Departamento de Engenharia Informática
Universidade de Coimbra

Web Semântica

05 de Janeiro de 2017

Pesquisa e Recomendação Semânticas com Dados Musicais

Rocha,	Fernando	fmrocha@student.dei.uc.pt	2012131752	LEI
Silva,	João	jacds@student.dei.uc.pt	2012131780	MEI

Conteúdo

Lista de Tabelas

Lista de Figuras

1 Introdução

Este relatório é o resultado do desenvolvimento de uma plataforma de Pesquisa e Recomendação Semânticas utilizando dados no contexto da Música.

O trabalho foi realizado no âmbito da unidade curricular de **Web Semântica**, em que nos foram dados 3 requisitos por parte do Professor:

- A utilização de uma ou mais ontologias representadas em linguagens orientadas à Web Semântica (RDF, RDFa, RDFs ou OWL);
- A utilização de uma base de dados *Triple Store* para o armazenamento da ontologia e de **SPARQL** para a sua manipulação e consulta;
- A aplicação deve ser baseada em ambiente *web*, ou seja, utilizada a partir de um *browser*.

A plataforma *web* desenvolvida ao longo deste semestre tem o objectivo de fornecer informações e recomendações de Artistas, Álbuns e Faixas, recorrendo a dados representados a partir de uma ontologia.

Este projecto foi dividido em 4 *milestones* com temas específicos, decididos por nós em conjunto com o Professor. Foram estas:

1. Definição e População da Ontologia - 02/12/2016
2. Interface Web simples com Pesquisa por palavras-chave - 16/12/2016
3. Pesquisa Semântica por artistas, álbuns, músicas e ano de lançamento - 20/12/2016
4. Entrega Final - 05/01/2017

As secções posteriores deste documento estão divididas em:

- **Arquitectura do Sistema** - Nesta secção é apresentada a arquitectura geral do sistema, mostrando e explicando os seus componentes;
- **Descrição da Ontologia** - Nesta secção é apresentada a ontologia utilizada no projecto, apresentando as suas Classes, Entidades e Propriedades;

- **Descrição de Módulos** - Neste capítulo são apresentados os módulos constituintes no projecto, nomeadamente as técnicas de Pesquisa e Recomendação Semânticas;
- **Estrutura do Código** - Neste capítulo é explicado a alto nível a estruturação do código necessário no desenvolvimento do projecto
- **Conclusões e Trabalho Futuro**

2 Arquitectura do Sistema

O sistema está dividido em dois componentes essenciais:

1. Servidor **Apache Tomcat[2]** - este componente tem tanto a função de servir de *host* a toda a plataforma *web*, assim como receber *inputs* provenientes do *browser*, aceder à base de dados e processar a sua informação e, posteriormente, apresentar os resultados novamente no *browser*;

2. Base de dados *Triple Store* **Apache Jena[1]** - este componente tem a função de guardar toda a ontologia e respectiva população em memória persistente, utilizando triplos.

Desta forma, para utilizar o nosso sistema, o utilizador apenas terá que iniciar o servidor Tomcat e, a partir daí, aceder a ele a partir do *browser*.

3 Descrição da Ontologia

A ontologia foi desenvolvida através do programa *Protégé*[6], um editor de ontologias, grátis e *open-source*, que permite guardar uma ontologia em vários formatos, contendo também um reasoner.

Inicialmente, foram criadas as classes e respetivas propriedades, que foram posteriormente agrupadas através de propriedades comuns, resultando na ontologia que é apresentada na figura 1.

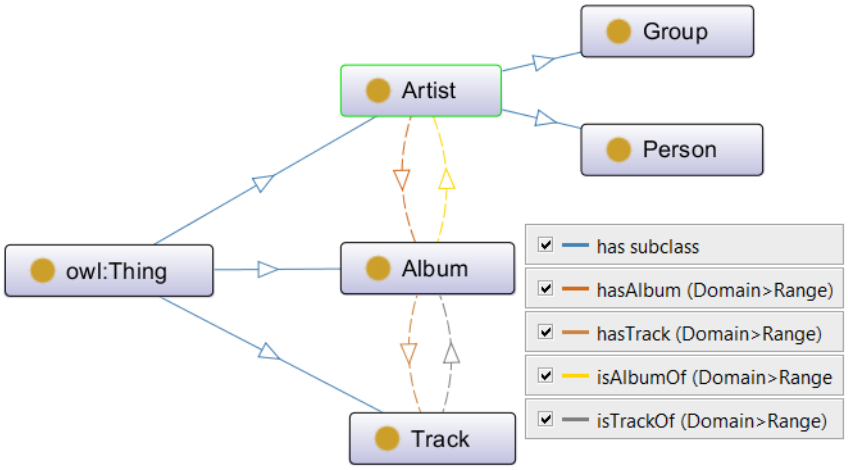

Figura 1: Ontologia

A descrição das classes, *data properties* e *object properties* estão descritas na figura 2.

De notar que, após a criação da ontologia inicial, não existem entidades, pois estas são posteriormente adicionadas aquando na população da ontologia, descrita de seguida.

A população da ontologia ocorre em dois momentos diferentes. No primeiro, utilizamos um programa escrito em *python* para obter os dados, explicado na secção seguinte. Os dados obtidos foram guardados em ficheiros JSON.

No segundo momento, foi utilizado um programa desenvolvido em *Java*, usado para carregar as informações escritas nos ficheiros JSON e, quando o auxílio da framework *Apache Jena*, carregávamos os dados para a nossa ontologia.

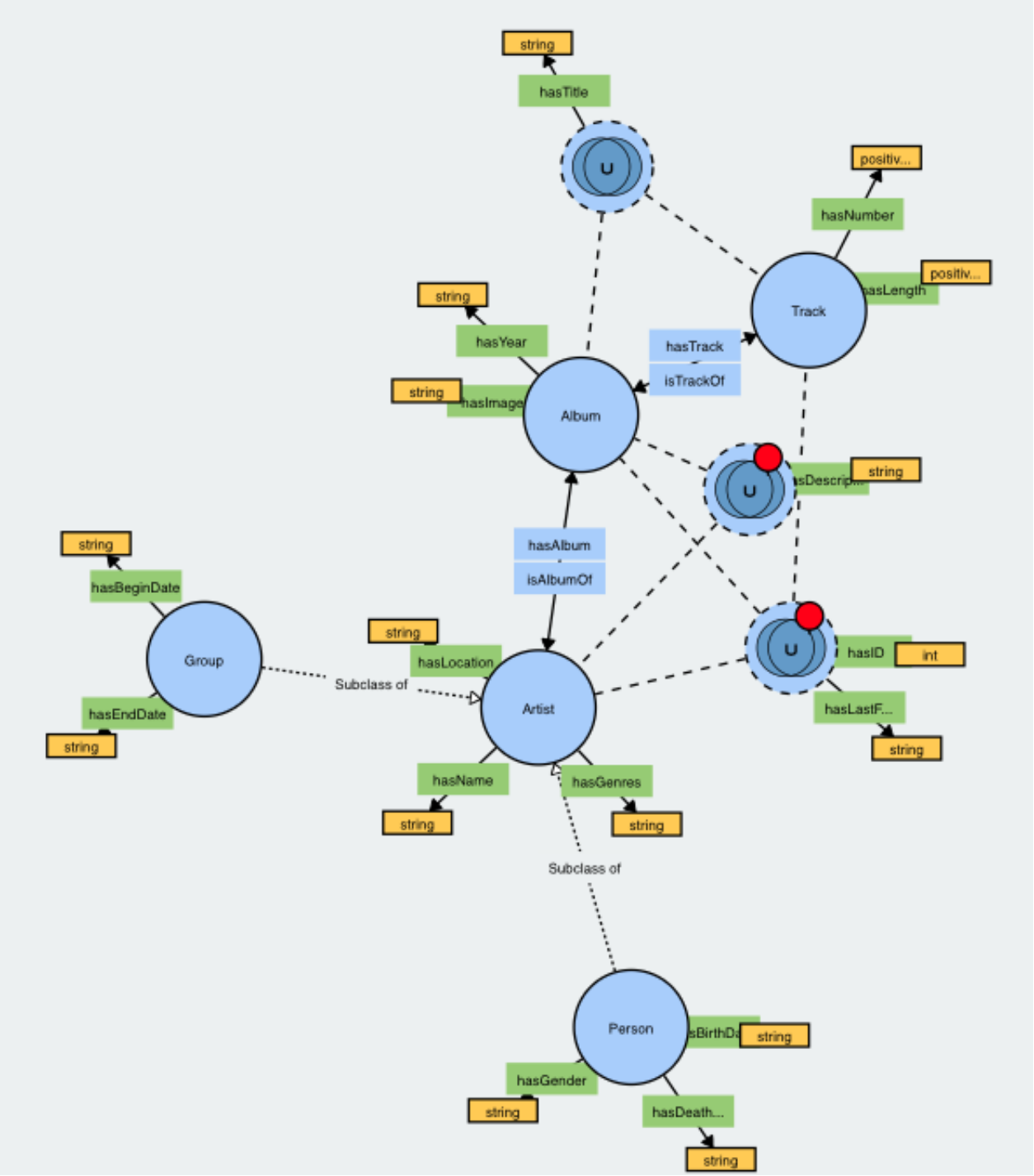

Figura 2: Ontologia completa

4 Descrição de Módulos

4.1 Obtenção de dados

Para obtenção dos dados utilizados para a população da nossa ontologia, foi necessário recorrer a um *script Python*, escrito por nós, e que faria pedidos a três APIs distintas. São essas: Last.fm API[4], MusicBrainz Web Service (recorrendo a uma *framework Python open-source* musicbrainzngs[5]) e também Spotify Web API[7].

Os dados recolhidos foram relativos aos 175 artistas mais ouvidos na Last.fm no momento da recolha final (03/01/2017), sendo a informação complementada com as informações obtidas pelas restantes APIs.

Para cada um desses 175 artistas, recolhemos os seus 15 álbuns mais ouvidos na Last.fm, as suas faixas e mais informações recorrendo novamente às restantes APIs.

4.2 Navegação

Este módulo representa a forma mais simples de interacção com um *website*. Desta forma, temos 2 abas denominadas de "Artists"e "Albums", estas servem para listar todos os artistas (Figura 3) e todos os álbuns (Figura 4) presentes na base de dados, respectivamente.

Carregando no nome de um artista, o utilizador vê-se deparado com uma página de informações do mesmo, representada nas Figuras 5 e 6, tais como: a sua imagem na Last.FM, o seu sexo (se de um artista singular se tratar), data de início e fim de carreira (ou data de nascimento e de óbito), os géneros musicais aos quais é mais identificado, uma breve descrição retirada também da Last.FM, a lista dos seus álbuns e, no final da página, seis artistas recomendados, semelhantes ao artista actual.

Carregando num título de um álbum, o utilizador vê-se deparado com uma página semelhante à do artista (Figuras 7 e 8), mas com informações relativas ao álbum seleccionado. Começa por apresentar a capa do álbum, o artista a quem pertence, a data de lançamento, uma breve descrição, a listagem das suas faixas e também albuns recomendados, a partir de artistas semelhantes.

Para além destas, a nossa plataforma tem ainda uma página de informação de Faixas (Figura 9), que possui apenas o seu título, álbum a que pertence e respectivo número, artista que a performa e sua duração.

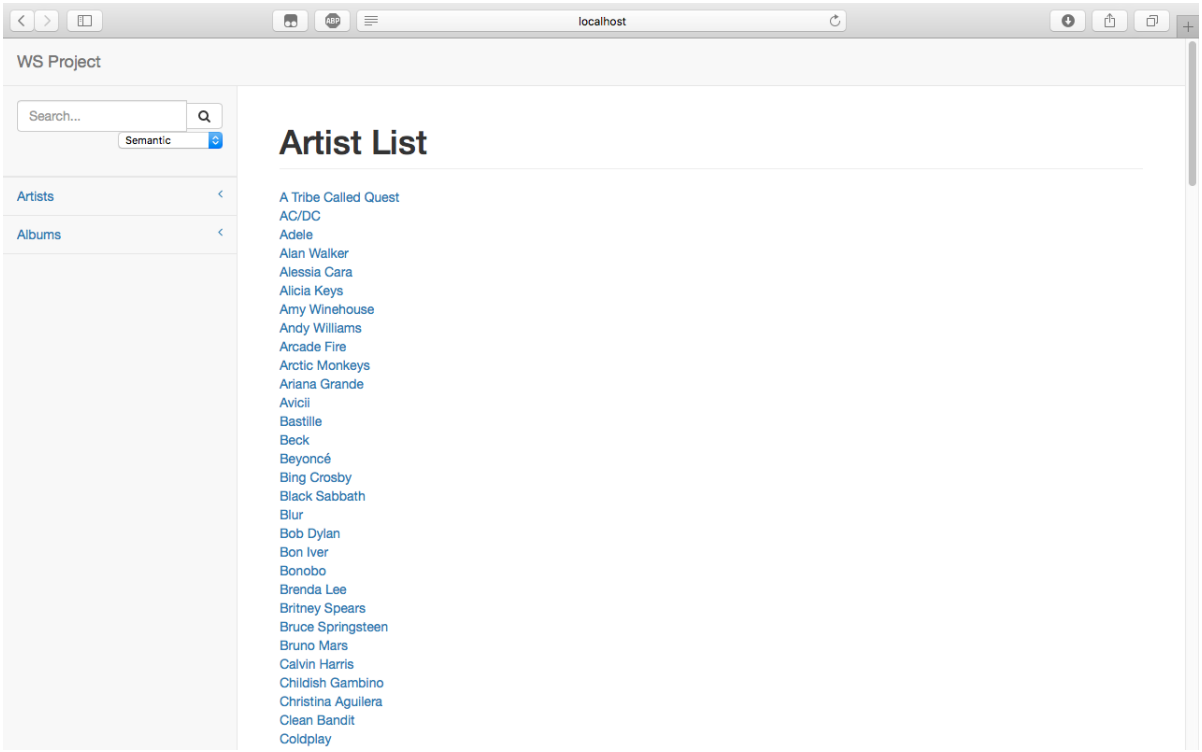

Figura 3: Listagem de Artistas

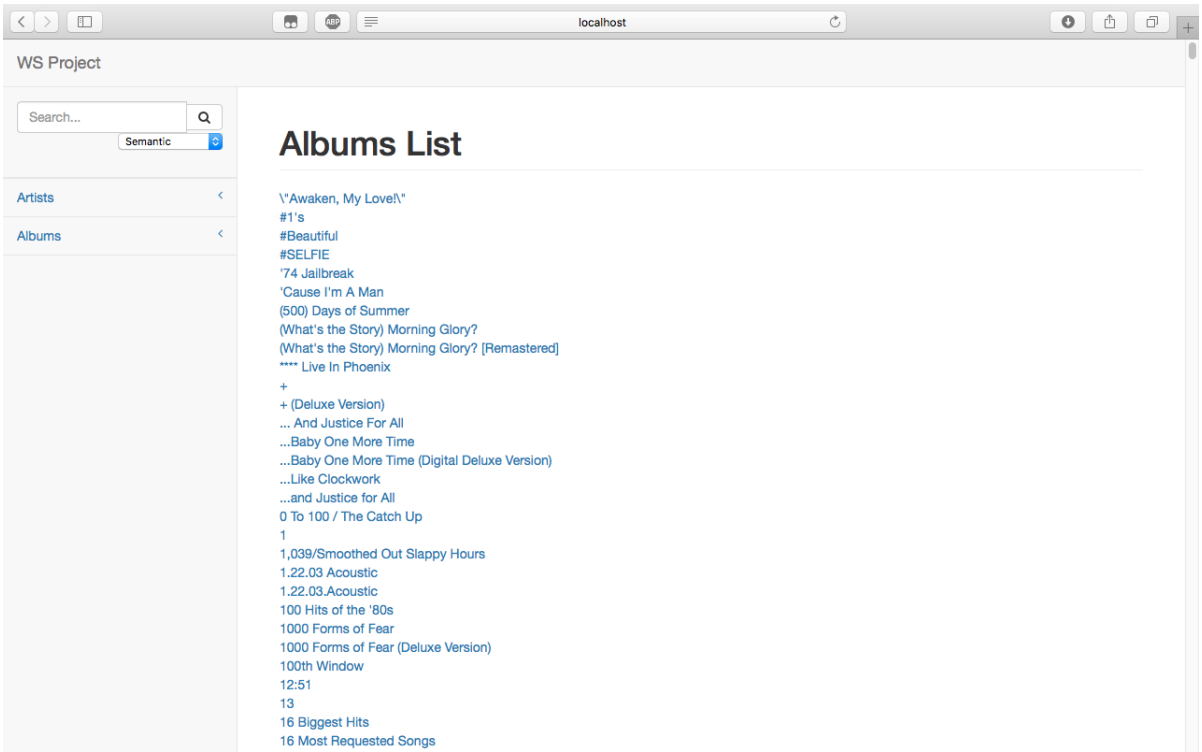

Figura 4: Listagem de Álbuns

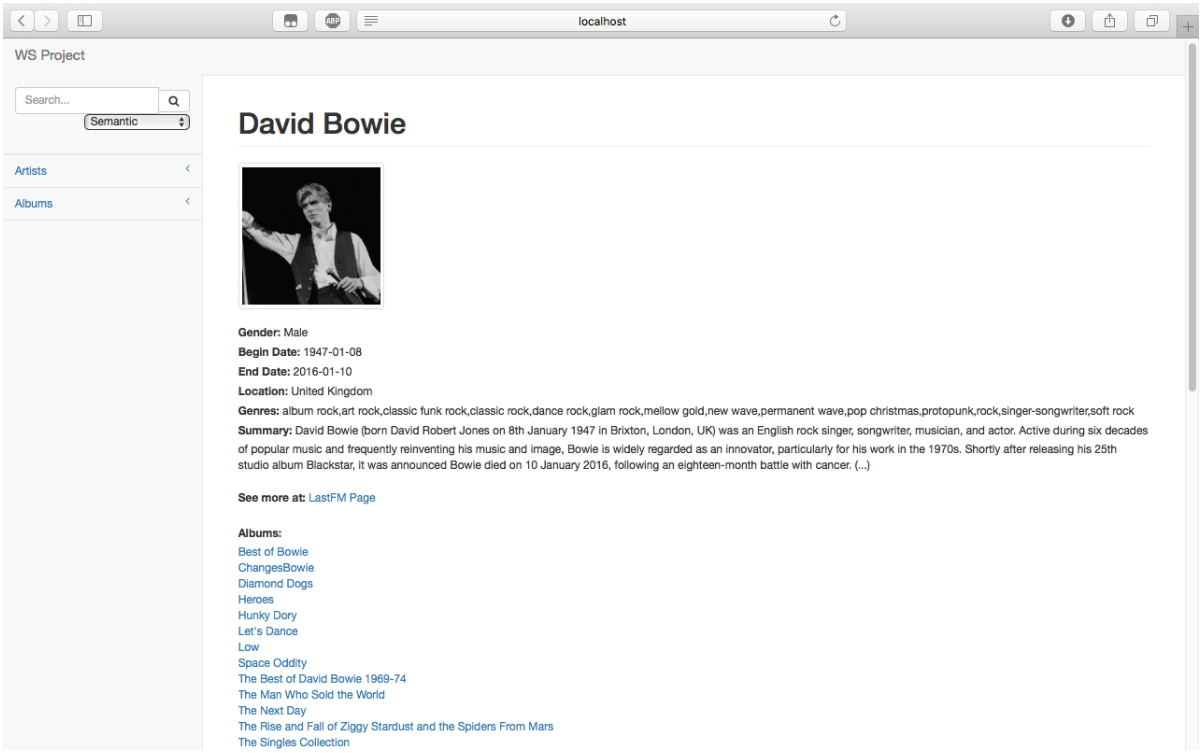

Figura 5: Página de Artista

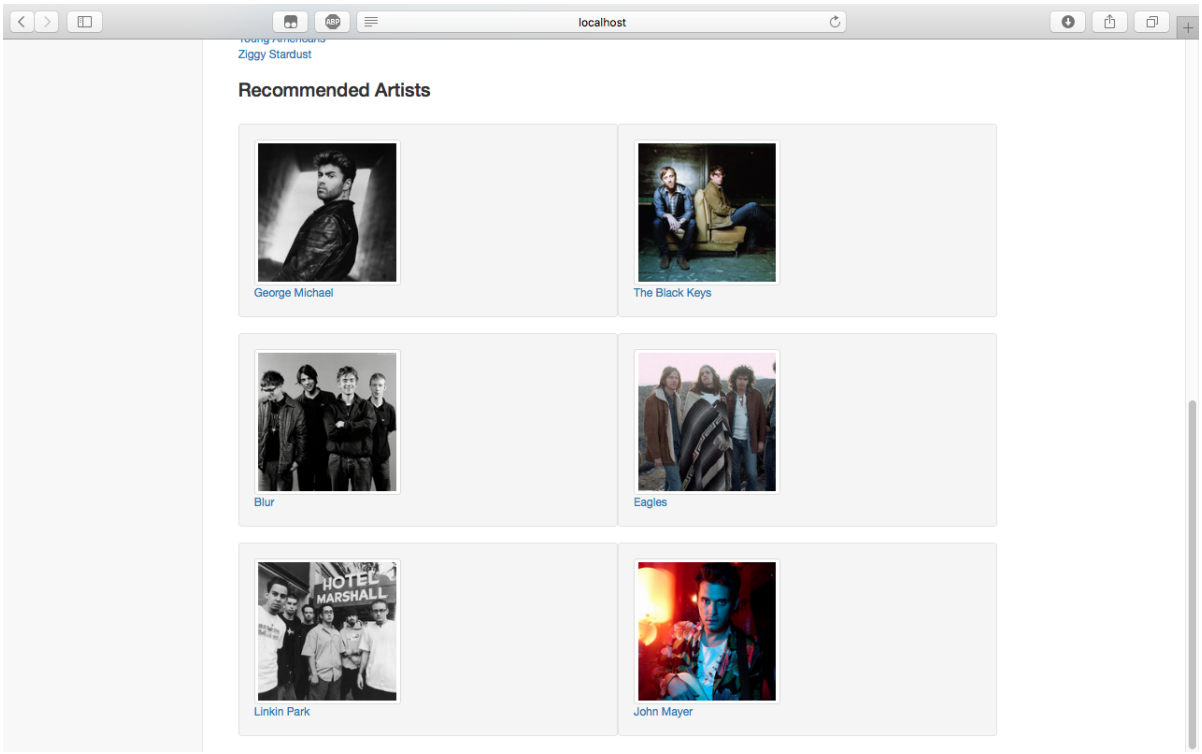

Figura 6: Página de Artista (continuação)

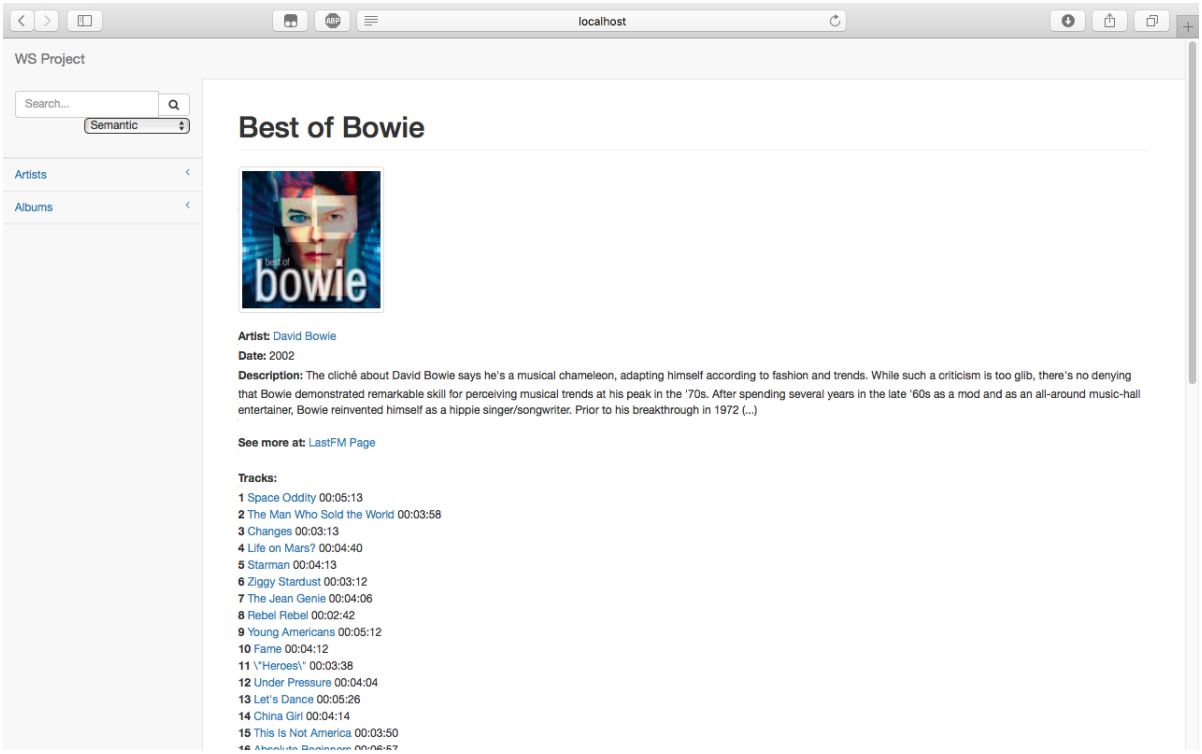

Figura 7: Página de Álbum

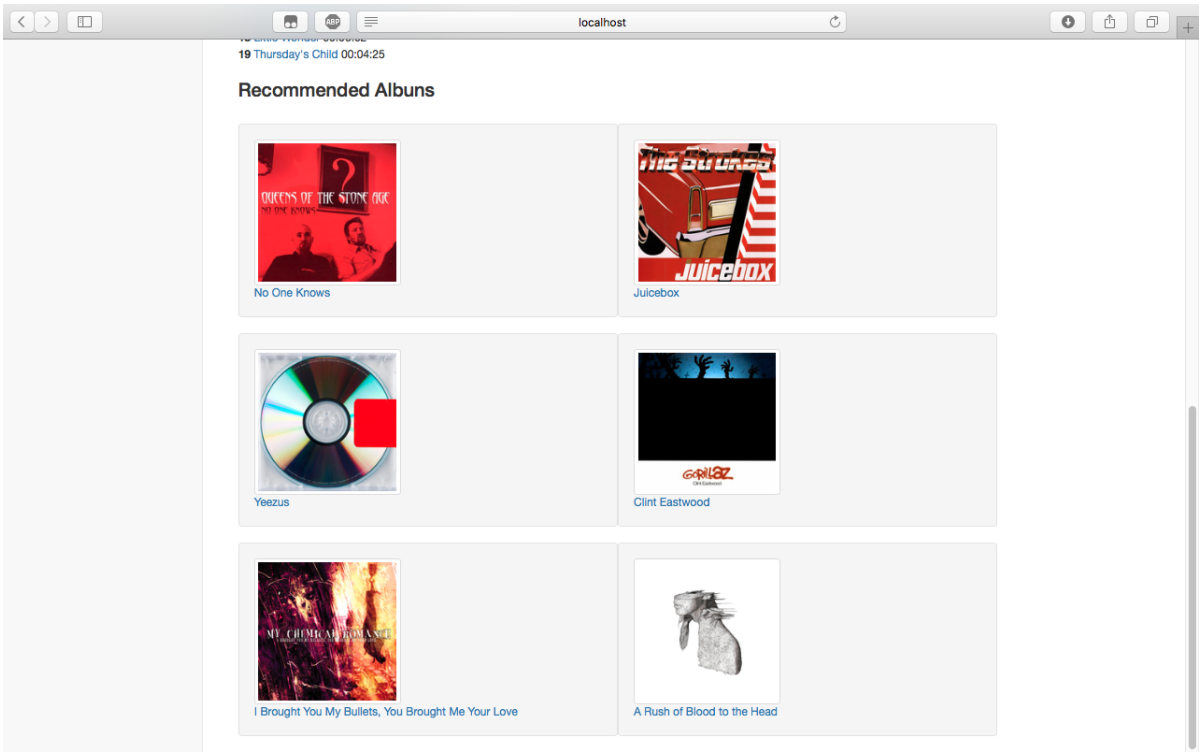

Figura 8: Página de Álbum (continuação)

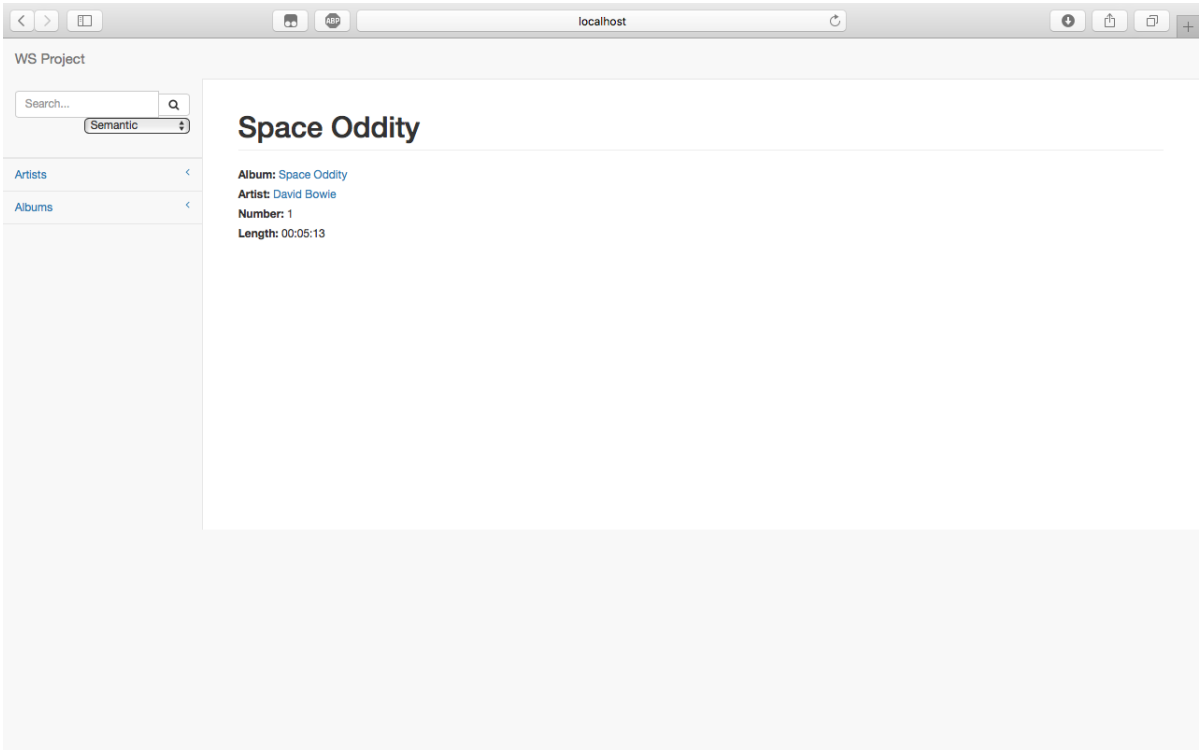

Figura 9: Página de Faixa

4.3 Pesquisa

No desenvolvimento da plataforma, e de forma a corresponder às *milestones* definidas no início, optámos por criar dois tipos de pesquisa: por palavras-chave e semântica.

4.3.1 Palavras-chave

Nesta pesquisa, o utilizador inseria o termo que pretendia pesquisar e, posteriormente, ser-lhe-iam devolvidas todas as entidades da população cujo nome/título tivesse como prefixo a expressão pesquisada. De notar que eram devolvidos Artistas, Álbuns e Faixas, tal como demonstrado na Figura 10.

4.3.2 Semântica

Quando o utilizador pretendia fazer uma pesquisa semântica, por exemplo *tracks by "Muse"*, eram seguidos alguns passos de forma a compreender o que esse pretendia. No final da explicação destes passos, também mostrados na figura 11, indicaremos na tabela 1 as palavras representantes de cada tipo de pesquisa possível. Por fim, de notar que se a pesquisa no utilizador fosse incorreta,

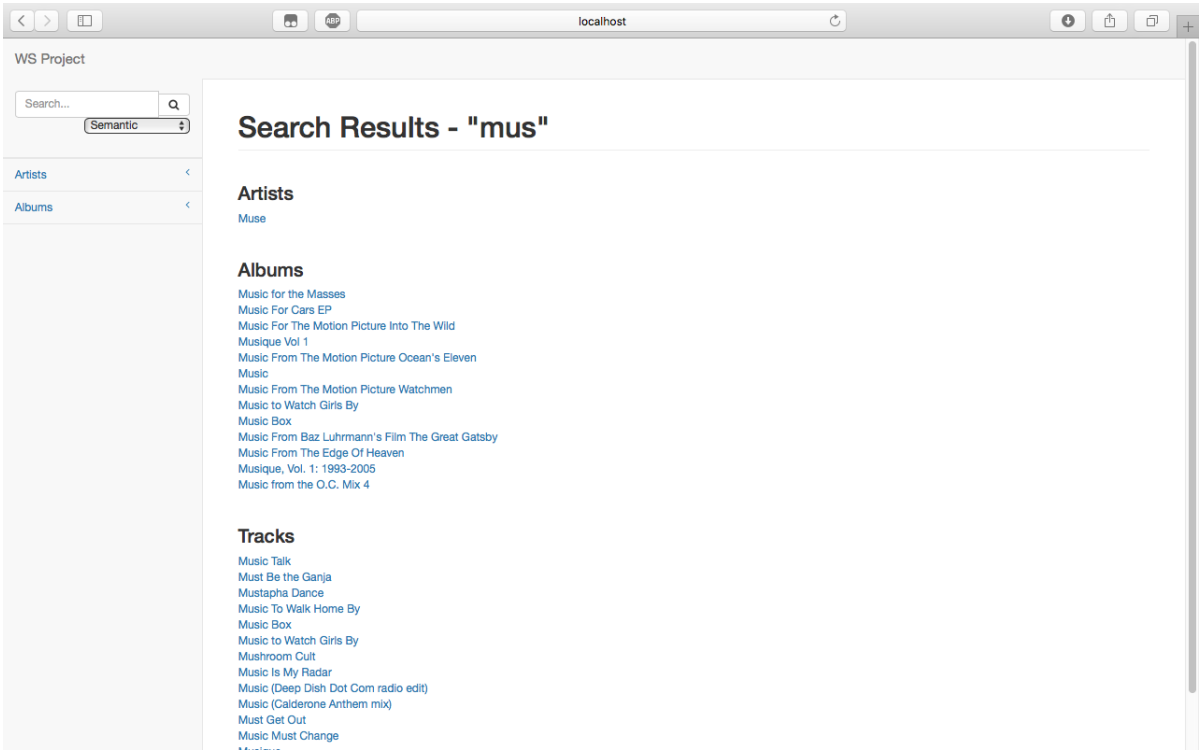

Figura 10: Resultados de Pesquisa por Palavras-Chave

seria redirecionado para a página inicial da plataforma.

1. **Parsing da pesquisa** - Neste processo a pesquisa era dividida em "palavras", podendo uma destas ser uma expressão, por exemplo aquando da pesquisa de um artista com mais de uma palavra no nome (Iron Maiden). Os termos presentes na base de dados (nomes de artistas, géneros musicais, localidade) teriam de ser pesquisados entre aspas.

2. **Obter classe da pesquisa** - Primeiro, vamos comparar as palavras pesquisadas com as classes existentes na ontologia, através de uma query SPARQL. De forma a dar alguma margem de erro ao utilizador, não comparamos diretamente as palavras, mas sim recorremos ao algoritmo *NormalizedLevenshtein*, disponibilizado na biblioteca [3], para calcular a distância entre a classe e cada palavra, com um limiar de 0.4, num intervalo de 0 a 1. De notar que, por pesquisa, apenas uma classe era representada.

3. **Obter propriedades da classe** - Para cada classe, existem palavras-chave específicas que determinam o que se pretende pesquisar. Neste passo, era então obtida a propriedade que o utilizador pretendia pesquisar, comparando as palavras da pesquisa com as palavras-chave esperadas.

4. **Definição da query** - De seguida, e com base em todos os passos anteriores, é definida a query a ser utilizada para a pesquisa, pois dependendo do que foi inserido, a forma de procura na base de dados é feita de forma ligeiramente diferente.

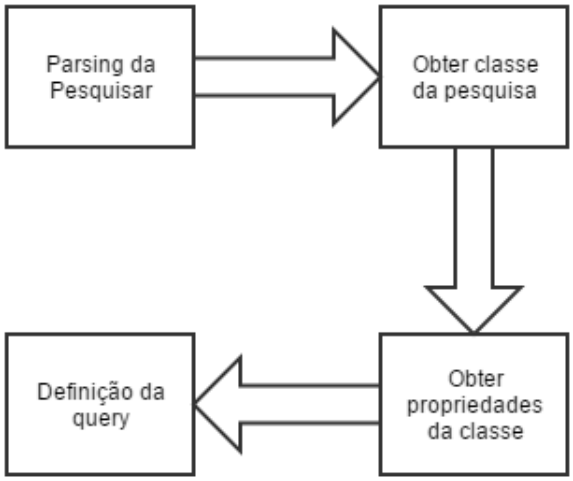

Figura 11: Pesquisa semântica

Classes	**Propriedades**	**Palavras**	**Exemplos**
Track	isTrackOf	{from}	tracks from "The Resistance"
	artist	{by, of}	tracks by "David Bowie"
Album	isAlbumOf	{by, of}	albums of "Metallica"
	hasYear	{from}	albums from 1981
Artist	hasLocation	{from}	artists from "London"
	hasBeginDate	{created, formed, founded, active, begin}	artists created "2002"
	hasBornDate	{born}	artists born "1988"
	hasEndDate	{shut, canceled, finished, inactive, end}	artists end "2014"
	hasDeathDate	{died, dead}	artists dead "2016"
	hasGender	{female, male}	female artists
	hasGenres	Género em questão	"rock"artists

Tabela 1: Palavras-chave da pesquisa semântica

De notar que na classe Track existe uma propriedade *artist* que não existe na ontologia. Isto

deve-se ao facto de as faixas não estarem diretamente ligadas ao artista, mas sim por intermediário do álbum.

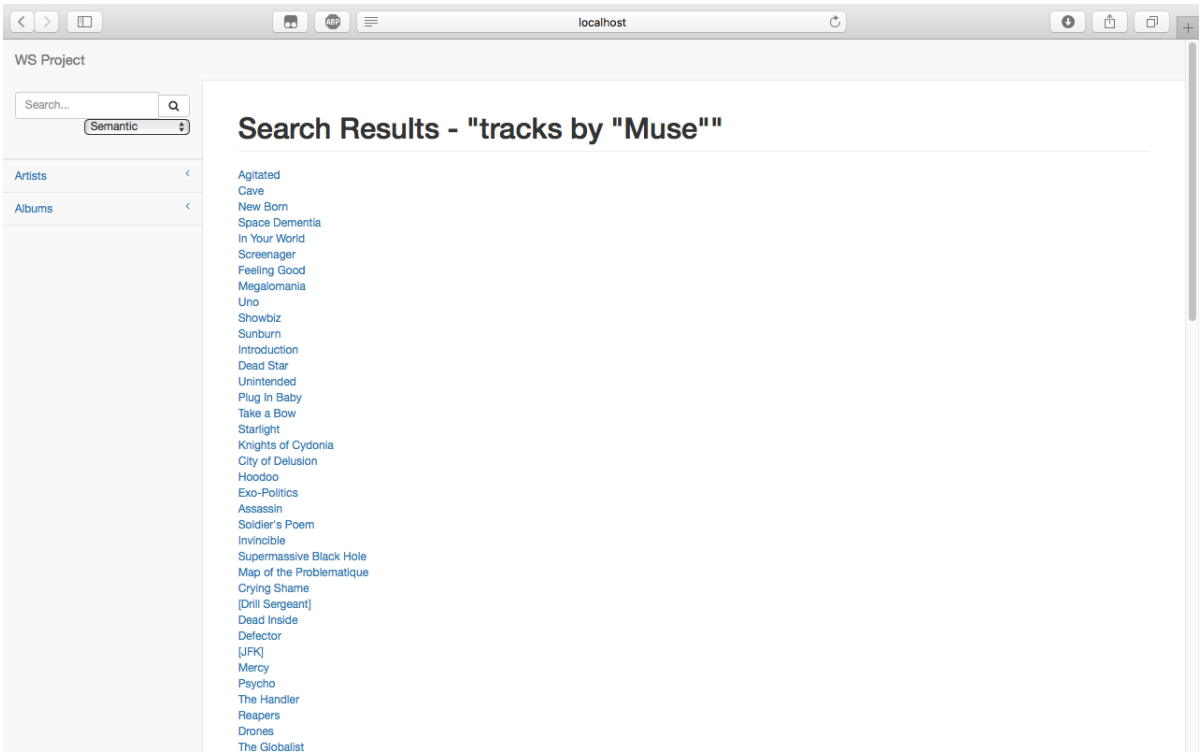

Figura 12: Resultados de Pesquisa Semântica

4.4 Recomendação

No nosso sistema de recomendação, optámos por seguir uma abordagem de recomendação por conteúdos. Desta forma, sempre que o utilizador aceder a um artista ou álbum, são-lhe recomendados artistas semelhantes (ou seja, com géneros iguais ao artista em questão), para o primeiro, e álbuns semelhantes ou do mesmo ano do álbum pesquisado.

Os passos do algoritmo utilizado para a recomendação por conteúdos são descritos de seguida, sendo também apresentados na figura 13.

1. **Obter artistas/álbuns semelhantes** - São obtidos todos os artistas ou álbuns semelhantes à pesquisa em questão. No caso de ser um álbum, são também procurados outros dois álbuns que tenham sido lançados no mesmo ano que o atual.

2. **Escolha de artistas/álbuns** - São gerados valores aleatórios, que serão os índices da lista de artistas/álbuns a serem recomendados. De notar que, no caso dos artistas, este valor é de

seis. Contudo, no caso dos álbuns, por já terem dois elementos recomendados pelo ano de lançamento, são selecionados apenas quatro.

3. **Pesquisa de imagens** - Neste passo são pesquisadas as imagens dos artistas/álbuns a retornar, tendo sido optado fazê-lo neste passo para poupar esforço desnecessário à query de obtenção de todos os artistas/álbuns semelhantes.

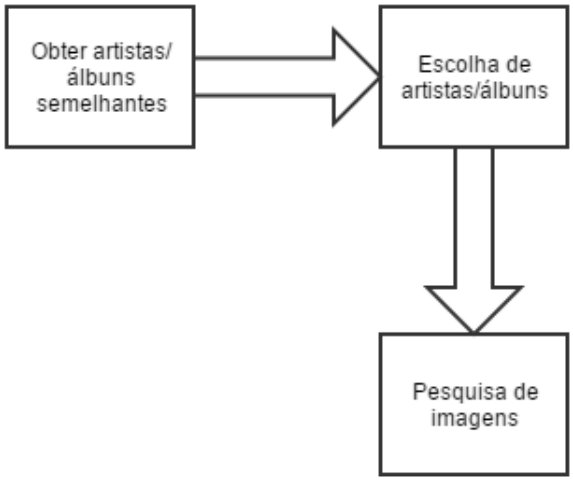

Figura 13: Recomendação

5 Estrutura do Código

O código do nosso projecto está essencialmente dividido em três partes:

- **Classes Java**
  - **CreatePopulation** - Classe responsável por pegar nos ficheiros JSON obtidos a partir do script python auxiliar, analisar a ontologia e gerar um ficheiro .owl escrito em RDF/XML com a ontologia populada;
  - **TDB** - Classe responsável por pegar na ontologia populada referida anteriormente e criar a base de dados *Triple Store* que será utilizada pela plataforma;
  - **QueryManager** - Classe responsável por todas as *queries* que o servidor necessita realizar à base de dados.
- ***Web Servlets*** - Classes responsáveis pela ligação entre a classe de QueryManager e todos os JavaServer Pages, a partir da recepção de pedidos *HTML GET* e *POST*;
- ***JavaServer Pages*** - Ficheiros responsáveis pela manipulação do *HTML* interpretado pelo *browser*.

6 Conclusões e Trabalho Futuro

Comparando a versão final da plataforma ao que tinha sido acordado entre o grupo e o professor, podemos estar satisfeitos com o que foi possível produzir. Apesar de dificuldade em obter os dados para popular a ontologia, que nos fizeram recorrer a três APIs diferentes, achamos que a nossa ontologia ficou satisfatória, com bastante informação (que poderia ter sido ainda aumentada, dada a forma que recolhemos os dados). Para além disso, a nossa aplicação é de simples utilização e intuitiva, e permite uma fácil visualização das informações pesquisadas.

As vantagens deste tipo de sistema, quando comparado aos sistemas convencionais, são óbvias, uma vez que a utilização de pesquisa semântica permite resultados mais rápidos e mais corretos. O único sítio em que peca é na necessidade de, muitas vezes, ser necessário conhecer a ontologia ou termos necessários para obter resultados satisfatórios.

Num futuro, uma forma clara de melhorar a plataforma seria adicionar recomendação baseada no histórico do utilizador, e não apenas em conteúdo. Outro ponto poderia ser pesquisar artistas/álbuns que tivessem vários géneros semelhantes, e não apenas um, como está feito agora.

Referências

[1] Apache jena, https://jena.apache.org.

[2] Apache tomcat, http://tomcat.apache.org.

[3] Java-string-similarity, https://github.com/tdebatty/java-string-similarity.

[4] Last.fm api, http://www.last.fm/api.

[5] Musicbrainz-ngs python framework, https://github.com/alastair/python-musicbrainzngs.

[6] Protégé, http://protege.stanford.edu.

[7] Spotify web api, https://developer.spotify.com/web-api/.