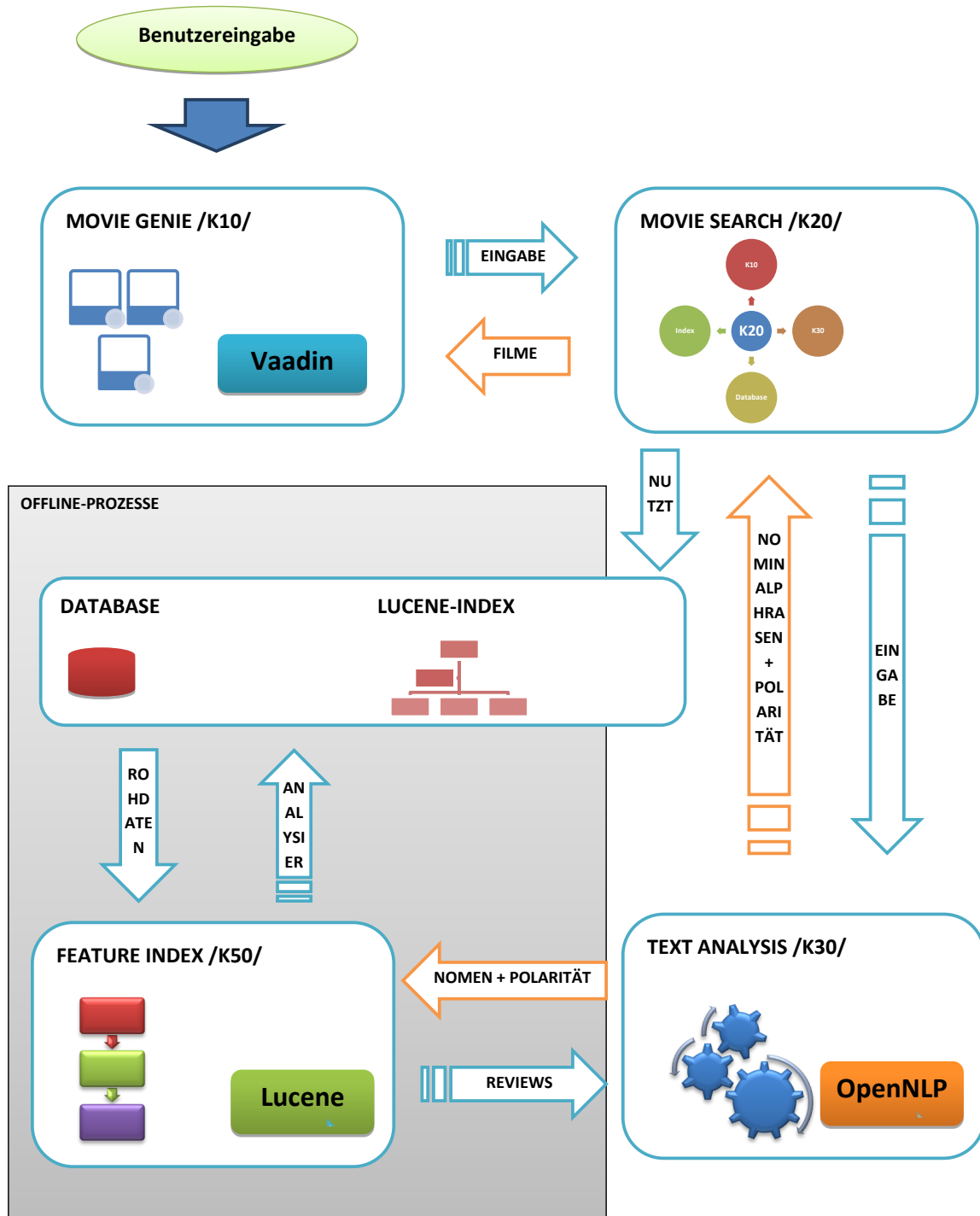
## Abbildungsverzeichnis

1.1	Funktionsablauf . . . . .	6
2.1	Beispiel-Dialog . . . . .	9
3.1	Komponentendiagramm . . . . .	13
3.2	Klassendiagramm GUI . . . . .	14
3.3	Menu . . . . .	17
3.4	AdvancedSearch . . . . .	18
3.5	Schematische Darstellung ResultPanel . . . . .	19
3.6	Schematische Darstellung MovieSite . . . . .	20
3.7	Klassendiagramm MovieSearch . . . . .	21
3.8	Klassendiagramm Text-Analysis . . . . .	27
3.9	Klassendiagramm UserProfile . . . . .	31
3.10	Klassendiagramm Feature Index . . . . .	33
4.1	Klassendiagramm der Datenbank . . . . .	38
4.2	Klassendiagramm User . . . . .	39

# 1 Einleitung

Der Feinentwurf gibt eine strukturierte und detaillierte Darstellung aller Implementierungsdetails des Softwareprojektes Movie Genie wieder. Das Dokument verfeinert zudem die im Grobentwurf gezeigten Analysemodelle. In Kapitel 2 werden noch einmal die im Pflichtenheft festgelegten Muss-, Wunsch- und Abgrenzungskriterien aufgegriffen und deren Umsetzung beschrieben. Speziell werden zu den Abgrenzungskriterien mögliche Lösungswege aufgezeigt, die aber in diesem Projektumfang nicht weiter realisiert werden müssen. Pakete, Attribute sowie die Umsetzung der einzelnen Komponenten mit ihren Klassen und Methoden werden in Kapitel 3, dem Implementierungsentwurf, beschrieben. Die Gesamtsicht zeigt zunächst das Gesamtsystem des Softwareprojektes mit den unterschiedlichen Komponenten aus dem Grobentwurf. In den Unterkapiteln werden dann die Komponenten näher beschrieben und die Umsetzung mithilfe von Klassendiagrammen erklärt. Dazu werden auch die unterschiedlichen Bibliotheken (OpenNLP, Lucene und Vaadin), die in den verschiedenen Komponenten zum Einsatz kommen, näher erläutert. Kapitel 4 zeigt das verwendete Datenmodell, welches durch ein Klassendiagramm veranschaulicht wird. Der Aufbau und die Struktur der verwendeten Datenbank, sowie die Beziehung zwischen den verschiedenen Entitäten, soll dadurch verdeutlicht werden. Zudem werden sämtliche Attribute der Entitäten sowie die Schlüsselattribute, durch die die Entitäten in Verbindung stehen, dargestellt. Die anschließende Tabelle in Punkt 4.2 greift die Entitäten auf und beschreibt deren Beziehungen genauer. Abschließend wird in Kapitel 5 die Konfiguration des Servers mit sämtlichen Konfigurationsdetails beschrieben und erklärt.

Das nachfolgende Diagramm soll zunächst einen Überblick über den Funktionsablauf von Movie Genie geben:



Nachdem der Benutzer die Homepage von Movie Genie besucht und sich erfolgreich autorisiert hat, kann er eine Eingabe tätigen (Suchanfrage an Movie Genie). Die GUI stellt die einzelnen Anzeigeelemente dar und zeigt dem Benutzer den Dialog in Chatform mit Movie Genie an. Die Eingabe durch den Benutzer wird dann an die MovieSearch weitergeleitet. Die MovieSearch erhält dabei die Daten aus der Datenbank, um zum Beispiel nach scharfen Kriterien zu suchen, und die Daten aus dem Luceneindex. Der Luceneindex enthält sämtlich Daten, die aus den Reviews gewonnen wurden, also alle wichtigen Nomen, Adverbien, Adjektive und deren Polarität. (positiv, negativ oder neutral). Die Datenbank (Database) enthält alle Daten von IMDb, die sich direkt auf einen Film beziehen. Die TextAnalysis Komponente analysiert die Eingabe des Benutzers und filtert aus ihr die weichen Kriterien, also die, die sich auf Reviewdaten beziehen(z.B. "good plot") und die scharfen Kriterien(Schauspieler, Genre, Regisseur) heraus und übergibt sie der MovieSearch Komponente. Die scharfen Kriterien können direkt durch die MovieSearch Komponente als SQL-Abfrage auf die Daten von IMDb angewendet werden. Die unscharfen Kriterien werden mit den Reviews abgeglichen, welche im Voraus von der Feature Index-Komponente indiziert wurden. Das Ergebnis der Analyse wird dann an die GUI übergeben, welche dieses in Form einer Liste von Filmen ausgibt. Der Benutzer kann sich nun die Ergebnisse ansehen, diese bewerten, die Suche ergänzen (erweiterte Suche) oder auch zurücksetzen. Die Bewertungen sollen dann bei einer erweiterten Suche berücksichtigt werden.

## 2 Erfüllung der Kriterien

Im nachfolgenden Kapitel wird genauer auf die Erfüllung der einzelnen im Pflichtenheft angegebenen Kriterien eingegangen. In der Beschreibung wird jeweils explizit auf das bestimmte Kriterium im Pflichtenheft verwiesen.

### 2.1 Musskriterien

- /M100/ Passwortgeschützter Zugriff auf die Webseite
  - Diese Anforderung wird dadurch realisiert, dass die Webapplikation über ein einfaches Domainpasswort geschützt ist, um nicht autorisierte Zugriffe zu verhindern.
- /M200/ Das Verstehen von Anfragen in natürlicher Sprache
  - Das Verstehen von Anfragen wird mit Hilfe der Bibliothek "OpenNLP" verwirklicht. Die Bibliothek bietet Methoden zur Zerlegung der eingegebenen Query in Nominalphrasen und ähnliche grammatikalische Konstrukte. Diese Zerlegung geschieht hauptsächlich in der Text-Analysis-Komponente (/K30/, siehe Grobentwurf), die insbesondere durch die Klassen "SemanticParser", "Phrase", "Polarity", "StringTokenizer", "TimeRegex", "Context" und "SentiWordProcessor" repräsentiert wird. Die Komponente nimmt den eingegebenen Suchstring ("verbose query") entgegen und zerlegt ihn innerhalb der einzelnen Klassen in weiche und harte Kriterien, wobei sie unwichtige Wörter, die nicht zur Verarbeitung der Suche benötigt werden, herausfiltert. Adverbien wird zunächst eine Polarität zugewiesen ("positive" oder "negative"), um im nächsten Schritt die Reviewdaten auf übereinstimmende Polarität hinsichtlich der zu den Adverbien gehörenden Nomen zu überprüfen.
- /M300/ Sortiertes Auflisten von bis zu 50 der relevantesten Filme anhand der herausgefilterten Informationen aus der Suchanfrage.
  - Nachdem die Anfrage des Nutzers in der Text-Analysis-Komponente wie bereits beschrieben zerlegt wurde, setzt die Klasse "MovieSearch" die einzelnen Konstrukte zu einer SQL-Query zusammen und sendet diese an die Datenbank. Harte Kriterien wie "Genre", "Schauspieler" oder Jahreszahlen können direkt abgefragt werden, wobei weiche Kriterien wie "good plot" nur mithilfe der zugewiesenen Polarität und



weiterer Analyse der Reviewdaten in die Berechnung der Relevanz mit einbezogen werden können. Die vorliegenden Reviews der Benutzer der IMDb werden im Pre-processing ähnlich der Suchanfrage des Nutzers zerlegt und während der Laufzeit nur auf Übereinstimmung bezüglich der weichen Kriterien überprüft.

- /M400/ Zu jedem Film begründen, warum dieser dem Benutzer vorgeschlagen wird.
  - Zu jedem vorgeschlagenen Film wird dem Benutzer angezeigt, warum dieser Film in der Ergebnisliste auftaucht. Dies sieht zum Beispiel so aus, dass der Benutzer vom System die Information erhält, wie viele der Reviewer die weichen Kriterien (wie zum Beispiel "plot") prozentual ähnlich bewertet haben, wie dies in der Suchanfrage getan wurde. Beispielsweise könnte ein Benutzer nach "movies with a good plot" suchen. Die Begründung für die Anzeige eines Films könnte demnach lauten: "more than 80% of the users refer to the plot of this movie in a positive manner".
- /M500/ Der Benutzer kann einen Dialog mit dem Movie Genie führen, indem er seine bisherige Suchanfrage durch zusätzliche Informationen, d.h. noch mehr Sucheingaben und das Markieren von Filmen, immer weiter konkretisiert.
  - Die Suchanfrage des Benutzers wird umgehend durch den "Movie Genie" kommentiert. Hierdurch wird eine Art Chat initialisiert, wobei der Benutzer seine Anfrage durch weitere Restriktionen einschränken kann. Sollte die erste Iteration der Suche nicht den Vorstellungen des Benutzers entsprechen, kann er durch einfache Eingabe in das Suchfeld die weichen bzw. harten Kriterien seiner Suche näher bestimmen, was natürlich wiederum vom "Movie Genie" kommentiert wird. Beispiel:

**You: hey, i want movies with Tim Robbins from the 90's**  
**Movie Genie: Let me do some magic....**  
**You: a thriller please**  
**Movie Genie: Ok. Hang on a moment, I've got this!**

The image shows a simple web form with a rectangular text input field on the left and a rounded rectangular button labeled "Add" on the right.

Abbildung 2.1: Beispiel-Dialog

- /M600/ Undo- und Redo-Funktionalität bei Eingabe der Suchanfrage, sowie Zurücksetzen aller bisherigen Eingaben.

- Die Undo- und Redo-Funktionalität und die Reset-Funktion werden von der "MovieSearch"-Klasse bereitgestellt. Bei einem Klick auf den "Undo"-Button wird die letzte Iteration der Suche rückgängig gemacht, wobei der aktuelle Stand der Suchanfrage für eventuelle "Redo"-Intentionen des Benutzers gespeichert wird. Der Benutzer wird somit in die Lage versetzt, fälschlicherweise getätigte Eingaben rückgängig zu machen bzw. seine Suche in eine neue Richtung zu lenken. Bei einem Klick auf den "Reset"-Button werden sämtliche Eingaben gelöscht, sodass der Benutzer die Suche von vorne beginnen kann.
- /M700/ Bei unsinnigen Eingaben, wie "2§!", bittet Movie Genie den Benutzer um eine Neuformulierung seiner Anfrage.
  - Kann die Anfrage innerhalb der Text-Analysis-Komponente aufgrund unsinniger Eingaben nicht verarbeitet werden, wird dies erkannt und als Antwort des "Movie Genie" erscheint eine Bitte um Verfeinerung der Anfrage. Leere Eingaben werden ebenfalls abgefangen und in ähnlicher Weise durch den "Movie Genie" kommentiert.

## 2.2 Wunschkriterien

Die Erfüllung folgender Kriterien für das abzugebende Produkt wird angestrebt:

- /W100/ Das System soll sich an die Markierung der Filme vom Benutzer erinnern und sie beim gleichen Benutzer in einem erneuten Suchprozess automatisch einfließen lassen.
  - Ein Wunschkriterium ist es, die Bewertungen eines Benutzers ("like" bzw. "don't like") zu den jeweiligen Filmen zu speichern, um diese beim erneuten Besuch der Webseite automatisch in die Suche einzubinden. Dies soll mit Hilfe von Cookies realisiert werden, wobei einem Benutzer vom System eine ID zugewiesen wird, die zusammen mit den etwaigen Bewertungen in der Datenbank gespeichert wird.

## 2.3 Abgrenzungskriterien

Folgende Funktionalitäten werden nicht durch das Produkt, sondern wie folgt beschrieben anderweitig erfüllt:

- /a100/ Eine Benutzerverwaltung ist nicht geplant.
  - Eine Benutzerverwaltung ist für die Dienste, die Movie Genie anbieten soll, nicht erforderlich, da die Speicherung von Likes und Don't likes wenn überhaupt über Cookies realisiert wird. Sollten später dem Nutzer noch weitere Dienste zur Verfügung

gestellt werden, welche eine Speicherung von benutzerspezifischen Daten benötigen, so kann man eine Benutzerverwaltung nachträglich in das Programm einbauen.

- /a200/ Movie Genie muss keinen hohen Sicherheitsansprüchen genügen.
  - Da wir keine Benutzerdaten speichern ist die Sicherheitsanforderung an die Applikation logischerweise geringer, da den Benutzern durch Fremdeingriffe kein Schaden entstehen kann. Sollte jemals nachträglich eine Benutzerverwaltung implementiert werden, so muss das Programm nachträglich auf einen höheren Sicherheitsstandard angehoben werden um mögliche Eingriffe in Benutzerdaten zu verhindern.
- /a300/ Die Analyse der Eingaben basiert auf Heuristiken, es wird keine künstliche Intelligenz implementiert.
  - Da wir, im Gegensatz zu einer richtigen KI, über keine eigene riesige Wissensdatenbank verfügen, müssen wir uns auf die Reviews der IMDb Benutzer verlassen, um zusätzliche Informationen über die Filme zu bekommen (wie z.B. "cool cars"). Das macht es allerdings erforderlich, auf heuristische Algorithmen zurückzugreifen um die Relevanz der einzelnen Reviews für die jeweilige Suchanfrage zu bestimmen. Sofern man keine eigene Wissensdatenbank, die über jeden Film Informationen wie "Wookies", "bad explosions", "cool cars" etc. besitzt, anlegt, wird man weiter auf die heuristische Methode angewiesen sein.
- /a400/ Die Anfragen können nur auf Englisch gestellt werden.
  - Da die verwendeten Spracherkennungsbibliotheken alle auf die Englische Sprache abgestimmt sind, ist zunächst eine Eingabe auf Englisch erforderlich. Der Benutzer kann jedoch, sollte er der englischen Sprache nicht mächtig sein, einen kostenlosen Übersetzungsdienst (bspw. Googles "Translate") im Internet benutzen, um einfache Anfragen übersetzen zu lassen. Durch das Einbinden weiterer Bibliotheken könnte man später auch andere Sprachen zulassen, allerdings müsste man die Applikation dann in gewissen Dingen an mehrere Sprachen anpassen.

## 3 Implementierungsentwurf

In diesem Abschnitt erläutern wir zuerst kurz den Gesamtentwurf auf der Basis des Komponentendiagramms aus dem Grobentwurf und gehen in den nachfolgenden Unterabschnitten, je Abschnitt eine Komponente, auf die detaillierte Implementierung ein. Wir verzichten hier auf das Paketdiagramm, da wir jede Komponente in einem eigenen Paket verwalten wollen, sodass ein Paketdiagramm bestehend aus einem Paket widersinnig wäre.

### 3.1 Gesamtsystem

Abbildung 3.1 zeigt alle Komponenten und ihre Interaktion über die jeweiligen Schnittstellen; also eine kurze Übersicht der Softwarearchitektur von Movie Genie.

Die Komponente **"Movie Genie"**, /K10/, steuert die Interaktion zwischen Benutzer und **"Movie Search"**. Durch sie werden Benutzereingaben an die **"Movie Search"** weitergeleitet, und die Ergebnisse, die diese Komponente daraufhin liefert, graphisch aufbereitet. Insbesondere werden in der Komponente sämtliche GUI-Prozesse wie etwa Anzeige von Dialogen, Ergebnislisten etc. abgewickelt. Die Komponente initialisiert zudem das **"User Profile"** und die **"Movie Search"**. **"Movie Search"**, /K20/, bereitet die Suchanfragen des Benutzers auf, indem sie die Eingabetexte analysiert, in Anfragen passend unserer Datenhaltung umwandelt und darauf basierende Filme zurückliefert. Die aktuelle Ergebnisliste kann zudem durch Filmbewertungen sowie **"Undo/Redo"**-Aktionen beeinflusst werden. Dazu hält die Komponente alle bisherigen Eingabeschritte vor.

**Text Analysis**, /K30/, zerlegt Eingabetexte in einzelne grammatikalische Objekte, sodass Adjektive und deren zugehörige Substantive (Nominalphrasen), z.B. **"beautiful tree"** in dem Eingabesatz **"I'm seeing a beautiful tree"**, alleinstehende Substantiv-Ausdrücke sowie Adverbialkonstruktionen wie etwa **"movies before 2000"** identifiziert werden. Ferner wird die Polarität der Nominalphrasen bewertet, d. h. es wird kontextabhängig überprüft, ob die vorliegende Phrase im positiven Sinne erwähnt wird. Negationen, im Sinne von Verneinungen, der Nominalphrasen werden ebenfalls berücksichtigt.

**Feature Index**, /K50/, ist für das Pre-Processing der Filmreviews verantwortlich. Sie extrahiert anhand der **"Text Analysis"**-Komponente Nominalphrasen (Substantive und die Polarität ihrer zugeordneten Adjektive) aus den Benutzerreviews zu allen Filmen. Für die Charakterisierung der

**User Profile**, /K40/, verwaltet Benutzerprofile, d.h. eine Liste von Filmbewertungen, die ein Benutzer getroffen hat. Über **"UserProfile"** erzeugt sie vorkonfigurierte **"MovieSearch"**-Komponenten. Umgekehrt wird die Komponente über neue und zurückgenommene Bewertungen per **"RatingListener"** informiert.

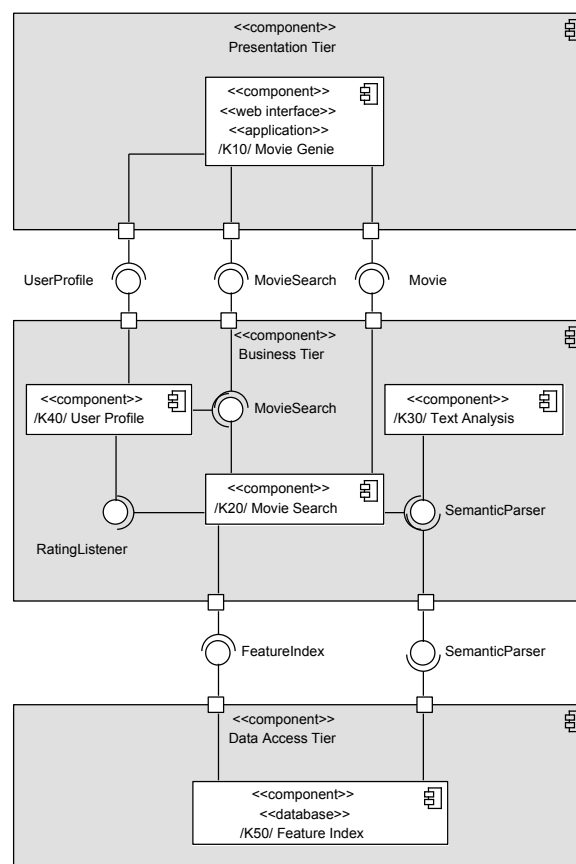


Abbildung 3.1: Komponentendiagramm

## 3.2 Implementierung von Komponente /K10/: Movie Genie:

Diese Komponente implementiert die grafische Benutzeroberfläche der Webapplikation "Movie Genie", die mit Hilfe des Java-Frameworks "Vaadin" implementiert werden soll. Vaadin stellt alle notwendigen Elemente zur Erstellung einer Internetseite zur Verfügung, die sich zum größten Teil durch CSS optisch anpassen lassen. Die GUI erfasst die Benutzereingaben und leitet sie zur Verarbeitung an die MovieSearch-Komponente weiter. Des Weiteren übernimmt sie sämtliche Kommunikation mit dem Benutzer und stellt die Ergebnisliste, die von der MovieSearch-Komponente zurückgegeben wird optisch dar. Rückmeldungen des Systems werden dem Benutzer durch den Movie Genie präsentiert, der eine Chat-ähnliche Konversation mit dem Benutzer führen soll. Überdies dient der Chat als eine Art "Search-History" in dem man den Suchverlauf der über mehrere Eingaben iterativ verfeinerten Suche nachvollziehen kann. Die GUI verwaltet darüberhinaus die für das User Profile benötigten Cookies und erkennt ob ein Nutzer die Seite bereits besucht hat.

### 3.2.1 Klassendiagramm

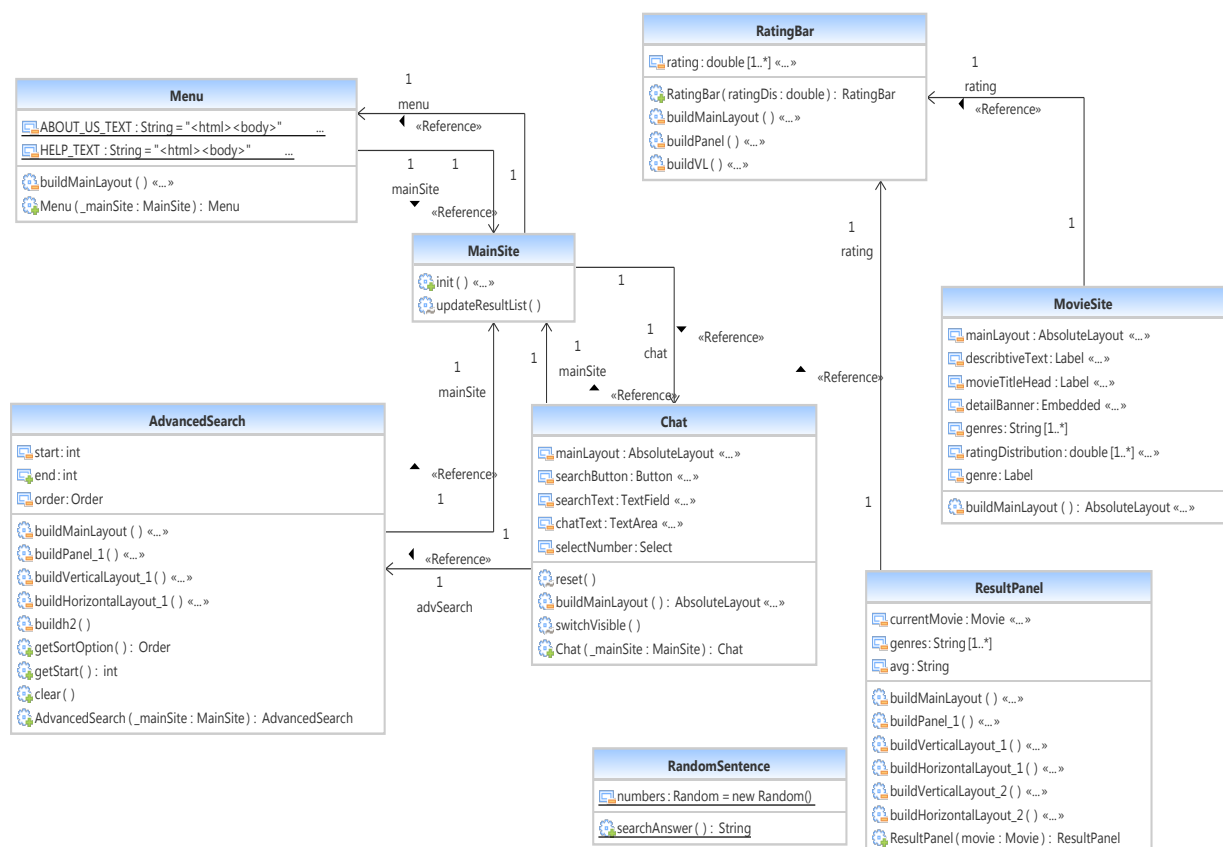


Abbildung 3.2: Klassendiagramm GUI

### 3.2.2 Erläuterung

Der Übersicht halber wurden Referenzen auf Vaadin-Klassen entfernt.

- **RandomSentence:** Die Klasse RandomSentence erzeugt die Antworten des Movie Genies, die in dem Chatfenster angezeigt werden. Dazu wird ein Random Numbers Generator von Java verwendet. Mithilfe der daraus erzeugten Zahlen wird dann ein bestimmter, vorgefertigter Satz ausgewählt und übergeben.

– Attribute:

\* `numbers`: Ein Objekt der Klasse Random, das für Zufallszahlen sorgt.

– Methoden:

\* `searchAnswer():String` : eine Methode die durch benutzen des numbers Objekts einen von mehreren verschiedenen Strings zufällig ausgeben soll

- **RatingBar:** Die Klasse erstellt zu jedem Film ein Histogramm, das die Verteilung der Bewertungen optisch darstellt. Dies erfolgt über Labels die jeweils einem bestimmten Wert zugeteilt werden, sprich ein Label für den Wert 10, eins für den Wert 9, etc.. Daraufhin wird für jedes der Label aus der vorliegenden Rating Verteilung ein Anteil an der Gesamtlänge berechnet, sodass bei einer Gesamtlänge von, beispielsweise, 320 Pixeln ein Label mit einem Anteil von 10 Prozent der Bewertungen eine Länge von 32 Pixeln erhält. Daraufhin erhält jeder Wert noch einen bestimmten CSS-Style um für eine Farbverteilung von Dunkelgrün (Wert 10) zu Dunkelrot (Wert 1) zu sorgen.

– Attribute:

\* `rating`: Speicherung der Ratingverteilung des Films aus der IMDb als double Array

– Methoden:

\* `RatingBar()` : Eine Methode die mit Hilfe einer RatingDis[tribution] (Parameter) 10 Labels erzeugt, welche in ihrer Länge dem prozentualen Anteil des jeweiligen Werts (10, 9, 8, ...) widerspiegeln. Außerdem werden den Labels bestimmte CSS Layouts zugewiesen, welche unterschiedliche Hintergrundfarben besitzen sollen (von Dunkelgrün[10] bis Dunkelrot[1]). Daraufhin werden sie dem Layout hinzugefügt.

\* `buildMainLayout()`: Standard Methode des Vaadin GUI Designers zum erstellen eines MainLayouts.

\* `buildPanel()`: Standard Methode des Vaadin GUI Designers zum erstellen eines Panels.

- \* `buildVL()`: Standard Methode des Vaadin GUI Designers zum erstellen eines VerticalLayouts. Standard Layout des Panels. Enthält ein HorizontalLayout, indem die Labels der Ratingbar liegen.
- **MainSite**: Die Klasse stellt die Startseite der Webapplikation dar. In dieser Klasse befindet sich das MainWindow in der die Angezeigten Komponenten dargestellt werden. Desweiteren wird die GUI von hier aus gestartet.
  - Methoden:
    - \* `init()`: Die Initialisierungsmethode der Vaadin GUI. Sie startet quasi das Programm.
    - \* `updateResultList()`: Eine Methode die neue Results von der MovieSearch Komponente beziehen soll um neue Informationen aus den Suchanfragen widerspiegeln zu können.
  - **Menu**: Die Klasse erstellt die Menüleiste der Webapplikation. Diese Menüleiste enthält Buttons für die Funktionalitäten die über die einfache Suche hinausgehen. Dazu zählen z.B. die Undo und Redo Funktionalitäten, aber auch jeweils ein Button um ein Hilfe oder About Us Fenster anzuzeigen.
    - Attribute:
      - \* `ABOUTUSTEXT`: In dieser Variable steht der Html formatierte Text für das Html Label im About Us Window.
      - \* `HELPTTEXT`: In dieser Variable steht der Html formatierte Text für das Html Label im Help Window.
    - Methoden:
      - \* `buildMainLayout()`: Standard Methode des Vaadin GUI Designers zum erstellen eines MainLayouts.
      - \* `menu(MainSite _mainSite)`: Eine Methode der eine MainSite übergeben werden muss um dort ein Menü einzufügen. Dieses Menü soll über die Buttons „New Search“ zum Starten einer neuen Suche, „Undo“ zum löschen der letzten Aktion, „Redo“ zum wiederherstellen der letzten gelöschten Aktion, „Reset“ zum kompletten zurücksetzen, „About Us“, „Help“ zum Öffnen eines Help Windows und „Advanced Search“ zum Anzeigen/Ausblenden der Advanced Search verfügen.





Abbildung 3.3: Menu

- **AdvancedSearch:** Die Klasse `AdvancedSearch` repräsentiert die erweiterte Suche, die im Layout eingeblendet werden kann. In dieser erweiterten Suche, hat der User dann die Möglichkeit die Sortierungseinstellungen zu verändern und die Zeitangabe leichter zu verwalten.

- Attribute:

- \* `start`: In dieser Variable steht der Startwert für die Timespan, welche mit dem Apply-Button zur MovieSearch hinzugefügt werden soll.
- \* `end`: In dieser Variable steht der Endwert für die Timespan, welche mit dem Apply-Button zur MovieSearch hinzugefügt werden soll.
- \* `order`: In dieser Variable wird die aktuell angewählte Sortierung für die MovieSearch gespeichert, welche mit dem Apply-Button zur MovieSearch hinzugefügt wird.

- Methoden:

- \* `advancedSearch(MainSite mainSite)`: Methode zum erstellen der Advanced Search in einer MainSite(Parameter).
- \* `buildMainLayout()`: Standard Methode des Vaadin GUI Designers zum erstellen eines MainLayouts.
- \* `buildPanel1()`: Standard Methode des Vaadin GUI Designers zum erstellen eines Panels.
- \* `buildVL1()`: Standard Methode des Vaadin GUI Designers zum erstellen eines VerticalLayouts. Standard Layout des Panels. Enthält ein HorizontalLayout, indem die Labels der Ratingbar liegen.
- \* `buildHL1()`: Standard Methode des Vaadin GUI Designers zum erstellen eines HorizontalLayouts. Enthält die Eingabefenster und Label für die Timespan.
- \* `buildh2()`: Standard Methode des Vaadin GUI Designers zum erstellen eines HorizontalLayouts. Enthält die OptionGroup für die Sortierung und den Apply-Button.
- \* `clear()`: Löscht alle Einträge aus der Advanced Search.

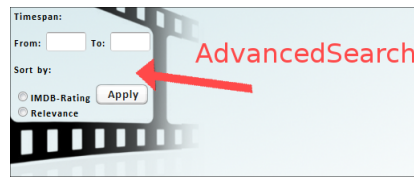


Abbildung 3.4: AdvancedSearch

- **ResultPanel:** Die Klasse erstellt ein Panel zur Vorschau der vorgeschlagenen Filme. Die Filme werden untereinander jeweils in eigenen Panels dargestellt und enthalten Informationen über den Titel, das Erscheinungsjahr, das Genre und die durchschnittliche Bewertung. Überdies **wir** auch noch ein Filmposter der Größe "Thumb" dargestellt, sofern ein entsprechendes Poster in der Datenbank gespeichert ist.

– Attribute:

- \* **currentMovie:** Das momentan verwendete Movie Objekt aus den Results der MovieSearch.
- \* **genres:** Ein String Array der alle Genres des momentan verwendeten Movie Objekts enthält.

– Methoden:

- \* **buildMainLayout():** Standard Methode des Vaadin GUI Designers zum erstellen eines MainLayouts.
- \* **buildPanel\_1():** Standard Methode des Vaadin GUI Designers zum erstellen eines Panels.
- \* **buildVL\_1():** Standard Methode des Vaadin GUI Designers zum erstellen eines VerticalLayouts. Standard Layout des Panels. Enthält horizontalLayout1.
- \* **buildhorizontalLayout1():** Standard Methode des Vaadin GUI Designers zum erstellen eines HorizontalLayouts. Enthält das Thumb-Poster.
- \* **resultPanel(Movie movie):** Methode zum erstellen des resultPanel zu einem bestimmten Movie Objekt(Parameter).



Abbildung 3.5: Schematische Darstellung ResultPanel

- **MovieSite:** Die Klasse erstellt die Detailansicht zu einem Film. Diese Detailansicht wird in einem Popup Fenster geöffnet und enthält Informationen über den Titel, das Erscheinungsjahr, die Genres, die Ratingverteilung, sowie eine Kurzbeschreibung.

– Attribute:

- \* `mainLayout`: Das verwendete `MainLayout`.
- \* `descriptiveText`: Ein Label das die Kurzbeschreibung des Films enthalten soll.
- \* `movieTitleHead`: Ein Label, das den Titel des Films enthalten soll.
- \* `detailBanner`: Eine Embedded Komponente, die das Cover des Films enthalten soll.
- \* `ratingDistribution`: Speicherung der Ratingverteilung des Films aus der IMDb als double Array.
- \* `genres`: Ein String der alle Genres des momentan verwendeten Movie Objekts enthalten soll.
- \* `genre`: Ein Label das die Genres des Films anzeigt.

– Methoden:

- \* `buildMainLayout()`: Standard Methode des Vaadin GUI Designers zum erstellen eines `MainLayouts`.
- \* `movieSite(Movie movie)`: Methode zum Erstellen der `MovieSite` zu einem bestimmten Movie Objekt (Parameter).

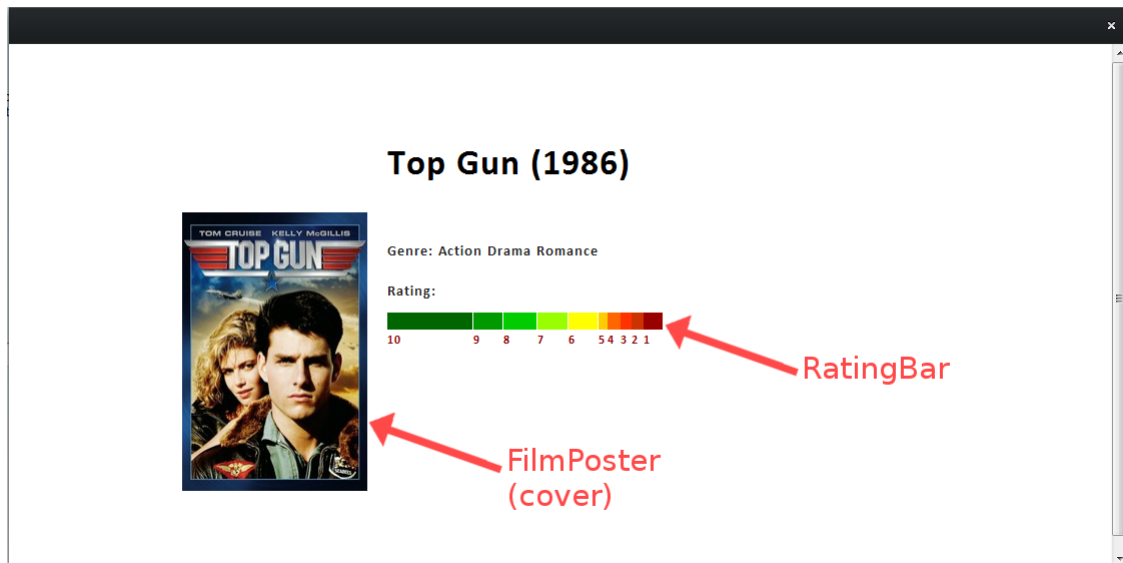


Abbildung 3.6: Schematische Darstellung MovieSite

### 3.3 Implementierung von Komponente /K20/: Movie Search:

Die Komponente MovieSearch ist für das Suchen der Filme zuständig. Sie bekommt die Benutzereingabe (z.B. "I want an good action movie with Brad Pitt from 2008 without Will Smith") übergeben und filtert daraus - mit Hilfe von Methoden aus dem Paket `text` - die nötigen Informationen und führt die Suche durch. Die Ergebnisse können in der Anzahl und in der Sortierung angepasst werden. Die Komponente ist in der Lage, harte Kriterien (hier: action movie, Brad Pitt, 2008) und weiche Kriterien (hier: good action movie) zu verarbeiten und Negationen (without Will Smith) zu erkennen.

Die Suchanfrage wird durch Aufruf der Methode `addQuery(String verboseQuery)` durchgeführt, bei der die Benutzereingabe übergeben wird. Als erstes wird mit Hilfe der Klasse `SemanticParser` aus dem Paket `text` die Benutzereingabe zerlegt, um anschließend in den Folgeschritten alle Nomen, Personennamen, Genres und Zeitangaben herauszufiltern. Die Namen werden mit `findNames()` bzw. `findNegatedNames()` gefiltert und jeweils in Listen gespeichert. Die Genres werden mit einer Liste aller in der Datenbank enthaltenen Genres abgeglichen und ebenfalls in Listen gespeichert, wobei auch hier die Negation beachtet wird. Zeitangaben werden mit der Klasse `TimeRegex` erkannt und verarbeitet. Die gefundenen Informationen werden mit der Klasse `QueryContext` zu einer Datenbankabfrage verarbeitet, in dem sie mit `addGenres(genres)`, `addPersons(names)` etc. hinzugefügt werden. Die Suche wird dann mit `fetchMovies()` **ausgeführt. entfernt.**

#### 3.3.1 Klassendiagramm



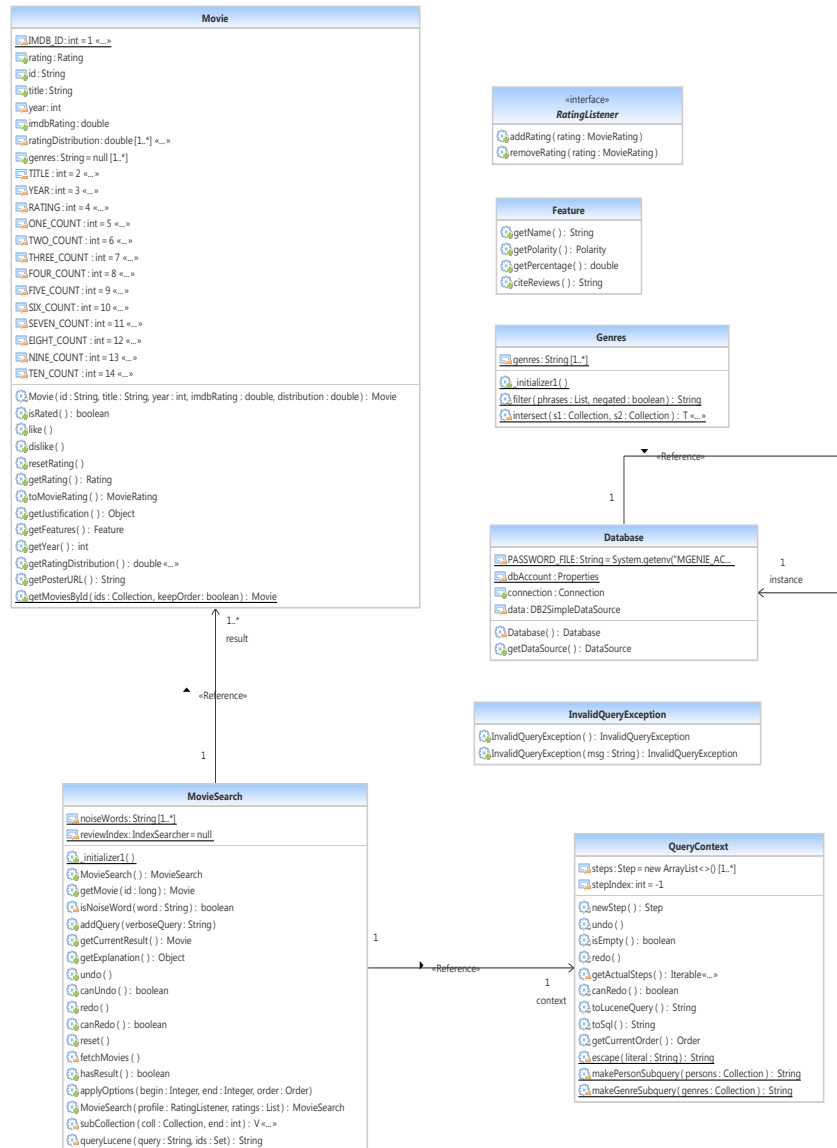


Abbildung 3.7: Klassendiagramm MovieSearch

### 3.3.2 Erläuterung

- **Movie Search:** Die Klasse ist für das Verarbeiten der Suchanfrage und der Ausgabe des Ergebnisses zuständig

– Attribute:

- \* `Set<String> noiseWords` enthält Wörter, die als störend eingestuft und gefiltert werden (z.B. bei "I want an action movie" ist 'movie' störend).

- \* `IndexSearcher reviewIndex` ist ein Index der Filmreviews.

– Methoden:

- \* `MovieSearch(RatingListener profile, List<Movie.MovieRating> ratings)` realisiert das Rating-Config

- \* Der Konstruktor `MovieSearch()` erzeugt ein neues `MovieSearch`-Objekt

- \* `getMovie(long id)` liefert den Film, der zu der ID passt

- \* Mit `addQuery(String verboseQuery)` wird eine Suchanfrage erstellt und durchgeführt

- \* Das Anwenden von Einstellungen wird durch `applyOptions(Integer begin, Integer end, Order order)` erledigt

- \* Um das aktuelle Ergebnis auszugeben wird `getCurrentResult()` verwendet

- \* Eine Erklärung, warum genau jener Film empfohlen wird, lässt sich mit `getExplanation()` abrufen

- \* Einzelne Schritte der Suche lassen sich mit `undo()` und `redo()` widerrufen und wiederholen

- \* `canUndo()` und `canRedo()` prüfen die Verfügbarkeit von `undo()` und `redo()`

- \* Um die Suche zurückzusetzen wird die Funktion `reset()` verwendet

- \* Ob eine Suche ein Ergebnis hat, lässt sich mit `hasResult()` überprüfen

- **Database:** Stellt die Verbindung zur Datenbank her

– Attribute:

- \* `String PASSWORD_FILE` definiert eine Benutzervariable auf dem PC des Anwenders, die die Zugangsdaten zur Datenbank enthalten muss

- \* `Properties dbAccount` enthält den Datenbankaccount

- \* `Connection connection` und `DB2SimpleDataSource data` sind für die Verbindung zur Datenbank

- Methoden:
  - \* `getInstance()` ist ein Singleton für eine Datenbankverbindung
  - \* `getConnection()` liefert die Datenbankverbindung der aktuellen Database
  - \* `getDataSource()` zeigt die `DataSource` der aktuellen Database
- **Feature:** Stellt bestimmte Funktionen für Filme bereit
  - Methoden:
    - \* Mit `getName()` wird der Name zurückgegeben
    - \* Die Polarität wird mit `getPolarity()` zurückgegeben
    - \* `getPercentage()` gibt die prozentuale Bewertung zurück
    - \* `citeReviews()` gibt kurze Zitatstellen aller Reviews, die den Film entsprechend bewertet haben, zurück
- **Genres:** Dient der Erkennung von Genres in Benutzereingaben
  - Attribute:
    - \* `Set<String> genres` enthält alle in der Datenbank vorhandenen Genres
  - Methoden:
    - \* Die Funktion `filter(List<Phrase> phrases, boolean negated)` findet die Genres in den Phrases und entfernt sie aus der Eingabe
- **InvalidQueryException:** Exception, die bei einer ungültigen Abfrage ausgelöst wird
  - Methoden:
    - \* Die Funktion `InvalidQueryException()` ist Konstruktor ohne Detailangaben
    - \* `InvalidQueryException(String msg)` ist ein Kontruktor mit Detailangaben
- **Movie:** Dient der Erstellung von Movie-Objekten, die alle Filmdaten enthalten
  - Attribute:
    - \* Die Variablen `int IMDB_ID`, `TITLE`, `YEAR`, `RATING`, `ONE_COUNT`, `TWO_COUNT`, `THREE_COUNT`, `FOUR_COUNT`, `FIVE_COUNT`, sind Felder zum schnelleren Zugriff bei `getMoviesById()` auf die Query-Abfrage
    - \* `Rating rating` dient der Zwischenspeicherung von Ratings
    - \* `String id` enthält eine Movie-ID
    - \* `String title` enthält einen Filmtitel
    - \* `int year` ist eine Jahreszahl

- \* `double imdbRating` enthält das Rating eines Films laut der IMDb
- \* `double[] ratingDistribution` hat die Verteilung der Bewertungen gespeichert
- \* `List<String> genres` speichert die Genres, die in der Suche berücksichtigt werden sollen

– Methoden:

- \* `MovieRating(long movieId, Rating rating)` speichert die Bewertung eines Films
- \* Die Methode `getMovieID()` liefert die MovieID
- \* Das Rating wird mit `getRating()` abgefragt
- \* Der Konstruktor `Movie(String id, String title, int year, double imdbRating, double[] distribution)` erstellt ein Film-Objekt, das Daten über einen bestimmten Film enthält
- \* Mit Hilfe von `isRated()` kann geprüft werden, ob der Film bewertet wurde
- \* Mit `like()` und `dislike()` kann ein Film geliked bzw gedisliked werden
- \* Das Rating kann durch die Funktion `resetRating()` wieder gelöscht werden
- \* `getRating()` ruft die Benutzerbewertung zu einem Film ab
- \* `MovieRating toMovieRating()` erstellt ein neues Movierating aus „id“ und „rating“
- \* Begründungen, warum ein Film im Ergebnis auftaucht, können mit `getJustification()` abgefragt werden
- \* Um die Features abzufragen wird `getFeatures()` benutzt
- \* Mit `getId()` wird die MovieID abgerufen
- \* `getTitle()` ruft den Titel und `getYear()` das Erscheinungsjahr ab
- \* Mit `getGenres()` wird das Genre eines Films abgerufen
- \* Die Funktion `getRatingDistribution()` gibt die prozentuale Verteilung der Ratings zurück, sodass das i-te Element den Anteil der Bewertungen mit i+1 Punkten angibt
- \* `getImdbRating()` gibt das IMDB-Rating zurück
- \* `getPosterURL()` gibt die URL für das Filmvorschaubild zurück
- \* Die Methode `getMoviesById(Collection<String> ids, boolean keepOrder)` gibt Movies passend zur ID zurück

- **QueryContext:** Konstruiert die SQL- und Luceneabfragen

– Attribute:



- \* `List<Step> steps` ist eine Liste mit den durchzuführenden Schritten für die Abfrage

- \* `int stepIndex` ist ein Zähler für den aktuellen Schritt

– Methoden:

- \* `newStep()` erzeugt einen neuen Schritt zur Abfrageerstellung

- \* Die Funktionen `undo()` und `redo()` widerrufen bzw. wiederholen einen Schritt

- \* `isEmpty()` prüft, ob ein Schritt leer ist

- \* Die Methode `canRedo()` prüft, ob ein Schritt wiederholt werden kann

- \* Genres werden mit `addGenres(Collection<String> newGenres)` hinzugefügt

- \* `isEmpty()` prüft, ob die Abfrage leer ist

- \* Das Abbrechen eines Schrittes erfolgt mit `cancel()`

- \* Mit `addPersons(List<String> names)` und `excludePersons(List<String> nnames)` werden Personen hinzugefügt bzw. ausgeschlossen

- \* Durch `addZeitAnfang(int anfangZeit)` und `addZeitEnde (int endeZeit)` werden die Anfangs- und Endzeiten festgelegt

- \* Mit `addFeature(String feature)` und `addExcludedFeature(String feature)` werden Features hinzugefügt bzw. ausgeschlossen

- \* Das Ausschließen von Genres erfolgt mit `addExcludedGenres(Set<String> excludedGenres)`

- \* `setOrder(Order order)` legt die Sortierung fest

- \* `toString()` ist ein Overwrite für die Stringausgabe

- \* Die Methoden `toLuceneQuery()` und `toSql()` erstellen Lucene- bzw. SQL-Abfragen

- \* Die aktuelle Sortierung wird mit `getCurrentOrder()` angezeigt

### 3.4 Implementierung von Komponente /K30/: Text Analysis:

Die Komponente ist für die gesamte Textanalyse von Movie Genie zuständig. Hierbei bildet die Klasse `SemanticParser`, Abbildung 3.4, das Herz dieser Komponente. Für jeden Eingabetext erzeugt die Komponente `MovieSearch` (bzw. `FeatureIndex` im Pre-Processing offline, d. h. vor dem Start der eigentlichen Applikation) eine neue Instanz vom `SemanticParser`, welcher den Eingabetext mithilfe verschiedener, paketinterner Klassen zerlegt und alles für die Suche Nötige in speziellen Datenstrukturen speichert und bereitstellt. Konkret wird der Eingabetext von der Komponente, nach dem Zerlegen durch den `StringTokenizer`, auf folgende Bestandteile untersucht:

- Personennamen, die an einem Film beteiligt sind (Schauspieler, Regisseure, Autoren, Komponisten und Produzenten)
  - negierte Personennamen
- Nominalphrasen, bestehend aus einer zusammengehörenden Menge von Adjektiven und Nomen. Einschließlich der Polarität (ist die Phrase in einem positiven/negativen Kontext gefallen?) dieser Nominalphrasen
  - negierte Nominalphrasen
- Datumsangaben, die angeben, zu welchem Jahr (oder ein Zeitintervall) die Filme gesucht werden sollen.

Für die Zerlegung und Analyse von natürlicher Sprache sollen die Bibliotheken OpenNLP von Apache (<http://opennlp.apache.org/>) und SentiWordNet (<http://sentiwordnet.isti.cnr.it/>), das auf WordNet aufbaut, verwendet werden. Sie stellen die Datenmodelle bereit, anhand derer:

- Part-of-speech-Tagging sämtlicher Texte ermöglicht wird. Das ist die grammatikalische Zuweisung von Wörtern in Wortgruppen, sodass Adjektive, Verben, Nomen, Satzzeichen u. a. auch als solche erkannt und markiert werden.
- Daraufhin mit Hilfe der produzierten POS-Tags Phrasenstrukturen erkannt werden, bspw. Nominal- oder Verbalphrasen.
- Eigennamen in einem Text gefunden werden, also bspw. Schauspieler, Firmen oder auch Institutionen.
- Die Polarität der Schlüsselwörter bestimmt wird, ob ein Nomen bspw. in einem positiven oder negativen Kontext gefallen ist ("**the plot of this movie is awesome**" vs. "**this plot is so crappy**").

Für die Bestimmung der Negation soll die Klasse ConText dienen, die den bekannten Algorithmus von Chapman et al implementiert. Es gibt dafür viele freie, fertige Klassen. Eine mögliche Implementierung findet sich in der Universitätsbibliothek von Utah.



### 3.4.2 Erläuterung

- **Negation und Polarity:** Sind zwei Enum-Klassen, die in je drei Werten, die Eigenschaften Negation und Polarität der Nomen von Phrasen-Objekten, speichern.
- **ConText:** Die Klasse implementiert den Algorithmus von Chapman et al, um basierend auf regulären Ausdrücken kontextabhängig die Negation von Wörtern zu bestimmen. Bspw. soll das Wort „cars“ in „show me some action movies **without** cars“ als negiert markiert werden. Der Konstruktor erzeugt eine neue ConText-Instanz, die man dann über `applyContext(concept: String, sentence: String)` aufrufen kann. Daraufhin liefert die Methode einen String, der die Negation von concept beschreibt („Negated“, „Affirmed“ oder „Possible“).
- **SentiWordProcessor:** Weist den Phrase-Objekten eine Polarität zu, inwieweit die Wörter in einem negativen, positiven oder neutralen Zusammenhang erwähnt wurden. Das heißt, dass so etwas wie "bad movie" mit der Konstanten NEGATIVE markiert wird. Die Klasse lädt bei der Erzeugung zunächst intern eine Liste aus der SentiWordBibliothek (siehe Einleitung) und erzeugt hieraus eine Lookup-Tabelle (mit ca. hunderttausend Einträgen), die dann über die einzige Methode `extract(word: String) : Polarity` nach Adjektiven abgefragt wird. Die Methode liefert daraufhin die Polarität zum Wort. Sollte das Wort nicht gefunden werden, soll die Methode per Default NEUTRAL ausgeben.
- **Phrase:** Die Klasse dient als Datenspeicher für die vom `SemanticParser` gefundenen Nominalphrasen. Dafür können die Parameter sowohl über den Konstruktor als auch über die Setter-Methoden gesetzt werden. Intern speichert `Phrase` seine Nomen und Adjektive in einer Liste. Zusätzlich dazu noch die Polarität, die vom `SentiWordProcessor` erzeugt wird und die Negation, die die Klasse `ConText` liefert. Auf den Inhalt von `Phrase` greift man dann über die Getter-Methoden zu.
- **StringTokenizer:** Zerlegt einen Eingabestring, den die Klasse im Konstruktor übergeben bekommt, in Tokens. Dabei darf man die Klasse nicht mit der von Oracle aus dem Util-Paket verwechseln. Diese hier arbeitet mit dem Datenmodell von OpenNLP und zerlegt die Eingabe entsprechend dieses Modells. Dabei werden bspw. auch solche Wörter sinngemäß getrennt: „didn‘t“ ==> „did“ und „n‘t“. Da das Laden des Modells eine Weile dauern kann, soll es nur einmal in einem statischen Initialisierungsblock geladen werden. Die `Tokenizer` liefert dann die Getter `getTokens() : String[]`.
- **TextUtils:** ist eine Utility-Klasse, die ein paar statische Methoden bereithält, um Zeichenketten zu manipulieren.
  - Die nötigen Methoden:

- \* `prepareInput(input: String[])`: durchläuft die vom `StringTokenizer` erzeugten Tokens und ersetzt alle „n't“ und „s“ durch „not“ und „is“. Weitere Optimierungsfälle sind denkbar.
- \* `arrayToString(input: String[]) : String`: konkateniert das übergebene Array zu einem einzigen String. Wird vom `SemanticParser` benötigt, um bspw. die optimierten Tokens wieder zusammenzuführen.
- \* `listToString(input: List<String>) : String`: konkateniert die übergebene Liste zu einem einzigen String. Wird vom `PreProcessor` benötigt, um alle Reviews zu einem Film zusammenzuführen und den String dem Tokenizer zu übergeben.
- **SemanticParser**: der Parser verbindet alle Funktionalitäten der Klassen aus dem Paket, um eine Texteingabe zu analysieren und in Datenstrukturen (Phrases) zu zerlegen.
  - Die wichtigen Attribute bestehen vor allem aus den Datenmodellen der OpenNLP-Bibliothek, die einmal vor der Erzeugung der Klasse in einem statischen Block initialisiert werden sollen.
- \* `stringTokenizer`: der `Stringtokenizer`.
- \* `posModel`: das POS-Model zum Taggen der Tokens.
- \* `tagger`: der eigentliche POS-Tagger.
- \* `chunkerModel`: der Chunker setzt die getaggten Tokens in Relation zu einander, um so die Phrasenstrukturen zu erkennen. Dies ist das Modell von OpenNLP dafür.
- \* `chunker`: der eigentliche Chunker.
- \* `nameModel`: das Modell für den NameFinder, der Eigennamen aus der Eingabe extrahiert.
- \* `nameFinder`: der eigentliche Namensfinder.
- \* `phrases`: Liste für die gefundenen Phrasen.
- \* `names` und `negatedNames`: eine `HashMap` für die gefundenen Namen. Dabei ist der Schlüssel jeweils der Name der Person und der Wert eine Liste mit den Berufen, die er laut Datenbank je ausübte. Die Liste soll absteigend der Relevanz der Berufe sortiert (bei Harrison Ford: (actor, producer)) werden.
- \* `context`: `ConText`-Klasse zur Ermittlung der Negation eines Wortes.
- \* `senti`: `SentiWordProcessor` wie oben beschrieben.

- Methoden: der einzige Konstruktor übernimmt einen String als Eingabe, den der SemanticParser daraufhin auswertet. Der Konstruktor zerlegt zuerst die Eingabe mithilfe des Tokenizers, um diese zu "taggen" und zu "chunken". Ist diese Vorarbeit getan, können die vier folgenden Methoden aufgerufen werden:

- \* `searchForPhrases()`: durchläuft die Eingabe und speichert die gefundenen Phrasen ab. Die Suche basiert hierbei algorithmisch auf den Tags des Chunkers.

- \* `checkNames()`: sucht die Namen aus den gefundenen Phrasen, und speichert sie in der entsprechenden Datenstruktur. Aus optimierungstechnischen Gründen sollte nicht die direkte Eingabe des Benutzers, sondern die daraus bereits gefilterten Phrasen, durchlaufen werden.

- \* `checkNegation()`: sucht nach negierten Phrasen und Namen. Werden welche gefunden, werden sie in den entsprechenden Strukturen abgespeichert und die Originale gelöscht.

- \* `checkPolarity()`: berechnet die Polarität der gefundenen Phrasen mithilfe des `SentiWordProcessors`.

- Die öffentlichen Schnittstellen für den Zugriff der anderen Komponenten lauten:

- \* `findNounPhrases() : List<Phrase>`

- \* `findNames() : HashMap<String, List<String>>`

- \* `findNounPhrases() : HashMap<String, List<String>>`

- **TimeRegex**: die Klasse hat die Aufgabe Zeitangaben, die evntl. in der Eingabe des Benutzers vorhanden sind, zu finden und über entsprechende Getter der `MovieSearch`-Komponente zugänglich zu machen. Der Konstruktor bekommt hierbei die Originaleingabe und leitet sie an die Methode `exists() : boolean` weiter. Die Klasse arbeitet mit regulären Ausdrücken, um Zeitangaben („movies between 2000 and 2010“ oder auch „action movies from the nineties“) zu erkennen. `getAnfang() : int` und `getEnde() : int` liefern hinterher den Beginn bzw. das Ende der gefundenen Zeitspanne (bei nur einem Jahr, sind beide Werte gleich - entsprechend soll bei anderen Kombinationen vorgegangen werden).

### 3.5 Implementierung von Komponente /K40/: User Profile:

Die Komponente User Profile übernimmt die Speicherung der Likes und Dislikes der anonymen Nutzer. Dies wird erreicht indem man für jeden Nutzer einen speziellen Datenbankseintrag mit seinen Likes und Dislikes anlegt. Die Erkennung des Nutzers läuft mit der Hilfe von Cookies, in denen die jeweilige UserID gespeichert wird, so dass man die Likes und Dislikes auch über

mehrere Besuche hinweg eindeutig zuordnen kann. Die gespeicherten Daten werden gelöscht sollte der jeweilige Nutzer die Site 30 Tage nicht besuchen, um Datenmüll zu vermeiden der entstehen würde, wenn die Nutzer ihre Cookies entfernen. Das entfernen der Cookies führt darüber hinaus auch dazu, dass die Likes und Dislikes nicht mehr abgerufen werden können.

### 3.5.1 Klassendiagramm

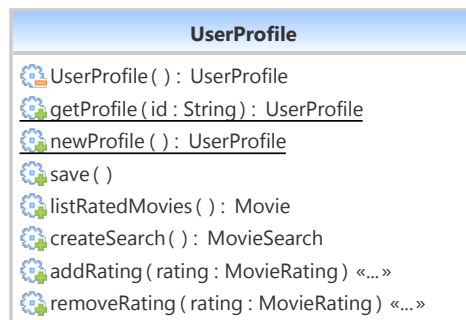



Abbildung 3.9: Klassendiagramm UserProfile

### 3.5.2 Erläuterung

#### Addon BrowserCookies

**BrowserCookies** ist ein ddon für das von uns verwendete Vaadin Framework, welches es vereinfacht persistente, also auf der Festplatte des Nutzers gespeicherte, Cookies mithilfe von Vaadin zu verwalten. Diese persistenten Cookies ermöglichen es, im Gegensatz zu den standardmäßig Verwendeten sogenannten Session Cookies, Nutzer auch Sessionübergreifend zu identifizieren und sind daher gut dazu geeignet sich an jemanden der die Website bereits besucht hat zu "erinnern". Das Addon BrowserCookies implementiert bereits alle benötigten Methoden und Listener, die dazu erforderlich sind Cookies im Browser des Nutzers anzulegen und bereits gespeicherte Cookies zu erkennen und wenn nötig zu verändern.

#### Klasse UserProfile

Die Klasse **UserProfile** dient der Verwaltung der jeweiligen Nutzerprofile.

- `getProfile(String id)`: Versucht über die angegebene ID ein Nutzerprofil in der Datenbank zu finden und soll die daraus entnommen Daten als UserProfile Objekt zurückgeben.
- `newProfile()`: Dient dazu ein neues Profil für einen neuen Nutzer in der Datenbank anzulegen.

- `addRating(MovieRating rating)`: Soll das angegebene `MovieRating` zu den Ratings des jeweiligen Nutzers hinzufügen.
- `removeRating(MovieRating rating)`: Löscht das angegebene `MovieRating` aus den Ratings des jeweiligen Nutzers.
- `save()`: Dient dazu das Profil des Nutzers zu speichern.
- `listRatedMovies()`: Erzeugt eine Liste aller bis jetzt bewerteten Filme, inkl. Bewertung, des Nutzers und gibt diese zurück.
- `createSearch()`: Erzeugt ein vorkonfiguriertes `MovieSearch`-Objekt, realisiert `SSearch Factory`

### 3.6 Implementierung von Komponente /K50/: Feature Index:

Die Komponente hat zwei Hauptaufgaben: zum einen extrahiert sie mithilfe der Komponente `"Text Analysis"` Nominalphrasen, d. h. fertige Phrase-Objekt bestehend aus Adjektiven und Substantiven inklusive Negation und Polarität, aus allen Film-Reviews der Filmdatenbank (ca. 1.800.000 Stück). Diese Objekte werden daraufhin denormalisiert zurück in die Datenbank geschrieben, um sie so zusammen für die Suche indizieren zu können. Die zweite Aufgabe wäre die Erstellung eines Suchindex', der neben den Phrasen alle nötigen Filminformationen (ID, Titel, Genre, beteiligte Personen etc.) enthält und für die Suche mit Lucene verwendet werden kann. Die gesamte Analyse und Indizierung erfolgt hierbei offline vor dem Start der Applikation, weil die reine Rechenzeit bei einigen Wochen liegt.

Für die Erstellung des Suchindex' (auf den im Datenmodell noch genauer eingegangen wird) und der Suche darauf soll Lucene, die freie Bibliothek von Apache (<http://lucene.apache.org/>), verwendet werden. Für die Datenbank-Abfragen und -Updates kann das `Sql`-Paket von Java benutzt werden.



### 3.6.1 Klassendiagramm

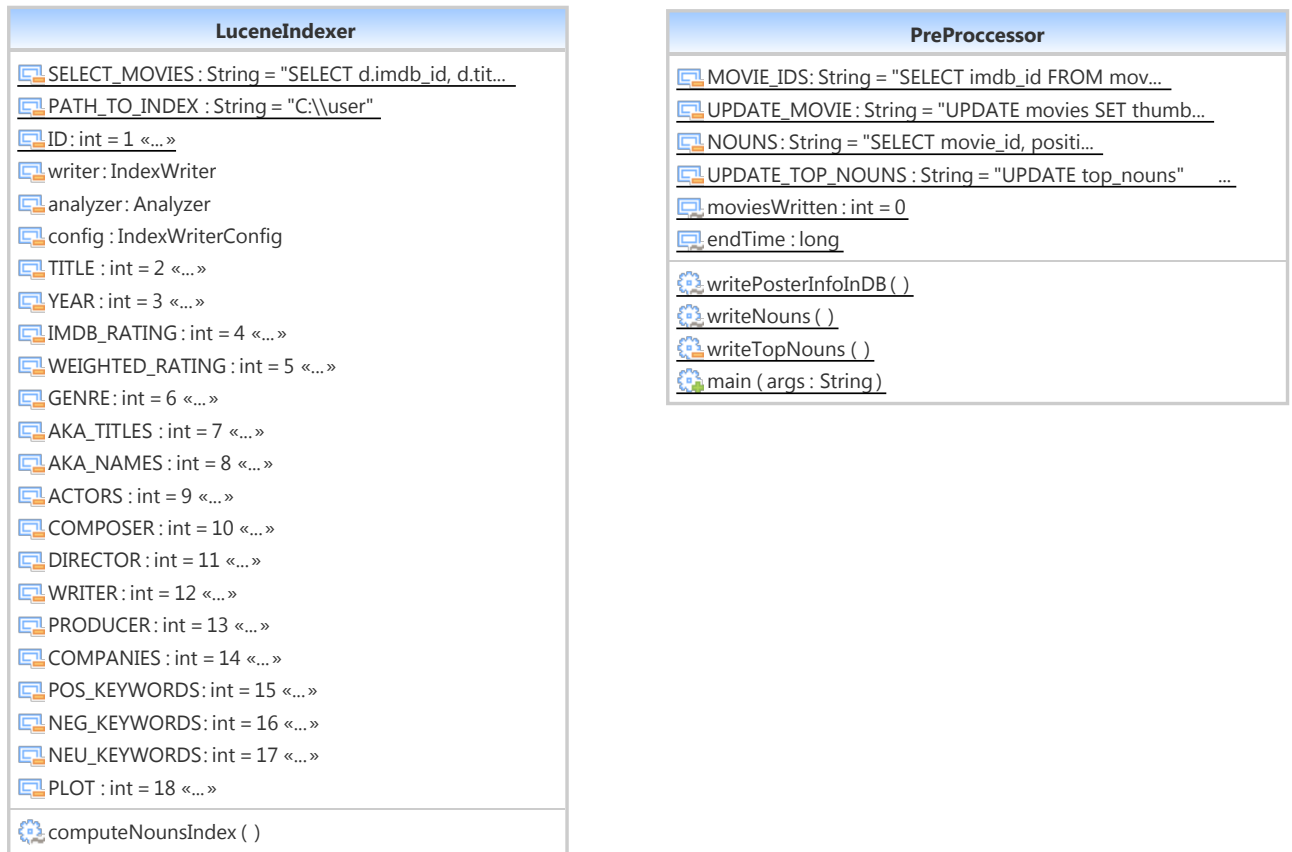


Abbildung 3.10: Klassendiagramm Feature Index

### 3.6.2 Erläuterung

- Die Klasse **PreProcessor** erledigt ein paar Vorarbeiten in der Datenbank. Ihre Hauptaufgabe ist es die Nominalphrasen aus den Reviews zu extrahieren und die fertigen Datensätze zurück in die Datenbank zu schreiben.
  - Die Klasse enthält nur ein paar statische **Sql**-Statements, die in Strings gespeichert werden können.
  - Methoden:
    - \* `writePosterInfoInDB()`: die Methode holt sich mithilfe eines standardmäßigen XML-Parsers alle auf der Webseite [www.themoviedb.org](http://www.themoviedb.org) (hier muss man sich registrieren und einen Api-Key beantragen) vorhandenen Vorschauposter zu den Filmen in der Datenbank, und schreibt die URL dazu in die Datenbank. Die Poster holt

sich hinterher die MovieSearch-Komponente und übergibt sie im Movie-Objekt weiter an die GUI.

- \* `writeNouns()`: die Methode extrahiert mithilfe des SemanticParsers (dabei sollte aus performanztechnischen Gründen je eine neue Instanz für jedes Review verwendet werden) sequenziell alle Nominalphrasen aus allen Reviews der Datenbank und schreibt die Nomen, samt Polarität, zurück in die Datenbank. Dafür existiert dort eine extra Tabelle NOUNS.

- \* `writeTopNouns()`: die Methode analysiert mit Hilfe von Lucene und dessen Index-Searcher alle in der Datenbank (NOUNS) vorhandenen Nomen und wählt sich die wichtigsten (bis zu 5 Stück) pro Film aus, um sie in eine neue Tabelle (TOP\_NOUNS) zu schreiben. Diese Nomen werden hinterher benötigt, um ein Teil der Funktionalität /M500/ implementieren zu können: Filme mit "like" oder "don't like" bewerten, woraufhin ähnliche Filme angezeigt bzw. aus dem Ergebnis aussortiert werden. Die Wichtigkeit ergibt sich hierbei aus einer (Lucene intern) stochastischen Formel, die sowohl die Termfrequenz, als auch die Regelmäßigkeit eines Wortes beachtet. So sollten markante Wörter wichtiger sein, als ganz typische, die in jedem Review auftauchen ("Wookie" vs. "film").

- Die Klasse **LuceneIndexer** ist für die Erstellung des Suchindex' mit Lucene verantwortlich

- Sie enthält neben den statischen Attributen zur Datenbankabfrage, wie den SQL-Queries, vor allem Lucene-Klassenobjekte:

- \* `analyzer`: der Analyzer definiert die Wortfilter, die für die Indizierung der Texte verwendet werden sollen. Es soll für Movie Genie der `EnglishAnalyzer` verwendet werden, der zwei Filter definiert: `StopWord`- und `PorterStemFilter`. Ersterer sorgt dafür, dass für die englische Sprache typische Wörter, wie "the", "in" etc., die für die Wiedererkennung wenig Relevanz haben, für den Index nicht beachtet werden sollen, was die Performanz der Suche steigert. Der zweite Filter sorgt dafür, dass die Wörter in Wortgruppen eingeteilt werden, jedes Wort also auf ein Stammwort reduziert wird. Vereinfacht kann man sagen, dass Wörter wie "rainy" und "rains" zu "rain" zusammengefasst werden. Dadurch wird die Trefferquote der Suche gesteigert, falls der Benutzer bspw. die Mehrzahl eines Schlüsselwortes eintippt.

- \* `writer` und `config`: der `IndexWriterConfig` konfiguriert den `IndexWriter`, welcher dann den Suchindex erstellt, der folgendermaßen aufgebaut ist: ein Index besteht aus beliebig vielen Dokumenten (Datensätzen), wobei jedes Dokument aus beliebig vielen Feldern (Datenfeldern) besteht. So kann man direkt eine Datenbanktabelle auf einen Index abbilden.

- `computeMoviesIndex()` ist die einzige Methode und startet den eigentlichen Indizierungsprozess. Dafür muss zunächst ein Verzeichnis definiert werden, in das dann der writer die Dokumente (auf Basis der Datenbanksätze) schreibt. Pro Film müssen neben den harten Kriterien auch die zuvor vom PreProcessor erstellten Nomen der Reviews in den Index geladen werden.

## 4 Datenmodell

In diesem Abschnitt werden die für unser Projekt relevanten Entitäten und deren Beziehungen dargestellt und beschrieben. Die verschiedenen Daten werden in einer relationalen Datenbank gespeichert und stehen somit für die unterschiedlichen Abfragen unserer Webapplikation zur Verfügung. Die Daten lassen sich grob **3** verschiedenen Bereichen zuordnen. Dazu gehören Daten, die in direktem Zusammenhang zu einem Film stehen (Titel, Jahr, Schauspieler, Produzenten etc.), Daten die aus den verschiedenen Reviews gewonnen wurden (Nomen und deren Polarität), also den Bewertungen der Filme und letztendlich Daten, die dem Benutzer zugeordnet werden können. Durch diese Zuordnung können wir auf die wesentlichen Entitäten des Datenmodells und deren Attribute schließen. Zur Erstellung einer Benutzerdatenbank müssen wir uns auch über deren Eigenschaften klar werden. Die Benutzerdaten umfassen zum einen den Benutzernamen und das Passwort zur Authentifizierung auf der Webseite, zum anderen werden hier die durch den Benutzer positiv oder negativ bewerteten Filme aufgelistet (Siehe Wunschkriterium /W100/ aus dem Pflichtenheft). Hieraus können wir ablesen, dass eine Relation zwischen der User-Entität und der Movie-Entität hergestellt werden muss. Die Speicherung der Daten ist auf Seite der Datenbank mit einer zusätzlichen Datentabelle realisiert (siehe Abbildung 4.2).

Um den Aufwand des Datenimports zu reduzieren, werden zu den verschiedenen Datentabellen in der Datenbank zusätzlich **3** Views erstellt (Plot, Recommendations und Reviews). Die Views bieten eine Sicht auf externe Daten, also Daten, die sich nicht direkt in einer Datenbank befinden. Mit Lucene, einer Bibliothek, die die Volltextsuche in Java erleichtert, kann man die Inhalte der Reviews indexieren. Der Index wird zu allen Filmen aus der denormalisierten D\_Movies-Tabelle für jedes Dokument erstellt. Er besteht aus einem Dokument, welches wiederum aus Feldern aufgebaut ist. Die Felder können die in der D\_Movies-Tabelle enthaltenen Attribute speichern. Dieses wird für jedes Attribut explizit bestimmt. Außerdem wird die Art der Analyse festgelegt, also mit welchem Filter der Text bearbeitet wird (StopWord und PorterStemFilter). Auf den Index können dann die Suchanfragen mit den harten und weichen Kriterien angewendet werden. Um die Menge der relevanten Filme einzuschränken, sollte man zuerst die harten Kriterien auf den Index anwenden. Im Anschluss daran können die weichen Kriterien aus der Suchanfrage analysiert und je nach Relevanz ausgewertet werden. Die Relevanz wird durch das Vorkommen eines weichen Kriteriums in den Reviews eines Films im Vergleich zu seinem Vorkommen in den gesamten Reviews bestimmt. Dadurch lässt sich zum Beispiel sagen, dass das weiche Kriterium „good movie“ für die Auswahl eines Films eine weniger relevante Rolle spielt, da der Ausdruck in sehr vielen Reviews vorkommt. Mit Hilfe von openNLP lässt sich ein Index von Nomen und

zugehörigen Adjektiven erstellen. Die Polarität, also ob ein Nomen positiv, negativ oder neutral bewertet wird, kann mit SentiWord realisiert werden. SentiWord ist eine lexikalische Datenbank von ca. 117.000 englischen Wörtern. Dabei werden Nomen, Adjektive, Adverbien und Verben, die in bestimmten Kombinationen auftauchen, einer Polarität zugeordnet. Dadurch kann ein Ausdruck wie „bad movies“ als negativ bewertet und mit einer Konstanten NEGATIVE markiert werden.

## 4.1 Diagramm

Das Klassendiagramm in Abbildung 4.1 umfasst alle von uns zu erstellenden Tabellen mit ihren jeweiligen Attributen sowie die Beziehungen zu anderen Tabellen. Schlüsselattribute werden durch einen „Schlüssel“ dargestellt, zudem können die verschiedenen Datentypen (varchar, int, double, bit oder clob) inklusive der maximalen Zeichenanzahl abgelesen werden. Primärschlüssel dürfen keine Null – Werte annehmen und müssen eindeutig sein (im Diagramm wird dies durch das U, unique, dargestellt). Attribute, die einen Null – Wert enthalten dürfen, bekommen ein N.

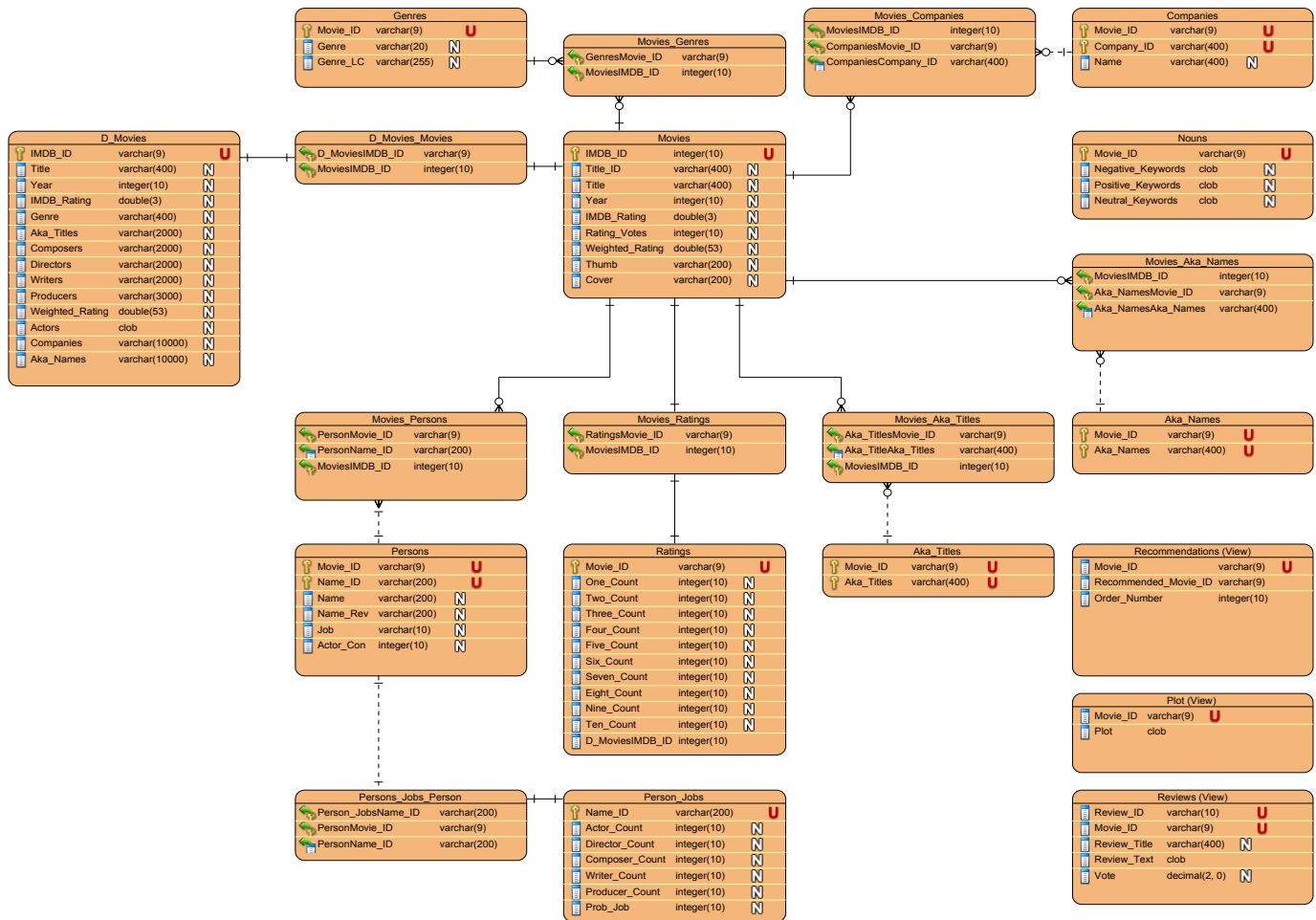


Abbildung 4.1: Klassendiagramm der Datenbank

Das Klassendiagramm stellt den grundsätzlichen Aufbau der unterschiedlichen Entitäten in unserer Datenbank dar. Desweiteren beinhaltet das Diagramm die 3 Views Plot, Recommendations und Reviews. Die Entität Nouns enthält eine indexierte Liste von Nomen und deren Polarität (positiv, negativ und neutral).

Zudem haben wir für die Benutzerverwaltung eine weitere Tabelle angelegt. Diese wird in Abbildung 4.2 gezeigt. Sie bezieht sich auf das Wunschkriterium /W100/ aus dem Pflichtenheft. Dabei geht es darum, einem Benutzer die Möglichkeit zu geben, Filme positiv oder negativ zu bewerten und diese Bewertung für ihn zu speichern. Sollte der Benutzer in einer weiteren Session nach Filmen suchen, können die bereits bewerteten Filme als Kriterien zur Suchanfrage hinzugezogen werden. Dazu erhält der Benutzer, nach seiner Einwilligung über die Speicherung der Daten, einen Cookie. Dieser Cookie enthält Informationen über die Session des Benutzers auf der Webseite und kann eine gewisse Zeit gespeichert werden. Sollte sich der Benutzer in der Zeit (bei uns sollen die Daten 30 Tage gespeichert werden) nicht mehr auf der Webseite anmelden, wird der Cookie und somit die Informationen die er enthält, gelöscht.

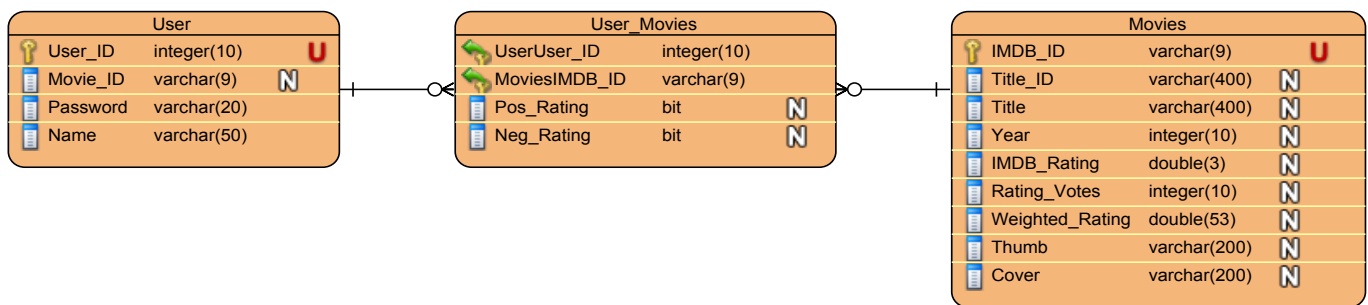


Abbildung 4.2: Klassendiagramm User

## 4.2 Erläuterung

Im Folgenden wird eine Tabelle gezeigt, die die Beziehungen zwischen den einzelnen Entitäten im Datenbankmodell des Movie Genies erläutert. Dabei wird auf die Kardinalitäten in der jeweiligen Beziehung eingegangen. Die MOVIES-Tabelle dient als Grundtabelle im Datenbank-Klassendiagramm. Sie liefert die eindeutige IMDB\_ID, eine Film-Id anhand welcher viele Beziehungen zu anderen Tabellen im Datenbankmodell hergestellt werden.

Entität	Beziehungen	
Movies_Aka_Titles	Titelsynonym	Mehrere Aka_Titles können zu genau einem Film aus der D_MOVIES-Tabelle gehören. Die Relation wird anhand der MOVIE_ID aus der D_Movies-Tabelle zugeordnet.
Movies_Aka_Names	Namensynonym	Ein Name aus den Aka_Names kann mehreren IMDB_IDs aus der D_Movies-Tabelle eingeteilt werden. Der Primärschlüssel der Relation besteht aus der D_MoviesIMDB_ID und der Aka_TitlesMovieID
Movies_Persons	Personenbeziehung	Mehrere Personen können mehreren MOVIE_IDs zugeordnet werden. Das bedeutet, dass ein Film aus mehreren Schauspielern besteht und ein Schauspieler in mehreren Filmen mitspielen kann. Die Zuordnung erfolgt durch die eindeutige MOVIE_ID.
Persons_Jobs_Person	Jobaufzählung	Für genau eine Person aus der Persons-Tabelle wird aufgelistet, wie oft sie in welchen Job tätig war. Dies passiert anhand der Zuordnung durch die eindeutige NAME_ID, welche in der Relationstabelle als Primärschlüssel verzeichnet ist.
Movies_Ratings	Ratingauflistung	Anhand von Movie_Ids aus der D_Movies-Tabelle wird genau einem Film genau eine Zeile mit Ratingcounts aus der Ratings-Tabelle eingeteilt. Der Primärschlüssel der Relationstabelle Persons_Jobs_Person besteht aus den Primärschlüsseln der teilnehmenden Tabellen Persons und Person_Jobs. Die Relation beschreibt eine One-To-One-Relation.
Person_Jobs	Jobsaufzählung	Für genau eine Person aus der PERSONS-Tabelle wird aufgelistet, wie oft sie in welchen Job tätig waren. Dies passiert anhand der Einteilung durch die eindeutige NAME_ID.
Movies_Companies	Companyzuordnung	Genau einem Film aus der D_MOVIE-Tabelle können mehrere Companies zugeteilt werden. Auf der anderen Seite kann eine Company mehrere Filme zugeordnet bekommen. Dies entspricht einer Many-to-Many-Beziehung zwischen Movies und Company. 



Entität	Beziehungen	
Movies_Genres	Genrezuordnung	Anhand von MOVIE_IDs werden genau einem Film mehrere Genres aus der GENRE-Tabelle zugewiesen. Ein Genre kann dabei mehreren Filmen zugeteilt werden. Daher besteht die Relation auf beiden Seiten aus einer One-To-Many-Relation. Sie wird durch die eindeutige IMDB_ID aus der D_Movies-Tabelle identifiziert.
D_Movies_Movies	Denormalisierung	Die D_Movies-Tabelle ist ein denormalisiertes Abbild der Movies-Tabelle mit all ihren Beziehungen. Das bedeutet, dass alle Informationen in D_Movies gesammelt wurden.

Es soll ebenfalls eine Userdatenbank erstellt werden, die die durch den User vorgenommenen Ratings zu einem Film speichert.

Entität	Beziehungen	
User_Movies	Ratingerstellung	Ein User kann für einen Film ein Rating speichern. Dies geschieht anhand der Zuordnung in der Relationstabelle User_Movies. Ein Rating des Users wird durch die eindeutige IMDB_ID genau einem Film zugeordnet. Die Beziehung aus der Seite der Movies-Tabelle beschreibt eine One-To-Many-Relation. Einem Film kann keine oder mehrere Ratings von Usern zugeteilt werden.

## 5 Serverkonfiguration

Für die Konfiguration des Servers zur Ausführung der Anwendung sind folgende Dateien notwendig: WEB-INF/web.xml (im Verzeichnis der Web-Applikation) conf/tomcat-users.xml (im Installationsverzeichnis von Tomcat)

### 5.1 web.xml

Hier werden die verwendeten XML-Schemata spezifiziert und das Wurzelement der Datei angegeben:

Listing 5.1: web.xml

```
1 <?xml version=1.0 encoding= UTF-8?>
2 <web-app id=WebApp_ID version= 2.5 xmlns= http://java.sun.com/xml/ns/javaee
3   xmlns:web= http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd
4   xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
5   xsi:schemaLocation=http://java.sun.com/xml/ns/javaee
6 http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd}>
```

Dieser Abschnitt beschreibt die Anwendung menschenlesbar und schaltet den Debug-Modus aus (ansonsten hätten Benutzer Zugriff auf Interna):

Listing 5.2: web.xml

```
1   <description>Movie Genie is a web application for verbose queries on movie data.</description>
2   <display-name>Movie Genie</display-name>
3   <context-param>
4     <description>MovieGenie production mode</description>
5     <param-name>productionMode</param-name>
6     <param-value>true</param-value>
7   </context-param>
```

Das zu startende Servlet wird angegeben (hier ein generisches Vaadin-Servlet, das die eigentliche Applikationsklasse wiederum selbst startet, festgelegt durch den Parameter application):

Listing 5.3: web.xml

```
1   <servlet>
2     <servlet-name>MovieGenie Application Servlet</servlet-name>
3     <servlet-class>com.vaadin.terminal.gwt.server.ApplicationServlet</servlet-class>
4     <init-param>
5       <description>MovieGenie application class to start</description>
6       <param-name>application</param-name>
7       <param-value>de.tu_bs.sep2012.ifis.mgenie_gui.MainSite</param-value>
8     </init-param>
9   </servlet>
```

Schließlich wird noch festgelegt, über welche URL-Pfade die Anwendung ansprechbar ist (in diesem Fall über alle, "/\*"), und nach welcher Zeitraum eine inaktive Benutzersitzung beendet wird:

Listing 5.4: web.xml

```
1 <servlet-mapping>
2   <servlet-name>MovieGenie Application Servlet</servlet-name>
3   <url-pattern>/*</url-pattern>
4 </servlet-mapping>
5 <session-config>
6   <session-timeout>30</session-timeout>
7 </session-config>
```

Um die Anwendung per HTTP-Authentifizierung zu schützen, sind zusätzlich folgende Zeilen notwendig, die von einer Tomcat-Rolle Gebrauch machen, die zusammen mit einem Benutzer in Abschnitt 5.2 konfiguriert wird.

Listing 5.5: web.xml

```
1 <security-constraint>
2   <web-resource-collection>
3     <web-resource-name>
4       MovieGenie
5     </web-resource-name>
6     <url-pattern>/*</url-pattern>
7   </web-resource-collection>
8   <auth-constraint>
9     <role-name>mgenie-user</role-name>
10  </auth-constraint>
11 </security-constraint>
12 <login-config>
13   <auth-method>BASIC</auth-method>
14   <realm-name>Movie Genie</realm-name>
15 </login-config>
16 </web-app>
```

## 5.2 tomcat-users.xml

Die Datei muss innerhalb des Wurzelements um folgende Zeilen ergänzt werden, wobei USER und PASSWORD durch die einzustellenden Zugangsdaten zu ersetzen sind. Durch diese Einstellungen wird ein neuer Benutzer angelegt, mit dem sich Nutzer der Anwendung per HTTP-Authentifizierung einloggen müssen.

Listing 5.6: tomcat-users.xml

```
1 <role rolename="mgenie-user"/>
2 <user username="USER" password="PASSWORD" roles="mgenie-user"/>
```