Department of Informatics

Web Applications

Report Phase 1

Group 20

27 March, 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

1. Introduction

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

Like movies, the Internet as 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 storaged, 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.

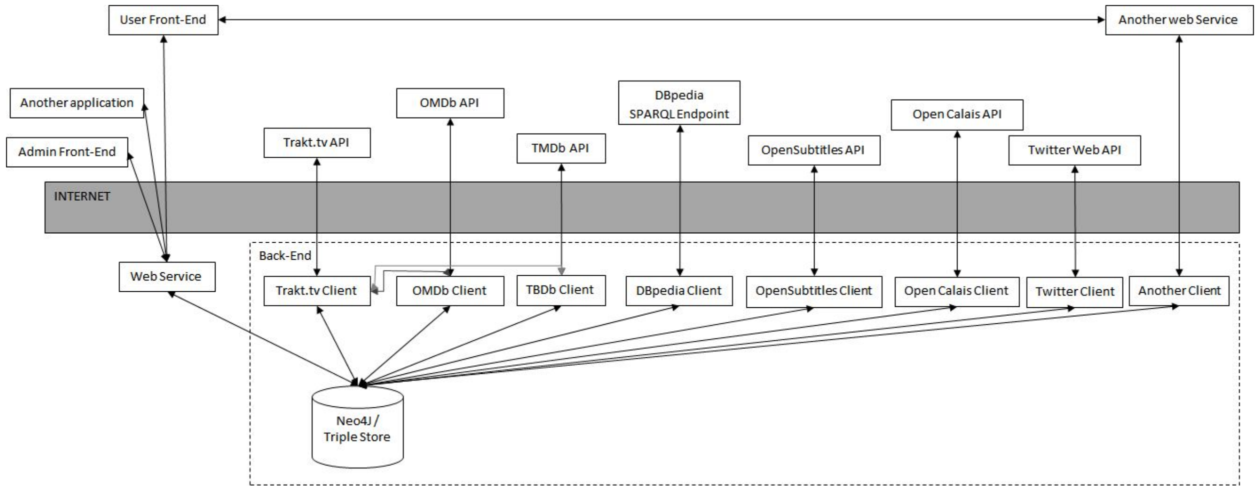
2. Planning

Week	Task	Student	State	Notes
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	In Progress	
3	Exploration of Twitter API	Tânia	Finished	
3	Exploration of OpenSubtitles	Ricardo	Finished	
3				
3	Exploration of Open Calais API	Ricardo	Finished	
4	Java and html/javascript project setup	Alexander		
4	Database setup	André	Finished	
4	Trakt data collection service	Alexander	Finished	
4	OMDB/TMDB data collection service			
4	OpenSubstitles data collection service	Ricardo	Finished	
4	Initial deployment of our data API		Some services	
4	Initial implementation of front-end interface			
5	Twitter data collection service	Tânia	Finished	Phase 1 (27-Mar)
5	front-end user feedback and crowdsourcing			
6	Identify quotes from tweets			
6	Data collection improvements			
7	front-end interface improvements			
8	Exploration of Wikipedia API			Easter
9	Collect data from			After

	Wikipedia			SPARQL class Phase 2 (24-Apr)
10				
11				
12				Phase 3 (15-May)
13				
14				

Tasks distribution will be performed after project discussion with the teachers. In the table above we can get an overview of the schedule of all the tasks to be performed.

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 will be using Neo4j graph database, which allow us to query complex matches through semantic. Alternatively a Triple Store can be 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 will use the Java Server Pages (JSP), beautified with HTML, Javascript and CSS.

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 will be 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

Movies subtitles:

- OpenSubtitles API (http://trac.opensubtitles.org/projects/opensubtitles/wiki/XMLRPC) - subtitles will be obtained using the search method SearchSubtitles (provinding the IMDb ID) and after they are found, they can be downloaded through the DownloadSubtitles method.
- OPEN CALAIS API (http://www.opencalais.com/opencalais-api/) - to parse and retrieve important content from the subtitles (semantic). English subtitles can be sent to OPEN CALAIS 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.

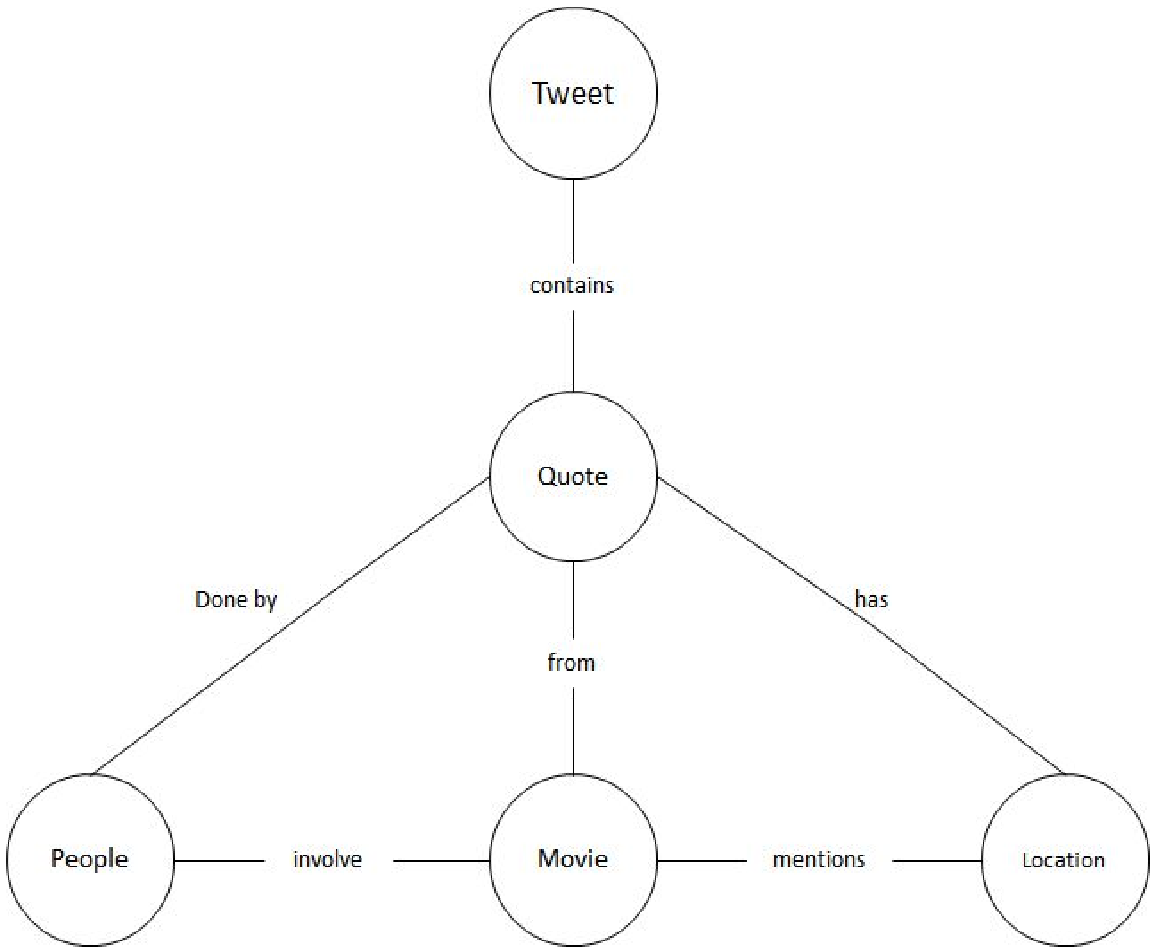
Social network:

- Twitter (https://dev.twitter.com/overview/api) - to search for movie 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. If an adequate RSS feed is found that contains relevant information (critics, news, etc) about the movies, this section will be updated in future deliveries.

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 will be used for data collection, minimal information (e.g. references and key information) will be stored on the database from these, the remaining will be fetched as needed, for the users.

OpenSubtitles API will be used to fetch subtitles which will be later supplied to Open Calais API to parse and return key entities such as location and people mentioned in the movie, which will be stored on the database.

QuoDB web site can be used to fetch movie's quotes which will be stored and later compared from tweets (from twitter API) to identify movie popularity statistics.

We will develop 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/{m_id}	GET	movie information
/rest/movie/{m_id}	PUT	insert/edit movie information
/rest/movie/{m_id}	DELETE	delete movie information
/rest/movie/{m_id}/locations	GET	locations mentioned on the movie
/rest/movie/{m_id}/locations	POST	insert locations mentioned on the movie
/rest/movie/{m_id}/locations/{l_id}	GET	information about and specific location
/rest/movie/{m_id}/locations/{l_id}	PUT	edit information about an specific location
/rest/movie/{m_id}/locations/{l_id}	DELETE	delete information about an specific location
/rest/movie/{m_id}/people	GET	People mentioned on the movie
/rest/movie/{m_id}/people	POST	insert new people mentions on the movie
/rest/movie/{m_id}/people/{p_id}	GET	information about specific person mentioned on the movie
/rest/movie/{m_id}/people/{p_id}	PUT	edit information about an specific person mentioned on the movie
/rest/movie/{m_id}/people/{p_id}	DELETE	delete information about an specific person mentioned on the movie
/rest/movie/{m_id}/quote	GET	quotes from the movie
/rest/movie/{m_id}/quote	POST	insert new information about a quote from the movie
/rest/movie/{m_id}/quote/{q_id}	GET	information about an specific quote
/rest/movie/{m_id}/quote/{q_id}	PUT	edit information about an specific quote from the movie
/rest/movie/{m_id}/quote/{q_id}	DELETE	delete information about an specific quote from the movie
/rest/movie/{m_id}/quote/{q_id}/tweet	GET	tweets related to thes specific quote
/rest/movie/{m_id}/quote/{q_id}/tweet	POST	add tweets related to the specific quote
/rest/movie/{m_id}/quote/{q_id}/tweet	DELETE	remove tweet from quote relation
/rest/movie/{m_id}/tweet	POST	write tweet about a given movie

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

5.2. Examples

1. Get movie list

```
{
  "id":190430,
    "title":"Deadpool"
  },
  {
    "id":120,
    "title":"The Dark Knight"
  },
  {
    "id":16662,
    "title":"Inception"
  },
  {
    "id":82405,
    "title":"Guardians of the Galaxy"
  },
  {
    "id":14701,
    "title":"The Avengers"
  },
  {
    "id":481,
    "title":"The Matrix"
  },
  {
    "id":193079,
    "title":"Suicide Squad"
  },
  {
    "id":102156,
    "title":"Interstellar"
  },
  {
    "id":94024,
    "title":"Star Wars: The Force Awakens"
  },
  {
    "id":77349,
    "title":"Frozen"
  }
]
```

2. Get movie details

```
{
   "title": "TRON: Legacy",
   "ids":
```

```
   {
          "our_id": "movie_id"
          "trakt": 1,
          "slug": "tron-legacy-2010",
          "imdb": "tt1104001",
          "tmdb": 20526,
          "wiki": "Tron:_Legacy"
   }
}
```

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 people

```
{
   "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

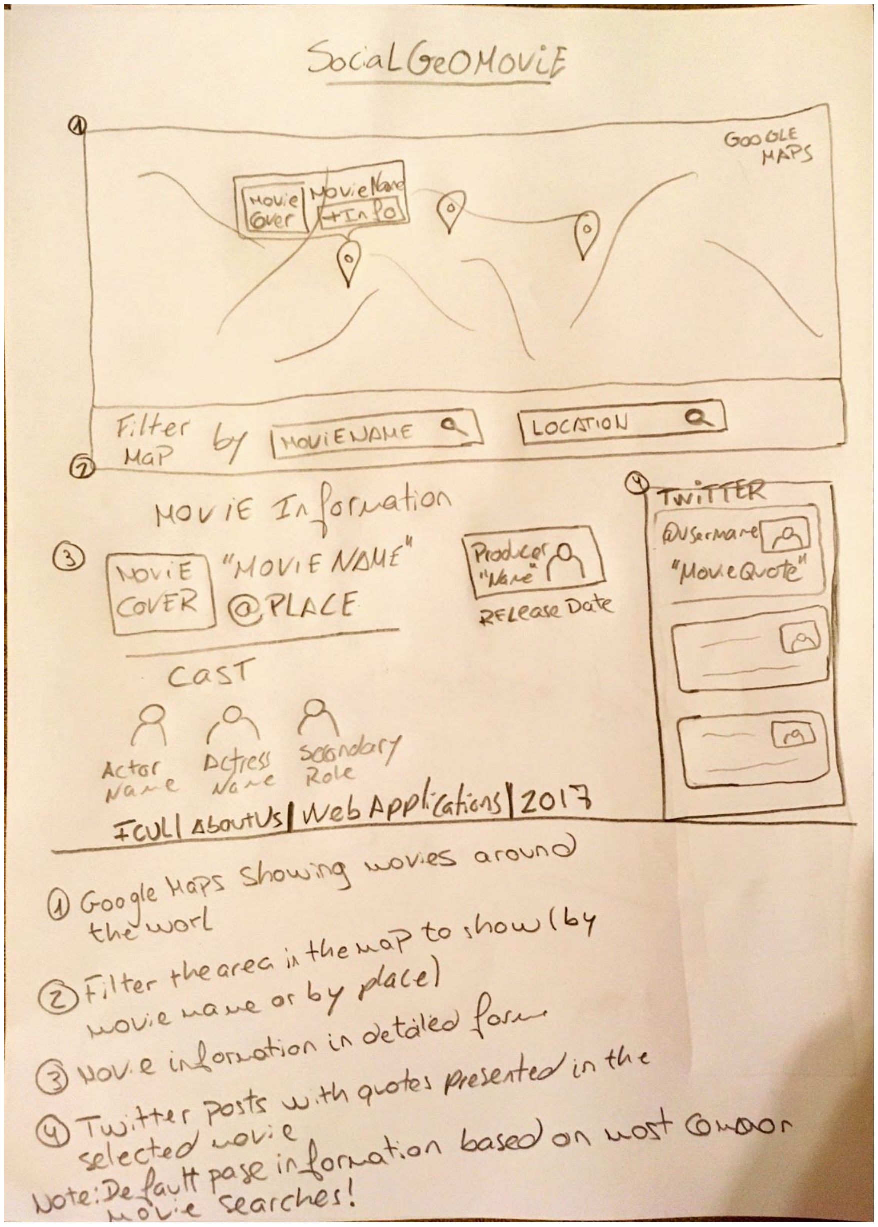
Our website will allow users to see a correlation between movies and places that are connected, in some way, with those movies (e.g. mentioned in the movie/subtitle, places the movie was shot on, etc). This will allow users to explore movies in a different way (navigating through places) and find movies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

Another key aspect about movie recommendations are social networks, therefore this system will allow users to identify the most talked movies in social networks. We will exhibit the most quoted movies on twitter as it can be viewed as a metric of popularity.

6.2. Web Services used

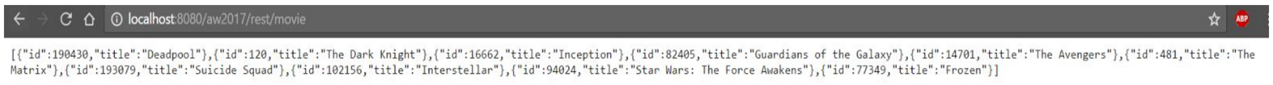
Data Visualization will require additional web services. We will use Maps Embed API (https://developers.google.com/maps/documentation/embed/guide) to display a map with the referenced locations for each movie in our database. This will require the use of Geocoding API (https://developers.google.com/maps/documentation/geocoding/start) to convert known locations to geographic coordinates.

6.3. Screenshots



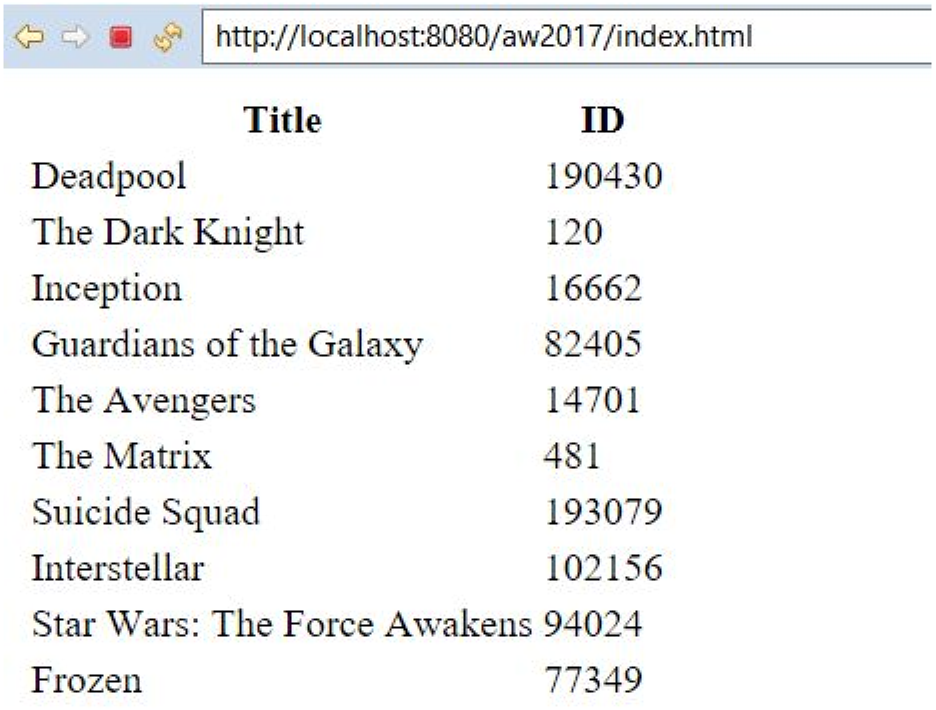
1 Google Maps showing movies around the worl
2 Filter the area in the map to show (by movie name or by place)
3 Movie information in detailed form
4 Twitter posts with quotes presented in the selected movie
Note: Default page information based on most common movie searches!

3. Interface Sketch



[{"id":190430,"title":"Deadpool"},{"id":120,"title":"The Dark Knight"},{"id":16662,"title":"Inception"},{"id":82405,"title":"Guardians of the Galaxy"},{"id":14701,"title":"The Avengers"},{"id":481,"title":"The Matrix"},{"id":193079,"title":"Suicide Squad"},{"id":102156,"title":"Interstellar"},{"id":94024,"title":"Star Wars: The Force Awakens"},{"id":77349,"title":"Frozen"}]

4. Json Response Example (as the first example)



Title	ID
Deadpool	190430
The Dark Knight	120
Inception	16662
Guardians of the Galaxy	82405
The Avengers	14701
The Matrix	481
Suicide Squad	193079
Interstellar	102156
Star Wars: The Force Awakens	94024
Frozen	77349

5. Result HTML

7. Discussion

7.1. Work done by Alexander

Exploration and development of the Trakt REST client.

IDE, SVN, Server and Project Set-up

The data import service, making use the API clients developed by the other team members.

7.2. Work done by André

Development of REST client to access Neo4J DB, using Jersey RESTful Web Services framework version 1.x and following the official guide from Neo4J. Saving movies and cast data to Neo4J. Development of GET movie method to get a movie list.

7.3. Work done by Ricardo

Consumed service to obtain OpenSubtitles information based on Imdb ID's and presented subtitle information and srt as plain text. Explored OpenCalais service.

7.4. Work done by Tânia

Exploring the existing Twitter API's and clients, and integrating the chosen client with the project. The chosen client was Hosebird Client (abbreviated "hbc"), a Java HTTP client for consuming Twitter's Streaming API, since it is developed by Twitter itself and it is the most used client in Java language.

7.5. Critical analysis

It was not possible to consume all web services and develop all our webservices. A draft implementation is provided at this stage.

8. Attachments

Examples from the API's that will 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"
      }
```

1. 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"
}
```