OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG

Applied Data Science with Python – Mr. Lengyel – SoSe23

Assignment for Task 3

Applied Data Science with Python

IMDB Software of Hollywood Actors and Actresses

Advisor: Istvan Lengyel
Students: Nguyễn Quốc Minh Thư - 3397231.

REGENSBURG, May 2023

Contents

1 Requirement Elicitation

1.1 Project Identify

Requirement: IMDB Software of Hollywood Actors and Actresses

♣ IDENTIFY THE CONTEXT OF THIS PROJECT:

The project involves creating a user-friendly software application that focuses on storing and extracting information about the top 50 popular Hollywood actors and actresses. The data will be sourced from the IMDb website and will include details such as actor/actress names, movie information, awards, genres, and ratings.

♣ WHO ARE RELEVANT SKATEHOLDERS ?

- Users: People who will be utilizing the software to access information about Hollywood actors and actresses.
- IMDb: The IMDb website, which serves as the primary source of data for the software.

♣ WHAT ARE THEIR CURRENT NEEDS ?

User Stories:

- **Users:** Users need a user-friendly software application that provides easy access to information about popular Hollywood actors and actresses. They seek comprehensive data, including movie names, years, awards, genres, and ratings, to explore the filmography and achievements of their favorite actors/actresses.
- **IMDb:** IMDb benefits from increased visibility and recognition as the primary data source for the software. It may also benefit from increased traffic to their website as users navigate to IMDb for more detailed information.

♣ WHAT BENEFITS OF THIS APPLICATION WILL BE FOR EACH STAKEHOLDER?

- **For User:**
  1. sers will have access to a user-friendly software application that provides comprehensive information about their favorite Hollywood actors and actresses.
  2. They can explore movie details, awards, genres, and ratings conveniently, enhancing their knowledge and entertainment experience.
- **For IMDb:**
  1. IMDb benefits from increased exposure and usage as the primary data source for the software.
  2. The software can potentially drive more traffic to IMDb's website, resulting in increased visibility and potential ad revenue.

1.2 PROBLEM-SOLVING APPROACH

In brief: To solve the problem, the following algorithm, tools, modules, and data structures can be considered:

1.2.1 ALGORITHM:

Requirement: The software should consist of various functions to handle different user requests, such as retrieving actor/actress information, movie details, awards, and ratings. The algorithm will involve parsing HTML data from the IMDb website, extracting relevant information, storing it in appropriate data structures, and presenting it to the user.

♣ DESIRED FUNCTIONALITIES:
Based on the requirements, there are some functions I will use for my IMDB character software to satisfy the desired functionality:

1. **listActorsAndActresses():** This function will list all the available actors and actresses from the provided IMDb link.
2. **getActorDetails(actorName):** This function will provide information about a specific actor/actress, such as their biography, birthdate, birthplace, and other relevant details.
3. **getMovies(actorName):** This function will retrieve the list of all-time movie names and years for a particular actor/actress.
4. **getAwards(actorName):** This function will fetch the awards received by the actor/actress in different years, including the name of the award and the corresponding movie.
5. **getMovieGenres(actorName):** This function will return the movie genres associated with a specific actor/actress.
6. **getAverageRating(actorName):** This function will calculate the average rating of all the movies of a particular actor/actress, both overall and each year separately.
7. **getTopMovies(actorName):** This function will provide the top 5 movies of an actor/actress, along with their respective years and genres.

These functions can be implemented and hopefully it can fulfill the requirements and provide a user-friendly and well-formatted interface for accessing information about Hollywood actors and actresses.

♣ DATA RETRIEVE FUNCTIONALITIES:
To retrieve the required data from IMDb, I consider using web scraping techniques to extract information from the IMDb website. Here are some backend functions that I may use to achieve this:

1. **'scrapeActorList()'**: This function will scrape the IMDb website and extract the list of actors and actresses along with their IMDb profile URLs.
2. **'scrapeActorDetails(actorURL)'**: This function will scrape the IMDb actor/actress profile page using the provided IMDb profile URL and extract details such as biography, birthdate, birthplace, and other relevant information.
3. **'scrapeMovies(actorURL)':** This function will scrape the IMDb actor/actress profile page to retrieve the list of movies they have been a part of, including the movie name and release year.
4. **'scrapeAwards(actorURL)':** This function will scrape the IMDb actor/actress profile page to extract the awards they have received, along with the corresponding years and movie titles.
5. **'scrapeMovieGenres(movieURL)':** This function will scrape the IMDb movie page using the provided IMDb movie URL and extract the genre(s) associated with that movie.
6. **'scrapeMovieRating(movieURL)':** This function will scrape the IMDb movie page to retrieve the movie rating and any additional rating information.
7. **'scrapeTopMovies(actorURL)':** This function will scrape the IMDb actor/actress profile page to identify their top-rated movies based on ratings or popularity metrics.

1.3 Tools and technologies

To develop the IMDb actor app, the following tools and technologies are those I am considering:

- **Front-End Framework:** React.js
- **Back-End Framework:** Django
- **Database:** PostgreSQL
- **Web Scraping:** Python with Beautiful Soup and request
- **UI Components:** Bulma library

1.3.1 Data Structure

The IMDb database for Hollywood actors and actresses can be represented using a relational data structure. The structure consists of the following tables/entities:

- **Actor/Actress Table**:
  - `actor_id` (Primary Key): Unique identifier for each actor/actress.
  - `actor_name`: Name of the actor/actress.
  - `biography`: Biography of the actor/actress.
  - `birthdate`: Birthdate of the actor/actress.
  - `birthplace`: Birthplace of the actor/actress.
- **Movie Table**:
  - `movie_id` (Primary Key): Unique identifier for each movie.
  - `movie_name`: Name of the movie.
  - `release_year`: Year of movie release.
  - `average_rating`: Average rating of the movie.
- **ActorMovie Table**:
  - `actor_id` (Foreign Key referencing Actor/Actress Table): Identifies the actor/actress.
  - `movie_id` (Foreign Key referencing Movie Table): Identifies the movie.
- **Award Table**:
  - `award_id` (Primary Key): Unique identifier for each award.
  - `award_name`: Name of the award.
  - `year`: Year in which the award was received.
  - `movie_id` (Foreign Key referencing Movie Table): Identifies the movie.
  - `actor_id` (Foreign Key referencing Actor/Actress Table): Identifies the actor/actress.
- **Genre Table**:
  - `genre_id` (Primary Key): Unique identifier for each genre.
  - `genre_name`: Name of the genre.
- **MovieGenre Table**:
  - `movie_id` (Foreign Key referencing Movie Table): Identifies the movie.
  - `genre_id` (Foreign Key referencing Genre Table): Identifies the genre.

This relational data structure allows for the organization and querying of data related to actors, movies, awards, and genres in the IMDb database.

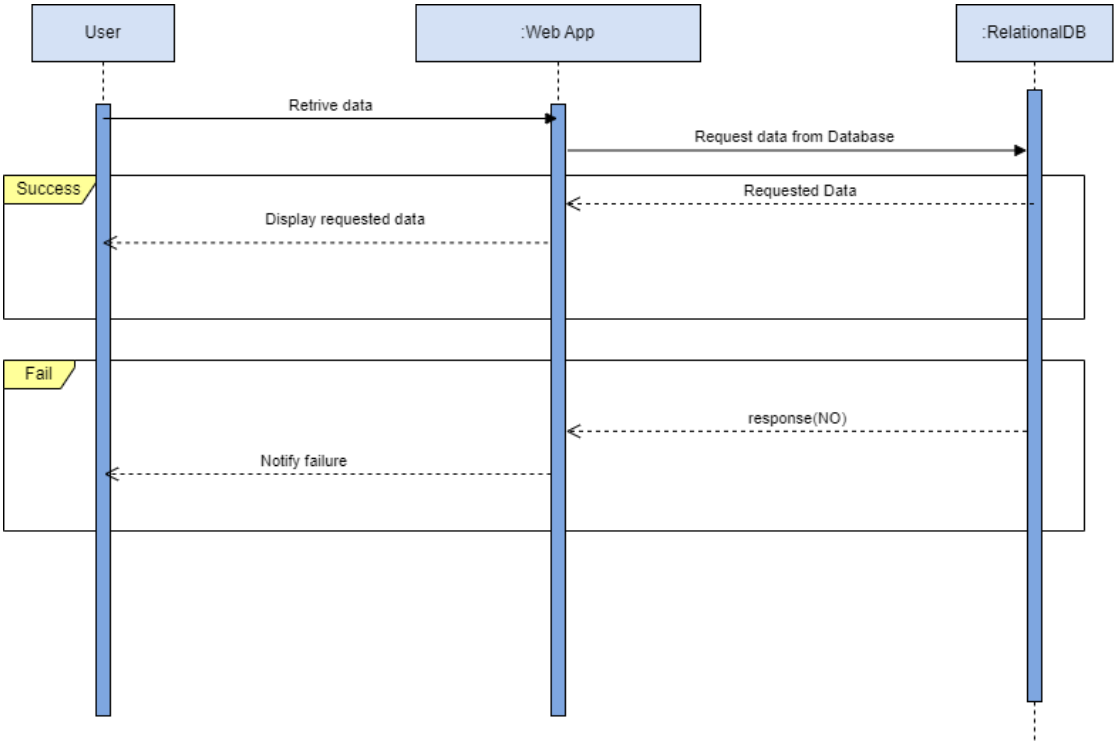
2 System modelling

In Brief: In this step of project, you have to provide a small write up (min 1 page) about the partial result of your project. Your write up should clearly state the part of the project you would like to show along with the pseudo code (or flowchart).

2.1 Sequence Diagram

In Brief: Show the sequence of some basic functionality of the software.

2.1.1 General sequence diagram:



2.2 Description of detail sequence :

1. **List of all available actors and actresses:**

- User selects "List Actors/Actresses"option.
- System retrieves and displays a list of all actors and actresses.

2. **About the actor/actresses:**

- User selects a specific actor/actress from the list.
- System retrieves the details of the selected actor/actress.
- System displays the actor/actress profile page with the retrieved information.

3. **All-time movie names and years:**

- User selects a specific actor/actress from the list.
- System retrieves the movies associated with the selected actor/actress.
- System displays the list of movie names and release years.

4. **Awards to actor/actresses in different years:**

- User selects a specific actor/actress from the list.
- System retrieves the awards received by the selected actor/actress.
- System displays the awards categorized by the respective years.

5. **Movie genre of actor/actresses:**

- User selects a specific actor/actress from the list.
- System retrieves the genres of movies associated with the selected actor/actress.
- System displays the list of movie genres.

6. **Average rating of their movies (overall and each year):**

- User selects a specific actor/actress from the list.
- System retrieves the movies associated with the selected actor/actress.
- System calculates and displays the average ratings of the movies overall and for each year.

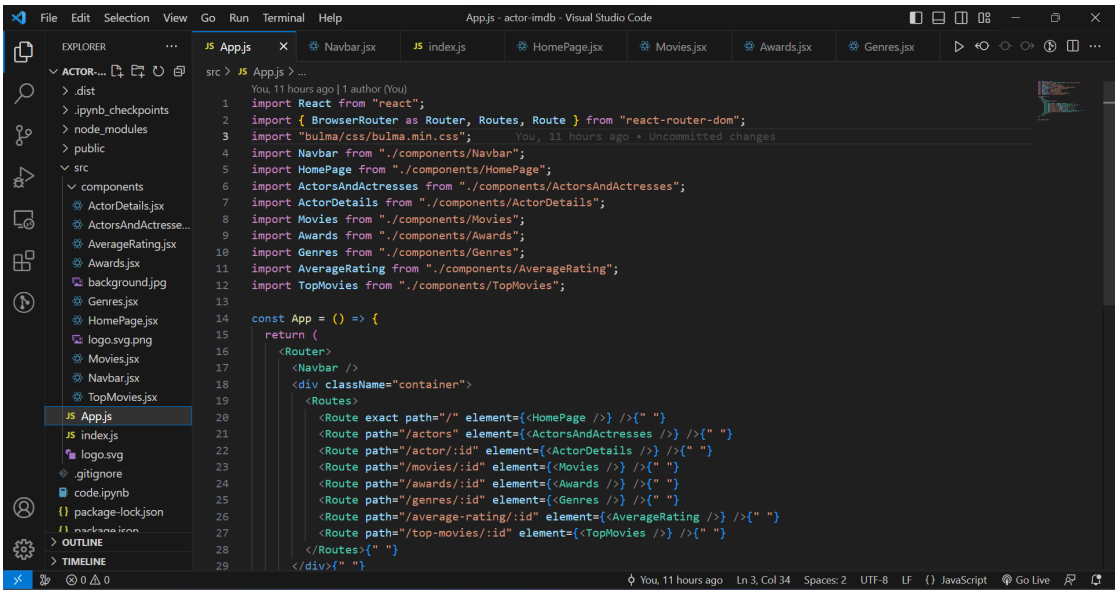
7. **Top 5 movies, their respective years and genre:**

- User selects a specific actor/actress from the list.
- System retrieves the movies associated with the selected actor/actress.
- System identifies the top 5 highest-rated movies based on ratings.
- System displays the top 5 movies along with their respective years and genres.

3 Frontend Setup

In Brief: In this step of project, I am setting up the Reactjs frontend following the sequence description above.

3.1 Sample:



3.2 Component structure:

- **App.js:** Main component that renders the IMDb software.
- **ActorList.js:** Component to display the list of actors and actresses.
- **ActorDetails.js:** Component to display the details of a specific actor/actress.
- **MovieList.js**: Component to display the list of movies for an actor/actress.
- **AwardsList.js:** Component to display the awards received by an actor/actress.
- **MovieGenres.js:** Component to display the movie genres associated with an actor/actress.
- **AverageRating.js:** Component to display the average ratings of an actor/actress.
- **TopMovies.js:** Component to display the top 5 movies of an actor/actress.

4 Data Scraping

4.1 Actors/Actresses scraping

4.1.1 Sample code:

```
for actor in list_of_actors:
            name = actor.find('h3', class_= "lister-item-header").find('a').text
    [1:-1]
            biography = actor.find('div', class_="lister-item-image").find('img').
    get('src')
            birth_info = actor.find('h3', class_= "lister-item-header").find('a').
    get('href')[6:]
            link = f"https://www.imdb.com/name/{birth_info}/bio/?ref_=nm_ov_bio_sm
    ".format(birth_info=birth_info)
            birth_page = requests.get(link, 'html.parser', headers={'User-Agent':
    'Mozilla/5.0 (Windows NT 10.0)'})
            birth_page = BeautifulSoup(birth_page.text, 'html.parser')
            birth_info = birth_page.find('section', class_="ipc-page-section ipc-
    page-section--base").find('ul').find_all('li')[0].find_all('a')
            birth_date = birth_info[0].text + " " + birth_info[1].text
            birth_place = birth_info[2].text
```

4.1.2 Brief exlaination:

1. **Name:** This line finds the HTML element that contains the actor's name, extracts the text, and assigns it to the `name` variable. The `[1:-1]` slicing is used to remove any leading or trailing whitespace.
2. **Biography Image URL:** This line finds the HTML element that contains the actor's biography image, extracts the source URL of the image, and assigns it to the `biography` variable.
3. **Birth Info URL:** This line finds the HTML element that contains the actor's birth information, extracts the `href` attribute value, and assigns it to the `birth_info` variable. The `[6:]` slicing is used to remove the initial characters from the `href` value.
4. **Biography Page URL:** This line constructs the URL for the actor's biography page on IMDb by combining the `birth_info` value with the base URL. The resulting URL is assigned to the `link` variable.
5. **Retrieve Biography Page:** This line sends an HTTP GET request to the `link` URL, retrieves the HTML content of the page, and assigns it to the `birth_page` variable.
6. **Create BeautifulSoup Object:** This line creates a BeautifulSoup object from the HTML content of `birth_page`.
7. **Extract Birth Information:** This line finds the specific section and list element in the HTML structure of `birth_page` and extracts a list of anchor elements (`<a>`) containing birth information.
8. **Extract Birth Date:**This line extracts the birth date information from the `birth_info` list and concatenates the text values of the first and second elements. The resulting birth date is assigned to the `birth_date` variable.
9. **Extract Birth Place:** This line extracts the birth place information from the `birth_info` list and assigns it to the `birth_place` variable.

4.2 Movie Information Scraping

4.2.1 Sample code:

```
for actor in actors_id_url:
    actor_page = requests.get(actor['url'], 'html.parser', headers={'User-Agent':
    'Mozilla/5.0 (Windows NT 10.0)'})
    soup = BeautifulSoup(actor_page.text, 'html.parser').find_all('section',
    class_="ipc-page-section ipc-page-section--base")[6].find('ul').find_all('li')
    for film in soup:
        film = film.find('a')
        movie_name = film.text
        release_year = film.parent.text[(film.parent.text.find('(') + 1):film.
    parent.text.find(')')]
        rating_page = requests.get('https://www.imdb.com/' + film['href'], 'html.
    parser', headers={'User-Agent': 'Mozilla/5.0 (Windows NT 10.0)'})
```

4.2.2 Brief explanation:

1. **Retrieve Actor's Page:** This line sends an HTTP GET request to the URL of the actor's page, retrieves the HTML content of the page, and assigns it to the `actor_page` variable.

2. **Create BeautifulSoup Object:** This line creates a BeautifulSoup object from the HTML content of `actor_page` and finds all the sections with the class `ipc-page-section-base`. The index [6] is used to access the specific section that contains the film information.

3. **Extract Film Information:** This line finds all the list elements (`<li>`) within the previously obtained section. It then iterates over each film and extracts its name and release year by accessing the anchor element (`<a>`) and using string manipulation to extract the desired information.

4. **Retrieve Rating Page:** This line sends an HTTP GET request to the URL of the specific movie's rating page on IMDb. It retrieves the HTML content of the page and assigns it to the `rating_page` variable. For each film in the extracted film information, the code performs the following steps:

5. **Find Film Anchor Element:** finds the anchor element (‘<a>‘) within the current film element.

6. **Extract Movie Name:** extracts the text content of the film anchor element, representing the movie's name, and assigns it to the ‘movie_name‘ variable.

7. **Extract Release Year:** retrieves the parent element of the film anchor element, which contains the movie's release information. It uses string manipulation to extract the release year from the parent element's text and assigns it to the ‘release_year‘ variable.

8. **Retrieve Rating Page:** It retrieves the HTML content of the page and assigns it to the ‘rating_page‘ variable.

5 Backend Setup

5.1 Model setup

In Brief: In this step of project, I am design detail model database for data scraping from IMDB. During the time considering, my plan has a little change in the backend technical decision from Nodejs that I decided to used in Task 1, to Django Python. Because I think it is more convenient for me to using it in building model PostgeSQL and also scrapping data from IMDB.

♣ Sample :

```python
from django.db import models
class Actors(models.Model):
    actor_id = models.AutoField(primary_key=True)
    actor_name = models.CharField(max_length=255)
    biography = models.TextField()
    birthdate = models.CharField(max_length=255)
    birthplace = models.CharField(max_length=255)
    bio_url = models.TextField()
    def __str__(self):
        return self.actor_name

class Movie(models.Model):
    movie_id = models.AutoField(primary_key=True)
    movie_name = models.CharField(max_length=255)
    release_year = models.CharField(max_length=255)
    average_rating = models.CharField(max_length=255, null=True)
    def __str__(self):
        return self.movie_name

class ActorMovie(models.Model):
    actor_id = models.ForeignKey(Actors, on_delete=models.CASCADE)
    movie_id = models.ForeignKey(Movie, on_delete=models.CASCADE)
    def __str__(self):
        return self.actor_id

class Award(models.Model):
    award_id = models.AutoField(primary_key=True)
    award_name = models.CharField(max_length=255)
    year = models.CharField(max_length=255)
    movie_id = models.ForeignKey(Movie, on_delete=models.CASCADE)
    actor_id = models.ForeignKey(Actors, on_delete=models.CASCADE)
    def __str__(self):
        return self.award_name

class Genre(models.Model):
    genre_id = models.AutoField(primary_key=True)
    genre_name = models.CharField(max_length=255)
    def __str__(self):
        return self.genre_name

class MovieGenre(models.Model):
    movie_id = models.ForeignKey(Movie, on_delete=models.CASCADE)
    genre_id = models.ForeignKey(Genre, on_delete=models.CASCADE)

class Rating(models.Model):
    movie_id = models.ForeignKey(Movie, on_delete=models.CASCADE)
    year = models.CharField(max_length=255)
```

5.2 API and Routes setup

In Brief: In this pseudo code, I define two API endpoints, /actors/ and /movies/, each with two methods, PUT and GET. When a PUT request is made to these endpoints, the backend will scrape data from IMDB and store it in the database. When a GET request is made, the backend will retrieve data from the database and return it as a response.

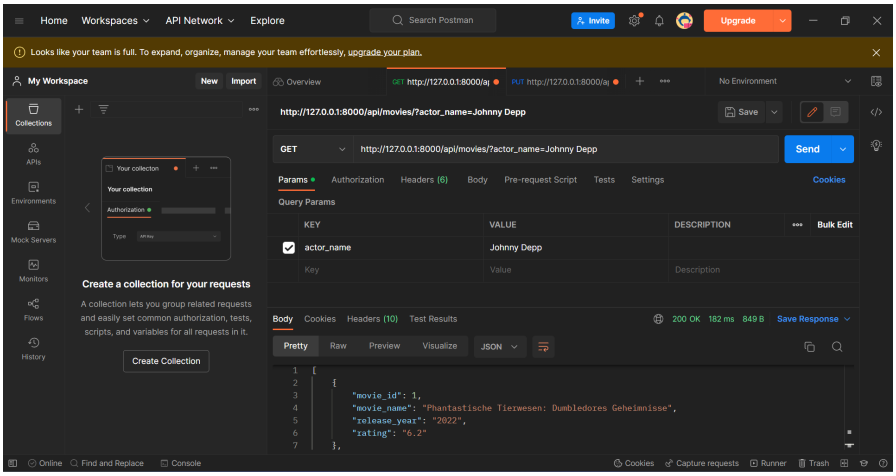
♣ Psuedo code :

```
# Import necessary modules
from django.urls import path
from . import actors, movies

# Define the API endpoints
urlpatterns = [
    path('actors/', actors.actors, name='actors'),
    path('movies/', movies.movies, name='movies'),
]

# Actors API endpoint
def actors(request):
    if request.method == 'PUT':
        # Scrap data from IMDB
        # Put scraped data into the database
        # Return appropriate response

    elif request.method == 'GET':
        # Get data from the database
        # Return the fetched data as a response

# Movies API endpoint
def movies(request):
    if request.method == 'PUT':
        # Scrap data from IMDB
        # Put scraped data into the database
        # Return appropriate response

    elif request.method == 'GET':
        # Get data from the database
        # Return the fetched data as a response
```

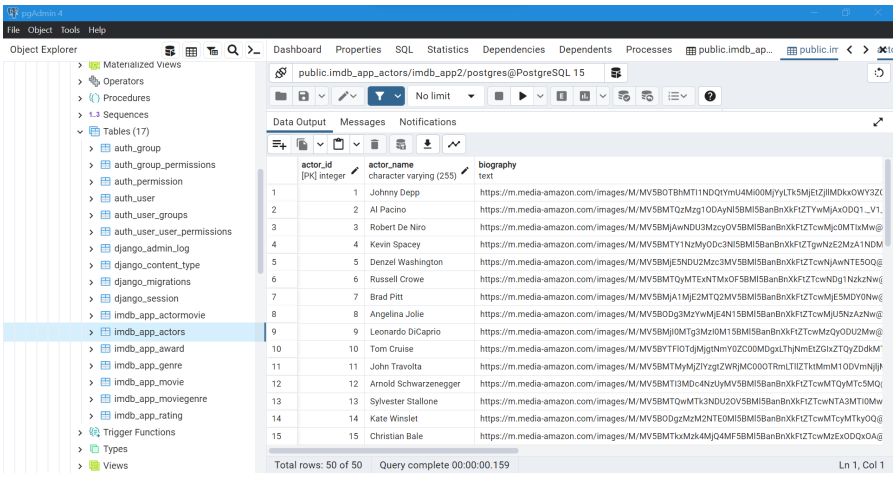
5.3 Database and API Testing

In Brief: The database design incorporates tables and relationships that allow for effective data retrieval and manipulation. It supports crucial operations such as querying, inserting, updating, and deleting data, providing a solid foundation for the IMDB app to handle and deliver accurate information to its users.

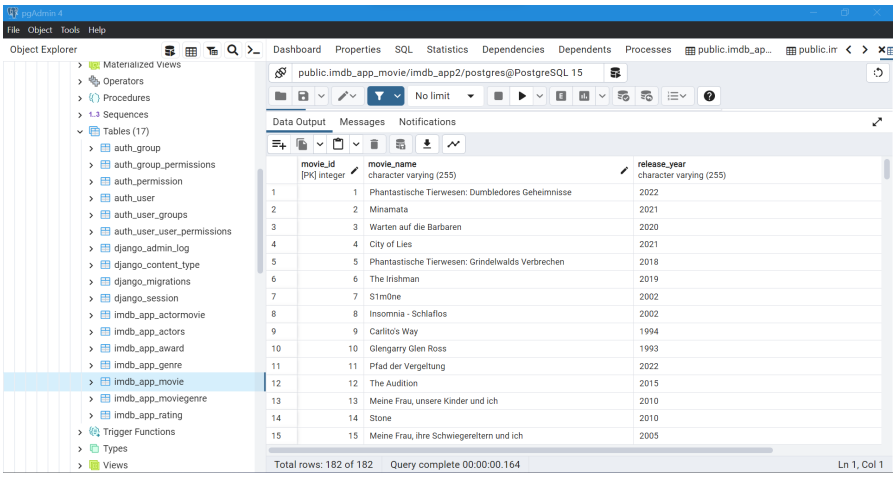
5.3.1 Postman testing environment



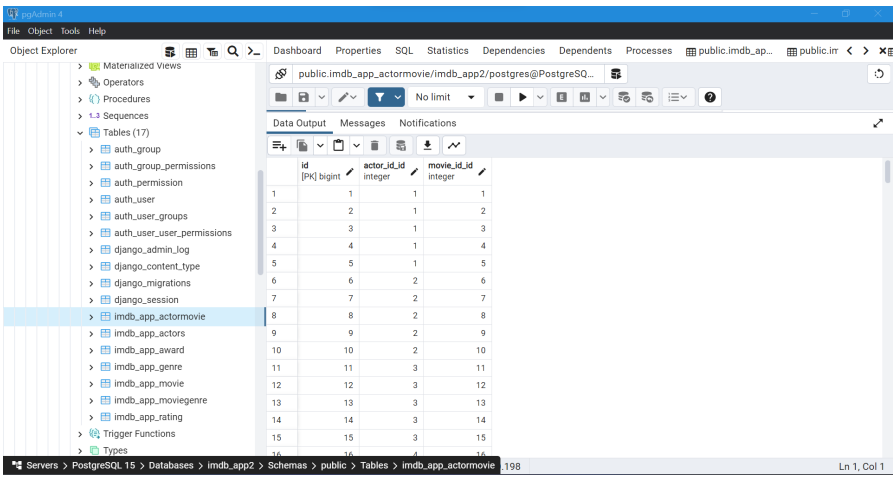
5.3.2 Actor/Actresses tables after Scraping



5.3.3 Movie tables after Scraping



5.3.4 Actors/Actresses-Movies tables after Scraping

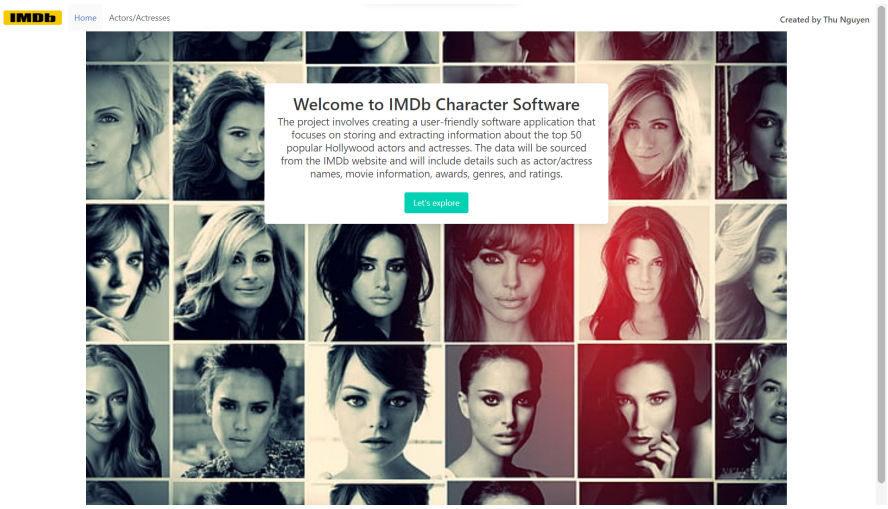


6 Final Result

In Brief: In my IMDb character app, I have implemented basic functionalities that meet some of the requirements of the project. However, due to time constraints, some functionalities have not been fully completed and may not be fully reflected in the interface. Nevertheless, here are some of the things I have accomplished so far.

6.1 Frontend

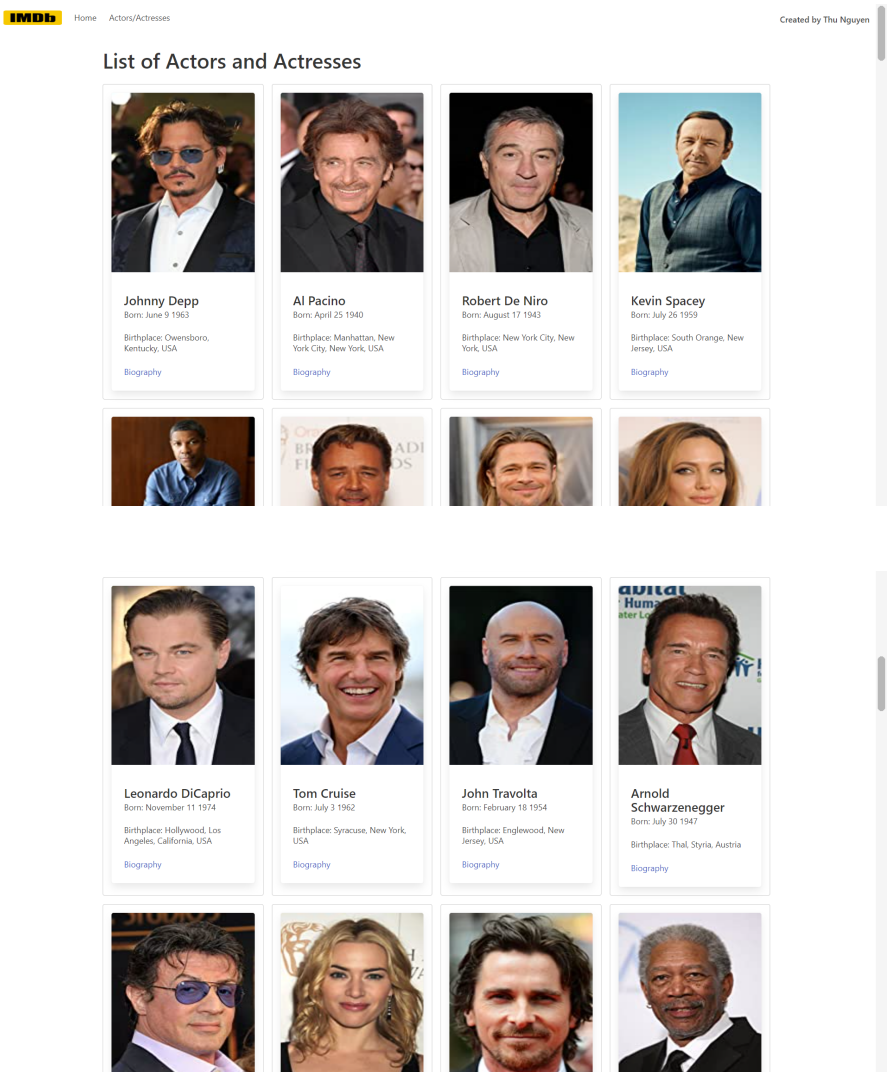
6.1.1 Home page Interface:



Home Page:

- Upon landing on the home page, users are presented with Navbar with homepage and actors/actresses components.
- In this context, users can explore actors/actresses by clicking on the "Explore"button or go there by the component name on the navbar.

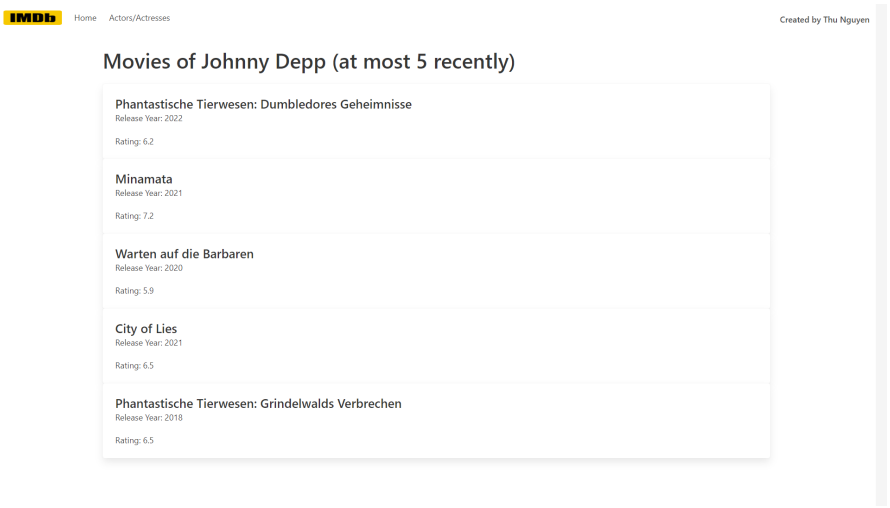
6.1.2 List and information of all the actors/actresses:



Actors/Actresses Component:

- When the "Explore"button is clicked, users are directed to the "Actors/Actresses"component.
- This component displays a list of 50 actors/actresses, showcasing their names and possibly their profile images and also their personal information.
- Users can scroll through the list to browse different actors/actresses.
- When users click on the 'biography', they are immediately redirected to the IMDb website, specifically to the corresponding biography page for that actor/actress.

6.1.3 List and information of moives of a specific actor/actress:



Movies Component:

- The "Movies"component is responsible for providing users with detailed information about a specific actor/actress and their recent movies.
- Upon arrival, users are presented with a list of the actor's/actress's recent movies.
- Each movie entry typically includes the movie title, release year, and possibly additional information such as the movie's genre or rating.
- Users can scroll through the list to explore the actor's/actress's at most 5 recent filmography.
- The year of movie and the average rating is also being shown to users.