MovieSpy: Movie Recommendation System

A Project Report

Submitted for the partial fulfillment for the award of the degree of

Bachelor of Technology (Computer Science)

GROUP ID- CS17

SUBMITTED BY:
OJASVI BANWAT (1812805)
PALAK GOEL (1812808)
PRACHI JAIN (1812818)

PROJECT MENTOR:
MRS. KARUNA SHARMA

PROJECT COORDINATORS:
DR. NEELAM SHARMA
DEEPAK KUMAR

Department of Computer Science and Mathematics

Banasthali Vidyapith

Banasthali - 304022

Session: 2019-20

ABSTRACT

A recommendation system is a type of information filtering system which attempts to predict the preferences of a user and make suggestions based on these preferences. There are a wide variety of applications for recommendation systems. These have become increasingly popular over the last few years and are now utilized in most online platforms that we use. The content of such platforms varies from movies, music, books, and videos, to friends and stories on social media platforms, to products on e-commerce websites, to people on professional and dating websites, to search results returned on Google. Often, these systems can collect information about user choices, and can use this information to improve their suggestions in the future.

“**Movie Spy”** is a web application which will learn about its user and give them a set of movies that they will most likely enjoy. The system will make extensive use of user data to come up with reasonable track prediction about user preferences and although there are two main paradigms in this: Content-based and Collaborative filtering, Movie Spy will be using a content-based filtering approach to recommend movies to the user. The Movie Recommender System will be a Web Service and the new users will be directly exposed to the suggestions made for them.

In our MovieSpy Model, we have added two special features of adding a movie and searching a movie. Users can add movie in our database if movie does not already exist in database. Users can also search for movies released in a particular year and by a particular director if they want to see a list of it.

ACKNOWLEDGEMENT

“It is not possible to prepare a project report without the assistance and encouragement of other people. This one is certainly no exception.”

On the very onset of this report, we would like to extend my sincere and heartfelt obligation towards all the people who have helped me in this endeavor. Without their active guidance, help, cooperation and encouragement, we would not have been able to complete this project.

We are indebted to our mentor, **Mrs. Karuna Sharma** for consistent guidance and encouragement to accomplish this feat.

We are extremely thankful to our coordinators, **Dr. Neelam Sharma** and **Dr. Deepak Kumar** for their constant guidance and support who arranged and managed all the details about the sessions regarding the project and made us comfortable in reaching them out for any queries or concerns.

We also extend our gratitude to Banasthali Vidyapith for giving us this opportunity and for including the project in our curriculum, so that we were able to convert our concepts and skills into solution for real world problems.

Thanks for all your encouragement!

TABLE OF CONTENT

OBJECTIVE

In today's digital world where there is endless variety of content to be consumed, finding the content of one's liking has become irksome task. Everyone likes to watch movies for entertainment purposes but nowadays no one wishes to invest their time in some kind of movie that is not relevant to their taste. With the hectic life schedule, no one is likely to spend time searching for movies they may or may not prefer.

MovieSpy is a web application that provides the user with a set of movies that they might want to watch. It is designed not only to search for movies but also to discover them through our recommendation process. The objective of MovieSpy is to provide most similar movie recommendations to its users. The overall goal is to ease the movie discovery process.

Software Requirement Specification

1. Introduction

1.1. Scope

The target software product is a Movie Recommendation system. "**Movie Spy"** is a web application which will learn about its user and give them a set of movies that they will most likely enjoy. The system will make an extensive use of user data to come up with reasonable track prediction about user preferences and there are two main paradigms in this: Content based and Collaborative filtering. The Movie Recommender System will be a Web Service and the new users will be directly exposed to the suggestions made for them using Content based filtering.

1.2. Intended Audience

The intended audience of the document are project managers, developers, testers and end users.

- **Project Manager** reviews the document and determines whether the planned system fulfils the requirements. They notify the developer to fill up missing parts.
- **Developers** provide consistency by using the documentation.
- **Testers** use the document for verification and validation.
- **End users** use this document to learn about the scope of the system and its capabilities.

1.3. Definitions, Acronyms and Abbreviations

- RAD :- Rapid Application Development
- PHP :- PHP : HyperText Processor
- HTML :- HyperText Markup Language
- OS :- Operating System
- DFD- Data Flow Diagram
- SQL- Structured Query Language
- E-R Diagram- Entity Relationship Diagram
- Admin- Administrator

1.4. Overview

This Project Report is further organized as follows:

- **Section 2** (under Software Requirements Specifications) gives an overall description of the project. The product perspective, major functions and constraints are covered here. It determines the level of proficiency which is expected from user, some general constraints while making this software and making some assumptions and dependencies.

- **Section 3** (under Software Requirements Specifications) gives details about the external interface required, i.e., hardware interface, software interface, user interface etc.
- **Section 4** (under Software Requirements Specifications) gives specific requirements which the project software is expected to deliver. Functional requirements, performance requirements and some design constraints are also given. The use cases are also depicted in this section.

- **Section 1 (**under Software Design Specifications) depicts the System overview.
- **Section 2** (under Software Design Specifications) shows the System architecture and the decomposition design which further explains the data flow in our model.
- **Section 3** (under Software Design Specifications) gives all the data design i.e., E-R diagram and database description, all the tables used in our system.

- **Coding Section** consists of all the code of our model, front-end , back-end etc.

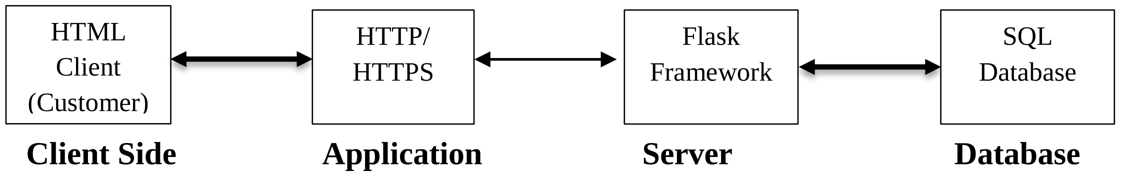
- **Testing Section** tells about different testing methods used to check the working and efficiency of this model.

- **User Interface Section** has two parts, module description and screen images. Module description tells what the inputs on different pages are and what the outputs respectively are.

2. General Description

2.1. Product Perspective

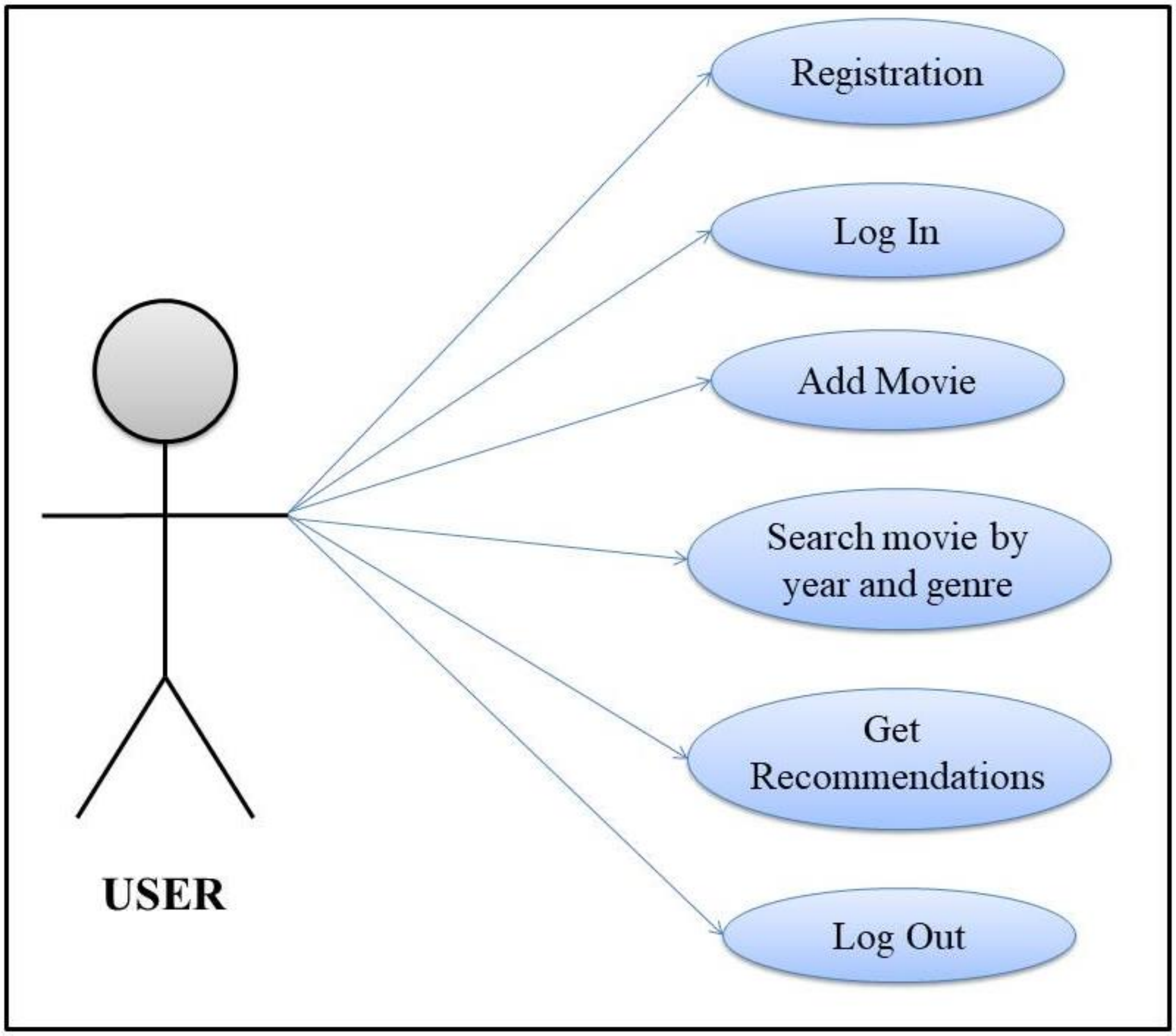
Recommendation system is a software tool that provides suggestions for items to a user. This web application provides user with a choice of set of movies which they might want to watch. The system here asks user to make account then according to their likes and dislikes recommend a set of movies to them. The system uses Pearson's correlation to recommend movies. Recommendation will be done based on 2 paradigms: Content based and collaborative filtering. Content based filtering uses item features to recommend other items similar to what the user likes, based on their previous actions or explicit search. Collaborative filtering uses *similarities between users and items simultaneously* to provide recommendations. The focus is to provide an easy to navigate application which allows user to navigate effortlessly.

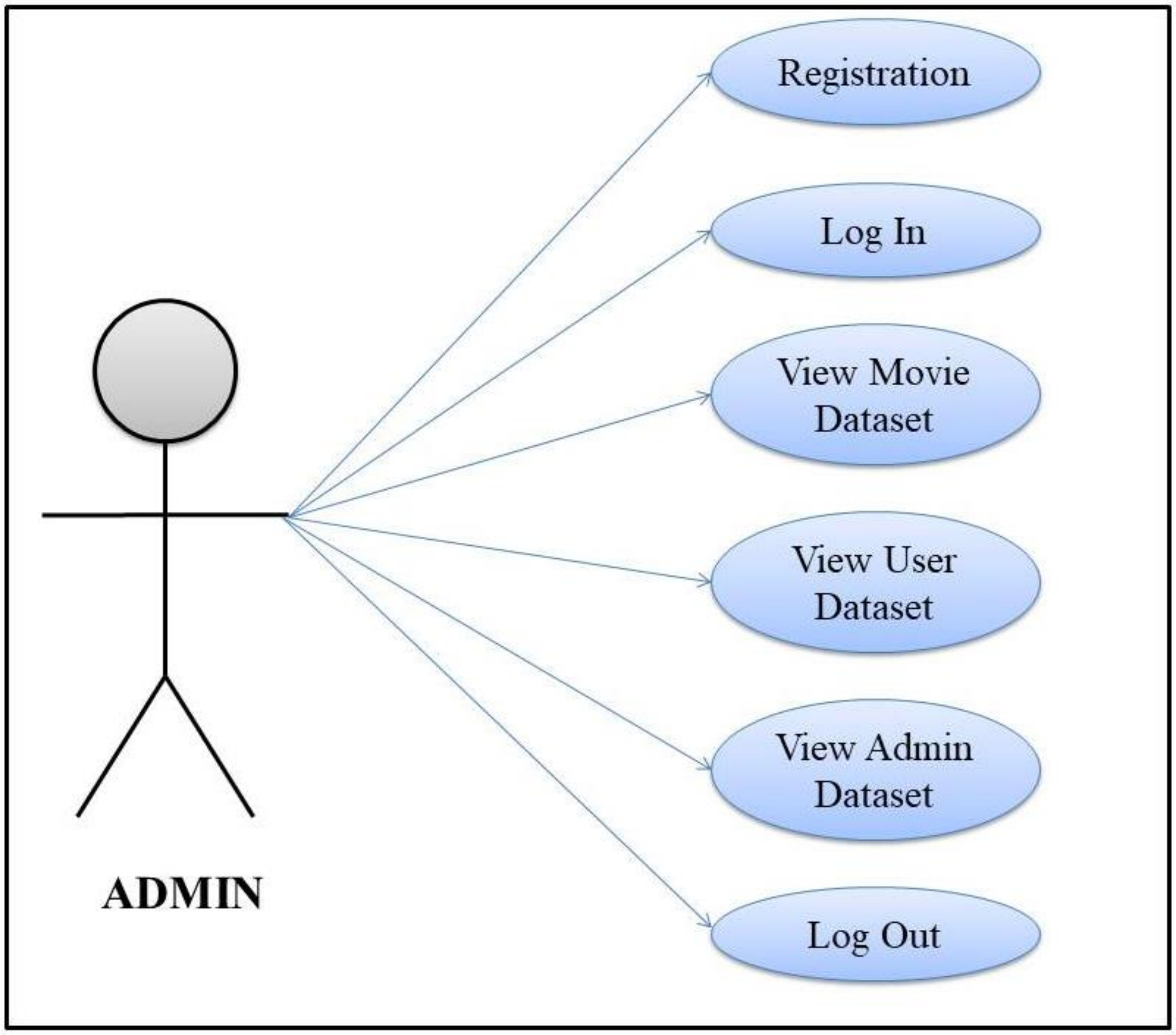


2.2. Product Functions

- It is online web supplication that recommends movies to user as per there likes.
- It is a fully automated and interactive System.
- **Get User Profile and Statistics:** This function asks user to provide their details and profile information which will be used by system later to work efficiently and provide recommendations.
- **Search Movie:** System provides us with search option through which user can search movies by year, genre etc.
- **Get Recommendations:** It uses algorithms to provide user a set.
- **Add Movie :** System gives users a option to add movie to the database if it is not already there.

2.3. Use Case Diagrams





2.4. Constraints

- Good Internet speed is critical for smooth functioning of web application.
- The password and a valid user id are required for logging in every time to ensure security.
- Admin will have special access to certain functionalities of the system.

3. External Interface Requirements

3.1. Hardware Interface

Developer Side:

- **RAM:** 4 GB Minimum, 8 GB Recommended.
- **HDD:** 20 GB or more (Free Space excluding data size)
- **Processor:** 2.50 GHz core i5

Customer Side:

- A system with web browser is required. High Internet speed is recommended.
- **RAM:** 4 GB Minimum
- **HDD:** 5 GB or more (Free Space excluding data size)
- **Processor:** 2.0 GHz core i5

3.2. Software Interface

Developer Side:

- **OS:** Any GUI Based Operating System
- **Front End:** HTML,CSS
- **Back End:** SQL Database
- **Framework:** Flask
- **Query Language:** SQL

Customer Side:

- **OS:** Any GUI Based Operating System
- **Browser:** Any Web Browser

3.3. Communication Interface

Customer on internet will be using HTTP/ HTTPS Protocol.

3.4. User Interface

The users of **“Movie Spy”** should be computer literate.
Users of the system can be categorized as follows:

- **User:** These are the people who will log in/ register, search movies on the basis of genre and year and get movies recommended as per their genre taste.

❖ **Admin:** These are the people who will be there to manage smooth functioning of app system. They will add and delete movies from dataset. Thus, admin will be able to manage and view movies dataset.

4. Specific Requirements

4.1. Functional Requirements

We will define functional requirements by following tables:

<u>REGISTER:</u>

Use Case No	01
Use Case Name	Register
Actors	Users, Admin
Description	This module helps the new user or admin to register in the system.
Input	Email-ID, Password, Security Question and Answer (Confirmation key for Admin only)
Normal Course Events	1. User enters the details as required. 2. User clicks on the 'Register' Button. 3. System connects to database thus saving user details for later use.
Alternative Courses	1. Same email Id has already been registered. 2. Continue with starting in the normal course events.
	1. Error message might appear during database operations. 2. System shows error message.
Output	New Account will be created.

Log In:

Use Case No	02
Use Case Name	Log In
Actors	Users, Admin
Description	This module helps the new user or admin to log in in the system.
Input	Email-ID and Password
Pre – Conditions	The user logging in must be a registered user of the system.
Normal Course Events	1. User enters their Email id. 2. User enters their password. 3. User clicks on ‘Log In’ Button. 4. System connects to database. 5. Homepage displayed.
Alternative Courses	1. User enters incorrect email-id and/or password. 2. Continue with starting in the normal course events.
	1. Error message might appear during database operations. 2. System shows error message.
Output	Homepage will be displayed.

Add Movie:

Use Case No	03
Use Case Name	Add Movie
Actors	User
Description	This module helps the user to add a movie in the dataset of the system.
Input	Movie details (Title, Genre, Year of release, Director of movie, Actors of Movie).
Pre – Conditions	User must be logged in.
Normal Course Events	1. User click ‘Add Movie’ Button. 2. New Movie details entered. 3. System connects to database. 4. New movie is added to database.
Alternative Course Events	1. Error message might appear during database operations. 2. System shows error message.
Output	New movie will be added.

Search by Year & Director:

Use Case No	05
Use Case Name	Search Movie
Actors	User
Description	This module helps the user to search a movie on the basis of its year and director.
Input	Year of release and Director Name
Pre – Conditions	User must be logged in.
Normal Course Events	1. User enter year of release and director of movie. 2. User click on ‘Search’ Button. 3. System connects to database. 4. Set of movies which met conditions are displayed.
Alternative Course Events	1. User enter incorrect year and/or genre. 2. Continue with starting in the normal course events.
	1. Error message might appear during database operations. 2, System shows error message.
Output	Set of movies displayed.

<u>Get Recommendations:</u>

Use Case No	06
Use Case Name	Get Recommendations
Actors	User
Description	This module recommends movies to user as per their genre taste.
Input	Movie name
Pre – Conditions	User must be logged in.
Normal Course Events	1. User enter movie name. 2. User click on ‘Get Recommendations’ Button. 3. System connects to database. 4. Set of movies which satisfy algorithm are displayed.
Alternative Course Events	1. User enter incorrect movie name. 2. Continue with starting in the normal course events.
	1. Error message might appear during database operations. 2. System shows error message.
Output	Set of movies displayed.

4.2. Non-Functional Requirements

4.2.1. Availability

The availability of this software is up to the Internet connection of the user. The system shall be attainable at all time. User should have an account to enter the system. If user doesn't have an account, then user can only see the information which are displayed on home page.

4.2.2. Security

After entering the password and user id the user is allowed to access system. The details of user must be kept safe and secure thus providing high level security for the database. The authorization system is as such if user enters wrong Id and/or password, user is not allowed to login the software.

4.2.3. Reliability

This System is reliable enough that it maintains the secure database of the record of the user. The unauthorized login is secured via password and id.

4.2.4. Portability

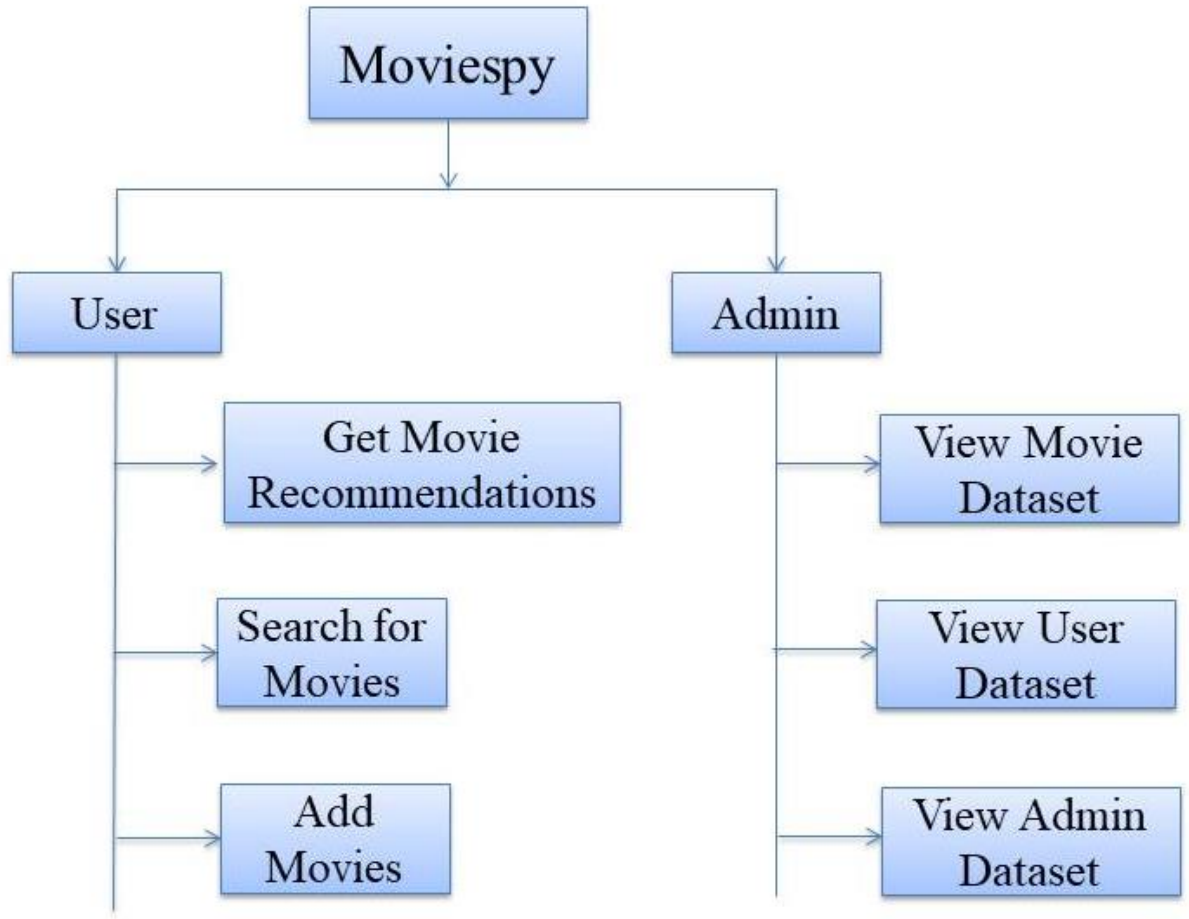
The System is developed using **Python** which provides a framework for developing applications. The code and supporting modules of the system will be well documented and easy to understand.

4.2.5. Maintainability

This document will follow the modular structure so it will be easy to maintain.

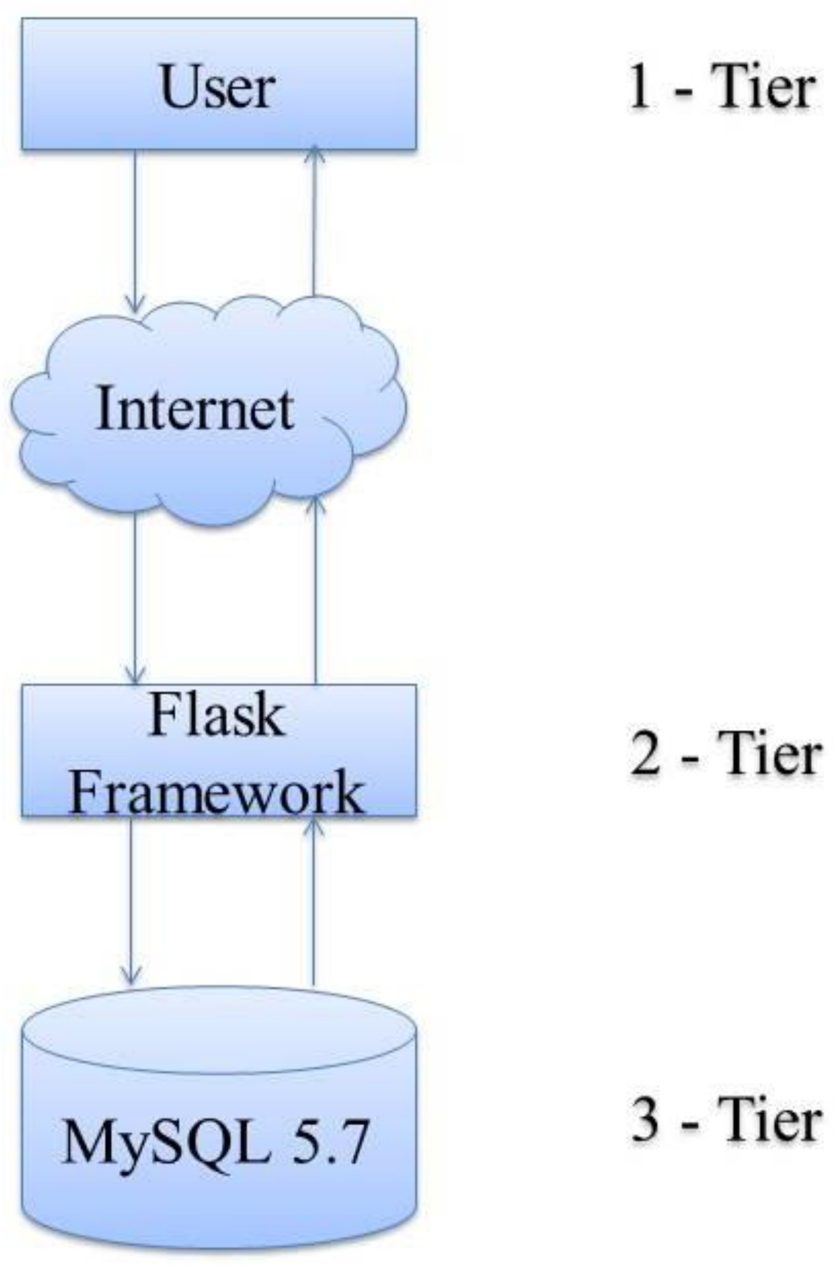
Software Design Specification

1. System Overview



2. System Architecture

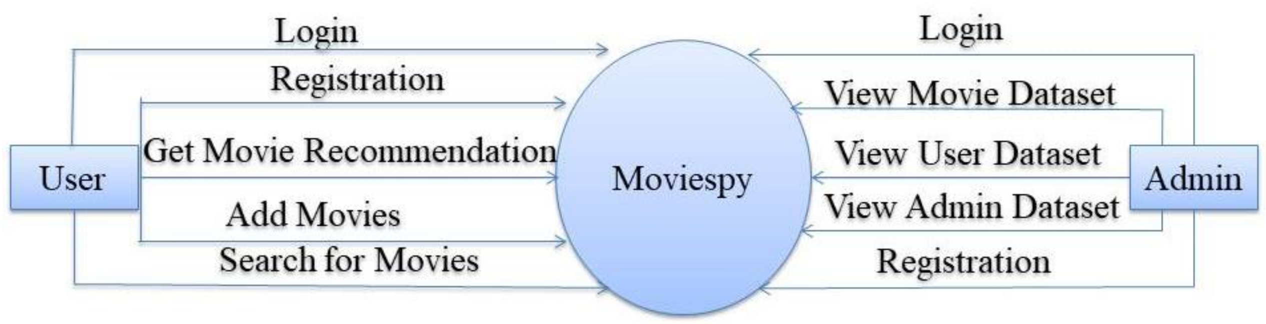
2.1. Architectural Design



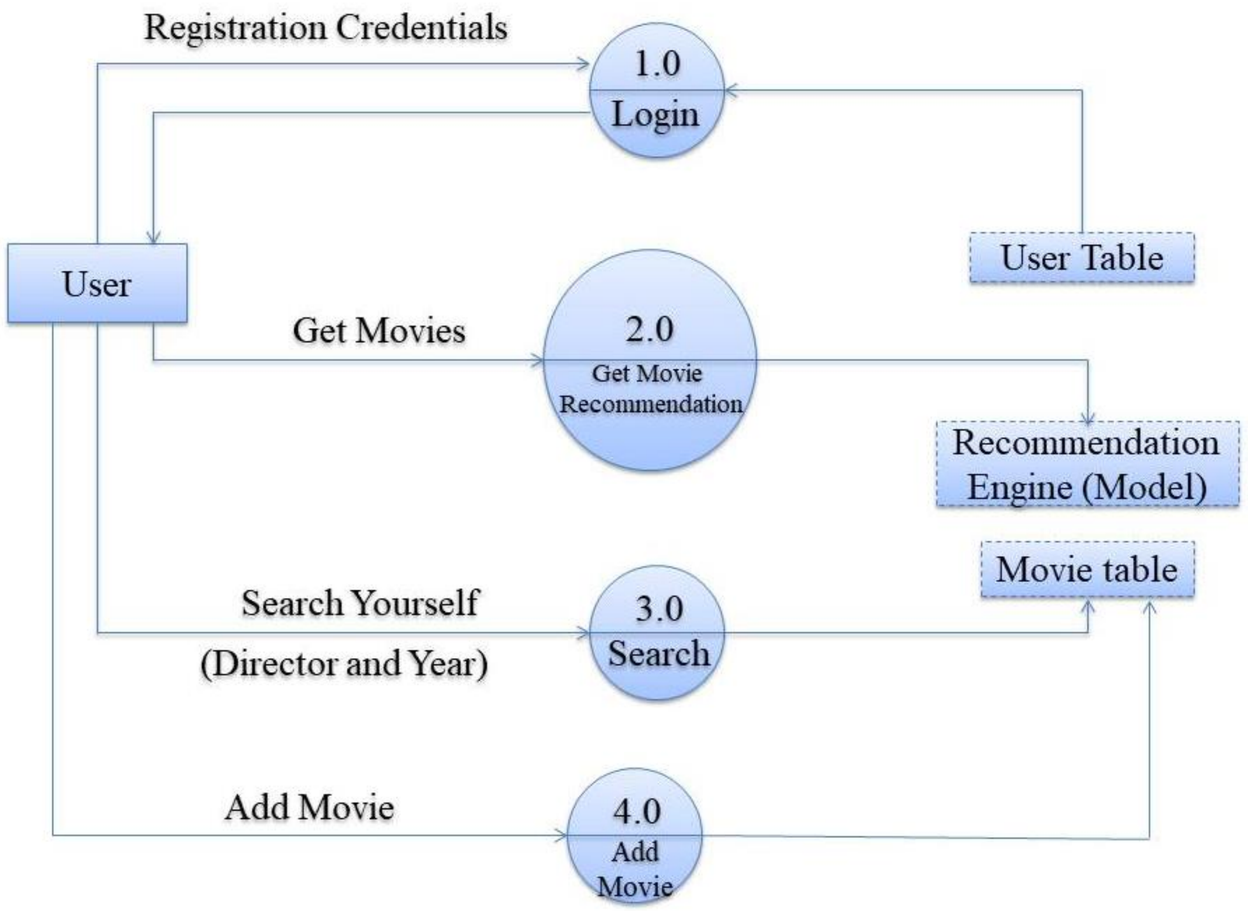
2.2. Decomposition Design

2.2.1. Data Flow Diagram

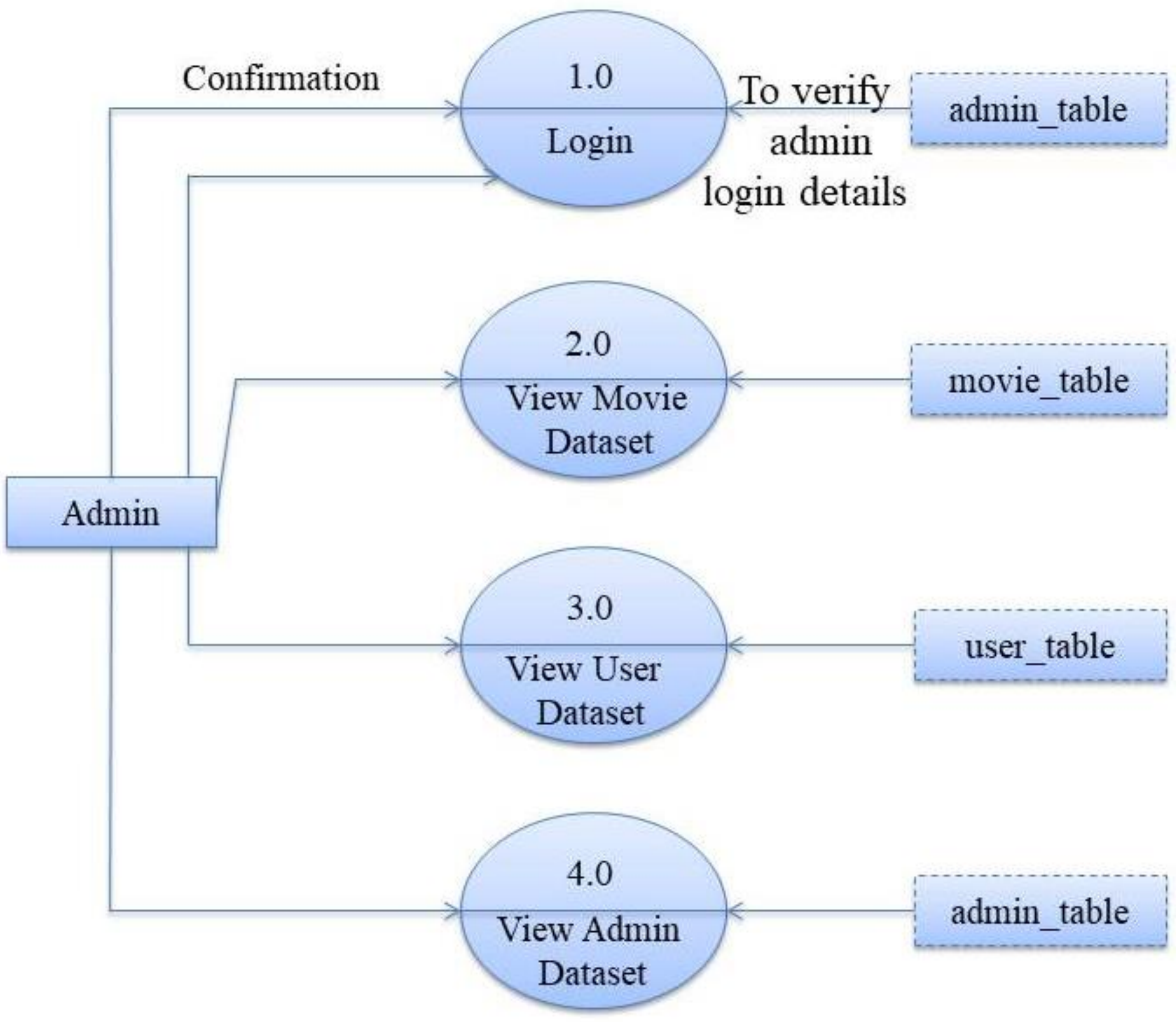
0 – Level DFD



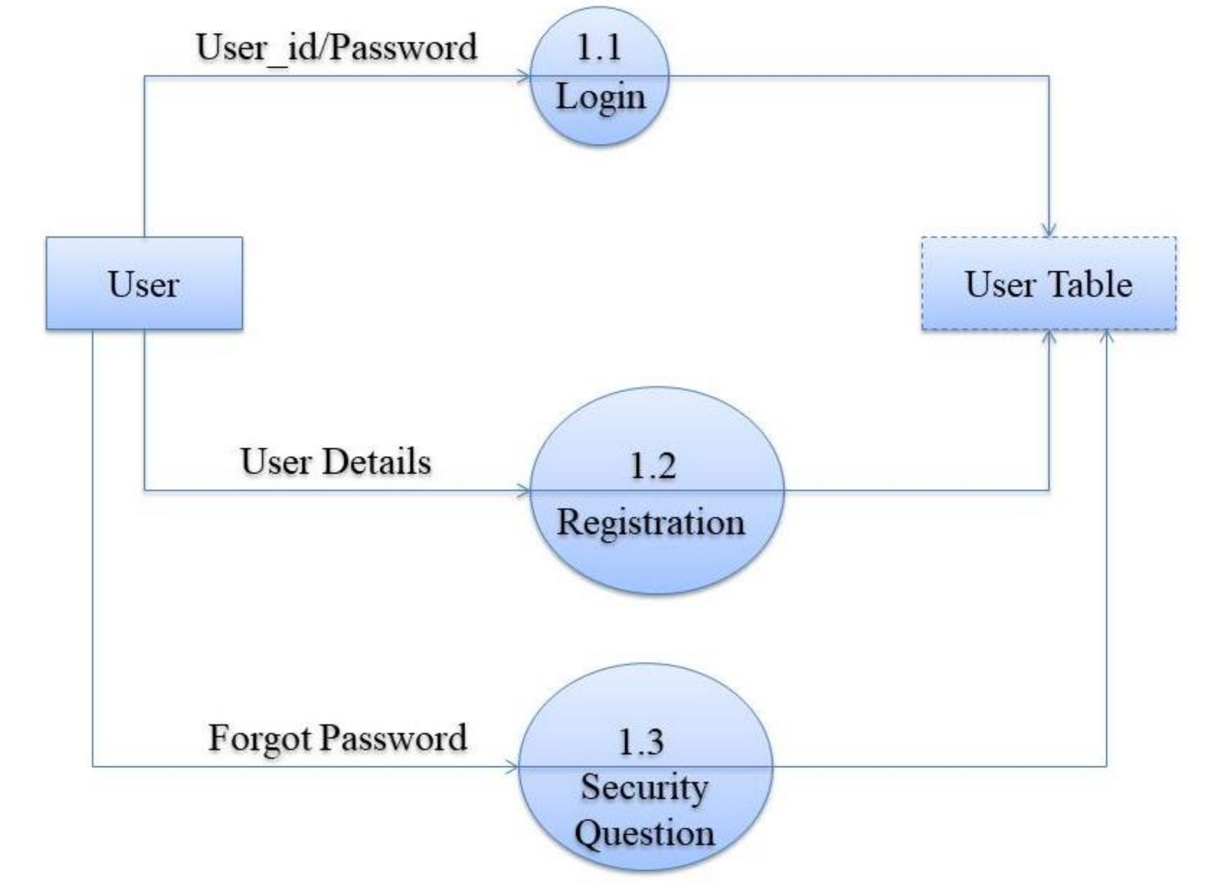
1 – Level DFD of User

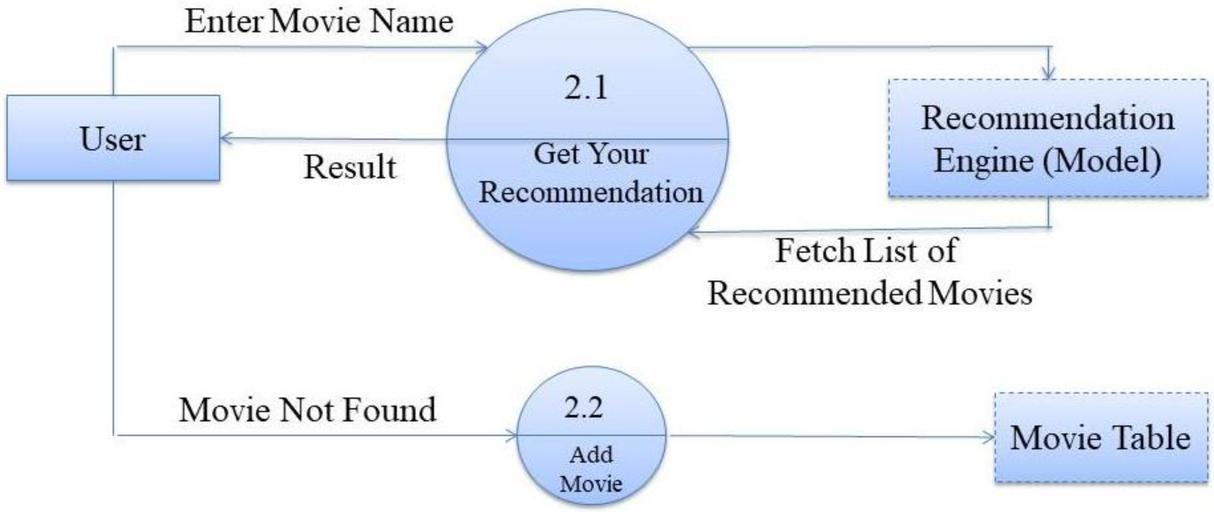


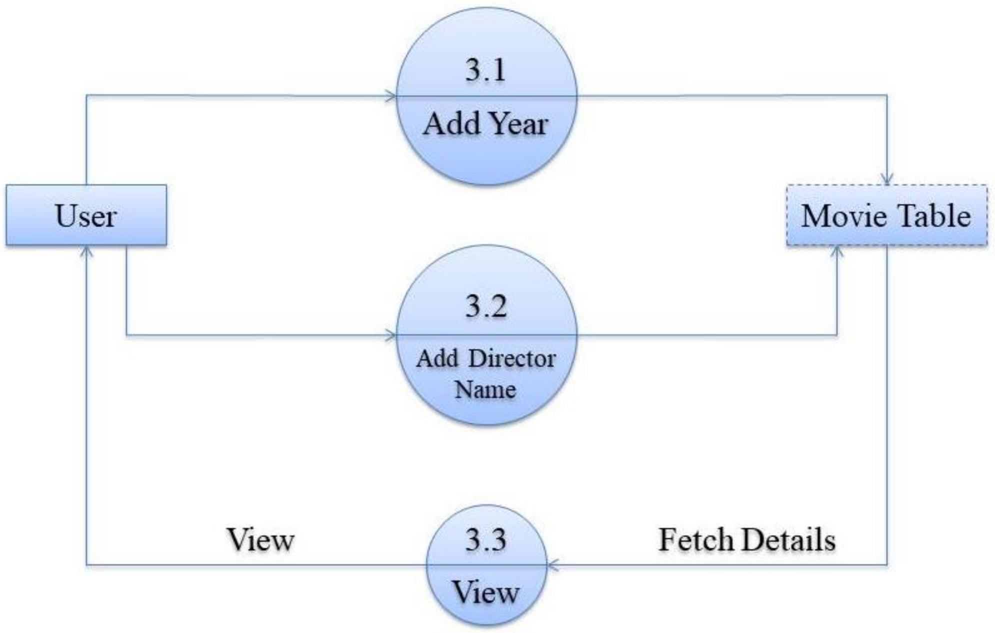
1 – Level DFD of Admin

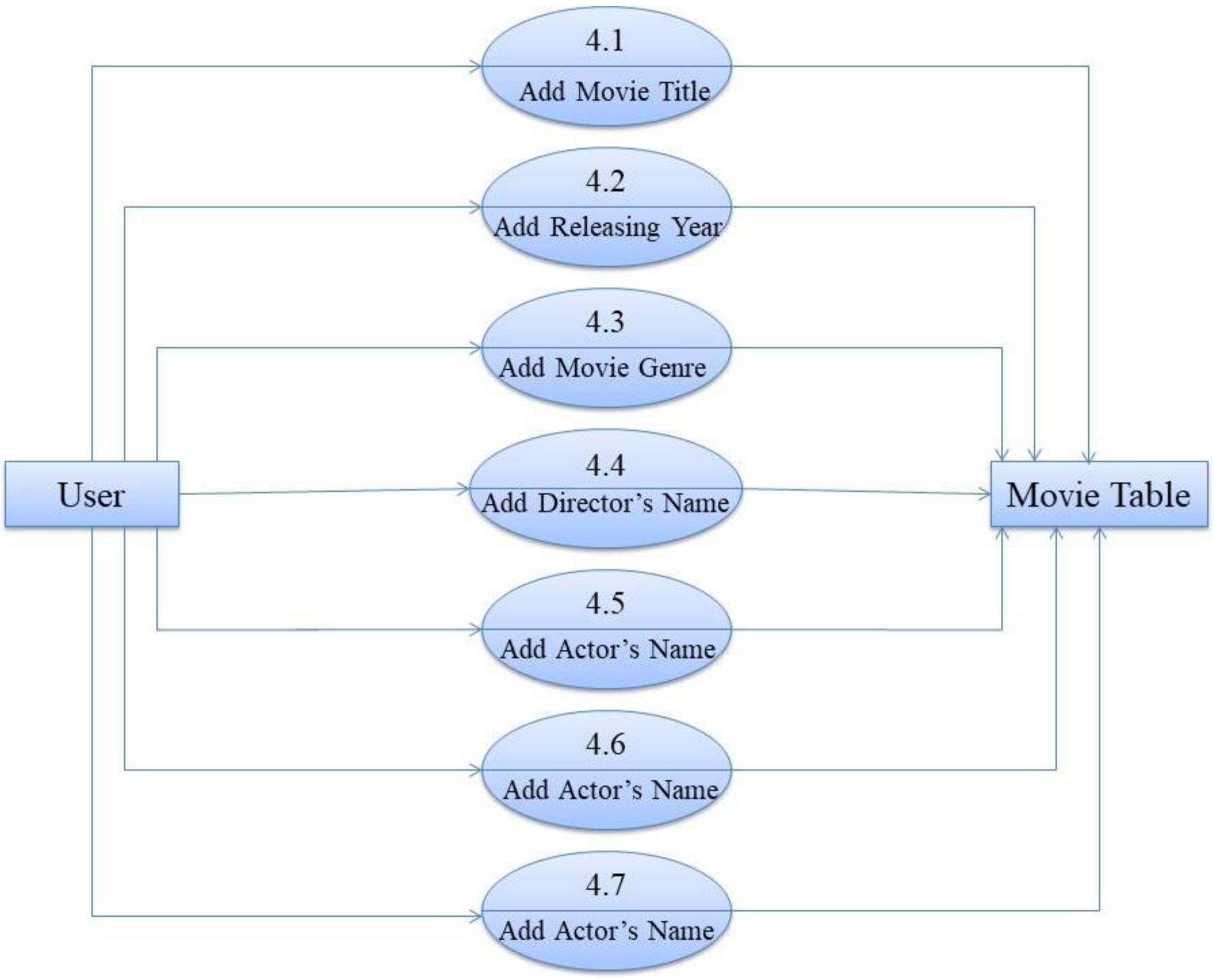


2 – Level DFD of User

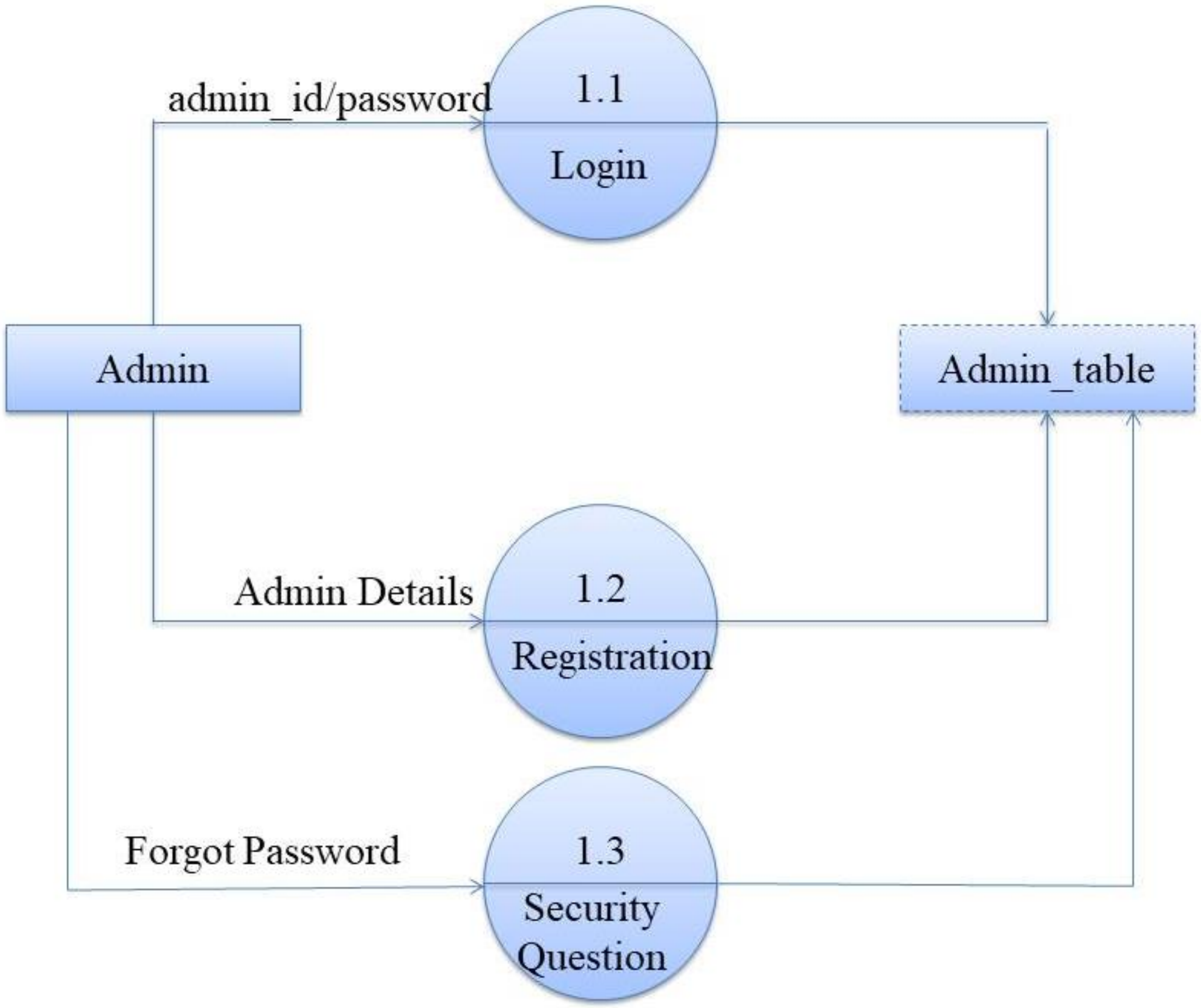





3.1
Add Year
User
Movie Table
3.2
Add Director Name
View
3.3
View
Fetch Details

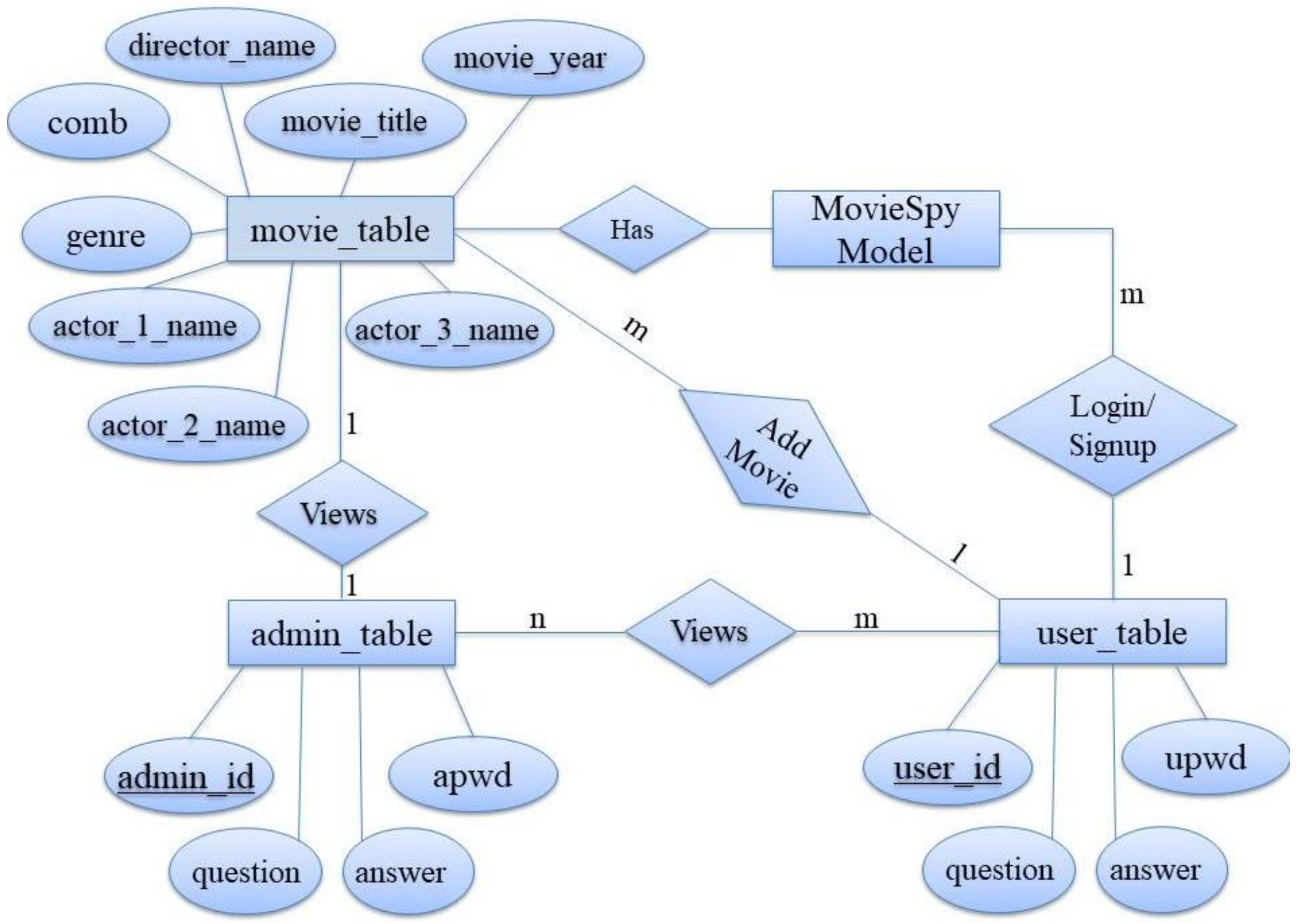

4.1
Add Movie Title
4.2
Add Releasing Year
4.3
Add Movie Genre
User
4.4
Add Director's Name
Movie Table
4.5
Add Actor's Name
4.6
Add Actor's Name
4.7
Add Actor's Name

2 – Level DFD of Admin

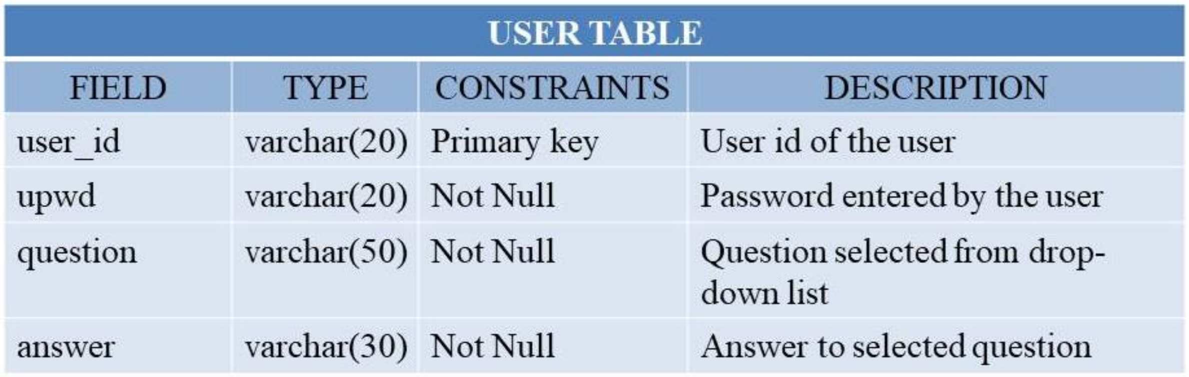


3. Data Design

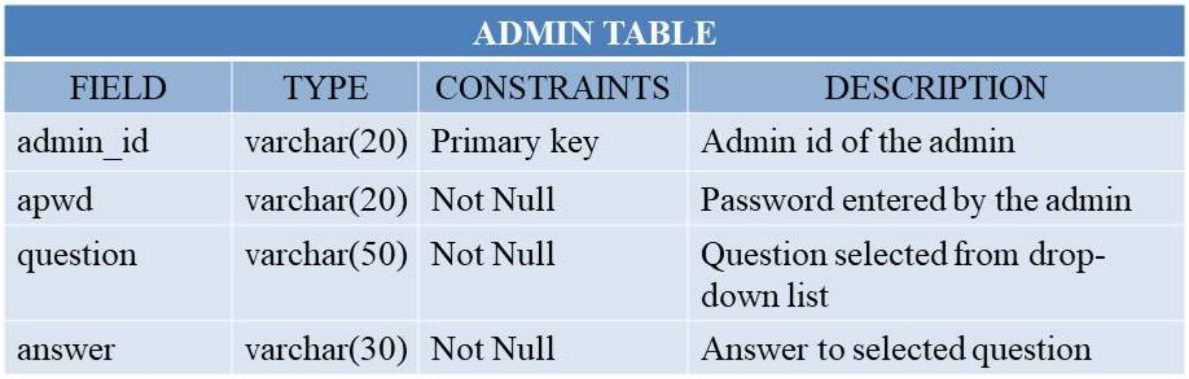
3.1. E-R Diagram



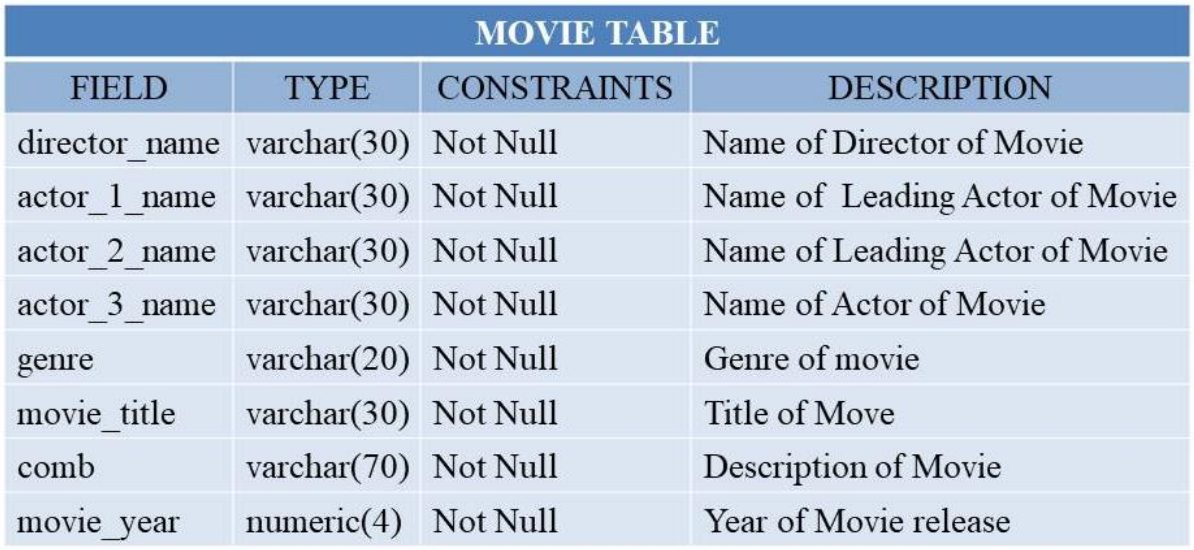
3.2. Database Description



USER TABLE			
FIELD	TYPE	CONSTRAINTS	DESCRIPTION
user_id	varchar(20)	Primary key	User id of the user
upwd	varchar(20)	Not Null	Password entered by the user
question	varchar(50)	Not Null	Question selected from drop-down list
answer	varchar(30)	Not Null	Answer to selected question



ADMIN TABLE			
FIELD	TYPE	CONSTRAINTS	DESCRIPTION
admin_id	varchar(20)	Primary key	Admin id of the admin
apwd	varchar(20)	Not Null	Password entered by the admin
question	varchar(50)	Not Null	Question selected from drop-down list
answer	varchar(30)	Not Null	Answer to selected question



MOVIE TABLE			
FIELD	TYPE	CONSTRAINTS	DESCRIPTION
director_name	varchar(30)	Not Null	Name of Director of Movie
actor_1_name	varchar(30)	Not Null	Name of Leading Actor of Movie
actor_2_name	varchar(30)	Not Null	Name of Leading Actor of Movie
actor_3_name	varchar(30)	Not Null	Name of Actor of Movie
genre	varchar(20)	Not Null	Genre of movie
movie_title	varchar(30)	Not Null	Title of Move
comb	varchar(70)	Not Null	Description of Movie
movie_year	numeric(4)	Not Null	Year of Movie release

Coding

main.py

```
import MySQLdb

from flask import *

from flask_mysqldb import MySQL

import pandas as pd

import numpy as np

from sklearn.feature_extraction.text import CountVectorizer

from sklearn.metrics.pairwise import cosine_similarity

from csv import writer


# from model import *

app = Flask(__name__, template_folder="template", static_url_path='/static')


app.secret_key = 'abcdef ghijkl mnopqr stuvw xyz'

app.config['MYSQL_HOST'] = 'localhost'

app.config['MYSQL_USER'] = 'root'

app.config['MYSQL_PASSWORD'] = 'Value@2015'

app.config['MYSQL_DB'] = 'MovieSpy'


mysql = MySQL(app)


test = pd.read_csv('FINAL_MOVIE_TABLE.csv')
```

```
def create_sim():
    test = pd.read_csv('FINAL_MOVIE_TABLE.csv')
    cv = CountVectorizer()
    count_matrix = cv.fit_transform(test['comb'])
    sim = cosine_similarity(count_matrix)
    return test, sim



def rcmd(m):
    m = m.lower()
    try:
        test.head()
    except:
        test, sim = create_sim()
    if m not in test['movie_title'].unique():
        print('Sorry! This movie is not in our database. Please check the spelling or try with some other
movies')
        return render_template('userportal.html')
    else:
        # getting the index of the movie in the dataframe
        i = test.loc[test['movie_title'] == m].index[0]

        # fetching the row containing similarity scores of the movie
        # from similarity matrix and enumerate it
        lst = list(enumerate(sim[i]))
        # sorting this list in decreasing order based on the similarity score
        lst = sorted(lst, key=lambda x: x[1], reverse=True)


        # taking top 1- movie scores
        # not taking the first index since it is the same movie
        lst = lst[1:11]
        # making an empty list that will containg all 10 movie recommendations
        l = []
        for i in range(len(lst)):
            a = lst[i][0]
            l.append(test['movie_title'][a])
        return l



def searchHelper(year, name):
    name = name.lower()
    test1 = pd.read_csv('FINAL_MOVIE_TABLE.csv')
    t = test1.loc[(test1['year'] == year)]


    t1 = t.loc[(t['director_name'] == name)]


    l = t1['movie_title'].tolist()
    return l
```

```
def csvToSQL():

    data = pd.read_csv('FINAL_MOVIE_TABLE.csv')

    df = pd.DataFrame(data, columns=['year', 'director_name', 'actor_1_name', 'actor_2_name',
'actor_3_name', 'genres',

                                     'movie_title', 'comb'])

    curr = mysql.connection.cursor(MySQLdb.cursors.DictCursor)

    for row in df.itertuples():

        curr.execute("INSERT INTO movie VALUES(%s, %s, %s, %s, %s, %s, %s, %d)", (

            row.director_name, row.actor_1_name, row.actor_2_name, row.actor_3_name, row.genres,
row.movie_title,

            row.comb,

            int(row.year)))

        curr.commit()




@app.route('/', methods={'GET', 'POST'})

def home():

    return render_template('homepage.html')




@app.route('/userportal')

def userportal():

    if 'loggedin' in session:
        return render_template('userportal.html')
    else:
        return redirect(url_for('user'))



@app.route('/user', methods={'GET', 'POST'})
def user():
    msg = ''
    if request.method == 'POST' and 'user_id' in request.form and 'upwd' in request.form:
        uid = request.form['user_id']
        pwd = request.form['upwd']
        # Now, check if account exists in database
        curr = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
        curr.execute("SELECT * FROM user where user_id = %s and upwd = %s", (uid, pwd,))
        account = curr.fetchone()
        if account:
            session['loggedin'] = True
            session['id'] = account['user_id']
            session['password'] = account['upwd']
            return redirect(url_for('userportal'))
        else:
            msg = 'Incorrect User_id/Password..!!!'
            # flash(u'Incorrect User_id/Password..!!!', 'error')
    return render_template('userLogIn.html', msg=msg)
```

```
@app.route('/user', methods={'GET', 'POST'})

@app.route('/user/register', methods={'GET', 'POST'})

def register():

    msg = ''

    if request.method == "POST" and 'user_id' in request.form and 'upwd' in request.form and 'ucnfrmpwd' 
in request.form:

        id = request.form['user_id']

        pwd = request.form['upwd']

        cpwd = request.form['ucnfrmpwd']

        curr = mysql.connection.cursor(MySQLdb.cursors.DictCursor)

        curr.execute("SELECT * FROM user where user_id = %s and upwd = %s", (id, pwd,))

        account = curr.fetchone()

        if account:

            msg = "Account Already Exist"

        elif pwd != cpwd:

            msg = "Password doesn't match..!!!"

        else:

            curr = mysql.connection.cursor(MySQLdb.cursors.DictCursor)

            curr.execute("INSERT INTO USER(user_id,upwd) VALUES (%s,%s) ", (id, pwd,))

            mysql.connection.commit()

            return redirect(url_for('user'))

    return render_template('userRegister.html', msg=msg)
```

```
@app.route('/admin', methods={'GET', 'POST'})
def admin():
    msg = ''
    if request.method == 'POST' and 'admin_id' in request.form and 'apwd' in request.form:
        id = request.form['admin_id']
        pwd = request.form['apwd']
        curr = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
        curr.execute("SELECT * FROM admin WHERE admin_id = %s and apwd = %s", (id, pwd,))
        account = curr.fetchone()
        if account:
            session['loggedin'] = True
            session['id'] = account['admin_id']
            session['password'] = account['apwd']
            return redirect(url_for('userportal'))
        else:
            msg = 'Incorrect Admin Id/Password..!!!'
            flash(msg, category="error")
    return render_template('adminLogIn.html', msg=msg)



@app.route('/admin', methods={'GET', 'POST'})
@app.route('/admin/register', methods={'GET', 'POST'})
def adminRegister():
    msg = ''

    if request.method == "POST" and "admin_id" in request.form and "apwd" in request.form and
"acnfrmpwd" in request.form and "key" in request.form:

        id = request.form['admin_id']

        pwd = request.form['apwd']

        cpwd = request.form['acnfrmpwd']

        akey = request.form['key']

        curr = mysql.connection.cursor(MySQLdb.cursors.DictCursor)

        curr.execute("SELECT * FROM admin WHERE admin_id = %s and apwd = %s", (id, pwd,))

        account = curr.fetchone()

        if account:

            msg = "Account Already Exist..!!!"

        elif pwd != cpwd:

            msg = "Password doesn't match..!!!"

        elif akey != "12345abc":

            msg = "Incorrect Confirmation Key..!!!"

        else:

            curr = mysql.connection.cursor(MySQLdb.cursors.DictCursor)

            curr.execute("INSERT INTO admin(admin_id,apwd) VALUES (%s,%s) ", (id, pwd,))

            mysql.connection.commit()

            return redirect(url_for('admin'))

    return render_template('adminRegister.html', msg=msg)



@app.route('/<person>')
```

```
def access(person):

    if person == 'User':

        return redirect(url_for('user'))

    else:

        return redirect(url_for('admin'))




@app.route('/addmovie', methods={'GET', 'POST'})

def addMovie():

    if request.method == 'POST' and 'title' in request.form:

        movie = request.form['title']

        y = request.form['year']

        gen = request.form['genre']

        dirc = request.form['director']

        act1 = request.form['actor_1']

        act2 = request.form['actor_2']

        act3 = request.form['actor_3']

        year = int(y)

        comb = act1 + " " + act2 + " " + act3 + " " + dirc + " " + gen + " "

        test, sim = create_sim()

        test.loc[len(test.index)] = [year, dirc, act1, act2, act3, gen, movie, comb]

        test, sim = create_sim()

        test.to_csv('FINAL_MOVIE_TABLE.csv',index=False)

    return render_template('addmovie.html')
```

```
@app.route('/recommend', methods={'GET', 'POST'})
def recommend():
    if request.method == 'POST':
        if request.form['action'] == 'get' and 'movie_title' in request.form:
            m = request.form['movie_title']
            if len(m) != 0:
                return redirect(url_for('predict', movie=m))
            # else:
            # flash message for incorrect movie input
        elif request.form['action'] == 'add':
            return redirect(url_for('addMovie'))
        elif request.form['action'] == 'portal':
            return redirect(url_for('userportal'))
    return render_template('recommendation.html')



@app.route('/search', methods={'GET', 'POST'})
def search():
    if request.method == 'POST':
        if 'year' in request.form and 'director' in request.form:
            year = request.form['year']
            dir = request.form['director']
            return redirect(url_for('searched_movies', y=year, d=dir))
    return render_template('search.html')



@app.route('/search')
@app.route('/search/searched_movies/<y>/<d>')
def searched_movies(y, d):
    r = list()
    year = int(y)
    r = searchHelper(year, d)
    d = d.upper()
    return render_template('searched_movies.html', year=year, name=d, lst=r)



@app.route('/recommend')
@app.route('/recommend/predict/<movie>')
def predict(movie):
    r = list()
    r = rcmd(movie)
    movie = movie.upper()
    return render_template('predict.html', movie=movie, r=r)


if __name__ == '__main__':
    app.run(debug=True)
```

Testing

The following types of testing have been used in our project:

1. Unit Testing: This test is applied on each of the module to find whether each module is properly working or not.

2. Integration Testing: After each module cleared the unit testing then modules is tested for their working all together in the integrated testing phase.

3. Acceptance Testing: This testing provides the final assurance that the website needed all behavioral and performance requirements.

4. System Tests: The system test environment allows multiple modules to be connected together and executed as in a typical use-case scenario. The choice as to whether this is done on a separate machine from unit testing is up to the implementation and test team. If the target deployment environment is different from the unit test environment, the system test environment should contain a machine that matches the target environment.

User Interface

1. Human Interface Design

1.1. Overview of User Interface

1.1.1. Home Page Module

Description of Component
This module lets a person to either go on to use services of MovieSpy or view dataset as Administrator.
Interface Description
Inputs: Click on User or Admin Button.
Outputs: Takes to User login or Admin login page respectively.
Processing Details

- Display two buttons, User and Admin.
- Open User login or Admin login page as selected respectively.

1.1.2. User registration Module

Description of Component
User can register themselves on Moviespy in this module, in order to use its services.
Interface Description
Inputs: User id (user_id), password (upwd), password again(to verify), security question(question) and answer(answer).
Outputs: Takes to User Login page.
Processing Details

- Display a form with user id, password, password again, security question (drop-down list) and answer field.
- Mark all fields as compulsory.
- Open the User Login page after taking input from user.

1.1.3. User Login Module

Description of Component
This module authenticates users, so that, only legitimate people can use the service.
Interface Description
Inputs: User id (user_id), password (upwd).
Outputs: Takes to User portal.
Processing Details

- Display a form with User id and password field. Also displays Forget Password button and Register button.

- If user forgets password, user can click on Forget password button and change password after confirming security question and answer.
- Mark all fields as compulsory.
- Validate input from database to authenticate user.
- Open the User Portal after validation and authentication.

1.1.4. User Portal

Description of Component

Users can choose which service they want to use or they want to logout from existing session.

Interface Description

Inputs: Three Buttons namely, Search Movies, Get Recommendations and Logout.

Outputs: Takes User to Search Movies Module, Get Recommendations Module or to Home Page as opted by User.

Processing Details

- Displays three buttons, Search Movies, Get Recommendations and Logout.
- Takes to Search Movies Module if user clicks on Search Movies Button.
- And similar for Get Recommendations Button.
- Takes User to Home Page if User clicks on Logout Button.

1.1.5. Search for Movies Module

Description of Component

Users can search movies of a particular year and director through this module.

Interface Description

Inputs: Year of release of movie (movie_year), director of movie (director_name).

Outputs: Displays list of movies belonging to that year and genre.

Processing Details

- Displays a form with Year of movie release and director of movie fields.
- Mark all fields as compulsory.
- List of movies gets displayed.

1.1.6. Get Recommendations Module

Description of Component

Users get movie recommendations in this module just by entering a movie of their choice.

Interface Description

Inputs: Title of Movie (movie_title), get recommendation button, add movie button and user portal button.

Outputs: Displays recommended movies, or user can add movie or can go back to user portal.

Processing Details

- Displays a form with title of movie field and three buttons namely Get Recommendation, Add Movie and User Portal.
- Find recommendations with the help of content based filtering using title of movie entered by user, and display them.
- If user clicks on Add Movie button, they will go to Add Movie Module.
- If user clicks on User Portal Button, they will go back to user portal.

1.1.7. Add Movie Module

Description of Component

Users can add movies to the database if they want.

Interface Description

Inputs: Title of movie (movie_title), Year of release of movie (movie_year), Genre of movie (genre), Director of movie (director_name), Actor of movie (actor_1_name), Actor of movie (actor_2_name), Actor of movie (actor_3_name).

Outputs: Add movie to the database.

Processing Details

- Displays a form with Title of movie, Year of release of movie, Genre of movie, Director of movie, Actor of movie, Actor of movie, Actor of movie.
- Movie is added to the database.

1.1.8. Admin Registration Module

Description of Component

Admins can register themself on Moviespy in this module, in order to view its dataset.

Interface Description

Inputs: Admin id (admin_id), password (apwd), Confirmation key (key), password again(to verify), security question(question) and answer(answer).

Outputs: Takes to Admin Login Page.

Processing Details

- Display a form with user id, password, password again, confirmation key, security question(drop-down list) and answer field.
- Mark all fields as compulsory.

- Open the Admin Login Page after validation of Confirmation key and creating a user.

1.1.9. Admin Login Module

Description of Component

This module authenticates admin, so that, only legitimate people can view dataset and access system.

Interface Description

Inputs: Admin id (admin_id), password (apwd).

Outputs: Takes to Admin portal.

Processing Details

- Display a form with Admin id and password field. And also displays Forget password button and Register Button.
- If admin forgets password, admin can click on forget password and change password using security question.
- Mark all fields as compulsory.
- Validate input from database to authenticate admin.
- Open the Admin Portal after validation and authentication.

1.1.10. Admin Portal

Description of Component

Admin can choose what dataset they want to view.

Interface Description

Inputs: Four Buttons namely, View Movie Dataset, View User Dataset, View Admin Dataset and Logout.

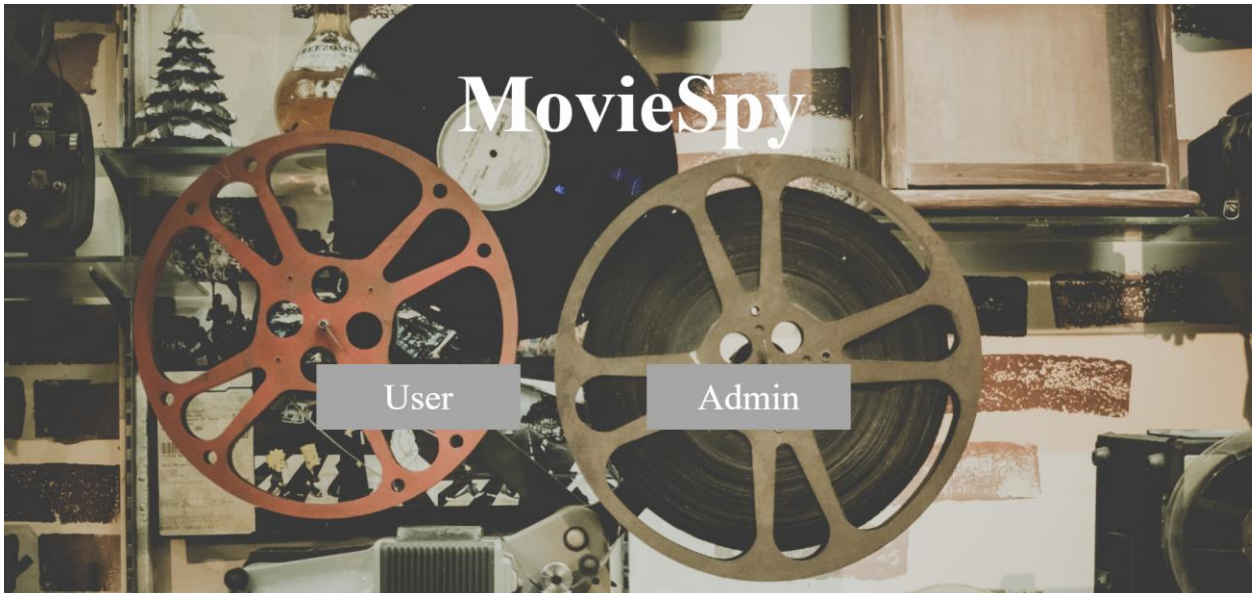
Outputs: Displays Movie dataset or User Dataset or Admin Dataset as selected by Admin or Logs out of the session.

Processing Details

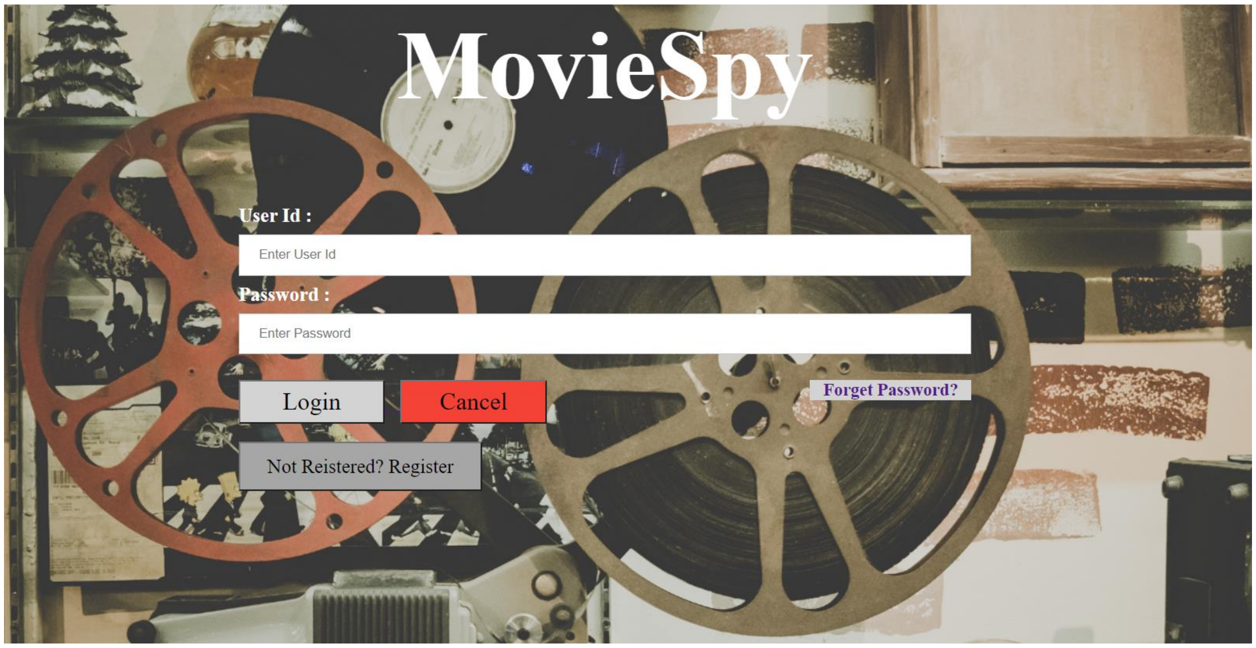
- Displays four buttons, View Movie Dataset, View User Dataset, View Admin Dataset and Logout.
- Displays Movie Dataset if Admin clicks on View Movie Dataset.
- Displays User Dataset if Admin clicks on View User Dataset.
- Displays Admin Dataset if Admin clicks on View Admin Dataset.
- Logs out of session if Admin clicks on Logout.

1.2. Screen Images

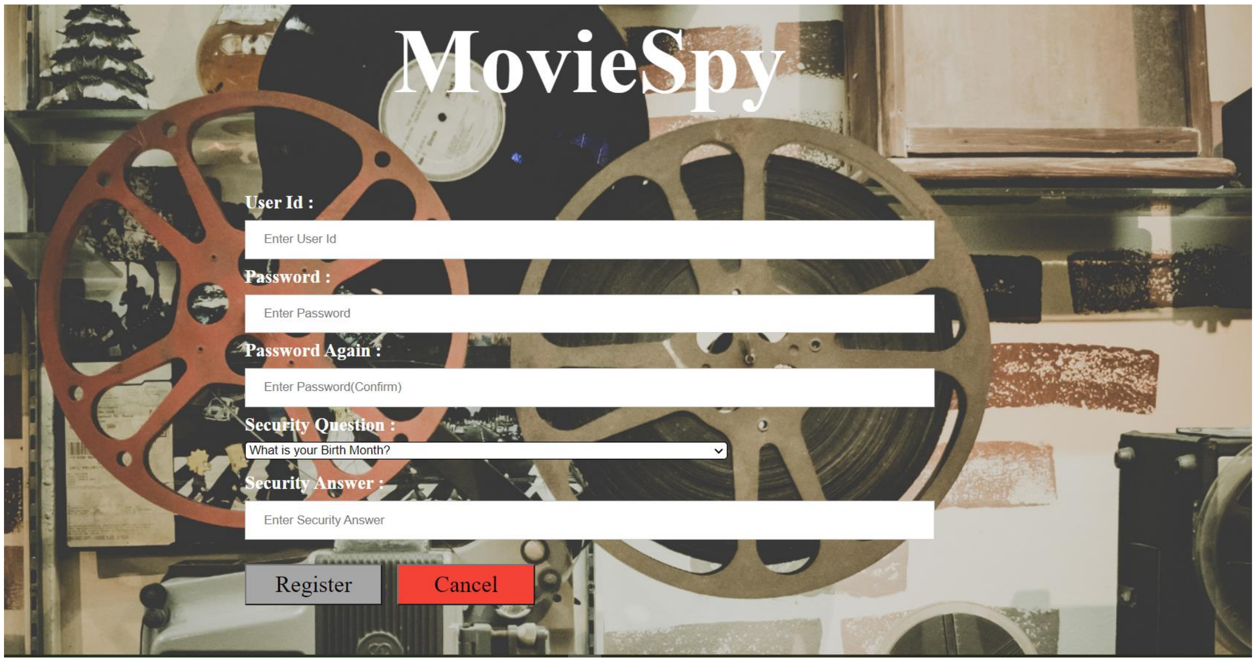
1.1.1 Home Page



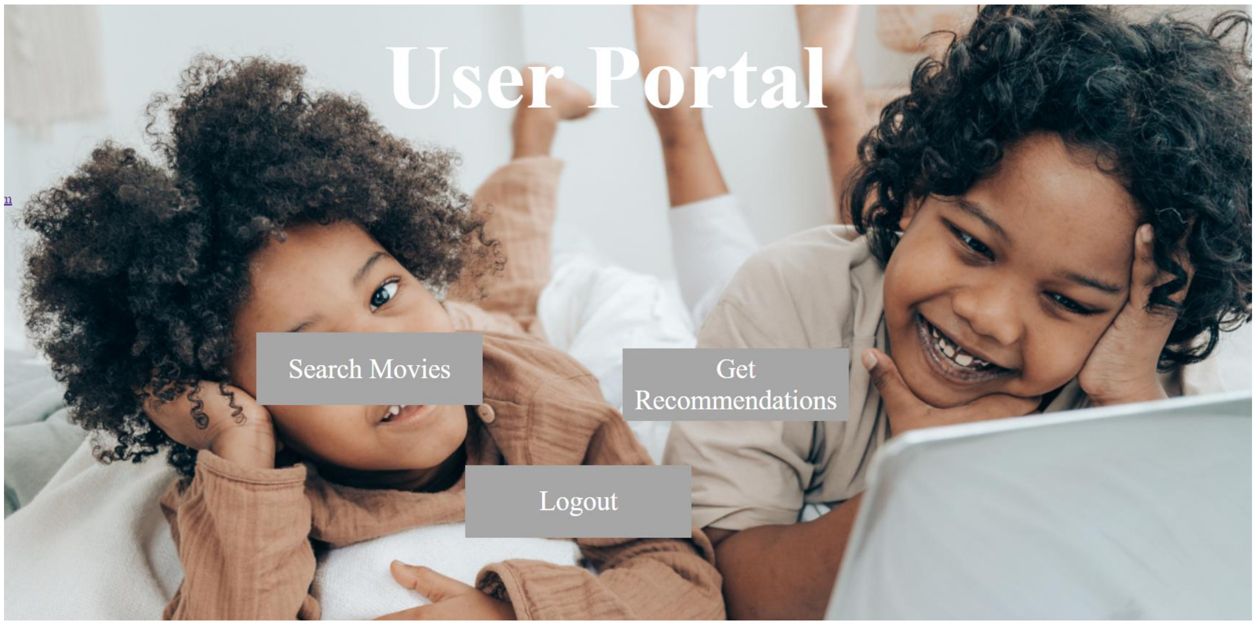
1.1.2 User Login Module



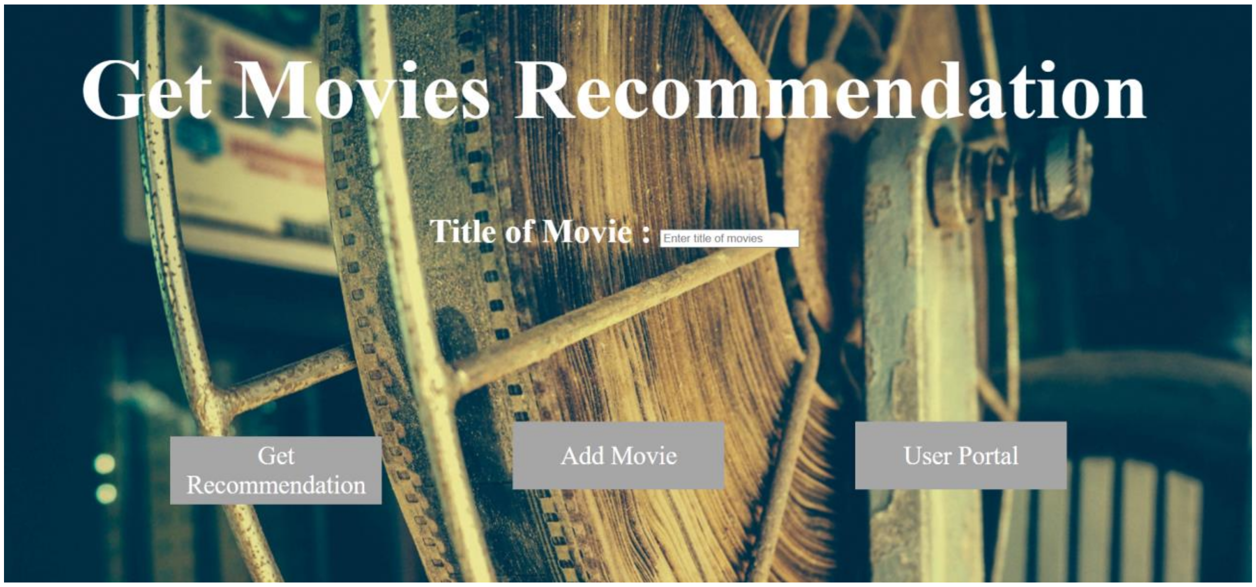
1.1.3 User Registration Module

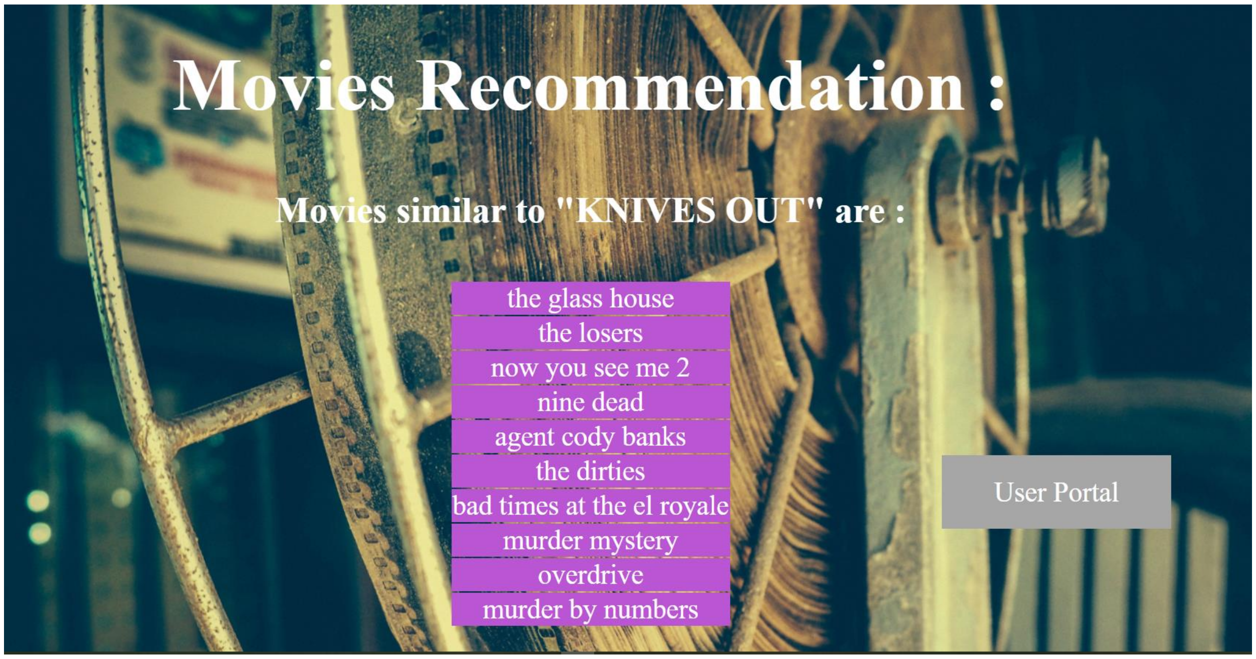


1.1.4 User Portal



1.1.5 Get Recommendations Module

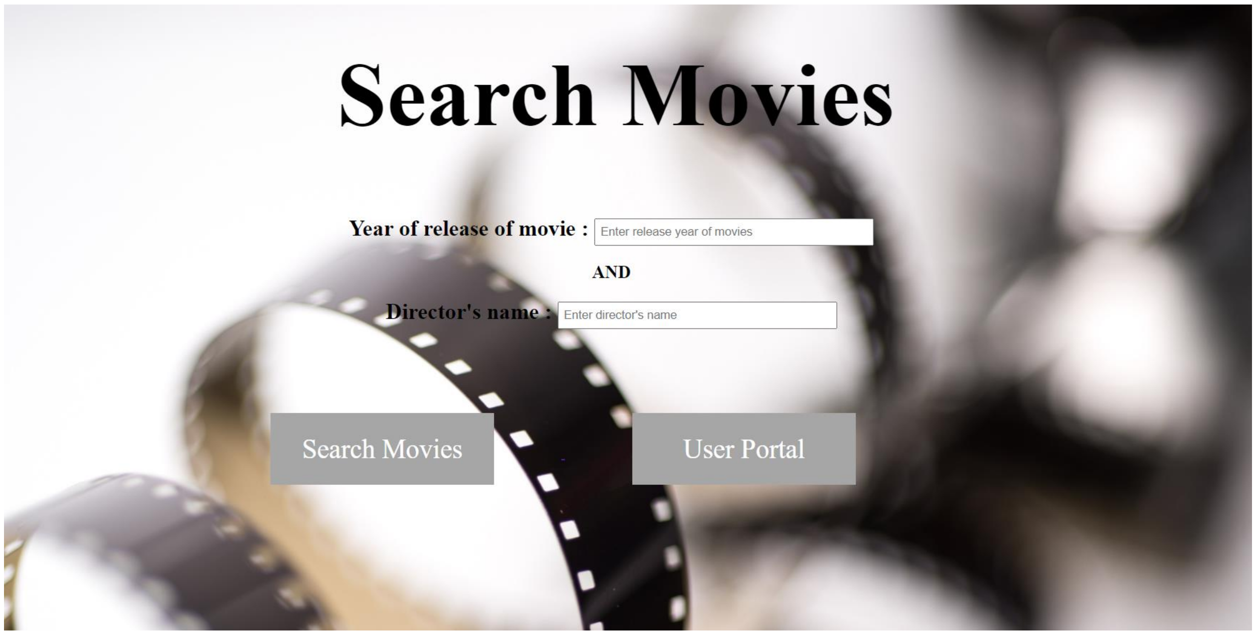




Movies similar to "KNIVES OUT" are :

the glass house
the losers
now you see me 2
nine dead
agent cody banks
the dirties
bad times at the el royale
murder mystery
overdrive
murder by numbers

1.1.6 Search Movies Module



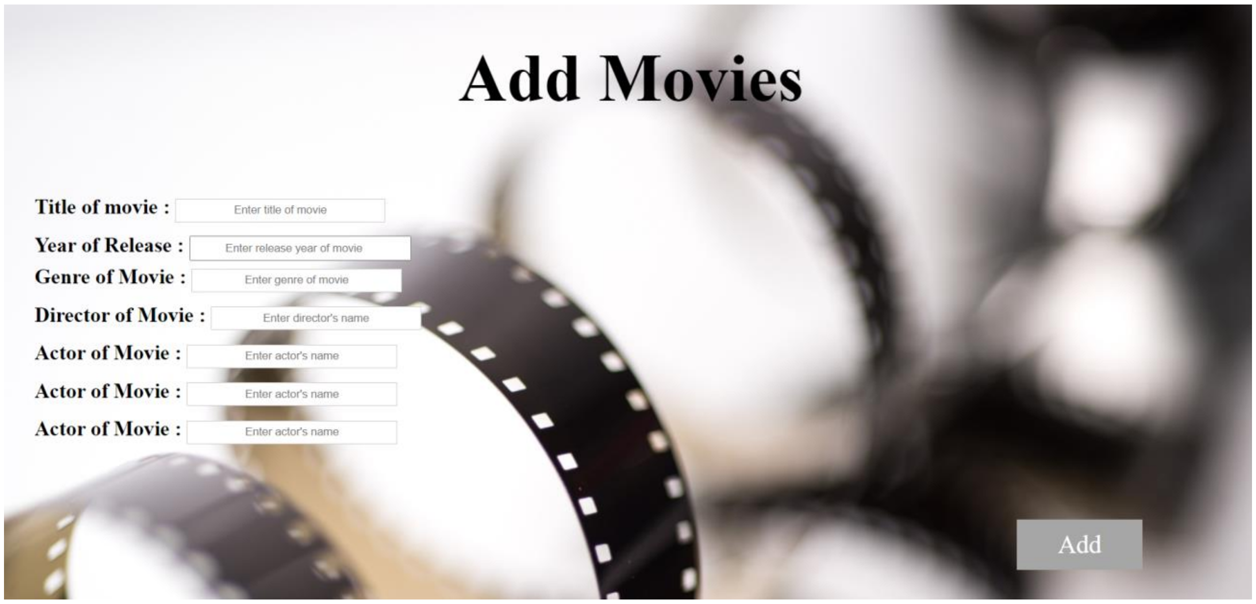
Search Movies

Year of release of movie : Enter release year of movies

AND

Director's name : Enter director's name

1.1.7 Add Movie Module



Add Movies

Title of movie : Enter title of movie

Year of Release : Enter release year of movie

Genre of Movie : Enter genre of movie

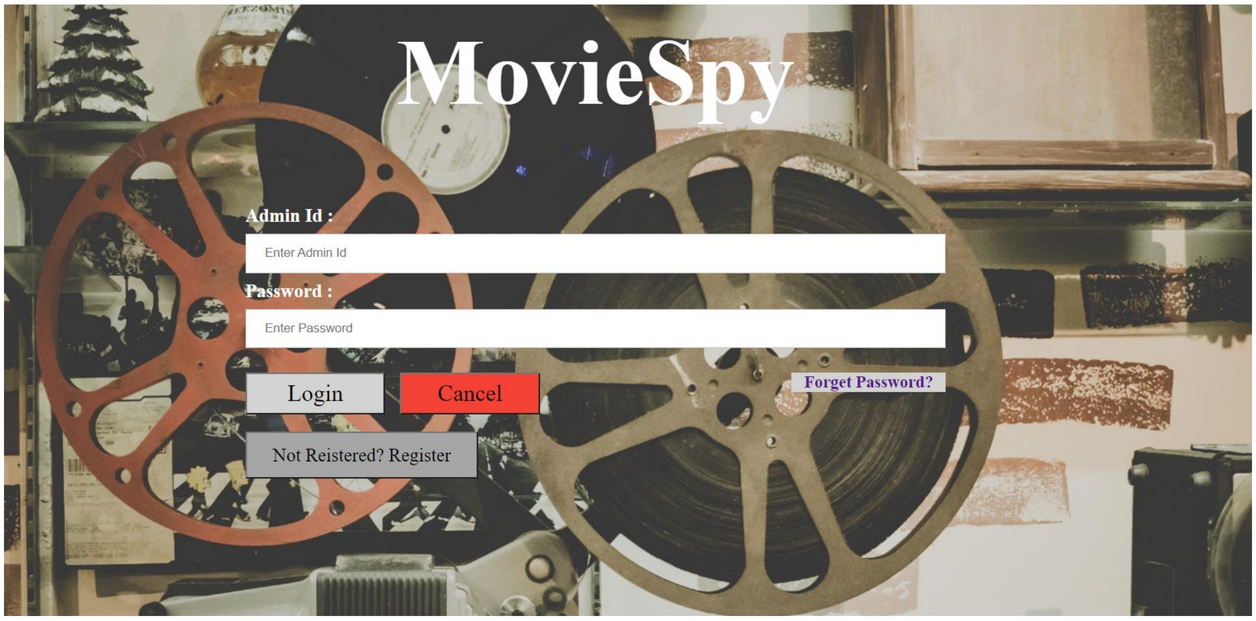
Director of Movie : Enter director's name

Actor of Movie : Enter actor's name

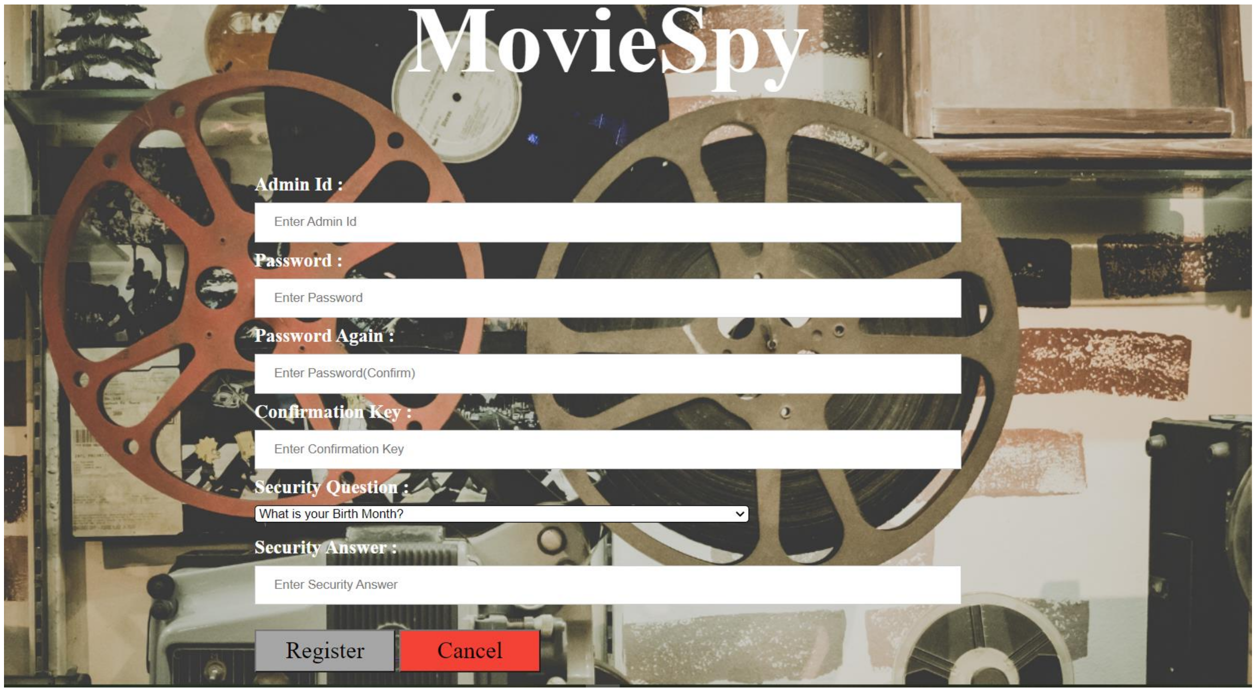
Actor of Movie : Enter actor's name

Actor of Movie : Enter actor's name

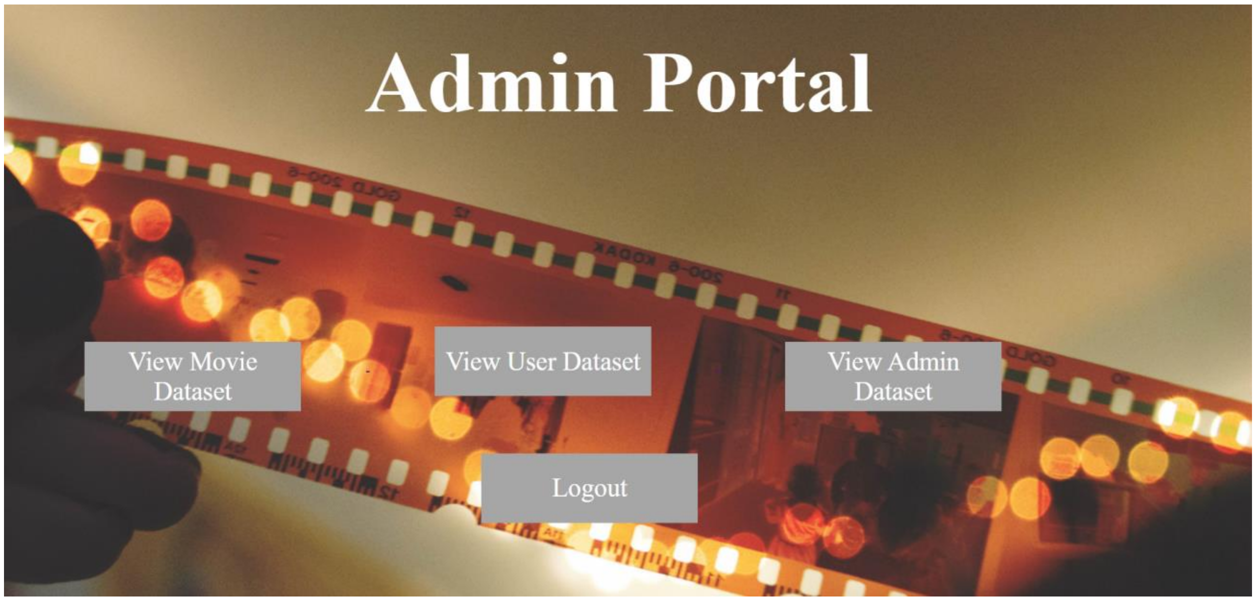
1.1.8 Admin Login Module



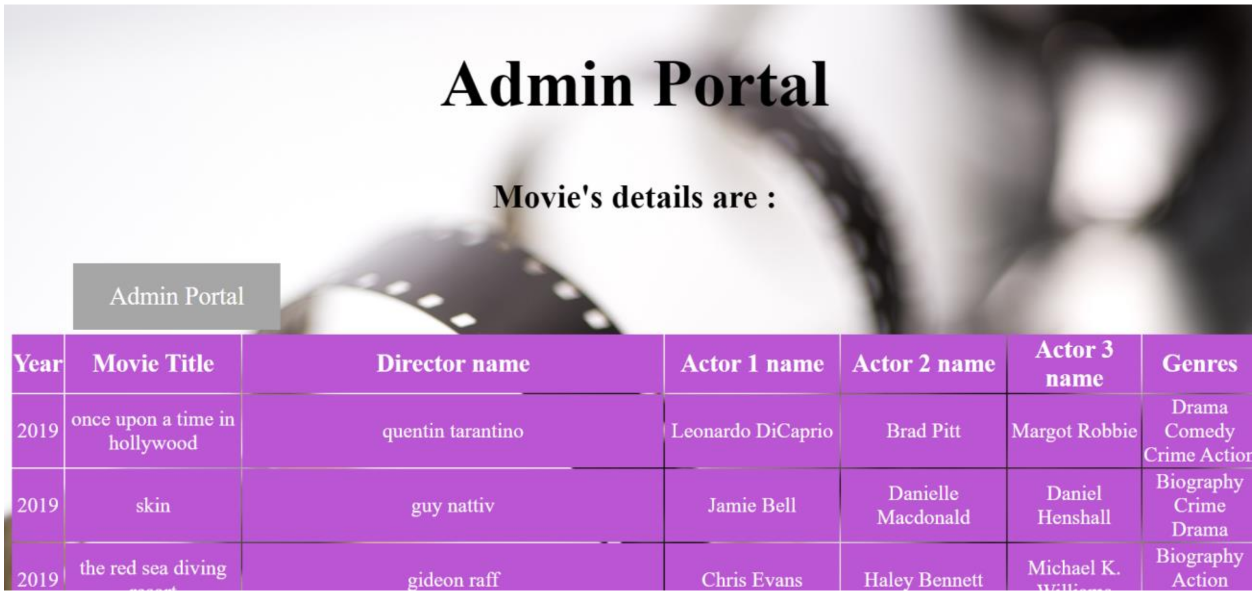
1.1.9 Admin Registration Module



1.1.10 Admin Portal



1.1.11 View Movie Dataset



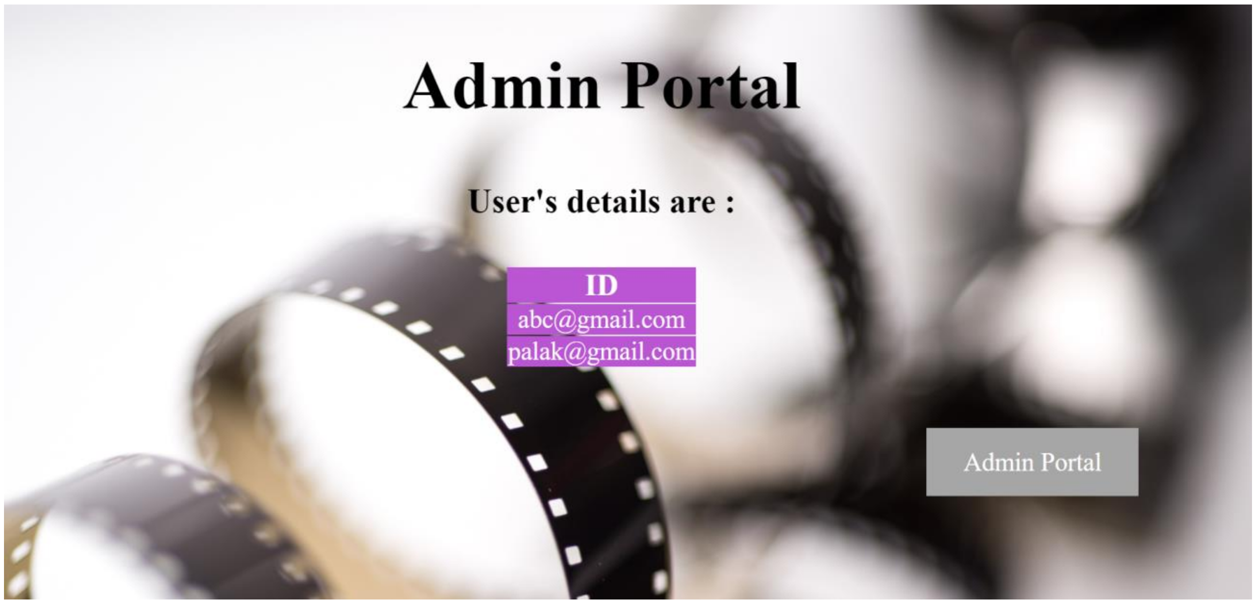
Admin Portal

Movie's details are :

Admin Portal

Year	Movie Title	Director name	Actor 1 name	Actor 2 name	Actor 3 name	Genres
2019	once upon a time in hollywood	quentin tarantino	Leonardo DiCaprio	Brad Pitt	Margot Robbie	Drama Comedy Crime Actio
2019	skin	guy nattiv	Jamie Bell	Danielle Macdonald	Daniel Henshall	Biography Crime Drama
2019	the red sea diving	gideon raff	Chris Evans	Haley Bennett	Michael K.	Biography Action

1.1.12 View User Dataset

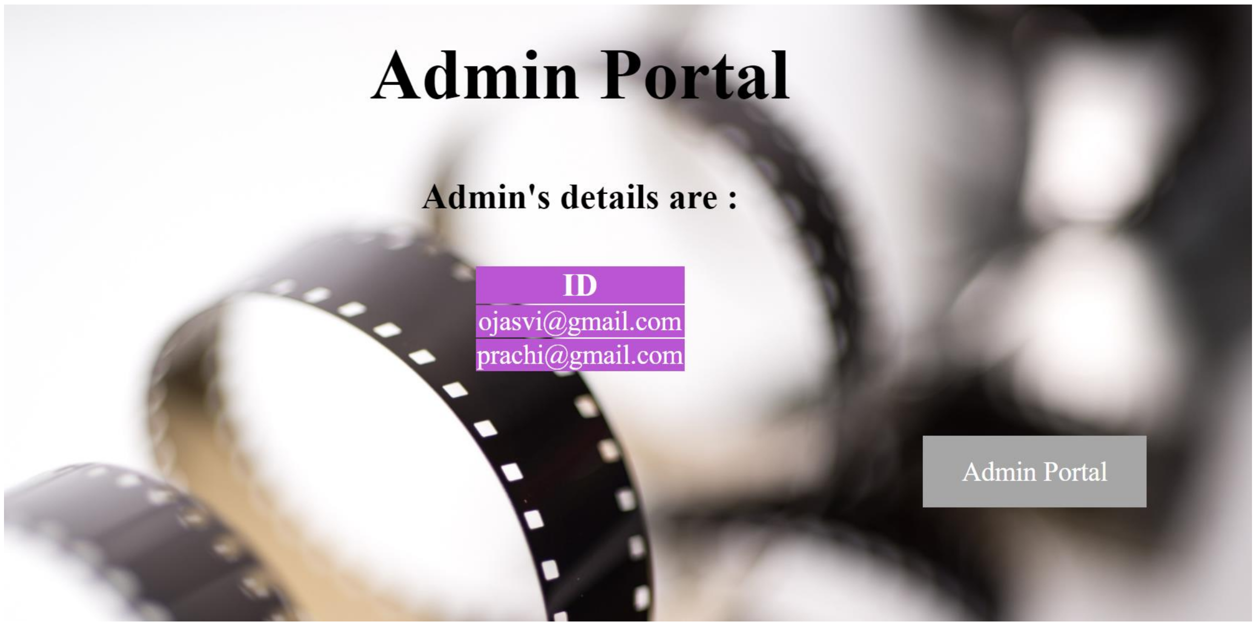


Admin Portal

User's details are :

ID
abc@gmail.com
palak@gmail.com

1.1.13 View Admin Dataset



Admin Portal

Admin's details are :

ID
ojasvi@gmail.com
prachi@gmail.com

Appendices

- ADMIN: Administrator is the controller of the entire system who maintains the all the records of the functionalities and the users.
- REGISTERED USER: A user who has signed up for an account and has logged in to the system.
- IEEE: Institute of Electrical and Electronics Engineers
- SERVER: The main computer on the network
- BROWSER: A software application used to locate and display the web pages.
- SRS: Software Requirement Specification
- HTML: Hyper Text Markup Language is a markup language used to design static web pages.
- RAM: Random Access Memory
- IIS: International Information Server
- DBA: Database Administrator
- HDD: Hard Disk Drive
- DFD: Data Flow Diagram
- AI: Artificial Intelligence
- ML: Machine Learning
- SQL: Structured Query Language

References

1. Pressman Roger S., Software Engineering “A Practitioner’s Approach” Fifth Edition, McGraw-Hill Publication, 2000.
2. Navathe Shamkant B., Fundamentals of Database Systems, Fifth Edition, Pearson Publication
3. Miguel Grinberg, Flask Web Development, Second Edition, O’Reilly Publication.
4. IEEE STD 830-1998, IEEE Recommended Practice for Software Requirement Specifications
5. Kaggle.com. 2021. *Rounak Banik | Kaggle*. [online] Available at: https://www.kaggle.com/rounakbanik
6. Medium. 2019. *Using Cosine Similarity to Build a Movie Recommendation System*. [online] Available at: <https://towardsdatascience.com/using-cosine-similarity-to-build-a-movie-recommendation-system-ae7f20842599>
7. GitHub. 2020. *oekosheri/Recommender-movie*. [online] Available at: https://github.com/oekosheri/Recommender-movie
8. L. Chameiko and Iateilang Ryngksai, “Recommender Systems: Types of Filtering Techniques,” vol. 3, November 2014.
9. Prem Melville and Vikas Sindhwani, "Recommender System", IBM T.J. Watson Research Center Yorktown Heights, pp. 1-18.
10. Chatrasada Gopan Sharma, “Movie Recommender System,” sites.google.com [Online] . Available: https://sites.google.com/site/chatrasadagopansharma/home/introduction.
11. Mahnoor Javed, “Using Cosine Similarity to Build a Movie Recommendation System,” towardsdatascience.com [Online]. Available: https://towardsdatascience.com/using-cosine-similarity-to-build-a-movie-recommendation-system-ae7f20842599.