University of Aveiro

2023 / 2024

Exploring a Netflix Catalog

Semantic Web – Assignment 1

Group 5
Bruno Nunes, 80614 | David Raposo, 93395 | Rafael Gil, 118377 | Salomé Dias, 118163

Index

Introduction

Humans and machines perceive information differently. Humans can read, interpret and interrelate information from multiple sources. With data just by itself, machines do not have this ability. A semantic web is the data extended world wide web with the help of metadata which allows machine to perceive and interrelate data.

Transforming data into graphs and interrelating them is a way of accomplishing a semantic web. *Resource Description Framework* or RDF is a technology that allows the graph representation in a *triplestore* standard format. This standard format can be used by *GraphDB*, a *triplestore* data oriented database which supports the SPARQL language which for its turn allows the interaction with the data within it.

To explore a dataset selected by us with these technologies we developed a python web application with the help of Django framework.

In this report we begin to define the dataset used and the transformations required to use it as a RDF standard example. Afterwards we explain the data operations performed and the SPARQL querys used to do so. Functionality of the developed web application on Django framework is presented subsequently and finally we present the conclusion from the development of this project. Additionally, a section describing the configurations and requirements to run the application is included in the report.

Dataset and transformation

In order to cope with the objectives of this assignment we needed to, as a first step, define the dataset to use. With the selected dataset we performed a file format conversion so as to create a *tiplestore* database using for that matter *GraphDB*.

Dataset Description

A well-known streaming platform, Netflix, was our base inspiration as it should provide a good amount of data from various movies and tv shows. The work base dataset was sourced from *Kaggle* [1] which is a repository of data and models.

The dataset came originally in *.csv* format. Its content has total amount of 8809 information entrys with information of a total of 11 columns which comprise the following content:

- show_id,
- type,
- title,
- director,
- cast,
- country,
- date_added,
- release_year,
- rating,duration,
- listed_in,
- description.

File format type conversion

To use this dataset with a *Triplestore* oriented database like *GraphDB* as intended, we converted to one of its recognizable and implemented standards as it is RDF (*Resource Description Framework*).

This framework as multiple accepted file formats being the most used NT (N-Triples), N3, RDF/XML and RDFa. Due to its simplicity and widely usage, our choice fell on NT format.

Owing to the large amount of data, it was not reasonable to perform the format conversion manually. Therefore, we developed a small python program to do so, *translation_script.py*, that can be found on the project repository. In this program, which make use of *rdflib* library, for each movie or tv show entry, a unique *URI* is generated for show_id value as well as all other column values as attribute objects. Excluding the show_id column name, all the other column names were used to generate the major predicate *URIs* to connect the information in the triple format which consists of subject, predicate, and object.

Additionally, to be able to get the attribute's real values (literals and/or numbers), another relation was created in this program between them through the "real_name" predicate.

As a demonstration of the conversion stands the following example that contains information before and after the conversion.

Before, on .csv file:

show_id,type,title,(...),rating,duration,listed_in,description
s2337,Movie,Thackeray (Hindi), (...),TV-14,135 min,"Dramas, International Movies","From controversial cartoonist to powerful Mumbai politician, this biopic maps the meteoric rise of far-right Shiv Sena party founder, Bal Thackeray."
(...)

After conversion, on the *.nt* file:

<http://ws.org/netflix_info/ss2337> <http://ws.org/netflix_info/pred/rating> <http://ws.org/netflix_info/TV-14> .
(...)
<http://ws.org/netflix_info/ss2337> <http://ws.org/netflix_info/pred/duration> <http://ws.org/netflix_info/135_min> .
(...)
<http://ws.org/netflix_info/ss2337> <http://ws.org/netflix_info/pred/title> <http://ws.org/netflix_info/Thackeray_%28Hindi%29> .

Example 1: Demonstration of convertion results.

Triplestore data base

Finally, with the converted file, we created the *triplestore* database on *GraphDB* to interact with the data.

Operations on Data

Interaction with the *GraphDB* database created can be performed through the SPARQL language. *Simple Protocol and RDF Query Language* (SPARQL), as pointed from the definition, is a query language which allows the search, insertion, deletion, and update of data.

Searching Data

On the searching category, SPARQL has 4 types of queries: SELECT, ASK, DESCRIBE, and CONSTRUCT. Here we privileged the SELECT query type which is also the most used one. On this project we perform different type of searches through data. In the main page of the project its possible to perform a generic search from within movies, tv shows, actors, directors. On other page we allow more specific search operation like search for a specific movie, search for a specific cast member, search for a specific director, search for movies between dates, search for movies release in a given year, and finally, search for movies based on genres.

Search for a specific movie

When searching for a specific movie, we want to retrieve several instances of data, such as the title, the director, the cast, etc, that represent the movie as a whole. In order to do that, the following query was constructed (Figure 1 and Figure 2).

```
query = """
        PREFIX net: <http://ws.org/netflix_info/pred/>

        SELECT ?type ?title ?director (GROUP_CONCAT(DISTINCT ?cast; separator=", ") AS ?mergedCasts)
            ?country ?date_added ?release_year ?rating ?duration
            (GROUP_CONCAT(DISTINCT ?genres; separator=", ") AS ?mergedGenres) ?description
        WHERE {
            ?title_code net:real_name "_movie_name" .
            ?show_id net:title ?title_code .

            OPTIONAL{
                ?show_id net:type ?type_code .
                ?type_code net:real_name ?type .
            }

            ?title_code net:real_name ?title .

            OPTIONAL {
                ?show_id net:director ?director_code .
                ?director_code net:real_name ?director .
            }
            OPTIONAL {
                ?show_id net:cast ?cast_code .
                ?cast_code net:real_name ?cast .
            }
            OPTIONAL {
                ?show_id net:country ?country_code .
                ?country_code net:real_name ?country .
            }
            OPTIONAL {
                ?show_id net:date_added ?date_code .
                ?date_code net:real_name ?date_added .
            }
```

Figure 1: SELECT query - search for a specific movie (1/2).

```
    OPTIONAL {
        ?show_id net:release_year ?release_code .
        ?release_code net:real_name ?release_year .
    }
    OPTIONAL {
        ?show_id net:rating ?rating_code .
        ?rating_code net:real_name ?rating .
    }
    OPTIONAL {
        ?show_id net:duration ?duration_code .
        ?duration_code net:real_name ?duration .
    }
    OPTIONAL {
        ?show_id net:listed_in ?genres_code .
        ?genres_code net:real_name ?genres .
    }
    OPTIONAL {
        ?show_id net:description ?description_code .
        ?description_code net:real_name ?description .
    }
}
GROUP BY ?type ?title ?director ?country ?date_added ?release_year ?duration ?rating ?description
    """
```

Figure 2: SELECT query - search for a specific movie (1/2).

There's two things that are worth highlighting in this query: the OPTIONAL keyword and the GROUP_CONCAT function.

The OPTIONAL is being use to prevent the query from malfunctioning when no object is found, which happens quite often after using the DELETE functionalities.

The GROUP_CONCAT is being used in order to group both the genres and the cast members, in a single object, to make the returned values simpler and easier to iterate in the interface.

Search for a specific cast member

This query (Figure 3 and Figure 4) is very similar to the previous one, with the exception that now we search for the "show_id" of the shows (movies and tv shows) where a given cast member participated in and proceed to get the rest of the information of those shows.

```
query = """
        PREFIX net: <http://ws.org/netflix_info/pred/>

        SELECT ?type ?title ?director (GROUP_CONCAT(DISTINCT ?cast; separator=", ") AS ?mergedCasts)
               ?country ?date_added ?release_year ?rating ?duration
               (GROUP_CONCAT(DISTINCT ?genres; separator=", ") AS ?mergedGenres) ?description
        WHERE {
            ?castMember_code net:real_name "_cast_name" .
            ?show_id net:cast ?castMember_code .

            OPTIONAL {
                ?show_id net:title ?title_code .
                ?title_code net:real_name ?title .
            }

            OPTIONAL{
                ?show_id net:type ?type_code .
                ?type_code net:real_name ?type .
            }

            OPTIONAL {
                ?show_id net:director ?director_code .
                ?director_code net:real_name ?director .
            }
            OPTIONAL {
                ?show_id net:cast ?cast_code .
                ?cast_code net:real_name ?cast .
            }
            OPTIONAL {
                ?show_id net:country ?country_code .
                ?country_code net:real_name ?country .
            }
            OPTIONAL {
                ?show_id net:date_added ?date_code .
                ?date_code net:real_name ?date_added .
            }
            OPTIONAL {
                ?show_id net:release_year ?release_code .
                ?release_code net:real_name ?release_year .
            }
            OPTIONAL {
                ?show_id net:rating ?rating_code .
                ?rating_code net:real_name ?rating .
            }
```

Figure 3: SELECT query - search for a specific cast member (1/2).

```
        OPTIONAL {
            ?show_id net:duration ?duration_code .
            ?duration_code net:real_name ?duration .
        }
        OPTIONAL {
            ?show_id net:listed_in ?genres_code .
            ?genres_code net:real_name ?genres .
        }
        OPTIONAL {
            ?show_id net:description ?description_code .
            ?description_code net:real_name ?description .
        }
    }
    GROUP BY ?type ?title ?director ?country ?date_added ?release_year ?rating ?duration ?description
```

Figure 4: SELECT query - search for a specific cast member (2/2).

Search for a specific director

We now search for the “show_id” of the shows directed by a specific director, proceeding to get the information related to those shows. For this purpose, the query presented in Figure 5 and Figure 6 was defined.

```
query = """
        PREFIX net: <http://ws.org/netflix_info/pred/>

        SELECT ?type ?title ?director (GROUP_CONCAT(DISTINCT ?cast; separator=", ") AS ?mergedCasts)
          ?country ?date_added ?release_year ?rating ?duration
          (GROUP_CONCAT(DISTINCT ?genres; separator=", ") AS ?mergedGenres) ?description
        WHERE {
            ?dir_code net:real_name "_director_name" .
            ?show_id net:director ?dir_code .

            OPTIONAL {
                    ?show_id net:title ?title_code .
                    ?title_code net:real_name ?title .
                }

                OPTIONAL{
                    ?show_id net:type ?type_code .
                    ?type_code net:real_name ?type .
                }

                OPTIONAL {
                    ?show_id net:director ?director_code .
                    ?director_code net:real_name ?director .
                }
                OPTIONAL {
                    ?show_id net:cast ?cast_code .
                    ?cast_code net:real_name ?cast .
                }
                OPTIONAL {
                    ?show_id net:country ?country_code .
                    ?country_code net:real_name ?country .
                }
                OPTIONAL {
                    ?show_id net:date_added ?date_code .
                    ?date_code net:real_name ?date_added .
                }
                OPTIONAL {
                    ?show_id net:release_year ?release_code .
                    ?release_code net:real_name ?release_year .
                }
```

Figure 5: SELECT query - search for a specific director (1/2).

```
        OPTIONAL {
            ?show_id net:rating ?rating_code .
            ?rating_code net:real_name ?rating .
        }
        OPTIONAL {
            ?show_id net:duration ?duration_code .
            ?duration_code net:real_name ?duration .
        }
        OPTIONAL {
            ?show_id net:listed_in ?genres_code .
            ?genres_code net:real_name ?genres .
        }
        OPTIONAL {
            ?show_id net:description ?description_code .
            ?description_code net:real_name ?description .
        }
}
GROUP BY ?type ?title ?director ?country ?date_added ?release_year ?rating ?duration ?description
```

Figure 6: SELECT query - search for a specific director (2/2).

Search for movies between dates

In this query, we make use of the FILTER function to filter the shows and only get those that aired between a given date interval (Figure 7 and Figure 8).

```
query = """
        PREFIX net: <http://ws.org/netflix_info/pred/>

        SELECT ?type ?title ?director (GROUP_CONCAT(DISTINCT ?cast; separator=", ") AS ?mergedCasts)
          ?country ?date_added ?release_year ?rating ?duration
          (GROUP_CONCAT(DISTINCT ?genres; separator=", ") AS ?mergedGenres) ?description
        WHERE {
            ?rel_code net:real_name ?year .
            ?show_id net:release_year ?rel_code .
            FILTER (?year >= "_date1" && ?year <= "_date2")

                OPTIONAL {
                    ?show_id net:title ?title_code .
                    ?title_code net:real_name ?title .
                }

                OPTIONAL{
                    ?show_id net:type ?type_code .
                    ?type_code net:real_name ?type .
                }

                OPTIONAL {
                    ?show_id net:director ?director_code .
                    ?director_code net:real_name ?director .
                }
                OPTIONAL {
                    ?show_id net:cast ?cast_code .
                    ?cast_code net:real_name ?cast .
                }
                OPTIONAL {
                    ?show_id net:country ?country_code .
                    ?country_code net:real_name ?country .
                }
                OPTIONAL {
                    ?show_id net:date_added ?date_code .
                    ?date_code net:real_name ?date_added .
                }
                OPTIONAL {
                    ?show_id net:release_year ?release_code .
                    ?release_code net:real_name ?release_year .
                }
```

Figure 7: SELECT query - search for movies between dates (1/2).

```
            OPTIONAL {
                ?show_id net:rating ?rating_code .
                ?rating_code net:real_name ?rating .
            }
            OPTIONAL {
                ?show_id net:duration ?duration_code .
                ?duration_code net:real_name ?duration .
            }
            OPTIONAL {
                ?show_id net:listed_in ?genres_code .
                ?genres_code net:real_name ?genres .
            }
            OPTIONAL {
                ?show_id net:description ?description_code .
                ?description_code net:real_name ?description .
            }
    }
    GROUP BY ?type ?title ?director ?country ?date_added ?release_year ?rating ?duration ?description
```

Figure 8: SELECT query - search for movies between dates (2/2).

Search for movies release in a given year

Similarly to the previous query, we use the FILTER function but with the exception of filtering based on a single date, getting the shows of a specific year (Figure 9 and Figure 10).

```
query = """
        PREFIX net: <http://ws.org/netflix_info/pred/>

        SELECT ?type ?title ?director (GROUP_CONCAT(DISTINCT ?cast; separator=", ") AS ?mergedCasts)
          ?country ?date_added ?release_year ?rating ?duration
          (GROUP_CONCAT(DISTINCT ?genres; separator=", ") AS ?mergedGenres) ?description
        WHERE {
            ?rel_code net:real_name ?year .
            ?show_id net:release_year ?rel_code .
            FILTER (?year = "_date")

            OPTIONAL {
                    ?show_id net:title ?title_code .
                    ?title_code net:real_name ?title .
                }

                OPTIONAL{
                    ?show_id net:type ?type_code .
                    ?type_code net:real_name ?type .
                }

                OPTIONAL {
                    ?show_id net:director ?director_code .
                    ?director_code net:real_name ?director .
                }
                OPTIONAL {
                    ?show_id net:cast ?cast_code .
                    ?cast_code net:real_name ?cast .
                }
                OPTIONAL {
                    ?show_id net:country ?country_code .
                    ?country_code net:real_name ?country .
                }
                OPTIONAL {
                    ?show_id net:date_added ?date_code .
                    ?date_code net:real_name ?date_added .
                }
                OPTIONAL {
                    ?show_id net:release_year ?release_code .
                    ?release_code net:real_name ?release_year .
                }
```

Figure 9: SELECT query - Search for movies in a given year (1/2).

```
        OPTIONAL {
            ?show_id net:rating ?rating_code .
            ?rating_code net:real_name ?rating .
        }
        OPTIONAL {
            ?show_id net:duration ?duration_code .
            ?duration_code net:real_name ?duration .
        }
        OPTIONAL {
            ?show_id net:listed_in ?genres_code .
            ?genres_code net:real_name ?genres .
        }
        OPTIONAL {
            ?show_id net:description ?description_code .
            ?description_code net:real_name ?description .
        }
}
GROUP BY ?type ?title ?director ?country ?date_added ?release_year ?rating ?duration ?description
```

Figure 10: SELECT query - Search for movies in a given year (2/2).

Search for movies based on genres

This search is somewhat special, as it allows for the user to search for multiple genres at a time. The way this query was built is based on the user input being separated by commas (,), allowing the user to specify various genres at a time. The user input is processed and is iterated over, constructing small queries that are added to the main query, forming a union between the genres specified (Figure 11 and Figure 12).

```
query = """
        PREFIX net: <http://ws.org/netflix_info/pred/>

        SELECT ?type ?title ?director (GROUP_CONCAT(DISTINCT ?cast; separator=", ") AS ?mergedCasts)
                ?country ?date_added ?release_year ?rating ?duration
                (GROUP_CONCAT(DISTINCT ?genres; separator=", ") AS ?mergedGenres) ?description
        WHERE {
        """

for index, genre in enumerate(genres_split):
    query = query + """
                    {
                        ?genre_code net:real_name "_genre" .
                        ?show_id net:listed_in ?genre_code .
                    }
            """
    query = query.replace("_genre", genre)

    if index < len(genres_split) - 1:
        query = query + """UNION"""

query = query + """
                OPTIONAL {
                    ?show_id net:title ?title_code .
                    ?title_code net:real_name ?title .
                }

                OPTIONAL{
                    ?show_id net:type ?type_code .
                    ?type_code net:real_name ?type .
                }

                OPTIONAL {
                    ?show_id net:director ?director_code .
                    ?director_code net:real_name ?director .
                }
                OPTIONAL {
                    ?show_id net:cast ?cast_code .
                    ?cast_code net:real_name ?cast .
                }
```

Figure 11: SELECT query - search for movies based on genres (1/2).

```
        OPTIONAL {
            ?show_id net:country ?country_code .
            ?country_code net:real_name ?country .
        }
        OPTIONAL {
            ?show_id net:date_added ?date_code .
            ?date_code net:real_name ?date_added .
        }
        OPTIONAL {
            ?show_id net:release_year ?release_code .
            ?release_code net:real_name ?release_year .
        }
        OPTIONAL {
            ?show_id net:rating ?rating_code .
            ?rating_code net:real_name ?rating .
        }
        OPTIONAL {
            ?show_id net:duration ?duration_code .
            ?duration_code net:real_name ?duration .
        }
        OPTIONAL {
            ?show_id net:listed_in ?genres_code .
            ?genres_code net:real_name ?genres .
        }
        OPTIONAL {
            ?show_id net:description ?description_code .
            ?description_code net:real_name ?description .
        }
    }
    GROUP BY ?type ?title ?director ?country ?date_added ?release_year ?rating ?duration ?description
```

Figure 12: SELECT query - search for movies based on genres (1/2).

Inserting and Deleting Data

Insertion and deletion of data on and from the database through SPARQL can be achieved with INSERT and DELETE query types. With this base functionalities we created the possibility of adding a movie/tv show information, deleting from a specific name and also an entire genre of movies and its movies.

Inserting a new movie/tv show

A movie/tv show as we have seen in the dataset has multiple parameters of information. To add a new one to the existing database, an extensive INSERT query was created in multiple steps.

It all begun with the query presented in Figure 13. From the information collected in the webpage was added to the query in way to allow the creation of *URI*s for each peace of information and a relation with its literal/numeric value through the “pred:real_name” *URI*.

```
# Start the INSERT DATA block
query = f"""
PREFIX pred: <http://ws.org/netflix_info/pred/>
PREFIX net: <http://ws.org/netflix_info/>
INSERT DATA {{
    net:{movie_id} pred:show_id "{movie_id}".
    net:{movie_id} pred:type net:{movie_type}.
    net:{movie_type} pred:real_name "{movie_type}".
    net:{movie_id} pred:title net:{name.replace(' ', '_')}.
    net:{name.replace(' ', '_')} pred:real_name "{name}".
    net:{movie_id} pred:director net:{director.replace(' ', '_')}.
    net:{director.replace(' ', '_')} pred:real_name "{director}".
"""
```

Figure 13: Insert query.

Alongside with information presented on Figure 13, we added the remaining information to the query following the same principles, as shown in Figure 14.

```
# Adding cast members
for actor in cast:
    actor_id = actor.strip().replace(' ', '_')
    query += f"net:{movie_id} pred:cast net:{actor_id}.\n"
    query += f"net:{actor_id} pred:real_name \"{actor.strip()}\".\n"

# Adding additional properties
query += f"""
    net:{movie_id} pred:country net:Country_{country.replace(' ', '_')}.
    net:Country_{country.replace(' ', '_')} pred:real_name "{country}".
    net:{movie_id} pred:date_added net:Date_{date_added.replace('-', '_')}.
    net:Date_{date_added.replace('-', '_')} pred:real_name "{date_added}".
    net:{movie_id} pred:release_year net:Year_{release_year}.
    net:Year_{release_year} pred:real_name "{release_year}".
    net:{movie_id} pred:rating net:Rating_{rating.replace(' ', '_')}.
    net:Rating_{rating.replace(' ', '_')} pred:real_name "{rating}".
    net:{movie_id} pred:duration net:Duration_{duration.replace(' ', '_')}.
    net:Duration_{duration.replace(' ', '_')} pred:real_name "{duration}".
"""

# Adding genres
for genre in genres:
    genre_id = genre.strip().replace(' ', '_')
    query += f"net:{movie_id} pred:listed_in net:Genre_{genre_id}.\n"
    query += f"net:Genre_{genre_id} pred:real_name \"{genre.strip()}\".\n"

query += f"net:{movie_id} pred:description net:Desc_{movie_id}.\n"
query += f"net:Desc_{movie_id} pred:real_name \"{description}\".\n"
query += "}\n"
```

Figure 14: Insert query - complimenting the query information to add to database

Deleting a name

Serving the purpose of deleting a given name from the database a simple DELETE query wase used (Figure 15).

```
query = """
            PREFIX pred: <http://ws.org/netflix_info/pred/>
            PREFIX net: <http://ws.org/netflix_info/>

            DELETE DATA {
                net:_encoded pred:real_name "_value" .
            }
        """
```

Figure 15: Delete query - delete name.

Here we replace "_encoded" and "_value" in the final query with the data submitted from the application.

After the data deletion, to certify the deletion was successful, we ASK the database for it with the query presented in Figure 16.

```
query = """
            PREFIX pred: <http://ws.org/netflix_info/pred/>
            PREFIX net: <http://ws.org/netflix_info/>

            ASK {
                net:_encoded pred:real_name "_value" .
            }
        """
```

Figure 16: ASK query - Check if name deletion was successfull.

Deleting an entire genre

Another data operation implemented in our application was the deletion of an entire genre. Doing so made us opt for the deletion of every movie within it. For that purpose, the query presented in Figure 17.

```
query = """
        PREFIX net: <http://ws.org/netflix_info/pred/>

        DELETE {
        ?show_id ?p ?o .
        }
        WHERE {
        ?genres_code net:real_name "_value" .
        ?show_id net:listed_in ?genres_code .

        ?show_id ?p ?o .
        }
    """
```

Figure 17: DELETE query - Deletion of every movie from a specific genre.

For the same reason as the previous deleting procedure, another ASK query was used to confirm the inexistence of any movie from the deleted genre. This time the ASK query works on the count of elements found for the deleted genre (Figure 18).

```
query = """
        PREFIX net: <http://ws.org/netflix_info/pred/>

        ASK
        WHERE {
        {
            SELECT (COUNT(?genres) AS ?count)
            WHERE {
            ?movie net:listed_in ?genres_code .
            ?genres_code net:real_name ?genres .
            FILTER (?genres = "_value")
            }
            GROUP BY ?genres
        }
        FILTER (?count > 0)
        }
    """
```

Figure 18: ASK query - checking deletion of an entire genre by counting elements.

Application Functionality

Homepage

When opening the website, we are presented with a homepage with the project title, a search bar and various recommendations (Figure 19).

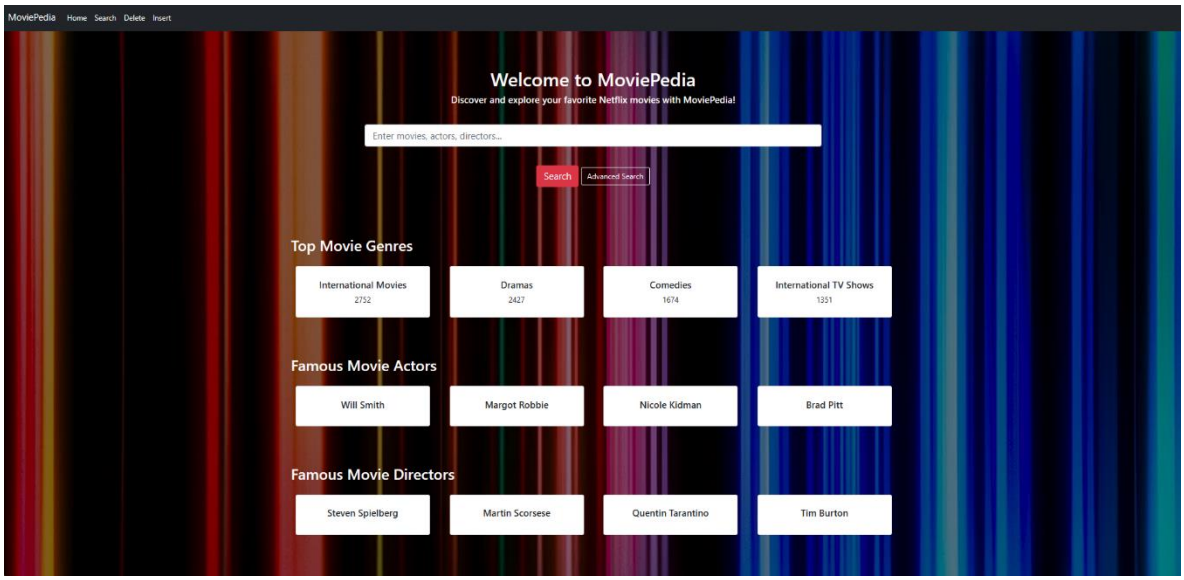

Figure 19: Homepage.

The search bar (Figure 20) provides us a more general search, that is, searching for films, films in which the searched actor acted, films by genre and films in which the searched director directed.

Here's an example of searching for the movies in which the actor "Johnny Depp" starred (scrolling down through the results) on Figure 21.

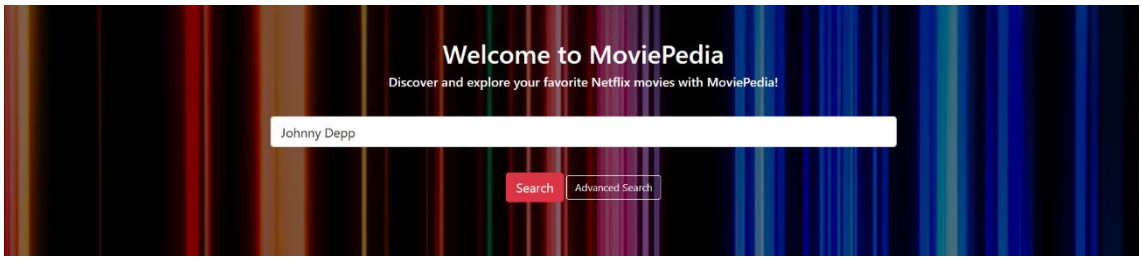

Figure 20: Homepage – search bar.

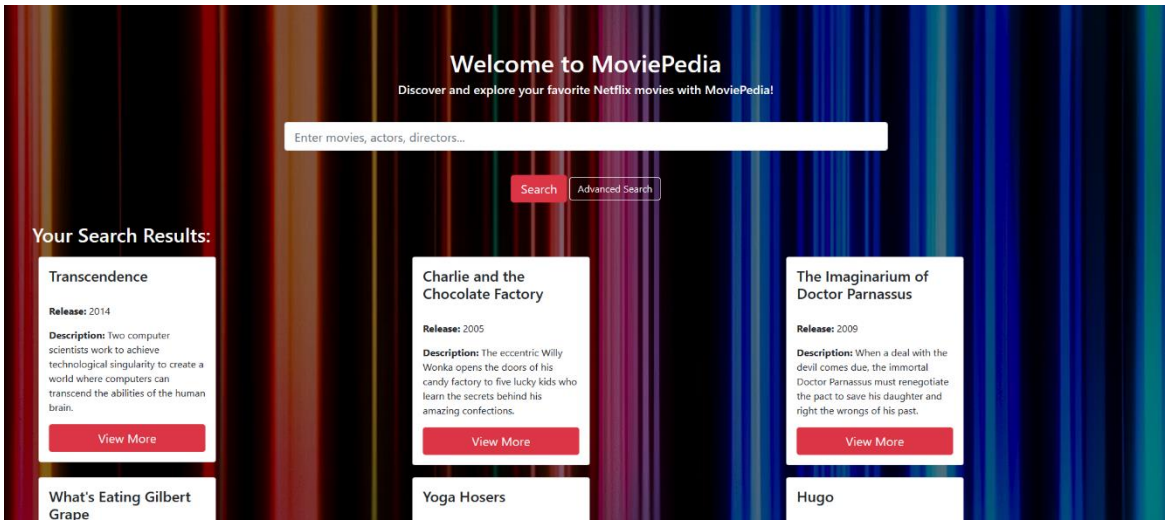

Figure 21: Homepage - search results.

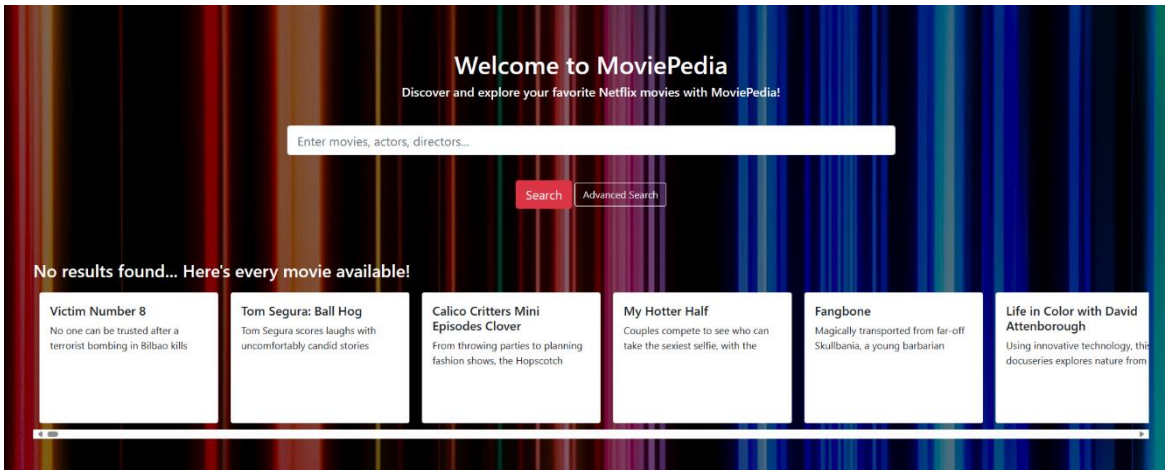

Figure 22: Homepage - no results found.

In case the name searched is not found in our catalog, the result will be the presented in Figure 22.

Advanced Search Page

This page contains a more detailed search for our movie catalog (Figure 23). On this page, there is more variety in the search.

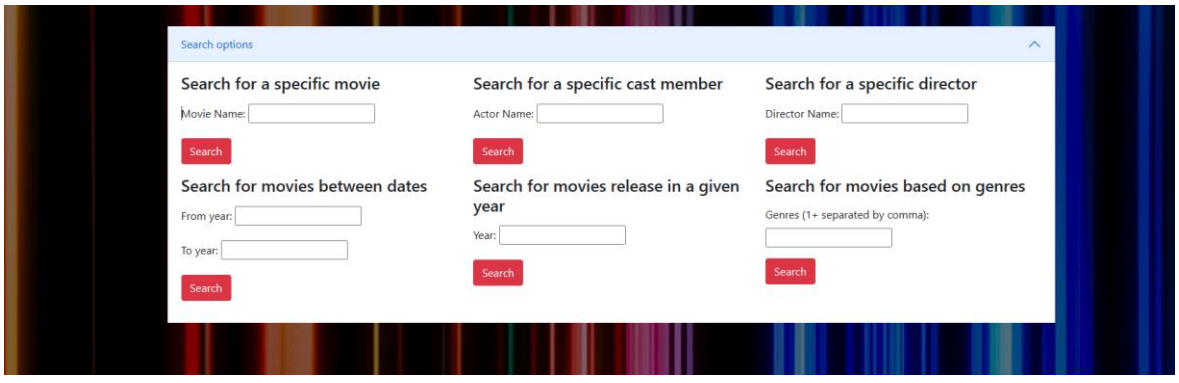

Figure 23: Advanced Search page.

In Figure 24 an example of searching for movies that were released in the year '2000'(scrolling right through the results) is presented.

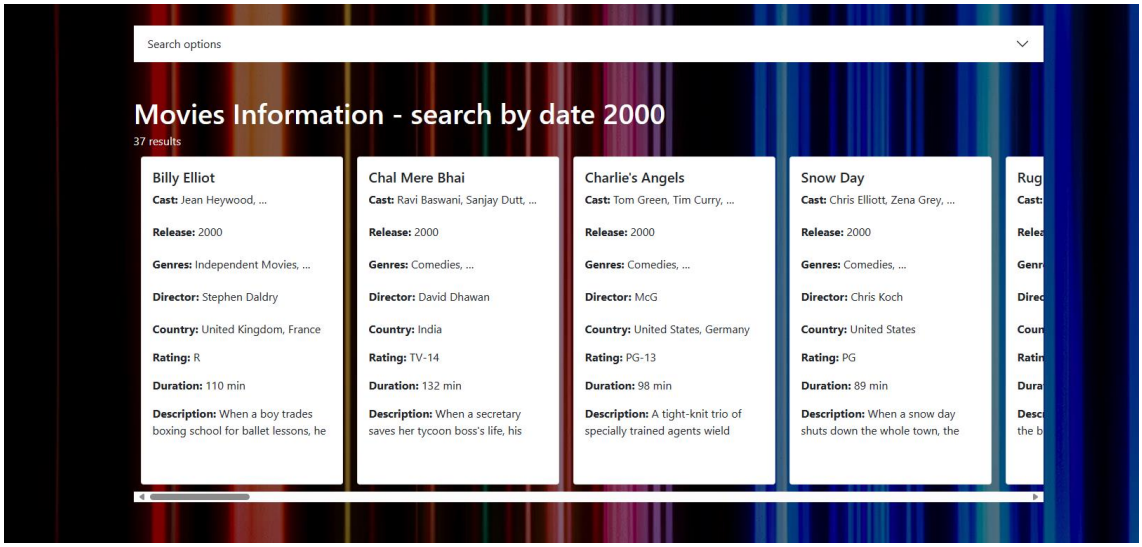

Figure 24: Advanced Search – search results.

Note: If the user clicks on one of the actors, genres, directors and the release date on the movie 'card', the page is redirected to the specific choice.

Delete Page

This page (Figure 25) provides the user the delete from database functionality. There's deletion by name, which is deleting the URI and the relations that it has with other entities (can be an actor, a genre, a movie, a director, the movie duration, release data, rating and country). Next, there's deletion by genre, which is deleting all movies that are related to the searched genre.

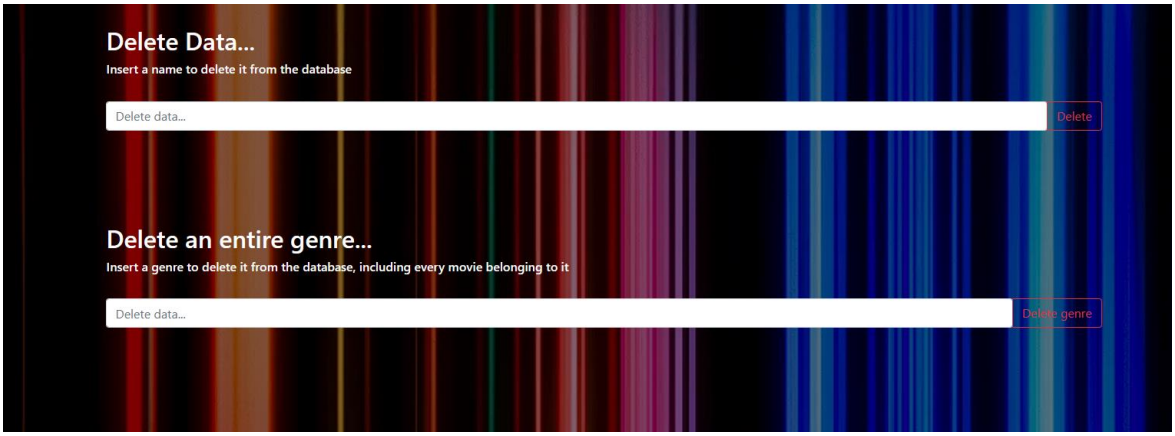

Figure 25: Delete page.

Insert Page

In this page, the user can insert a movie into our database, filling the form that is presented (Figure 26).

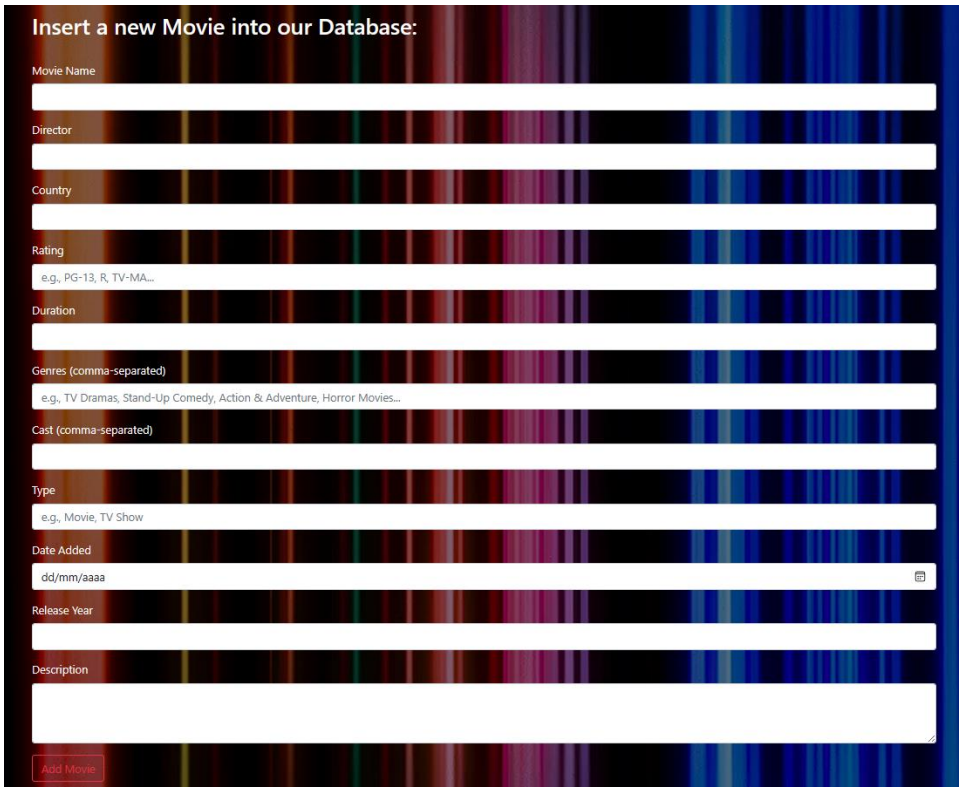

Figure 26: Insert page.

Conclusions

As we embarked on this project journey, we immersed ourselves in the world of cutting-edge technologies like Python/Django, RDF, *Triplestore GraphDB*, and SPARQL. It was a thrilling ride as we learned to wield these tools to transform our raw dataset into the RDF format, unlocking new possibilities in data management and representation.

One big thing we learned was how important it is to break our project into smaller parts. It made everything easier to manage and made our system more reliable.

Overall, this project gave us a solid understanding of how web-based information systems and their tech work together. It was a great learning experience, and it's got us excited to keep exploring and trying new things in this field.

Instructions to application execution

Initial Setup

1. **Create a virtual environment** - In the terminal, run the following commands:

   ```
   python3 -m venv .venv
   source .venv/bin/acivate
   ```

2. **Install requirements** - In the terminal, use the following command to install the dependencies listed in the requirements.txt file:

   ```
   pip install -r requirements.txt
   ```

GraphDB Database

1. **Download GraphDB:** Download GraphDB from https://graphdb.ontotext.com/.
2. **Create a repository:** In the GraphDB dashboard, go to "Configuration -> Repositories -> Create New Repository" and create a new repository, naming it "MoviePedia".
3. **Import data:** In the GraphDB dashboard, go to "Import -> Load RDF Files" and upload the netfli_titles.nt file, located in the "Data" folder, then click the "Import" button.

Running the Server

1. **Start the server** - In the terminal, run the following command to start the server:

   ```
   python manage.py runserver
   ```

References

[1] S. Bansal, "Netflix Movies and TV Shows," 2021. [Online]. Available: https://www.kaggle.com/datasets/shivamb/netflix-shows.