

# **Escuela de Ingenierías Industrial, Informática y Aeroespacial**

## **GRADO EN INGENIERÍA INFORMÁTICA**

Sistemas de Información de Gestión y Business Intelligence

Memoria de la aplicación: RECOMENDIT

Cristian De Nicola



## 1. INTRODUCCIÓN

En la presente memoria se detalla la aplicación web *RECOMENDIT*, desarrollada para la asignatura Sistemas de Información de Gestión y Business Intelligence (SIBI), perteneciente al primer semestre del cuarto curso del Grado en Ingeniería Informática de la Universidad de León.

A lo largo de la misma, se detallará el problema abordado por la aplicación, las herramientas empleadas para su desarrollo, los algoritmos implementados así como su funcionamiento a través de varios casos de uso. De forma complementaria, se realiza un análisis DAFO (Debilidades, Amenazas, Fortalezas y Oportunidades) del proyecto y se proponen varias líneas de futuro para completar su funcionalidad.

Finalmente, se concluye haciendo un repaso de las distintas lecciones y experiencia adquirida por mi persona a lo largo de la elaboración de este proyecto.



## 2. APLICACIÓN DESARROLLADA

En el siguiente apartado se realiza una descripción del diseño de la aplicación. Como ya se intuye a partir del punto anterior, podemos dividir la estructura de la aplicación en 3 partes, la base de datos, la parte servidora o *back-end*, y la parte correspondiente a la interfaz de usuario o *front-end*.

### 2.1. BASE DE DATOS

La base de datos consiste en un grafo con cuatro tipos de nodos o vértices: films (***Film***, en la base de RECOMENDIT), genres (***Genres***), *release\_year* (***release\_year***) y *Director* (***Director***).

### Node labels

\*(13,923) Director Film Genre Release\_year

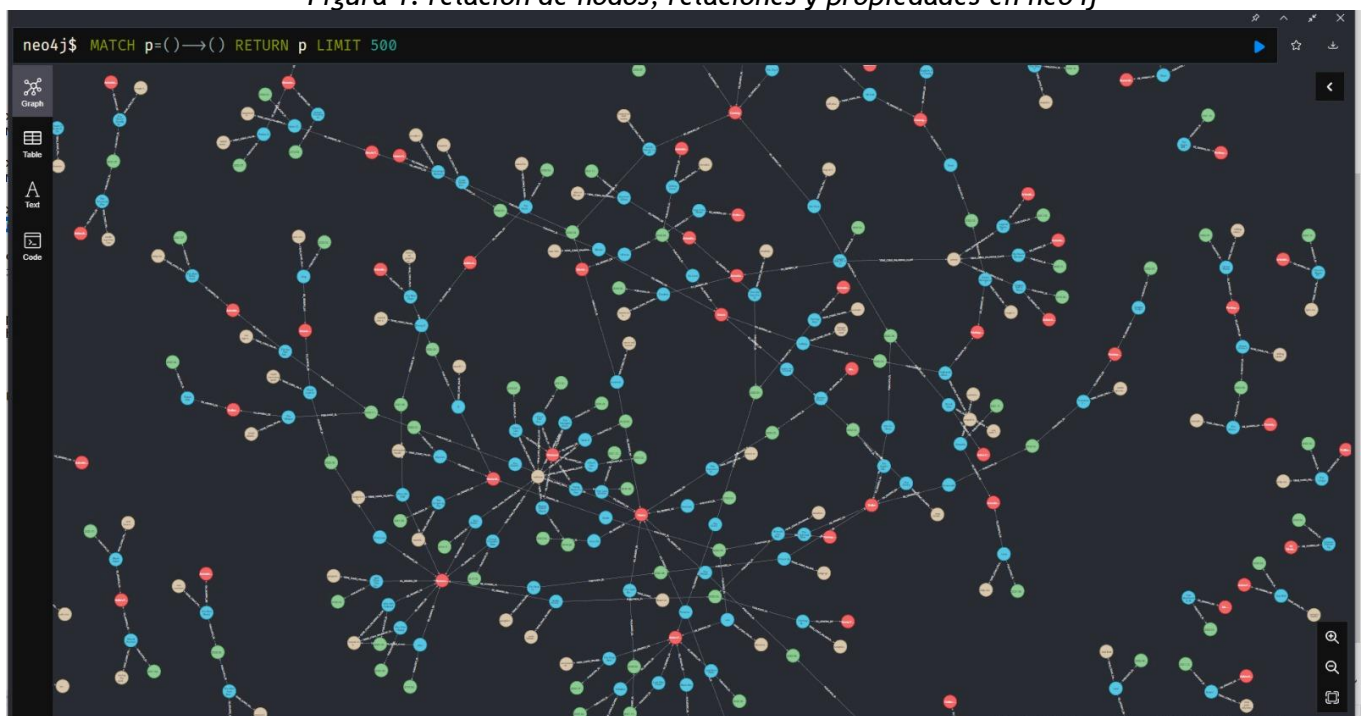
### Relationship types

\*(23,787) DIRIGIDA\_POR ES\_GENERO\_DE PUBLICADO\_EL

### Property keys

date\_added description director duration listed\_in rating release\_year show\_id title type

Figura 1: relación de nodos, relaciones y propiedades en neo4j



### 3. HISTORIA:

Recomendit tenía 3 versiones diferentes:

Lo recomiendo tenía 3 versiones diferentes: la primera en python (completada), la segunda en vue3 (desafortunadamente tuve que abandonar porque tuve problemas con un error de neo4j) y la última hecha en java.

Primera version: python.

en primer lugar, vamos a leer el conjunto de datos e importarlo al proyecto.

```
from sklearn.preprocessing import MinMaxScaler
import plotly.express as px
from sklearn.metrics.pairwise import cosine_similarity
from nltk.stem.porter import PorterStemmer

movies = pd.read_csv("C:\\Users\\crist\\OneDrive\\Desktop\\Dataset_progetto\\tmdb_5000_movies.csv")
credits = pd.read_csv("C:\\Users\\crist\\OneDrive\\Desktop\\Dataset_progetto\\tmdb_5000_credits.csv")

movies = movies.merge(credits, on='title')
movies.head(2)

#print (movies.shape) - se si vuole di nuovo vedere lo studio del dataset togliere da commentato -
#movies.info() - se si vuole di nuovo vedere lo studio del dataset togliere da commentato -
```

entonces vamos a tener una idea general de lo que contiene el ds.

```
24 #print (movies.shape) - se si vuole di nuovo vedere lo studio del dataset togliere da commentato -
25 #movies.info() - se si vuole di nuovo vedere lo studio del dataset togliere da commentato -
26
27 movies.isna().sum().to_frame()
28
29 def overview_dataset(df):
30     # Observations
31     print("Number of observation:", df.shape[0])
32     # Features
33     print("Number of features:", df.shape[1])
34     print("-"*60)
35     #Categorical Features
36     print("Categorical Features:", df.select_dtypes(include = [object]).columns)
37     print("")
38     # Numerical Features
39     print("Numerical Features:", df.select_dtypes(include = [int,float]).columns)
40
41 overview_dataset(movies)
42
```

aquí le damos peso a los valores que encontramos, a través de votos de usuarios y promedios.

```
movies[movies.select_dtypes(include=[object]).columns].describe().T

movies = movies.drop(columns = ['homepage', 'tagline', 'status', 'production_countries', 'release_date']) #elimino le colonne de
#mi servono

#movies.info() - se si vuole di nuovo vedere lo studio del dataset togliere da commentato -

v = movies['vote_count']
R = movies['vote_average']
C = movies['vote_average'].mean()
m = movies['vote_count'].quantile(0.70)

movies['Weighted_avg'] = ((R*v)+(C*m)) / (v+m)

movies_sorted = movies.sort_values('Weighted_avg', ascending = False)
movies_sorted[['original_title', 'vote_count', 'vote_average', 'Weighted_avg', 'popularity']]
```

aquí vamos a combinar los distintos pesos para tener una evaluación final con el fin de unir el sistema de recomendación

```
98
99 #non possiamo usare questi due valori separatamente, quindi dobbiamo usarli assieme
100 #uso la lib sklearn.preprocessing
101 scale = MinMaxScaler()
102 normalized_df = scale.fit_transform(movies[['Weighted_avg', 'popularity']])
103 scaled_df = pd.DataFrame(normalized_df, columns = ['Weighted_avg', 'popularity'])
104
105 movies[['scaled_weighted_avg', 'scaled_popularity']] = scaled_df
106 movies.head(5)
107
108 #creo un punteggio score (basato su scale creato in precedenza)
109 movies['score'] = movies['scaled_weighted_avg'] * 0.5 + movies['scaled_popularity'] * 0.5
110 movies_scored = movies.sort_values('score', ascending = False)
111 movies_scored[['original_title', 'scaled_weighted_avg', 'scaled_popularity', 'score']].head(10)
112
113
```

aquí vamos a hacer las primeras consultas para obtener los principales datos que necesitamos

```

147 a=ast.literal_eval('{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}, {"id": 14, "name": "Fantasy"}, {"id": 878, "name": "Scien
148 print(type(a))
149 a[1]['name']
150
151 ## funzione che prende il nome
152 def convert(obj):
153     l=[]
154     for i in ast.literal_eval(obj):
155         l.append(i['name'])
156     return l
157
158 ##funzione che prende il nome del direttore del film
159 def director(obj):
160     l=[]
161     for i in ast.literal_eval(obj):
162         if i['job']=='Director':
163             l.append(i['name'])
164     return l

```

aquí vamos a aplicar una función lamda para eliminar los espacios entre los distintos datos y administrar el resto

```

movies['genres']=movies['genres'].apply(convert)
movies['keywords']=movies['keywords'].apply(convert)
movies['cast']=movies['cast'].apply(convert)
movies['crew']=movies['crew'].apply(director)

movies_genres=movies.explode("genres")
movies_genres

count_genres = movies_genres.groupby('genres').count()[['movie_id']].sort_values(by='movie_id', ascending=False)
print(count_genres)
count_genres.plot(kind='pie',figsize=(10,10),legend=False,subplots=True)
plt.title('Number of movies per genre')
plt.show()
movies

movies['overview']=movies['overview'].apply(lambda x:x.split())
movies.iloc[0].overview

#funzione che rimuove lo spazio tra le stringhe del dataset
def remove_space(string):
    l=[]
    for i in string:
        l.append(i.replace(" ",""))
    return l

```

Voy a aplicar la función y empezar a hacer un join luego pongo todas las cadenas del datset en minúsculas para evitar problemas durante la búsqueda

```
#applico la funzione
movies['genres']=movies['genres'].apply(remove_space)
movies['keywords']=movies['keywords'].apply(remove_space)
movies['cast']=movies['cast'].apply(remove_space)
movies['crew']=movies['crew'].apply(remove_space)

movies.head()

movies['tags']=movies['overview']+movies['genres']+movies['keywords']+movies['cast']+movies['crew']

movies.head()

new_movies=movies.drop(columns=['overview','genres','keywords','cast','crew'])

new_movies.head()

#unisco con la join e poi metto tutte le stringhe del dataset in lower case per non avere problemi durante la ricerca
new_movies['tags']=new_movies['tags'].apply(lambda x:" ".join(x))
new_movies['tags']=new_movies['tags'].apply(lambda x:x.lower())
new_movies.head()

ps=PorterStemmer()
```

aquí empiezo a aplicar la función de similitud para empezar a encontrar las diversas similitudes entre los diversos datos de la ds.

```
#funzione che inizia a trovare le varie cose in comune tra i vari film, lavorerà con la tabella new movies
def stem(text):
    y=[]
    for i in text.split():
        y.append(ps.stem(i))
    return " ".join(y)

new_movies['tags']=new_movies['tags'].apply(stem)
new_movies.head()

cv = CountVectorizer(max_features = 2000)
V=cv.fit_transform(new_movies['tags']).toarray()
V

#qui si trova il valore di similitudine tra i vari film del nuovo dataset
similarity=cosine_similarity(V)
similarity
```



finalmente, como última cosa, aplico todo a la función final que, tomando como entrada un título, encuentra las 10 mejores películas que coinciden con la indicada.

```
def recommend(movie):
    index = new_movies[new_movies['original_title']== movie].index[0]
    movie_name = sorted(list(enumerate(similarity[index])), reverse= True, key =lambda x:x[1])
    l=[]
    for i in movie_name[1:10]:
        a=new_movies.iloc[i[0]].original_title
        l.append(a)

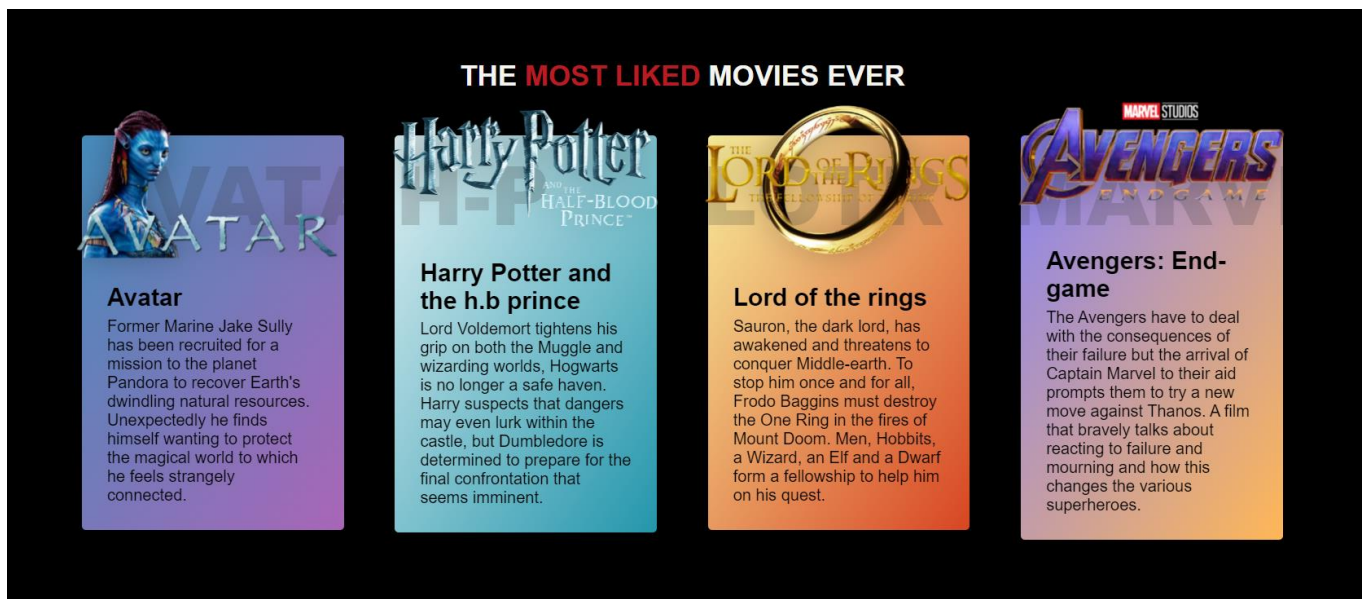
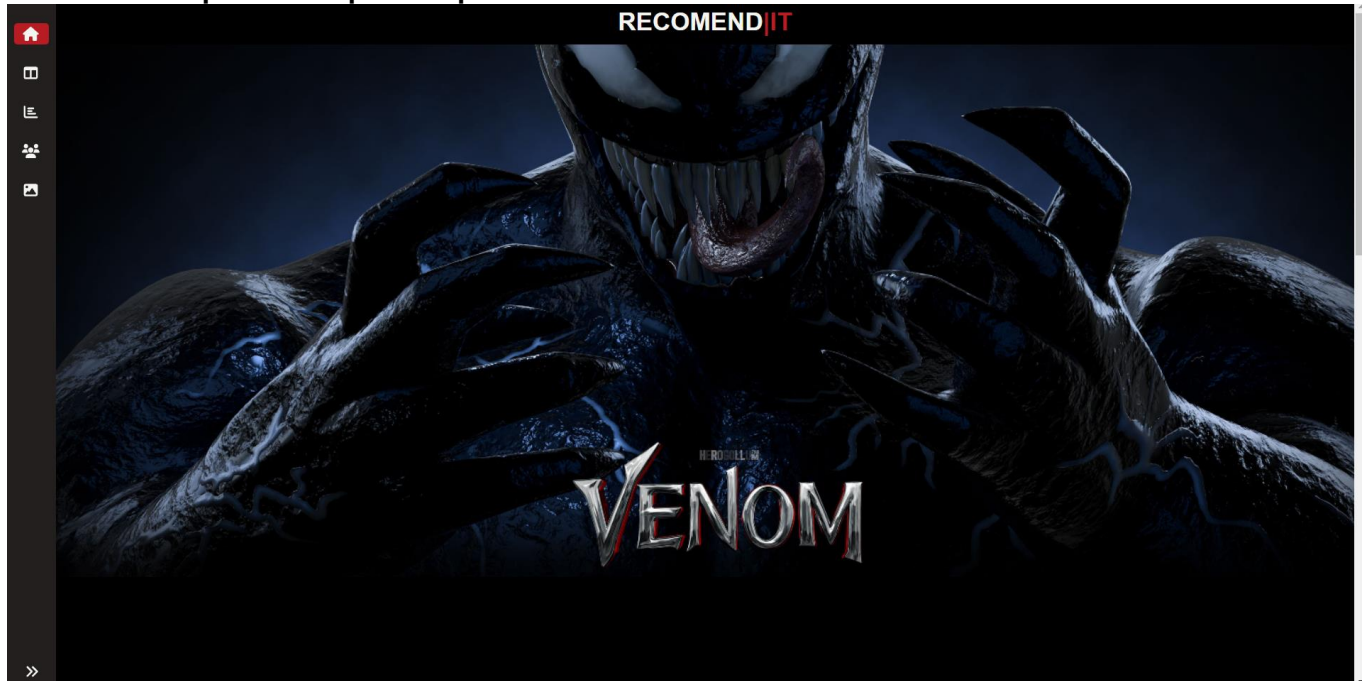
    rec_movies=pd.DataFrame(l)
    rec_movies=rec_movies.set_axis(['Recommend Movies'], axis=1, inplace=False)
    print(rec_movies)

film = input ("inserisci un film: ")
print (film)


recommend(film)

pickle.dump(new_movies,open('movies_list.pkl','wb'))
pickle.dump(similarity,open('smilarity.pkl','wb'))
```

2: segunda version: Vue3.  
Esta es la patalla principal.



## BEST MOVIES OF 2022




**Doctor Strange**

To restore a world where everything is changing, Strange seeks help from his ally Wong, the Sorcerer Supreme, and the Avengers' most powerful Scarlet Witch, Wanda. But a terrible threat looms over humanity and the entire universe that no longer can be done by their power alone.




**Black adam**

Nearly 5,000 years after being bestowed with the all-powerful powers of the Egyptian gods and just as quickly imprisoned, Black Adam is freed from his earthly grave, ready to unleash his justice upon the modern world.



**One Piece: Red**

The Straw Hats journey to Elegia island to see the iconic Uta in concert. But it's not all music and games as quickly chaos breaks out with the Big Mom pirates, Navy, Heart pirates, and more joining the fray. Luffy and Uta are old friends, but lives are at risk.



**Thor love and thunder**

Thor's quest for a dimension of inner peace is interrupted by the arrival of the galactic killer Gorr the God Butcher, bent on destroying all gods. Thor will thus join forces with Valkyrie, Korg and ex-girlfriend Jane Foster who will demonstrate his uniqueness to him.

```

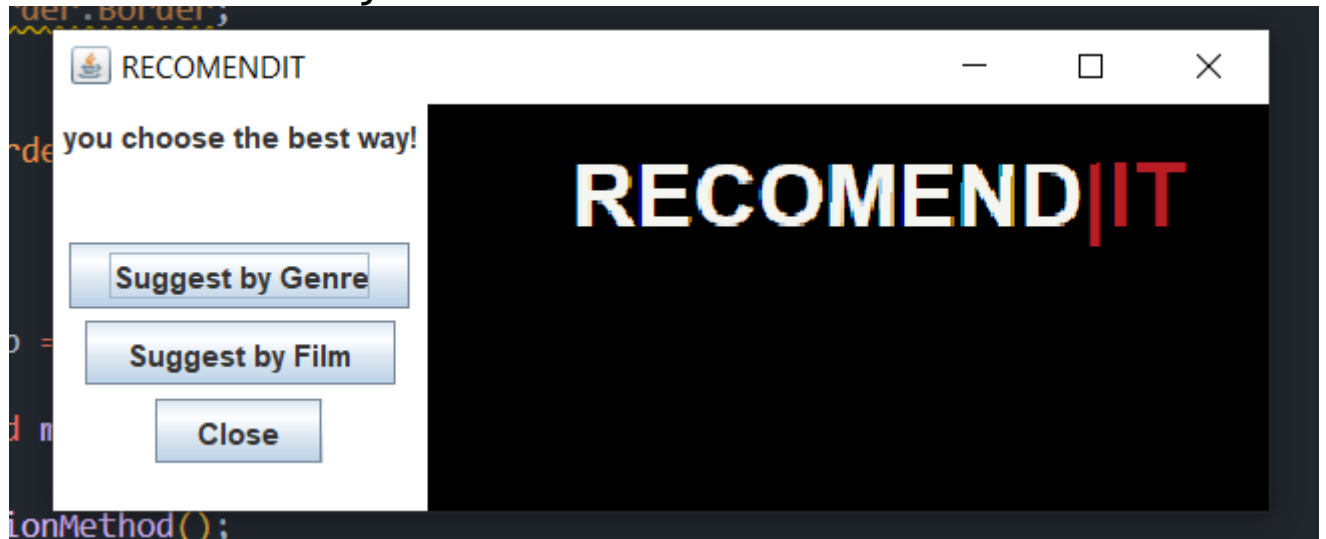
</div>
</template>

<script>
  //import { RouterView } from "vue-router"
  import Sidebar from '@components/sidebar/Sidebar.vue'
  import { sidebarWidth } from '@components/sidebar/state.js'

  export default {
    components: { Sidebar },
    setup() {
      return { sidebarWidth }
    }
  }
</script>

```

3: ultima version: java.



aquí creo la ventana principal donde puedo decidir si elegir de categoría o de título

```
private static void selectSuggestionMethod()
{
    final JFrame frame = new JFrame(title: "Film Suggestion Menu");

    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(width: 200, height: 500);
    frame.setLocationRelativeTo(c: null);

    final JPanel panel = new JPanel();
    panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
    panel.setBorder(BorderFactory.createEmptyBorder(top: 10, left: 10, bottom: 10, right: 10));

    JButton suggestCategoryButton = new JButton(text: "Suggest by Genre");
    suggestCategoryButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            frame.setVisible(b: false);
            showCategories();
        }
    });
    panel.add(suggestCategoryButton);
    panel.add(Box.createHorizontalGlue());

    JButton suggestFilmButton = new JButton(text: "Suggest by Film");
    suggestFilmButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            frame.setVisible(b: false);
            showFilms();
        }
    });
    panel.add(suggestFilmButton);
    panel.add(Box.createHorizontalGlue());
}
```

aquí está el fragmento de código que maneja la parte de las películas

```
private static void showFilms() {
    final JFrame frame = new JFrame(title: "Film Suggestion");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(width: 800, height: 500);
    frame.setLocationRelativeTo(c: null);

    final JPanel panel = new JPanel();
    panel.setLayout(new BorderLayout(panel, BorderLayout.Y_AXIS));
    panel.setBorder(BorderFactory.createEmptyBorder(top: 10, left: 10, bottom: 10, right: 10));

    final List < String > films = collectFilms();
    DefaultListModel < String > filmsListModel = new DefaultListModel < > ();
    for (String film: films) {
        filmsListModel.addElement(film);
    }
    final JList < String > filmsList = new JList < > (filmsListModel);
    JScrollPane scrollPane = new JScrollPane(filmsList);
    panel.add(scrollPane);

    final JTextField searchBar = new JTextField();
    searchBar.setMaximumSize(new Dimension(width: 1300, height: 25));

    panel.add(searchBar);
    searchBar.getDocument().addDocumentListener(new DocumentListener() {
        public void changedUpdate(DocumentEvent e) {
            filterList();
        }
        public void removeUpdate(DocumentEvent e) {
            filterList();
        }
    });
}
```

```
}  
final JList < String > filmsList = new JList < > (filmsListModel);  
JScrollPane scrollPane = new JScrollPane(filmsList);  
panel.add(scrollPane);  
  
final JTextField searchBar = new JTextField();  
searchBar.setMaximumSize(new Dimension(width: 1300, height: 25));  
  
panel.add(searchBar);  
searchBar.getDocument().addDocumentListener(new DocumentListener() {  
    public void changedUpdate(DocumentEvent e) {  
        filterList();  
    }  
    public void removeUpdate(DocumentEvent e) {  
        filterList();  
    }  
    public void insertUpdate(DocumentEvent e) {  
        filterList();  
    }  
})
```

aquí en cambio es la parte que gestiona la parte de categorías:

```
private static void showCategories() {  
    final JFrame frame = new JFrame(title: "Film Suggestion");  
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    frame.setSize(width: 800, height: 500);  
    frame.setLocationRelativeTo(c: null);  
  
    final JPanel panel = new JPanel();  
    panel.setLayout(new BorderLayout(panel, BorderLayout.Y_AXIS));  
    panel.setBorder(BorderFactory.createEmptyBorder(top: 10, left: 10, bottom: 10, right: 10));  
  
    List < String > categories = collectCategories();  
    DefaultListModel < String > categoriesListModel = new DefaultListModel < > ();  
    for (String category: categories) {  
        categoriesListModel.addElement(category);  
    }  
  
    final JList < String > categoriesList = new JList < > (categoriesListModel);  
    JScrollPane scrollPane = new JScrollPane(categoriesList);  
    panel.add(scrollPane);  
  
    //OK  
    //OK  
    //OK  
  
    //Suggestion button  
    JButton suggestButton = new JButton(text: "Suggest");
```





```
//suggestion button
JButton suggestButton = new JButton(text: "Suggest");
suggestButton.addActionListener(new ActionListener() {
@Override
public void actionPerformed(ActionEvent e) {
    String selectedCategory = categoriesList.getSelectedValue();
    final List < String > films = requestSuggestion(selectedCategory);
    frame.setTitle(selectedCategory + " Films");
    panel.removeAll();

    //create a new JList with the updated list of strings
    DefaultListModel < String > newStringListModel = new DefaultListModel < > ();
    for (String film: films) {
        newStringListModel.addElement(film);
    }
    final JList < String > suggestedFilms = new JList < > (newStringListModel);
    JScrollPane scrollPane = new JScrollPane(suggestedFilms);
    panel.add(scrollPane);

    final JTextField searchBar = new JTextField();
    searchBar.setMaximumSize(new Dimension(width: 1300, height: 25));

    panel.add(searchBar);
    searchBar.getDocument().addDocumentListener(new DocumentListener() {
        public void changedUpdate(DocumentEvent e) {
            filterList();
        }
        public void removeUpdate(DocumentEvent e) {
```

```
panel.add(searchBar);
searchBar.getDocument().addDocumentListener(new DocumentListener() {
    public void changedUpdate(DocumentEvent e) {
        filterList();
    }
    public void removeUpdate(DocumentEvent e) {
        filterList();
    }
    public void insertUpdate(DocumentEvent e) {
        filterList();
    }
}

    public void filterList() {
        String filterText = searchBar.getText();
        List<String> filteredFilms = new ArrayList<>();
        for (String film : films) {
            if (film.toLowerCase().contains(filterText.toLowerCase())) {
                filteredFilms.add(film);
            }
        }
        DefaultListModel<String> newFilmsListModel = new DefaultListModel<>();
        for (String film : filteredFilms) {
            newFilmsListModel.addElement(film);
        }
        suggestedFilms.setModel(newFilmsListModel);
        panel.revalidate();
```



```
        JButton seeMoreButton = new JButton(text: "See More");
        seeMoreButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String selectedFilm = suggestedFilms.getSelectedValue();
                String descriptionRetrieved = retrieveDescription(selectedFilm);

                //frame.setVisible(false);
                showFilmDescription(descriptionRetrieved);
            }
        });
        panel.add(seeMoreButton);

        JButton backButton = new JButton(text: "Back");
        backButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                frame.setVisible(b: false);
                showCategories();
            }
        });
        panel.add(backButton);

        JButton exitButton = new JButton(text: "Exit");
        exitButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                db.terminate();
            }
        });
        panel.add(exitButton);
    }
}
```

este es el método que genera la descripción de la película seleccionada



```
private static void showFilmDescription(String descriptionRetrieved) {
    final JFrame frame = new JFrame(title: "Book Suggestion");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(width: 800, height: 500);
    frame.setLocationRelativeTo(c: null);

    final JPanel panel = new JPanel();
    panel.setLayout(new BorderLayout(panel, BorderLayout.Y_AXIS));
    panel.setBorder(BorderFactory.createEmptyBorder(top: 10, left: 10, bottom: 10, right: 10));

    JLabel label = new JLabel(descriptionRetrieved);
    StringBuilder strBuilder = new StringBuilder();
    strBuilder.append(descriptionRetrieved + "<br>");
    label.setText("<html>" + strBuilder.toString() + "</html>");
    panel.add(label);

    JButton backButton = new JButton(text: "Back");
    backButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {

            frame.setVisible(b: false);
            //showBooks();
        }
    });
    panel.add(backButton);
}
```

```
private static List < String > collectCategories() {
    return db.retrieveGenres();
}

private static List < String > requestSuggestion(String category) {
    return db.suggestFilm(category);
}

private static String retrieveDescription(String selectedBook)
{
    return db.retrieveDescription(selectedBook);
}

private static String getFilmCategory(String bookTitle) {
    return db.getFilmCategory(bookTitle);
}

private static List < String > collectFilms() {
    return db.retrieveFilms();
}
}
```

aquí conectamos la aplicación a la base de datos a través de los controladores neo4j

```
public class Database {
    static Driver driver = GraphDatabase.driver(uri: "neo4j+s://d33b7603.databases.neo4j.io", AuthTokens.basic(username: "neo4j", password: "A9I37u_6aVW3hXQML4ot_vNxfU6Zsj"));
    static Session session = driver.session();

    List<String> retrieveGenres()
    {
        int index = 0;
        List<String> genres = new ArrayList<String>();

        Result result = session.run(query: "MATCH (g:Genre) RETURN g.listed_in as genre");
        while (result.hasNext()) {
            Record record = result.next();
            genres.add(index, record.get(key: "genre").asString());
            index++;
        }

        Collections.sort(genres);
        return genres;
    }
}
```

aquí tenemos los principales métodos del sistema de recomendación, por lo que aquellos que toman los datos como entrada devolverán las mejores películas al usuario

```
List<String> suggestFilm(String category)
{
    List<String> films = new ArrayList<String>();
    List<Integer> rating = new ArrayList<Integer>();
    System.out.println("Here are some " + category + " books that you might like:");
    System.out.println(x: "-----");

    Result result = session.run("MATCH (f:Film)-[ES_GENERO_DE]->(g:Genre {listed_in: \"" + category + "\"}) RETURN f.title AS title, f.rating AS stars");
    while (result.hasNext()) {
        Record record = result.next();
        films.add(record.get(key: "title").asString());
        rating.add(record.get(key: "rating", defaultValue: 0));
    }

    //Sorting based on rating of the books
    final Map<String, Integer> filmRatingMap = new HashMap<>();
    for(int i = 0; i < films.size(); i++) {
        filmRatingMap.put(films.get(i), rating.get(i));
    }
    Collections.sort(films, new Comparator<String>() {
        @Override
        public int compare(String o1, String o2) {
            return filmRatingMap.get(o1) - filmRatingMap.get(o2);
        }
    });

    return films;
}
```

```
String retrieveDescription(String bookTitle)
{
    Result result = session.run("MATCH (f:Film {title: \"" + bookTitle + "\"}) RETURN f.description AS description");
    Record record = result.next();

    return record.get(key: "description").asString();
}

String getFilmCategory(String bookTitle)
{
    Result result = session.run("MATCH (g:Genre)-[ES_GENERO_DE]->(f:Film {title: \"" + bookTitle + "\"}) RETURN g.listed_in AS category");

    Record record = result.next();

    return record.get(key: "category").asString();
}
```

```
List<String> retrieveFilms()
{
    int index = 0;
    List<String> films = new ArrayList<String>();

    Result result = session.run(query: "MATCH (f:Film) RETURN f.title as title");
    while (result.hasNext()) {
        Record record = result.next();
        films.add(index, record.get(key: "title").asString());
        index++;
    }

    Collections.sort(films);
    return films;
}

//close program
static void terminate()
{
    session.close();
    driver.close();
    System.exit(status: 0);
}
```

## 4. CONCLUSIONES

en este párrafo hablaremos sobre las ventajas y desventajas de la aplicación, los problemas futuros que podrá enfrentar y cómo moverse.

### VENTAJAS Y DESVENTAJAS:

las principales ventajas de la aplicación son un conjunto de datos muy completo y completo, la simplicidad y la velocidad de la interfaz de usuario.

sin embargo, las principales desventajas son que se refiere solo a títulos de Netflix.

### POSIBLES DESARROLLOS FUTUROS:

en un futuro cercano podríamos implementar una mejor interfaz y encontrar una base de datos mejor (más completa) o intente comprender mejor neo4j, tratando de resolver el error que detuvo el desarrollo 2 (vue3).

## lo que aprendí de este curso:

en este curso aprendí a usar neo4j y chipher, entendiendo como moverme en este tipo de base de datos.  
No ocultaré que tuve dificultades en algunos momentos, pero también fue bueno para eso.

otras cosas que aprendí fue cómo crear un sistema de recomendación y cómo se ve un filtro de burbujas.  
Tema muy útil en mi opinión también para posibles desarrollos futuros.

Gracias,  
Cristian De Nicola.