



xPAND
YOUR MUSIC HUB.

MIECT - EDC 2017/2018 - Trabalho Prático 2

O Professor:
Hélder Zagalo

Realizado por:
André Rodrigues e Cristiano Vagos

Aveiro, 14 de Dezembro de 2017

ÍNDICE

Introdução	3
Dados, suas fontes e sua transformação	5
Introdução	5
Last.fm API	5
Transformação dos dados	8
Acesso aos dados	11
Wikidata	11
Como pesquisar informação relevante na Wikidata	12
RSS Feed (Music News)	15
Operações sobre os dados (SPARQL)	15
Inserção dos dados após transformação	16
Verificação de Inferências	17
Similaridades	17
Membro de uma banda	19
Inserção, alteração e remoção de Comentários	21
Seleccionar os dados vindos da Wikidata	25
Publicação de Dados Semânticos usando RDFa	26
Integração de Dados da Wikidata	27
Funcionalidades da Aplicação (UI)	29
Barra de Navegação	29
Página Inicial	30
Página de Notícias	31
Página de Resultados da Pesquisa	32
Página de Artista	33
Página de Álbum	34
Página de Tags	36
Página Top Artists	37
Página Top Tracks	37
Página Top Tags	38
Página Top Countries	39
Footer	40
Conclusões	41
Como executar a aplicação	42

INTRODUÇÃO

Neste trabalho pretendeu-se desenvolver uma aplicação Web com o objectivo de agregar todos os conceitos abordados nas aulas, utilizando a plataforma Django, o uso de XSLT para transformação dos dados para RDF (a principal forma de representação de dados usada neste projeto), e outras ferramentas baseadas em ou envolvendo RDF para o desenvolvimento da aplicação Web, bem como o uso de uma triplestore (GraphDB) por forma a poder guardar os dados, dispostos em triplos (sujeito, predicado, objeto) tal como o uso de SPARQL para obter/gravar os dados na dita triplestore. Desta forma é possível depois relacionar os dados entre si, fazendo julgamentos e inferências entre os dados obtidos e guardados na triplestore.

De forma aos nossos dados poderem ser vistos de fora como entidades RDF, é usado o RDFa na representação das páginas HTML para ser possível visualizar como os dados estão ligados, a nível semântico.

ABORDAGEM

A ideia/tema do grupo foi desenvolver uma aplicação Web relacionada com Música, isto é, agregar informação sobre vários géneros musicais, nomeadamente artistas, álbuns, músicas, entre outros, e para tal utilizámos uma API de acesso a uma fonte externa de informação no formato XML, tal como abordado nas primeiras aulas da disciplina, bem como utilizámos neste trabalho um RSS feed para fornecer notícias diárias do mundo da música.

O nome escolhido para a aplicação Web foi “xPand”, retirado da palavra inglesa “expand”, pelo facto de que a nossa aplicação Web permite expandir conhecimento relativo a artistas, os seus álbuns e músicas, bem pelo facto de trabalharmos em torno do XML (daí o uso da letra “x”). O portal criado tem como objetivo mostrar informação importante sobre cada um dos artistas e álbuns, bem como mostrar rankings (tops) de forma a mostrar ao utilizador quais os trabalhos musicais e artistas que estão “na moda”.

Para este segundo trabalho prático resolvemos reutilizar o mesmo projeto do trabalho prático 1 pelo facto de já termos uma base e estrutura criada que nos permitisse trabalhar sobre a mesma, sendo que já temos criados todos os mecanismos de funcionamento e design. Este trabalho teve mais mudanças relativamente ao primeiro ao nível do *backend*, alterando a forma com que trabalhamos e guardamos os dados, demonstrando que o mesmo projeto também é possível com a tecnologia RDF, tal como abordado e praticado na "segunda metade" das aulas. Resolvemos acrescentar às funcionalidades do trabalho anterior mais algumas funcionalidades que nos pareceram úteis de forma a demonstrar o potencial que as ferramentas abordadas podem ter na vida real, como por exemplo a integração de dados provenientes da Wikidata (conforme sugerido pelo Professor), bem como a possibilidade de ver vídeos do YouTube em cada álbum de um artista, e de inferir conhecimento a partir de uma maior quantidade de dados, que obtemos a partir de vários *datasets*.

Desde já pedimos desculpa pelo facto da maioria do código de desenvolvimento não estar devidamente comentado, o que não ajuda na compreensão dos conceitos e das técnicas aplicadas para de facto programar a aplicação. O código que está disponível neste relatório é comentado tendo em vista a

compreensão dos conceitos nele aplicados. Tal facto deve-se por termos sido duas pessoas a fazer o trabalho, em vez das três como estava inicialmente previsto, e discutido previamente com o Professor.

Este relatório terá algumas semelhanças com o primeiro, uma vez que a maior parte do conceito e dados de origem são reutilizados do primeiro trabalho, como já referido. Tentámos ao máximo reduzir a quantidade de conceitos e de situações semelhantes, mantendo apenas o que de facto muda relativamente a este trabalho comparando com o trabalho anterior, sendo que este trabalho completa e acrescenta funcionalidades ao outro.

DADOS, SUAS FONTES E SUA TRANSFORMAÇÃO

INTRODUÇÃO

Aqui iremos descrever os dados que utilizamos para a utilização da aplicação. Todos os dados são obtidos a partir de APIs (Application Programming Interfaces) externas de forma a podermos ter na nossa aplicação dados reais, confiáveis e seguros sobre o nosso tema, que é música. A fonte de dados principal que usamos na nossa aplicação é a API da Last.fm (<https://www.last.fm>), um website reconhecido mundialmente que reúne informação sobre artistas, álbuns, músicas e tags (categorias) de música.

Utilizando a sugestão do Professor utilizamos também o Wikidata (<https://www.wikidata.org>), uma base de conhecimento da Wikimedia (também responsável pela Wikipédia) para complementar e completar a informação que já obtivemos. Reutilizando do primeiro trabalho, também utilizamos na nossa aplicação um feed RSS do website Music News para obter notícias diárias do mundo da música.

LAST.FM API

Para uma aplicação sobre música é importante encontrar um conjunto de dados sobre esta categoria de forma a poder mostrar informação credível, fidedigna e completa, como capas de álbuns, foto identificativa do artista/banda, bem como as músicas que dizem respeito a cada álbum, entre outros. Para tal, utilizamos a API da Last.fm para obter dados relativos aos artistas, álbuns, músicas e tags (categorias), que identificam e categorizam os tipos de dados obtidos.

Para aceder a estes dados, precisamos de aceder à API de desenvolvedor deste website, que está disponível em <https://www.last.fm/pt/api>, após registo, que é gratuito. Este registo é necessário para obter uma chave de API que terá de estar presente no URL de acesso ao método pretendido para cada pedido feito à API.

Quanto à API em si, esta é relativamente simples de utilizar e é fácil obter informação a partir da mesma, utilizando os métodos disponíveis no website da API, que também servem de documentação para um uso correto da mesma. A documentação indica os parâmetros de utilização e os dados devolvidos por esta de acordo com a pesquisa efetuada a partir de um URL.

É possível obter dados a partir desta API via XML (eXtensible Markup Language) e via JSON (JavaScript Object Notation). Como o abordado na primeira parte das aulas foi o uso do XML, e ferramentas e tecnologias usando o XML, fizemos uso da API em XML para obter os dados e modelá-los ao nosso gosto. Para este segundo trabalho, continuamos com esta abordagem, transformando depois os dados obtidos para RDF. Iremos explicar o procedimento adiante neste relatório.

API Methods

Album

- Album.addTags
- Album.getInfo
- Album.getTags
- Album.getTopTags
- Album.removeTag
- Album.search

Artist

- Artist.addTags
- Artist.getCorrection
- Artist.getInfo
- Artist.getSimilar
- Artist.getTags
- Artist.getTopAlbums
- Artist.getTopTags
- Artist.getTopTracks
- Artist.removeTag
- Artist.search

Auth

- Auth.getMobileSession
- Auth.getSession
- Auth.getToken

Chart

- Chart.getTopArtists
- Chart.getTopTags
- Chart.getTopTracks

Geo

- Geo.getTopArtists
- Geo.getTopTracks

Library

- Library.getArtists

Tag

- Tag.getInfo
- Tag.getSimilar
- Tag.getTopAlbums
- Tag.getTopArtists
- Tag.getTopTags
- Tag.getTopTracks
- Tag.getWeeklyChartList

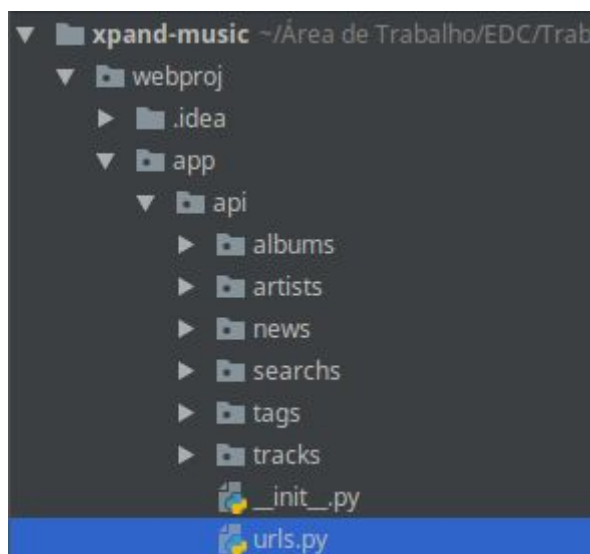
Track

- Track.addTags
- Track.getCorrection
- Track.getInfo
- Track.getSimilar
- Track.getTags
- Track.getTopTags
- Track.love
- Track.removeTag
- Track.scrobble
- Track.search
- Track.unlove
- Track.updateNowPlaying

User

- User.getArtistTracks
- User.getFriends
- User.getInfo
- User.getLovedTracks
- User.getPersonalTags
- User.getRecentTracks
- User.getTopAlbums
- User.getTopArtists
- User.getTopTags
- User.getTopTracks
- User.getWeeklyAlbumChart
- User.getWeeklyArtistChart
- User.getWeeklyChartList
- User.getWeeklyTrackChart

Por forma a facilitar o uso da API na nossa aplicação Web tendo em conta os vários parâmetros existentes para cada tipo de pedido à API, foi criado um ficheiro em Python (ver ficheiro urls.py na pasta api/) que devolve cada URL da API de acordo com os requisitos e parâmetros disponíveis para personalizar cada pesquisa tendo em conta o que pretendemos pesquisar. Este ficheiro é usado no contexto pedido na aplicação Web, e o seu uso é feito sempre que queremos obter dados a partir da API Last.fm, pelo que no código exemplificado neste relatório poderá haver referências ao mesmo.



Localização do ficheiro urls, com os URLs da API disponíveis para uso na aplicação e seus parâmetros

Entre os vários métodos disponíveis na API, é possível obter uma grande variedade de dados sobre artistas, álbuns, músicas, tags e até mesmo um Top de acordo com os dados da Last.fm. Para este segundo trabalho apenas utilizamos os métodos que nos permitem obter informação relativa aos artistas, álbuns,

músicas e tags respeitando apenas os dados sobre cada um destes tipos de informação. Os rankings/tops, bem como a descoberta de similaridades (artistas semelhantes e álbuns semelhantes - o que é possível a partir da Last.fm por si só) é feita também por nós, fazendo uso da capacidade de inferência aprendida nas aulas, ao contrário do primeiro trabalho, em que fazíamos uso dessa informação por parte da Last.fm. Esta decisão foi tomada tendo em vista a demonstração do uso de inferências num conjunto de dados.

```
<album>
  <name>Believe</name>
  <artist>Cher</artist>
  <id>2026126</id>
  <mbid>61bf0388-b8a9-48f4-81d1-7eb02706dfb0</mbid>
  <url>http://www.last.fm/music/Cher/Believe</url>
  <releasedate>6 Apr 1999, 00:00</releasedate>
  <image size="small">...</image>
  <image size="medium">...</image>
  <image size="large">...</image>
  <listeners>47602</listeners>
  <playcount>212991</playcount>
  <tags>
    <tag>
      <name>pop</name>
      <url>http://www.last.fm/tag/pop</url>
    </tag>
    ...
  </tags>
  <tracks>
    <track rank="1">
      <name>Believe</name>
      <duration>239</duration>
      <mbid/>
      <url>http://www.last.fm/music/Cher/_/Believe</url>
      <streamable fulltrack="0">1</streamable>
      <artist>
        <name>Cher</name>
        <mbid>bfcc6d75-a6a5-4bc6-8282-47aec8531818</mbid>
        <url>http://www.last.fm/music/Cher</url>
      </artist>
    </track>
    ...
  </tracks>
</album>
```

Exemplo de resposta da API Last.fm (caso do Álbum "Believe" da artista "Cher")

Assim, apenas recolhemos dados que possam ser úteis para a disponibilização de informação relativa ao artista que de outra forma não seria possível obter, cabendo-nos a nós arranjar uma solução para relacionar artistas e definir os rankings que usamos na aplicação, rankings esses que são feitos com o dado

"playcount" (que pode ser visto na imagem acima), que indica o número de reproduções que um determinado artista/álbum/música/tag teve na Last.fm.

Os dados recolhidos por esta API são os dados principais da nossa aplicação.

TRANSFORMAÇÃO DOS DADOS

Assim que os dados são adquiridos a partir da API, estes são transformados com recurso a transformações XSLT, para transformar o XML obtido a partir da API Last.fm para o formato RDF pretendido. Temos cerca de 4 ficheiros XSLT que são usados neste segundo trabalho para a transformação para RDF: "transformAlbumRDF.xsl", "transformArtistRDF.xsl", "transformTagRDF.xsl" e "transformTrackRDF.xsl" para transformar as respostas obtidas a partir dos métodos "[Album.getInfo](#)", "[Artist.getInfo](#)", "[Tag.getInfo](#)" e "[Track.getInfo](#)" da API Last.fm, respectivamente.

Os ficheiros foram desenvolvidos já prevendo que nem todas as informações estarão completas (exemplo de biografias e imagens/capas de álbum inexistentes) e tendo em conta os seguintes prefixos (namespaces) que são utilizados no decorrer de toda a aplicação, bem como nas queries SPARQL e no código HTML para introduzir RDFa no contexto da visualização da página Web:

PREFIX "cs" (custom): <http://www.xpand.com/rdf/>

PREFIX "rdf" (RDF base): <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX "foaf" (Friend of a Friend): <http://xmlns.com/foaf/0.1/>

O namespace "cs" indica "custom", e foi o link fictício que criámos para a referência dos URIs que irão ser criados no decorrer das transformações e das queries. Decidimos para distinção dos tipos e das propriedades do RDF fictício colocar os tipos com letra maiúscula e as propriedades com letra minúscula (por exemplo "cs:Artist" para indicar um artista e "cs:biography" para a propriedade biografia, que pode existir num artista/álbum/tag).

Nas transformações são criados os URIs que identificam cada um dos tipos criados nas mesmas: Artist, Album, Track e Tag. O tipo Comment, referente a comentários é criado com queries SPARQL quando um comentário é feito num artista ou álbum na aplicação.

Os URIs usados na aplicação foram criados da seguinte forma:

Artista: <http://www.xpand.com/artists/><artista>

(em que <artista> corresponde ao resto da string retirada do atributo "url" vindo da resposta API, retirando o "<https://www.last.fm/music/>")

Album: <http://www.xpand.com/album/><artista>/<album>

(construído da mesma forma que a URI Artista, retirado do "url" vindo da resposta API tanto para <artista> como para <album>)

Track: <http://www.xpand.com/track/><artista>/<track>

(<artista> construído da mesma forma que acima, <track> sendo o nome da Track, retirando os espaços em branco de forma ao URI ser válido)

Tag: <http://www.xpand.com/tags/><tag>

(construído da mesma forma que a Track, usando o nome da Tag e retirando os espaços em branco)

Comment (Artista): <URI Artist>/comment/<número do comentário>

(em que <URI Artist> é o URI do Artista, e <número do comentário> o número de comentários existentes no artista>)

Comment (Album): <URI Album>/comment/<número do comentário>

(igual ao descrito acima para Comment - Artista, só que referente ao Álbum em questão)

O formato de RDF que escolhemos para o formato final da transformação foi o RDF/XML, uma vez que é um formato que mais se assemelha ao formato com o qual obtemos a resposta da API e trabalhamos no primeiro trabalho (e na primeira parte da disciplina). Este formato não foi posto em prática nas aulas, no entanto conseguimos fazer a transformação com sucesso de forma a poder converter os dados para RDF, que é o que pretendemos. Desta forma obtivemos também experiência ao trabalhar com este tipo de formato RDF, diferente do N3 que estávamos familiarizados. Comparando com o primeiro trabalho, não realizamos qualquer Schema para verificar a integridade dos dados nem se a transformação foi válida, uma vez que já demonstrámos no primeiro trabalho que o conseguimos fazer.

Então, uma vez transformados, os dados em RDF são inseridos num grafo RDF, com recurso à biblioteca Python RDFLib (<https://github.com/RDFLib/rdfliib>) que resolvemos usar para o parse do RDF/XML obtido pela transformação. Este trecho de código exemplifica a transformação XSLT e criação do grafo com recurso ao RDFLib.

```
try:
    # Abrir o URL obtido a partir do método "Artist.getInfo"
    url = urlopen( getArtistInfoURL(quote(self.name)) )
except HTTPError:
    # Capturar a exceção. Artista não existe, ou não foi possível obter dados
    print('Artist doesn\'t exist!')
    self.artistExists = False
else:
    # Obter Current Working Directory - diretório de trabalho
    currentPath = os.getcwd()

    # Transformacao do artista para RDF/XML
```

```

xmlParsed = ETree.parse(url)
artistXSLT = ETree.parse(open(currentPath + '/app/xml/transformArtistRDF.xsl',
'r'))

transform = ETree.XSLT(artistXSLT)
finalArtist = transform(xmlParsed)
finalArtist = str.replace(str(finalArtist), '<?xml version="1.0"?>', '')

# Criar grafo RDFLib e inserir nele o RDF/XML decorrente da transformação
g = rdflib.Graph()
g.parse(data=finalArtist, format="application/rdf+xml")

```

Depois do parse do resultado da transformação, inserimos os dados inseridos no grafo RDFLib na nossa triplestore (GraphDB). Para tal, percorremos todo o grafo gerado pelo RDFLib e usamos uma query SPARQL INSERT DATA para inserir todos os dados do grafo na triplestore, que é o que pretendemos.

```

# ligar ao GraphDB
client = ApiClient(endpoint=ENDPOINT)
accessor = GraphDBApi(client)

print("Inserting artist into GraphDB...")

# Percorrer todo o grafo e inserir cada triplo (sujeito, predicado, objeto) no
GraphDB
for s, p, o in g:
    # verificar se o objeto é um URI, verificando os headers HTTP
    if (urlparse(o).scheme and urlparse(o).netloc):
        # É um URI
        query = """
insert data {<%s> <%s> <%s>}
""" % (s, p, o)
    else:
        # É uma string
        query = """
insert data {<%s> <%s> '%s'}
""" % (s, p, quote(o))

# Enviar a query via API do GraphDB
payload_query = {"update": query}
res = accessor.sparql_update(body=payload_query, repo_name=REPO_NAME)

```

Após esta operação, todos os dados estão disponíveis na triplestore para utilização posterior na aplicação.

ACESSO AOS DADOS

Tal como no primeiro trabalho, os dados são acedidos a partir de uma classe, que tem métodos que retornam os valores para cada um dos atributos definidos, ao estilo da programação orientada a objetos. Achámos que esta abordagem é a melhor pelo que no ficheiro `views.py` do Django, que chama os ficheiros HTML a serem renderizados, basta chamar as funções de acordo com um determinado valor obtendo a classe e obtendo os seus atributos a partir de uma função "get".

Após a transformação dos dados e a inserção dos mesmos na triplestore com queries SPARQL INSERT DATA, são feitas queries SPARQL SELECT para obtermos os dados pretendidos, que depois são guardados nos atributos definidos na classe, para serem usados no decorrer da aplicação.

É então percorrido o objeto JSON resultante da query, onde obtemos os valores que pretendemos. A obtenção dos dados é algo que fazemos extensivamente em praticamente tudo o que fazemos pois neste segundo trabalho tentámos depender dos dados que temos, sendo que para adicionar dados é necessário usar a funcionalidade de pesquisa na barra de pesquisa da aplicação de forma a obter dados da API Last.fm.

WIKIDATA

Por forma a completar os dados obtidos pela API Last.fm, foi nos sugerido pelo Professor o uso de dados recolhidos da Wikidata (<https://www.wikidata.org>), uma base de conhecimento da Wikimedia, também responsável pela Wikipédia, Wikivoyage, Wikisource, entre outros.

A partir da Wikidata podemos aceder à mais variada informação sobre tudo, sendo esta informação mantida pela comunidade, que pode alterar e acrescentar qualquer tipo de informação. Isto torna-se útil para as mais variadas tarefas, bem como no âmbito do nosso projeto, onde queremos obter informação para completar e agregar à informação já obtida, tal como é possível a qualquer pessoa alterar a informação disponível na Wikipédia, também é possível obter a informação disponível na Wikidata para uso na nossa aplicação.

Para o fazer usamos a biblioteca Wikidata disponível na Package Index do Python, tal como sugerido pelo Professor no enunciado do trabalho. É possível descarregar bem como saber mais sobre esta biblioteca a partir do link <https://pypi.python.org/pypi/Wikidata/0.6.1>. Esta biblioteca tem pouca documentação e exemplos de uso da mesma, pelo que inicialmente foi uma tarefa de investigação difícil da nossa parte: saber como obter a informação usando esta biblioteca sugerida. Também houve várias dificuldades na programação da aplicação. Consideramos que esta biblioteca não é a melhor para obtenção dos dados da Wikidata, e que a falta de documentação aliada ao trabalho que era necessário para compreender como a biblioteca está construída e como de facto obtemos os dados é desnecessária, simplesmente pelo facto da biblioteca estar mal feita e não haver documentação da mesma. Também consideramos que pelo uso da biblioteca referida o código resultante da mesma não é feito de forma eficiente, nem é previsível, obrigando-nos no decorrer da aplicação a adotar um método de programação defensiva, testando exceções para que nada no sistema falhe. No entanto apesar de todas as dificuldades encontramos conseguimos

aprender como usá-la para o contexto deste trabalho, acrescentando mais uma funcionalidade a esta aplicação.

COMO PESQUISAR INFORMAÇÃO RELEVANTE NA WIKIDATA

Introduzindo uma pesquisa sobre algo diretamente na Wikidata, e acedendo a qualquer item seleccionado podemos ver a informação disponível sobre o mesmo, com vários dados referentes ao item pesquisado. Cada item da Wikidata tem um identificador único associado, começado com a letra Q, que pode ter várias propriedades associadas ao mesmo (identificadas pela letra P), que estão também associadas a um outro item que descreve a mesma (identificador com a letra Q). Antes de programar o procedimento para obter informação relevante à nossa aplicação foi necessário compreender como é que a Wikidata tem os dados organizados para então os podermos utilizar para o nosso contexto.

Assim, como o nosso contexto é a música, resolvemos pesquisar um artista para sabermos que informação um item da Wikidata contém. Usamos como exemplo a artista Cher, cujo link na Wikidata é o seguinte: <https://www.wikidata.org/wiki/Q12003>. Analisando este link conseguimos obter o ID desta artista, que neste caso é "Q12003". Necessitamos então de numa primeira fase obter qualquer ID tendo em conta um nome de pesquisa. Para isso fazemos uso de um URL que faz um pedido à API da Wikidata, que devolve uma resposta no formato JSON, que depois analisamos para obter o ID pretendido. O URL é o seguinte (para uma pesquisa pela frase "Cher" na Wikidata em Inglês):

<https://www.wikidata.org/w/api.php?action=wbsearchentities&search=Cher&language=en&format=json>

Também é possível obter estes dados a partir de XML, bastando mudar o tipo de formato da resposta no parâmetro "format". Resolvemos usar o JSON (JavaScript Object Notation) para podermos demonstrar versatilidade, uma vez que este formato é também muito usado na Web hoje em dia.

Os dados que nos são importantes obter são os dados disponíveis na página da Wikidata em "Statements" (ver imagem abaixo), que nos indicam a instância do item (a artista Cher é uma instância de humano), o seu sexo, país, entre outras informações. Estas informações são as propriedades que o item tem, identificadas pela letra P, conforme referido acima. Efetuando mais algumas pesquisas sobre outros artistas conseguimos encontrar um padrão que nos permite identificar itens e propriedades de acordo com o que pretendemos, nomeadamente artistas e álbuns (as informações que mostramos numa página na nossa aplicação - página do artista e página do álbum).

Então, um artista pode ser uma pessoa (instância de humano), cuja ocupação é ser cantor, compositor, músico, rapper, ou uma combinação destas. No entanto, um artista pode também ser uma banda, que pode ter vários membros. Todos estes dados disponíveis na Wikidata fazem com que seja possível distinguir pessoas ligadas à música das que não estão ligadas, podendo nós então obter a informação que pretendemos.

- Main page
- Community portal
- Project chat
- Create a new item
- Recent changes
- Random item
- Query Service
- Nearby
- Help
- Donate
- Tools
- What links here
- Related changes
- Special pages
- Permanent link
- Page information
- Concept URI
- Cite this page

Cher (Q12003)

American singer and actress

Cherilyn Sarkisian | Cheryl LaPiere | Cherilyn Sarkisian LaPiere | Bonnie Jo Mason | Cleo | Cher Bono | Chér

[edit](#)

In more languages [Configure](#)

Language	Label	Description	Also known as
English	Cher	American singer and actress	Cherilyn Sarkisian Cheryl LaPiere Cherilyn Sarkisian LaPiere Bonnie Jo Mason Cleo Cher Bono Chér
Portuguese	Cher	No description defined	
French	Cher	actrice et chanteuse américaine	
Spanish	Cher	música estadounidense	

All entered languages

Statements

instance of

human

[edit](#)

» 1 reference

[+ add value](#)

part of

Sonny & Cher

[edit](#)

» 1 reference

[+ add value](#)

image

Cher by Ian Smith.jpg

[edit](#)

» 0 references

[+ add reference](#)

Cher - Casablanca.jpg

[edit](#)

» 0 references

[+ add reference](#)

» 2 values

[+ add value](#)

sex or gender

female

[edit](#)

» 3 references

[+ add value](#)

Exemplo de página Wikidata (Cher)

Já um álbum é um item cuja instância é álbum. Fazemos uso destes dados sobre as propriedades de instância e ocupação no caso da instância de humano para analisarmos o JSON obtido com vários IDs para sabermos qual o ID que queremos de acordo com a pesquisa efetuada. No caso de um artista, colocamos o nome do artista na procura, e no caso do álbum colocamos na procura o nome do álbum.

Para podermos saber quais são as propriedades a procurar, basta aceder às ligações disponíveis na página de qualquer item. Assim, conseguimos obter à partida os IDs de várias propriedades que usamos no decorrer da aplicação para obter a mais variada informação para agregar à já existente:

Instância de: propriedade P31;
Ocupação: propriedade P106;
Editora: propriedade P264;
Estilo musical: propriedade P136;

Website oficial: propriedade P856;
Identificador Facebook: propriedade P2013;
Identificador Twitter: propriedade P2002;
Identificador Instagram: propriedade P2003;

Identificador YouTube: propriedade P2397;
Sexo: propriedade P21;
País: propriedade P27;
Nome de nascimento: propriedade P1477;
Data de nascimento: propriedade P569;
Data de morte: propriedade P570;
País de origem da banda: propriedade P495;

Ano de criação da banda: propriedade P571;
Membros da banda: propriedade P527;
Artista intérprete: propriedade P175;
Data de lançamento: propriedade P577;
Produtor: propriedade P162;
Álbum anterior: propriedade P155;
Álbum seguinte: propriedade P156;

Claro que cada propriedade apenas irá ter valor conforme a instância de cada item procurado na Wikidata. Nem todas as propriedades existem num determinado item. Estas propriedades foram obtidas após investigação de algumas páginas na Wikidata sobre artistas e álbuns procurados por nós no seu website. Cabe-nos a nós depois no lado de desenvolvimento da aplicação controlar para cada propriedade a sua existência ou não-existência para cada item procurado, o que nesta biblioteca em específico é primordial.

Depois de obtidas as propriedades principais que queremos obter, apenas temos de testar a sua ocorrência no ID (item) obtido. Vejamos o seguinte trecho de código com um exemplo de utilização:

```
client = Client()
entity = client.get(id, load=True)
instanceProperty = client.get('P31')
try:
    instance = entity[instanceProperty].label
except Exception:
    instance = None

if 'band' in str(instance):
    print("Band detected")
elif str(instance) == 'human':
    print("Human detected")
elif str(instance) == 'album':
    print("Album detected")
else:
    print("Not valid!")
```

Inicialmente criamos um objeto do tipo Client que irá fazer a ligação à Wikidata (de acordo com o exemplo descritivo da biblioteca Python Wikidata). Obtemos a seguir a entidade com o ID que queremos analisar, e vamos de seguida verificar a sua instância (propriedade P31), acedendo ao índice correspondente à propriedade no ID obtido. Se a label da instância contiver a string "band" (uma vez que analisando a Wikidata podemos ver em alguns artistas instâncias de "band", "rock band", entre outros), significa que o ID analisado corresponde a uma banda (no exemplo referido acima). Este procedimento de acesso ao índice da propriedade do ID para verificação e retirada de informação é replicado para as propriedades mencionadas acima, o que faz com que possamos obter a informação pretendida e depois agregá-la usando queries

SPARQL para inserir um triplo na nossa triplestore de acordo com o artista/álbum, adicionando propriedades de acordo com a informação obtida.

RSS FEED (MUSIC NEWS)

Como um dos conceitos abordados na primeira metade das aulas foi o uso de RSS Feed e Atom, achámos por bem incorporar um destes conceitos na nossa aplicação de forma a deixá-la mais completa. Sendo a nossa aplicação web baseada no tema Música, faz todo o sentido usarmos um RSS Feed de um website de música, pelo que escolhemos o website Music News.

Optámos por este RSS Feed pelo facto da informação estar bem estruturada e abordar vários estilos musicais, facilitando a introdução da funcionalidade na nossa aplicação Web. Quanto à opção do uso do RSS em detrimento do Atom, foi uma opção baseada unicamente pelo facto do RSS ser o standard, daí termos decidido usá-lo para obter as notícias e demonstrar como podemos usar essa tecnologia no contexto da nossa aplicação.

Para este segundo trabalho não foi feita nenhuma modificação, sendo esta funcionalidade algo complementar e não requerida para este trabalho. Resolvemos deixar como está porque o seu propósito é completar a aplicação, tendo sido desenvolvida com sucesso no âmbito do primeiro trabalho.

OPERAÇÕES SOBRE OS DADOS (SPARQL)

Uma vez obtidos os dados, é necessário guardá-los, e para tal usamos a triplestore GraphDB, disponível gratuitamente em <http://graphdb.ontotext.com/>. Esta triplestore torna possível o uso da linguagem SPARQL (SPARQL Protocol and RDF Query Language) para manipulação da triplestore e dos triplos (constituídos por [sujeito, predicado, objeto]) nela guardados.

Ao nível do Python, as queries são executadas através da biblioteca s4api, criada pelos mesmos criadores do GraphDB (Ontotext). Essa biblioteca está disponível aqui: <https://pypi.python.org/pypi/s4api>

Para a execução das queries, é necessário enviar um objeto JSON com a query, e especificar o tipo de operação a efetuar (SELECT ou UPDATE). Estes exemplos foram dados nas aulas, pelo que não iremos especificar aqui neste relatório a sua execução. Analisando os exemplos que iremos dar a seguir é possível ver os dois métodos e a comunicação com o GraphDB através de Python.

Relativamente ao primeiro trabalho resolvemos mudar um pouco o paradigma de como os dados são guardados. Por exemplo, os tops/rankings no primeiro trabalho estavam sempre atualizados pois era feito um pedido à API Last.fm que nos devolvia os dados do momento. Também na primeira página era possível mostrar um top 5 de artistas e músicas, que agora simplesmente não aparece se não houver nada guardado na triplestore. Isto torna a nossa aplicação totalmente dependente dos dados da triplestore, evitando recorrer à API Last.fm, e apenas recorrer aos dados que temos. Foi uma decisão tomada para de facto podermos demonstrar dinamismo e reactividade na nossa aplicação.

Inicialmente quando é criado um repositório novo na triplestore, não existem dados nenhuns. A forma como foi desenvolvida a aplicação faz com que tenhamos que procurar um artista/álbum para que este seja integrado na aplicação. Assim, uma vez requerido o artista/álbum, toda a informação sobre este é adquirida através da API Last.fm (e transformada para o formato pretendido - RDF), os dados sobre este são completados com a aquisição de dados na Wikidata se existirem e uma vez obtidos os dados, estes são guardados na triplestore e só depois mostrados ao utilizador. Assim inicialmente o processo de procura/inserção/armazenamento de dados é lento, no entanto uma vez obtida a informação esta fica sempre disponível, sendo que as API externas não são mais utilizadas pois temos toda a informação disponível do nosso lado.

INSERÇÃO DOS DADOS APÓS TRANSFORMAÇÃO

Após a transformação dos dados obtidos, é chegada a altura de aplicar as queries SPARQL para inserir os dados na triplestore.

Como explicado acima, quando aplicamos as transformações, o RDF/XML gerado por estas é depois introduzido num grafo através da biblioteca de Python RDFLib. Depois percorremos todo o grafo e introduzimos todos os valores nele existentes (sujeito, predicado e objeto) na triplestore, com uma simples distinção entre os tipos de objeto (URI ou valor), da seguinte forma (tanto para um artista como para um álbum):

```
# Criação do Grafo RDFLib
g = rdflib.Graph()
# Inserção do RDF/XML resultante da transformação no grafo
g.parse(data=finalAlbum, format="application/rdf+xml")

# Dados para ligação ao GraphDB
client = ApiClient(endpoint=ENDPOINT)
accessor = GraphDBApi(client)

print("Inserting album into GraphDB...")
# Percorrer todo o Grafo
for s, p, o in g:
    # Verificar se o objeto é um URI
    if (urlparse(o).scheme and urlparse(o).netloc):
        # É um URI
        query = """
            insert data {<%s> <%s> <%s>}
            """ % (s, p, o)
    else:
        # É uma string (valor)
        query = """
```

```
insert data {<%s> <%s> '%s'}  
"" % (s, p, quote(o))
```

```
# Enviar a query via API do GraphDB  
payload_query = {"update": query}  
res = accessor.sparql_update(body=payload_query,  
                             repo_name=REPO_NAME)
```

O código mostrado acima é elucidativo quanto à forma como inserimos os dados na triplestore após a transformação.

VERIFICAÇÃO DE INFERÊNCIAS

Após a transformação e inserção dos dados na triplestore, verificamos se existe a possibilidade de poder fazer inferências sobre os dados obtidos. De facto, é o que torna a Web Semântica uma realidade, poder fazer julgamentos de forma aos dados estarem ligados entre si.

SIMILARIDADES

Uma das inferências que fazemos é a existência de similaridades entre artistas e entre álbuns (apenas mostrámos na apresentação a inferência de artistas semelhantes - lapso nosso). Na API Last.fm existia a possibilidade de podermos obter os dados que a própria API devolve, sobre artistas semelhantes. Analisando uma resposta da API a um pedido "Artist.getInfo" (ver exemplo [aqui](#)) podemos ver que existe informação sobre os artistas semelhantes a um artista. Decidimos então fazer nós as nossas próprias inferências sobre os dados obtidos.

Nós consideramos um artista ou álbum semelhante a um outro se existirem pelo menos cerca de 3 tags em comum entre os dois. As tags são atribuídas pela Last.fm, e fazemos uso delas para essa verificação. O procedimento é o seguinte:

```
# Vamos obter artistas com as mesmas tags que o artista atual  
query = ""  
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>  
PREFIX foaf: <http://xmlns.com/foaf/0.1/>  
PREFIX cs: <http://www.xpand.com/rdf/>  
SELECT ?artist1 ?artist2 ?artist2Name  
WHERE{  
    ?artist1 rdf:type cs:MusicArtist .  
    ?artist1 foaf:name "%s" .  
    ?artist1 cs:Tag ?tag .
```

```

        ?artist2 rdf:type cs:MusicArtist .
        ?artist2 cs:Tag ?tag .
        ?artist2 foaf:name ?artist2Name
    }
    """ % (quote(self.name))

# Executamos a query no GraphDB e guardamos o resultado em "res"
payload_query = {"query": query}
res = accessor.sparql_select(body=payload_query,
                             repo_name=REPO_NAME)
res = json.loads(res)

# Array para guardar todos os artistas com as mesmas tags
artistsWithSameTags = []

# Percorrer os resultados da query
for e in res['results']['bindings']:
    # Obter URI do artista atual
    uriArtist = e['artist1']['value']

    # Se artista é diferente do atual, guardar
    if(e['artist2Name']['value'] != quote(self.name)):
        artistsWithSameTags.append(e['artist2']['value'])

# Contar número de ocorrências de cada artista no array
auxCount = Counter(artistsWithSameTags)

# Para cada artista se o número de ocorrência de tags
# for maior ou igual a 3 encontramos um artista semelhante
for artist in auxCount:
    if auxCount.get(artist) >= 3:
        # artista1 é semelhante ao 2
        # artista2 é semelhante ao 1
        query = """
            PREFIX cs: <http://www.xpand.com/rdf/>
            INSERT DATA
            {
                <%s> cs:similarArtist <%s> .
                <%s> cs:similarArtist <%s> .
            }
            """ % (artist, uriArtist, uriArtist, artist)

```

```
# Enviar a query via API do GraphDB
payload_query = {"update": query}
res = accessor.sparql_update(body=payload_query, repo_name=REPO_NAME)
```

Este procedimento é feito de forma semelhante para os álbuns, para encontrar álbuns semelhantes de outros artistas.

MEMBRO DE UMA BANDA

Recolhendo dados da Wikidata sobre uma banda, uma das propriedades disponíveis que existe relativamente a uma banda é quais são os seus membros. Como é sabido, no mundo da música existe a possibilidade de um membro de uma banda começar ou ter uma carreira a solo, ou vice-versa.

A partir daqui podemos fazer uma inferência sobre um artista ser já membro de uma banda existente na nossa aplicação, e não ao contrário.

Como definimos para a banda os seus membros, apenas temos de verificar para cada membro da banda se este já existe como artista na nossa triplestore. Se tal for verdade, então esse artista é membro de uma banda, sendo mostrada na página, mostrando mais informação relativa ao artista dito, informação essa que é inferida por nós, não existindo na Wikidata nem na API Last.fm.

O procedimento desta inferência é o seguinte:

```
# Obter URIs de banda(s) que o artista seja membro e do próprio artista
query = """
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX cs: <http://www.xpand.com/rdf/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?artist ?artist2
WHERE{
    ?artist rdf:type cs:MusicArtist .
    ?artist foaf:name "%s" .
    ?artist cs:bandMember ?artist2Name .
    ?artist2 rdf:type cs:MusicArtist .
    ?artist2 foaf:name ?artist2Name .
}
""" % (quote(self.name), quote(self.name))
```

```
# Executar a query no GraphDB e guardar o seu resultado em "res"
payload_query = {"query": query}
res = accessor.sparql_select(body=payload_query, repo_name=REPO_NAME)
res = json.loads(res)
```

```
# Percorrer os resultados obtidos
for e in res['results']['bindings']:
    # Guardar os URIs
    uriBand = e['band']['value']
    uriArtist = e['artist']['value']
    # O artista é agora membro de uma banda
    query = """
        PREFIX cs: <http://www.xpand.com/rdf/>
        INSERT DATA {
            <%s> cs:isMember <%s> .
        }
        """ % (uriArtist, uriBand)
    # Executar query no GraphDB
    payload_query = {"update": query}
    res = accessor.sparql_update(body=payload_query,
                                repo_name=REPO_NAME)
```

A verificação deve ser feita dos dois lados (artista e banda). Acima verificámos para o caso do artista, agora verificamos para o caso da banda

Aplicar o mesmo procedimento acima, mas com a seguinte query:

```
query = """
    PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    PREFIX cs: <http://www.xpand.com/rdf/>
    PREFIX foaf: <http://xmlns.com/foaf/0.1/>
    SELECT ?artist ?artist2
    WHERE{
        ?artist rdf:type cs:MusicArtist .
        ?artist foaf:name "%s" .
        ?artist2 rdf:type cs:MusicArtist .
        ?artist2 foaf:name ?artist2Name .
        ?artist2 cs:bandMember "%s" .
    }
    """
```

```
}  
"" % (quote(self.name), quote(self.name))
```

Desta forma garantimos dos dois lados (artista solo e banda) que ambos estão ligados entre si, realizando mais uma inferência que mais tarde irá indicar na página as bandas em que um determinado artista é membro, se existirem.

INSERÇÃO, ALTERAÇÃO E REMOÇÃO DE COMENTÁRIOS

Aproveitando a funcionalidade criada para o primeiro trabalho, resolvemos manter o suporte a comentários tanto para o artista como para o álbum, de forma a podermos demonstrar mais operações que o SPARQL pode oferecer.

A inserção acontece da mesma forma que anteriormente mostrada para outras situações, sendo que primeiro temos de saber o número de comentários existentes no artista/álbum para que possamos atribuir um URI único a cada comentário, executando a query seguinte:

```
query = ""  
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>  
PREFIX foaf: <http://xmlns.com/foaf/0.1/>  
PREFIX cs: <http://www.xpand.com/rdf/>  
SELECT ?artista (count(?comment) as ?numComments)  
WHERE {  
    ?artista rdf:type cs:MusicArtist .  
    ?artista foaf:name "%s" .  
    OPTIONAL {  
        ?artista cs:Comment ?comment .  
    }  
}  
GROUP BY ?artista  
"" % (quote(self.name))
```

Com esta query conseguimos obter o número de comentários existentes sobre um artista, bem como o URI do artista para depois podermos guardar o comentário referente ao mesmo.

De seguida, acrescentamos o comentário ao artista (neste caso). O sistema de comentários está disponível para o artista e para o álbum.

```

# percorrer resultados da query
for e in res['results']['bindings']:
    artistURI = e['artista']['value']
    try:
        numComment = int(e['numComments']['value'])
    except Exception:
        numComment = 0

# se conseguimos obter o URI do Artista, adicionamos o comentário
if artistURI:
    numComment += 1
    # Criação do URI do comentário
    commentURI = str(artistURI) + '/comment/' + str(numComment)

    query = """
        PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
        PREFIX cs: <http://www.xpand.com/rdf/>
        INSERT DATA {
            <%s> rdf:type cs:Comment .
            <%s> cs:commentNum "%s" .
            <%s> cs:commentUser "%s" .
            <%s> cs:commentText "%s" .
            <%s> cs:Comment <%s> .
        }
    """ \
        % (commentURI, commentURI, numComment, commentURI, quote(user),
commentURI, quote(text), artistURI, commentURI)

```

Para a alteração de comentários, o procedimento é mais complexo. É necessário obter primeiro o texto antigo do comentário pois em SPARQL a alteração de algo é feito removendo o item e inserindo o novo valor. Depois de obtido o texto a alterar (texto antigo) podemos fazer esta operação. As queries foram as seguintes:

```

# Obter o texto antigo para ser alterado
# pelo novo no comentário com um certo número (numComment)
query = """

```

```

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX cs: <http://www.xpand.com/rdf/>
SELECT *
WHERE {
    ?artista rdf:type cs:MusicArtist .
    ?artista foaf:name "%s" .
    ?artista cs:Comment ?id .
    ?id cs:commentNum "%d" .
    ?id cs:commentText ?text .
}
""" \
% (quote(self.name), int(numComment))

# Alterar o texto do comentário para o novo texto
query = """
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX cs: <http://www.xpand.com/rdf/>
DELETE {
    ?id cs:commentText "%s" .
}
INSERT {
    ?id cs:commentText "%s" .
}
WHERE {
    ?artista rdf:type cs:MusicArtist .
    ?artista foaf:name "%s" .
    ?artista cs:Comment ?id .
    ?id cs:commentNum "%d" .
}
""" \
%(quote(oldText), quote(newText), quote(self.name), int(numComment))

```

A remoção de um comentário é feita da seguinte forma: verificamos se o número do comentário a eliminar existe na triplestore. Caso exista, eliminamos todos os dados referentes a esse dito comentário. As queries são as seguintes:

Obter todos os comentários ao artista

query = """

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX cs: <http://www.xpand.com/rdf/>
SELECT ?comment ?commentNum
WHERE {
    ?artista rdf:type cs:MusicArtist .
    ?artista foaf:name "%s" .
    OPTIONAL {
        ?artista cs:Comment ?comment .
        ?comment cs:commentNum ?commentNum .
    }
}
```

""" % (quote(self.name))

query = """

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX cs: <http://www.xpand.com/rdf/>
DELETE {
    ?id ?p ?v .
}
WHERE {
    ?id rdf:type cs:Comment .
    ?artista rdf:type cs:MusicArtist .
    ?artista foaf:name "%s" .
    ?artista cs:Comment ?id .
    ?id cs:commentNum "%d" .
    ?id ?p ?v .
}
```

""" \

% (quote(self.name), int(numComment))

SELECIONAR OS DADOS VINDOS DA WIKIDATA

Para seleccionar os dados vindos da Wikidata para preenchimento das classes, fazemos a seguinte query:

```
# WikiData Info
query = """
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX cs: <http://www.xpand.com/rdf/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT *
WHERE{

    ?artista rdf:type cs:MusicArtist .
    ?artista foaf:name "%s" .
    OPTIONAL {
        ?artista cs:bandMember ?member .
    }
    OPTIONAL {
        ?artista foaf:gender ?gender .
    }
    OPTIONAL {
        ?artista cs:occupation ?occupation .
    }
    OPTIONAL {
        ?artista cs:recorder ?recorder .
    }
    OPTIONAL {
        ?artista cs:genre ?genre .
    }
    OPTIONAL {
        ?artista cs:website ?website .
    }
    OPTIONAL {
        ?artista cs:facebookID ?facebookID .
    }
    OPTIONAL {
        ?artista cs:twitterID ?twitterID .
    }
    OPTIONAL {
        ?artista cs:instagramID ?instagramID .
    }
    OPTIONAL {
        ?artista cs:youtubeID ?youtubeID .
    }
    OPTIONAL {
        ?artista cs:country ?country .
    }
    OPTIONAL {
        ?artista foaf:givenName ?givenName .
    }
}
```

```
OPTIONAL {  
  ?artista cs:birthDate ?birthDate .  
}  
OPTIONAL {  
  ?artista cs:yearFounded ?yearFounded .  
}  
OPTIONAL {  
  ?artista cs:isMember ?band .  
  ?band foaf:name ?bandName .  
}  
}  
""" % (quote(self.name))
```

O OPTIONAL é usado em todos os atributos porque não sabemos se eles de facto existem na triplestore. Caso existam eles são mostrados nos resultados da query, e assim podemos utilizá-los, o que não aconteceria se não usássemos o OPTIONAL pois bastaria um atributo não existir para a query não devolver nenhum resultado.

PUBLICAÇÃO DE DADOS SEMÂNTICOS USANDO RDFa

De forma à nossa aplicação ser reconhecida por um analisador/crawler semântico, é necessário informar no código HTML através das tags "span" e dos atributos "property" e "about" de que URI estamos de facto a "falar", para que os dados possam ser compreendidos e serem usados.

Uma vez obtida a estrutura dos URIs e tendo presente os atributos/propriedades finais no decorrer do desenvolvimento da aplicação, tornou-se fácil do lado do HTML especificar sobre de que propriedade/atributo semântico estamos a mostrar no momento.

O atributo "about" indica o URI que estamos a descrever no momento. Vejamos o exemplo da página HTML onde irá aparecer os dados de um álbum:

```
{% block content %}
<section>
<div class="container" style="background-color: #f0f0f0; padding: 10px; border: 1px solid #ccc;" about="http://www.xpand.com/album/{{ artist }}/{{ album }}">
<div class="row">
<ol class="breadcrumb">
<li><a href="{% url 'artist' artist=artist %}" property="cs:MusicArtist">{{ artist }}</a></li>
<li>Albums</li>
<li class="active" property="foaf:name">{{ album }}</li>
</ol>
</div>
<div class="row">
<div class="col-md-4">

</div>
<div class="col-md-7">
<h1 class="my-4"><span property="foaf:name">{{ album }}</span><br>
<small>
by
<a href="{% url 'artist' artist=artist %}" style="background-color: #f0f0f0; padding: 2px 5px; border: 1px solid #ccc;" about="http://www.xpand.com/artist/{{ artist }}" property="foaf:name">
{{ artist }}
</a>
</small>
</h1>
{% if tags|length > 1 %}
<br>
<span class="glyphicon glyphicon-tags"></span>
{% for tag in tags %}
<a href="{% url 'tag' tag=tag %}">
<span class="label label-info" style="background-color: #f0f0f0; padding: 2px 5px; border: 1px solid #ccc;" about="http://www.xpand.com/tags/{{ tag }}" property="foaf:name">
{{ tag }}
</span>
</a>
{% if not forloop.last %}
{% endif %}
{% endfor %}
{% endif %}
<br><br>
<p><span property="cs:biography" class="text-justify">{{ wiki }}</span></p>
{% if datePublished and age %}
<p><b>Published: </b><span property="cs:datePublished">{{ datePublished }}</span> ({{ age }} years ago)</p>
{% endif %}
{% if genres %}
<p>
<b>Genres:</b>
{% for genre in genres %}
<a href="{% url 'tag' tag=genre %}" property="cs:genre">{{ genre }}</a>

```

Através deste exemplo do nosso código HTML referente ao álbum podemos ver que se trata de um álbum, cujo URI irá ser mostrado no "div" que irá conter a informação sobre o álbum.

Também é possível ver alguns atributos "property" em tags "span" que indicam qual a propriedade do URI que estamos a mostrar no momento.

É algo bastante simples de adicionar ao código HTML mas que pode fazer toda a diferença para termos informação ligada na Internet, e contribuir para um conhecimento mais profundo na Web Semântica.


Este exemplo reflete o trecho de código onde irá ser mostrado o álbum. O código HTML do artista é semelhante, e usa as mesmas propriedades/atributos como definidos aqui. Os namespaces estão declarados no ficheiro layout.html, de forma a fazer a referência dos namespaces a serem usados na aplicação uma vez só.

INTEGRAÇÃO DE DADOS DA WIKIDATA


Conforme descrito anteriormente, fazemos uso de dados da Wikidata para completar a informação disponibilizada ao utilizador final da aplicação. A API da Last.fm não nos dá toda a informação acerca de artistas e álbuns, pelo que a adição de dados vindos de uma fonte segura e extensível como a Wikidata permite-nos oferecer muito mais informação do que usando apenas a API Last.fm.

A integração de dados da Wikidata foi feita apenas nas páginas de artistas e álbuns pois são as páginas que de facto mais informação mostram. Se eventualmente tivéssemos uma página para cada música




individualmente, seria de facto mais uma tarefa a desempenhar de forma a poder ter ainda mais dados relativos a algo da nossa aplicação. Os dados são mostrados nas páginas apenas se disponíveis.

 xPand News Top Artists Top Tags Top Tracks Top Countries




Artist / **Chris Cornell**



Chris Cornell



Grunge alternative rock alternative rock singer-songwriter





Chris Cornell (born Christopher John Boyle; July 20, 1964 - May 17, 2017), Seattle, WA, United States, was an American singer/songwriter and guitarist. In 1984 he formed the band Soundgarden with bassist Hiro Yamamoto. Throughout the late 1980s and 90s, Soundgarden experienced major success as one of the major grunge bands (along with groups such as Alice in Chains, Nirvana, and Pearl Jam). In 1990, Chris Cornell worked together with the (then) Soundgarden drummer Matt Cameron

Given Name: Christopher John Boyle
Born: 20/07/1964
Death: 18/05/2017 (52 years old)
Also a member of: Audioslave
Occupation: Singer / Guitarist / Banjoist / Singer-songwriter / Pianist / Composer / Songwriter / Musician
Genres: Rock music / Traditional heavy metal / Grunge / Alternative metal / Rhythm and blues / Psychedelic music / Alternative rock / Post-grunge / Pop music / Hard rock
Record Label: Sub Pop
Official Website: <http://www.chriscornell.com>

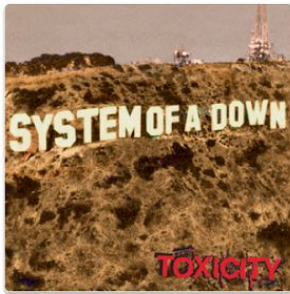
Analisando a imagem acima, podemos ver desde já algumas diferenças relativamente à informação disponível acerca do artista. A partir dos dados da Wikidata é nos possível identificar o sexo do artista, bem como o seu país, a sua data de nascimento, bem como outros dados relevantes que podem informar ainda mais o utilizador, como o website oficial e as redes sociais do artista.

As bandeiras dos países são na verdade uma única imagem que depois é preenchida por CSS tendo em conta um país, a partir do seu código de 2 dígitos. Obtemos isso a partir do website Flag Sprites (<https://www.flag-sprites.com/>). Já o código de 2 dígitos de cada país foi obtido através da API REST Countries (<https://restcountries.eu/>), já que no Wikidata obtemos apenas o nome do país. Assim, e uma vez que temos a experiência de retirar dados a partir de APIs externas, fazemos uso desta para também obter dados que nos possam ajudar. O pedido é feito a partir de um URL, que devolve uma resposta em JSON. (Por exemplo, para a procura feita sobre Portugal: <https://restcountries.eu/rest/v2/name/Portugal>) Achámos por bem mostrar a bandeira do país de forma à aplicação ter um melhor impacto visual e ser interativa para o utilizador.

Já o sexo, a indicação que o artista já morreu e links para as redes sociais são mostrados através de ícones (sendo os das redes sociais clicáveis para a rede social pretendida), pois imediatamente sabemos do que se trata.


 xPand [News](#) [Top Artists](#) [Top Tags](#) [Top Tracks](#) [Top Countries](#) 

[System of a Down](#) / [Albums](#) / [Toxicity](#)



Toxicity

by [System of a Down](#)

 [rock](#) [Nu Metal](#) [metal](#) [alternative metal](#)

Toxicity is System of a Down's second album release. Produced by Rick Rubin, Toxicity was released on September 4, 2001 by American Recordings, debuting at #1 on both the United States and Canadian charts. Toxicity received mass critical acclaim, making many end-of-year "best of" lists (such as being named SPIN Magazine's #1 record of the year), and earning a Grammy nomination for lead single "Chop Suey!". The album has sold over 6.5 million copies worldwide and is multi-platinum in the United States.

Published: 04/09/2001 (16 years ago)

Genres: [Alternative metal](#)

Record Label: American Recordings

Producer: Rick Rubin

Previous album:

[< System of a Down](#)

Next album:

[Steal This Album! >](#)

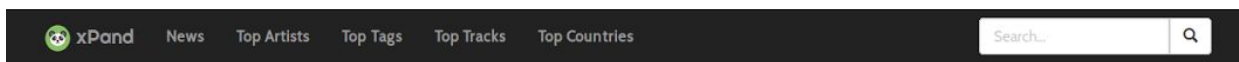
Relativamente aos álbuns, mostramos alguns dados relevantes (sempre quando disponíveis), como a data de publicação do álbum, o seu género musical, a editora e o produtor (no caso deste álbum). Tornamos também possível a navegação entre álbuns de uma forma cronológica, sendo que na Wikidata é possível adquirir qual foi o álbum anterior e o próximo álbum relativamente ao que está a ser visto. Clicando nos géneros musicais leva-nos à página da Tag que o identifica (com a experiência que temos ao trabalhar com a API Last.fm compreendemos que os géneros são Tags).

FUNCIONALIDADES DA APLICAÇÃO (UI)

Algumas funcionalidades da aplicação são semelhantes ao primeiro trabalho. No entanto para este segundo trabalho apresentamos algumas novidades que achámos por bem implementar para mostrar o potencial que uma aplicação deste tipo pode ter, integrando vários dados obtidos do exterior na mesma de forma a torná-la mais completa.

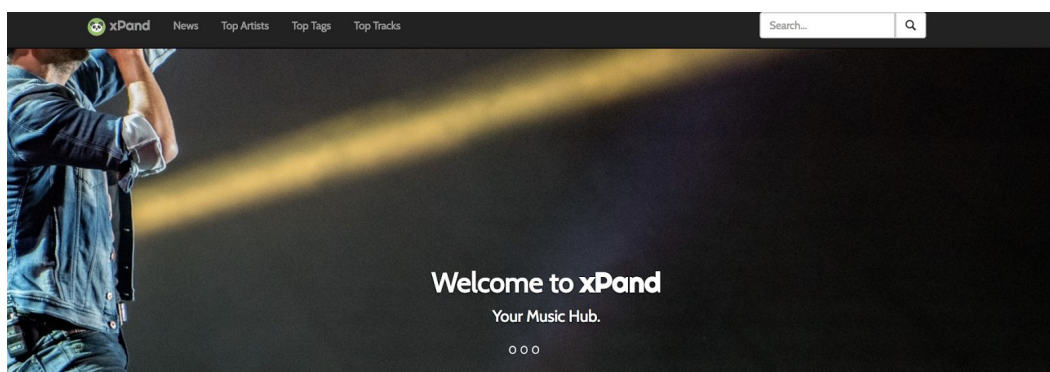
A UI da nossa aplicação é simples, intuitiva e fácil de utilizar. Decidimos usar o esqueleto fornecido pelo professor nas aulas práticas com a framework Bootstrap do Twitter, que nos oferece a possibilidade de obter uma interface limpa e agradável. Tentámos ao máximo disponibilizar uma interface capaz de dar uma experiência de navegação agradável e intuitiva ao utilizador, e acreditamos que esse objetivo foi cumprido.

BARRA DE NAVEGAÇÃO

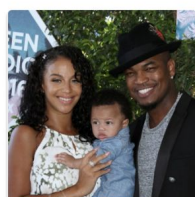


Na barra de navegação da aplicação é possível vermos o logótipo da nossa aplicação, bem como ligações para algumas páginas da aplicação. Também contém uma barra de pesquisa que permite procurar por qualquer artista ou álbum, sendo a partir daqui que se introduz novos artistas/álbuns na aplicação. Foi adicionado o Top Countries relativamente ao primeiro trabalho, pois como agora conseguimos ter acesso ao país de cada banda, conseguimos mostrar ao utilizador um ranking de países, sendo os países os das bandas/artistas existentes.

PÁGINA INICIAL



Latest News



Ne-Yo to become a father of four

The baby news is a pleasant surprise for the couple.

[Read more](#)



Fats Domino dead at 89

The musician was inducted into the Rock & Roll Hall of Fame in 1986.

[Read more](#)



Top Artists

Top Tracks



Radiohead



The Killers



Coldplay




Ed Sheeran





Red Hot Chili Peppers

Na página inicial é possível ver um slider com imagens referentes a música para dar uma ligação direta ao tema da aplicação web, e as últimas notícias referentes ao mundo da música obtidas via Feed RSS. Clicando nas notícias (botão, título da notícia ou imagem) irá abrir o link respetivo da notícia na fonte da notícia. Na lateral da página podemos ver se disponível um Top 5 de Artistas e Músicas obtidas a partir da triplestore, e link para os mesmos. Não foram feitas alterações para este trabalho.


PÁGINA DE NOTÍCIAS

 [News](#) [Top Artists](#) [Top Tags](#) [Top Tracks](#)

 **News**




Fats Domino dead at 89
The musician was inducted into the Rock & Roll Hall of Fame in 1986.





Robbie Williams was admitted to intensive care ahead of tour cancellation
Robbie Williams "can't wait to get back to work" after battling illness for the past few weeks.

[Read more](#)



John Mayer celebrates first year of sobriety
John Mayer recently took a trip to Brazil with Andy Cohen to celebrate his 40th birthday.



[Read more](#)



Na página de notícias são apresentadas as últimas notícias retiradas do RSS feed. O slider contém as últimas 6 notícias do feed, e as restantes mostradas abaixo, disponibilizando uma imagem, título da notícia e

uma breve descrição. Clicando no botão ou no título da notícia obtém uma ligação externa para a notícia original. Não foram feitas alterações para este trabalho.


PÁGINA DE RESULTADOS DA PESQUISA


 [News](#) [Top Artists](#) [Top Tags](#) [Top Tracks](#) 


Hey! Good news!


Here's what we found for "fergie":

Artists & Tracks


 Fergie


 Fergie, Q-Tip & GoonRock


 Nelly feat. Fergie


 Fergie Feat. Sean Kingston

Albums

 The Dutchess
by Fergie

 Fergie Hit Pac - 5 Series
by Fergie


 Double Dutchess
by Fergie

 Music From Baz Luhrmann's Film The Great Gatsby
by Fergie


É possível a utilização de pesquisa (Search) como o utilizador desejar. Esta irá fornecer os dados encontrados que vão de encontro com o pedido do utilizador, sendo que esses dados poderão ser artistas, músicas e/ou álbuns, podendo então aceder a eles clicando no nome ou na imagem. Caso a procura seja inválida (por exemplo acedendo ao link da pesquisa sem ter preenchido a barra de pesquisa) ou não haja resultados da pesquisa, ser-lhe-á mostrada uma mensagem com o sucedido de forma a informar-lhe.

Clicando num artista/álbum irá dar início ao processo de introdução do mesmo à triplestore se este não existir. Esta pesquisa é feita diretamente à API Last.fm através dos métodos "[Album.search](#)" e "[Artist.search](#)", sendo os resultados mostrados referentes à pesquisa efetuada e resposta recebida da API. Esta é a única forma de poder adicionar dados à triplestore.

PÁGINA DE ARTISTA





[News](#)
[Top Artists](#)
[Top Tags](#)
[Top Tracks](#)
[Top Countries](#)

[Artist](#) / [System of a Down](#)



System of a Down

[Alternative](#)
[Rock](#)
[Metal](#)
[Alternative Metal](#)
[Nu Metal](#)

System of a Down is an Armenian American alternative metal band, formed in 1994 in Los Angeles, California, USA. All four members are of Armenian descent, and are widely known for their outspoken views expressed in many of their songs confronting the Armenian Genocide of 1915 by the Ottoman Empire and the ongoing War on Terror by the US government. The band consists of Serj Tankian (vocals), Daron Malakian (vocals, guitar), Shavo Odadjian (bass, vocals) and John Dolmayan (drums).

Founded in 1995 (22 years ago)


Genres: [Rock music](#) / [Alternative metal](#) / [Progressive metal](#)

Record Label: [Sony Music Entertainment](#) / [Columbia Records](#)


Band Members: [Shavo Odadjian](#) / [Serj Tankian](#) / [Daron Malakian](#) / [John Dolmayan](#)

Official Website: <http://www.systemofadown.com/>


Top Albums




Toxicity



Hypnotize




Mezmerize




Steal This Album!


Top Songs




Chop Suey!




Toxicity



Lonely Day




B.Y.O.B.



Animals

Related



Rage Against the Machine


Be the first to comment!

Leave a comment...


Na página referente a cada artista é possível ver os dados relativos ao Artista, nomeadamente a sua imagem, biografia, tags, e se possível os dados retirados da Wikidata (informações referentes ao artista, páginas nas redes sociais, etc) bem como os seus melhores trabalhos (Álbuns e Músicas) e Artistas semelhantes, de acordo com os dados da triplestore, sendo também possível ouvir a música clicando na imagem da mesma se esta contiver um ícone "play". É então aberto o respectivo vídeo de YouTube (funcionalidade explicada adiante). Como referido anteriormente os artistas semelhantes são encontrados através de inferências. Caso haja notícias do RSS Feed relativas ao artista procurado, estas são

disponibilizadas abaixo. Também é possível ver e deixar um comentário ao artista referido, bem como apagar e modificar o comentário. O título dos comentários varia de acordo com o número de comentários feitos.

PÁGINA DE ÁLBUM

 [News](#) [Top Artists](#) [Top Tags](#) [Top Tracks](#) [Top Countries](#)

[Nirvana](#) / [Albums](#) / [In Utero](#)



In Utero

by [Nirvana](#)

[Link](#) [Genre](#) [Stats](#)

In Utero is the third and final studio album by the American alternative rock band Nirvana, released on September 21, 1993 by DGC Records. Following the success of the group's 1991 breakthrough album Nevermind, Nirvana sought to avoid that album's polished production and hired recording engineer Steve Albini to produce the record. The band recorded In Utero with Albini in two weeks in February 1993, capturing an abrasive and naturalistic sound for the record.

Published: 21/09/1993 (14 years ago)

Genres: [Grunge](#)

Record Label: DGC Records

Producer: Steve Albini


Previous album:
[Incesticide](#)

Next album:
[MTV Unplugged in New York](#)


Tracks in In Utero:


♪ Very Ape	Ⓢ
♪ Scumless Apprentice	Ⓢ
♪ Heart-Shaped Box	Ⓢ
♪ Dumb	Ⓢ
♪ Radio Friendly Unit Shifter	Ⓢ
♪ Tourette's	Ⓢ
♪ All Apologies	Ⓢ
♪ Pennyroyal Tea	Ⓢ
♪ Milk It	Ⓢ
♪ Frances Farmer Will Have Her Revenge on Seattle	Ⓢ
♪ Rape Me	Ⓢ
♪ Serve the Servants	Ⓢ

More albums by Nirvana:

[Nevermind](#)

Other albums related to In Utero:

[The Colour and the Shape](#)
by [Foo Fighters](#)

[Foo Fighters](#)
by [Foo Fighters](#)

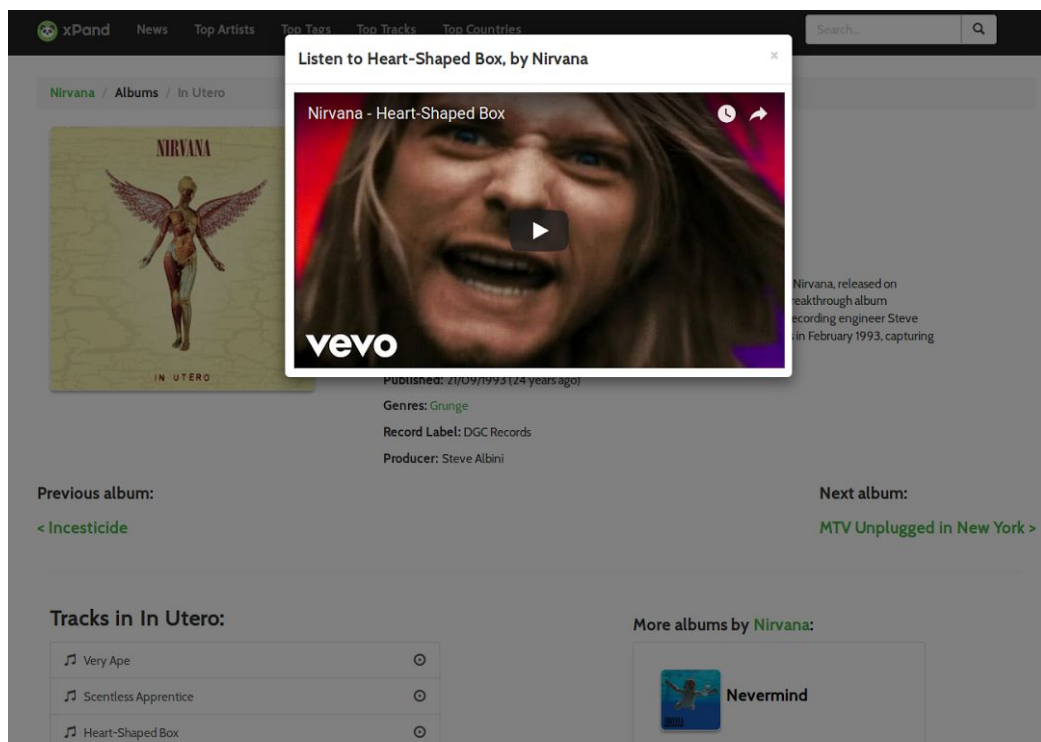
Be the first to comment!

Leave a comment...

Type your message:

[Comment](#)

Nesta página é possível ver os dados relativos ao Álbum pretendido, como a biografia (se disponível), a capa do mesmo, entre outros dados que possam existir na Wikidata referente ao álbum (álbum seguinte e anterior, data de lançamento, editora, produtor) bem como as músicas que o constituem. Na lateral da página pode ver também outros álbuns deste artista, bem como álbuns semelhantes obtidos através de inferências, sendo também possível ver esses álbuns clicando neles para aceder ao outro álbum.



Todas as músicas que tenham um ícone no lado direito (como na imagem exemplo todas apresentam) ou que tenham um símbolo de "play" (casos das páginas de artista, tag, e Top Tracks) têm um vídeo associado no YouTube (obtido na introdução da Track na triplestore - é apenas obtido uma vez), sendo que clicando nelas abre uma popup com o vídeo, sendo possível depois reproduzi-lo, ouvindo a música pretendida.


Os vídeos (nomeadamente o seu ID) foram obtidos através da API da Google, e foi necessário criar uma chave de API bem como registar uma aplicação para podermos ter acesso à dita API. Por exemplo, para obter o vídeo visualizado na imagem o seguinte link é gerado:

https://www.googleapis.com/youtube/v3/search?part=snippet&q=Nirvana%20Heart%20Shaped%20Box&key=AlzaSyCONP9A04cW6GrUPdsIC5Hd7_vplDpVVaA

Este link já contém a nossa chave de API, bem como a pesquisa a ser efetuada. A resposta da API é um objeto em JSON que depois temos que analisar para obtermos o ID do vídeo, que depois é acrescentado a uma "modal" de Twitter Bootstrap embutida no código HTML da página do álbum que através de uma função JavaScript essa "modal" é então atualizada com o vídeo e o seu título. Caso não consigamos obter o ID do vídeo, não é apresentado o ícone mostrado na página do álbum para podermos visualizar o vídeo, pelo que não será possível abrir nada.

Nesta página também é possível ver e deixar comentários ao artista referido, bem como apagar e modificar o comentário. O título dos comentários varia de acordo com o número de comentários feitos.

PÁGINA DE TAGS


 [News](#) [Top Artists](#) [Top Tags](#) [Top Tracks](#) [Top Countries](#)

Tag / rock


rock

Rock music is a genre of music started in America. It has its roots in 1940s and 1950s rock and roll and rockabilly, which evolved from blues, country music and other influences. According to the All Music Guide, "In its purest form, Rock & Roll has three chords, a strong, insistent back beat, and a catchy melody. Early rock & roll drew from a variety of sources, primarily blues, R&B, and country, but also gospel, traditional pop, jazz, and folk."


Top Artists



System of a Down




Nirvana




Foo Fighters


Top Albums




"Toxicity"
by System of a Down




"Nevermind"
by Nirvana



"Mezmerize"
by System of a Down




"In Utero"
by Nirvana




"Steal This Album!"
by System of a Down


Top Tracks




"Smells Like Teen Spirit"
by Nirvana




"Come as You Are"
by Nirvana



"Chop Suey!"
by System of a Down



"Everlong"
by Foo Fighters

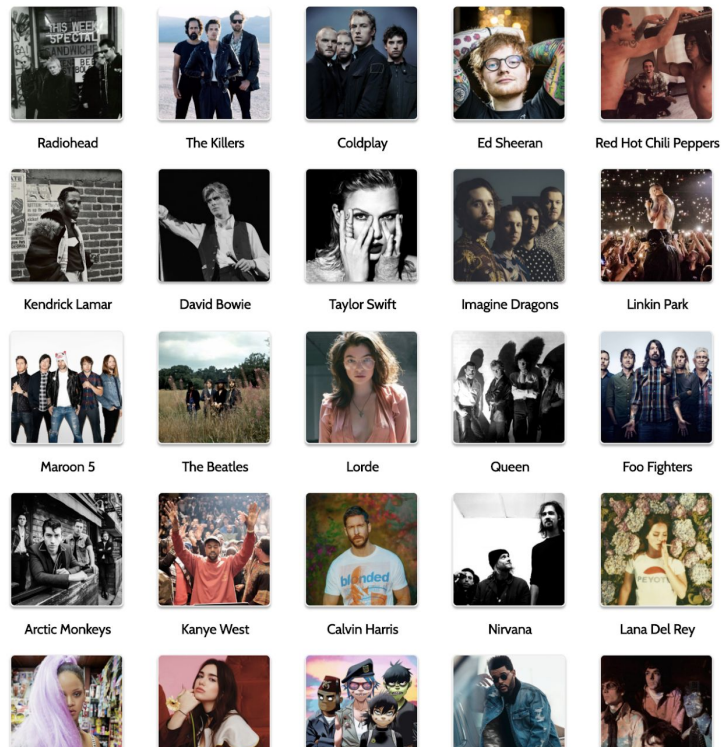


"Lithium"
by Nirvana

Ao carregar numa tag disponibilizada na descrição de um artista e/ou de um álbum, é possível obter uma breve descrição da tag, e os melhores artistas, álbuns e músicas que correspondem à tag seleccionada, sendo que os artistas/álbuns/músicas são mostrados tendo em conta a classificação criada por nós, através do "playCount" da API Last.fm. Clicando num artista ou álbum irá direccioná-lo para a sua página respectiva, clicando em cima da imagem da música abre o respectivo vídeo de YouTube caso apareça o símbolo de "play" para reproduzir o vídeo, tal como na página de álbum.

PÁGINA TOP ARTISTS

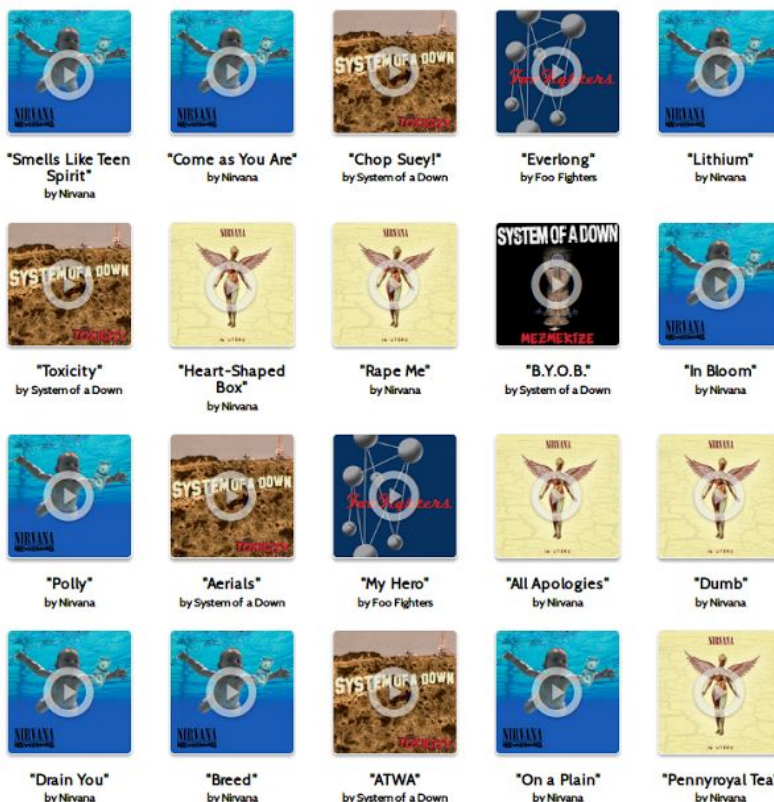
Top Artists



A página Top Artists mostra a classificação dos artistas de acordo com os dados disponíveis na triplestore, através do "playCount" que foi obtido da API Last.fm ao adicionar o artista à triplestore. No fundo desta página é possível ir para a página seguinte para ver os próximos artistas de acordo com o ranking. Clicando num artista leva-o para a sua página respectiva.

PÁGINA TOP TRACKS

Top Tracks



A página Top Tracks mostra a classificação das músicas, mostrando também os respetivos artistas de acordo com os dados disponíveis na triplestore. Caso esteja disponível o vídeo de YouTube respectivo da música é possível reproduzir o vídeo da música seleccionada, da mesma forma que mostrado anteriormente na página do álbum. No fundo desta página é possível ir para a página seguinte para ver as próximas músicas de acordo com o ranking. Clicando num artista leva-o para a sua página respetiva.

Top Tags

rock



Coldplay



Linkin Park



Red Hot Chili Peppers



David Bowie

electronic



Daft Punk



Depeche Mode



Crystal Castles



Björk

seen live



The National



Editors



Biffy Clyro



Major Lazer

alternative

A página Top Tags tem como objectivo mostrar as melhores tags (palavras-chave) e os melhores artistas de cada tag. Clicando numa Tag leva-o à página respetiva de cada tag, clicando num artista leva-o à sua página respetiva.

PÁGINA TOP COUNTRIES

Top Countries

United States of America



System of a Down



Nirvana



Foo Fighters

Portugal



Da Weasel



Sam The Kid

United Kingdom



Muse

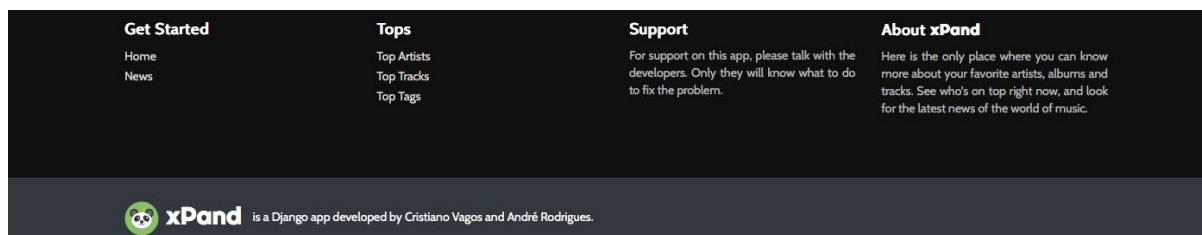


Iron Maiden



Def Leppard

A página Top Countries tem como objectivo mostrar os melhores artistas de cada país, obtidos através dos dados da Wikidata que estão disponíveis para cada artista quando possível. Clicando num artista leva-o à sua página respetiva.



O footer, situado no final de cada página fornece o acesso à página principal, à página de notícias e aos Tops (Artistas, Tracks e Tags), ainda disponibiliza suporte da aplicação e uma breve descrição sobre a aplicação (About). Apenas acrescentamos o Top Countries na secção Tops, relativamente ao primeiro trabalho.

CONCLUSÕES

Acreditamos plenamente que os objetivos deste trabalho foram cumpridos. A partir desta aplicação conseguimos demonstrar o uso de uma plataforma Django (uma das mais usadas hoje em dia para desenvolvimento Web) bem como o uso de tecnologias que envolvam o tipo de representação de dados em RDF, que permitem que se possam fazer inferências de forma a poder relacionar os dados obtidos a partir de várias fontes de dados, um dos objectivos da Web Semântica.

Este trabalho final reflete a junção dos dois projectos, sendo que este último acaba por completar os dados de forma a reunir muito mais informação sobre os artistas, álbuns e músicas, permitindo também ao utilizador ter uma melhor experiência de navegação. Consideramos que fizemos uma boa escolha relativamente ao tema aplicado, tirando proveito pelo facto de termos desenvolvido uma boa estrutura e base para melhorar, que foi o que foi feito do primeiro para o segundo trabalho. As funcionalidades criadas para melhorar o projeto são de facto úteis no seu contexto, e seriam claramente utilizadas numa aplicação real. A junção de vários dados de várias fontes faz com que seja possível criar uma aplicação completa e que junte informação útil ao utilizador.

Chegamos ao fim desta disciplina com este trabalho sabendo que tudo fizemos para de facto podermos demonstrar e pôr em prática os objetivos da mesma. Mesmo trabalhando com apenas dois elementos, o que nos sobrecarregou imenso durante a execução do mesmo, achamos que fizemos um excelente trabalho (e a nota obtida no primeiro trabalho demonstra isso). Tal como no primeiro trabalho, muito mais poderia ser feito e certamente haveria espaço e potencial para melhorar. Sabemos plenamente que o poderíamos fazer e temos capacidades para isso, e pensamos que o trabalho efetuado o evidencia.

Acabamos com o sentimento de dever cumprido, aprendendo mais uma tecnologia que certamente iremos utilizar no futuro, sendo que a Web Semântica e a interligação entre APIs e vários tipos de informação mostrou-nos alguns casos de sucesso hoje em dia (Trivago, Kuantokusta, entre outros) que fazem uso disso mesmo, mostrando ao utilizador a informação que este pretende sem precisar de comparar por si próprio em

várias plataformas. A nossa aplicação é em parte o agregar de vários tipos de informação para um objetivo final: mostrar ao utilizador informação do mundo musical.

Apesar da integração de dados da Wikidata ser interessante no ponto de vista académico, tendo grande impacto no desenvolvimento e no resultado final, achamos que a biblioteca sugerida foi mal desenvolvida, tendo impacto no tempo dispendido para a aprender e pôr em prática no trabalho. O facto de não haver documentação da mesma causou dificuldades na compreensão dos conceitos e do que era necessário para a integrar com sucesso na aplicação. Sugerimos a investigação de uma outra biblioteca ou até mesmo a inclusão de algo que por exemplo faça uso do SPARQL Query Service da Wikidata (https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service), permitindo fazer pesquisas diretamente à Wikidata, de uma forma mais eficiente e que seja "programmer friendly", de forma a facilitar a abordagem a ter do ponto de vista do desenvolvedor, ou uma outra biblioteca ou dataset que seja mais fácil de usar e de aprender.

COMO EXECUTAR A APLICAÇÃO

A aplicação foi desenvolvida usando na sua maioria sistemas Unix. No entanto acreditamos que a mesma é multi-plataforma desde que tenha as bibliotecas necessárias à sua correta utilização. Tentámos usar ao máximo ferramentas nativas e apenas recorrer a outras caso estritamente necessário.

Para executar a nossa aplicação corretamente deve instalar o Python3, Django e o GraphDB (<http://graphdb.ontotext.com/>). Depois a partir do terminal Unix executar:

```
sudo apt-get install python3-lxml
sudo pip3 install rdflib Wikidata s4api
```

Deve depois iniciar o GraphDB, e criar um repositório com o nome "**xpand-music**".

Para criar um server Django e executar a aplicação deve executar o seguinte comando na pasta "webproj" do projeto:

```
python3 manage.py runserver
```

Junto com todo o código iremos enviar um dataset já preenchido com alguns dados. Basta fazer import no GraphDB Workbench ao ficheiro "**dataset.rdf**" e executar a aplicação.

Relembramos que a nossa aplicação pode ser executada com um repositório vazio, sendo que irá acrescentar dados à triplestore caso queira aceder a um artista/álbum que não exista na triplestore.