Musiki - DBpedia Visual Explorer

Markup Languages and Document Processing

Eduardo Silva
up201603135@fe.up.pt

Alexandra Mendes
up201604741@fe.up.pt

João Pedro Franco
up201604828@fe.up.pt

May 27, 2020

Abstract

This article describes the project developed in the context of the Markup Languages and Document Processing course. It consists of a visual tool that aims to help users navigate the DBpedia website. For that purpose, the authors present and discuss related work and introduce the data sources used in the project. In addition, the team expands on the technology chosen and Wikipedia's data model. Next, the architecture for this project is presented and described and the user requirements outlined. Finally, this article will conclude with web app displays of our finalized project followed by development details and result evaluation.

1 Introduction

1.1 Topic

For our Markup Languages and Document Processing Project, we chose Theme 8 which focuses on DBpedia and creating visual tools to help users navigate through the DBpedia graph.

DBpedia - "DB" for "DataBase" - is a project which aims to extract structured content from the information existing in Wikipedia. DBpedia allows users to semantically query relationships and properties of Wikipedia resources, including links to other related datasets. It is described as one of the most famous parts of the decentralized Linked Data effort.[1]

1.2 Context

Our project focused on implementing a dynamic graph that shows users, visually, data relating to artists, bands, songs, albums or music genres that they have searched for. The user is also able to see how all this information is connected to each other. For this purpose, we use the data available in DBpedia.

1.3 Motivation

We chose this theme not only because of its simplicity but also because it seemed a very open-ended project that gave us a wide range of areas to choose from. Seeing as we have at our disposition all of Wikipedia's data it would be infeasible to take it all on as such, so it was decided to focus on the music sphere. Furthermore, the visual-centric aspect of this project made it stand out among the other theme proposals because it was different and required the use of design and UI/UX skills that we do not apply often.

1.4 Goals

As for our goals, we expected to have by the end of this semester, a functioning tool that shows the links between various artists and genres of music in a visually interesting, interactive and clear way, so as to ease the browsing of DBpedia articles, while also allowing a comprehensive view of the search history. We also hoped to expand our knowledge on markup languages as a whole and document processing.

2 State-Of-The-Art and Related Work

Regarding related state-of-the-art work / applications, there are many libraries focused on the visualization of abstract data already implemented. Some of the most popular examples are:

- D3.js - A JavaScript library for manipulating documents based on data [2]. One of its features includes the ability to turn node and link data structures into graphical representations (figure 1).

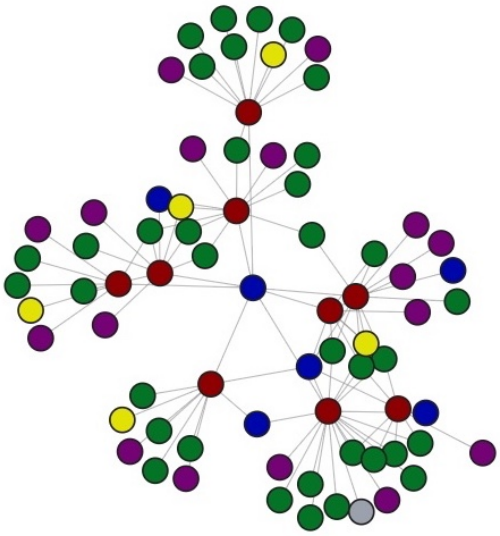

Figure 1: *D3.js graph example*

- Neovis.js - Aimed specifically at graph visualizations, it is based on another two libraries: Vis.js, which focuses on data visualization, and Neo4j, a graph database management system. It allows for an extensive customization based on labels, properties, nodes and their relationships (figure 2).

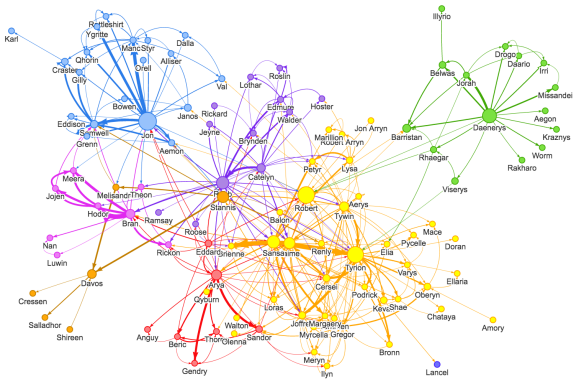

Figure 2: *Neovis.js graph example*

- Three.js - Aimed at creating animations using WebGL, it's flexible enough to allow for the visualization of data in 2D or even 3D [3] (figure 3).

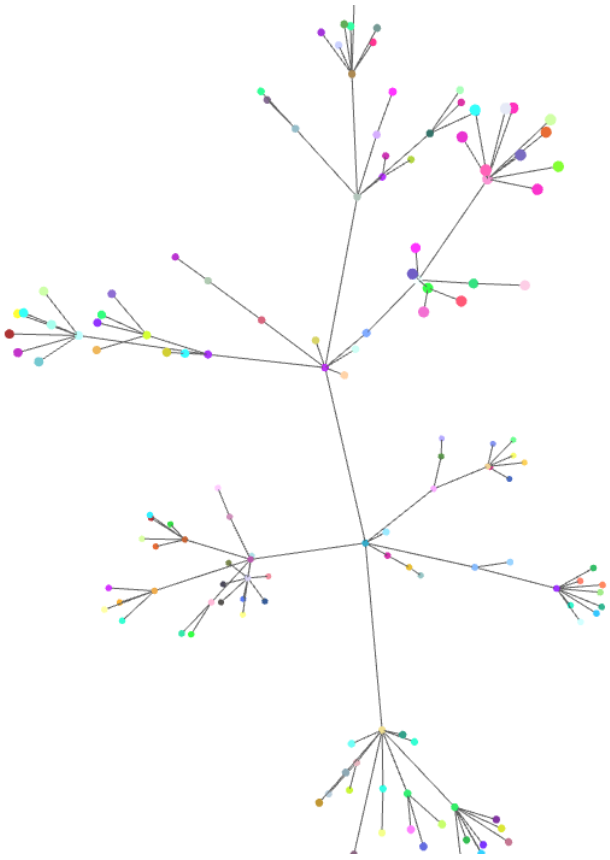

Figure 3: *Three.js graph example*

There is also a number of applications directed

towards DBpedia specifically, however, due to the nature of our project, we were only interested in those with strong visual components and goals. Some of the most relevant are:

- LODmilla - As the name implies, it's a tool for the visualization of associations in LOD (linked open data) graphs. Some additional features include the search and exploration of the neighbourhood of a node, as well as saving and sharing graph views [4] (figure 4).

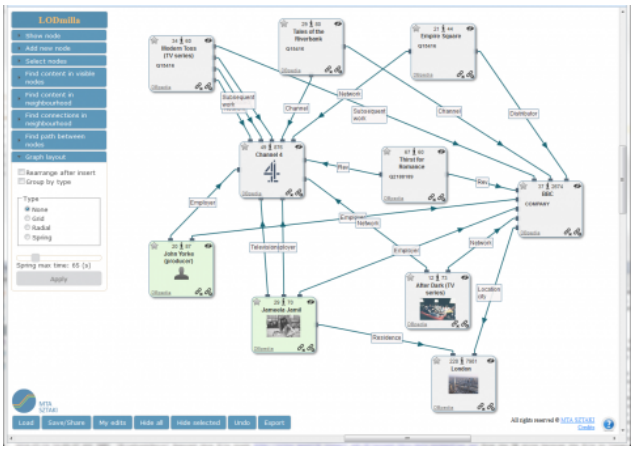

Figure 4: *Lodmilla*

- Bubble Navigator - A tool to visually navigate semi-structured or web data, providing full text search in real time [5] (figure 5).

Figure 5: *Bubble Navigator*

- Music Genre Map - A simple force directed graph, illustrating the relations between today's music genres and their popularity [6] (figure 6). However, it relates to our application due to their somewhat common theme.

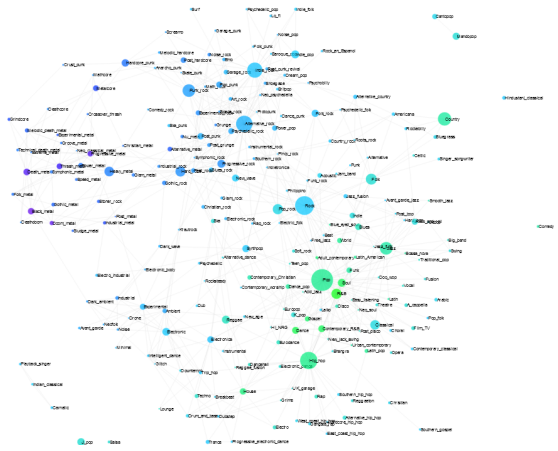

Figure 6: *Music Genre Map*

3 Data Sources

The data present in Wikipedia articles is extracted into DBpedia's RDF database using known relationships between Wikipedia infoboxes and the DBpedia Ontology. The general public can then make various requests to said database using SPARQL, an RDF query language. The RDF data model stores data in the 'subject–predicate–object' format which allows it to efficiently store and process its data as well as be represented in an abstract form (directed graph)[9].

When it comes to extracting this data from DBpedia, there are several approaches available:

- Ontology-driven REST API [7] - This is the easiest method since it would only require to set up a local server acting as as intermediary between our application and DBpedia endpoint, turning http requests into SPARQL queries.

- Extraction framework - Slightly more complicated method than the API, however it offers

more options. Ultimately, the data is presented in the form of RDF statements, easily accessible through the various modules such as the destination one [8].

- Individual data sets - DBpedia also provides monthly data sets available to save to a local storage in a number of formats and organized in a number of subsets.

It was decided that the Ontology-driven REST API would be the base for the project. This decision comes from the ease of use of this format, which allows the developers responsible for the back-end to easily make queries to DBpedia without having to implement SPARQL queries.

4 Architecture

Our application is very similar in structure to most web apps, having a front-end which is presented to the user, serving as an interface to the back-end, where user requests are processed. However, it also has some notable differences, such as the presence of two separate "back-end". The database from which data is gathered is, of course, DBpedia. To access this data, an ontology driven REST API is used, acting as a proxy server between our back-end and DBpedia. This API transforms our HTTP requests into SPARQL queries which are then sent to DBpedia's endpoint. The results are then returned to our Node.js based back-end in a JSON format, ready to be processed.
Finally, the results are then processed and presented to the user in our React powered front-end.

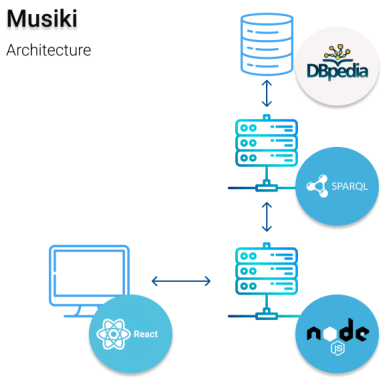

Figure 7: *Architecture*

5 User Requirements

Here are listed the main user requirements of our application.

- Allow the search of music related topics, based on available filters.
- Create corresponding children nodes when applying a filter to the selected node.
- Remove children and respective descendants when deselecting a filter on a given node.
- Display a small description about a selected node when available.
- Display an image related to a selected node when available.
- Re-arrange the graph's nodes in an interactive way.

6 Web App Displays

In this section we describe the web app by showing some of its screens and features. Its design is intended to be as clean and straightforward as possible, while also being visually appealing to the user.

When starting the app, the user is presented with a search bar where he can search for any music related topic available on DBpedia. This initial search will be filtered according to the selected filter on the options menu. If successful, the app will display a single node containing the result of that search (figure 8), and if not a message will be displayed to the user (9).

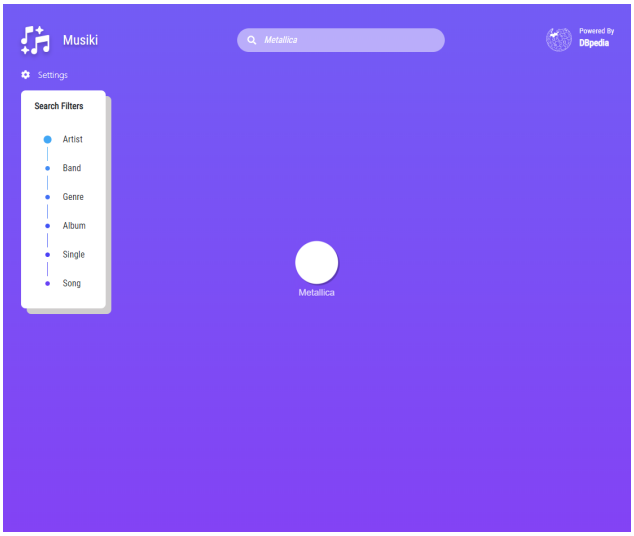

Figure 8: *Simple search scenario*

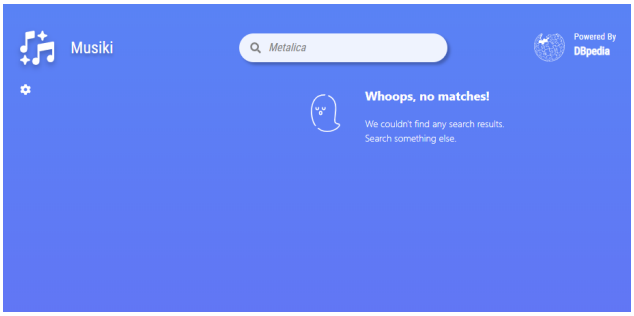

Figure 9: *Scenario where the user searches a non-existent entity*

From here, the user can select the node and pick one or more filters from the options menu. Note that the filters displayed there change according to the type of the selected node. In addition to this, a small description and picture are shown to the user regarding the selected node, if available (figure 11).

Upon activation of a filter, a search will be conducted and all pertinent results found will be displayed as children nodes of the selected node (figure 10). In the same manner, the user can also deselect a filter from a node and all children corresponding to that filter (and their descendants) will be removed.

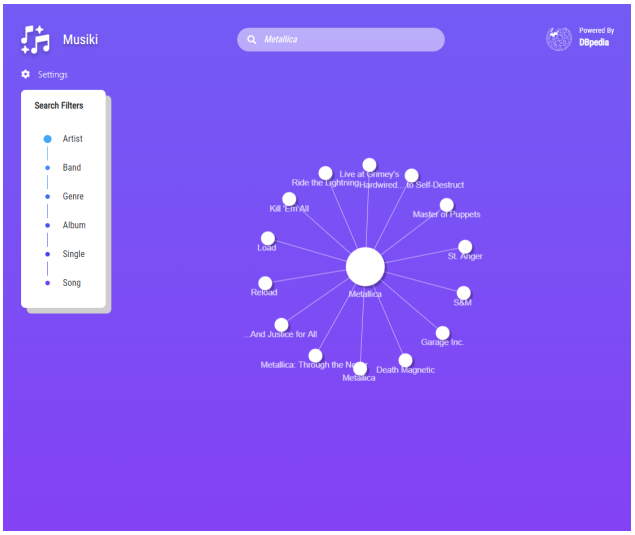

Figure 10: *Simple node expansion of Metallica albums*

The user can then repeat this process to each node, discovering new information along the way.

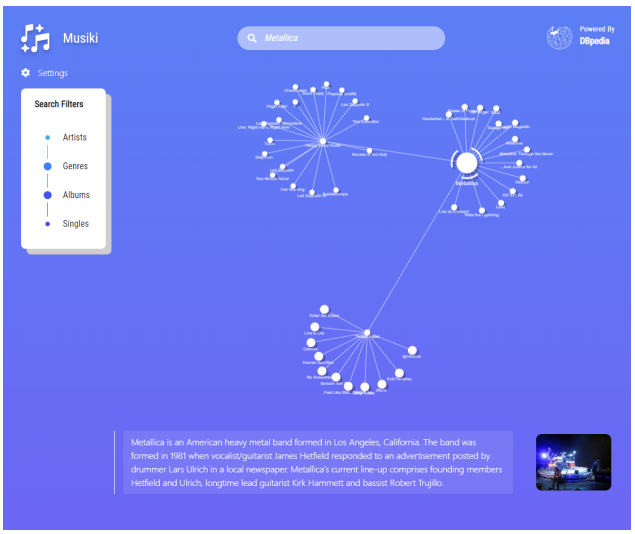

Figure 11: *Fully expanded graph with Metallica's albums and genres. The genres have their related albums*

The graph building and display was powered by the library react-force-graph[10], which is itself based on the Three.js[3] package.

7 Development Details

As stated before, to retrieve the data from DBpedia, we use an ontology-driven REST API. Here, we conduct two main request types, a "values" and "entities" request. The former is used to obtain the properties of a given entity, while the latter is used to get the entities that share the value of a property. Note that both of these requests have an upper limit on the number of results returned.
Taking this in consideration, a data structure was created containing the possible node types we wanted to feature in our application, as well as the possible filters we could apply to each of them. This is done through a set of keys and values, where we validate if a result is of a certain type depending on the value of a certain property.
This strategy gives us a good response time, however, it has its drawbacks as well. Due to the immense number of properties and values present in DBpedia, it would be infeasible to test for each single one.
Our back-end then mimics these requests (as well as adding new ones) so that our front-end can fetch the required data at any time, such as when there are filter changes, for example.

8 Result Evaluation

Regarding the initial plans for the application, some aspects changed during development, concessions and improvements alike.
First off, we chose to not implement a caching database of our own since DBpedia already seems to have this behaviour. Our testing showed that, when repeating the same requests in a short time period, the server response was much faster than when performing that request for the first time.

Secondly, we opted for scrapping the home page idea in order to not confuse the user, which might have gotten overwhelmed by the graph presented. This way, exploration by his part is encouraged.

Regarding the data provided by DBpedia, that is obtained through an ontology driven API as mentioned before. However, this still proved challenging due to the specific format of the arguments accepted and returned responses. Furthermore, it seems that not all of the features present in the API documentation seem to work. One such feature that would have been of great use to us was the option to filter results by their string start pattern. Unfortunately, since this didn't appear to work, in some request types the number of results returned contains many repeated values. Of course we filter these, but since there is a maximum of results returned parameter, the overall number of distinct results returned is sometimes not as numerous as we would like.
In light of this, we opted for not giving the user control of the maximum number of branches / children in favour of other, more interesting new features, instead replacing this with a fixed maximum number of children a node can have.
Such features include the display of a portion of the descriptive text associated with a selected node, as well as an image when available. This way the user obtains a bit more information, knowing exactly which resource that node pertains to. Some small, but noticeable, improvements were also made to the display, from small and fluid animations to the warning of users regarding possible issues in the conducted search.

Overall we consider the final developed product improved from the initial proposed design, offering a more complete experience to the user.

9 Conclusion

In conclusion, this project aims to ease the browsing of DBpedia articles, while allowing a comprehensive view of the search history. All

of this is possible due to the graph nature of the GUI. When comparing the implemented solution to other similar applications, while some have a similar basis (graph-based exploration), their visual appearance is relatively generic and lacking refinement. On the other hand, our solution has a more appealing display which, when combined with the graph based packages mentioned, results in a more dynamic and interactive experience. However, since we chose to focus on the music industry, the number of results returned are more limited than some other similar, more general, programs.

Overall, the project allowed the participants to extend their knowledge of JavaScript, XML, RDF and SPARQL as well as better understanding of document annotation and Internet of things (IoT).

References

[1] DBpedia.
https://en.wikipedia.org/wiki/DBpedia.
Accessed in April, 2020.

[2] Mike Bostock.
D3.js.
https://d3js.org/.
Accessed in April, 2020.

[3] Jonathan Saring.
11 Javascript Data Visualization Libraries for 2019.
https://blog.bitsrc.io/11-javascript-charts-and-data-visualization-libraries-for-2018-f01a283a5727.
Accessed in April, 2020.

[4] Andras Micsik.
LODmilla.
https://wiki.dbpedia.org/projects/lodmilla.
Accessed in April, 2020

[5] Dr. Miloslav Konopík.
Bubble Navigator.
https://wiki.dbpedia.org/projects/bubble-navigator.
Accessed in April, 2020.

[6] Unknown.
Music Genre Map.
https://wiki.dbpedia.org/projects/music-genre-map.
Accessed in April, 2020.

[7] Denis Streitmatte.
Ontology driven REST-Api V1.0.0.
https://github.com/dbpedia/ontology-driven-api.
Accessed in April, 2020.

[8] Rricha, Dimitris Kontokostas.
DBpedia Information Extraction Framework.
https://github.com/dbpedia/extraction-framework.
Accessed in April, 2020.

[9] Massimo Marchiori.
The RDF Advantages Page.
https://www.w3.org/RDF/advantages.html
Accessed in April, 2020.

[10] Vasco Asturiano.
react-force-graph.
https://github.com/vasturiano/react-force-graph
Accessed in May, 2020.