DKE PR

Semantic Web Service

Kino

Inhaltsverzeichnis

[1 Anleitung zum Starten der Applikation](#_Toc31129036)

[2 Projektorganisation](#_Toc31129037)

[3 Screenshots des Frontends](#_Toc31129038)

[4 Lösungsarchitektur](#_Toc31129039)

[4.1 Grundsätzliches](#_Toc31129040)

[4.1.1 Frontend & Services](#_Toc31129041)

[4.1.2 Triple Store & SPARQL Endpoint](#_Toc31129042)

[4.2 Webservice](#_Toc31129043)

[4.3 Metaservice](#_Toc31129044)

[4.3.1 Controller](#_Toc31129045)

[4.3.1.1 AbstractController](#_Toc31129046)

[4.3.1.2 AdminController](#_Toc31129047)

[4.3.1.3 IndexController](#_Toc31129048)

[4.3.1.4 MoviesController](#_Toc31129049)

[4.3.1.5 SnacksController](#_Toc31129050)

[4.3.2 Fuseki](#_Toc31129051)

[4.3.2.1 FusekiQueryExecution](#_Toc31129052)

[4.3.3 Names](#_Toc31129053)

[4.3.3.1 MetadataTypes](#_Toc31129054)

[4.3.4 Objects](#_Toc31129055)

[4.3.4.1 Metadata](#_Toc31129056)

[4.3.5 Services](#_Toc31129057)

# 

# Anleitung zum Starten der Applikation

Zum erfolgreichen Start der Anwendung wird Maven und die dazugehörigen Befehle benötigt (mvn spring-boot:run). Zum Starten reicht es, die “start\_semantic\_web.bat” Datei auszuführen.

Vorrausetzungen für einen erfolgreichen Start der Anwendung:

* Java JDK muss installiert sein (Getestet unter JAVA 11 -> wird benötigt aufgrund der Files.readString(...) Methode. Ältere Versionen funktionieren möglicherweise nicht)
* Maven muss installiert/in den Umgebungsvariablen eingefügt sein
* Die Umgebungsvariable “JAVA\_HOME” muss auf den Ordner der JAVA JDK zeigen (nicht auf das bin Verzeichnis)
* Wir haben es auf mehreren Rechnern getestet. Bei den meisten hat es funktioniert - es kam jedoch vereinzelt zu Komplikationen mit z.B. der JAVA Version.

# Projektorganisation

Die Verantwortlichkeiten wurden wie folgt aufgeteilt:

MetaService + Frontend: Florian Pössl, Ronald Moritz

WebService 1: Erkut Karadeniz

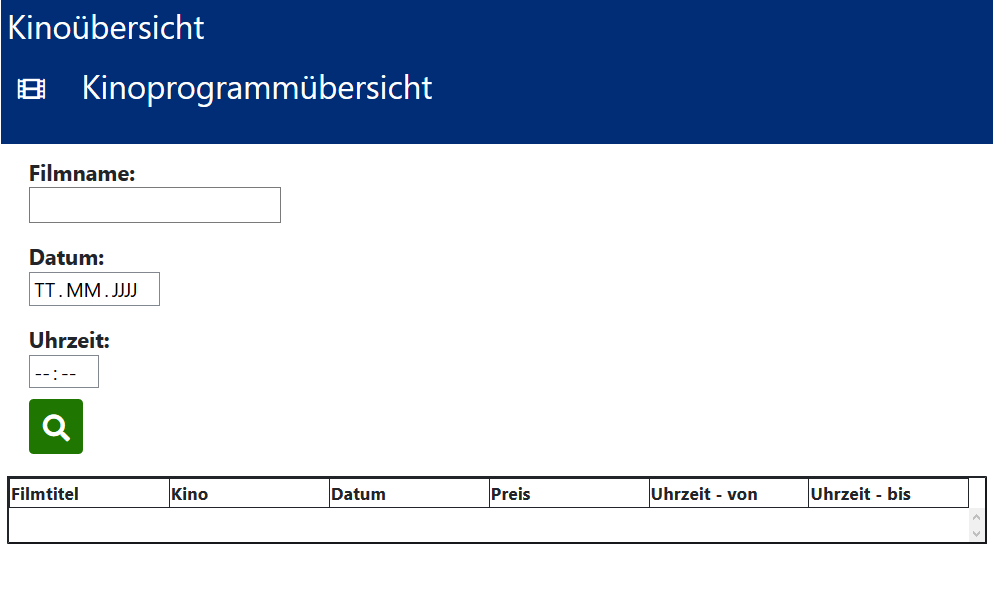
Webservice 2: Florian Pössl, Ronald Moritz, Erkut Karadeniz

# Screenshots des Frontends

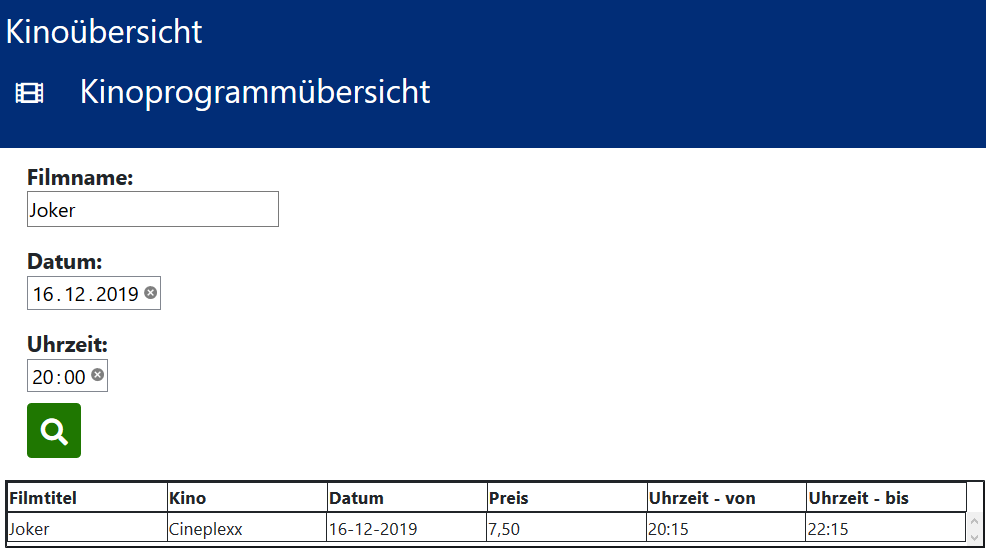
## Startseite:



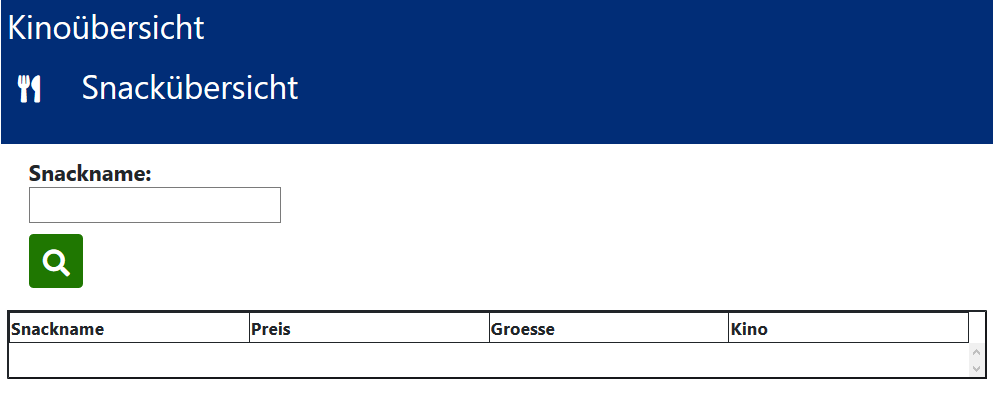
## Kinoprogramm anzeigen:



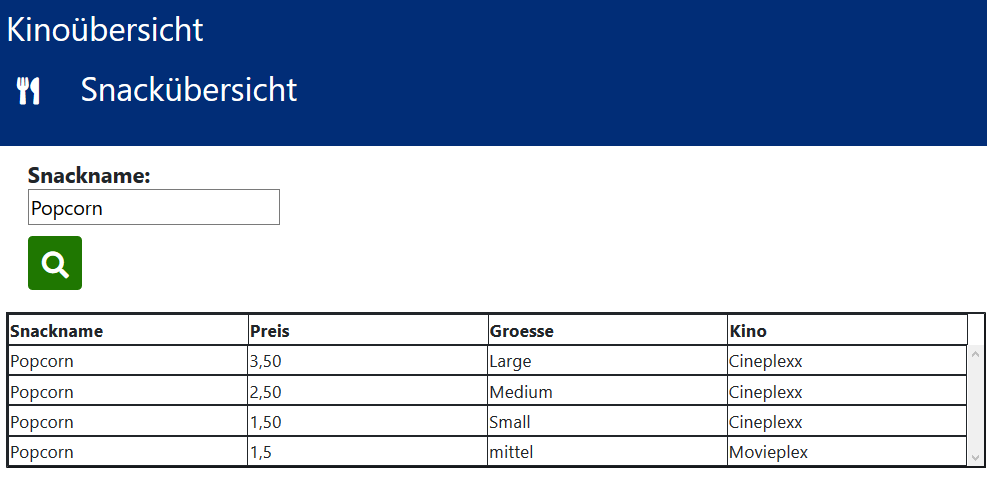
## Beispielhafte Suche nach dem Film “Joker”:



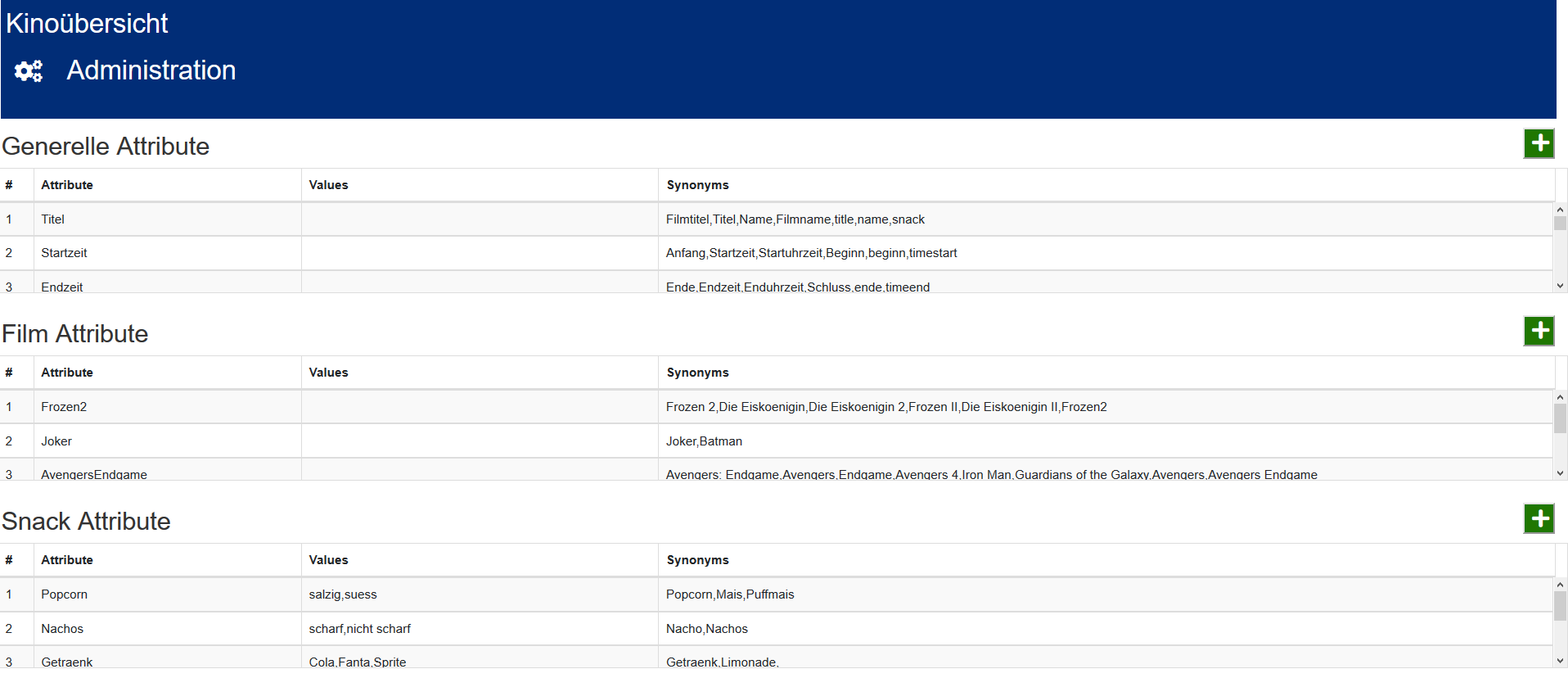
## Snackübersicht:



## Beispielhafte Suche nach “Popcorn”:



## Administration:



# Lösungsarchitektur

## Grundsätzliches

### Frontend & Services

Als Frontend wurden einfache HTML Seiten angelegt und mittels Javascript wurde die Kommunikation mit dem Backend implementiert. Die Anwendungen basieren auf Spring-Boot und dementsprechend wurden auch die benötigten REST-Controller implementiert.

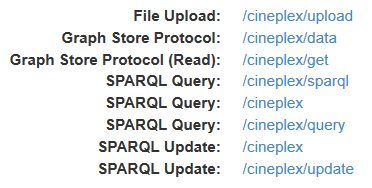
### Triple Store & SPARQL Endpoint

Als SPARQL Server wird Apache Jena Fuseki verwendet. Es handelt sich hierbei um die Standalone-Version. Der Fuseki-Server bietet durch TDB einen persistenten Storage-Layer an. Außerdem können darauf Queries ausgeführt werden.

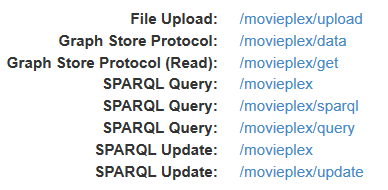
Um den Metaservices ordnungsgemäß verwenden zu können, muss ein Fuseki-Server auf Port 3330 gehostet werden (wird durch ausführen der .bat Datei automatisch gemacht).

Die SPARQL Endpoints dazu sehen wie folgt aus:

Für die Filme:



## Für die Snacks:



## Webservice

Die beiden Webservices basieren auf einer Spring Boot Anwendung. Es gibt einen Controller, in dem die Anfragen des Metaservices angenommen werden. Es gibt jeweils einen “movieWebservice” und einen “snackWebservice” REST Endpoint und dieser regelt die Lieferung der jeweiligen Daten. Beim Start der Webservices werden die Daten einmal aus einem RDF/XML und bei dem anderen aus einem Turtle File geladen. Das heißt bei einem Webservice wird mit RDF/XML gearbeitet und bei dem Anderen mit Turtle. Diese geben die Daten bezüglich Filme und Snacks auch in dem verwendeten Format zurück an den Metaservice.

## Metaservice

### Controller

#### AbstractController

Ein abstrakter Controller welcher nur als Basis für die anderen Controller fungiert. In diesem Controller wird der MetadataService instanziert.

#### AdminController

In diesem Controller werden alle Requests bezüglich der Admin Seite bearbeitet.

**/admin**

Diese Methode regelt das Laden der admin.html Seite.

**/loadMetadata**

In dieser Methode werden die Daten für die Konfigurationstabelle aus drei JSON Files geladen und zurück gegeben.

**/saveMetadata**

In dieser Methode werden die Daten aus der Konfigurationstabelle mitgegeben und in die drei JSON Files gespeichert, aus denen die Daten ursprünglich kommen.

#### IndexController

In diesem Controller werden alle Requests bezüglich der Startseite bearbeitet.

**/index**

Diese Methode regelt das Laden der index.html Seite.

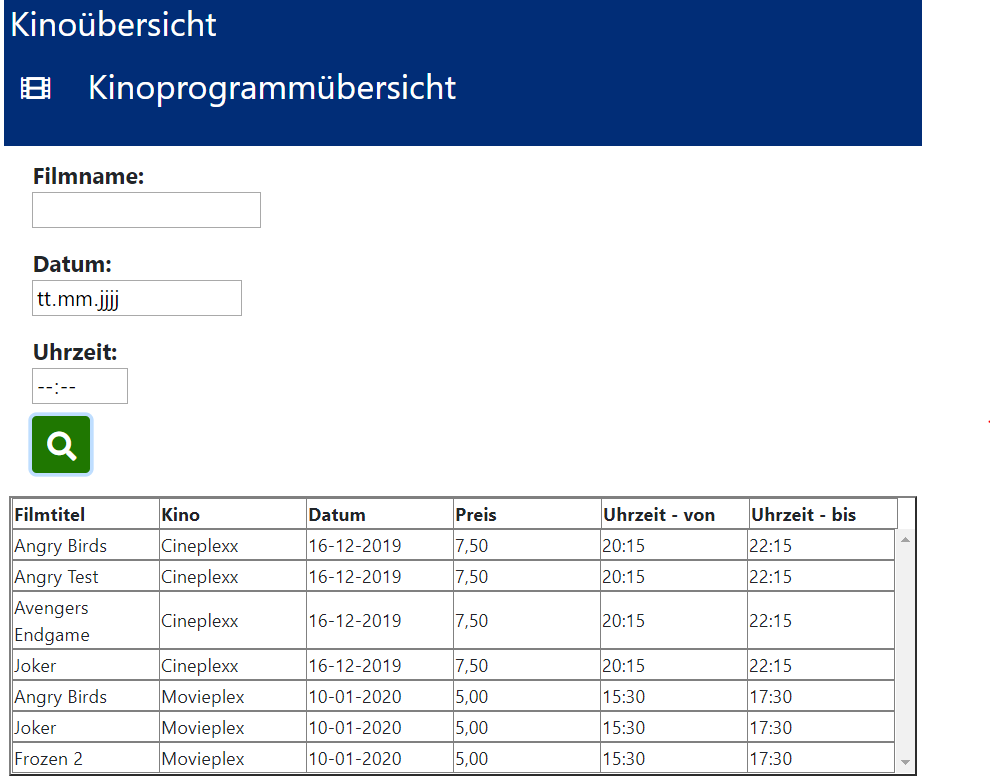
#### MoviesController

In diesem Controller werden alle Requests bezüglich der Movies Seite bearbeitet.

**/movies**

Diese Methode regelt das Laden der movies.html Seite.

Vor dem Anzeigen der Seite werden die Daten von den Webservices geladen und in den Fuseki Server des Metaservices gespeichert bzw. die bestehenden Daten werden aktualisiert. Bevor die Daten gespeichert werden, werden diese analysiert und auf das Schema des Fuseki Servers des Metaservices angepasst. Dazu werden die Daten in der Konfigurationstabelle der Admin Seite verwendet.

****

Ein Klick auf den grünen Suchbutton löst den searchMovies Request aus. Falls keine Suchparameter angegeben werden, werden die dazugehörigen Teile der Query einfach leer gelassen und nicht berücksichtigt (Wenn in keinem der drei Felder ein Wert eingegeben wird, werden z.B. alle Filme aller Kinos zurück gegeben)

**/searchMovies**

Parameter:

movieName: der eingegebene/gesuchte Film

moviedate: das gesuchte Datum

movietime: der gesuchte Startzeitpunkt

Mit den drei Eingangsparametern wird dynamisch eine SPARQL Query erstellt, welche dann am Fuseki Server des Metaservices ausgeführt wird.

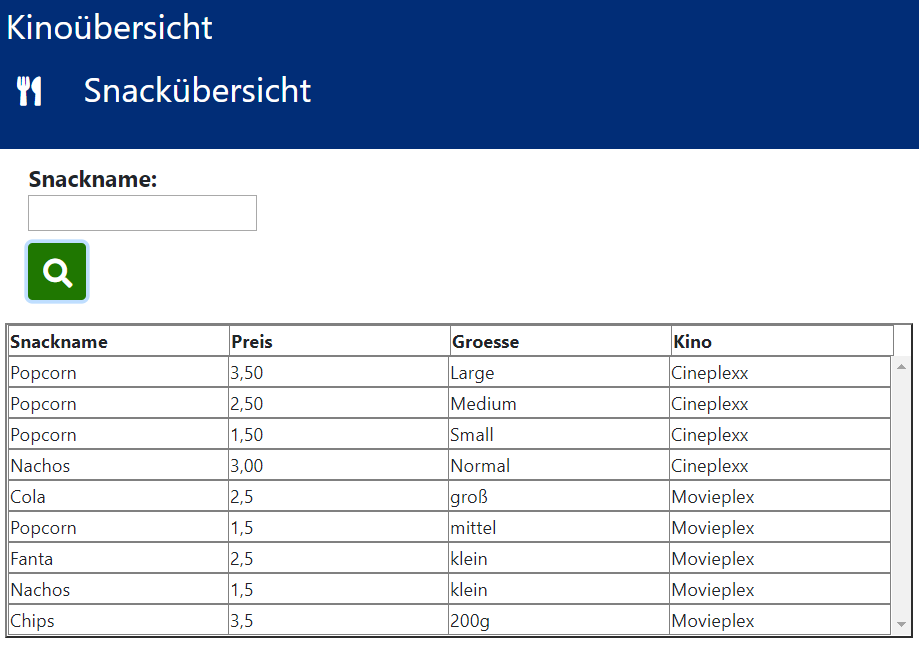
#### SnacksController

In diesem Controller werden alle Requests bezüglich der Snacks Seite bearbeitet.

**/snacks**

Diese Methode regelt das Laden der snacks.html Seite.

Vor dem Anzeigen der Seite werden die Daten von den Webservices geladen und in den Fuseki Server des Metaservices gespeichert bzw. die bestehenden Daten werden aktualisiert. Bevor die Daten gespeichert werden, werden diese analysiert und auf das Schema des Fuseki Servers des Metaservices angepasst. Dazu werden die Daten in der Konfigurationstabelle der Admin Seite verwendet.



Ein Klick auf den grünen Suchbutton löst den searchSnacks Request aus. Falls keine Suchparameter angegeben werden, werden die dazugehörigen Teile der Query einfach leer gelassen und nicht berücksichtigt (Wenn in keinem der drei Felder ein Wert eingegeben wird, werden z.B. alle Filme aller Kinos zurück gegeben)

**/searchSnacks**

Parameter:

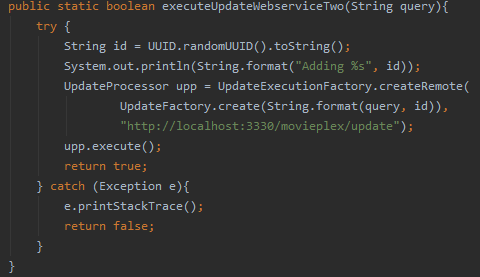
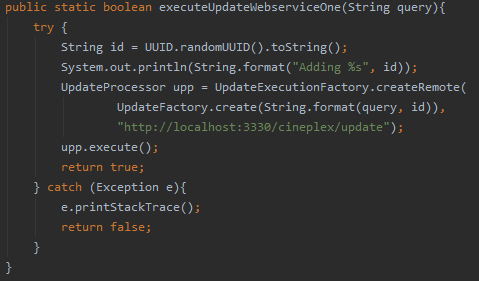
snackName: der eingegebene/gesuchte Snack

Mit den drei Eingangsparametern wird dynamisch eine SPARQL Query erstellt, welche dann am Fuseki Server des Metaservices ausgeführt wird.

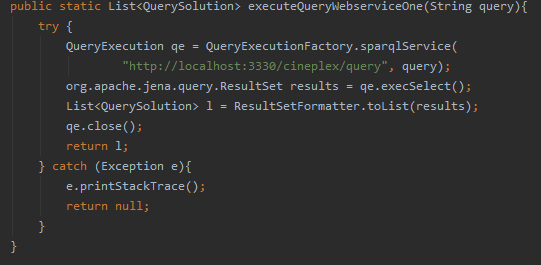
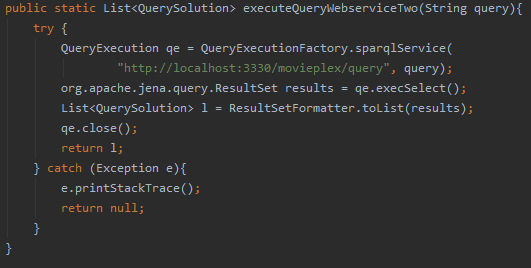
### Fuseki

#### FusekiQueryExecution

Diese Klasse dient dazu, um auf SPARQL Endpoint des Fuseki-Servers zugreifen zu können. Hierbei gibt es vier verschiedene Methoden: executeUpdateWebserviceOne(String query), executeUpdateWebserviceTwo(String query), executeQueryWebserviceOne(String query) und executeQueryWebserviceTwo(String query).



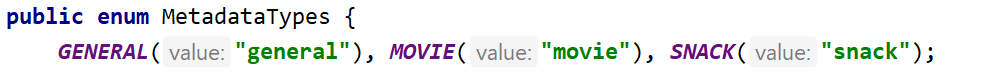
Die Methoden executeUpdateWebserviceOne(String query) und executeUpdateWebserviceTwo(String query) sind für alle Updates des Triple Stores (d.h. INSERT, UPDATE und DELETE Operationen auf das Dataset) zuständig. Bei einem erfolgreichen Update wird ein Boolean mit dem Wert true zurückgegeben, andernfalls wird false zurückgegeben. Je nachdem welche Methode verwendet wird, wird das Update auf das jeweilige Dataset ausgeführt.



Die Methoden executeQueryWebserviceOne(String query) und executeQueryWebserviceTwo(String query) sind für alle Abfragen auf das Dataset des Fuseki-Servers zuständig. Es wird ein Parameter mit der Query als String übergeben und eine List<QuerySolution> zurückgegeben. Falls die Abfrage nicht durchgeführt werden konnte, wird NULL zurückgegeben. Je nachdem welche Methode verwendet wird, wird die Query auf das jeweilige Dataset ausgeführt.

### Names

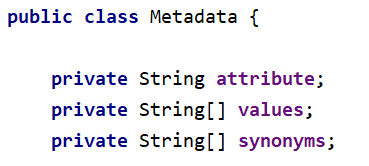
#### MetadataTypes



Kleine Hilfsklasse zur Unterscheidung der einzelnen Metadata bezüglich genereller Daten und Film- und Snackdaten. Wird beim Import und Export in die JSON Files verwendet.

### Objects

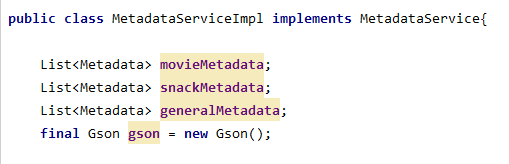
#### Metadata



Diese Klasse dient zur Verwendung der verschiedenen Metadaten im JAVA Code. Wenn z.B. nach “Angry” gesucht wird. Wird in den Synonymen einer Metadata Instanz geschaut, ob “Angry” darin vorhanden ist. Wenn dies der Fall ist, wird die Suchanfrage z.B. auf “Angry Birds” generalisiert. “Angry Birds” repräsentiert hierbei die Variable attribute.

### Services

MetadataService/MetadataServiceImpl



Dieser Service erstellt zur Laufzeit drei Listen mit Metadaten. Diese Listen werden dann für diverse Operation verwendet (z.B. die Generalisierung der Suchanfrage wie oben beschrieben).