IIT Bombay

CS108 Project

Movie Mania

By:
Gulshan Kumar

TA:
Sabyasachi Samantaray

Instructor: Prof. Kameswari Chebrolu

Contents

1 Introduction

In today's digital age, the consumption of movies has become an integral part of our entertainment landscape. "Movie Mania," a comprehensive web application designed to serve as a one-stop repository for movie aficionados.

1.1 Details Shown

Movie Mania offers users access to movie titles, release years, durations, ratings from reputable sources (IMDB), category ratings(R, PG-13, etc), genres, directors, main cast members, plot summaries, trailer, user reviews(both positive and negative reviews), famous critics reviews and image related to each movie. Average Rating of movies by users on our site is also shown for some movies(which has been rated by atleast one user). I have collected these data for IMDB top 250 movies.
Additionally, the platform provides users with personalized recommendations based on their viewing preferences, further enhancing the movie discovery experience.

2 Technology Used

In this section, we provide a brief overview of the methods, techniques, and technologies employed in the development of Movie Mania, including data collection, database design, web interface design, recommender system implementation, customization features implementation and testing and validation.

2.1 Data Collection

All the datas has been extracted from IMDB site except Trailer URL, which is extracted from YouTube.

I have used Python Package selenium, requests and Beautiful Soup to extract the datas from IMDB site. Extracting from YouTube required me to use YouTube API available on Google Developer's Page.

2.2 Website Interface

For making the website interface, I have used HTML, CSS, Javascript as basic requirements and NodeJS to handle the server, Bootstrap to style the page and make it completely responsive on different screen widths, Express JS for dynamically adding movie-contents, ejs view template as a view engine.

2.3 Data Formatting

I have used Python Packages pandas and json.
While collecting data, which I have shown in the shared jupyter notebook, I have stored every data in a separate list. To combine all the datas in a single

dataframe, I have used pandas. Also pandas helped me to directly read the json file in case if some new coumns need to be added. This was very useful as I collected data in several days and several runs.
JSON package gives me functionality to read the json formatted data, convert data into json format. Basically, it gives some extended power for json parsing and writing.

3 Data Collection

Used selenium, requests module along with beautiful soup and YouTube API(for trailer only).

3.1 Using Selenium

Selenium is particularly helpful, if the website to be scraped is heavily loaded with JavaScript. In IMDB home page [6] there was not much of the information shown - only Information was Title, Ratings and Poster image.
A button is present which if clicked show many more details. But this button does this by JavaScript.

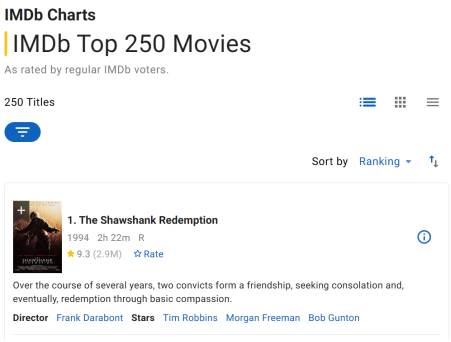


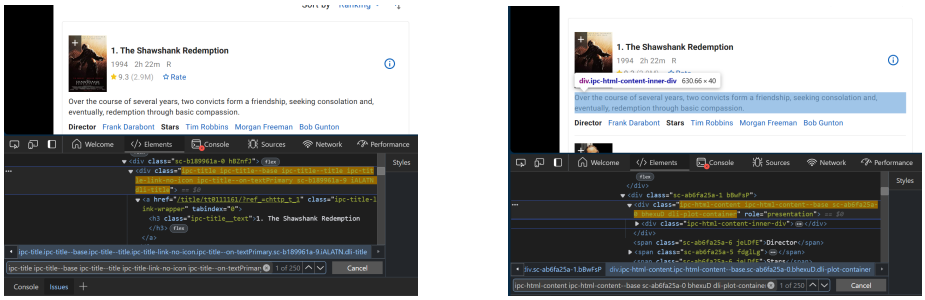
Figure 1: Button highlighted in blue triple menu icon

As shown in 1, it contains many more information. So I wanted this button to be clicked once on start of the page. For these kind of tasks selenium is the best choice. I refered to the basic code syntax from docs[9] and a YouTube video[15].

Then, we have to to the site [6] and go to developer tools and analyse the content, look for the various html tags, their classes and xpaths[11]. The main problem here is we have to choose a proper identifiers(id, class or xpath) corresponding to the content so that only what is required is referred to.
For Example if you will just look for **div**, there will be thousands of **div** tags in that code. Choosing the required tags and classes and then combining using xpath took very much time.

```
1 div=driver.find_elements(by='xpath',value='//div[@class="ipc-title
      ipc-title--base ipc-title--title ipc-title-link-no-icon ipc-
      title--on-textPrimary sc-b0691f29-9 klOwFB dli-title"]')
2 titles=[movie.text for movie in div]
```

Listing 1: Extracting title code



(a) Title Extraction (b) Plot Extraction

Figure 2: Some Examples to find the proper tags and classes, NOTE that exactly 250 movies must match.

From here I have extracted:

- Title of the movie
- Director
- Stars
- Metadata (Year, Duration and R/PG-13 etc)
- Rating
- Short Plot
- Poster Image of Various Sizes

Note that Genre is not available on this page1, but Genre is available by clicking on **i** button in fig1. This comes on clicking the button:3. This was also JavaScript based, so selenium can be used effectively.

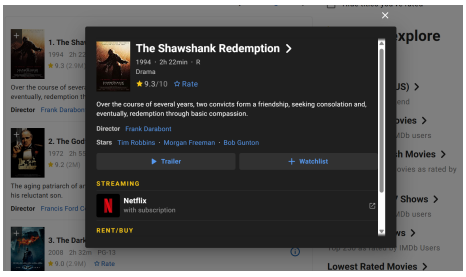


Figure 3: Genre Extraction Page

Info button address is found by:

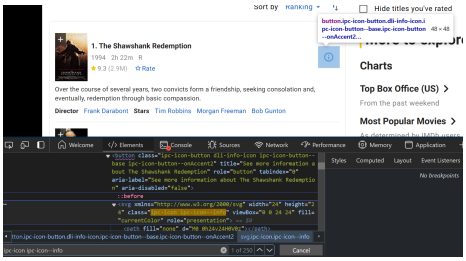


Figure 4: ID of info button

Reference for automatic button clicking in selenium: [3]

Problem Faced in clicking btn Fig:3

Selenium has been specifically written to NOT allow interaction with hidden elements. The rational is that if a person cannot perform that action, then neither should Selenium.
That means by normal method, I can click only on 4 or 5 movies(max to max 15-16 movies by zooming out).

Solution: Found a way on stackoverflow by executing script using selenium and clicking button with script. JavaScript can click hidden buttons also. [16].

Genre also extracted...
Basic extraction done.

3.2 Using Beautiful Soup And Requests

There is a package called bs4 and inside it there is a module BeautifulSoup. Request is an another package.
This was basically used for advanced scraping in my project(user and critics reviews).

What is the advantage of using bs4 and request over selenium in static page?

The simple answer is BeautifulSoup is much faster than selenium when the page is static. Selenium needs Chrome to open and then search the link then find and so on.. Also selenium loads all the styles, scripts, etc. This makes useless consumption of time, if you nee only a part of the website[17].
One thing I found was 'Metacritic' Reviews are directly linked to IMDB movie page5.

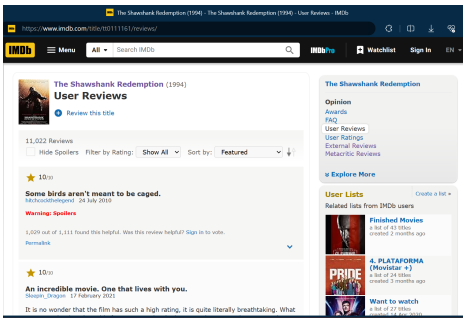


Figure 5: Metacritic reviews on IMDB site

The syntax of the url pattern is: **https://www.imdb.com/title/IMDBuniqueID/reviews/** eg: `https://www.imdb.com/title/tt0111161/reviews/` for *The Shashwank Redemption*.

So I need to extract the id of all the movies first. I used the references provided in the Problem Statement Docs[8] particularly section on Fake User Agent[5].

Get the source code and noticed that imdb-id starts with tt and some number.

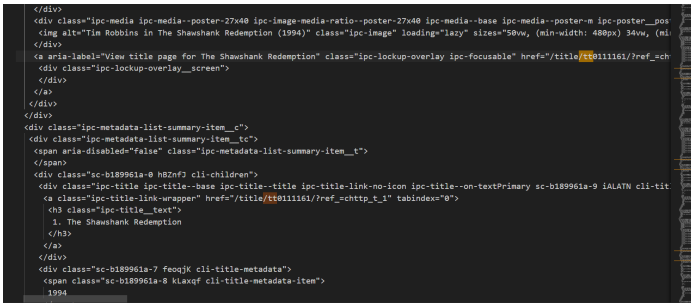


Figure 6: IMDB id: starts with tt

To extract these, I used regex package of python: **re**. Also id was like: **/title/tt0468569/?ref_=chttp_t_3**, so I extracted texts beween */title* and *?ref* using regex pattern.

Having got the imdb-id of all the movies, now it easy to put in the review link page url.5

To find the positive review, I applied the filter of 10, and hide the spoiler as shown in Fig:7 and the url becomes `https://www.imdb.com/title/tt0111161/reviews?spoiler=hide&sort=curated&dir=desc&ratingFilter=10`. So I just replaced the imdb-id with the extracted ones and got the positive and negative reviews. Did this by inspecting the source code as explained in jupyter notebook **extract_user_review.py**.

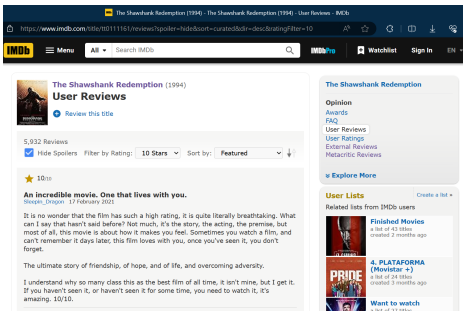


Figure 7: Positive Reviews Filter

Extracting critics reviews was also similar: The url becomes `https://www.imdb.com/title/tt0111161/criticreviews/` 8

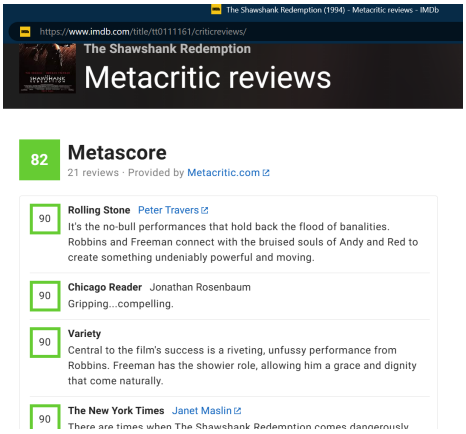


Figure 8: Critic Review Page

Fortunately all the critic reviews was inside json inside script as shown in 9. So I just extracted it easily.

Figure 9: Critic reviews inside script tag

All the required scraping is done now. Just for my personal interest, I also scraped YouTube trailer url using YouTube API.[14]

4 Website Interface

I have used nodejs for backend and routes handling with express js as web framework and ejs as engine view template, Bootstrap for styles and making the page responsive. I was not determined to use any nodejs in my project.

4.1 Why used NodeJS? Problem with client-side JavaScript

First of all, I made everything in client-side javascript(Traditional JavaScript) only, and it was working except for the SignUp page. It is very difficult to write on local files using client-side JS due to some security reason.[13]
But It can be easily done using server-side JavaScript. Due to this, I have to transfer all my code to nodejs project. I learnt nodejs using YouTube.[12] Also used Docs of expressJS [4], mostly in Guide section: rendering, middlewares use, URLencoder, etc.

4.2 Storing Email and Password

I have used inbuilt package *session* to store the email and password. Code Snippet: [10].

ExpressJS and ejs

ExpressJS is widely used as a web framework in nodejs, so I choose it. ejs is also widely used template view engine, but there are many other alternatives to it like pug. There is one advantage to using ejs, which is the syntax to render is very similar to basic HTML, and so easy to learn.

In Express.js, a popular web application framework for Node.js, we can utilize EJS (Embedded JavaScript) as a view engine to generate dynamic HTML content. With EJS, we can define HTML templates that contain JavaScript code to dynamically generate content based on data.

When we define a template using EJS in Express.js, we essentially create a reusable blueprint for generating HTML content. This template can contain placeholders or variables that will be dynamically replaced with actual data when the template is rendered.

By defining a template in one central location within our Express.js application, we can then reuse it across multiple routes or views within our application. This promotes code reusability, maintainability, and consistency throughout our project.

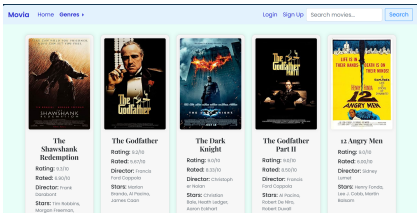
Example: I have defined card template at one place, and reused at many places. Similarly for navbar.

4.3 Bootstrap

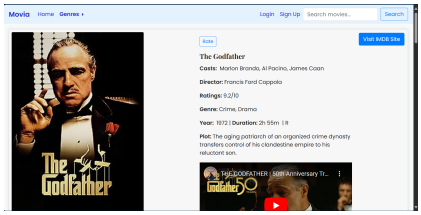
I used Bootstrap to make the styling of the website and to make it responsive. It has pre-written template of many of the components like navbar, card(which I have use in showing movie-card), pagination, etc.[2] [1]

Our Website is Completely Responsive!

Our website can change the style depending on the screen-size. These Responsiveness is done using BootStrap inbuilt classes like .collapse .md-3, .sm-6, etc.



(a) Home Page



(b) detail-Page

Figure 10: Website on laptop

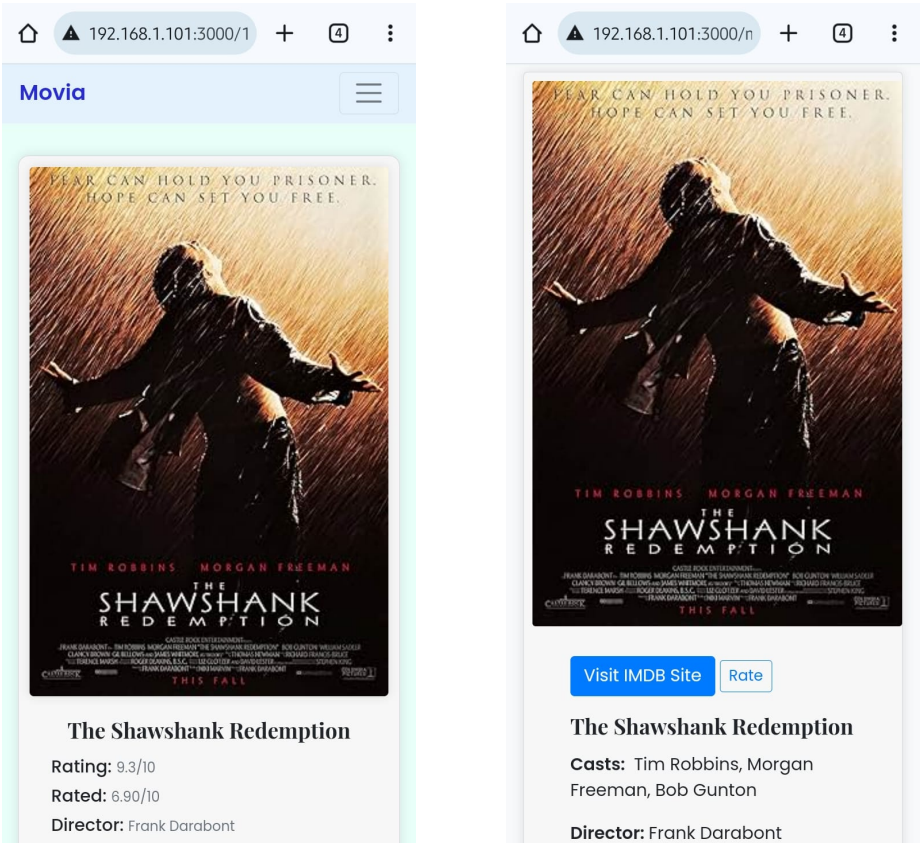
4.4 Login And SignUp

You can Login and Signup for personal recommendation and watching your rated movies and their ratings. Email and password are stored in local files, so even if the server is closed and restarted again, you can login with same email and password.
Some form-validation has been done -
In SignUp page:

- E-mail Format Checked Using Regex
- You cannot use Email already taken
- Password must be atleast 8 characters long
- password and confirm password must match

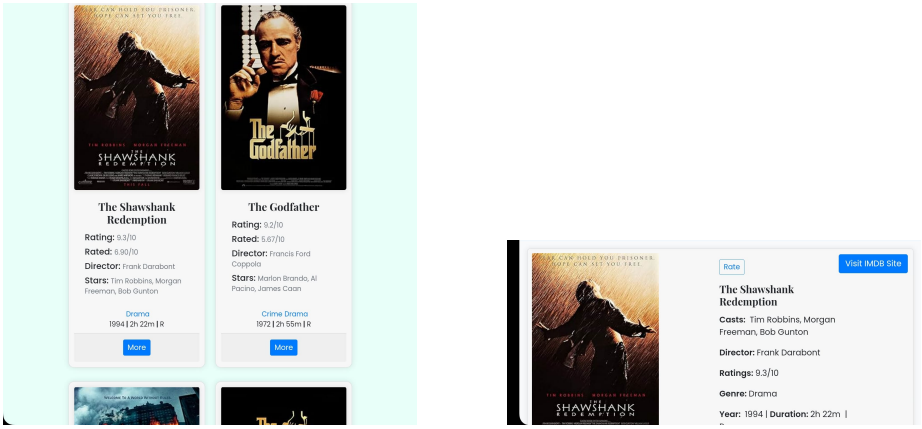
In Login Page:
Proper message is displayed:

- If email exists in users.json but password incorrect
- If email does not exists users.json



(a) Home Page (b) detail-Page

Figure 11: Website on Mobile



(a) Home Page (b) detail-Page

Figure 12: Website on Tablet

Initial Problem while Refreshing Page

Initially when the pages refreshes, the user get logged out. Solved this using **sessions** in nodejs[7].

4.5 Rated and Recommendation Button:

These buttons will be visible in navbars only if you are logged in.
Rated will show the movies Rated by you and what you have rated.
Recommendation will show the Recommended movies to watch based on the movies rated by you.
If you are a new User then both rated and recommendation will be empty displaying message: 'First Rate some movies.'

4.6 Describing Cards

Refer Figure:12(a).

- Rating: IMDB rating
- Rated: Average Rating of all our user, this is shown only if movie is rated by atleast one user.
- Director: Director Names
- Stars: Notable actor and actress in the movie
- Genre: Shown in blue, when clicked will show other movies of same genre
- Metadata: Year of Release, Duration, and type of audience assumed

- More: When More button is clicked, a detailed page will appear showing more details

4.7 Describing Detail Page

On the detail page, as shown in Fig:10, there are details as available on home page along with:

- Rate Button: A button to rate the movies, It can be used only when you are logged in.
- Visit IMDB site: A redirection to imdb page of the movie.
- Trailer: YouTube Trailer
- User Reviews: Both positive and negative user reviews, their ratings and usernames. Approximately each movie has 10-20 total reviews of each category
- Critic Reviews: Famous critic reviews, and some of their personal url link (when clicked on their names).

5 Recommender System

Here is the algorithm used for Recommending the movies based on the Raings given by the user.

Algorithm 1: Recommendation Algorithm

Data: User's rated movies, Database
Result: Recommended movies
if *user has not rated any movies* **then**
    Display message to rate some movies;
else
    Initialize genreCount object to keep track of number of movies user has rated in each genre;
    foreach *movie user has rated* **do**
        Increment count for each genre of movie in genreCount;
    end
    Initialize recommendedMovies array to store recommended movies and their similarity scores;
    foreach *movie in database* **do**
        Initialize movieSimilarity to store similarity score of movie;
        foreach *movie user has rated* **do**
            Calculate genreSimilarityScore using Jaccard similarity;
            Calculate castSimilarityScore using Jaccard similarity;
            Calculate directorSimilarityScore using Jaccard similarity;
            Add weighted sum of similarity scores to movieSimilarity;
        end
        Add movie's rating to movieSimilarity;
        Add movie and movieSimilarity to recommendedMovies;
    end
    Sort recommendedMovies in descending order of similarity score;
    if similarity is same; First calculate the most watched genre of the movie, then sort on the basis of less value of the most watched genre;
    Filter out movies user has already rated from recommendedMovies;
    Display top 10 recommendedMovies on page;
end

```
1    // Recommendation sorting
2    recommendedMovies.sort((a, b) => {
3    roundedA = Math.round(a.similarity * 10) / 10 // round to 1
         decimal places
4    roundedB = Math.round(b.similarity * 10) / 10
5    if (roundedA === roundedB) {
6      // console.log('here inside if rounded')
7      // sum of the number of movies rated by the user in the
           genres of the movie
8        sumA = -1
9        sumB = -1
10       for (let i = 0; i < a.movie.genre.length; i++) {
11           if (sumA < genreCount[a.movie.genre[i].toLowerCase()]){
12             // console.log('here inside if sumA')
13               sumA = genreCount[a.movie.genre[i].toLowerCase()]
14             }
15           }
```

```
            console.log(sumA)
        for (let i = 0; i < b.movie.genre.length; i++) {
            if (sumB < genreCount[b.movie.genre[i].toLowerCase()]){
                sumB = genreCount[b.movie.genre[i].toLowerCase()]
            }
        }
        // console.log(sumA, sumB)
        if (sumA === sumB) {
            return parseFloat(b.movie.Rating[0]) - parseFloat(a.
                movie.Rating[0])
            //console.log('here')
        }
        return sumA - sumB // sort in ascending order of sum of the
             number of movies rated by the user in the genres of
            the movie
    }
    return b.similarity - a.similarity
})
```

Listing 2: Recommendation Sorting

```
// Recommendation Filtering
recommendedMovies = recommendedMovies.filter((movie) => {
    return !myratings[user.email].some((ratedMovie) => ratedMovie.
        movie.Title === movie.movie.Title) // filter out movies the
         user has already rated by the user
})
```

Listing 3: Rated Movie Filtering

Jacard Similariry:

```
// jacard similarity
function jacardSimilarity(set1, set2) {
    let intersection = 0
    let union = 0
    for (let i = 0; i < set1.length; i++) {
        if (set2.includes(set1[i])) {
            intersection++
        }
    }
    union = set1.length + set2.length - intersection
    return intersection / union
}
```

Listing 4: Jacard Similarity

Algorithm 2: Jaccard Similarity

Data: Two sets, set1 and set2
Result: Jaccard similarity between the sets
Function `jacardSimilarity`(*set1*, *set2*)**:**
  intersection $\leftarrow 0$;
  union $\leftarrow 0$;
  for $i \leftarrow 0$ **to** *length of set1* **do**
    if *set2.includes(set1[i])* **then**
      intersection $\leftarrow$ intersection $+ 1$;
    end
  end
  union $\leftarrow$ length of set1 + length of set2 − intersection;
  return $\frac{intersection}{union}$;

6 More Customization Features

6.1 AutoCorrect Search

Algorithm 3: Bigramize Function

Data: Title
Result: Bigrams
`bigramize`(*Title*) Initialize *bigrams* array;
for *each character in title except the last one* **do**
  Add the bigram starting at the character to *bigrams*;
end
return *bigrams*;

Algorithm 4: Correct Query Function

```
Data: SearchQuery
Result: CorrectedQuery
correctQuery(searchQuery) Split searchQuery into words;
Initialize mostSimilarWords array with null words and zero similarity;
for each movie in database do
    Split movie title into words;
    for each word in searchQuery and movie title do
        if title word is too short, skip it then
            Calculate bigrams of search word and title word;
            Calculate word similarity using jaccard_similarity;
            if word similarity is greater than current maximum then
                Update mostSimilarWords;
            end
        end
    end
end
Join most similar words into correctedQuery;
return correctedQuery;
```

Example of Bigramize function

Calling the `bigramize` function with the title "Hello World":

$$\texttt{bigramize}(\textit{"Hello World"})$$

This would produce bigrams as follows:

$$[\text{"}He\text{"},\text{"}el\text{"},\text{"}ll\text{"},\text{"}lo\text{"},\text{"}o\text{"},\text{"}W\text{"},\text{"}Wo\text{"},\text{"}or\text{"},\text{"}rl\text{"},\text{"}ld\text{"}]$$

Why It Suffices?

This algorithm suffices for the following reasons:

1. It corrects the search query by replacing each word with the most similar word from the movie titles. This helps to find relevant results even if the user makes a typo.

2. It uses the Jaccard similarity of bigrams, which is a robust measure of string similarity. It can handle differences in word order, and it gives a high similarity score to words that share many bigrams, even if they are not exactly the same.

3. It returns the movie with the highest average word similarity to the corrected query. This ensures that the most relevant movie is returned, even if it does not exactly match the search query.

Problem Faced

Earlier, instead of correcting word by word, I compared the whole search query with all the 250 complete titles. I had also set threshold of 60% similarity to show the movies. But It caused a problem: Suppose the movie title is *Spider-Man: Into The Spider Verse*, and you just typed *sdiper*, then the percentage match will be much less and this movie will not be shown.
To solve this problem, I thought to compare word by word for each word in search query and each word in Title.

Limitation

- Only 1 movies will be shown, even if *spider man* is contained in three movies *Spider-Man: Into the Spider-Verse* and *Spider-Man: Across the Spider-Verse*, only one movie is shown if spider is written as query. It is because our search algorithm only returns one movie. But writing spider-man (without space) will show all three movies
- If some random stuff if typed, it will still show some movies, undesired.

Idea

In the context of autocorrect, this technique is used to find the word in the movie titles that is most similar to each word in the search query. This is done by comparing the bigrams of the search word to the bigrams of each title word and finding the title word with the highest Jaccard similarity.

This approach is effective because it can handle small typos and differences in word order. For example, if the user types "hte" instead of "the", the bigrams of the typo are very similar to the bigrams of the correct word, so the correct word will have a high similarity score. Similarly, if the user types "dark knight" instead of "knight dark", the bigrams of the two phrases are the same, so they will have a perfect similarity score.

7 Folder Structure

The folder contains following files:

- `scraping.ipynb`: Jupyter notebook showing the scraping code for all the basic parts and imdb id.
- `extract_reviews.ipynb`: Jupyter notebook showing extraction of critics reviews.
- `extract_user_review.ipynb`: Jupyter notebook showing the scraping of user reviews- positive and negative
- `extractingTrailersURL.ipynb`: Jupyter notebook showing extraction of YouTube Trailer URL.

- `titles.txt`: Containing titles of all the movies
- `genres.txt`: containing genres of all movies
- `project_ejs_express` **Folder**: Containing All json datas, node modules, webpages and other templates required for the working of website.

  Inside this folder

  - `data` **Folder**: containing datas extracted
    - `criticReviews.json`: containing critics reviews
    - `movies.json`: containing basic movie information
    - `myratings.json`: containing our site users rating and corresponding movies
    - `userReviews.json`: containing positive and negative reviews of users across metacritic.
    - `users.json`: containing user emails and passwords
  - `node_modules` **Folder**: Auto generated by npm (node package manager) for dependencies on our site
  - `public` **Folder**: contains styles and client-side javascripts needed.
    - `movieCards.css`: styles for cards
    - `movieDetails.css`: styles for detailed page
    - `movieDetails.js`: javascript file for detail page, like clicking on button *read more*, etc
    - `signup.js`: Form validation for signup page and checking if email already registered.
  - `sessions` **Folder**: contains session for user, to do login and logout.
  - `templates` **Folder**: contains reusable templatescomponents
    - `avgRating.ejs`: This will just add one line to show the average rating of our site user.
    - `cardForRatedRecom.ejs`: Card to be used in *Rated* tab and *Recommendations* tab.
    - `eachMovieCard.ejs`: card to be used elsewhere.
    - `navbar.ejs`: Adding navigation bar
  - `views` **Folder**: Containing files what to show when user clicks on some buttons/links.
    - `genre.ejs`: Genre dropdown option
    - `index.ejs`: home page
    - `login.ejs`: login page
    - `movieDetails.ejs`: movie detail page
    - `pagination.ejs`: to add pagination to bottom on home page
    - `recommend.ejs`: recommendation tab

    * `search.ejs`: search bar
    * `signup.ejs`: signup page
    * `userRated.ejs`: rate tab
  - `1.js`: server to run

  Other files were automatically generated by git and npm

8 JSON Data structure

1. `criticReviews.json`
   - **Score Column:**
     - **Description:** Rating out of 100 given by the critic for the movie.
   - **Reviewer Column:**
     - **Description:** Name of the critic who provided the review.
   - **Site Column:**
     - **Description:** Publication or website where the review was published.
   - **URL Column:**
     - **Description:** URL to the review (if available). If null, it means the review is not available online or the URL was not provided.
   - **Quote Column:**
     - **Description:** Excerpt from the review provided by the critic.
2. `movies.json`
   - **Title Column:** The title of the movie.
   - **Rating Column:** The IMDb rating of the movie.
   - **Metadata Column:** Additional metadata about the movie, including the release year, duration, and rating (e.g., R-rated).
   - **Info Column:** A brief summary or description of the movie's plot.
   - **Director Column:** The director(s) of the movie.
   - **Cast Column:** The main cast members of the movie.
   - **Image Column:** URLs to different sizes of movie poster images.
   - **Trailer Column:** URL to the movie's trailer.
   - **imdbID Column:** The IMDb identifier for the movie.
   - **Genre Column:** The genre(s) of the movie.
3. `myRatings.json`:
   - **email**: user email

- **movie**: the movie and its detail as in movies.json
- **myrating**: rating given by the user

4. `userReviews.json`:
   - **Movie Title**
   - **positive column**: Positive reviews structured in numbers which has *title*, *review* and *username*.
   - **negative column**: Negative reviews structured in numbers which has *title*, *review* and *username*.
5. `users.json`
   - **email**
   - **password**

9 Project SetUp

Download the submission folder. Extract it using command:

```
unzip <submission folder name>.zip
```

During the whole process, make sure that you have internet connection, as images, scripts, trailers, and bootsrap styles are all loaded online. Move into the `submission` folder. First of all make sure nodejs and npm are installed on your PC. If not then install it first using: `https://nodejs.org/en/download`. After Installing, Move into the `project_ejs_express` Folder of the `submission` folder and write(This is required to do only once):

```
npm i
```

on your terminal. It will install the necessary packages.
Then type in the `project_ejs_express` Folder

```
node 1.js
```

on the terminal to run the server. The server is running on port 3000. Please **DO NOT CLOSE IT.**
Go to `http://localhost:3000/` on your browser. Enjoy the site!

To check on other devices:
First connect your device(both your laptop and the device) to same WiFi network.

Then, follow these steps:

1. **Start Your Node.js Server:** Run your Node.js project/server on your computer. Navigate to the directory containing your Node.js project in the terminal or command prompt and run your server script using the `node` command. For example:

```
node 1.js
```

2. **Find Your Local IP Address:** Determine the local IP address of your computer where the Node.js server is running. You can typically find this information in your network settings or by running a command like `ipconfig` (on Windows) or `ifconfig` (on Unix-based systems) in your terminal or command prompt.

3. **Access Your Node.js Server:** Once your Node.js server is running, you should be able to access it from any device connected to the same WiFi network. On another device (e.g., smartphone, tablet, or another computer), open a web browser and navigate to your computer's local IP address followed by the port number. For example:

   ```
   http://192.168.x.x:3000
   ```

   Replace `192.168.x.x` with your actual local IP address and `3000` with the port number your Node.js server is listening on.

To stop the server press `CTRL+C` or `CMD+C` on the terminal on which server was running...

Bibliography

[1] Bootstrap classes: W3schools. https://www.w3schools.com/bootstrap/bootstrap_ref_all_classes.asp.

[2] Bootstrap components. https://getbootstrap.com/docs/4.0/components/card/.

[3] Click Buttons in selenium. https://www.geeksforgeeks.org/how-to-click-a-button-on-webpage-using-selenium/.

[4] Express js docs. https://expressjs.com/en/starter/hello-world.html, https://expressjs.com/en/resources/middleware/body-parser.html#bodyparserurlencodedoptions, https://expressjs.com/en/resources/middleware/body-parser.html#bodyparserjsonoptions, https://expressjs.com/en/4x/api.html#res.render.

[5] Fake User Agents. https://scrapeops.io/web-scraping-playbook/403-forbidden-error-web-scraping/.

[6] IMDB top 250 Movies. https://www.imdb.com/chart/top/.

[7] Keeping user session. https://www.geeksforgeeks.org/session-management-using-express-session-module-in-node-js/.

[8] Problem Statement. https://docs.google.com/document/d/1EKm1F_SBxaBs4hLl5LWFjiimTbtBdOCRdUVhMCLRZ4M/edit.

[9] Selenium Documentation. https://www.selenium.dev/documentation/.

[10] Storing user info using session. https://dev.to/readymadecode/how-to-use-session-in-nodejs-2c5e.

[11] W3School - Xpath in HTML. https://www.w3schools.com/xml/xpath_intro.asp.

[12] Web development codewithharry.

[13] Why is local File writing not allowed using client-side JS? https://stackoverflow.com/a/585247.

[14] YouTube API Docs. https://developers.google.com/youtube/v3/docs/search/list.

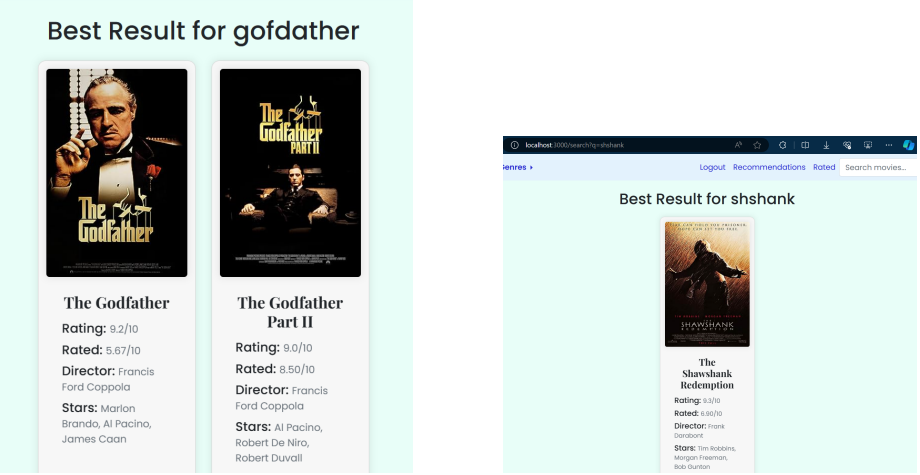
[15] YouTube Selenium Tutorial. https://www.youtube.com/watch?v=6M6BggVcUso.

[16] Stack Overflow Community. Stack Overflow Answer. https://stackoverflow.com/a/22125510.

[17] Stack Overflow Community. Stack Overflow Answer. https://stackoverflow.com/a/17436663.

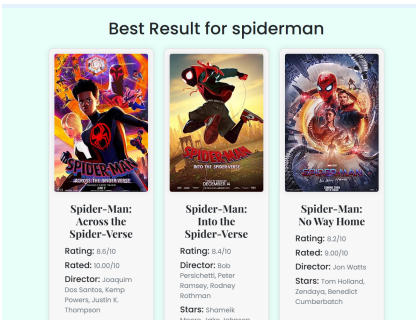
10 Appendices

10.1 Searching Samples

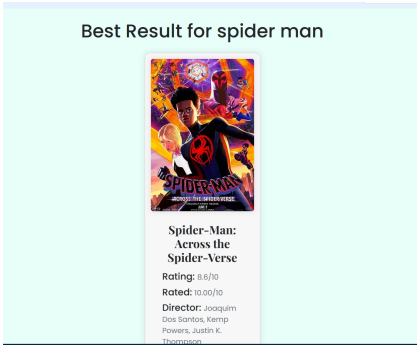


(a) Query: gofdather (b) Query: shshank

Figure 13: Search sample



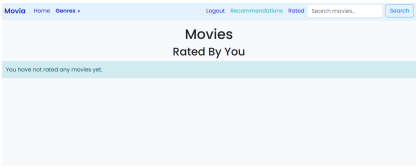
(a) Query: spiderman



(b) Query: spider man

Figure 14: Search Examples

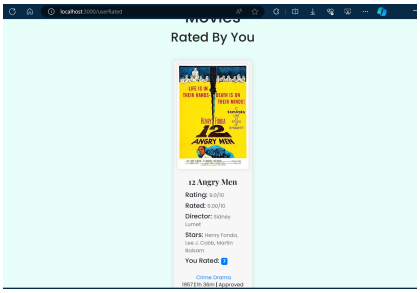
10.2 Recommendation Example



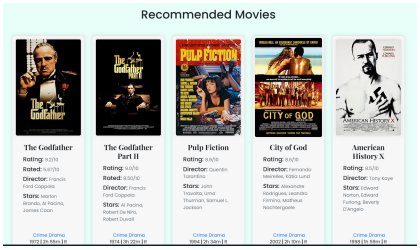
(a) Rated Page

(b) Recommend Page

Figure 15: New User

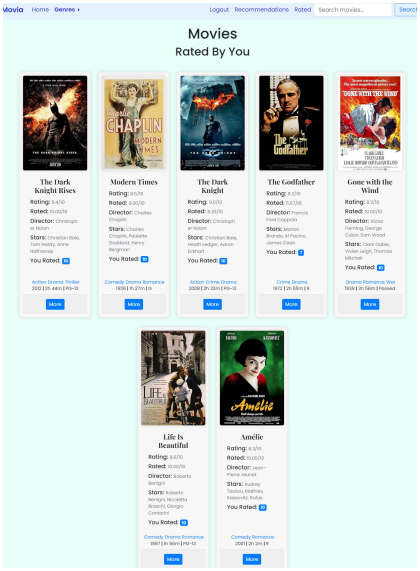


(a) One movie rated

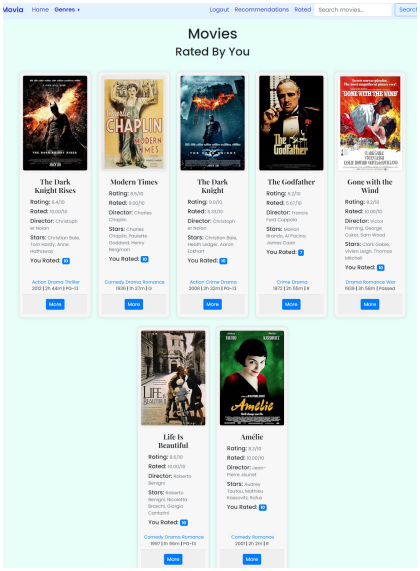


(b) Recommendation

Figure 16: Recommendation Examples



(a) Many Movies rated



(b) Recommendation

Figure 17: Recommendation Example