

Department of Informatics

Web Applications

Final Report

Group 20

May 22nd, 2017

Members:

- Alexander José Pereira Fernandes (MEI) - 35931, TP21;
- André Filipe Bernardes Oliveira (MI) - 45648, TP21;
- Ricardo Ferreira Vallejo (MEI) - 43338, TP21;
- Tânia Sofia Guerreiro Maldonado (MBBC) - 44745, TP22

Index

Index	2
Introduction	3
Planning	4
3. Architecture	5
4. Knowledge Base	6
4.1. Data Sources	6
4.2. Data schema (from phase 2)	7
4.3. Data Collection	7
5. Data Access Point	8
5.1. API Reference	8
5.2. Examples	8
6. Data Visualization and Interaction	11
6.1. Functionalities	11
6.2. Web Services used	11
6.3. Screenshots	12
7. Discussion	17
7.1. Work done by Alexander	17
7.2. Work done by André	17
7.3. Work done by Ricardo	17
7.4. Work done by Tânia	17
7.5. Critical analysis	17
8. Attachments	18

1. Introduction

The history of movies goes back as far as centuries ago. In the late 19th century, a single compact reel of film stored several minutes of action; today, thousands of movies can be stored in a device as small as a USB drive.

Like movies, the Internet has grown as time goes by, and as of today, it is no longer seen as an independent and disconnected information repository. What decades ago needed a full room to be stored, can now be saved, searched and viewed in a single web page, interacting with all kinds of information about that specific movie.

In this project, the aim is to explore the large set of information related with movies to build a web service that allows its users to interact with information at worldwide level, ie, searching for a particular movie will link its viewer to any information that it is attached to it - its cast, location, quotes and social media feedback.

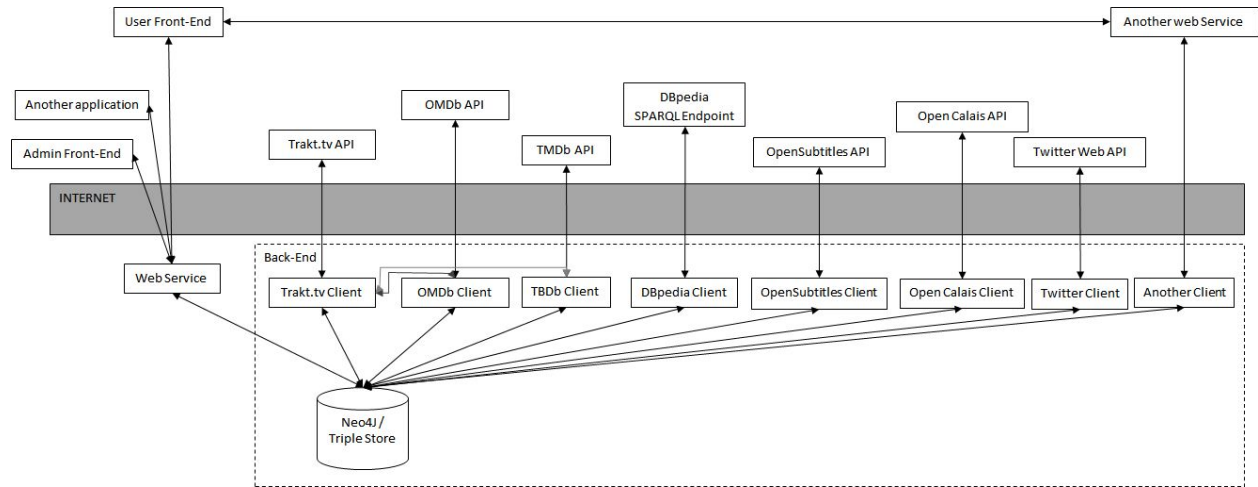
That being said, our main focus is to deliver a match between social media provided by Twitter through movie quotes and user posts, geographic locations provided by Google Maps based on movie shootings and at last, movie informations provided by imdb and track.tv.

In order to accomplish the demanded request, we will communicate with multiple diverse API's ensuring updated information and persisting this same information with the guarantee of easy and faster access for a better user experience.

2. Planning

Week	Task	Student	State	Notes	Delivery
3	Exploration of Neo4J	André	Finished		Phase 0 (6-Mar)
3	Exploration of Trakt API	Alexander	Finished		
3	Exploration of OMDB and TMDB APIs	André	Finished		
3	Exploration of Twitter API	Tânia	Finished		
3	Exploration of OpenSubtitles	Ricardo	Finished		
3	Exploration of Open Calais API	Ricardo	Finished		
4	Java and html/javascript project setup	Alexander	Finished		Phase 1 (27-Mar)
4	Database setup	André	Finished		
4	Trakt data collection service	Alexander	Finished		
4	OMDB/TMDB data collection service	André	Finished		
4	OpenSubtitles data collection service	Ricardo	Finished		
4	Initial deployment of our data API	André	Finished		
4	Cloud Natural Language API	Ricardo	Finished		
5	Twitter data collection service	Tânia	Finished		
5	front-end user feedback and crowdsourcing	Alexander	Finished		
6	Initial implementation of front-end interface	Alexander	Finished		Phase 2 (4-May)
8	Exploration of Wikipedia API	Ricardo		Easter	
9	Collect data from Wikipedia	André	Finished	After SPARQL class	
10	Our API improvements	André	Finished		
10	Identify quotes from tweets	Tânia			
11	Front-end interface improvements	Alexander			
12	Google Geolocation API	Ricardo	Finished		
13	Front end API endpoints	Ricardo	Finished		
14	Apply RDFa hints to HTML	Tânia	Finished		
15					Phase 3 (22-May)

3. Architecture



1. System Architecture Schema.

The above schema displays our system organization with most of its components (Frontend, Backend and NoSQL Database). For the backend we have chosen to use Java as the main programming language because every member of the group has prior experience.

For the network-based software architecture style we have chosen the Representational State Transfer more commonly called REST, because of its simplicity. More specifically, a RESTful web service where every resource is identified by an URI and resources are handled using POST, GET, PUT, DELETE operations (verbs) which are similar to Create, Read, Update and Delete (CRUD) operations. Our external web service communication will be made (preferably) with the MIME-type JSON (if it is not available, by XML) via the HTTP protocol and our internal web service communication is restricted to JSON. Our web services will be tested using Postman.

The chosen API for the REST support is JAX-RS (Java API for RESTful Web Services), where the use of annotations defines the REST relevance of Java classes. We will be using a library to implement this RESTful webservices in a Java servlet container called Jersey which is the reference implementation for the Java Specification Request (JSR) 339.

As web server and web container we have chosen Tomcat (who combines them and can handle HTTP requests/responses and implements Java Servlet API into one).

On the Persistence side we used the Neo4j graph database, which allowed us to query complex matches through semantic. Alternatively a Triple Store could have been used, but Neo4j is preferable because of its simplicity and because it has a graphical interface, which enables the visualization of the database's content.

For the communication between backend and frontend, we have used the Java Server Pages (JSP), beautified with HTML, Javascript and CSS. RDFa 1.1 was used in order to express structured data, using mainly terms from the Dublin Core Metadata Initiative and the Friend Of A Friend ("FOAF") namespaces.

4. Knowledge Base

4.1. Data Sources

General movie information:

- Trakt.tv API (<http://docs.trakt.apiary.io>) - main source of movie information including IMDb's id. The methods used are: GET /movies/trending (Get trending movies); GET /movies/popular (Get popular movies); GET /movies/id (Get a movie); GET /movies/id/people (Get all people for a movie)
- IMDb information - using OMDb API (<http://www.omdbapi.com/>) and/or The Movie Database API (<https://www.themoviedb.org/documentation/api>) with the id's returned from Trakt.tv request. Information is retrieved specifying the id of the movie: parameter i for OMDb API and GET /movie/{movie_id} for TMDb.
- Wikipedia information - access DBpedia SPARQL Endpoint; data is obtained through SPARQL queries
- OMDb (<http://www.omdbapi.com/>) - website that makes available additional movie information and movie images.
- TMDb (<https://www.themoviedb.org>) - website that makes available additional movie information and movie images.

Movies subtitles:

- OpenSubtitles API (<http://trac.opensubtitles.org/projects/opensubtitles/wiki/XMLRPC>) - subtitles are obtained using the search method SearchSubtitles (providing the IMDb ID) and after they are found, they can be downloaded through the DownloadSubtitles method.
- Google Cloud Natural Language API (<https://cloud.google.com/natural-language/>) - to parse and retrieve important content from the subtitles (semantic). English subtitles can be sent to Google Cloud Natural Language API with POST method available. Possible alternatives are Geographic Ontology like W3C (<https://www.w3.org/2003/01/geo/>) or GeoNames Ontology (<http://www.geonames.org/ontology/documentation.html>).
- quodb (<http://www.quodb.com/>) - website that returns a movie list that match a given string (quote). Optional, the idea was to use the subtitles file. Currently unavailable API, right now, data can only be obtained by Web scraping.

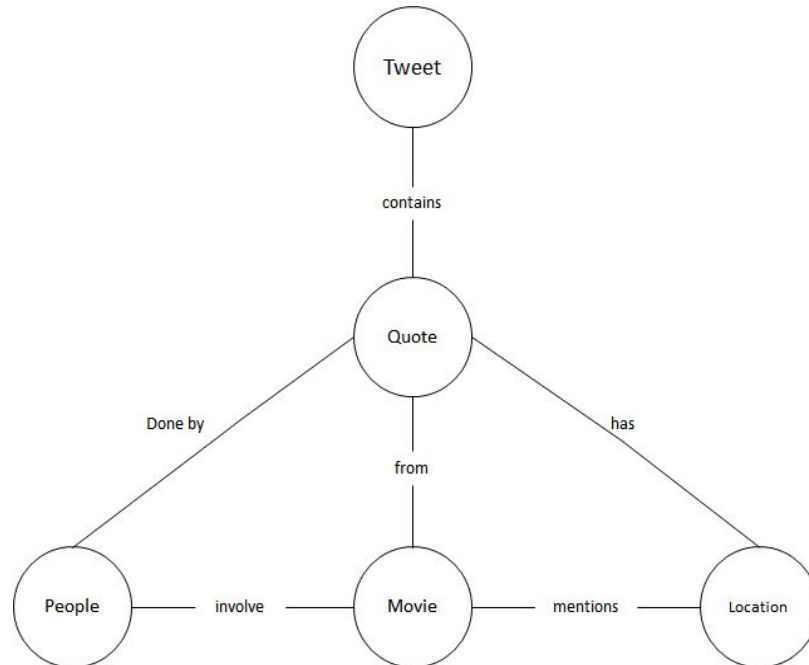
Social network:

- Twitter (<https://dev.twitter.com/overview/api>) - to movie related searches: movie title, cast members or quotes (GET search/tweets), or to tweet automatically about a given movie (POST statuses/update).

RSS Feed:

No RSS feed was used because no adequate use (that fitted our view of the project) was found to justify using it. The most appropriate findings were Metacritic reviews (<http://www.metacritic.com/rss>) or future premiers (“Coming Soon”), but they were not relevant enough to be included in our system.

4.2. Data schema (from phase 2)



2. Data Schema with most relevant entities, and their associations.

4.3. Data Collection

For the general movie information the various Rest APIs were used for data collection. Minimal information (e.g. references and key information) is stored on the database from these, the remaining is fetched as needed, for the users.

OpenSubtitles API is used to fetch subtitles, which is later supplied to Cloud Natural Language API (from the Google Cloud Platform) to parse and return key entities such as location and people mentioned in the movie, which are stored on the database. Google’s Geocoding API is then used to retrieve the coordinates of the entities found.

Tweets related to the according movie are imported and displayed to the user.

We developed most of the clients to consume the external API’s, since no good alternative was found.

5. Data Access Point

5.1. API Reference

The following resources and collections are provided:

Resource	Method	Returns
/rest/movie/	GET	all movies
/rest/movie/{m_uri}	GET	information about a specific movie
/rest/movie/{m_uri}	PUT	insert/edit movie information
/rest/movie/{m_uri}	DELETE	delete movie information
/rest/movie/{m_uri}/locations	GET	locations mentioned on the movie
/rest/movie/{m_uri}/locations	POST	insert locations mentioned on the movie
/rest/movie/{m_uri}/locations/{l_uri}	GET	information about and specific location
/rest/movie/{m_uri}/locations/{l_uri}	PUT	edit information about an specific location
/rest/movie/{m_uri}/locations/{l_uri}	DELETE	delete information about an specific location
/rest/movie/{m_uri}/people	GET	people mentioned on the movie
/rest/movie/{m_uri}/people/{p_uri}	PUT	insert new people mentions on the movie
/rest/movie/{m_uri}/people/{p_uri}	DELETE	delete information about an specific person mentioned on the movie
/rest/movie/{m_uri}/tweet	GET	tweets related to a specific movie
/rest/movie/{m_uri}/tweet	POST	add tweets related to the specific movie
/rest/movie/{m_uri}/tweet	DELETE	remove tweet from movie relation
/rest/movie/person	GET	get all people
/rest/movie/person/{p_uri}	GET	information about a specific person
/rest/movie/person/{p_uri}	PUT	update about a specific person
/rest/movie/person/{p_uri}	DELETE	delete information about an specific person

All requests use REST architecture, and when the output is required, it follows JSON format.

5.2. Examples

1. Get movie list

```
[
  {
    "title":"The Dark Knight",
    "uri":"TheDarkKnight"
  },
  {
    "title":"Inception",
    "uri":"Inception"
  },
  {
```



```

    "title":"Guardians of the Galaxy",
    "uri":"GuardiansOfTheGalaxy"
  },
  {
    "title":"The Avengers",
    "uri":"TheAvengers"
  },
  {
    "title":"Suicide Squad",
    "uri":"SuicideSquad"
  },
  {
    "title":"The Matrix",
    "uri":"TheMatrix"
  },
  {
    "title":"Interstellar",
    "uri":"Interstellar"
  },
  {
    "title":"Doctor Strange",
    "uri":"DoctorStrange"
  },
  {
    "title":"Star Wars: The Force Awakens",
    "uri":"StarWarsTheForceAwakens"
  },
  {
    "title":"Deadpool",
    "uri":"Deadpool"
  }
]

```

2. Get movie details

```

{
  "overview":"Based upon Marvel Comicsâ€™ most unconventional anti-hero, DEADPOOL tells the origin story of former Special Forces operative turned mercenary Wade Wilson, who after being subjected to a rogue experiment that leaves him with accelerated healing powers, adopts the alter ego Deadpool. Armed with his new abilities and a dark, twisted sense of humor, Deadpool hunts down the man who nearly destroyed his life.",
  "runtime":108.0,
  "title":"Deadpool",
  "uri":"Deadpool",
  "certification":"R",

```

```

"trailer":"http://youtube.com/watch?v\u003dONHBaC-pfsk",
"genres":[
  "action",
  "adventure",
  "comedy",
  "romance"
],
"id_trakt":"190430",
"id_imdb":"tt1431045",
>tagline":"Witness the beginning of a happy ending",
"imdb_rating":8.20508,
"url_trakt":"https://trakt.tv/movies/deadpool-2016",
"released":"2016-02-12",
"homepage":"http://www.foxmovies.com/movies/deadpool",
"id_tmdb":"293660"
}

```

3. Get movie locations

```

{
  "movie": "movie_id",
  "locations":
  [
    {
      "id": "location_id_1",
      "latitude": 36,
      "longitude": -6,
      "location": "somewhere on portugal",
      "quote": ["quote_id_1", "quote_id_2",...]
    },
    ...
  ]
}

```

4. Get movie cast members

```

{
  "movie": "movie_id",
  "people":
  {
    {
      "id": "people_id_1",
      "name": "The character/person name",

```

```

        "external_ids":
        {
            "wiki": "wiki_id",
            ...
        }
    },
    ...
}

```

5. Get movie quotes

```

{
  "movie": "movie_id",
  "quotes":
  [
    {
      "id": "quote_id",
      "text": "the quote from the movie",
      "timing": "1:39:45",
      "location": ["location_id_1", "location_id_2", ...]
    },
    ...
  ]
}

```

6. Data Visualization and Interaction

6.1. Functionalities

Our website allows users to see a correlation between movies and places that are connected, in some way, with those movies (e.g. mentioned in the movie/subtitle, places the movie was shot on, etc). This allows users to explore movies in a different way (navigating through places) and find movies that they might like, because a similar set of places is present/is mentioned.

Another key aspect about movie recommendations are social networks. In our website, we show tweets that are directly related to the movie in question, so that guests can see the public opinion on that specific movie.

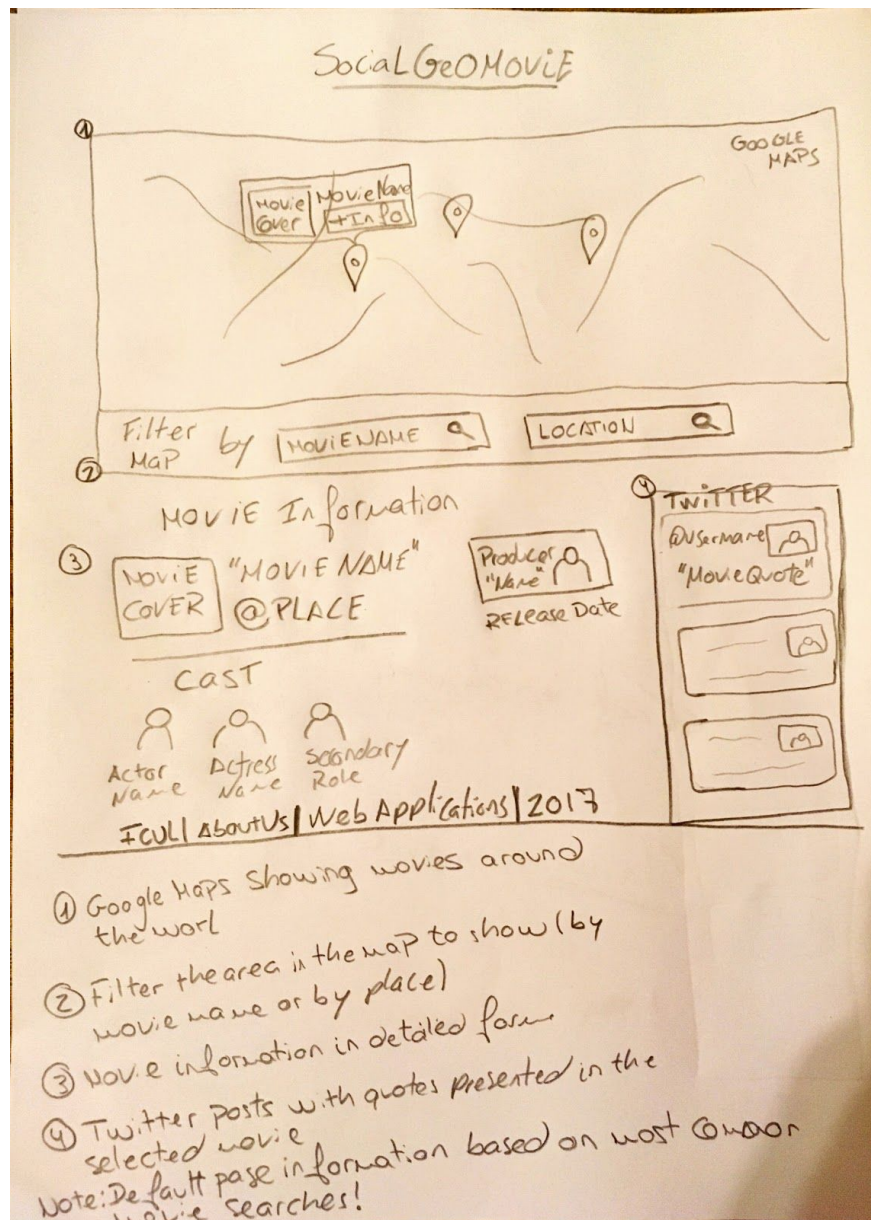
6.2. Web Services used

We used Google's "Cloud Natural Language API" (<https://cloud.google.com/natural-language>) to extract location entities from each movie subtitle.

Data Visualization requires additional web services. We used Maps Embed API

(<https://developers.google.com/maps/documentation/embed/guide>) to display a map with the referenced locations for each movie in our database. This also requires the use of Geocoding API (<https://developers.google.com/maps/documentation/geocoding/start>) to convert known locations to geographic coordinates.

6.3. Screenshots



3. Interface Sketch

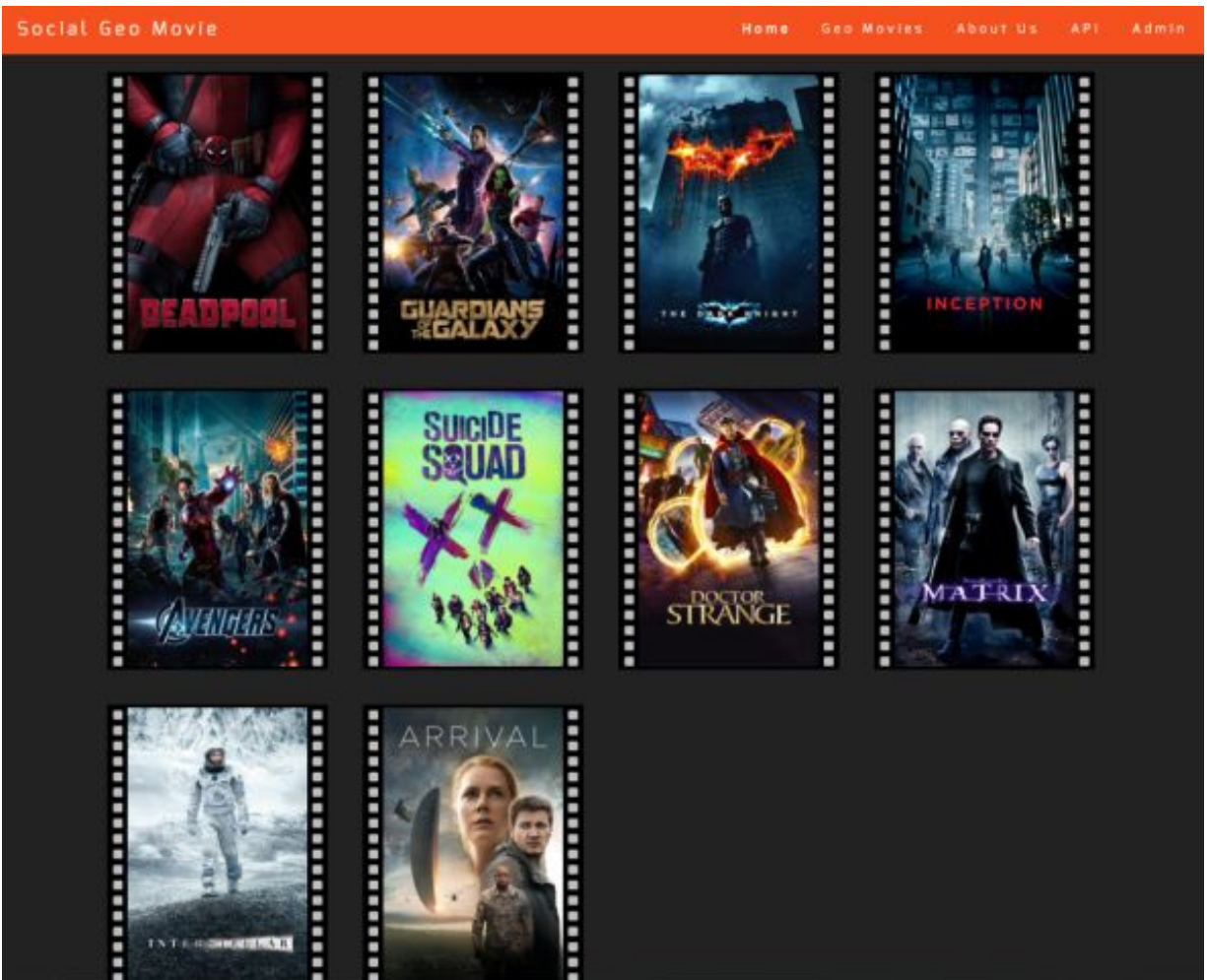
localhost:8080/aw2017/rest/movie/

```
[{"title":"The Dark Knight","uri":"TheDarkKnight"}, {"title":"Inception","uri":"Inception"}, {"title":"Guardians of the Galaxy","uri":"GuardiansOfTheGalaxy"}, {"title":"The Avengers","uri":"TheAvengers"}, {"title":"Suicide Squad","uri":"SuicideSquad"}, {"title":"The Matrix","uri":"TheMatrix"}, {"title":"Interstellar","uri":"Interstellar"}, {"title":"Doctor Strange","uri":"DoctorStrange"}, {"title":"Star Wars: The Force Awakens","uri":"StarWarsTheForceAwakens"}, {"title":"Deadpool","uri":"Deadpool"}]
```

4. Json Response Example (as the first example)


Tabela com todos os filmes		Tabela com 20 pessoas		Tabela com cast do filme id: Deadpool	
Title	ID	Name	ID	Name	ID
The Dark Knight	TheDarkKnight	Ryan Reynolds	RyanReynolds	Dan Zachary	DanZachary
Inception	Inception	Morena Baccarin	MorenaBaccarin	Olesia Shewchuk	OlesiaShewchuk
Guardians of the Galaxy	GuardiansOfTheGalaxy	Ed Skrein	EdSkrein	Anthony J. Sacco	AnthonyJSacco
The Avengers	TheAvengers	T.J. Miller	TJMiller	Sean Quan	SeanQuan
Suicide Squad	SuicideSquad	Gina Carano	GinaCarano	Michael Neumeyer	MichaelNeumeyer
The Matrix	TheMatrix	Leslie Uggams	LeslieUggams	David Longworth	DavidLongworth
Interstellar	Interstellar	Brianna Hildebrand	BriannaHildebrand	Greg LaSalle	GregLasalle
Doctor Strange	DoctorStrange	Karan Soni	KaranSoni	Tony Chris Kazoleas	TonyChrisKazoleas
Star Wars: The Force Awakens	StarWarsTheForceAwakens	Jed Rees	JedRees	Matthew Hoglie	MatthewHoglie
Deadpool	Deadpool	Stefan Kapičić	StefanKapicic	David Hardware	DavidHardware
		Randal Reeder	RandalReeder	Victoria De Mare	VictoriaDeMare
		Isaac C. Singleton Jr.	IsaacCSingletonJr	Fabiola Colmenero	FabiolaColmenero
		Stan Lee	StanLee	Heather Ashley Chase	HeatherAshleyChase
		Michael Benyaer	MichaelBenyaer	Rob Hayter	RobHayter
		Style Dayne	StyleDayne	Paul Lazenby	PaulLazenby
		Kyle Cassie	KyleCassie	Rachel Sheen	RachelSheen
		Taylor Hickson	TaylorHickson	Benjamin Wilkinson	BenjaminWilkinson
		Ayzee	Ayzee	Jason Day	JasonDay

5. Draft HTML



6. Initial page with movie list





Social Geo Movie
Home
Geo Movies
About Us
API
Admin



Deadpool

Witness the beginning of a happy ending

Release: 12-02-2016

R


8.21

108

Certification
Rating
Runtime


Based upon Marvel Comics' most unconventional anti-hero, DEADPOOL tells the origin story of former Special Forces operative turned mercenary Wade Wilson, who after being subjected to a rogue experiment that leaves him with accelerated healing powers, adopts the alter ego Deadpool. Armed with his new abilities and a dark, twisted sense of humor, Deadpool hunts down the man who nearly destroyed his life.

Deadpool | Trailer [HD] | 20th Century FOX



TRAILER

Cast



Tweets

SimplyPogba1 says:

Best = Deadpool or guardians of the galaxy
Worst = green lantern <https://t.co/5w6b3u6k>

retweets: 0 22-05-2017 22:11:12

Seankitskit says:

Roses real red violats are cat? Superman
Wark Jurassic Of . Dayl . Ryan Black Men Zone
Harry The Caribbean Titanic Remnant Wolf
Deadpool

retweets: 0 22-05-2017 22:11:22

GoGoPromo says:

#Deadpool #Marketing #Humor in
#SocialMedia and How To Step Up Your
Marketing Message <https://t.co/c3HG74E15W>

retweets: 0 22-05-2017 22:11:37

CheckKtheCircuit says:

7. Movie page with corresponding information, trailer, cast, ratings, relevant links, tweets and user comments.



8. Map page for showing movie geographic information and statistics

7. Discussion

7.1. Work done by Alexander

Implementation of the frontend, the user's real-time movie comments system, and overhaul of the tweet import and service process.

7.2. Work done by André

Finishing collect data from OMDb, and start to collect data from TMDb. Performed improvements in our web services (performance and handling exceptions). Processed data to make inference on the data collected (create relationships with locations and dates). Improvements in methods to save data (tweets and subtitles). Implemented a way to query the DBpedia SPARQL endpoint in order to retrieve more information about movies and cast (sending SPARQL query to Virtuoso endpoint through REST and receiving the result as JSON). Developed the method necessary to provide autocomplete while searching.

7.3. Work done by Ricardo

Provided more information and filtered Subtitles data with Google Cloud Natural Language API, usage of Google GeoLocation service to inform Latitude and Longitude based on Locales and given subtitle match content. Explored SPARQL and DBpedia. Added front end API endpoints.

7.4. Work done by Tânia

Implemented RDFa 1.1 on HTML pages and improved the page front-end.

7.5. Critical analysis

We managed to implement a great deal of what we had setup to achieve but, sadly, time was short to really polish the frontend and backend to the level we had hoped to. Despite that, we consider that this application is a proof of concept that could be improved to become a full product. We think that we implemented the main functionalities (NoSQL database, backend with several sources, making use of linked data with SPARQL and RDFa and a rich visualization with d3) required and that the applications works as expected.

A video overview on the webpage was uploaded on YouTube for future reference, and can be viewed at <http://www.youtube.com/watch?v=bndZ9A244j0>.

8. Attachments

Examples from the API's be used:

1. Find Movie Quotes within Twitter Posts:

<https://api.twitter.com/1.1/search/tweets.json?q=portugal>

```
{
  "created_at": "Sat Mar 04 11:50:39 +0000 2017",
  "id": 837993690182402000,
  "id_str": "837993690182402049",
  "text": "RT @SouBenfica1904: 4 de Março de 1992... A data em que nascia uma lenda, sem igual em Portugal. Muitos parabéns, rapazes. 25 anos! "Sempre...",
  "truncated": false,
  "entities": {
    "hashtags": [],
    "symbols": [],
    "user_mentions": [
      {
        "screen_name": "SouBenfica1904",
        "name": "Sou Benfica",
        "id": 2842529048,
        "id_str": "2842529048",
        "indices": [
          3,
          18
        ]
      }
    ],
    "urls": []
  },
  "metadata": {
    "iso_language_code": "pt",
    "result_type": "recent"
  }
}
```

2. Query movie information on Trakt.tv API:

<https://api.trakt.tv/movies/tron-legacy-2010>

```

{
  "title": "TRON: Legacy",
  "year": 2010,
  "ids": {
    "trakt": 343,
    "slug": "tron-legacy-2010",
    "imdb": "tt1104001",
    "tmdb": 20526
  },
  "tagline": "The Game Has Changed.",
  "overview": "Sam Flynn, the tech-savvy and daring son of Kevin Flynn, investigates his father's disappearance and is pulled into The Grid. With the help of a mysterious program named Quorra, Sam quests to stop evil dictator Clu from crossing into the real world.",
  "released": "2010-12-16",
  "runtime": 125,
  "updated_at": "2014-07-23T03:21:46.000Z",
  "trailer": null,
  "homepage": "http://disney.go.com/tron/",
  "rating": 8,
  "votes": 111,
  "language": "en",
  "available_translations": [
    "en"
  ],
  "genres": [
    "action"
  ],
  "certification": "PG-13"
}

```

Project Link (Java):

<https://drive.google.com/open?id=0B66hMAdYXIVEQjE0MWZpX2p4VTA>