Trabalho Prático 2

Departamento de Eletrónica, Telecomunicações e Informática
Universidade de Aveiro
Web Semtântica 2022

Filipe Gonçalves
98083

Gonçalo Machado
98359

João Borges
98155

Afonso Campos
100055

May 30, 2023

1 Introdução

A World Wide Web revolucionou a maneira como as pessoas comunicam e partilham informação entre elas. No entanto, a enorme quantidade disponivel na Web deu origem a vários desafios, não só na sua gestão, mas também endender esta informação.

Para tentar resolver estes desafios, o campo da Web Semântica (WS) emergiu como uma forma de fazer a Web mais organizada e com mais significado. A WS envolve o uso de standards e tecnologias que dão estrutura e significado aos recursos da web, tornando mais eficiente a pesquisa, partilha e analise da informatica.

Neste trabalho prático, onde nos foi pedido para desenvolvermos um sistema de informação baseado na web, escolhemos utilizar informação sobre **anime**.

Anime, palavra japonesa derivada da palavra inglesa *animation*, é um tipo de animação feita à mão ou gerada por computadores com origem japonesa. Este estilo de animação tem vindo a ganhar cada vez mais popularidade ao longo do tempo, sendo que nos últimos anos, as grandes plataformas de streaming como Netflix e Disney+ aumentaram em larga escala a sua oferta de anime.

No desenvolvimento deste trabalho, foi utilizado:

- **Python/Django** - Programação da aplicação
- **RDF/NT/N3** - Formatação dos dados
- **Triplestore GraphDB** - Repositório de dados
- **SPARQL** - Pesquisa e alteração de dados na triplestore
- **RDFS/OWL** - Criação da ontologia

- **SPIN** - Conjunto de inferências
- **SPARQLwrapper** - Acesso ao endpoint da DBpedia e Wikidata;
- **RDFa/Micro-formatos** - Publicação da semântica

Durante o desenvolvimento deste sistema, tivemos como objetivos desenvolver um sistema:

- Com um maior nivel possivel de exploração e inter-relação entre as tecnologias mencionadas previamente
- Com uma interface fácil e intuitiva para o utilizador
- Com um dataset rico, com dados bastante relacionados entre si, e cmo uma boa exploração destas, e com a sua integração com dados da DBpedia e Wikidata
- Desenvolvido modularmente, separando os dados, a lógica e a apresentação
- Com uma ontologia completa, descrevendo exaustivamente o domínio de conhecimento dos dados
- Com regras de inferencia que permitam, não só o estabelecimento de novas relações entre as entidades, mas também que motore de inferência automática consigam gerar novas relaoes
- Com a publicação da semântica dos dados

2 A Ontologia - RDFS e OWL

Para o bom funcionamento de qualquer sistema de informação semântico, é necessário criar uma boa ontologia, para futuramente pudermos inferir novo ocnhecimento sobre os dados que anteriormente usamos.

Utiliamos o editor de ontologias open-source **Protégé** para pudermos testar a nossa ontologia, e perceber se as inferências criadas são as que tinhamos pensado, e quais inferências não foram criadas para pudermos usar SPIN.

Também foi necessário fazer tratamento aos nossos dados, já que havia inconsistências na criação dos triplos em relação à ontologia criada.

2.1 Entidades e Classes

Deste modo, começamos por, primeiramente, perceber se as entidades anteriormente encontradas eram as certas, se faltavam algumas ou se deviamos retirar outras.

Anteriormente, tinhamos identificado estas cinco entidades:

- Anime
- Voice Actors
- Characters
- Openings e Endings
- Opening Artists e Ending Artists

No entanto, conseguimos ver que por exemplo **Opening Artists** e **Ending Artists** podem ser acuplados numa só entidade **Singer**. Consequentemente, dizer que existem duas entidades, **Singer** e **Voice Actor**, implica que exista uma entidade superior **Person**.

Usando esta mesma lógica, podemos ter duas entidades diferentes **Opening** e **Ending**, com uma superentidade **Music**.

Com isto, conseguimos compactar as nossas entidades e, ao invés das antigas cinco, temos agora:

- Anime
- Character
- Person
  - Voice Actor
  - Singer
- Music
  - Opening
  - Ending

Contudo, se uma entidade for um **Anime**, por exemplo, também não pode ser um **Character**, ou qualquer outra entidade, e vice-versa. O mesmo acontece para as subentidades de **Person**: Um **Voice Actor** não pode ser um **Singer** e vice-versa. Por isso, estas entidades têm de ser disjuntas.

Assim, podemos criar as nossas classes, de acordo com cada uma das nossas entidades, criar as subclasses e as relações principais entre estas classes e, por fim, adicioná-las à nossa ontologia:

```
ent:Person rdf:type rdfs:Class ;
    rdfs:subClassOf owl:Thing ;
    owl:disjointWith ent:Character, ent:Anime, ent:Music .

ent:Singer rdf:type rdfs:Class ;
    rdfs:subClassOf ent:Person ;
    owl:disjointWith ent:VoiceActor .

ent:VoiceActor rdf:type rdfs:Class ;
    rdfs:subClassOf ent:Person ;
    owl:disjointWith ent:Singer .

ent:Music rdf:type rdfs:Class ;
    rdfs:subClassOf owl:Thing ;
    owl:disjointWith ent:Person, ent:Anime, ent:Character .

ent:Opening rdf:type rdfs:Class ;
    rdfs:subClassOf ent:Music ;
    owl:disjointWith ent:Ending .

ent:Ending rdf:type rdfs:Class ;
    rdfs:subClassOf ent:Music ;
    owl:disjointWith ent:Opening .

ent:Anime rdf:type rdfs:Class ;
    rdfs:subClassOf owl:Thing ;
    owl:disjointWith ent:Person, ent:Character, ent:Music .

ent:Character rdf:type rdfs:Class ;
    rdfs:subClassOf owl:Thing ;
    owl:disjointWith ent:Person, ent:Anime, ent:Music .
```

Usamos **rdfs:subClassOf** para criar as subclasses, **rdf:Class** para criar as classes, **owl:disjointWith** para criar as relações disjuntas e **owl:Thing** para adicioanr as nossas hiperclasses ao Universo de recursos.

2.2 Propriedades

Depois, tivemos de criar as nossas propriedades, que são todas subclasses de **rdf:Property**.

Tivemos de indicar o domínio e o range de cada uma, já que ao invés de serem propriedades de dados, como poderíamos ter definido usando **owl:DatatypeProperty**, são propriedades de objetos, que relacionam entidades com entidades, podendo utilizar **owl:ObjectProperty** para os identificar. Neste caso, decidimos não utilizar estas referências, e usarmos a simples **rdf:Property**.

Também criamos relações inversas de cada uma, para ser mais fácil de procurar por dados usando SPARQL.

```
pred:starring rdf:type rdf:Property ;
    owl:inverseOf pred:starred_at ;
    rdfs:domain ent:Anime ;
    rdfs:range ent:Character .

pred:played rdf:type rdf:Property ;
    owl:inverseOf pred:voiced_by ;
    rdfs:domain ent:VoiceActor ;
    rdfs:range ent:Character .

pred:opening rdf:type rdf:Property ;
    owl:inverseOf pred:op_played_in ;
    rdfs:domain ent:Anime ;
    rdfs:range ent:Opening .

pred:ending rdf:type rdf:Property ;
    owl:inverseOf pred:end_played_in ;
    rdfs:domain ent:Anime ;
    rdfs:range ent:Ending .

pred:voiced_at rdf:type rdf:Property ;
    owl:inverseOf pred:voiced_in ;
    rdfs:domain ent:Anime ;
    rdfs:range ent:VoiceActor .

pred:played_by rdf:type rdf:Property ;
    owl:inverseOf pred:sang ;
    rdfs:domain ent:Music ;
    rdfs:range ent:Singer .

pred:sequel rdf:type rdf:Property ;
    owl:inverseOf pred:prequel ;
    rdfs:domain ent:Anime ;
    rdfs:range ent:Anime .
```

```
pred:prequel rdf:type rdf:Property ;
    owl:inverseOf pred:sequel ;
    rdfs:domain ent:Anime ;
    rdfs:range ent:Anime .
```

Por fim, criamos as nossas propriedades de dados, como vemos a seguir dois exemplos:

```
pred:role rdf:type rdf:Property ;
    rdfs:domain ent:Character ;
    rdfs:range rdfs:Literal .

pred:title rdf:type rdf:Property ;
    rdfs:domain ent:Anime ;
    rdfs:range rdfs:Literal .
```

A diferença principal é que o range neste caso não é uma entidade mas um literal, e para o referenciar usamos **rdfs:Literal**.

3 Inferências - SPIN

Após obtermos os dados iniciais, optamos por expandi-los, tornando-os mais expressivos. Uma forma de o fazer foi recorrendo à inferência, isto é, uma forma de tornar relações antes **implícitas** em relações **explicitas** incluídas nos dados finais.

Para isto demos uso à técnica de SPIN (SPARQL Inference Notation) para gerar os novos dados atravéz daqueles já existentes. Com isto, isolámos dois novos tipos de relações:

- **pred:music_cast**, que representa a relação entre um Anime e um Singer quando o cantor está creditado como Singer de um Opening ou Ending associado ao Anime. O request SPARQL para esta inferência é o seguinte:

```
INSERT {
    ?anime pred:music_cast ?singer .
} WHERE {
    {
        ?anime pred:opening ?music .
        ?music pred:played_by ?singer .
    }
    UNION
    {
        ?anime pred:ending ?music .
        ?music pred:played_by ?singer .
    }
}
```

- pred:worked_with, que representa a relação entre dois Voice Actors quando estes representaram personagens num anime em comum. O SPARQL request:

```
INSERT {
    ?va1 pred:worked_with ?va2 .
} WHERE {
    {
        ?va1 pred:played ?character1 .
        ?anime pred:starring ?character1 .
        ?anime pred:starring ?character2 .
        ?va2 pred:played ?character2 .
        FILTER(?va1 != ?va2) .
    }
}
```

4 SPARQL

Em relação às queries que faziamos ao GrapshDB, elas foram mudadas de modo a terem uma maior segurança de que estavamos a procurar e a receber a informação certa.

Quando antes fazíamos:

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX pred:<http://anin3/pred/>
PREFIX ent:<http://anin3/ent/>
SELECT DISTINCT ?title ?rk
WHERE {
    ?anime pred:rank ?rk .
    ?anime pred:title ?title .
    FILTER ( xsd:integer(?rk) < xsd:integer("11") )
}
```

Agora fazemos:

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX pred:<http://anin3/pred/>
PREFIX ent:<http://anin3/ent/>
SELECT DISTINCT ?title ?rk
WHERE {
    ?anime a ent:Anime .
    ?anime pred:rank ?rk .
    ?anime pred:title ?title .
    FILTER ( xsd:integer(?rk) < xsd:integer("11") )
}
```

No entanto, algumas queries tiveram mudanças mais evidentes que estas:

- Anime por Título

  Antes, como pode ser visto a seguir, tinhamos primeiramente junto qualquer tipo de musica e agora separamos, também era feito um relacionamento entre **Character**, **VoiceActor** e **Anime** de acordo com as relaões que **Anime** tinha e agora é feito de acordo com as relações entre as entidades

  Antes faziamos a query assim:

```
PREFIX ent: <http://anin3/ent/>
PREFIX pred: <http://anin3/pred/>
SELECT *
WHERE {
    {
        ?anime pred:title "{title}" .
        ?anime ?pred ?object .
        FILTER (isliteral(?object))
    }
    UNION
    {
        ?anime pred:title "{title}" .
        ?anime ?pred ?object .
        ?object pred:name ?charname .
        ?object pred:role ?charrole .
        ?vc pred:played ?object .
        ?vc pred:name ?vcname .
    }
    UNION
    {
        ?anime pred:title "{title}" .
        ?anime ?pred ?object .
        ?object pred:name ?opname .
        ?object pred:played_by ?op .
        ?op pred:name ?opa .
    }
}
```

  E agora, é feita assim:

```
PREFIX ent: <http://anin3/ent/>
PREFIX pred: <http://anin3/pred/>
SELECT *
WHERE {
    {
        ?anime pred:title "{title}" .
        ?anime ?pred ?object .
```

```
        FILTER (isliteral(?object))
    }
    UNION
    {
        ?object a ent:Character .
        ?object pred:starred_at ?anime .
        ?anime a ent:Anime .
        ?anime pred:title "{title}" .
        ?object pred:char_name ?charname .
        ?object pred:role ?charrole .
        ?vc pred:played ?object .
        ?vc pred:va_name ?vcname .
        ?vc a ent:VoiceActor .
    }
    UNION
    {
        ?object a ent:Opening .
        ?object pred:op_played_in ?anime .
                ?anime a ent:Anime .
        ?anime pred:title "{title}" .
        ?object pred:op_name ?opname .
        ?object pred:played_by ?op .
        ?op a ent:Singer .
        ?op pred:sing_name ?opa .
    }
    UNION
    {
        ?object a ent:Ending .
        ?object pred:end_played_in ?anime .
                ?anime a ent:Anime .
        ?anime pred:title "{title}" .
        ?object pred:end_name ?endname .
        ?object pred:played_by ?end .
        ?end a ent:Singer .
        ?end pred:sing_name ?enda .
    }
}
```

- Pesquisa por Nome + Voice Actor por Nome

  A principal mudança nestas queries é a forma como organizamos a nossa pesquisa.

  Antes faziamos:

```
PREFIX ent: <http://anin3/ent/>
PREFIX pred: <http://anin3/pred/>
SELECT ?character_name ?role ?animename
WHERE {
    ?voice_actor pred:name "{nome}".
    ?voice_actor pred:played ?character .
    ?character pred:name ?character_name .
    ?character pred:role ?role .
    ?anime pred:starring ?character .
    ?anime pred:title ?animename .
}
```

  O que implica que para conseguirmos o Anime do qual aquele personagem participou do qual aquele Voice Actor deu voz, tinhamos de fazer back tracking para encontrar o Anime.

  Agora fazemos:

```
PREFIX ent: <http://anin3/ent/>
PREFIX pred: <http://anin3/pred/>
SELECT DISTINCT ?character_name ?role ?animename
WHERE {
    ?voice_actor pred:va_name "{nome}".
    ?voice_actor a ent:VoiceActor .
    ?voice_actor pred:played ?character .
    ?character a ent:Character .
    ?character pred:char_name ?character_name .
    ?character pred:role ?role .
    ?character pred:starred_at ?anime.
    ?anime a ent:Anime .
    ?anime pred:title ?animename .
} LIMIT 100
```

  Assim, conseguimos encontrar o Anime de acordo com a propriedade inversa criada de acordo com a nossa ontologia, sendo mais fácil a pesquisa.

Por fim, criamos também mais uma query, de acordo com o que achavamos que era bom mostrar ao utilizador do nosso website:

```
PREFIX ent: <http://anin3/ent/>
PREFIX pred: <http://anin3/pred/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?charname ?animename WHERE {
    ?char a ent:Character .
    ?char pred:char_name ?charname .
    ?char pred:starred_at ?anime .
    ?char pred:role ?role .
    ?anime a ent:Anime .
    ?anime pred:title ?animename .
    ?anime pred:rank ?rank .
    FILTER (?role = "Main")
} ORDER BY ASC(xsd:integer(?rank)) LIMIT 200
```

Com esta query, consegumos encontrar os 200 Characters, que contém o papél principal e em que Anime. Os animes são ordenados pelo seu ranking, por isso vamos conseguir os principais Characters dos principais Animes.

5 Wikidata e DBpedia

Na internet existem uma quantidade imensurável de dados, sendo que a maior parte destes dados estão espalhados e isolados uns dos outros, tornando a sua utilização e conexão com outros dados extremamente dificil ou até mesmo impossivel. Foi por este motivo que foram criados alguns projetos com o objetivo de pegar em dados e estruturá-los de maneira a tornar mais fácil o seu uso e relacionamento com outros dados.

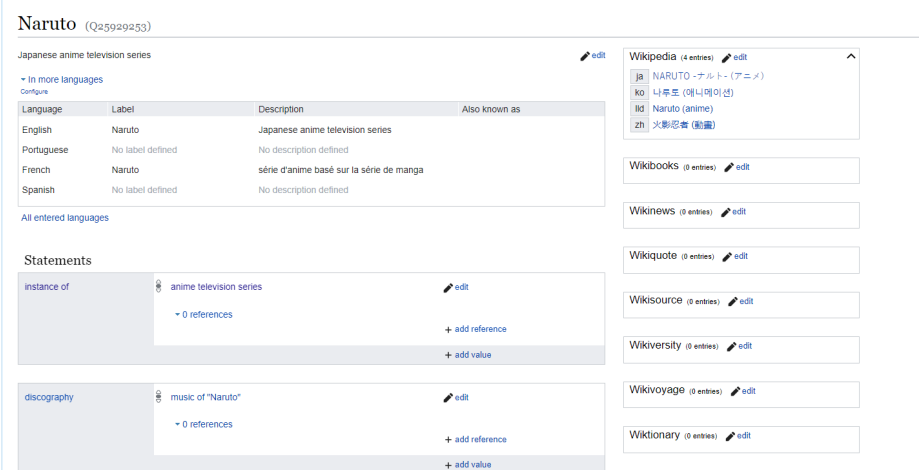
5.1 Wikidata

Wikidata é uma base de dados grátis, colaborativa e multilingue, onde dados estruturados são agregados e guardados para servir de suporte a outros projetos do movimento Wikimedia, como a Wikipedia, assim como a qualquer pessoa que queira utilizar estes dados.

A Wikidata organiza os seus dados em triplos RDF, pesquisáveis através de SPARQL, onde entidades, propriedades e objetos de um triplo são referidos utilizando códigos. Entidades são identificadas de forma única pela letra **Q** seguida de um número (Q11571 identifica a entidade "Cristiano Ronaldo"), e propriedades são identificadas utilizando a letra **P** seguida de um número (P22 identifica a propriedade "pai de").

5.1.1 Wikidata - Organização dos dados

Figure 1: Página da Wikidata da entidade Q25929253



Na figura 1 é possivel observar um exemplo de uma página da Wikidata, neste caso da entidade **Q25929253** que identifica o anime *Naruto*. Nesta página

podemos observar alguns elementos que existem em todas as páginas e que são fundamentais para a organização dos dados:

- **Item Identifier** - Como já foi mencionado anteriormente, um item ou entidade é identificado usando a letra *Q* seguida de um número. No exemplo a entidade tem o identificador Q25929253.
- **Label** - Consiste no nome mais comum dado à entidade. No exemplo a entidade tem a label *Naruto*.
- **Description** - É uma frase curta usada para descrever a entidade, assim como para distinguir entidades com labels iguais ou semelhantes. No exemplo a entidade tem a description "*Japanese anime television series*".
- **Alias** - São nomes alternativos dados à entidade. Ao contrário de uma label, uma entidade pode ter tantos alias como necessários.
- **Statements** - Nesta secção aparecem todos os triplos RDF onde a entidade atual é o sujeito do triplo. Uma entidade pode ter mais do que um objeto para a mesma propriedade. No exemplo, podemos ver que existe o triplo *Naruto - instance of - anime television series-*

5.1.2 Wikidata - Acesso e Integração

O tema do nosso projeto, **Anime**, tem origem japonesa, e embora a Wikidata seja um projeto multilingue, tem bastantes mais dados em inglês. Esta desproporção, assim como a falta de dados sobre os atores de voz e outras classes dos nossos dados, levou-nos a utilizar dados da Wikidata apenas na classe de **Anime**. O acesso à Wikidata foi feito utilizando a biblioteca **SPARQLwrapper** e através do endpoint "**https://query.wikidata.org/sparql**".

A query seguinte utiliza o titulo do anime que está presente no nosso dataset e retorna o identificador do anime na Wikidata, assim como triplos que tenham o anime como sujeito.

```
SELECT ?id ?pred_label ?sub_label
WHERE {
    ?id rdfs:label|skos:altLabel "{title}"@en.
    ?id p:P31 ?statement0.
    ?statement0 ps:P31 wd:Q63952888.
    ?id ?pred ?sub .
    ?pred2 wikibase:directClaim ?pred;
    rdfs:label ?pred_label.
    ?sub rdfs:label ?sub_label.
    FILTER(?pred = wdt:P57 || ?pred = wdt:P364 || ?pred = wdt:P750 || ?pred =
    FILTER(((LANG(?sub_label)) = "en") && ((LANG(?pred_label)) = "en"))
    SERVICE wikibase:label {{ bd:serviceParam wikibase:language "en". }}
}
```

Sendo uma query com alguma complexidade, abaixo encontra-se a explicação mais detalhada da clausula *WHERE* da query:

- Primeiramente, encontra-se todos os identificadores que possuem a variavel *title*, que contém o titulo do anime que queremos ir buscar dados sobre, como label ou alias.
- Para cada um dos identificadores, encontra-se todos os triplos que tenham a propriedade *P31* que significa "*instance of*" ou "instância de" e que tenham como objeto a entidade *Q63952888*, que é "anime television series", ou seja, pretende-se encontrar todas as entidades que são animes, que como o titulo de um anime é único ou deveria ser, apenas um identificador fica associado à variavel *?id*.
- Depois de termos o identificador da entidade pretendida, vamos buscar todos os triplos que tenham esta entidade como sujeito.
- Para os dados a apresentar na aplicação não serem códigos sem significado para seres humanos, é necessário a label das propriedades e dos objetos. Para obter as labels de uma propriedade é necessário utilizar a propriedade "*wikibase:directClaim*" que é utilizada para relacionar uma propriedade com uma entidade que representa essa propriedade, onde podem ser associados triplos, nomeadamente labels.
- Posteriromente procedemos à filtragem dos triplos para obtermos dados que não existem no nosso dataset, como:
  - P57 - Director
  - P364 - Original language of film or TV show
  - P750 - Distributed by
  - P8670 - Character designer
  - P272 - Production company
- Finalmente fazemos outra filtragem para apenas obtermos as labels das propriedades e objetos em inglês.

5.2 DBpedia

A DBpedia é um esforço comunitário baseado em crowdsource para a extração de informação criada em projetos da Wikimedia. A informação é estruturada na forma de um grafo aberto de conhecimento (OKG – Open Knowledge Graph). Um grafo de conhecimento é uma base de dados especial que guarda conhecimento de maneira a permitir a leitura por parte de máquinas e disponibiliza meios de coletar, organizar, pesquisar, partilhar e utilizar esse conhecimento.

5.2.1 DBpedia - Acesso e Integração

Tal como aconteceu com a Wikidata, existem poucos dados relativos à temática de animes, sendo que tambem optámos por apenas ir buscar dados relativos à classe **Anime**. O acesso à DBpedia foi feito utilizando a biblioteca **SPARQLwrapper** e através do endpoint "**https://dbpedia.org/sparql**".

Para ir buscar dados sobre um anime, são feitas duas queries.

Antes da primeira query, é necessário alterar o titulo do anime a pesquisar, colocando uma barra antes do titulo e substituindo todos os espaços por underscores. Ou seja, o titulo "Naruto" ficaria "/Naruto" e o titulo "Fullmetal Alchemist:Brotherhood" ficaria "/Fullmetal_Alchemist:Brotherhood".

A primeira query, que é possivel ver abaixo, vai buscar todos os identificadores que são do tipo Anime e que contêm o titulo alterado do anime do qual queremos dados no identificador.

Existem bastantes recursos do tipo Anime que não possuem label, e para aumentar a utilidade do acesso à DBpedia, decidimos fazer a pesquisa é feita no identificador do recurso.

```
SELECT * WHERE {
    ?res a dbo:Anime
    FILTER regex(?res, "{title}", "i")
}
```

Não é possivel fazer apenas uma query pois na DBpedia não são utilizados códigos para simbolizar os recursos, mas sim um URI onde uma parte está associada ao nome mais comum do recurso. Por exemplo, o URI do recurso que representa o anime "*Naruto*" é *http://dbpedia.org/resource/Naruto__tv_serie__1*. Esta organização mais o facto de que existem animes com nomes iguais ou que possuem parte do nome de outro anime, é necessário fazer uma filtragem para garantir que temos o anime certo.

Outra razão da necessidade de duas queries é que como um anime é normalmente baseado num *manga* com o mesmo nome, o URI do anime possui "*__tv_serie__1*" para distinguir o manga do anime. Por este motivo, antes de irmos buscar informações sobre o anime, é necessário garantir que temos o identificador do anime e não do manga do qual é baseado.

Após a confirmação que temos o identificador do anime, basta utilizar esse identificador e ir buscar as labels em inglês dos objetos de triplos onde o identificador (que na query está na constante *dbpedia_uri*) é um sujeito e a propriedade é *dbo:network*. Foi escolhida apenas esta propriedade pois foi a única propriedade encontrada que aparecia em bastantes recursos na DBpedia e que não estava no dataset nem vinha da Wikidata.

```
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT distinct ?sub_label ?pred_label
WHERE {
    <{dbpedia_uri}> ?property ?value .
    ?property rdfs:label ?pred_label .
    ?value rdfs:label ?sub_label .
    FILTER(?property = dbo:network)
    FILTER(lang(?pred_label) = "en"
           && lang(?sub_label) = "en")
}
```

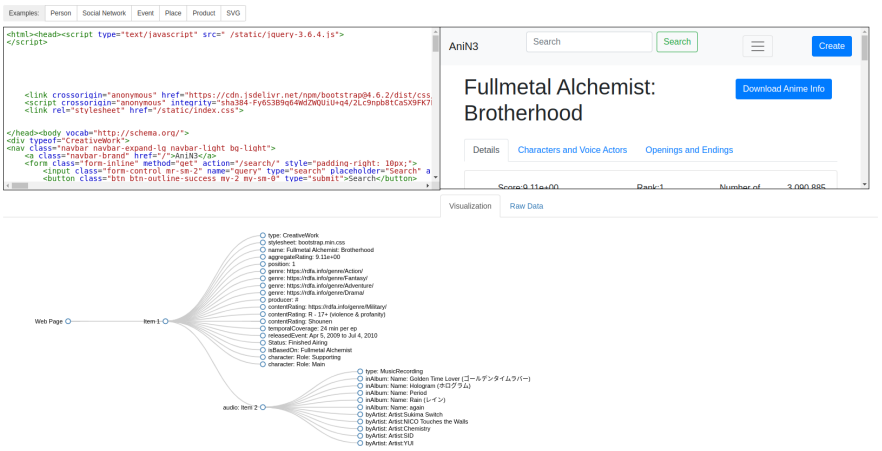
6 Publicação da semântica dos dados - RDFa

RDFA permite fazer markups de elementos numa página web, para que motores de busca e serviços web consigam gerar melhores resultados de pesquisa e mais adequados áquilo que foi requesitado, dando assim mais visibilidade á página. Na nossa implementação, nós utilizamos um schema generalista do site schema.org, chamado CreativeWork (através do campo typeof), ao qual seguidamente identificamos a informação relevante com o campo property, como demonstra o seguinte código:

```
<!DOCTYPE html>
<html lang="en">
  <body vocab="http://schema.org/" typeof="CreativeWork">
    <p property="Title">
      Anime title
    </p>
  </body>
</html>
```

Verificamos se a implementação se encontra correta através do site .

Figure 2: Página de verificação de RDFa



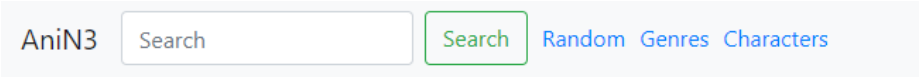
Esta implementação é relativamente simples mas suficiente para a melhora de identificação e publicação de dados na web, sendo que assim, se um determinado utilizador pesquisar, por exemplo, por género comédia no seu motor de busca, ele terá a informação que este site possui exatamente essa informação.

7 Funcionalidades da Aplicação

Visto que este trabalho é uma continuação do primeiro trabalho prático, existem partes que, embora tenham mudado o funcionamento, a interface com o utilizador continua a mesma. Por esse motivo, apenas iremos mostrar as partes da UI que foram alteradas ou adicionadas.

7.1 Barra de Navegação

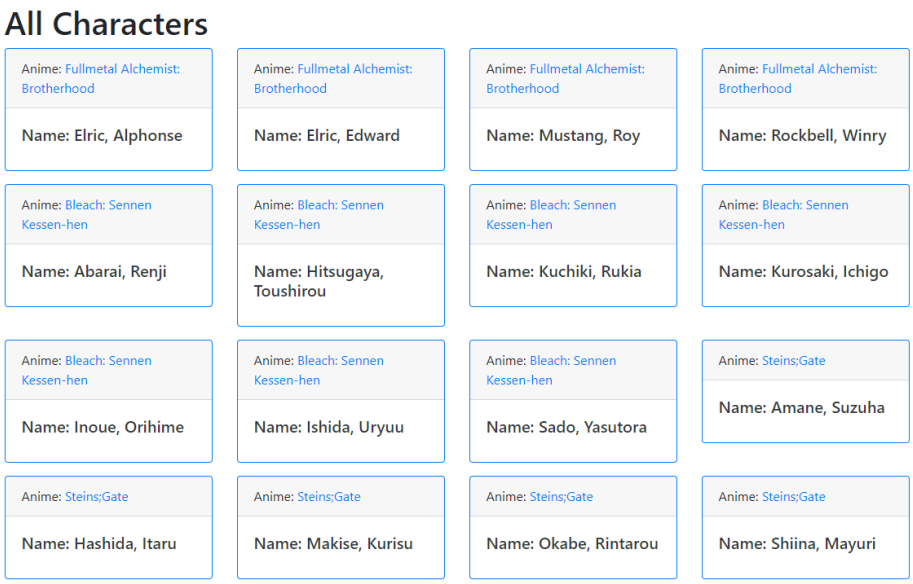
Figure 3: Barra de navegação



Foi adicionado um novo elemento à barra de navegação: **Characters**. Este novo elemento quando clicado redireciona o utilizador para uma nova página, que está explicada na subseccção *Characters*

7.2 Characters

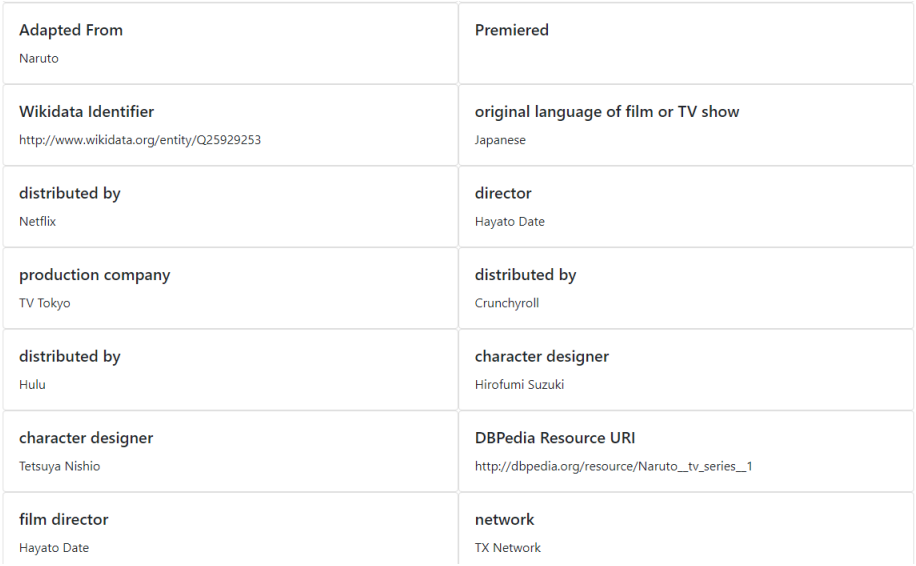
Figure 4: Página das personagens



Nesta página, estão presentes as personagens principais dos animes com rank mais alto. Como podemos ver na figura 5, em cada cartão está o nome do personagem, assim como o nome do anime ao qual esse personagem pertence, sendo que clicar no nome do anime redireciona o utilizador para a página com os detalhes do anime.

7.3 Anime Details

Figure 5: Parte da página com os detalhes do anime "Naruto"



Esta página, que já existia, foi melhorada para incluir dados tanto da Wikidata como da DBpedia, sendo que não existe distinção entre os dados que estão no dataset, os dados que vêm da Wikidata e os dados que vêm da DBpedia. Apenas são mostrados dados de fontes externas caso eles existam, sendo que como foi mencionado previamente, muitas vezes não existem.

Embora não exista uma distinção visivel entre os dados vindos de fontes internas e externas, os dados que vêm da Wikidata começam com o cartão "Wikidata Identifier" e acabam imediatamente antes do cartão "DBPedia Resource URI" caso exista, ou caso não exista até ao fim da página. Os dados que vêm da DBpedia, caso existam, começam no cartão "DBPedia Resource URI" e vão até ao fim da página.

8 Conclusão

Neste trabalho expandimos a expressividade do nosso modelo semântico através da criação de uma ontologia. Ao inserir regras de consistência tornamos o nosso modelo mais robusto, complementamndo com regras formais de inferência.

Fomos também mais longe, enriquecendo os nossos dados com triplos anteriormente implícitos na rede, tornando-os explícitos através de SPIN.

Aprofundámos o funcionamento das queries SPARQL anteriormente definidas após a definição da ontologia de modo a torná-las mais precisas nas entidades e relações que retornam e enriquecemos o modelo semântico com dados tanto da DBpedia como da Wikidata.

Finalmente, anotamos as nossas páginas HTML com RDFa, possibilitando assim a publicação dos dados que temos vindo a recolher.

Com tudo isto, diríamos que este trabalho cumpre os requisitos pedidos para o sistema, e achamos que o nosso sistema mostra uma boa integração das tecnologias abordadas.

Em conclusão, queremos novamente dizer que foi um projeto interessante que expandiu os nossos conhecimentos adquiridos no trabalho anterior e que também nos proporcionou com conhecimentos abrangentes sobre a disciplina da Web Semântica e sobre a utilidade da semântica no futuro.

Pode encontrar o nosso repositório de github, com o código do projeto em FlipGoncalves/WebSemanticaTrabalho1[1].

9 Configurações para executar a aplicação

Para executar a aplicação, deve seguir os seguintes passos:

- GraphDB

  Iniciar o GraphDB no seu computador em http://localhost:7200/ e certificar-se que não tem um repositório chamado "anin3" inicialmente.

- Django Application

  Para iniciar a aplicação django, basta executar os seguintes comandos:

```
cd path/to/project/folder/
cd WSProject1/
pip install -r requirements.txt
python3 manage.py runserver
```

De maneira a automatizar a utilização da nossa aplicação, a criação do repositório e posterior inserção de dados neste é feita automaticamente caso o repositório ainda não tenha sido criado. O código que foi escrito para automatizar este processo encontra-se no ficheiro *apps.py*, localizado na pasta WebSemanticaTrabalho1/WSProject1/app, e é corrido uma vez quando a aplicação é inicializada. O funcionamento do código é o seguinte:

- O sistema envia um GET request ao GraphDB para verificar se o repositório já existe. Se o repositório já existir, é utilizado o repositório existente, não havendo assim problemas de terminar e inicializar a aplicação várias vezes.
- No caso do repositório não existir, é necessário ir buscar os caminhos absolutos para o ficheiro de configuração do repositório, **anin3-config.ttl**, e para o ficheiro com os dados que serão inseridos no repositório, **animes.nt**.
- O caminho absoluto do ficheiro com os dados é escrito no ficheiro de configuração.
- É feito um POST request ao GraphDB com o ficheiro de configuração como argumento, sendo este o request que cria e insere os dados automaticamente.

References

[1] Filipe Gonçalves (98083) - Gonçalo Machado (98359) - João Borges (98155) - Afonso Campos (100055). *WebSemanticaTrabalho1*. URL: https://github.com/FlipGoncalves/WebSemanticaTrabalho1.