# CS 108 Project - Movie Mania

IIT Bombay

Awez

April 28, 2024

## Contents

# 1 Objective

To design a website interface which serves as a one stop movie repository, by scraping data from online repositories where users can search about movie information and to design an algorithm to recommend movies to users based on their ratings. Additional customisations have been added.

# 2 Usage Instructions

## 2.1 Requirements

- Node JS installed on your device.
- A web browser.
- Internet (for loading Bootstrap CDN, given it's not in browser cache)

## 2.2 Instructions

- Extract the `cs108-project.zip` file
- Open the directory through your terminal
- Navigate to `cs108-project/web/` directory
- Run `node app.js`
- You'll get an output like `Ctrl + Click on this link: 'link'`, open that link with your browser, or `Ctrl + Click` the link if it works.

# 3 Introduction

The whole project is divided into 4 categories, namely

- Data Extraction
- Website Design
  - Developing backend to display extracted data.
- Movie Recommender
- Customizations
  - Implementing a Registration System
  - Scraping user/critic reviews from online repositories
  - User based recommender
  - Spell Check
  - UI Enhancement

Let's dive deep into each category to see how the website works

# 4 Data Extraction

## 4.1 Objective

To collect information about top IMDb movies.

## 4.2 What I've used

1. **Selenium** - It's a python module which provides **webdriver**, this is like an automatic browser which can open a url, click links/buttons in it etc without the need of manual control. It's faster than us too.
   (a) It can group a part of html as a `WebElement` object instance. This WebElement is what can be clicked/scrolled into view by the webdriver. Note that this can also be done by webdriver, but this is used to find specific elements and click them.
2. **Scrapy** - Also a python module, this can take a `WebElement` and extract the text, attribute information like `src`, `href` etc. of the HTML content in the respective `WebElement`.
3. **Chromedriver** - `webdriver` needs a browser support, with chromedriver, the automatic browser made by webdriver is basically chrome.
4. **Numpy**: Used this just for `numpy.NaN` for undefined number values like movie rating etc.
5. **Pandas**: Used this to create a csv file containing all the movie data, which I later converted into a .json file using another script.
6. **time**: A python library, which I used to wait till something loads etc.
7. : To learn about how selenium & scrapy are used, my code structure is inspired by their examples.

## 4.3 What I've done

1. It first loads up the IMDb top 250 movies page.
2. Then it makes a list of `WebElement`s of all movies, whose info page it clicks afterwards.
3. Then it uses scrapy's Selector() to extract the following data of each movie:
   (a) **Title**
   (b) **Image** - Link of image displayed by IMDb
   (c) **Release Year**
   (d) **Age Rating**
   (e) **Duration of the movie**

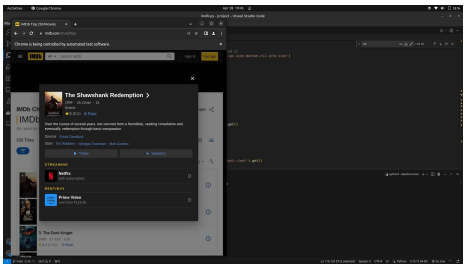


Figure 1: Data scraping by chrome webdriver

(f) **Genres** - as a list

(g) **Rating** - according to IMDb

(h) **Plot** of the movie

(i) **Directors** - as a list

(j) **Cast** - as a list

(k) **Trailer** - Link as given by IMDb

(l) **Streaming Platforms** - List of two lists, first one containing names of platforms and the second one containing corresponding links to those platform (trailer specifically)

(m) **Ratings by Other Sites** - I've extracted them using another script (./dataExtraction/findGoogleData.py) which searches google for the movie reviews, and then selects the reviews (3) as displayed by google. Ofcourse, the order and raters could be different for each movie

It makes list for each data.

4. Then **Pandas** comes into use to merge these arrays into a 2D csv file - named imdb.csv (in dataExtraction/imdb.csv)

5. Then converted imdb.csv to imdb.json, since it's easy to parse and use, Using another script ./dataExtraction/converters/csvToJson.py

## 4.4 Challenges Faced (and how I solved)

1. **Learning many functions from Selenium, Scrapy modules** -

   (a) Used official documentation from their respective websites

   (b) Reference mentioned

   (c) Used help from AI (chatgpt, github copilot) to get to know about the functions of these modules and how they're used

2. **Driver was clicking button before they're even loaded** -

   (a) used `WebDriverWait` class which has an `until` function which wait untils the `WebElement` is clickable (in the view & completely loaded)
   (b) If it's not clickable, the driver scrolls down using the function `location_once_scrolled_into_view` until it's clickable

3. **Scraping Ratings from websites other than IMDb** -
   (a) Used google search for each movie's reviews, took the top reviews.
4. **webdriver is slow** -
   (a) Tried using it in headless mode, using an Option() class argument (which is available in selenium module)
   (b) Still took a lot of time, let it run overnight.
5. **Couldn't Gather ratings data of 6 movies** -
   (a) These movies are: - The Hunt, The Kid, Rocky, To Be or Not to Be, Psycho, Untouchable
   (b) This was because the ratings data couldn't load when the scrapy.Selector() tried to extract data
   (c) Made another python script to extract this data seperately.
   (d) It's present as (./dataExtraction/findGoogleData.py)

# 5 Making Website

## 5.1 Objective

To make a basic website where I can search for a movie, get it's info from imdb.json

## 5.2 What I've used

1. `Node.js` - handles the server
2. `ejs` - to render .HTML files dynamically
3. `express.js` - to handle request, response cycle
4. `bodyParser` - to parse HTTP requests
5. `fileSystem module` - to use server files

### 5.3 What I've done

1. I have an `app.js` file, which is my server & `index.ejs` file which would render the home page
2. I've kept all .ejs files in ./web/views directory since the renderer checks views/ by default.
3. There's a movies array, which contains all the searched movies
4. When there's a GET request to /, i.e home, it renders `index.ejs` which contains a form to enter movie name & a submit button.
5. When submitted it sends a POST request to /submit. This takes the request and processes and adds it to movies array, and redirects to /
6. / then checks last element of movies array and displayes index.ejs accordingly.
7. Note that we change all the request responses to lower case, hence the search is case insensitive

### 5.4 Challenges Faced (and how I solved)

1. While parsing the data, each element of the array was enclosed in ", but JSON.parse() only works when they're enclosed in " ", so had to replace each ' with ".

## 6 Recommender System

### 6.1 Objective

Recommending the user 5 movies based on the movie user has rated

### 6.2 What I've Used

1. Nothing new, whatever I've mentioned before.

### 6.3 What I've Done

1. Used a different route (/ratings) for this page, which when sent GET request renders 'ratings.ejs' page.
2. This page has a form which takes movie name (again with case insensitivity) and corresponding rating, which is sent through a POST request to /ratings.
3. /ratings stores all the rated movies in an array and finds out top 5 best to suggest movies with them using the following algorithm:

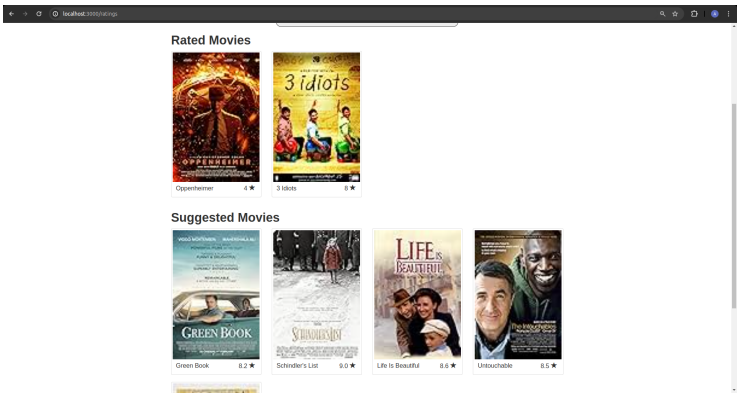


Figure 2: Ratings Page

(a) First I'll calculate average rating of each genre, if the genre isn't there in any movie rated, it's average rating is 0

$$\text{avg\_rating(genre)} = \frac{\sum_{\text{movie has genre}} \text{rating(movie)}}{\text{number of movies}}$$

(b) Then I'd assign each genre a weight, which is directly proportional to the frequency of genre

$$\text{weight(genre)} = \frac{\sum_{\text{movie has genre}} \text{rating(movie)}}{\sum_{\text{all movies}} \sum_{\text{movie has genre}} \text{rating(movie)}}$$

(c) Then I'd give each movie a score which is a weighted sum of average rating of each genre (weight as calculated above)

$$\text{score(movie)} = \sum_{\text{genres of movie}} \text{weight(genre)} * \text{avg\_rating(genre)}$$

(d) According to me, movies with higher score are better suggestions, so i rate all the movies according to their score and filter out the top 5 movies and suggest them.

(e) All this can be seen in `suggestMovies()` function in `./web/app.js`

## 6.4 Possible Drawbacks

1. It just compares movie with respect to genre, irrespective of language etc.
2. It considers only ratings of the current user, which isn't even for the suggested movies, ratings of the suggested movies by people of similar interest of him should be taken into account

# 7 Customisations

## 7.1 User Registration

### 7.1.1 Objective

To allow a user to make (sign up) his account, and use his account (by logging in) everytime he uses the website to get personal recommendations.

### 7.1.2 What I've used

1. **Query Strings** - When a user is using the website, all the routes are configured to carry the query string "username=username_of_the_user"

### 7.1.3 What I've done

1. I've created a .json file in the server, you can find it as `users.json`
2. This .json file stores username, password, rated movies, corresponding ratings of users as objects
3. Whenever the website has to use this data it uses fs.readFile() function
4. Whenever it has to append the data (like when signing up) it uses fs.writeFile() function
5. I've create a signup page which has a form, GET request to /signup renders this `signup.ejs`
6. When the form is submitted, a POST request is sent to /signup, which verifies if the username is already taken or not, if yes, it gives an alert, else it appends `users.js`
7. I also have a login page which has a form, POST request to /login renders this `login.ejs`
8. When the form is submitted, a POST request is sent to /login, which verifies if the username and password is correct, if yes, it gives an alert, else it starts using the query string I've mentioned.
9. When there's a query string "username=username" is used, instead of sending request to /ratings, requests are now sent to /userRatings route, which renders userRatings.ejs

### 7.1.4 Possible Drawbacks

1. This doesn't have proper authentication, because if some other person can use the query string and use someone's account without their password.

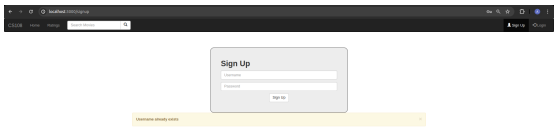


Figure 3: Signup Page

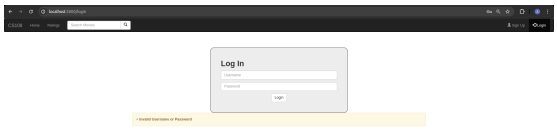


Figure 4: Login Page

## 7.2 Scraped Critic Reviews

### 7.2.1 Objective

To display rotten tomatoes top critic reviews.

### 7.2.2 What I've used

1. **ast** - a module of python which can destringify a stringified array in complex situations, i.e when an array is in string form like this "['awez', 'meh/'tab']", it's hard to do it using normal methods, ast module has a function `literal_eval` which does this task easily

### 7.2.3 What I've done

1. Used url manipulation to get rotten tomatoes reviews, like I used url of the form https://www.rottentomatoes.com/m/movie_name/reviews/?type=top_critics where movie_name is a specific version of the original movie name, eg. Schindler's List becomes schindlers_list

2. I've done this through the script `./dataExtraction/rt.py`

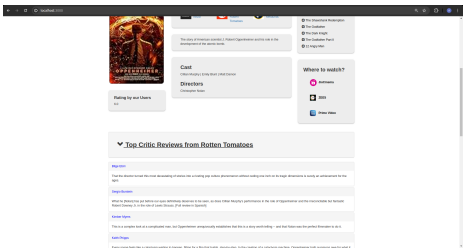


Figure 5: Rotten Tomatoes Reviews

#### 7.2.4 Challenges Faced

1. Loading Metacritic was very slow, hence I left reviews of metacritic
2. Some Movies got skipped in rotten tomatoes
3. So I had to use webdriver again, to search the movie specifically, specifying director, cast in search in rotten tomatoes, I've done this through `./dataExtraction/rtImprovement.py` script.
4. There was still a problem since few movies didn't have reviews at all.
5. These are those movies: - 'Spider-Man: Across the Spider-Verse','Your Name.', 'Jai Bhim', 'My Father and My Son'
6. For 'Star Wars: Episode VI - Return of the Jedi', the first search result was always of some 40th-anniversary version of it, so I had to do it manually.

### 7.3 SpellCheck

#### 7.3.1 Objective

When a user searches a wrong movie, or tries to rate a wrong movie, I should show 5 potential corrections of what he searched.

#### 7.3.2 What I've used

1. **Levenshtein's distance**- It defines a "distance" between two words quantifying how close two words are, by measuring minimum number of edits (insertion, deletion, replacements) of characters would take from one word to another. It's recursive relation is given by

$$\text{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0 \\ |b| & \text{if } |a| = 0 \\ \text{lev}(\text{tail}(a), \text{tail}(b)) & \text{if head}(a) = \text{head}(b) \\ 1 + \min \begin{cases} \text{lev}(\text{tail}(a), b) \\ \text{lev}(a, \text{tail}(b)) \\ \text{lev}(\text{tail}(a), \text{tail}(b)) \end{cases} & \text{otherwise} \end{cases}$$

2. **fastest-levenshtein** - an npm module which has `distance()` function which calculates levenshtein's distance in an efficient way.

#### 7.3.3 What I've done

1. I defined my own distance using `levenshtein's distance`, this can be seen in `my_distance()` function in `./web/app.js`
2. It works this way:
   (a) if the length of the searched term is less than a movie, search for minimum levenshtein distance between searched term and a substring of movie of same length
   (b) if the length of the searched term is higher, use only levenshtein distance

   This way, if the searched term is completely a substring of some movie, `my_distance(searched\_term, movie)` would be 0 hence giving highest priority.
3. There are many cases when user would just search a substring of the movie they want to search, this is most relevant then.

#### 7.3.4 Possible Drawbacks

1. I'm giving 5 suggestions, which is really helpful for some search terms
2. But in few cases, it gives irrelevant suggestions
3. I couldn't modify the algorithm properly to remove irrelevant suggestions because in that case it's narrowing the chance of showing our thought's movie, which is in major cases (in my opinion)

### 7.4 Rating meter

#### 7.4.1 Objective

To show the average rating from other users.

#### 7.4.2 What I've done

1. Read through each user in the `users.json` file and checking if the movie has been rated if yes we'll add it to our user_rating(movie) and then divide it by number of users who've rated that movie

No possible drawbacks I feel. Except that it doesn't show anything when no one has rated that movie

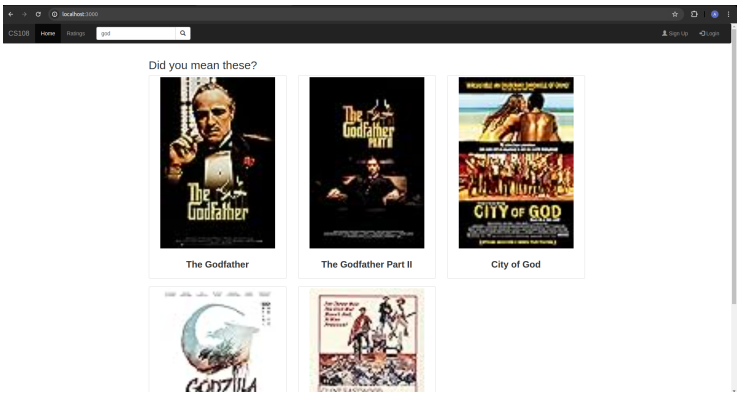


Figure 6: Spell Check

## 7.5 Recommender For Registered Users

### 7.5.1 Objective

To show a registered user suggestions based on what they've rated before

### 7.5.2 What I've done

1. Used users.json file to get what the user has rated and used the algorithm given above to give 5 suggestions.

### 7.5.3 Possible Drawbacks

It's still considering genre as the only accounting factor. Whereas rating from other users of similar interest should also have been taken

## 7.6 Enhanced UI

### 7.6.1 Objective

To make the website look more appealing and user friendly.

### 7.6.2 What I've used

1. Bootstrap - A CSS framework which facilitates styling in a much simpler way. Also makes the styling responsive
2. Bootstrap CDN - So that we don't need to have Bootstrap installed on our system, but might need internet, if the browser doesn't have it in cache
3. CSS - Usual plain CSS

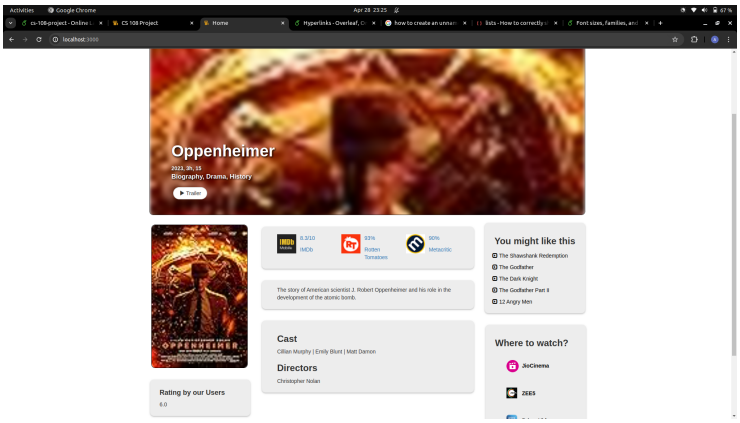


Figure 7: Home Page

### 7.6.3 What I've Done

1. Made a nav bar for searching, user info, home, ratings page, login, signup pages.
2. Placed the items in groups called containers
3. This allows them to be responsive
4. Used Bootstrap classes
5. Showed sugggested (top 5 IMDb movies) for a non-registered user on home page
6. Showed suggested movies for registered users (by our algorithms)
7. .CSS files are present in `./web/css`

# Bibliography


- `https://www.analyticsvidhya.com/blog/2022/07/scraping-imdb-reviews-in-python-using-selenium/`
- `https://www.sitepoint.com/build-a-simple-web-server-with-node-js/`
- CS108 Lecture & Tutorial Slides
- Help from chatgpt, gemini while learning about backend, selenium

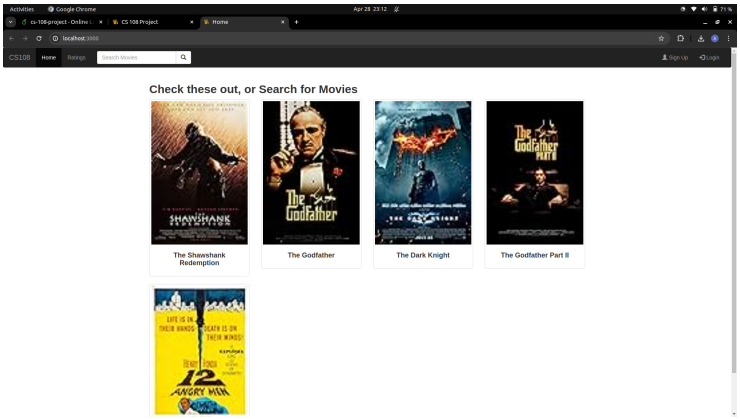


Figure 8: Suggestions in Home Page