universidade de aveiro deti departamento de eletrónica, telecomunicações e informática

Trabalho Prático 1

Web Semântica

Docente Hélder Zagalo
2022/2023

Mestrado em Engenharia Informática

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

Índice

1. Introdução

O trabalho prático proposto pelo docente da Unidade Curricular Web Semântica é o desenvolvimento de um sistema de informação baseado na web, sobre um tema à nossa escolha.

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 e SPARQL para a pesquisa e alteração dos dados no repositório.

Para desenvolver o projeto descrito, foi utilizado um dataset[1] 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. Dados, suas fontes e sua transformação

Como referido anteriormente, os dados foram retirados do Kaggle (Soumik 2020) e possui 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*.

```
# 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*

A segunda informação adicionada referente a categorizar os livros, visa também melhorar os dados e a experiência do utilizador. Os livros foram categorizados como "*long*", "*short*", "*good*", "*bad*" e "*popular*", tendo em conta as diversas informações retiradas dos mesmo, tais como número de páginas (para "*long*" e "*short*"), classificação (para "*good*" e "*bad*) e o número de *reviews* (para "*popular*"). Esta nova informação foi adicionada através de várias queries (Figura 3, 4, 5, 6 e 7), tendo em conta os diversos parâmetros e podendo cada livro ter mais de uma categoria. Posteriormente foram editados os valores de "*good*" para rating superior a 4.0, e de "*short*" para livros inferiores a 300 páginas.

```
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_genre ?genre .
}
INSERT {
    ?book pred:has_genre "short" .
}
WHERE {
    ?book pred:has_pages ?pages .
    FILTER (xsd:integer(?pages) < 1000)
}
"""
```

Figura 3 - Categoria “Short”

```
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_genre ?genre .
}
INSERT {
    ?book pred:has_genre "good" .
}
WHERE {
    ?book pred:has_rating ?rating .
    FILTER (xsd:decimal(?rating) > 4.5)
}
"""
```

Figura 4 - Categoria “Good”

```
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_genre ?genre .
}
INSERT {
    ?book pred:has_genre "long" .
}
WHERE {
    ?book pred:has_pages ?pages .
    FILTER (xsd:integer(?pages) > 1000)
}
"""
```

Figura 5 - Categoria “Long”

```
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_genre ?genre .
}
INSERT {
    ?book pred:has_genre "popular" .
}

WHERE {
    ?book pred:rated_by ?reviews .
    FILTER (xsd:integer(?reviews) > 10000)
}
"""
```

Figura 6 - Categoria “Popular”

```
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_genre ?genre .
}
INSERT {
    ?book pred:has_genre "bad" .
}
WHERE {
    ?book pred:has_rating ?rating .
    FILTER (xsd:decimal(?rating) < 3.5)
}
"""
```

Figura 7 - Categoria “Bad”

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. Em seguida, o arquivo 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 uma informação sobre o livro ter sido visto 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.

```
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 8 - Código conversão do ficheiro csv para N-Triples

A arquitetura final do modelo de dados, ao final, fica como o diagrama da figura 9.

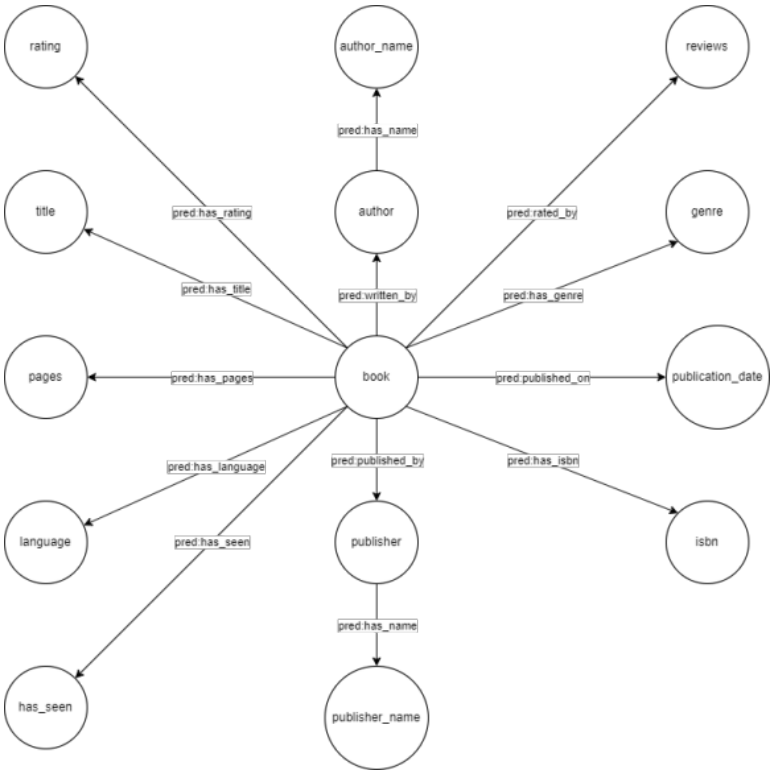

Figura 9 - Diagrama do modelo de dados.

3. Operações sobre os dados (SPARQL)

Os dados foram assim armazenados num repositório "*Triplestore* GraphDB", e para se poder consultar, editar e pesquisar dados foram realizadas diversas *queries*, presentes no ficheiro queries.py dentro da pasta GraphDB, muitas destas *queries* foram criadas para explorar o *dataset* e testar/criar as futuras queries a serem usadas na aplicação, que se encontram no ficheiro queries.py dentro do projeto Django.

Uma das *queries* utilizadas na aplicação (Figura 9), seleciona informações sobre títulos, autores, páginas, género, classificação, número de *reviews*, língua, editora, data de publicação e ISBN de livros que correspondem a uma pesquisa do utilizador, num dos campos de título, ISBN, nome do autor ou nome da editora. A cláusula UNION combina os resultados das quatro consultas, cada uma correspondente a um campo diferente.

```
#Search for a book by title, author, isbn or publisher
searchBook = """
PREFIX books: <http://books.com/books/>
PREFIX pred: <http://books.com/preds/>
PREFIX authors: <http://books.com/authors/>
PREFIX publishers: <http://books.com/publishers/>
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:written_by ?author .
    ?author pred:has_name ?author_name .
    ?book pred:has_pages ?pages .
    ?book pred:has_genre ?genre .
    ?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 .
    FILTER regex(?title, "toSearch", "i")
    }
    UNION
    {
    ?book pred:has_title ?title .
    ?book pred:written_by ?author .
    ?author pred:has_name ?author_name .
    ?book pred:has_pages ?pages .
    ?book pred:has_genre ?genre .
    ?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 .
    FILTER regex(?isbn, "toSearch", "i")
    }
    UNION
    {
    ?book pred:has_title ?title .
    ?book pred:written_by ?author .
    ?author pred:has_name ?author_name .
    ?book pred:has_pages ?pages .
    ?book pred:has_genre ?genre .
    ?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 .
    FILTER regex(?author_name, "toSearch", "i")
    }
    UNION
    {
    ?book pred:has_title ?title .
    ?book pred:written_by ?author .
    ?author pred:has_name ?author_name .
    ?book pred:has_pages ?pages .
    ?book pred:has_genre ?genre .
    ?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 .
    FILTER regex(?publisher_name, "toSearch", "i")
    }
}
"""
```

Figura 10 - Query Search

Outra *query* utilizada (figura 10), serve para dar update no livro, quando um utilizador marca o mesmo “como visto”, ou quando o desmarca de “como visto”.

```
#Updates the book to the inverse of the current value
updateBook = """
PREFIX books: <http://books.com/books/>
PREFIX pred: <http://books.com/preds/>
DELETE {
    ?book pred:has_seen ?has_seen .
}
INSERT {
    ?book pred:has_seen ?new_has_seen .
}
WHERE {
    ?book pred:has_isbn ?isbn .
    ?book pred:has_seen ?has_seen .
    BIND (IF(?has_seen = "true"^^xsd:boolean, "false"^^xsd:boolean, "true"^^xsd:boolean) AS ?new_has_seen)
    FILTER regex(?isbn, "replace", "i")
}
"""
```

Figura 11 - Query de update ao campo “has_seen”

Outra *query* foi utilizada (Figura 11) para consultar os livros de um dado autor, tendo em consideração que se pretende saber os restantes autores que participaram no livro.

```
#Get books from author and the rest of the authors from the same books
getBooksByAuthor = """
PREFIX books: <http://books.com/books/>
PREFIX pred: <http://books.com/preds/>

SELECT ?title ?author_name ?pages ?genre ?rating ?reviews ?has_seen ?language ?publisher_name ?publication_date ?isbn ?co_author_name
WHERE {
  # Subquery to find all books written by J.K. Rowling
  {
    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_genre ?genre .
  ?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 .

  # Retrieve the names of all co-authors for the book
  OPTIONAL {
    ?book pred:written_by ?co_author .
    ?co_author pred:has_name ?co_author_name .
    FILTER(?co_author_name != ?author_name)
  }
}

"""
```

Figura 12 - Query que devolve os livros de um autor e os restantes autores também

Foram utilizadas muitas outras *queries*, como por exemplo para saber o número de livros de cada categoria e o número total, e também esses mesmos livros. Também foram utilizadas *queries* para pesquisar livros pelo isbn, pesquisar livros por um intervalo de anos, e ter os livros vistos pelo utilizador. Estas queries estão presentes no ficheiro queries.py que se encontra dentro da pasta GraphDB.

4. Funcionalidades da Aplicação (UI)

Ao inicializar o projeto é exposta uma página inicial da aplicação *Your Library* (Figura 12), tendo acesso às maiores funcionalidades da mesma, a pesquisa por valores, e os livros filtrados por categorias.

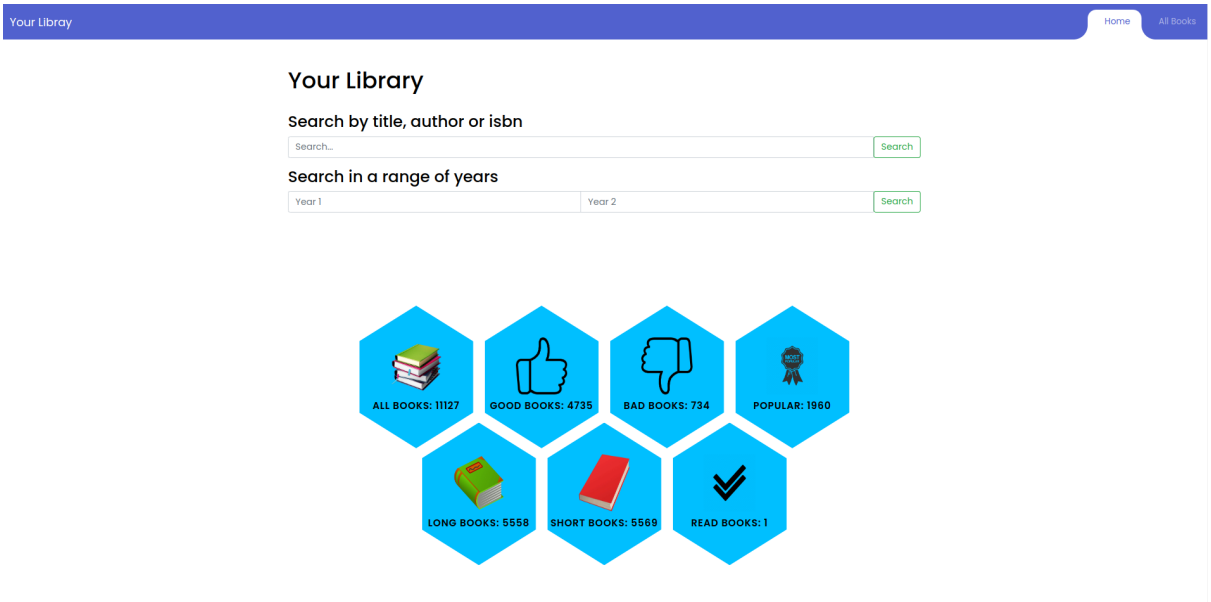

FIgura 13 - Página Inicial

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 13).

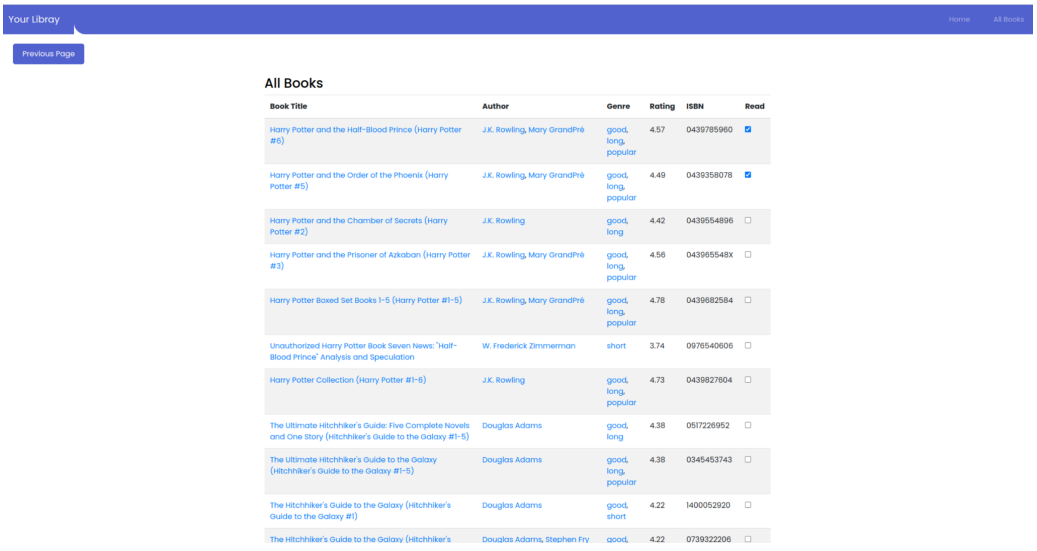

FIgura 14 - Página Inicial

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 (Figura 14).

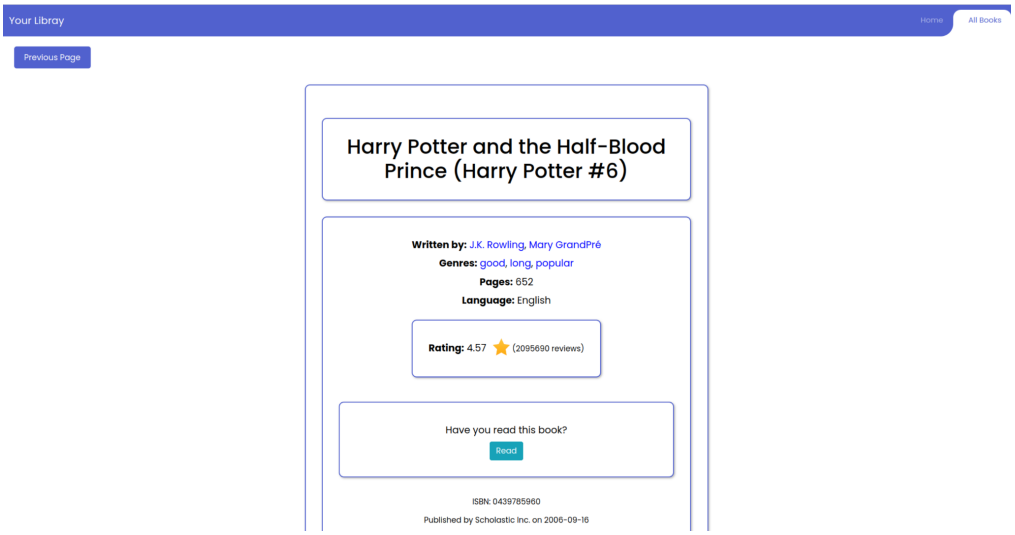

FIgura 15 - Página individual do Livro

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 15), que tem uma tabela semelhante às anteriores. Esta pesquisa compara a palavra dada com as características associadas aos livros, título, autor, classificação, etc.

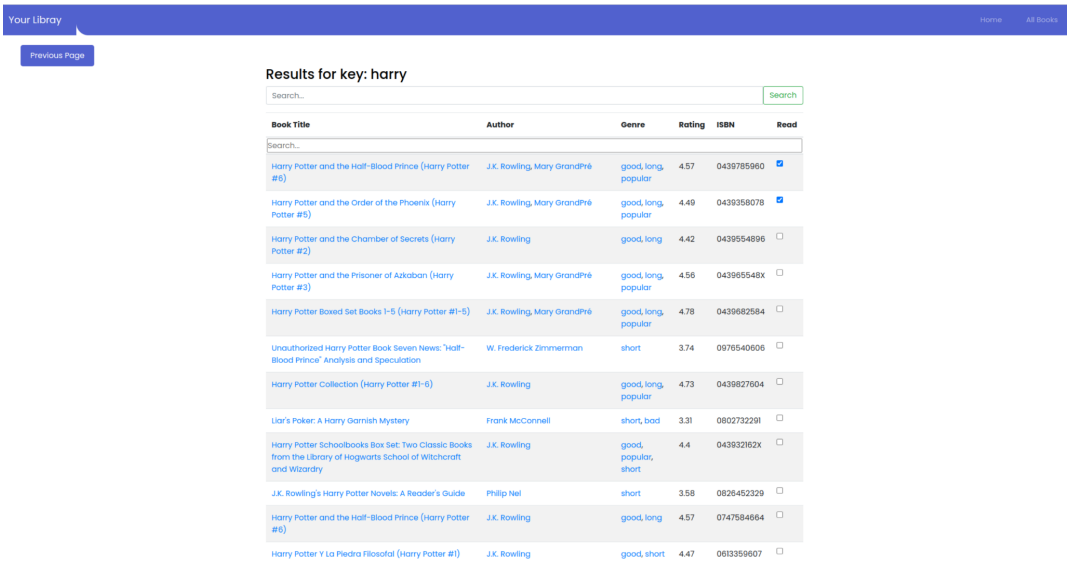

FIgura 15 - Pesquisa por palavra-chave

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

5. 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 dá-nos bastante flexibilidade nos dados que podemos mostrar aos utilizadores.

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, por exemplo, e criar uma entidade utilizador para poder melhorar a usabilidade da aplicação.

6. 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/new.nt
    c. Import
3. Executar projeto Django dentro de um ambiente virtual
    a. pip install -r requirements.txt
    b. python3 books/manage.py runserver
4. Executar conversor
    a. python3 Converter/csv_to_nt.py Converter/books.csv

7. Referências

Soumik. 2020. “Goodreads-books.” Kaggle.

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