



# Dharmsinh Desai University, Nadiad

Faculty of Technology, Department of Computer Engineering

B.Tech. CE Semester – VI

Subject: System Design Practice

Project Title:

## Movie Recommendation Engine

Submitted By:

Harsh Vaghasiya (ID:17CEUOS053) (ROLLNO.:CE139)

Sani Ranpariya (ID:17CEUOF017) (ROLLNO.:CE112)

Prakruti Vavdiya (ID:17CEUOS079) (ROLLNO.:CE143)

Guided By:

Prof. Hariom A. Pandya



# Dharmsinh Desai University, Nadiad

Faculty of Technology, Department of Computer Engineering

## **CERTIFICATE**

This is to certify that System Design Practice project entitled “Movie Recommendation Engine” is the bonafied report of work carried out by

**1) Harsh Vaghasiya (17CEUOS053) (CE139)**

**2) Sani Ranpariya (17CEUOF017) (CE112)**

**3) Prakruti Vavdiya (17CEUOS079) (CE143)**

Of Department of Computer Engineering, Semester V, academic year 2019-20,  
under our supervision and guidance.

---

**Guide**

**Prof. Hariom A. Pandya**  
Assistant Professor of  
Department of Computer  
Engineering, Dharmsinh Desai University,  
Nadiad.

**HOD**

**Dr. C. K. Bhensdadia**  
Head of the Department of  
Department of Computer  
Engineering, Dharmsinh Desai University,  
Nadiad.

## Table of Contents

1 Abstract.....	4
2 Introduction.....	5
2.1 Project Details: Brief Introduction .....	5
2.2 Technology and Tools Used .....	5
3 Software Requirement Specifications .....	6
3.1 Scope .....	6
3.2 System Functional Requirements .....	6
3.3 Other Non-Functional Requirements .....	10
4 Design.....	11
4.1 Use-case Diagram .....	11
4.2 Sequence Diagram .....	13
4.3 Activity Diagram .....	14
5 Implementation Details .....	16
5.1 Dataset Used .....	16
5.2 Related Works/ Literature Survey .....	18
5.3 Proposed Solution .....	18
5.4 Model Building And Function Prototypes .....	19
5.5 Common Terminology .....	24
6 Testing .....	26
7 Screen-shots of the System .....	27
8 Conclusion .....	33
9 Limitations and Future Extensions of System .....	33
10 Bibliography .....	33

# 1. Abstract

---

Most internet products we use today are powered by recommender systems. Youtube, Netflix, Amazon, Pinterest, and long list of other internet products all rely on recommender systems to filter millions of contents and make personalized recommendations to their users. Recommender systems are well-studied and proven to provide tremendous values to internet businesses and their consumers. If we look at the root of any recommendation engine, they all are trying to find out the amount of similarity between two entities. Then, the computed similarities can be used to deduce various kinds of recommendations and relationships between them.

## 2. Introduction

---

### **2.1 Brief Introduction**

A movie recommendation Engine is a machine learning based web application that recommend you movies based on your preferences. The site may be focused on your activities or other users' choices. The purpose of website is an established fact that Internet users are increasing today. One of the main purposes of the website is to facilitate the user with the best movie recommendations based on his taste with the intention to save users time so that they don't have to spend their precious time and effort trying to find out the best movie to watch. We will be putting an effort to provide the right choice to the people when they want to watch a movie. In this website you can get recommendations based on different parameters and rate watched movies which helps you and others getting better recommendations.

The application provides specialized suggestions It saves your time for searching movies according to your taste. We help users to find movies according to their taste and according to ratings given by them and other users. The application's aim is to provide helpful recommendations based on user's preferences.

### **2.2 Tools/Technologies Used**

**Technologies: Flask, Python3, HTML, CSS, Bootstrap**

**Libraries: sklearn, pandas, scipy, fuzzywuzzy**

## 3. Software Requirement Specifications

---

### **3.1 Project Scope**

Our project scope is to provide efficient platform for movie recommendation based on user's preferences and not on a popularity basis. Our project will help people to find movies for watching very easily according to their taste.

### **3.2 User types**

**1. End user**

**2. Admin**

**1 End user:**

**R.1: Login:**

Description: User can enter his credential as Email and

Password and can explore personalized functionalities.

Input: Email and password

Output: Confirmation message

Process: Authentication and redirect to home page

Exception: If not authenticated then display error

**R.2: Register:**

Description: User can enter his details like first name, last name, email, password, contact no. etc. for registration process.

Input: First name, Last name, Email-Id, Password, Contact no.

Output: Confirmation message

Process: User-Registration into database and redirect to login page

Exception: If credentials are improper shows error message

### **R.3: View movies recommendations:**

Description: User can view all movie recommendation based on his/her preferences.

Input: User-selection

Output: Movies Recommendations

#### **R.3.1 View Movie:**

Description: User can view any movie's info.

Input: Selected Movie

Output: Movie info

##### **R.3.1.1 Give Ratings:**

Description: User can give ratings to any movie.

Input: Ratings

Output: Ratings records in database are updated.

### **R.4: Search movie:**

Description: User can search any particular movie by its name.

Input: Movie Name

Output: Matching movie names list is shown

**R.5: View Profile:**

Description: User can view his/her profile.

Input: User-selection

Output: User-profile

**2 Admin:****R.6: Manage movies:**

Description: Admin can add movie, search movie, view movie, view movie statistics.

**R.6.1: Add movie:**

Description: Admin can add new movie by entering its details.

Input: new movie details

Output: Movie is added into the database.

**R.6.2: Search movie:**

Description: Admin can search any particular movie by its ID.

Input: Movie ID

Output: Movie details

**R.6.3: View Movies:**



Description: Admin can view all available movies-list.

Input: User-selection

Output: Movies-list

#### **R.6.4. View Movie Statistics:**

Description: Admin can view movie's statistics info.

Input: Selected Movie

Output: Movie Statistics details

### **R.7: Manage users:**

Description: Admin can search user, view user, view  
user statistics.

#### **R.7.1: Search user:**

Description: Admin can search any particular user by its ID.

Input: User ID

Output: User details

#### **R.7.2: View Users:**

Description: Admin can view all available users-list.

Input: User-selection

Output: Users-list

#### **R.7.3. View User Statistics:**

Description: Admin can view user`s statistics info.

Input: Selected User

Output: User Statistics details

### **3.4 Other Nonfunctional Requirements**

#### **1. Performance**

The system must be interactive and the delays involved must be less. So in every action-response of the system, there are no immediate delays. In case of opening App components, of popping error messages and saving the settings or sessions there is delay much below 3 seconds.

#### **2. Safety**

User details should be securely stored to the server. The main security concern is for user account hence proper login mechanism should be used to avoid hacking.

#### **3. Reliability**

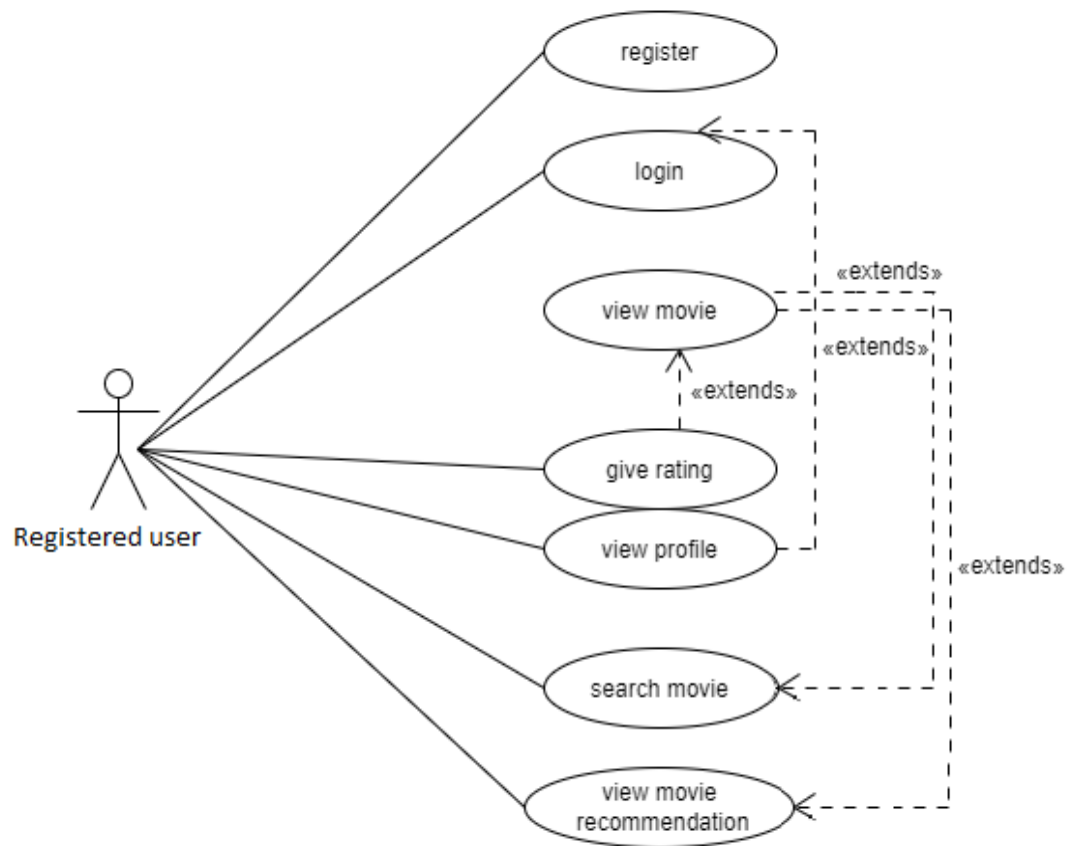
As the system provides the right tools for discussion, problem solving it must be made sure that the system is reliable in its operations and for securing the sensitive details.

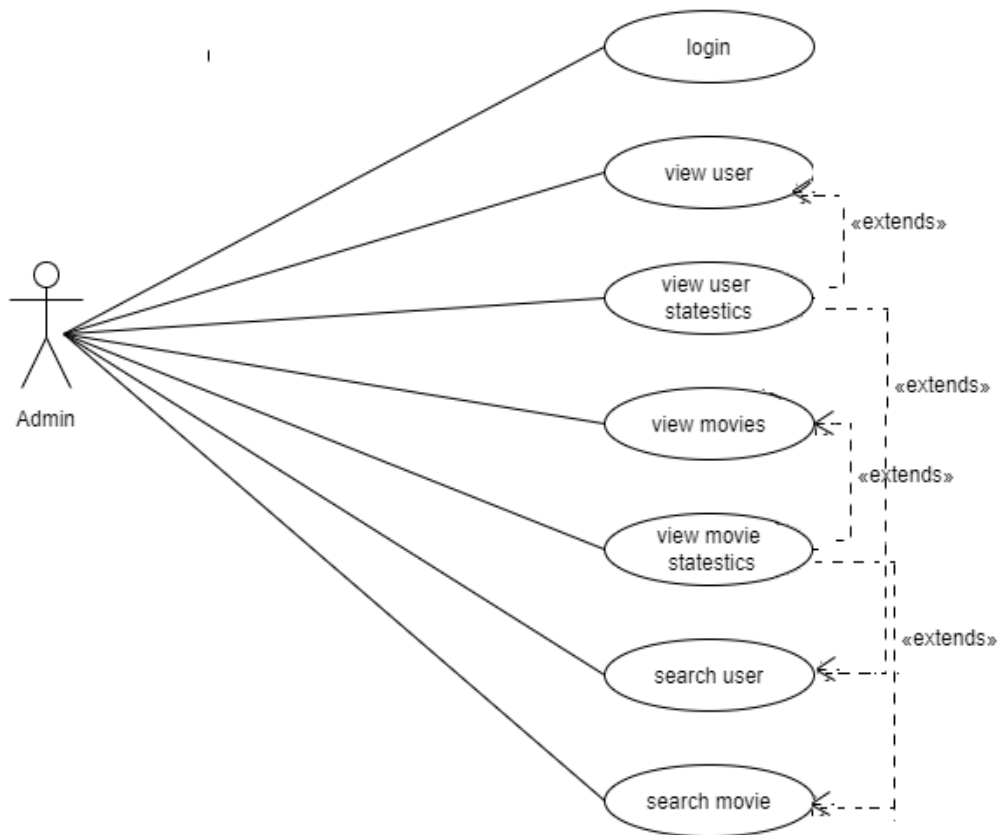
#### **4. Database**

System requires to access users data fast to maintain the performance.

## 4. Design

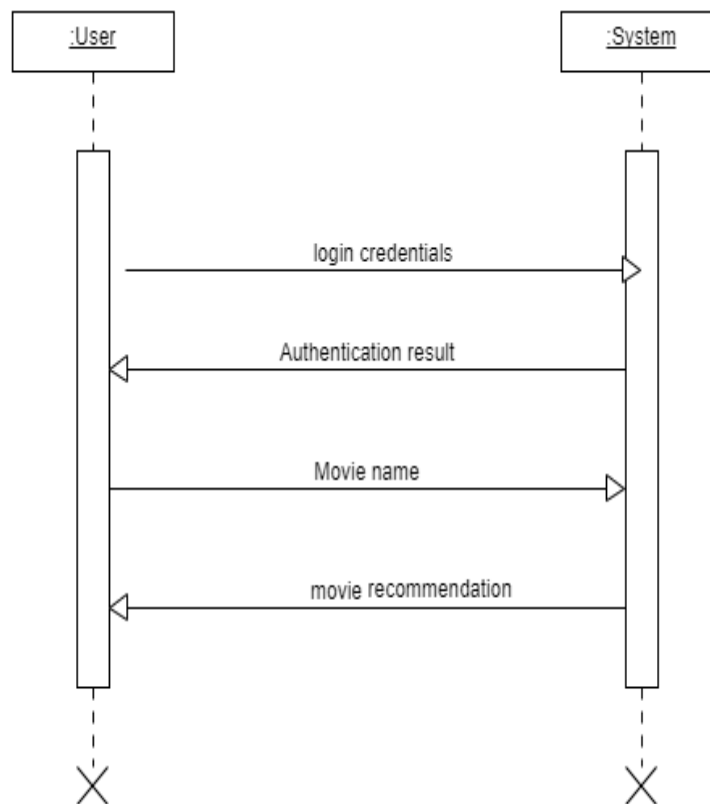
### 4.1 Use-case Diagram:





## 4.2 Sequence Diagram:

View movie recommendations



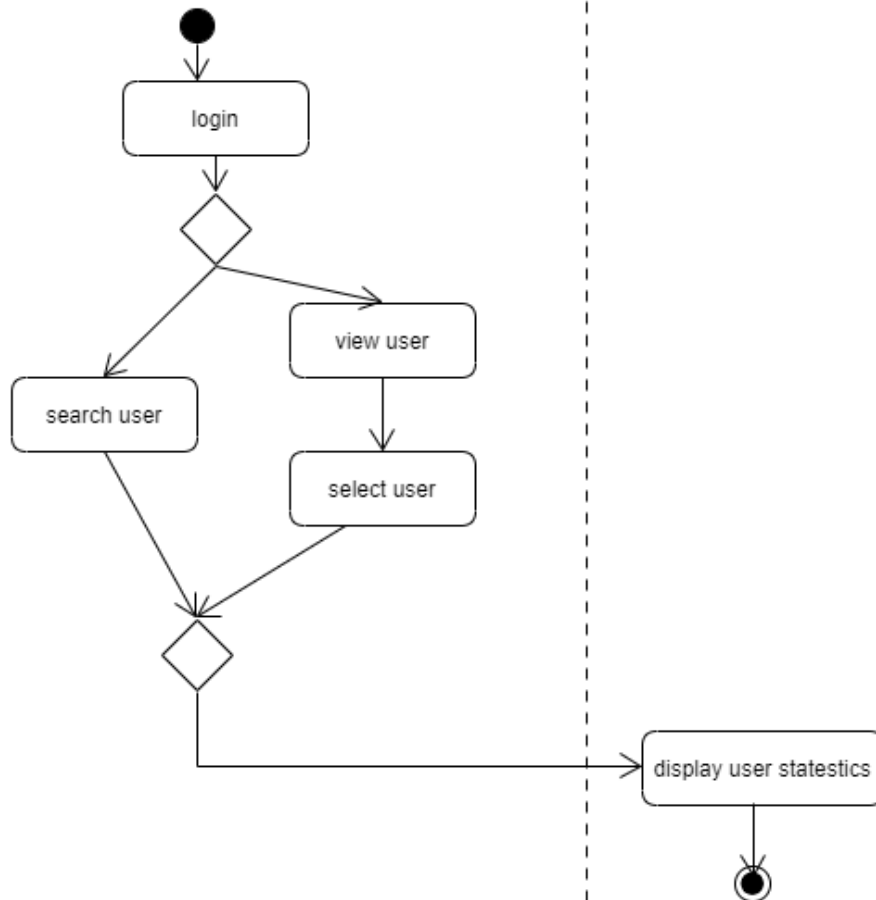
### 4.3 Activity diagram:

View user statistics

User



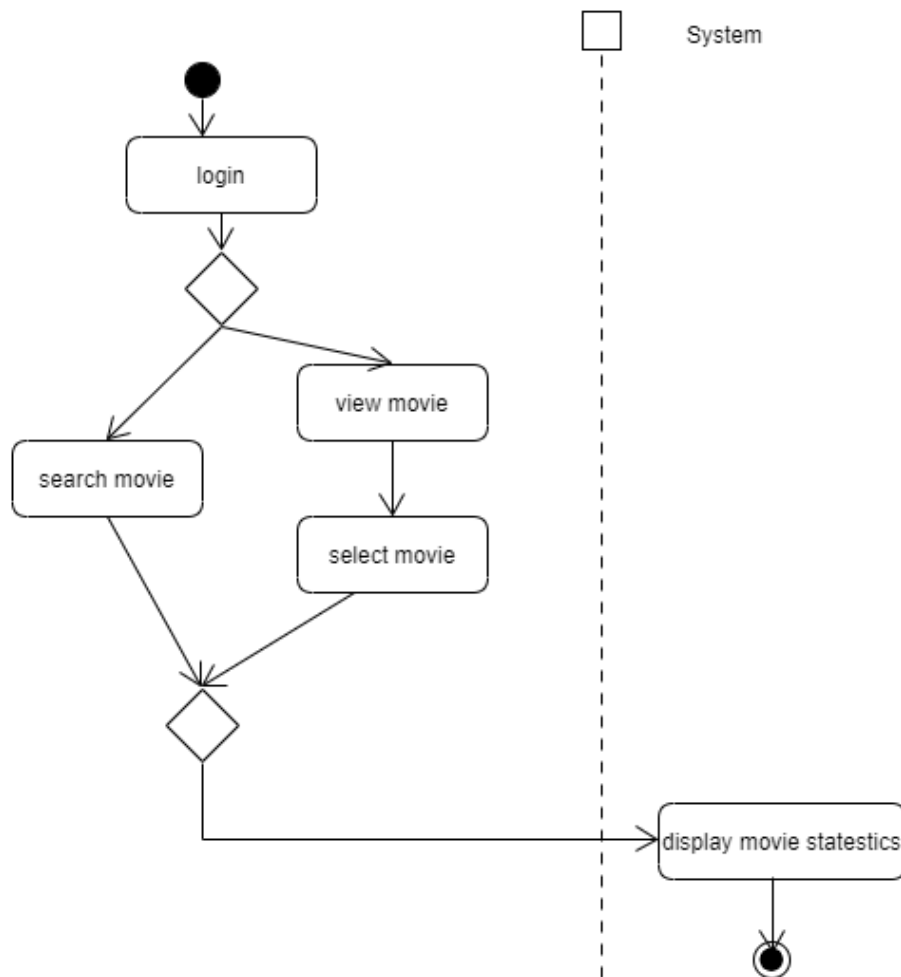
System



## View movie statistics

User

System



## 5. Implementation Details

---

### **5.1 Dataset Used:**

We used a standard MovieLens Dataset from official website of GroupLens , [www.grouplens.org](http://www.grouplens.org) . The details of the dataset are as following:

1) users.csv : It contains records of total 610 users.

userId: Unique Id of user

first\_name: First name of user

last\_name: Last name of user

email: Email-Id of user

password: Password of user

phone: Contact no. of user

ratingCount: No. of movies to which user has given ratings

2) movies.csv : It contains records of almost 9740 movies. Movies can have any Genre like, Action, Adventure, Animation, Children's, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, Western etc.

movieId: Unique Id of movie

title: Title of movie

genres: Genres list of movie

ratingCount: No. of users who has given ratings to movie



avgRating: Average rating of movie

3)ratings.csv : It contains info about which user has given how much rating to a particular movie. It has almost 100000 ratings records.

userId: Unique Id of user

movieId: Unique Id of movie

rating: Rating given by user to the movie

4)links.csv : It contains important source links of all movies in movies.csv file.

movieId: Unique Id of movie

imdbId: Unique imdbId of movie

tmdbId: Unique tmdbId of movie

5)tags.csv : It contains almost 3600 tag applications for movies in movies.csv file.

userId: Unique Id of user

movieId: Unique Id of movie

tag: Tag given by the user to movie

6)userAvgRating.csv : It contains relationship between user and average rating given to each genre by user.

7)userRatingCount.csv : It contains relationship between user and no. of

ratings given to each genre by user.

## **5.2 Related Works/Literature Survey:**

During the survey, we found that Recommendation Engines are mostly based on the following techniques:

1. Popularity Based Filtering.
2. Collaborative Filtering (User Based / Item Based)
3. Content Based Filtering.
4. Hybrid Filtering

First of all , Popularity based filtering technique is mostly common in general to all users and we wanted our model to be user-specific , so we decided to discard this technique from our solution.

Then, Content based filtering doesn't take into account what others think of the item, so low quality movie recommendations might happen. That's why we also decided to discard this technique from our solution and hence Hybrid filtering technique was also discarded.

Hence, finally we decided to build model using COLLABORATIVE FILTERING technique.

## **5.3 Proposed Solution:**

In collaborative filtering, two entities collaborate to deduce recommendations on the basis of certain similarities between them. These filtering techniques are broadly of two types:

1. **User Based Collaborative Filtering:** In user based collaborative filtering, we find out the similarity score between the two users. On the basis of similarity score, we recommend the movies liked by one user to other user assuming that he might like these items on the basis of similarity. Major online streaming service, **Netflix** have their recommendation engine based on user based collaborative filtering.

- 2. Item Based Collaborative Filtering:** In item based collaborative filtering, the similarity of an movie is calculated with the existing movie being consumed by the existing users. Then on the basis of amount of similarity, we can say that if user X likes movie A and a new movie P is most similar to movie A then it highly makes sense for us to recommend movie P to user X.

As both approaches have their own set of specific advantages, we decided to build our model using both approaches. So user can view all recommendations from User based Collaborative Filtering and if user visits any movie then similar movies can recommended using Item Based Collaborative Filtering.

## **5.4 Model Building And Function Prototypes:**

We decided to build hybrid model using both approaches of Collaborative Filtering. Detailed description both model are as following:

### **1. User Based Collaborative Filtering:**

- First of all we imported data from all .csv files from our dataset into our program and converted it into dataframes using Pandas library.

```
def recommend(userEmailAdd):
    movie = pd.read_csv('ml-latest-small/movies.csv',encoding = "ISO-8859-1")
    rating = pd.read_csv('ml-latest-small/ratings.csv')
    imdb = pd.read_csv('ml-latest-small/links.csv')
    userData=pd.read_csv('ml-latest-small/users.csv')

    userID=userData[userData['email']==userEmailAdd]
    if (len(userID)==0):
        return False
    userID=userID['userId'].iat[0]
    gener =['Action','Adventure','Animation','Children','Comedy','Crime','Documentary','Drama','Fantasy','Film-Noir','Horror','Musical','Mystery','Romance','Sci-Fi','Thri
    user=rating['userId'].unique().tolist()
    similar_user=similarUser(rating, movie,gener,gener,user,userID,gener,user)
    dic=recomendation(rating,movie,imdb,similar_user[1],userID,similar_user[0],user,similar_user[2])
    return dic
```

- Then we passed this data to similar\_user() function which returns the userId of similar user to logged in user. In this function we inherently used sortdata() function to find out user's genre's preferences and we used cosine\_similarity() function from sklearn library between two users. And finally we sorted the similarity score and based on that we found out the best possible similar user.

```

def similarUser(rating, movie, genre, column,index,userID,gener,user):
    genre_ratings = pd.DataFrame() #dataframe for genre avg rating
    similar_User=[]
    userRating=rating[rating['userId']==int(userID)]
    m=[] #litr for movie similar userID and genrelist
    for g in genre:
        l=[]
        l.append(g)
        mgroup = movie[movie['genres'].str.contains(g)] #grp same genre movie
        avgRating = rating[rating['movieId'].isin(mgroup['movieId'])].loc[:, ['userId', 'rating']].groupby(['userId'])['rating'].mean().round(2)
        genre_ratings = pd.concat([genre_ratings, avgRating], axis=1) #concat genre with user
        totalRating=len(userRating[userRating['movieId'].isin(mgroup['movieId'])])
        l.append(totalRating)
        totalAvgRating=userRating[userRating['movieId'].isin(mgroup['movieId'])]
        totalAvgRating=totalAvgRating["rating"].mean()
        l.append(totalAvgRating)
        m.append(l)
    m=sorted(m,key=itemgetter(1),reverse=True) #sort array in descending order
    genre_ratings.columns = column #add column name
    genre_ratings=genre_ratings.fillna(0)
    genre_ratings=sortdata(genre_ratings,gener,userID) #sort data to get most four rated genre
    Genres_list=list(genre_ratings[0:1,:4].ravel() )
    Genres_list_most_view=[]
    for i in range(4):
        Genres_list_most_view.append(m[i][0])
    genre_ratings=pd.DataFrame(genre_ratings[1:,:4],columns=genre_ratings[0:1,:4].ravel()) #convert sorted array in dataframe
    genre_ratings=sim(genre_ratings) #find cosine similarity
    genre_ratings=pd.DataFrame(genre_ratings,columns=user) #convert into dataframe
    genre_ratings=sortdata(genre_ratings,user,userID) #sort user with similarity in descending order
    genre_ratings=pd.DataFrame(genre_ratings[1:,:],columns=genre_ratings[0:1,:].ravel()) #convert into dataframe
    genre_ratings=genre_ratings.loc[userID:userID,:] #similar user row for given user
    similar_User.append(genre_ratings.columns.values.tolist()) #append similar user id into list
    similar_User.append(Genres_list)
    similar_User.append(Genres_list_most_view) #append genre list into list
    return similar_User

```

```

#sort data by perticuer user rating
def sortdata(df,gener,user):
    user=int(user)
    narray=df.to_numpy()
    array=np.asarray(gener)
    array=np.vstack((array,narray))
    array=array.transpose()
    array=array[array[:,user].argsort()[::-1]]
    array=array.transpose()
    return array

```

- Now, as we have found out similar user, we gathered data about all those movies which are rated as high by similar user but are not rated by logged-in user. Thus we found out the best movie recommendation for logged-in user.

```

def recommendation(rating, movie, imdb, genre_list, userID, similar_user_Id, column, Genres_list_most_view):
    recommendedData={}
    ownRating=rating[rating['userId']==(int(userID))]          #user all rating data
    ownMovie=ownRating['movieId'].tolist()                     #movieid list for given user rated movie
    recommendedData['genreList']=genre_list
    for g in genre_list:
        i=0
        j=1
        li=[]
        while(i<6):
            userRating=rating[rating['userId']==(int(similar_user_Id[j]))]          #similar user all rating data
            j=j+1
            userMovie=movie[movie['movieId'].isin(userRating['movieId'])]           #similar user movie data which he give rating
            movieList=userMovie[userMovie['genres'].str.contains(g)]                 #movie data for perticular genre
            movieLists=userRating[userRating['movieId'].isin(movieList['movieId'])]   #movie rating for that perticular genre
            movieLists=movieLists.sort_values('rating',ascending=False)              #sort according to rating value
            for x in range(len(movieLists)):
                movieId=movieLists['movieId'].iat[x]
                if(movieId not in ownMovie ):
                    i=i+1
                    recommended=movie[movie['movieId']==movieId]
                    l=[]
                    l.append(recommended['movieId'].iat[0])
                    l.append(recommended['title'].iat[0])
                    imdbId=imdb[imdb['movieId']==movieId]
                    l.append(recommended['avgRating'].iat[0])
                    ln = "../static/img/Movie_Poster_Dataset/tt0"+str(imdbId['imdbId'].iat[0])+".jpg"
                    l.append(ln)
                    li.append(l)
            if(i==6):
                break
        recommendedData[g]=li

```

```

for g in Genres_list_most_view:
    i=0
    j=1
    li=[]
    while(i<6):
        userRating=rating[rating['userId']==(int(similar_user_Id[j]))]          #similar user all rating data
        j=j+1
        userMovie=movie[movie['movieId'].isin(userRating['movieId'])]           #similar user movie data which he give rating
        movieList=userMovie[userMovie['genres'].str.contains(g)]                 #movie data for perticular genre
        movieLists=userRating[userRating['movieId'].isin(movieList['movieId'])]   #movie rating for that perticular genre
        movieLists=movieLists.sort_values('rating',ascending=False)              #sort according to rating value
        for x in range(len(movieLists)):
            movieId=movieLists['movieId'].iat[x]
            if(movieId not in ownMovie ):
                i=i+1
                recommended=movie[movie['movieId']==movieId]
                l=[]
                l.append(recommended['movieId'].iat[0])
                l.append(recommended['title'].iat[0])
                imdbId=imdb[imdb['movieId']==movieId]
                l.append(recommended['avgRating'].iat[0])
                ln = "../static/img/Movie_Poster_Dataset/tt0"+str(imdbId['imdbId'].iat[0])+".jpg"
                l.append(ln)
                li.append(l)
            if(i==6):
                break
        recommendedData[g]=li

recommendedData['mostView']=Genres_list_most_view
return recommendedData

```

## 2. Item Based Colloborative Filtering:

- First we call make\_recommendation() function by passing input movie and no. of recommendation required as arguments.

```

def make_recommendations(fav_movie, n_recommendations):
    """
    make top n movie recommendations
    Parameters
    -----
    fav_movie: str, name of user input movie
    n_recommendations: int, top n recommendations
    """

    # get data
    movie_user_mat_sparse, hashmap = _prep_data()

    model_knn = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=20, n_jobs=-1)

    # get recommendations
    raw_recommends = _inference(
        model_knn, movie_user_mat_sparse, hashmap,
        fav_movie, n_recommendations)
    # print results
    reverse_hashmap = {v: k for k, v in hashmap.items()}
    print('Recommendations for {}'.format(fav_movie))
    for i, (idx, dist) in enumerate(raw_recommends):
        print('{0}: {1}, with distance '
              'of {2}'.format(i+1, reverse_hashmap[idx], dist))

```

- In this function first we prepare data in particular format using `_prep_data()` function, which ultimately prepares movie-user scipy sparse matrix and hashmap of movie to row index in movie-user scipy sparse matrix.

```
def _prep_data():
    # read data
    df_movies = pd.read_csv(
        r"D:\ML001_MOVIE_RECOMMENDATION_ENGINE-master\ml-latest-small\movies.csv",
        usecols=['movieId', 'title'],
        dtype={'movieId': 'int32', 'title': 'str'})
    df_ratings = pd.read_csv(
        r"D:\ML001_MOVIE_RECOMMENDATION_ENGINE-master\ml-latest-small\ratings.csv",
        usecols=['userId', 'movieId', 'rating'],
        dtype={'userId': 'int32', 'movieId': 'int32', 'rating': 'float32'})

    # filter data
    df_movies_cnt = pd.DataFrame(
        df_ratings.groupby('movieId').size(),
        columns=['count'])
    popular_movies = list(set(df_movies_cnt.query('count >= 0').index)) # noqa
    movies_filter = df_ratings.movieId.isin(popular_movies).values

    df_users_cnt = pd.DataFrame(
        df_ratings.groupby('userId').size(),
        columns=['count'])
    active_users = list(set(df_users_cnt.query('count >= 0').index)) # noqa
    users_filter = df_ratings.userId.isin(active_users).values

    df_ratings_filtered = df_ratings[movies_filter & users_filter]

    # pivot and create movie-user matrix
    movie_user_mat = df_ratings_filtered.pivot(
        index='movieId', columns='userId', values='rating').fillna(0)
    # create mapper from movie title to index
    hashmap = {
        movie: i for i, movie in
        enumerate(list(df_movies.set_index('movieId').loc[movie_user_mat.index].title)) # noqa
    }
    # transform matrix to scipy sparse matrix
    movie_user_mat_sparse = csr_matrix(movie_user_mat.values)

    # clean up
    del df_movies, df_movies_cnt, df_users_cnt
    del df_ratings, df_ratings_filtered, movie_user_mat
    gc.collect()
    return movie_user_mat_sparse, hashmap
```

- Then we develop KNN Model using NearestNeighbors() function of sklearn library in which we set the similarity metric to cosine. So NearestNeighbors() function return the KNN model consisting of nearest neighbours to the input movie based on user's ratings.
- Then we feed this model to \_inference() function which returns the best n possible similar movies to the input movies based on KNN model.

```

def _inference(model, data, hashmap,
               fav_movie, n_recommendations):

    # fit
    model.fit(data)
    # get input movie index
    print('You have input movie:', fav_movie)
    idx = _fuzzy_matching(hashmap, fav_movie)
    # inference
    print('Recommendation system start to make inference')
    print('.....\n')
    t0 = time.time()
    distances, indices = model.kneighbors(
        data[idx],
        n_neighbors=n_recommendations+1)
    # get list of raw idx of recommendations
    raw_recommends = sorted(
        list(
            zip(
                indices.squeeze().tolist(),
                distances.squeeze().tolist()
            )
        ),
        key=lambda x: x[1]
    )[:0:-1]
    print('It took my system {:.2f}s to make inference \n'.format(time.time() - t0))
    # return recommendation (movieId, distance)
    return raw_recommends

```

And finally we get similar movie recommendation to the input movie.

## 5.5 Common Terminology:

**1. Cosine Similarity:** Cosine similarity is a metric used to measure how similar the movies are irrespective of their ratings. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. The cosine similarity is advantageous because even if the two similar movies are far apart by the Euclidean distance (due to the rating of the movie), chances are they may still be oriented closer together. The smaller the angle, higher the cosine similarity.

It is calculated as,



$$\cos\theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|} = \frac{\sum_1^n a_i b_i}{\sqrt{\sum_1^n a_i^2} \sqrt{\sum_1^n b_i^2}}$$

where,  $\vec{a} \cdot \vec{b} = \sum_1^n a_i b_i = a_1 b_1 + a_2 b_2 + \dots + a_n b_n$  is the dot product of the two vectors.

**2. K-Nearest Neighbour:** The k-nearest neighbors (KNN) algorithm is simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems. KNN is a perfect go-to model and also a very good baseline for recommender system development. **KNN** is a **non-parametric, lazy** learning method. It uses a database in which the data points are separated into several clusters to make inference for new samples.

KNN does not make any assumptions on the underlying data distribution but it relies on **item feature similarity**. When KNN makes inference about a movie, KNN will calculate the “distance” between the target movie and every other movie in its database, then it ranks its distances and returns the top K nearest neighbour movies as the most similar movie recommendations.

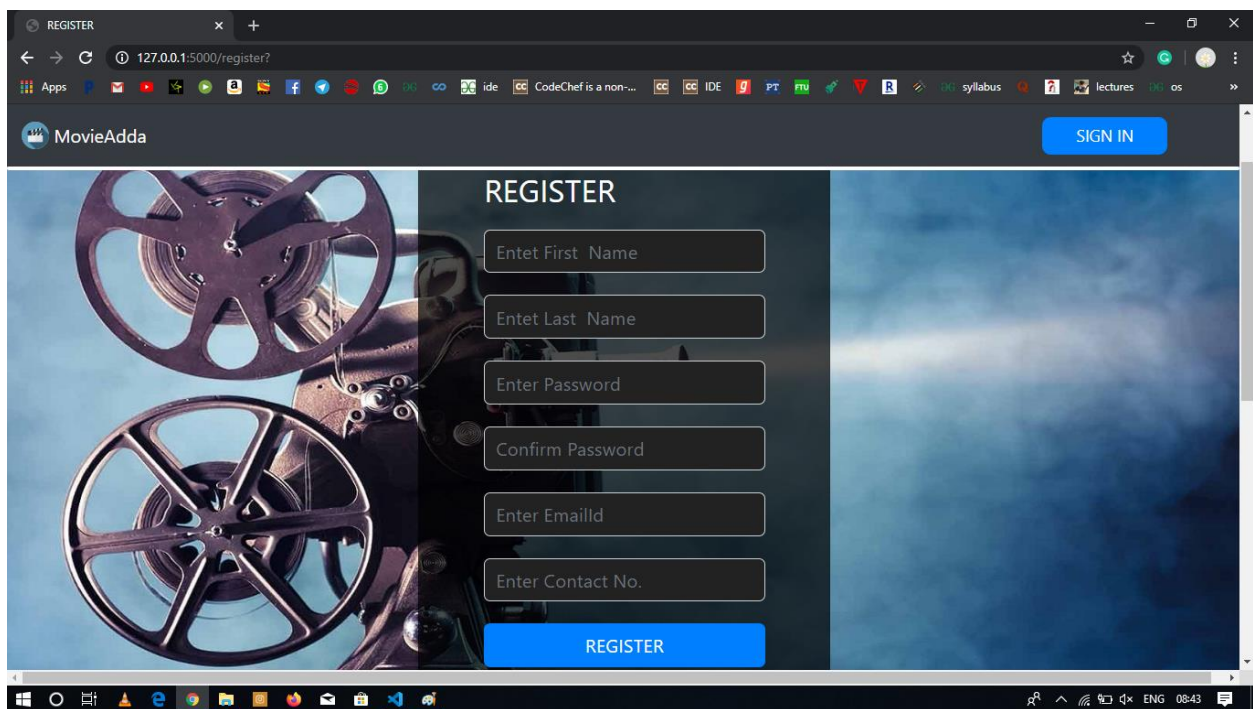
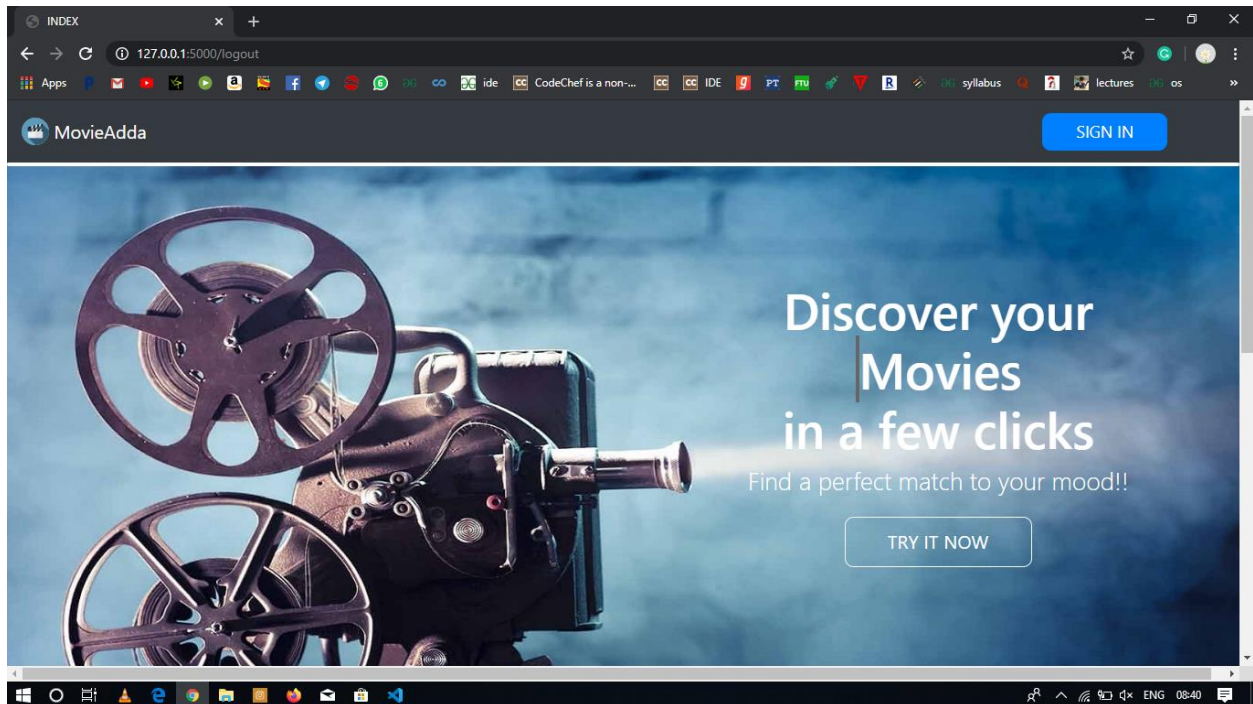
## 6. Testing

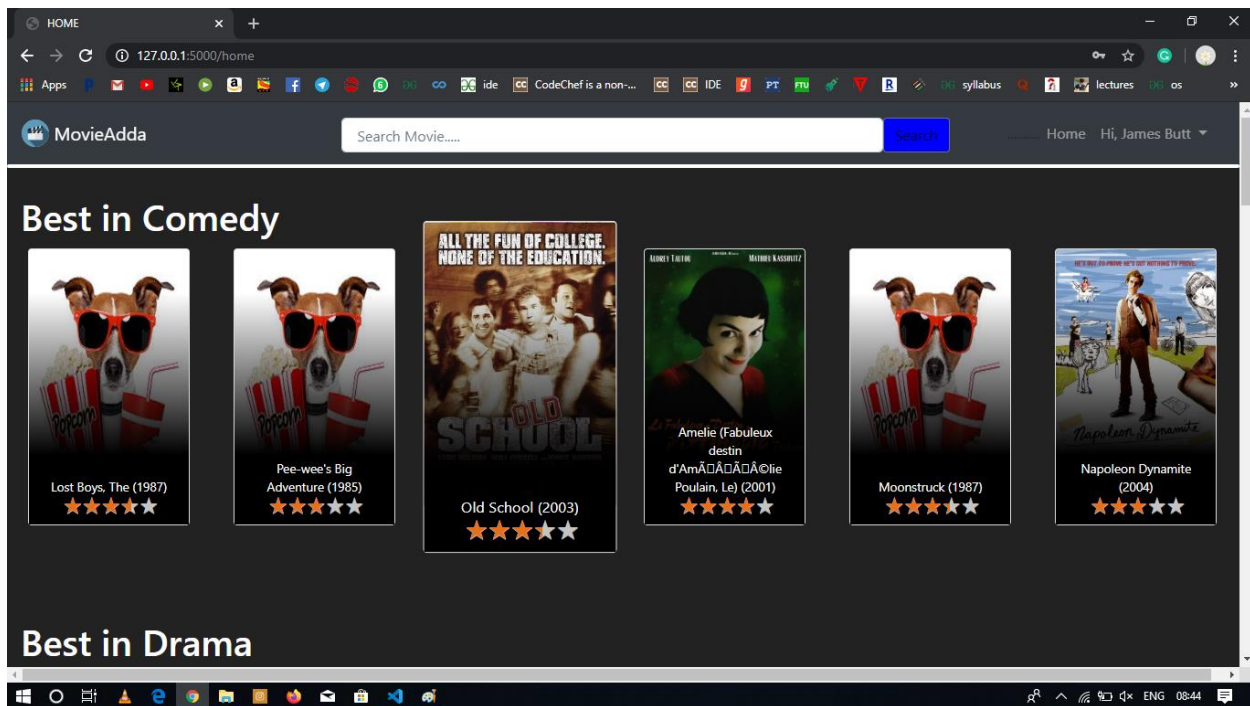
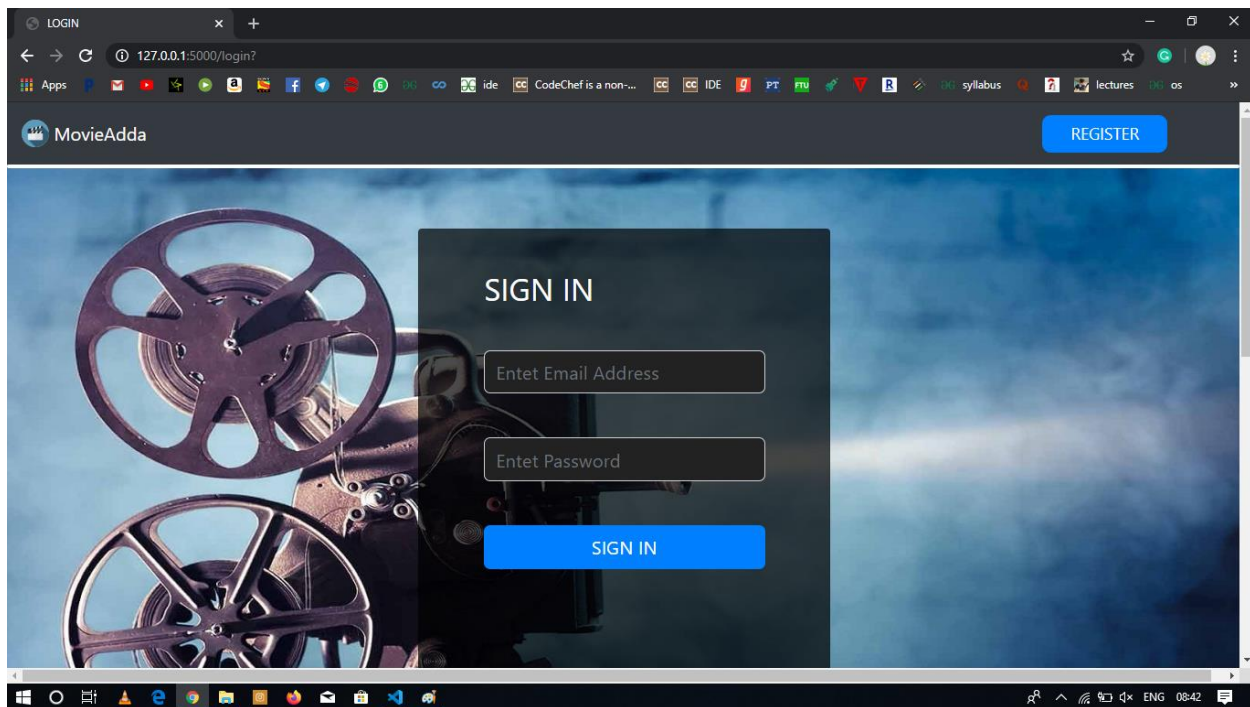
---

We have tested our system for following test cases:

- User can not access any functionality if user is not logged in into the system.
- End-User cannot access functionalities authorized by Admin.
- Admin cannot modify user`s data and his/her preferences in dataset.
- User can give rating to any particular movie only once.

## 7. screen shots



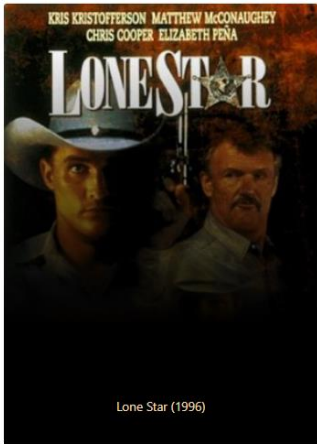


MOVIE INFORMATION

127.0.0.1:5000/movieInfo

MovieAdda Search Movie.....

Home Hi, James Butt



Lone Star (1996)

Genre : Drama|Mystery|Western

Average Rating : ★★★★★

Total Rating : 19

Release Date : 21 June 1996 (USA)

Run Time : 2h 15min

Cast : John Sayles

Director : John Sayles

Writer : John Sayles

Summary : When the skeleton of his murdered predecessor is found, Sheriff Sam Deeds unearths many other long-buried secrets in his Texas border town.

Give Rating : ★★★★★


Submit

PROFILE

127.0.0.1:5000/profile

MovieAdda Search Movie.....

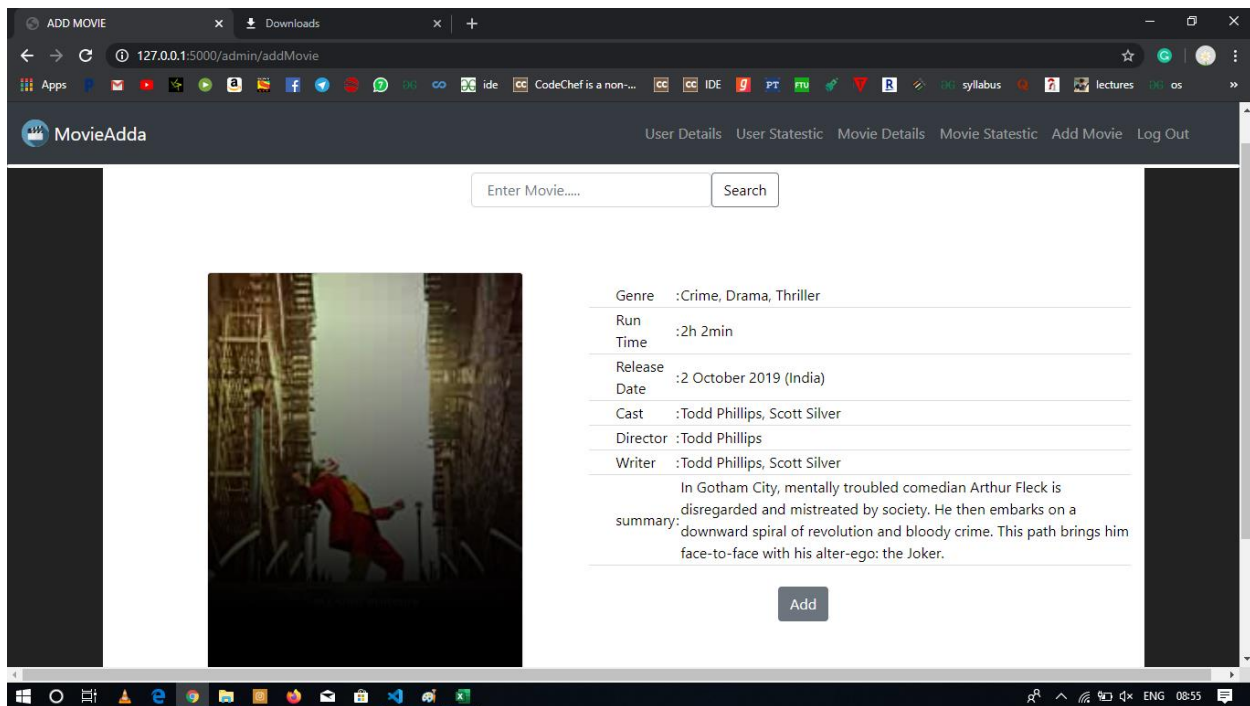
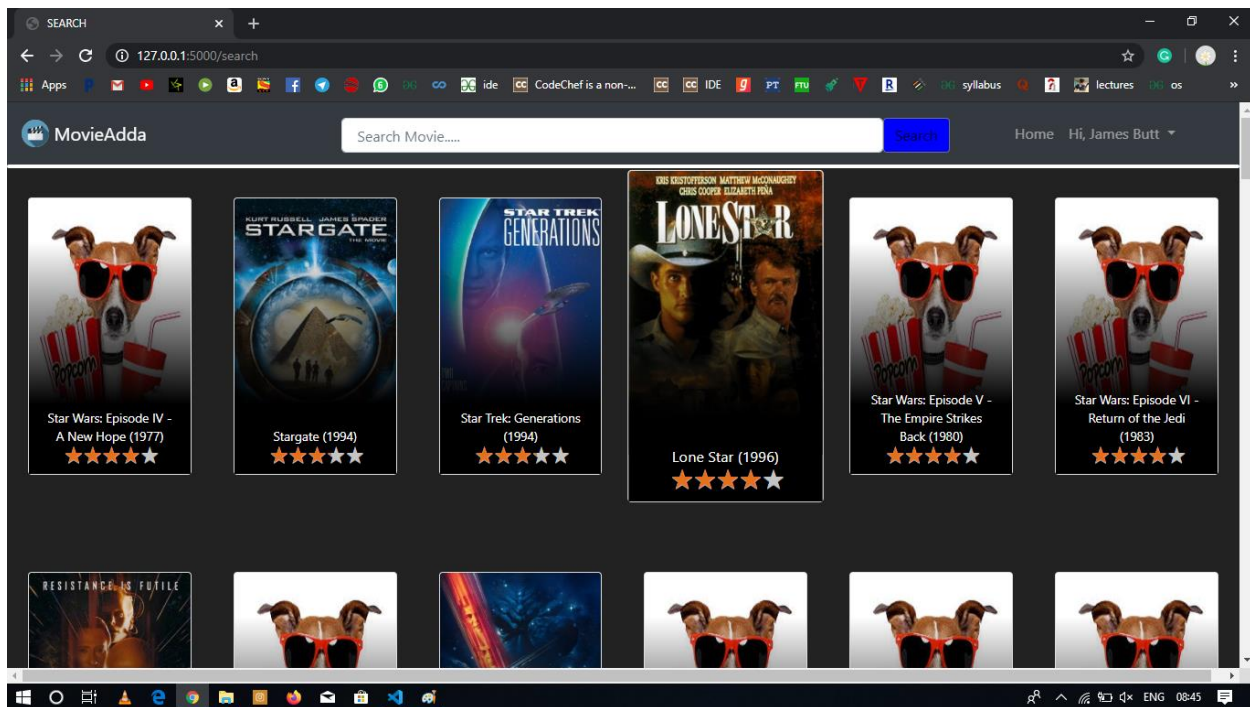
Home Hi, James Butt



James Butt

Favourite Genre	Rated Movies	Average Rating
Film-Noir	2	4.75
Animation	31	4.63
Musical	23	4.61
Children	43	4.53





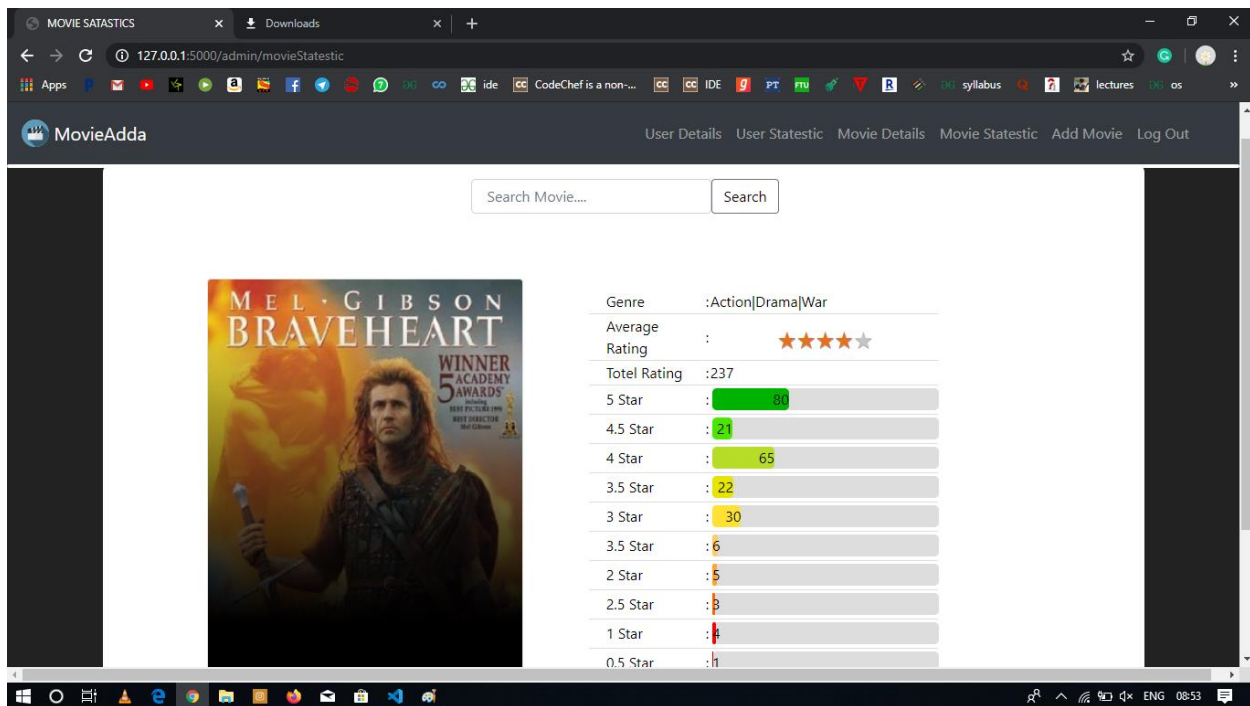
MOVIE DETAILS

127.0.0.1:5000/admin/movieDetails

MovieAdda

User Details User Statistic Movie Details Movie Statistic Add Movie Log Out

Movie Id	Title	Rated Count
1	Toy Story (1995)	215
2	Jumanji (1995)	110
3	Grumpier Old Men (1995)	52
4	Waiting to Exhale (1995)	7
5	Father of the Bride Part II (1995)	49
6	Heat (1995)	102
7	Sahrina (1995)	54



USER DETAILS

127.0.0.1:5000/admin/userDetails

MovieAdda

User Details User Statistic Movie Details Movie Statistic Add Movie Log Out

user Id	Email Address	Rated Moive Count
1	jbutt@gmail.com	238
2	josephine_darakjy@darakjy.org	30
3	art@venere.org	39
4	lpaprocki@hotmail.com	216
5	donette.foller@cox.net	44
6	simona@morasca.com	314
7	mitsue_tollner@yahoo.com	152

USER STATISTICS

127.0.0.1:5000/admin/userStatistic

MovieAdda

User Details User Statistic Movie Details Movie Statistic Add Movie Log Out

Genre	Count
Action	94
Adventure	87
Animation	31
Children	44
Comedy	92
Crime	50
Documentary	0



## 8. conclusion

---

We have successfully implemented all the functionalities described in SRS document. We have successfully implemented recommendation module, search module, add movie module, statistics modules of users and movies.

## 9. Limitations and Future Enhancements

---

Our system is not highly secure. We will be in future updating and extending for the same. We will also like to have a machine learning model with higher efficiency. We would also like to give more control to Admin user over other end-users.

## 10. Reference / Bibliography

---

Following links and websites were referred during the development of this project.

<https://www.google.co.in>

<http://www.wikipedia.org>

<http://www.getbootstrap.com>

<http://www.w3schools.com>

<http://www.tutorialspoint.com>

<http://www.stackoverflow.com>