# 1 Disclaimer

For the sake of readability, we do not include too much of our code base in here, if you want to have a look at the code, you can do so at all times by visiting our github repository. Where applicable, we provide more specific links, maybe also to *outsourced* repositories. At the end of this document we also provide a list of links to source code that might be of interest.

# 2 Emotional Reasoning Chat Bot

## 2.1 Introduction

The goal of the proseminar was to create a knowledge graph for a given domain. As our domain we chose to create a chat bot that is able to perform emotional reasoning.

> Emotional reasoning is a cognitive process by which a person concludes that his/her emotional reaction proves something is true, regardless of the observed evidence.
>
> -- *Wikipedia*

This project will collect and design ontologies in order to represent human emotions, as well as design reasoning mechanisms employing these. The reasoning mechanisms can be inbuilt in practical scenarios e.g. in online marketing - designing of a chat bot that appeals to emotions of humans and trying to change their behaviour. Or a character that goes beyond the "personal assistant" mode.

We will make use of this on a specific domain, namely movie recommendation. By trying to extract emotions out of pre-exisiting movie reviews, we mapped emotions to movies, and by extracting emotions out of the users chat log, a reasoning mechanism will try to find a good movie recommendation for the user.

### 2.1.1 Domain Overview

Chat bots are programs where you can communicate via text with an artificial agent. These programs are usually designed to simulate how humans will react or behave in a conversation. In a perfect world, chat bots should be capable of recognizing human emotions in order to respond more smoothly with situation awareness.

In general, our choice of movies are dependent on our mood. So in order to actually get something out of our bot, we focussed on a chat bot that recommends movies depending on the user's mood.

While talking to the chat bot, it is able to detect your mood (happy, sad, angry...), and from peoples earlier reviews can recommend the best matching movies.

### 2.1.2 Use Case

Below the application's typical behaviour is described:

1. A user sends a message - e.g.: "Hello"
2. The bot responds with an arbitrary message - e.g.: "How are you?"
3. The user responds - e.g.: "I am sad. My cat died this morning."
4. The user's message is analyzed in order to obtain an emotion.
5. The emotion is used to suggest a movie to the user.
6. The user can choose if they would like to watch a movie reflecting their current mood or if they would rather try to change their current mood.
7. Based on the user's choice a movie is recommended.

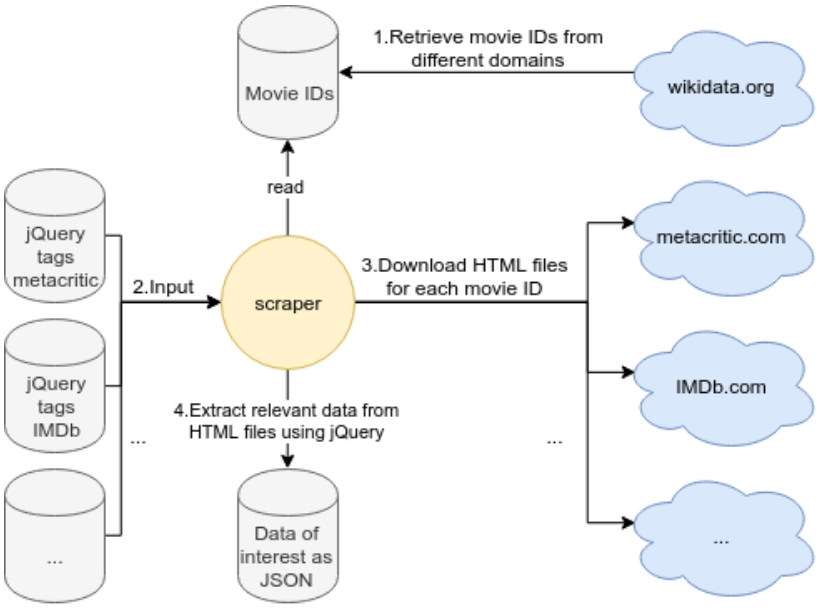
# 2.2 Data Sources

Sadly, there was no data set to be found that provided everthing we needed, however there are web sites out there that do so. Thus, in order to categorize movies, preprocessing steps had to be made. We gathered information about movies from IMDb and metacritic using web scraping techniques. Since different resources (movies) have different URLs, we needed to somehow gather the movie's internal ids for both web sites. We did so using the *wikidata knowledge base*, which provides exactly that with the `P345` property for IMDb movie IDs as well as the `P1712` property for metacritic IDs. So by using the following query, we were able to retrieve a list of all IMDb IDs, and if there is also a metacritic ID for that movie, we get that as well.

```
SELECT ?item ?imdb ?metacritic
WHERE
{
  ?item wdt:P345 ?imdb.
  OPTIONAL {?item wdt:P1712 ?metacritic}
}
```

## 2.2.1 Web Scraping

For the gathering of data to be done effeciently and scalable, a generalized web scraping service was built. It iterates over every movie ID we previously retrieved, then constructs the necessary *URL*s to send an *HTTP* request in order to get the *HTML* files. From these *HTML* files it constructs a virtual *Document Object Model (DOM)*, with respective predefined tags, it then can use *jQuery* on the *DOM* to extract the data of interest.




## 2.2.2 Other Metadata

Initially, we planned to provide links to linkedmbd, however, sadly this service seems to be discontinued.

Since the IDs were gathered using the *wikidata knowledge base*, there also for every movie we have in our database, is an equivalent on wikidata, Due to that, we were easily able to create linking to wikidata, and thus, profit from the data provided, by using federated *SPARQL*-queries.

Also since *DBpedia* maintains links in form of *owl:sameAs* relations to *wikidata*, we were also able to enrich our database with links to *DBpedia*.

```
PREFIX schema: <http://schema.org/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
```

```
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
SELECT ?film ?wdlink ?dblink WHERE {
    SERVICE <http://dbpedia.org/sparql> {
        ?dblink a dbpedia-owl:Film .
        ?dblink owl:sameAs ?wdlink
    }
    ?film a schema:Movie .
    ?film owl:sameAs ?wdlink
}
```

# 2.3 Knowledge Graph

The requirements to our knowledge graph were kind of clear from the get-go. They also did not change too much, making lots of refactoring unnecessary. At the end of this section a visualization of the main parts of our graph is provided.

## 2.3.1 Vocabularies Used

Reuse of existing vocabularies is important, since the definition says, that an ontology should be a *shared* conceptualization.

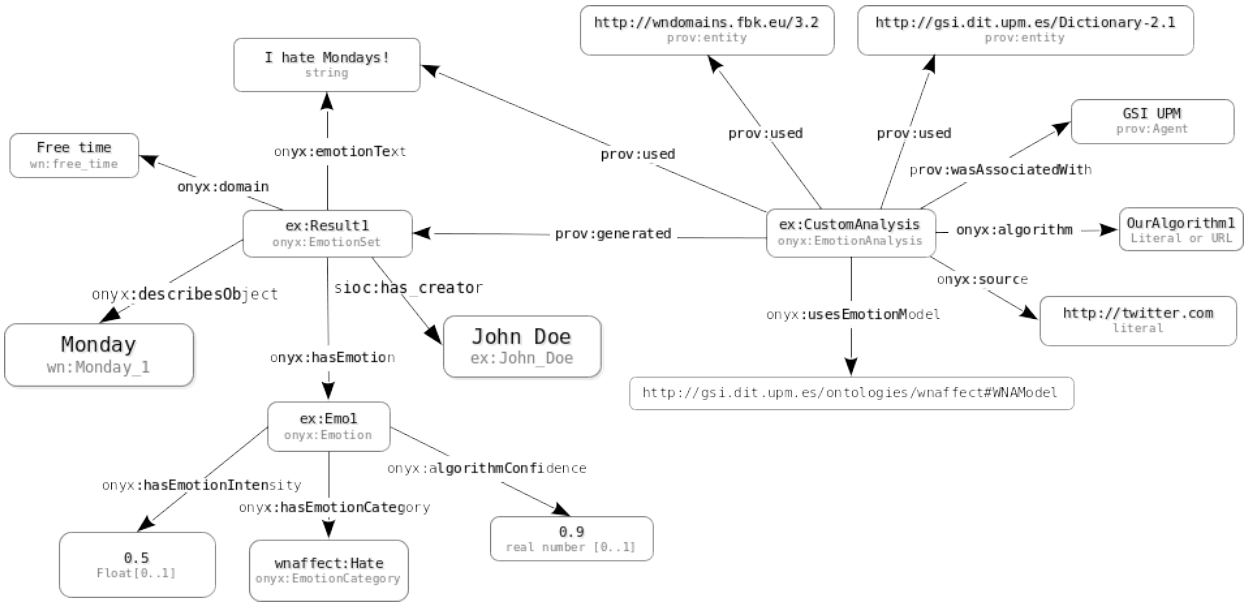
### 2.3.1.1 Schema.org

> Schema.org is a collaborative, community activity with a mission to create, maintain, and promote schemas for structured data on the Internet, on web pages, in email messages, and beyond.
>
> -- *https://schema.org/*

We mainly make use of *schema.org*'s movie and review model, to describe our scraped data sets.

### 2.3.1.2 Onyx - An Emotion Modelling Ontology

Onyx was designed for modelling emotions which have been extracted from text. This suits our needs perfectly, since we want to extract the user's emotional state from their chat messages as well as the emotional response to a movie from the movie's reviews. A basic example below shows a single opinion annotated with Onyx metadata (taken from the specification page):



Onyx: A Linked Data Approach to Emotion Representation (paper)

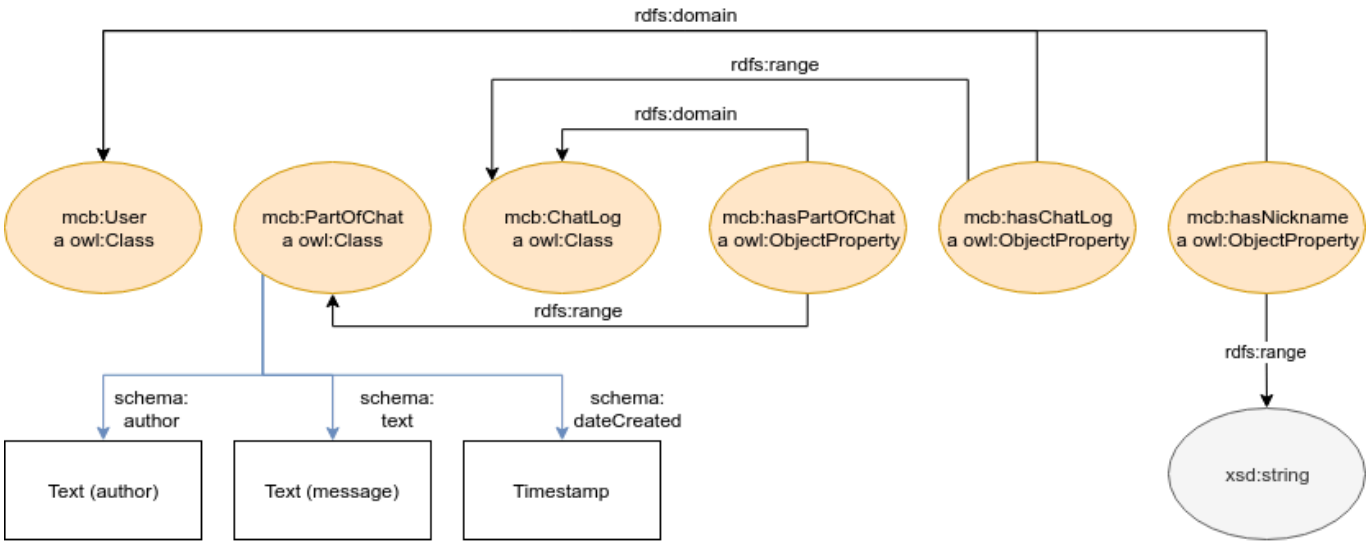
#### 2.3.1.3 WNAffect

*WNAffect* was designed to link words to affects (emotions). Again this makes a lot of sense for us and *Onyx* uses the terminology of *WNAffect*.

#### 2.3.1.4 Self Defined Parts

##### 2.3.1.4.1 User Data

Since we wanted to be able to restore the chat log of a user, we had to somehow model that in our graph. Usually, you probably would use some other type of database, for instance, document stores. However, we did not want to include even more dependencies, so we decided to just stick with *GraphDB* for that.




##### 2.3.1.4.2 NLP Review Annotations

We used the ontotext tagging service, for annotating natural language. Ontotext provides a REST-API to their service, which is not as powerful (as in computational load capacities), as we initially expected, thus we did not annotate that many reviews (since there are about a million in our graph "that many" is relative).

There are several different types provided by Ontotext. Parts of natural language texts are then assigned to those types, according to the underlying algorithms.

As an example for a successful annotation: "...disney bought everything from marvel comic..." was classified as `ontotext:RelationAcquisition`.

However, the linked ontology is nowhere to be found, we still keep the types though and use some artificial namespace.

##### 2.3.1.4.3 Owl Axioms

Owl axioms are used to provide information about classes and properties, as well as to associate class and properties with either partial or complete specifications of their characteristics, and to give other logical information about classes and properties.


#### 2.3.1.4.4 Schema.org Action

In order to actually make use of a schema.org action, we implemented a REST-API, which on `/api/v1/movie/:id` returns a `potentialAction`. It's a watch action and the target property leads to the IMDb videogallery site for a specific movie, since we can not link from an IMDb ID to some web site where you can actually watch the full movie.

```
"@context": "http://schema.org",
"@type": "Movie",
```

```
"@id": "/api/v1",
"title": "Pulp Fiction",
"potentialAction": {
    "@type": "WatchAction",
    "target": "https://www.imdb.com/title/tt0110912/videogallery?ref_=tt_pv_vi_sm"
}
```

#### 2.3.1.4.5 Shacl Shapes

Since one could theoretically add any kind of triple, we use *SHACL* shapes in order to validate our produced RDF-files. We do that mainly for movie-related data.

##### 2.3.1.4.5.1 User/Chat Related Data

```
@prefix dash: <http://datashapes.org/dash#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix schema: <http://schema.org/> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix mcb: <http://movie.chatbot.org/> .

mcb:UserShape
    a sh:NodeShape ;
    sh:targetClass mcb:ChatLog ;
    sh:closed false ;
    sh:property [
        sh:path mcb:hasChatLog ;
        sh:datatype sh:IRI ;
        sh:maxCount 1 ;
        sh:minCount 1 ;
        sh:severity sh:Warning ;
        sh:message "A user should have exactly 1 ChatLog!"@en ;
    ] ;
    sh:property [
        sh:path mcb:hasNickName;
        sh:datatype xsd:string ;
        sh:pattern "[a-zA-Z0-9]" ;
        sh:minLength 1 ;
        sh:maxLength 16 ;
        sh:maxCount 1 ;
        sh:minCount 1 ;
        sh:severity sh:Violation ;
        sh:message "A user has to have exactly 1 nickname!"@en ;
    ] .

mcb:ChatLogShape
    a sh:NodeShape ;
    sh:targetClass mcb:ChatLog ;
    sh:property [
        sh:path mcb:hasPartOfChat ;
        sh:datatype sh:IRI ;
        sh:minCount 1 ;
    ] .

mcb:PartOfChatShape
    a sh:NodeShape ;
```

```
    sh:targetClass mcb:PartOfChat ;
    sh:property [
        sh:path schema:author ;
        sh:datatype xsd:string ;
        sh:pattern "[a-zA-Z0-9]" ;
        sh:minLength 1 ;
        sh:maxLength 16 ;
        sh:maxCount 1 ;
        sh:minCount 1 ;
    ] ;
    sh:property [
        sh:path schema:text ;
        sh:datatype xsd:string ;
        sh:pattern "[a-zA-Z0-9!?.]" ;
        sh:minLength 1 ;
        sh:maxLength 256 ;
        sh:maxCount 1 ;
        sh:minCount 1 ;
    ] ;
    sh:property [
        sh:path schema:dateCreated ;
        sh:datatype xsd:integer ;
        sh:maxCount 1 ;
        sh:minCount 1 ;
        sh:minInclusive 0 ;
    ] .
```

An example for a faulty validation run looks like this:

```
$ Conforms? false
$  - Severity: Violation for http://www.w3.org/ns/shacl#DatatypeConstraintComponent
$  - Severity: Violation for http://www.w3.org/ns/shacl#MaxCountConstraintComponent
$  - Severity: Violation for http://www.w3.org/ns/shacl#MinInclusiveConstraintComponent
$ ValidationReport {
$  graph:
$   [
$    { '@id': '_:b0',
$       '@type': [ 'http://www.w3.org/ns/shacl#ValidationResult' ],
$       'http://www.w3.org/ns/shacl#focusNode': [
$            { '@id': 'http://movie.chatbot.org#o15484207065' }
$        ],
$       'http://www.w3.org/ns/shacl#resultMessage': [
$            { '@value': 'Value does not have datatype <http://www.w3.org/2001/XMLSchema#string>' }
$        ],
$       'http://www.w3.org/ns/shacl#resultPath': [
$            { '@id': 'http://schema.org/author' }
$        ],
$       'http://www.w3.org/ns/shacl#resultSeverity': [
$            { '@id': 'http://www.w3.org/ns/shacl#Violation' }
$        ],
$       'http://www.w3.org/ns/shacl#sourceConstraintComponent': [
$            { '@id': 'http://www.w3.org/ns/shacl#DatatypeConstraintComponent' }
$        ],
$       'http://www.w3.org/ns/shacl#sourceShape': [
$            { '@id': '_:b1' }
$        ],
$       'http://www.w3.org/ns/shacl#value': [
$            { '@type': 'http://www.w3.org/2001/XMLSchema#integer',
$              '@value': '1' }
```

```
$         ]
$     },
$     { '@id': '_:b2',
$       '@type': [ 'http://www.w3.org/ns/shacl#ValidationResult' ],
$       'http://www.w3.org/ns/shacl#focusNode': [
$            { '@id': 'http://movie.chatbot.org#o15420147065' }
$         ],
$       'http://www.w3.org/ns/shacl#resultMessage': [
$            { '@value': 'More than 1 values' }
$         ],
$       'http://www.w3.org/ns/shacl#resultPath': [
$            { '@id': 'http://schema.org/author' }
$         ],
$       'http://www.w3.org/ns/shacl#resultSeverity': [
$            { '@id': 'http://www.w3.org/ns/shacl#Violation' }
$         ],
$       'http://www.w3.org/ns/shacl#sourceConstraintComponent': [
$            { '@id': 'http://www.w3.org/ns/shacl#MaxCountConstraintComponent' }
$         ],
$       'http://www.w3.org/ns/shacl#sourceShape': [ { '@id': '_:b1' } ] },
$     { '@id': '_:b3',
$       '@type': [ 'http://www.w3.org/ns/shacl#ValidationResult' ],
$       'http://www.w3.org/ns/shacl#focusNode': [
$            { '@id': 'http://movie.chatbot.org#o15420147065' }
$         ],
$       'http://www.w3.org/ns/shacl#resultMessage': [
$            { '@value': 'Value is not >= 0' }
$         ],
$       'http://www.w3.org/ns/shacl#resultPath': [
$            { '@id': 'http://schema.org/dateCreated' }
$         ],
$       'http://www.w3.org/ns/shacl#resultSeverity': [
$            { '@id': 'http://www.w3.org/ns/shacl#Violation' }
$         ],
$       'http://www.w3.org/ns/shacl#sourceConstraintComponent': [
$            { '@id': 'http://www.w3.org/ns/shacl#MinInclusiveConstraintComponent' }
$         ],
$       'http://www.w3.org/ns/shacl#sourceShape': [ { '@id': '_:b4' } ],
$       'http://www.w3.org/ns/shacl#value': [
$            { '@type': 'http://www.w3.org/2001/XMLSchema#integer',
$            '@value': '-1' }
$         ]
$      },
$     { '@id': '_:b5',
$       '@type': [ 'http://www.w3.org/ns/shacl#ValidationReport' ],
$       'http://www.w3.org/ns/shacl#conforms':
$        [ { '@type': 'http://www.w3.org/2001/XMLSchema#boolean',
$            '@value': 'false' } ],
$       'http://www.w3.org/ns/shacl#result': [
$             { '@id': '_:b0' },
$             { '@id': '_:b2' },
$             { '@id': '_:b3' }
$         ]
$      }
$     ],
$   validationNode:
$    { '@id': '_:b5',
$      '@type': [ 'http://www.w3.org/ns/shacl#ValidationReport' ],
$      'http://www.w3.org/ns/shacl#conforms':
```

```
$       [
$          { '@type': 'http://www.w3.org/2001/XMLSchema#boolean',
$            '@value': 'false' }
$       ],
$      'http://www.w3.org/ns/shacl#result': [
$          { '@id': '_:b0' },
$          { '@id': '_:b2' },
$          { '@id': '_:b3' }
$       ]
$     }
$}
```

The validated graph:

```
<http://movie.chatbot.org#o15484201470> a <http://movie.chatbot.org/PartOfChat>;
    <http://schema.org/author> "Trumpy";
    <http://schema.org/dateCreated> 1548420147065;
    <http://schema.org/text> "Welcome o!" .

<http://movie.chatbot.org#o15484207065> a <http://movie.chatbot.org/PartOfChat>;
    <http://schema.org/author> 1;
    <http://schema.org/dateCreated> 1548420147065;
    <http://schema.org/text> "Welcome o!" .

<http://movie.chatbot.org#o15420147065> a <http://movie.chatbot.org/PartOfChat>;
    <http://schema.org/author> "Mandela";
    <http://schema.org/author> "Trumpy";
    <http://schema.org/dateCreated> -1;
    <http://schema.org/text> "Welcome o!" .


<http://movie.chatbot.org#o> a <http://movie.chatbot.org/User>;
    <http://movie.chatbot.org/hasChatLog> <http://movie.chatbot.org#osChatLog>;
    <http://movie.chatbot.org/hasNickName> "o" .

<http://movie.chatbot.org#o> a <http://movie.chatbot.org/User>;
    <http://movie.chatbot.org/hasChatLog> <http://movie.chatbot.org#osChatLog>;
    <http://movie.chatbot.org/hasNickName> "o" .
```

##### 2.3.1.4.5.2 Movie Related Data

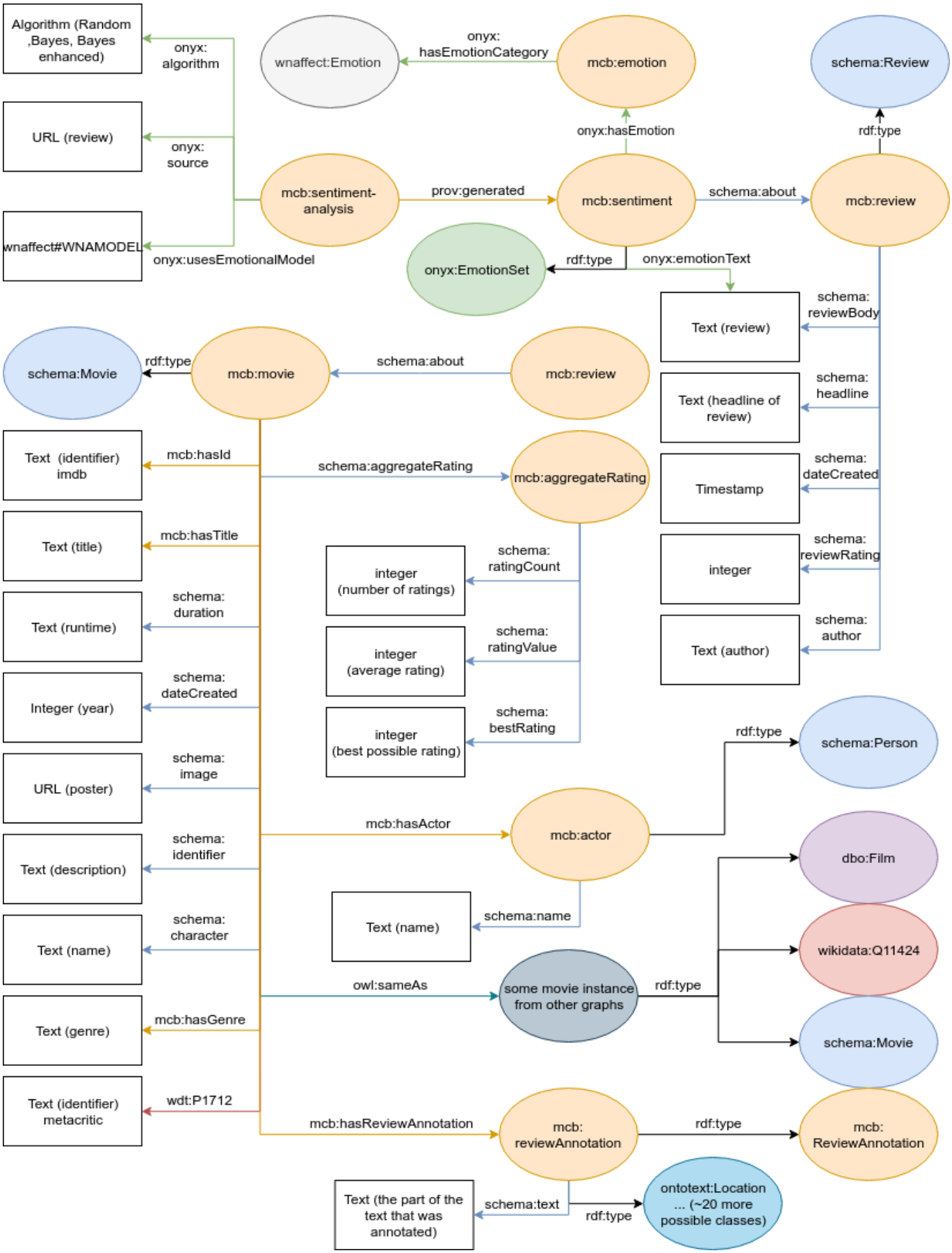
Due to document size restrictions we only have space to present one of our shapes graphs. If you want to see this one as well follow this link.

### 2.3.1.5 Linked Open Data

As previously described in the section about data sets, we maintain links to wikidata for each of our movie entries, and links to DBpedia whenever possible, using `owl:sameAs` relations.

## 2.3.2 The Final Knowledge Graph

The following figure describes all main parts of our ontology not including some minor components, as the OWL axioms. Colors declare namespaces.


Algorithm (Random ,Bayes, Bayes enhanced)
onyx: algorithm
URL (review)
onyx: source
wnaffect#WNAMODEL
onyx:usesEmotionalModel
wnaffect:Emotion
onyx: hasEmotionCategory
mcb:emotion
schema:Review
onyx:hasEmotion
rdf:type
mcb:sentiment-analysis
prov:generated
mcb:sentiment
schema:about
mcb:review
onyx:EmotionSet
rdf:type
onyx:emotionText
Text (review)
schema: reviewBody
schema:Movie
rdf:type
mcb:movie
schema:about
mcb:review
Text (headline of review)
schema: headline
Text (identifier) imdb
mcb:hasId
schema:aggregateRating
mcb:aggregateRating
Timestamp
schema: dateCreated
Text (title)
mcb:hasTitle
integer (number of ratings)
schema: ratingCount
integer
schema: reviewRating
Text (runtime)
schema: duration
integer (average rating)
schema: ratingValue
Text (author)
schema: author
Integer (year)
schema: dateCreated
integer (best possible rating)
schema: bestRating
URL (poster)
schema: image
rdf:type
schema:Person
mcb:hasActor
mcb:actor
Text (description)
schema: identifier
dbo:Film
Text (name)
schema:name
wikidata:Q11424
Text (name)
schema: character
owl:sameAs
some movie instance from other graphs
rdf:type
Text (genre)
mcb:hasGenre
schema:Movie
Text (identifier) metacritic
wdt:P1712
mcb:hasReviewAnnotation
mcb: reviewAnnotation
rdf:type
mcb: ReviewAnnotation
Text (the part of the text that was annotated)
schema:text
rdf:type
ontotext:Location ... (~20 more possible classes)

### 2.3.2.1 Knowledge Graph Statistics

In this section we focus on some statistics that seem to be the most interesting.

| | occurrences |
|---|---|
| triples | "50,197,603"^^xsd:integer |
| instances | "5,179,490"^^xsd:integer |
| distinct classes | "45"^^xsd:integer |
| distinct properties | "56"^^xsd:integer |

#### 2.3.2.1.1 Total number of instances per class per data source

| graphs | classes | instances |
|---|---|---|
| http://imdb.com | http://www.w3.org/2002/07/owlObjectProperty | "1"^^xsd:integer |
| http://metacritic.com | http://www.w3.org/2002/07/owlObjectProperty | "1"^^xsd:integer |
| http://imdb.com | http://www.w3.org/2002/07/owlDatatypeProperty | "4"^^xsd:integer |
| http://metacritic.com | http://www.w3.org/2002/07/owlDatatypeProperty | "4"^^xsd:integer |
| http://imdb.com | http://www.w3.org/2002/07/owlFunctionalProperty | "1"^^xsd:integer |
| http://metacritic.com | http://www.w3.org/2002/07/owlFunctionalProperty | "1"^^xsd:integer |
| http://imdb.com | http://www.w3.org/2002/07/owlClass | "2"^^xsd:integer |
| http://metacritic.com | http://www.w3.org/2002/07/owlClass | "2"^^xsd:integer |
| http://imdb.com | http://www.w3.org/2002/07/owlRestriction | "3"^^xsd:integer |
| http://metacritic.com | http://www.w3.org/2002/07/owlRestriction | "3"^^xsd:integer |
| http://imdb.com | schema:Person | "565063"^^xsd:integer |
| http://imdb.com | schema:Movie | "171536"^^xsd:integer |
| http://imdb.com | onyx:EmotionSet | "1768320"^^xsd:integer |
| http://metacritic.com | schema:Movie | "11681"^^xsd:integer |
| http://metacritic.com | onyx:EmotionSet | "110876"^^xsd:integer |
| http://emotions.org | onyx:EmotionSet | "938334"^^xsd:integer |
| http://emotion.org | onyx:EmotionSet | "2608120"^^xsd:integer |
| http://tag.ontotext.com | http://www.w3.org/2002/07/owlClass | "1"^^xsd:integer |
| http://tag.ontotext.com | http://www.w3.org/2002/07/owlObjectProperty | "1"^^xsd:integer |

| graphs | classes | instances |
|---|---|---|
| http://tag.ontotext.com | mcb:ReviewAnnotation | "79854"^^xsd:integer |
| http://tag.ontotext.com | ontotext:Keyphrase | "7240"^^xsd:integer |
| http://tag.ontotext.com | ontotext:RelationAcquisition | "1"^^xsd:integer |
| http://tag.ontotext.com | ontotext:RelationPersonRole | "1020"^^xsd:integer |
| http://tag.ontotext.com | ontotext:Person | "25334"^^xsd:integer |
| http://tag.ontotext.com | ontotext:RelationPersonQuotation | "60"^^xsd:integer |
| http://tag.ontotext.com | ontotext:Organization | "5749"^^xsd:integer |
| http://tag.ontotext.com | ontotext:Location | "4492"^^xsd:integer |
| http://tag.ontotext.com | ontotext:Work | "22308"^^xsd:integer |
| http://tag.ontotext.com | ontotext:Event | "1844"^^xsd:integer |
| http://tag.ontotext.com | ontotext:Thing | "7800"^^xsd:integer |
| http://tag.ontotext.com | ontotext:Species | "2941"^^xsd:integer |
| http://tag.ontotext.com | ontotext:Software | "740"^^xsd:integer |
| http://tag.ontotext.com | ontotext:CelestialBody | "133"^^xsd:integer |
| http://tag.ontotext.com | ontotext:Currency | "31"^^xsd:integer |
| http://tag.ontotext.com | ontotext:RelationOrganizationAffiliatedWithOrganization | "25"^^xsd:integer |
| http://tag.ontotext.com | ontotext:RelationOrganizationQuotation | "6"^^xsd:integer |
| http://tag.ontotext.com | ontotext:Device | "7"^^xsd:integer |
| http://tag.ontotext.com | ontotext:RelationOrganizationAbbreviation | "2"^^xsd:integer |
| http://tag.ontotext.com | ontotext:Sport | "96"^^xsd:integer |
| http://tag.ontotext.com | ontotext:RelationOrganizationLocation | "24"^^xsd:integer |
| http://tag.ontotext.com | ontotext:RelationOrganizationHasCompetitor | "1"^^xsd:integer |

# 2.4 Application

As previously discussed, a lot of preprocessing had to be made. Technically, the web scraping explanation also belongs here, however, in order to not be too repetitive we left it out here.

The frontend contains of a simple `expressjs` web server, using `socket.io` for bidirectional communication between client and server, in order to simplify the implementation of the chat application.

Other components are:

### 2.4.1 Data to RDF Mapping

A simple and lightweight tool was built, that helped us automating the process of mapping all our JSON-formatted data to RDF in turtle format. It automatically also validates the produced RDF files for validity, including error messages.

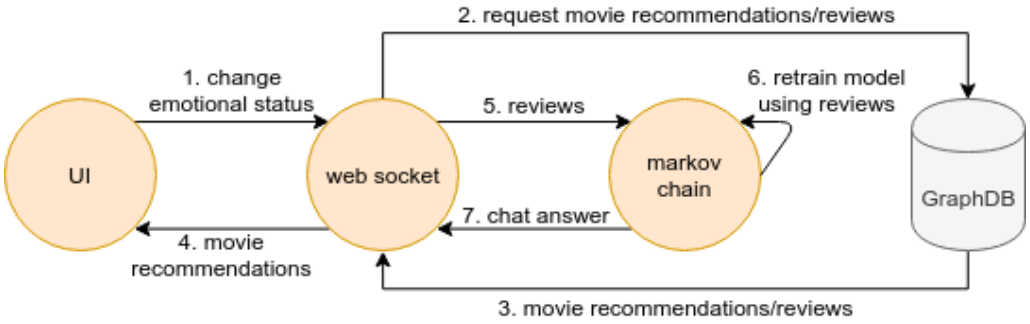
### 2.4.2 SHACL Validation

Our SHACL shapes have already been mentioned, however in order to validate our files in an automated fashion, we need some sort of library. We decided to use *SHACL-JS*, the backend implementation for shacl playground.

### 2.4.3 Markov Chains

> A Markov chain is a mathematical system that experiences transitions from one state to another according to certain probabilistic rules. The defining characteristic of a Markov chain is that no matter how the process arrived at its present state, the possible future states are fixed.
>
> -- *Wikipedia*

It is a very simple technique also used to create Natural Language. However, its lack of context-awareness is kind of a problem, which we tried to tackle by training it dynamically. So as the figure below shows, it is retrained every time a new movie is recommended to the user, using the reviews of the respective movie.




### 2.4.4 Triple Store - GraphDB

We decided to use *GraphDB* for the project. It provides a very simple way to upload our *ttl* files, and a neat interface for testing *SPARQL* queries. Another argument for *GraphDB* was its REST API which basically works with every programming language, instead of limiting us to some library and thus to a certain set of languages. This is an important point, since we wanted to use *Node.js* simply because it's really nice for handling *JSON* formatted data, and web techniques as in *HTTP* requests in general.

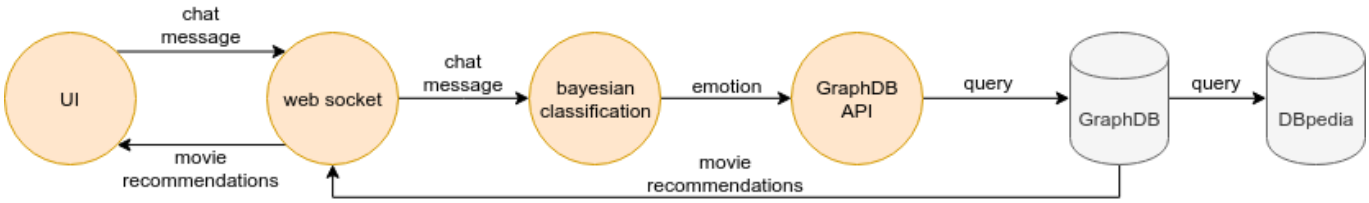
> GraphDB is a family of highly-efficient, robust and scalable RDF databases. It streamlines the load and use of linked data cloud datasets as well as your own resources. For an easy use and compatibility with the industry standards, GraphDB implements the RDF4J framework interfaces, the W3C SPARQL Protocol specification and supports all RDF serialisation formats.
>
> -- *GraphDB Web Site*

We mainly used a *SPARQL* API for requests to our server, however, problems occurred when trying to run "insert" queries. For these cases we use custom *HTTP* requests.

### 2.4.5 Bayesian Classifier

We had to analyze nearly one million reviews, to extract their assumed emotional values and accomplished that

using a bayesian classifier. The classifier is also used to extract the emotional value out of the user's chat messages as shown below:




We also used *stemming* as well as *map reduce* techniques in order to enhance our results. Using these techniques, we were able to detect a much more evenly distribution of emotion population in our reviews, which seems to be way more realistic than our first approach without these techniques.

### 2.4.6 REST Interface

In order to put our schema.org action to use, we implemented a REST API next to the frontend. One can now request `/api/v1/movies` to get a list of movies from our database (not exhaustive due to the database's large size). From there you can pick an ID and request `/api/v1/movies/` to obtain the description of a watchaction, as already described in the "Schema.org action" section.

## 2.5 Deployment

We publish our app on the web using *ngrok*, however, since we use its free version a random URL will be generated every time we restart the process, thus it is highly unlikely, that if we posted a URL here, it would still be alive at the time you test it. So please feel free to shoot a quick message and we will make sure the application/our GraphDB instance is published.

## 2.6 Conclusion

To conclude, we accomplished most of the things we wanted to do. However, improvements could be made regarding the frontend, namely usage design, as well as the natural language generation, since markov chains are not context-aware. We also don't make enough use of our linking, since we could actually display way more data than we do as of now. By retraining it on movie reviews, we could make it seem a little more like it is actually knowing what the conversation is about.

The *ontotext tagging service* was kind of a disappointment, since the computational load it can handle is extremely limited and times out way to often. However we could at least get some data out if it.

Since we're all not machine learning experts, our Naive Bayes approach for classifying natural language is probably also not optimal, but we were able to improve it a lot.

We had success acquiring lots of data about movies, which let to a huge knowledge base. Sometimes this big amount of data is also not so beneficial, especially when running more complex queries, since they require a lot of processing time. But also when validating our graph, using SHACL shapes, we observed excessive runtimes. Our database is also very well linked to *LOD* which is very nice, whenever we want to extend functionality or present additional data we could do so, by using those links.

## 2.7 Appendix

- Ontology
- Web Scraping
  - IMDb
  - metacritic