**Faculty of Computer Science,**

**“Alexandru Ioan Cuza” University of Iași**

Academic year 2015-2016

MUXar

Using web semantics to provide music suggestions

Technical report

Project coordinator : **Dr. Sabin-Corneliu Buraga**

Project members :

* **Alexandru Iulian Covașă** - 2nd year Student, Master of Distributed Software Systems
* **Corina Gabriela Covașă** - 2nd year Student, Master of Distributed Software Systems

### Abstract

With the increase of web semantic use around the Internet, more and more websites use linked data to provide a better user experience, or use semantic markup to make them more accessible to search engines.

This technical report presents an approach to building a website that uses the advantages of web semantics to present users with music suggestions based on their preferences. The application will gather preliminary data from existing music services (e.g. Google Play Music) and will display connections between this data and other musical entities using knowledge offered by linked data repositories (e.g. DBpedia).

**Keywords :** web semantics, sparql, rdf, music

# **Introduction**

The objective of this project is to create a system which will function as a web application that users will access from their PCs, laptops or mobile devices. This application will be able to run on any server or PaaS (e.g. *Amazon EC2*).

The end-user will have access to information and music suggestions based on their use of other musical services (e.g. *Google Play Music*). The information presented will be enhanced by the MUXar application using linked data and semantic web technologies from external APIs.

In order to achieve the objective of building the **Musical UX Smart Enhancer** (*MUXar*) application, some software development considerations must be taken into account.

The development of this application aims to serve as an example that such applications that make use of web semantics, from an academical research standpoint, can be built. At most, this application will serve as prototype or reference for commercial applications. As such, this report will not cover a State-of-the-Art analysis.

However, the following sections of this document will discuss user requirements, the architecture of the application, the technologies and external services used.

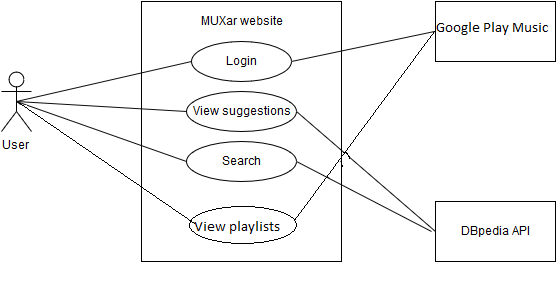
# 2. **User requirements**

This section will address what operations we will expect the user to perform using the MUXar application.

At least the application must be able to provide the following functionality :

* Allow the user to log into the website with an existing account from the existing music service - G*oogle Play Music*.
* Allow the user to logout from the website and be redirected to the login page.
* Allow the user to view a playlist with songs that were marked as favorites.
* Allow the user to view his favorites songs, artists and albums.
* Allow the user to search for music.
* Allow the user to see some recommendations: for a specified artist there will be recommended a list of bands whom he collaborates with; for 2 artists there will be recommended some songs they have in common; for a given artist pick a genre and recommend some songs that belong to that genre; for a given artist recommend other songs that belong to that artist. This recommendations will be seen as a list of songs.

These requirements are reflected in the following Use-Case diagram with only *Google Play Music* and *DBPedia* as external sources :



*Fig. 2.1. Use-Case Diagram*

The diagram shown in figure 2.1 describes which use case interacts with which entity. As such, the *login* function interact with the APIs that provide **user** information via simple *HTTP* requests. The *search* and *suggestions* functions provide **music** information via semantic web technologies (e.g.*SPARQL*).

# 3. **Technologies**

In order to accommodate the user requirements described in the previous sections, the technologies used must be chosen accordingly. From a software development standpoint, considerations regarding the time required to develop or other such criteria might be taken into account. However, for the purpose of this project, we will only focus on technologies that have been proven as suitable for web application development and that support the protocols required.

The **front-end** of this application will consist of an *Angular.js* controller that will handle user interactions and the communication with the MUXar web service. On this controller the user interface will be built with *Angular Material.*

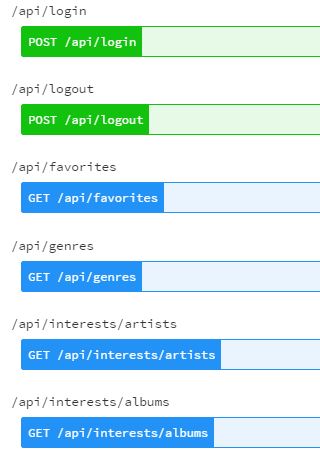
*The UI is based on* ***HTML5*** *and* ***AngularJS****. The code is split into three main components:*

* The **view**: contains all the HTML that is used to build the UI.
* **Controller**: the middleware that connects the UI with our business logic. Used to manipulate the state and behavior of the UI.
* **Services**: Angular services are substitutable objects that are wired together using dependency injection (DI). You can use services to organize and share code across your app. Angular services are:
  + Lazily instantiated – Angular only instantiates a service when an application component depends on it.
  + Singletons – Each component dependent on a service gets a reference to the single instance generated by the service factory.

Each component has its own service.

On the **back-end**, the main controller of the application will be built using *Node.js* which has native server support for HTTP/HTTPS connections. Modules for *Node.js* that facilitate the development of this application will be used, some of the more important is *Express.js* middleware for routing. Other modules will be used as needed for *SPARQL.*

Below you may see a picture of what our Rest API provides. Our API provides 2 types of data: the one which is simply get from Google API (what songs the user listens to and the playlists) and the one that is obtained in context of semantic Web (in order to obtain this data we are using the data provided by Play.js API) – this data will be described later in chapter 5.



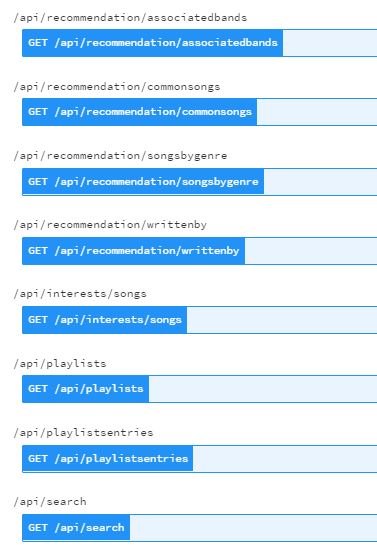
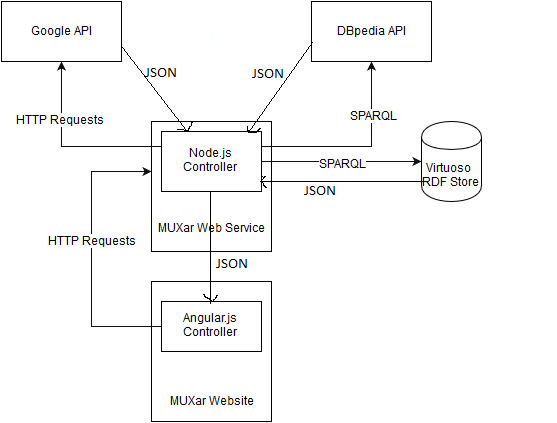


Fig. 2.2 REST API services

# 4. **Architecture**

Using the technologies described in section 3, the following components diagram describes the application architecture :

*Fig. 4.1. Components diagram*

# 5. **External services**

For this project to satisfy the requirements specified in section 2 of this document, the application will have to rely on a number of external services in order to work:

* **DBpedia** – from this external service we used the linked data containing information about music.

We interrogated this service for obtaining information about music genres, artists and songs. Below you may see some queries we used:

Query used to insert a username into a graph we create:

insert data into <http://wadeproject.com/> {

<http://wadeproject.com/users/user1> foaf:name "user1".

}

This graph will contain information and relations created between user which is interested in artists, songs and albums.

Query used to interrogate our graph in order to obtain the artists our user is interested in:

prefix dbo: <http://dbpedia.org/ontology/>

select ?artist from <http://wadeproject.com/> where {

<http://wadeproject.com/users/user1> foaf:interest ?artist.

?artist rdf:type dbo:MusicalArtist.

}

Query used to interrogate our graph in order to obtain all albums our user is interested in:

prefix dbo: <http://dbpedia.org/ontology/>

select ?album from <http://wadeproject.com/> where {

<http://wadeproject.com/users/user1> foaf:interest ?album.

?album rdf:type dbo:MusicalWork;

rdf:type dbo:Album.

}

Query used to interrogate our graph in order to obtain all songs our user is interested in:

prefix dbo: <http://dbpedia.org/ontology/>

select ?song from <http://wadeproject.com/> where {

<http://wadeproject.com/users/user1> foaf:interest ?song.

?song rdf:type dbo:MusicalWork;

rdf:type dbo:Single.

}

Query used to make a recommendation: if user listens to 2 artists then we can check if the 2 artists have songs in common and recommend songs to user that are sang by the 2 artists:

select ?song where {

?song dbo:musicalArtist <http://dbpedia.org/resource/Robbie\_Williams>;

dbo:musicalArtist <http://dbpedia.org/resource/Gary\_Barlow>.

}

* **Google Play Music** - external service that allows user to listen to music, buy albums and create playlists. There is no official API but several unofficial APIs exist.

We used **Play.js** library and got the following data: favorites songs and playlists.

*Previously considered sources :*

* **Last.fm** - external beta service that allows users to listen to music and create playlists. It no longer provides API keys.
* **Seevl.fm** - external service that provides music discovery and personalization. Suffers from frequent downtime and very high response times.
* **Freebase** - external service that provides linked data sets. Has announced that it has become read-only since March 2015 and the API will be retired.

**6. Virtuoso server**

User data and preferences must be persisted in order ensure fast systems. For the persistence layer we chose a Virtuoso server.

Virtuoso is a modern enterprise grade solution for data access, integration, and relational database management (SQL Tables and/or RDF based Property/Predicate Graphs).[[1]](#footnote-1)

* Relational Tables Data Management (Columnar or Column-Store SQL RDBMS)
* Relational Property Graphs Data Management (SPARQL RDF based Quad Store)
* Content Management (HTML, TEXT, TURTLE, RDF/XML, JSON, JSON-LD, XML)
* Web and other Document File Services (Web Document or File Server)
* Five-Star Linked Open Data Deployment (RDF-based Linked Data Server)
* Web Application Server (SOAP or RESTful interaction modes).

Virtuoso is used to save data such as artists, songs and albums that the users have interest in.

# 7. **Entities and relationships**

In order to link the data used in our application, a standard model for data representation must be chosen. The first step is to identify the main entities related to music:

* **Song**
* **Artist**
* **Album**
* **Genre**
* **Band**
* **Playlist**

For each of these entities, an appropriate model from *schema.org* will be identified and used. The markup displayed by the website for each of the entities will identify with the semantic meaning associated for them and their properties as described by the *schema.org* vocabulary.

The properties of these entities that will be taken into account when providing suggestions are:

|  |  |  |  |
| --- | --- | --- | --- |
| **Song** entity:  - name  - artist  - album | | **Artist** entity:  - name | **Album** entity:  - title |
|  |

# **8. Conclusion**

This technical report serves as a reference for the requirements needed to undertake this project.

As future directions, the project can be improved by adding some functionality as: creating smart playlists and giving recommendations based on the users’s calendar.

The details of this project and the code which implements it can be viewed at the following Github page : <https://github.com/wadeproject/muxar_wade>

# **9. References**

* DBpedia Wiki : <http://wiki.dbpedia.org/>
* Node.js Documentation : <https://nodejs.org/en/docs/>
* Angular.js Documentation : <https://docs.angularjs.org/guide>
* Unofficial Google Play Music API : <https://github.com/jamon/playmusic>
* OpenLink Virtuoso Documentation : <http://docs.openlinksw.com/virtuoso/>
* Schema.org Documentation : <http://schema.org/docs/documents.html>
* DBpedia data used for interrogations:
* <http://dbpedia.org/ontology/MusicGenre>
* <http://dbpedia.org/ontology/MusicalArtist>
* <http://dbpedia.org/ontology/MusicalWork>

1. OpenLink SW home page: http://virtuoso.openlinksw.com/ [↑](#footnote-ref-1)