



Trabalho Prático 2

Web Semântica

Docente Hélder Zagalo
2022/2023

Mestrado em Engenharia Informática

Inês Leite, 92928
Renan Ferreira, 93168
Tiago Coelho, 98385

Índice

1. Introdução	2
2. Definição de Ontologia	3
3. Conjunto de Inferências	6
4. Novas operações sobre os dados	8
5. Integração de dados do WikiData e DBpedia	11
6. RDFa e Microformatos	15
6.1. RDFa	15
6.2. Microformatos	16
6.3. Conclusão	17
7. Funcionalidades da Aplicação	18
8. Conclusões	21
9. Configuração para executar a aplicação	22
10. Referências	23

1. Introdução

O trabalho prático proposto pelo docente da Unidade Curricular Web Semântica é a continuação do desenvolvimento de um sistema de informação baseado na web adicionando certas funcionalidades.

O objetivo deste projeto é a exposição e gestão de toda a informação constante no sistema, utilizando Python/Django para a programação da aplicação, RDF como formato dos dados usados pela mesma, *Triplestore* GraphDB como repositório desses mesmos dados, SPARQL para a pesquisa e alteração dos dados no repositório, RDFS e OWL para a ontologia, SPIN para ter um conjunto de inferências, SPARQLwrapper para o acesso ao endpoint da DBpedia e Wikidata e, por último, RDFa e Micro-formatos para a publicação da semântica.

Para desenvolver o projeto descrito, foi utilizado um dataset¹ retirado do Kaggle em formato CSV que contém cerca de 11 mil livros existentes no site GoodReads (Soumik 2020).

¹ Dataset *Goodreads-books*: <https://www.kaggle.com/datasets/jealousleopard/goodreadsbooks>

2. Definição de Ontologia

Como referido anteriormente, os dados foram retirados do Kaggle (Soumik 2020) e possuem cerca de 11 mil livros presentes no site GoodReads. Este *dataset* encontra-se no formato CSV e foi criado usando o GoodReads API e foi atualizado até 8 de dezembro de 2020.

O *dataset* possui diversas informações diferentes para cada livro, desde título, autores, classificação (do site GoodReads), número de avaliações total e escritas, isbn, língua e número de páginas. Posteriormente, adicionámos outras informações, uma delas durante a conversão para N-Triples, e outra através de *queries*, sendo, respetivamente, se o utilizador já leu ou não aquele livro e categorias.

A primeira informação adicionada referente a se o utilizador já leu ou não um certo livro, visa a melhorar os dados e adicionar uma nova funcionalidade à aplicação podendo assim o utilizador marcar e guardar os livros que já leu, explorando também o *update* de dados do repositório. Esta informação foi adicionada durante a conversão (Figura 1) e posteriormente passada para *boolean* através de uma *query* (Figura 2). Esta funcionalidade teve como objetivo a exploração do *update* de dados, e no caso de a aplicação ter um rumo não educativo, seria necessário este *update* ser conjugado com o perfil do utilizador, havendo a necessidade de termos outros recursos, tais como o *login*.

Na criação da ontologia, houve conversão do tipo principal das principais entidades do sistema, o *book*, o *author* e o *publisher*, para tipos específicos do sistema, no caso *book:Book*, *author:Author* e *publisher:Publisher*, respetivamente. Os tipos também foram padronizados para definir seus respectivos domínios e valores. A grande adição do modelo foi converter a propriedade dos livros *has_genre* para a subclasse de livro, através de inferências, que serão melhor descritas na próxima secção. O modelo final ficou como está apresentado na Figura 3.

```
# add row to seen or unseen books put all as unseen
output.write(book.n3()+" "+seen.n3()+" "+Literal("unseen",datatype=XSD.string).n3()+"\n")
```

Figura 1 - Inserção de informação na conversão

```
query = """
PREFIX books: <http://books.com/books/>
PREFIX pred: <http://books.com/preds/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
DELETE {
  ?book pred:has_seen_by ?seen .
}
INSERT {
  ?book pred:has_seen "false"^^xsd:boolean .
}
WHERE {
  ?book pred:has_seen ?seen .
}
"""
```

Figura 2 - Query para passar para *boolean*

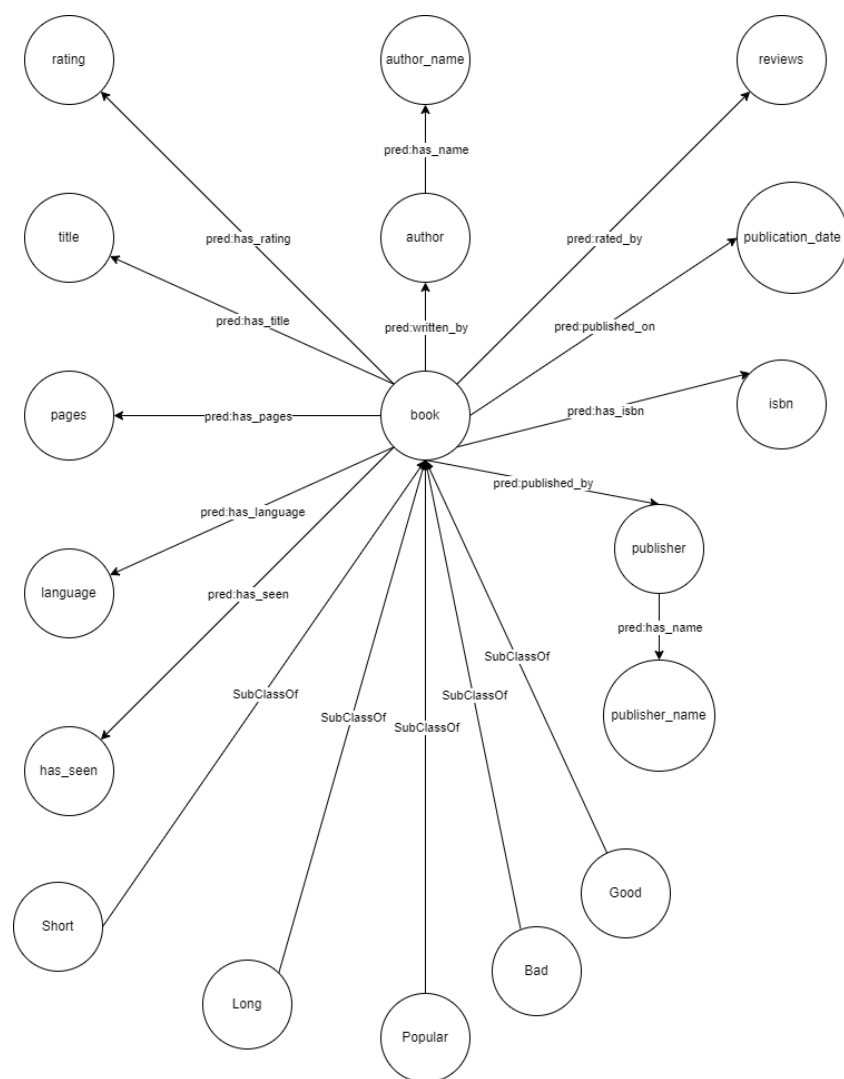


Figura 3 - Diagrama do modelo de dados.

Para a conversão dos dados de CSV para N-Triples foi usado o ficheiro “csv_to_nt.py” criado por nós que usa o rdflib.

O *script* começa por definir os *namespaces* e predicados que serão usados no grafo RDF. De seguida, o documento CSV é lido e para cada linha, um nó é adicionado ao grafo RDF com as informações da linha. Adicionando informações sobre o livro, como título, ISBN, número de páginas, idioma, data de publicação, autores e editora do livro. Por fim, o código adiciona a informação sobre o livro ter sido lido ou não pelo utilizador.

O ficheiro “books.csv” é o ficheiro retirado do Kaggle, “books.nt” é o ficheiro gerado pelo script e “new.nt” foi o novo ficheiro resultante após a adição e alteração de informações através das *queries*.

Para este projeto, o programa foi modificado de acordo com os requisitos da Ontologia num *script ad-hoc*. Os resultados dos N-Triples foram passados, juntamente com a ontologia no ficheiro “ontology.nt” para o Protegé, que criou o ficheiro final a passar a base de dados, “results.ttl”.

```

author_id = 0
book_id = 0
publisher_id = 0
authors_l = []
publishers_l = []

#URI
base = Namespace("http://books.com/")
authors = Namespace("http://books.com/authors/")
books = Namespace("http://books.com/books/")
publishers = Namespace("http://books.com/publishers/")
predicate = Namespace("http://books.com/preds/")

#Predicates
title = predicate.has_title
isbn = predicate.has_isbn
pages = predicate.has_pages
language = predicate.has_language
published_on = predicate.published_on
rating = predicate.has_rating
rated_by = predicate.rated_by
written_by = predicate.written_by
published_by = predicate.published_by
pages = predicate.has_pages
name= predicate.has_name # for authors and publishers
seen= predicate.has_seen # for website users

with open('books.csv','r') as input:
    with open('books.nt','w') as output:
        reader = csv.DictReader(input)
        for row in reader:
            print(row)
            book = books[str(book_id)]
            output.write(book.n3()+ " "+RDF.type.n3()+ " "+FOAF.Document.n3()+".\n")
            output.write(book.n3()+ " "+title.n3()+ " "+Literal(row['title'],datatype=XSD.string).n3()+".\n")
            output.write(book.n3()+ " "+isbn.n3()+ " "+Literal(row['isbn'],datatype=XSD.string).n3()+".\n")
            output.write(book.n3()+ " "+pages.n3()+ " "+Literal(row['num_pages'],datatype=XSD.integer).n3()+".\n")
            output.write(book.n3()+ " "+language.n3()+ " "+Literal(row['language_code'],datatype=XSD.language).n3()+".\n")
            output.write(book.n3()+ " "+rating.n3()+ " "+Literal(row['average_rating'],datatype=XSD.float).n3()+".\n")
            output.write(book.n3()+ " "+rated_by.n3()+ " "+Literal(row['ratings_count'],datatype=XSD.integer).n3()+".\n")

            #convert date to datetime
            date = datetime.strptime(row['publication_date'], '%m/%d/%Y')
            output.write(book.n3()+ " "+published_on.n3()+ " "+Literal(date,datatype=XSD.date).n3()+".\n")

            for author in row['authors'].split('/'):
                if author not in authors_l:
                    authorm = authors[str(author_id)]
                    output.write(authorm.n3()+ " "+RDF.type.n3()+ " "+FOAF.Person.n3()+".\n")
                    output.write(authorm.n3()+ " "+name.n3()+ " "+Literal(author,datatype=XSD.string).n3()+".\n")
                    authors_l.append(author)
                    author_id += 1
                    output.write(book.n3()+ " "+written_by.n3()+ " "+authorm.n3()+".\n")
                else:
                    output.write(book.n3()+ " "+written_by.n3()+ " "+authors[str(authors_l.index(author))].n3()+".\n")

            if row['publisher'] not in publishers_l:
                publisher = publishers[str(publisher_id)]
                output.write(publisher.n3()+ " "+RDF.type.n3()+ " "+FOAF.Organization.n3()+".\n")
                output.write(publisher.n3()+ " "+name.n3()+ " "+Literal(row['publisher'],datatype=XSD.string).n3()+".\n")
                publishers_l.append(row['publisher'])
                publisher_id += 1
            else:
                publisher = publishers[str(publishers_l.index(row['publisher']))]
                output.write(book.n3()+ " "+published_by.n3()+ " "+publisher.n3()+".\n")

            # add row to seen or unseen books put all as unseen
            output.write(book.n3()+ " "+seen.n3()+ " "+Literal("unseen",datatype=XSD.string).n3()+".\n")
            book_id += 1

```

Figura 4 - Código conversão do ficheiro csv para N-Triples

3. Conjunto de Inferências

As Inferências foram feitas com recurso ao *SPIN*, do SPARQL, e foram usadas para classificar os livros nas subclasses “*long*”, “*short*”, “*good*”, “*bad*” e “*popular*”, tendo em conta as diversas informações retiradas dos mesmos, tais como número de páginas (para “*long*” e “*short*”), classificação (para “*good*” e “*bad*”) e o número de *reviews* (para “*popular*”). Estas inferências foram adicionadas através de vários comandos, tendo em conta os diversos parâmetros e podendo cada livro ter mais de uma categoria, e sendo criadas no momento que é requisitada informação do sistema. Os comandos usados para criar as diferentes classes encontram-se nas Figuras seguintes, da 5 à 9.

```
create_short_books = """
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX book: <http://books.com/books/>
PREFIX bookp: <http://books.com/preds/>

INSERT {
    ?X rdf:type book:Short .
}
WHERE {
    ?X rdf:type book:Book .
    ?X bookp:has_pages ?N .
    FILTER (?N < 1000)
}

"""
```

Figura 5 - Categoria “Short”

```
create_good_books = """
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX book: <http://books.com/books/>
PREFIX bookp: <http://books.com/preds/>

INSERT {
    ?X rdf:type book:Good .
}
WHERE {
    ?X rdf:type book:Book .
    ?X bookp:has_rating ?N .
    FILTER (?N >= 4)
}

"""
```

Figura 6 - Categoria “Good”

```

create_long_books = """
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX book: <http://books.com/books/>
PREFIX bookp: <http://books.com/preds/>

INSERT {
    ?X rdf:type book:Long .
}
WHERE {
    ?X rdf:type book:Book .
    ?X bookp:has_pages ?N .
    FILTER (?N >= 1000)
}

"""

```

Figura 7 - Categoria "Long"

```

create_popular_books = """
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX book: <http://books.com/books/>
PREFIX bookp: <http://books.com/preds/>

INSERT {
    ?X rdf:type book:Popular .
}
WHERE {
    ?X rdf:type book:Book .
    ?X bookp:rated_by ?N .
    FILTER (?N >= 10000)
}

"""

```

Figura 8 - Categoria "Popular"

```

create_bad_books = """
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX book: <http://books.com/books/>
PREFIX bookp: <http://books.com/preds/>

INSERT {
    ?X rdf:type book:Bad .
}
WHERE {
    ?X rdf:type book:Book .
    ?X bookp:has_rating ?N .
    FILTER (?N < 4)
}

"""

```

Figura 9 - Categoria "Bad"

4. Novas operações sobre os dados

Ao longo do projeto, foram realizadas algumas alterações nas queries com o objetivo de aprimorar as operações sobre os dados. Essas alterações foram fundamentais para melhorar a eficiência e a precisão das consultas. Abaixo, detalhamos cada modificação realizada e justificamos os procedimentos adotados:

- **Remoção da propriedade "has_genre" nas queries:** Anteriormente, as queries eram baseadas na propriedade "has_genre" para obter o gênero dos livros. No entanto, essa abordagem foi substituída pela inferência de gênero a partir da propriedade "rdf:type". Dessa forma, tornou-se possível obter o gênero dos livros de forma mais eficiente, aproveitando a estrutura RDF existente. Além disso, essa alteração permitiu simplificar as consultas, removendo a dependência da propriedade "has_genre".

```
SELECT DISTINCT ?title ?author_name ?pages ?genre ?rating ?reviews ?has_seen ?language ?publisher_name ?publication_date ?isbn
WHERE {
  ?book pred:has_title ?title .
  ?book pred:writen_by ?author .
  ?author pred:has_name ?author_name .
  ?book pred:has_pages ?pages .
  ?book pred:has_rating ?rating .
  ?book pred:rated_by ?reviews .
  ?book pred:has_seen ?has_seen .
  ?book pred:has_language ?language .
  ?book pred:published_by ?publisher .
  ?publisher pred:has_name ?publisher_name .
  ?book pred:published_on ?publication_date .
  ?book pred:has_isbn ?isbn .
  ?book rdf:type books:Popular .
```

Figura 10 - Remoção do *has_genre*

- **Adição de cláusula "OPTIONAL" para inferir o gênero dos livros:** Para incorporar a inferência de gênero, foi utilizada uma cláusula "OPTIONAL" nas queries. Por meio dessa cláusula, foi possível incluir as regras de inferência para cada tipo de gênero ("Long", "Short", "Good", "Bad", "Popular") com base no *rdf:type* dos livros. Essa abordagem proporcionou uma maneira flexível e dinâmica de obter o gênero inferido dos livros, sem a necessidade de incluir explicitamente a propriedade "has_genre" nas consultas.

```
# Include inferred genres based on rules
OPTIONAL {
  {
    ?book rdf:type books:Long .
    BIND("Long" AS ?genre)
  }
  UNION
  {
    ?book rdf:type books:Short .
    BIND("Short" AS ?genre)
  }
  UNION
  {
    ?book rdf:type books:Good .
    BIND("Good" AS ?genre)
  }
  UNION
  {
    ?book rdf:type books:Bad .
    BIND("Bad" AS ?genre)
  }
  UNION
  {
    ?book rdf:type books:Popular .
    BIND("Popular" AS ?genre)
  }
}
```

Figura 11- Cláusula OPTIONAL

- **Alteração das queries de contagem de livros:** As queries que retornam o número de livros longos, curtos, bons, maus e populares foram modificadas para refletir a nova estrutura RDF do projeto. Agora, a contagem é realizada com base na contagem de instâncias das classes

correspondentes. Essa abordagem permite obter o número de livros de forma mais precisa e eficiente, aproveitando a estrutura RDF que foi definida.

```
# Get the number of short books (less than 1000 pages)
nShortBooks = ""
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX books: <http://books.com/books/>
PREFIX pred: <http://books.com/preds/>
SELECT (COUNT(?book) AS ?count)
WHERE {
    ?book rdf:type books:Short .
}
```

Figura 12 - Query que retorna número de livros curtos

- **Modificação da função "get_books_by_author":** A função "get_books_by_author" foi ajustada para se adequar às alterações nas queries. Agora, utiliza a nova estrutura RDF e inclui a inferência de gênero dos livros. Isso significa que, ao buscar os livros de um determinado autor, os resultados serão filtrados corretamente com base no gênero inferido de cada livro. Essa modificação permite apresentar resultados mais relevantes e alinhados com as intenções do usuário.

```
# Get books from author
getBooksByAuthor = ""
PREFIX books: <http://books.com/books/>
PREFIX pred: <http://books.com/preds/>

SELECT ?title ?author_name ?pages ?genre ?rating ?reviews ?has_seen ?language
WHERE {
    # Subquery to find all books written by the author
    {
        SELECT ?book
        WHERE {
            ?book pred:written by ?author .
            ?author pred:has_name "replace" .
        }
    }

    # Retrieve the details of the books and their co-authors
    ?book pred:has_title ?title .
    ?book pred:written by ?author .
    ?author pred:has_name ?author_name .
    ?book pred:has_pages ?pages .
    ?book pred:has_rating ?rating .
    ?book pred:rated by ?reviews .
    ?book pred:has_seen ?has_seen .
    ?book pred:has_language ?language .
    ?book pred:published by ?publisher .
    ?publisher pred:has_name ?publisher_name .
    ?book pred:published on ?publication_date .
    ?book pred:has_isbn ?isbn .

    OPTIONAL {
        {
            ?book rdf:type books:Long .
            BIND("Long" AS ?genre)
        }
        UNION
        {
            ?book rdf:type books:Short .
            BIND("Short" AS ?genre)
        }
        UNION
        {
            ?book rdf:type books:Good .
            BIND("Good" AS ?genre)
        }
        UNION
        {
            ?book rdf:type books:Bad .
            BIND("Bad" AS ?genre)
        }
        UNION
        {
            ?book rdf:type books:Popular .
            BIND("Popular" AS ?genre)
        }
    }
}
```

Figura 13 - Query getBooksByAuthor

- **Tentativa de adicionar a popularidade do autor:** Foi realizada uma tentativa de adicionar a informação de popularidade do autor. No entanto, devido à quantidade de dados e ao desempenho das queries necessárias para percorrer todos os livros e autores a fim de obter o número de livros de cada autor, essa implementação não foi concluída. As queries demoravam um tempo de execução bastante elevado devido à complexidade e ao volume de informações a serem processadas. Portanto, optou-se por não incluir a informação de popularidade do autor neste momento.

```

# Get author popularity
getAuthorPopularity = """
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX book: <http://books.com/books/>
PREFIX bookp: <http://books.com/preds/>

SELECT ?popularity
WHERE {
    ?author rdf:type book:Author ;
            bookp:has_name "replace" ;
            rdf:type ?popularity .

    FILTER (?popularity IN (book:Unpopular, book:Popular, book:VeryPopular))
}
"""

```

Figura 14 - Query getAuthorPopularity

```

# Create popularity for authors
# if author has less than 5 books, then unpopular
# if author has more than 5 books, and less than 15, then popular
# if author has more than 15 books, then very popular
create_popularity_author = """
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX book: <http://books.com/books/>
PREFIX bookp: <http://books.com/preds/>

INSERT {
    ?X rdf:type book:Unpopular .
}
WHERE {
    ?X rdf:type book:Author .
    ?X bookp:has_books ?N .
    FILTER (?N < 5)
}

INSERT {
    ?X rdf:type book:Popular .
}
WHERE {
    ?X rdf:type book:Author .
    ?X bookp:has_books ?N .
    FILTER (?N >= 5 && ?N < 15)
}

INSERT {
    ?X rdf:type book:VeryPopular .
}
WHERE {
    ?X rdf:type book:Author .
    ?X bookp:has_books ?N .
    FILTER (?N >= 15)
}
"""

```

Figura 15 - Query create_popularity_author

5. Integração de dados do WikiData e DBpedia

Neste capítulo, abordaremos a integração dos dados provenientes do WikiData e DBpedia como parte do processo de enriquecimento do conjunto de dados do nosso projeto.

Essa integração foi realizada por meio do acesso programático aos endpoints SPARQL disponibilizados por essas plataformas. Um dos principais objetivos do projeto era complementar os dados existentes com informações adicionais, como a capa do livro e a imagem do autor, por forma a enriquecer a experiência do utilizador na nossa plataforma.

Para isso, desenvolvemos algumas funções que nos permitiram acessar e extrair essas informações das fontes relevantes. Uma dessas funções é a "get_author" (Figura 15), que recebe como parâmetro o nome de um autor e retorna um dicionário contendo o nome do autor, a lista de livros associados a ele e, se disponível, a imagem do autor. Essa função utiliza a biblioteca SPARQLWrapper para construir e executar consultas SPARQL tanto no WikiData quanto no DBpedia. Caso não seja possível obter a imagem do autor nessas fontes, a função utiliza a API do Google Books como alternativa. Optámos por adicionar esta última opção visto tanto a Wikidata como a DBpedia possuírem muitas poucas fotos de autores, e no caso da wikidata, retornava muitas vezes falsos positivos, pois retornava resultados com link para imagem, porém os links não funcionam, retornando "value not found".

```
def get_author(self, author):  
    string = str(author)  
    query = self.getBooksByAuthor.replace("replace", string)  
    author = dict()  
    author['author_name'] = string  
    author['books'] = self.get_books(query)  
    if len(author['books']) > 0:  
        author['author_image'] = self.get_author_image(author['author_name'])  
    return author
```

Figura 15 - Função get_author

Outra função importante é a "get_author_image" (Figura 16), que é a função chamada pela função anterior e recebe o nome de um autor como parâmetro e busca a imagem do autor no WikiData e DBpedia. Através do SPARQLWrapper, a função constrói e executa consultas SPARQL para obter a imagem. Em caso de falha, a função utiliza a API do Google Books para obter a imagem do autor.

```

def get_author_image(self, author_name):
    sparql = SPARQLWrapper("https://query.wikidata.org/sparql")
    sparql.setQuery(f"""
    SELECT ?image
    WHERE {{
        ?author wdt:P31 wd:Q5 .
        ?author wdt:P18 ?image .
        ?author rdfs:label "{author_name}"@en .
    }}
    """)
    sparql.setReturnFormat(JSON)
    # check if endpoint is up
    try:
        results = sparql.query().convert()

        if len(results["results"]["bindings"]) > 0:
            return results["results"]["bindings"][0]["image"]["value"]
    except:

        sparql = SPARQLWrapper("http://dbpedia.org/sparql")
        sparql.setQuery(f"""
        SELECT ?image
        WHERE {{
            ?author rdf:type dbo:Person .
            ?author dbo:thumbnail ?image .
            ?author rdfs:label "{author_name}"@en .
        }}
        """)
        sparql.setReturnFormat(JSON)
        try:
            results = sparql.query().convert()
            if len(results["results"]["bindings"]) > 0:
                return results["results"]["bindings"][0]["image"]["value"]
        except:

            url = f"https://www.googleapis.com/books/v1/volumes?q={author_name}&maxResults=1"
            response = requests.get(url)
            data = response.json()
            if "items" in data and data["items"]:
                volume_info = data["items"][0]["volumeInfo"]
                if "imageLinks" in volume_info and "thumbnail" in volume_info["imageLinks"]:
                    return volume_info["imageLinks"]["thumbnail"]
            return None

```

Figura 16 - Função get_book_genre

Além disso, desenvolvemos a função "get_book_image" (Figura 17) para obter a capa do livro. Essa função utiliza apenas a API do Google Books para realizar uma busca pelo título do livro e obter a imagem da capa, caso esteja disponível. No caso do get_book image, colocámos apenas a API da google, visto a wikidata encontrar constantemente falsos positivos fazendo com que nunca houvesse imagem.

```
def get_book_image(self, book_title):
    url = f"https://www.googleapis.com/books/v1/volumes?q={book_title}&maxResults=1"
    response = requests.get(url)
    data = response.json()

    if "items" in data and data["items"]:
        volume_info = data["items"][0]["volumeInfo"]
        if "imageLinks" in volume_info and "thumbnail" in volume_info["imageLinks"]:
            return volume_info["imageLinks"]["thumbnail"]

    return None
```

Figura 17 - Função get_book_image

Essas funções permitiram enriquecer o conjunto de dados do projeto com novas informações, como a capa do livro e a imagem do autor, utilizando como fontes de dados o WikiData, o DBpedia e a API do Google Books. Exploramos também outros dados, como gênero, idade apropriada e descrição do autor, que poderiam ser aproveitados. No entanto, essas informações estavam disponíveis apenas para alguns livros, o que nos levou a optar por não adicioná-las ao site. Contudo, no repositório encontram-se os diversos testes na pasta “wikidata”.

```
def get_book_genre(self, book_title):
    sparql = SPARQLWrapper("https://query.wikidata.org/sparql")
    sparql.setQuery(f"""
    SELECT ?genreLabel WHERE {{
        ?book wdt:P31 wd:Q571 .
        ?book wdt:P136 ?genre .
        ?genre rdfs:label ?genreLabel .
        ?book rdfs:label "{book_title}"@en .
        FILTER (lang(?genreLabel) = 'en')
    }}
    """)
    sparql.setReturnFormat(JSON)
    results = sparql.query().convert()

    if len(results["results"]["bindings"]) > 0:
        return results["results"]["bindings"][0]["genreLabel"]["value"]

    sparql = SPARQLWrapper("http://dbpedia.org/sparql")
    sparql.setQuery(f"""
    PREFIX dct: <http://purl.org/dc/terms/>
    SELECT ?genreLabel WHERE {{
        ?book rdf:type dbo:Book .
        ?book dct:subject ?genre .
        ?genre rdfs:label ?genreLabel .
        ?book rdfs:label "{book_title}"@en .
        FILTER (lang(?genreLabel) = 'en')
    }}
    """)
    sparql.setReturnFormat(JSON)
    results = sparql.query().convert()

    if len(results["results"]["bindings"]) > 0:
        return results["results"]["bindings"][0]["genreLabel"]["value"]

    url = f"https://www.googleapis.com/books/v1/volumes?q={book_title}&maxResults=1"
    response = requests.get(url)
    data = response.json()

    if "items" in data and data["items"]:
        volume_info = data["items"][0]["volumeInfo"]
        if "categories" in volume_info and volume_info["categories"]:
            return volume_info["categories"][0]

    return None
```

Figura 18 - Função para ter gênero de livro

```

def get_author_info(self, author_name):
    sparql = SPARQLWrapper("https://query.wikidata.org/sparql")
    sparql.setQuery("""
SELECT ?birthDate ?occupationLabel ?genreLabel WHERE {{
    ?author wdt:P31 wd:Q5 .
    ?author wdt:P569 ?birthDate .
    ?author wdt:P106 ?occupation .
    ?author wdt:P136 ?genre .
    ?occupation rdfs:label ?occupationLabel .
    ?genre rdfs:label ?genreLabel .
    ?author rdfs:label "{author_name}"@en .
    FILTER (lang(?occupationLabel) = 'en')
    FILTER (lang(?genreLabel) = 'en')
}}
""")
    sparql.setReturnFormat(JSON)
    results = sparql.query().convert()

    if len(results["results"]["bindings"]) > 0:
        return results["results"]["bindings"][0]

    return None

```

Figura 19 - Função para ter informação de autor

A integração dos dados do WikiData e DBpedia foi um processo fundamental para aprimorar o conjunto de dados do projeto. Através dessas fontes, fomos capazes de complementar as informações dos autores e das capas dos livros, enriquecendo a experiência dos usuários. O acesso programático aos endpoints SPARQL e o uso das APIs forneceram uma abordagem eficiente e automatizada para obter os dados necessários.

6. RDFa e Microformatos

6.1. RDFa

O RDFa (Resource Description Framework in Attributes) é uma extensão do HTML que permite a incorporação de dados estruturados no conteúdo da página. Esses dados podem ser interpretados por máquinas, de forma a facilitar a indexação e a recuperação de informações por mecanismos de busca e outras plataformas.

No projeto, foram aplicados elementos RDFa nas páginas "book.html" e "author.html" para enriquecer o conteúdo com metadados semânticos. Esses metadados são interpretados, para melhorar a compreensão e a apresentação dos dados.

No código, foram utilizados os atributos **property** para definir propriedades RDF e **href** para estabelecer links entre recursos. Estes elementos RDFa foram aplicados nas páginas correspondentes para adicionar contexto semântico aos dados apresentados, que facilita a indexação dos livros e autores pelos mecanismos de busca, melhorando a visibilidade e a compreensão do conteúdo pelos utilizadores

Além disso, foi realizado um teste de validação dos elementos RDFa presentes nas páginas utilizando o site "<https://rdfa.info/play/>". Essa ferramenta permite verificar a correta implementação dos elementos RDFa e sua conformidade com as especificações.

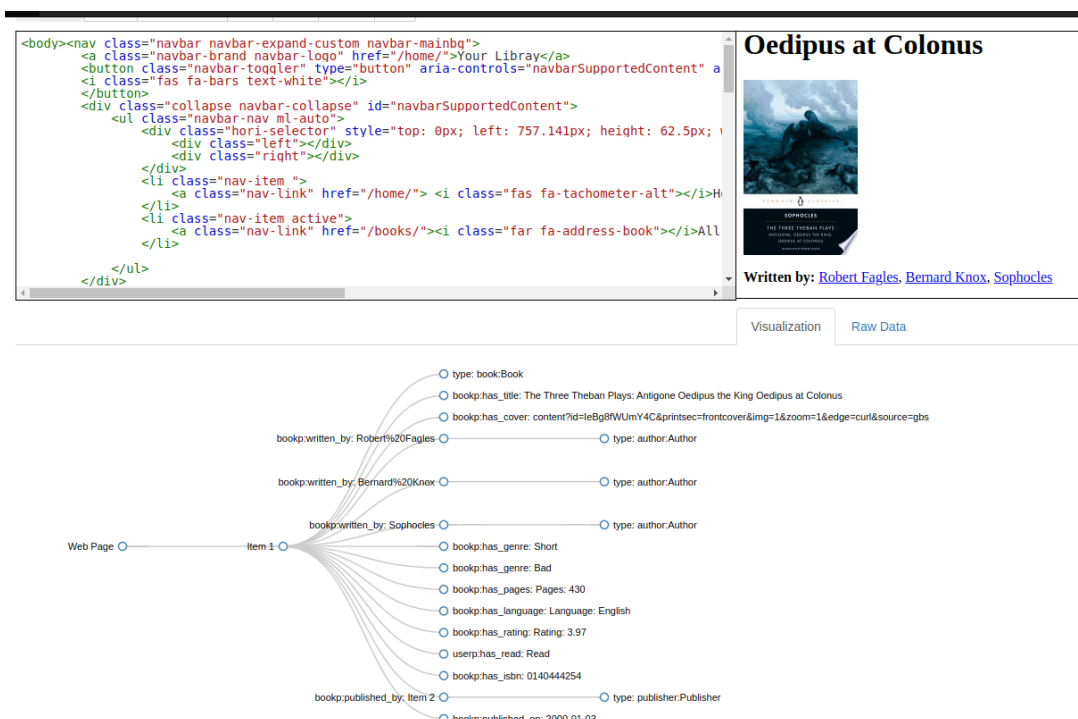


Figura 20 - Validação RDFa da página do livro

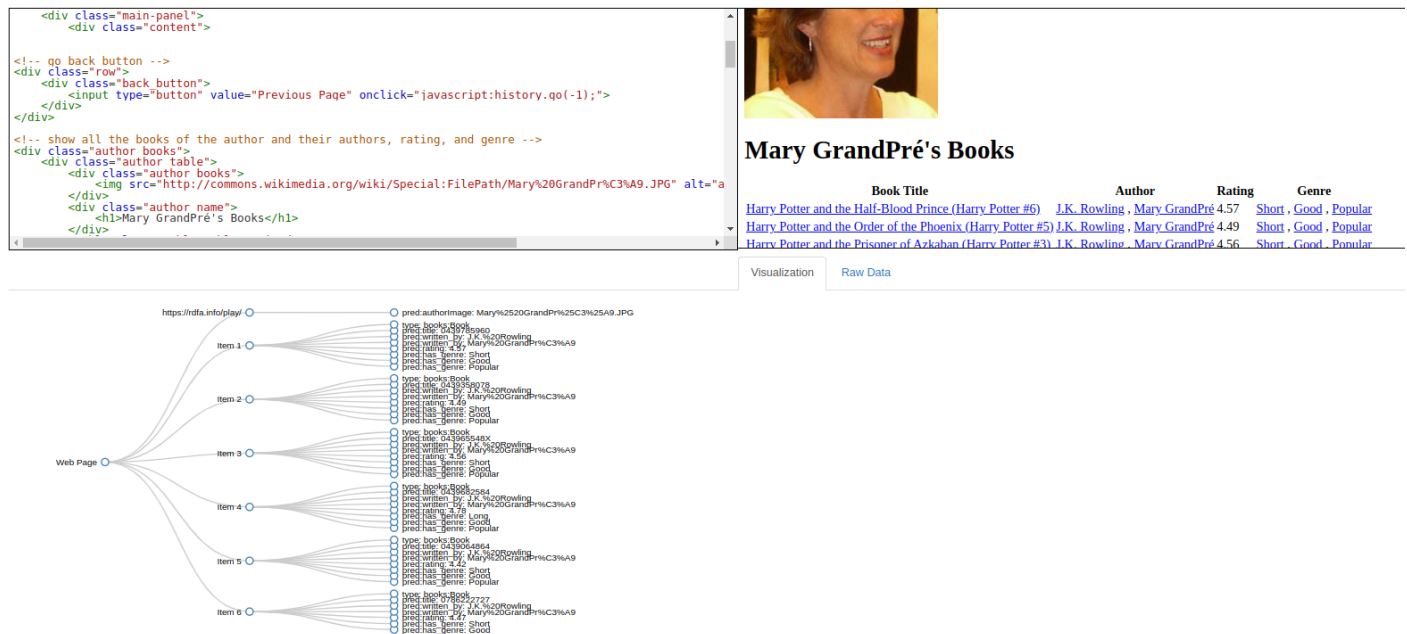


Figura 21 - Validação RDFa da página do autor

6.2. Microformatos

De forma a utilizar os Microformatos lecionados na unidade curricular adicionámo-los às páginas *Book* e *Author*, para guardar o título do livro/nome do autor e a imagem da capa do livro/do autor (Figura 22). A Figura 23 contém um exemplo do que é retornado pelo *parser*.

```
<div class="book h-card">
  <div class="book title">
    <h1 class="p-name">{{ book.title }}</h1>
  </div>
  <div class="book info">
    <!-- all authors with link to their page -->
    <div class="row" style="display: flex">
      <div class="column" style="flex: 50%; text-align: center;">--
    </div>
    <div class="column" style="flex: 50%; text-align: center;">
      
    </div>
  </div>
</div>
```

Figura 22 - Atribuição dos microformatos

```
JSON
{
  "items": [
    {
      "type": [
        "h-card"
      ],
      "properties": {
        "name": [
          "Harry Potter and the Half-Blood Prince (Harry Potter #6)"
        ],
        "photo": [
          {
            "value": "http://books.google.com/books/content?id=qakenvL29UC&printsec=frontcover&img=1&zoom=1&edge=cur&l&source=gbs_api",
            "alt": "Harry Potter and the Half-Blood Prince (Harry Potter #6)"
          }
        ]
      }
    }
  ],
  "rels": {},
  "rel-urls": {},
  "debug": {
    "package": "https://packagist.org/packages/mf2/mf2",
    "source": "https://github.com/indieweb/php-mf2",
    "version": "v0.5.0",
  }
}
```

Figura 23 - Microformato

6.3. Conclusão

Consideramos que tanto os microformatos como os RDFa não são muito úteis para esta aplicação visto que, apesar de termos uma quantidade grande de dados, não temos uma quantidade grande de propriedades, não sendo necessário facilitar a sua compreensão.

7. Funcionalidades da Aplicação

As funcionalidades desta aplicação são semelhantes às do projeto anterior. Desta forma, ao inicializar o projeto é exposta uma página inicial da aplicação *Your Library* (Figura 24), tendo acesso às maiores funcionalidades da mesma, a pesquisa por valores, e os livros filtrados por categorias.

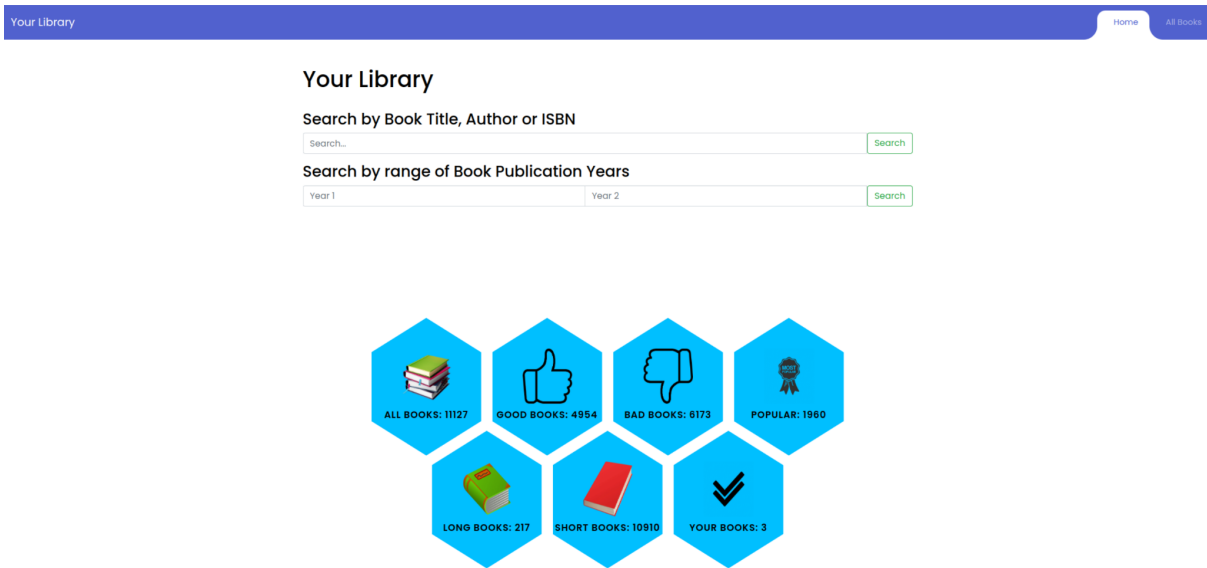


Figura 24 - Página Inicial

Ao escolher a opção 'All Books' é mostrada uma tabela paginada com todos os livros existentes, sendo possível ordená-la por vários parâmetros: título do livro, nome dos autores, nome do tipo de livros, classificação, ISBN e pelos livros já lidos pelo utilizador; clicando no nome da coluna (Figura 25).

Your Library

Home

All Books

Previous Page

All Books

Book Title	Author	Genre	Rating	ISBN	Read
Harry Potter and the Half-Blood Prince (Harry Potter #6)	J.K. Rowling, Mary GrandPré	good, long, popular	4.57	0439785960	<input checked="" type="checkbox"/>
Harry Potter and the Order of the Phoenix (Harry Potter #5)	J.K. Rowling, Mary GrandPré	good, long, popular	4.49	0439358078	<input checked="" type="checkbox"/>
Harry Potter and the Chamber of Secrets (Harry Potter #2)	J.K. Rowling	good, long	4.42	0439554896	<input type="checkbox"/>
Harry Potter and the Prisoner of Azkaban (Harry Potter #3)	J.K. Rowling, Mary GrandPré	good, long, popular	4.56	043985548x	<input type="checkbox"/>
Harry Potter Boxed Set Books 1-5 (Harry Potter #1-5)	J.K. Rowling, Mary GrandPré	good, long, popular	4.78	0439682584	<input type="checkbox"/>
Unauthorized Harry Potter Book Seven News: "Half-Blood Prince" Analysis and Speculation	W. Frederick Zimmerman	short	3.74	0976540606	<input type="checkbox"/>
Harry Potter Collection (Harry Potter #1-6)	J.K. Rowling	good, long, popular	4.73	0439827604	<input type="checkbox"/>
The Ultimate Hitchhiker's Guide: Five Complete Novels and One Story (Hitchhiker's Guide to the Galaxy #1-5)	Douglas Adams	good, long	4.38	0517226952	<input type="checkbox"/>
The Ultimate Hitchhiker's Guide to the Galaxy (Hitchhiker's Guide to the Galaxy #1-5)	Douglas Adams	good, long, popular	4.38	0345453743	<input type="checkbox"/>
The Hitchhiker's Guide to the Galaxy (Hitchhiker's Guide to the Galaxy #1)	Douglas Adams	good, short	4.22	1400052920	<input type="checkbox"/>
The Hitchhiker's Guide to the Galaxy (Hitchhiker's	Douglas Adams, Stephen Fry	good,	4.22	079322206	<input type="checkbox"/>

Figura 25 - Página All Books

Além da ordenação também é possível filtrar os livros por autor e tipo clicando no item desejado. Para ver detalhes do livro apenas é necessário clicar no título do mesmo, aqui também é possível marcar um livro como lido ou remover essa opção (Figura 26).

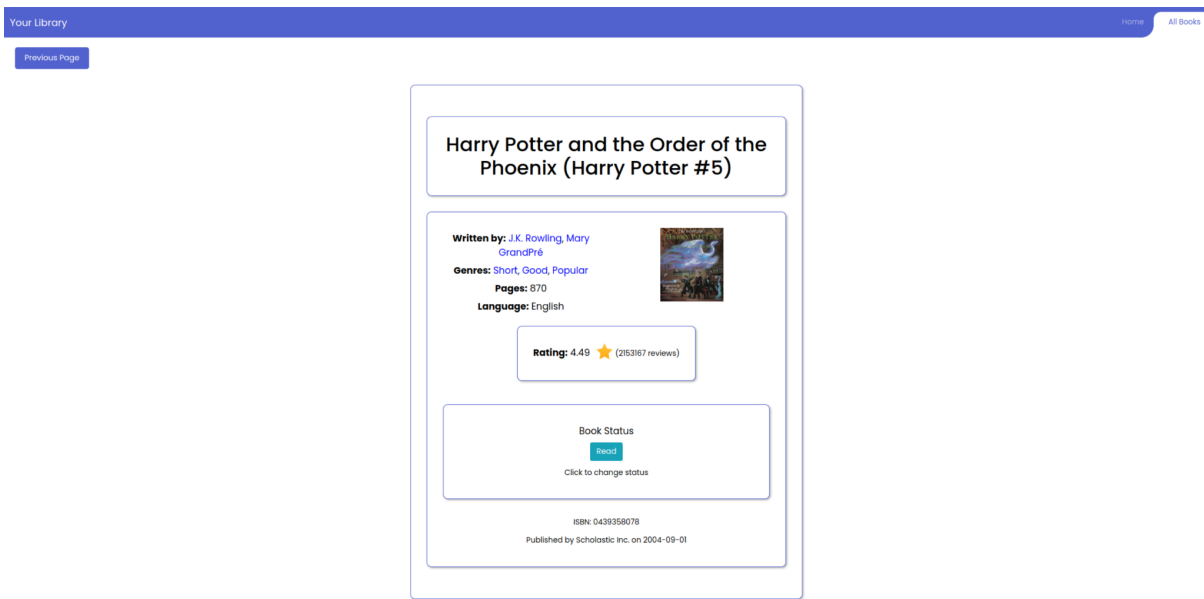


Figura 26 - Página individual do Livro

Aqui é possível encontrar uma diferença do projeto anterior, pois incluímos também a imagem da capa do livro, que foi encontrada através do Wikidata e DBpedia.

As restantes opções do menu inicial têm todas tabelas semelhantes, mas filtradas pela especificação escolhida, caso tenha boas ou más classificações, seja popular, seja longo ou curto, ou caso parte dos livros lidos pelo utilizador.

Realizando uma pesquisa a partir do menu, o utilizador é redirecionado para outra página (Figura 27), que tem uma tabela semelhante às anteriores. Esta pesquisa compara a palavra dada com as características associadas aos livros, título, autor, classificação, etc.

Results for key: harry						Results for years: 2000 to 2002					
Search...						Year 1 Year 2					
Search...						Search...					
Book Title	Author	Genre	Rating	ISBN	Read	Book Title	Author	Genre	Rating	ISBN	Read
Harry Potter and the Half-Blood Prince (Harry Potter #6)	J.K. Rowling, Mary GrandPré	good, long, popular	4.57	0439785960	<input checked="" type="checkbox"/>	The Three Trabant Plays: Antigone, Oedipus the King, Oedipus at Colonus	Robert Fagles, Bernard Knox, Sophocles	Short, Best	3.97	0140444254	<input type="checkbox"/>
Harry Potter and the Order of the Phoenix (Harry Potter #5)	J.K. Rowling, Mary GrandPré	good, long, popular	4.49	0439358078	<input checked="" type="checkbox"/>	The Tao of Physics: An Exploration of the Parallels between Modern Physics and Eastern Mysticism	Fritjof Capra	Short, Best, Popular	3.97	1570925190	<input type="checkbox"/>
Harry Potter and the Chamber of Secrets (Harry Potter #2)	J.K. Rowling	good, long	4.42	0439554896	<input type="checkbox"/>	A Carl Hiaasen Collection: Stormy Weather, Tougat Season and Ship These	Carl Hiaasen, Edward Auer	Short, Good	4.45	0375404465	<input type="checkbox"/>
Harry Potter and the Prisoner of Azkaban (Harry Potter #3)	J.K. Rowling, Mary GrandPré	good, long, popular	4.56	043965548X	<input type="checkbox"/>	The Crime of the Curious Bride (Perry Mason Mystery)	Erie Stanley Gardner	Short, Bad	3.87	0345437637	<input type="checkbox"/>
Harry Potter Boxed Set Books 1-5 (Harry Potter #1-5)	J.K. Rowling, Mary GrandPré	good, long, popular	4.78	0439682584	<input type="checkbox"/>	Working with Emotional Intelligence	Daniel Goleman	Short, Bad	3.81	0553376589	<input type="checkbox"/>
Unauthorized Harry Potter Book Seven News: "Half-Blood Prince" Analysis and Speculation	W. Frederick Zimmerman	short	3.74	0970540606	<input type="checkbox"/>	The Guardianship (Thomas Moreau #1)	James L. Nelson	Short, Good	4.01	0380804622	<input type="checkbox"/>
Harry Potter Collection (Harry Potter #1-6)	J.K. Rowling	good, long, popular	4.73	0439627004	<input type="checkbox"/>	The Seeds of the Disturber (Annelia Peabody #5)	Elizabeth Peters	Short, Good, Popular	4.05	0380739599	<input type="checkbox"/>
Liar's Poker: A Harry Gamish Mystery	Frank McConnell	short, bad	3.31	0802732291	<input type="checkbox"/>	Angels Right (Harry Bosch #6, Harry Bosch Universe #7)	Michael Connelly	Short, Good, Popular	4.18	0446607274	<input type="checkbox"/>
Harry Potter Schoolbooks Box Set: Two Classic Books from the Library of Hogwarts School of Witchcraft and Wizardry	J.K. Rowling	good, popular, short	4.4	043932962X	<input type="checkbox"/>	The Poisonwood Bible	Barbara Kingsolver	Short, Good	4.06	05730075X	<input type="checkbox"/>
J.K. Rowling's Harry Potter Novels: A Reader's Guide	Philip Nel	short	3.58	0825452329	<input type="checkbox"/>	Children of the Star (Children of the Star #1-3)	Sylvia Engelsh	Short, Good	4.24	1892005142	<input type="checkbox"/>
Harry Potter and the Half-Blood Prince (Harry Potter #6)	J.K. Rowling	good, long	4.57	0747584664	<input type="checkbox"/>						
Harry Potter y La Piedra Filosofal (Harry Potter #1)	J.K. Rowling	good, short	4.47	0613359607	<input type="checkbox"/>						

Figura 27 - Pesquisa por palavra-chave ou anos


A partir da aplicação também podemos ver os livros escritos por cada autor, clicando em qualquer menção do mesmo (Figura 28). Podendo também ver a foto do autor que foi adquirida com recurso à Wikidata, DBpedia ou googleAPI.

Your Library

HomeAll Books

Previous Page

Pauline Kael's Books



Book Title	Author	Rating	Genre
Afterglow: A Last Conversation With Pauline Kael	Francis Davis, Pauline Kael	3.69	Short, Bad
For Keeps: 30 Years at the Movies	Pauline Kael	4.48	Long, Good
Hooked: Film Writings 1985-1988	Pauline Kael	4.05	Short, Good

Figura 28 - Pesquisa por palavra-chave ou anos

Por último, ao navegar pela aplicação, existe um botão “Previous Page” pelo qual é possível voltar às páginas anteriores.

8. Conclusões

Com este projeto aprendemos sobre dados e todo o processo que é necessário fazer até estarem prontos a ser vistos e usados pelos utilizadores. Notámos que esta abordagem é muito interessante, com a possibilidade de criar inferências com a ontologia.

No futuro, gostaríamos de implementar autorização e autenticação de forma a poder ter vários utilizadores. Além disso, gostaríamos de ter uma página para o utilizador para ser possível realizar as operações CRUD nos dados. Por último, acreditamos que nosso sistema pode ser melhorado no sentido de intensificar sua complexidade, por incluir mais informações para entidades como autor e editora, bem como com a criação de mais inferências.

9. Configuração para executar a aplicação

Para correr a aplicação é necessário executar os seguintes passos:

1. Criar repositório no graphDB
 - a. Setup > Repositories > Create new repository > GraphDB Repository
 - b. Repository ID: books
2. Importar os dados para o GraphDB
 - a. Import > Upload RDF files
 - b. Open /Converter/results.ttl
 - c. Import
3. Executar projeto Django dentro de um ambiente virtual
 - a. pip install -r requirements.txt
 - b. python3 books/manage.py runserver

10. Referências

Soumik. 2020. "Goodreads-books." Kaggle.

<https://www.kaggle.com/datasets/jealousleopard/goodreadsbooks>.