# Film Recommendation System

# Contents

## Contents

Contents ..... 2
Description of the website will do ..... 8
Introduction ..... 8
Features for website for films ..... 8
Friends and Chatroom ..... 9
Conclusion ..... 9
Objectives ..... 10
Background research ..... 14
Current systems looked at ..... 14
Website tutorial. ..... 14
Best searching algorithm article ..... 15
Code wrote ..... 15
Results ..... 15
Finding best way to create recommendations ..... 18
End users ..... 18
Questionnaire which was asked ..... 18
Response from person 1 ..... 19
Response from person 2 ..... 19
Response from person 3 ..... 19
Response from person 4 ..... 20
Response from person 5 ..... 20
Response from person 6 ..... 21
My opinion from the research carried out ..... 21
Colours and images used ..... 21
Limitations ..... 25
Modal ..... 26
Encryption ..... 26
Web scraping for titles and IMDB link ..... 27
Web scraping for descriptions from the IDMB link of film ..... 28
Tested mini search algorithm on mini database ..... 28
Tested to make sure flask displayed everything nicely ..... 28
Modules used ..... 29

flask.session extra information ..... 33
Why is Flask used? ..... 33
Why Jinja2 is being used for template code? ..... 33
Why Bootstrap is being used for styling? ..... 33
Why is JavaScript used? ..... 34
Gathering Information For Database ..... 35
Step 1 ..... 35
Imports used ..... 35
Functions defined in this file ..... 35
Step 2 ..... 35
Database structure ..... 36
Imports used ..... 36
Functions defined in this file ..... 36
Film Class ..... 37
Step 3 ..... 39
Database structure ..... 39
Imports used ..... 39
Functions defined in this file ..... 40
Step 4 ..... 42
Imports used ..... 42
Functions defined in this file ..... 42
Database structure ..... 43
PersonClass Class ..... 44
Step 5 ..... 46
Database structure ..... 46
Functions defined in this file ..... 46
Step 6 ..... 48
Database structure ..... 48
Imports used ..... 48
Functions defined in this file ..... 48
Step 7 ..... 50
Database structure ..... 50
Functions defined in this file ..... 50
Step 8 ..... 51
Database structure ..... 51
Functions defined in this file ..... 52

Step 9 ..... 52
Database ..... 55
Entity Relationship Diagram (ERD) ..... 55
Box meanings ..... 55
Information about each field ..... 56
Film table ..... 58
Economy table ..... 59
Actors table ..... 60
ActorIntegrator table ..... 61
Director table ..... 62
DirectorIntegrator table ..... 63
GenreNames table ..... 64
GenreToFilm Table ..... 65
Language table ..... 66
LanguageToFilms table ..... 67
Users table ..... 68
Favourites table ..... 68
Ratings table ..... 70
TopRated table ..... 71
Recommendations table ..... 72
Flowcharts ..... 73
Box colours and meaning ..... 73
Searching algorithm ..... 74
Searching for films, actors and directors page ..... 76
Leaderboard algorithm ..... 79
Leaderboard page ..... 81
Recommendation algorithm ..... 83
Starts the algorithm ..... 83
Directors ..... 84
Actors ..... 85
Checks amount of films gathered ..... 86
Runtime ..... 86
Release date ..... 87
Colour ..... 87
Language ..... 88
Genre ..... 89

Adding the recommendations to the database ..... 92
Recommendation page ..... 93
SQL Statements used ..... 95
Information about functions used ..... 95
select_info_from_database function ..... 97
searching_algorithm.py ..... 97
leaderboard.py ..... 99
gathering_types.py ..... 100
formulating_recommendations_from_types.py ..... 101
formats_and_creates_dictionary_for_films.py ..... 102
inserts_info_to_database function ..... 103
user_interactions_with_database.py ..... 103
deletes_from_database function ..... 103
app.py ..... 104
updates_database function ..... 104
user_interaction_with_database.py ..... 105
Navigation ..... 105
Meaning of boxes ..... 106
Before the user logs in ..... 106
How it will look on the website ..... 108
Once the user has logged in ..... 110
Left hand side ..... 110
How it will look on the website ..... 112
Right hand side ..... 123
How it will look on the website ..... 124
Home page ..... 124
News modal ..... 124
Logout ..... 124
Account Settings ..... 124
Change password ..... 125
Forgot password ..... 125
Enter email address ..... 125
Folders, Files, Classes and Functions ..... 126
app.py ..... 127
Imports used ..... 127
Functions ..... 128

classes.py ..... 133
Classes ..... 133
formulas.py ..... 135
Imports used ..... 135
Functions ..... 135
database folder ..... 136
MainDB.db ..... 136
user_information folder ..... 137
login_register_changepassword_form.py ..... 137
encryption.py ..... 141
sending_emails.py ..... 142
user_interactions_with_database.py ..... 143
email_details.json ..... 144
rating_questions.json ..... 144
security_questions.json ..... 144
email_contents folder ..... 144
searching folder ..... 145
searching_algorithm.py ..... 145
leaderboards.py ..... 148
description_of_films.py ..... 149
recommend folder ..... 150
gathering_types.py ..... 150
formulating_recommendation_from_types.py ..... 153
dictionary_creator_for_html_pages folder ..... 158
formats_and_creates_dictionary_for_films.py ..... 158
Functions ..... 158
formats_and_creates_dictionary_for_people.py ..... 159
templates folder ..... 160
home.html ..... 160
layout.html ..... 160
search/films folder ..... 160
search/people folder ..... 161
user_information folder ..... 161
static folder ..... 162
main.css ..... 162
Python code to create database ..... 163

These are the python scripts used in order ..... 163
Python code ..... 163
Folder and file structure ..... 164
layout.html ..... 165
home.html ..... 166
register.html ..... 166
login.html ..... 166
account_settings.html ..... 166
forgot_password.html ..... 166
search_for_films.html ..... 166
leaderboard.html ..... 167
display_film_information.html ..... 167
search_form_people.html ..... 167
display_people_information.html ..... 167
main.css ..... 167
all_genres.json ..... 167
all_languages.json ..... 167
common_words.json ..... 167
email_details.json ..... 168
security_questions.json ..... 168
rating_questions.json ..... 168
Test plan ..... 170
Test 12.1 ..... 190
Test 14.2 ..... 190
Objectives met ..... 191
Reason for not adding chatroom ..... 196
Reason for not adding friend feature ..... 196
Timeline ..... 197
User's opinions on the outcome ..... 200
Questions asked ..... 200
Response from person 1 ..... 200
Response ..... 200
Response from person 3 ..... 200
Response ..... 200
My opinion on the outcome ..... 200
What I want to add later ..... 201

# Analysis

## Description of the website will do

### Introduction

For this project/website I will be designing and creating a film recommendation system. People will be able to register as users on the website to access many features. The main reason why I chose this specific project is because I spend a lot of my spare time watching films. Films are a great way to bring people together and share common interests. Regardless of their background most people can find a film that they will bond with someone else with a common interest.

Another reason why I want to create a film recommendation system is because of my fascination with how recommendations are created. For as long as I can remember I have wanted to learn more about recommendations and systems which predict users' preferences. I am especially interested in how Google creates their YouTube recommendations and how Amazon recommends which products users may like to buy. In designing and creating a film recommendation system, I intend to learn more about Data Science, this is important to me because I want to focus on developing my skills further around. Investigating and working with data by analysing and predicting large data and finding and identifying patterns in them.

### Features for website for films

There will be many features of the film recommendation system that I will create. It will be a cross platform system so that the user can access it on any device with an internet connection, as long as it has a browser on it.

For the user's recommendation, I will want to create my own predictive algorithm which will take all the user's currently favourited and rated films and look into more depth and detail about them to create a personalised recommendation for the user. I will base all recommendations personally on the user including information about their age and location, to find people with similar recommendations and films they liked who live around the same area and are around the same age. I will also look at details about the film, including the actors which are in it and the directors which have directed it to find similar films which they have acted in and directed. In addition, I will examine the languages and genres of the film to see if the user likes films from a large scope of languages and genres to find which the user prefers over others. I will also look to see if the user prefers black and white films or coloured films, and the length of the film to see if there is a specific runtime that the user wants. Another area that will be taken into consideration will be the release date, to predict if the user likes newer films or really old films.

To enable the website to work at its full capacity and create recommendations, the user will need to create an account by registering. This will be essential for users if they would like to rate, favourite and gain recommendations. After the user has registered to the website, they will be able to login and access all features on the website. When not logged on to the website the only options for the user will be to login, register forget their password, or use the random chatroom.

When logged in to the website the user will be able to search for a film and use filters to search by language, genre, release date, or colour of the film. The filters available when the user is searching for an actor or a director include the year which the actor or director was born, and the location where they were born.

When the user has searched for a film, a window will appear with all the film information available. On this page all the information about the film will be displayed, including an image of the film, title, release date, genres, languages, actors, directors, film length, and colour of the film. There will also be the ability for the user to rate, favourite, and dislike or like the recommendation if the user searched for a film through the recommendations page.

When the user is looking at the information about the films, they will be able to view all the actors and directors which have been in the film. The user can then click on a specific actor or director to find more information about them, this will be displayed in the same style as when the user searches for an actor or a director.

Once the user has searched for an actor or a director, a page will be displayed with all the information about them, this will include their place of birth, their date of birth, their date of death (if they have died), all their spouses and children, and all the films which they have been in. The user is able to click on the films and they will be redirected to a page in a similar format to when the user searches for a film. If the user decides to view their favourite films, they can click on their favourites page. This will display all their favourited films. When the user has rated and favourited enough films, a recommendation will be created for them. They can access their recommendations through their recommendations page and these will be displayed to like or dislike. The user can click on the recommendations and view them all, and if they like the recommendation they can click the like button, or if they don't like it, click the dislike button. They can also favourite the recommendations so that they can remember to watch it later if they have not already watched it.

## Friends and Chatroom

The user also has the ability to add a friend by searching for them by their user ID. To obtain their friend's user ID they will need to ask them for it so that they can search for them. Once they find their friend, they will send their friend a request and they can accept it. Through the friend feature they will have the ability to directly message their friend if they choose to, as well as the option to create a chatroom with multiple friends up to 5 friends.

One feature of the chatroom for the user includes the ability to message anybody they want. Some of the designs of the chatroom also includes the ability to message a random person, generate a chatroom with random people, generate a chatroom with 5 people, or generate a chatroom with specific people through their friends. Users can create chatrooms with their friends, but their friends don't have to be friended with others in the chatroom.

## Conclusion

In conclusion, my goal of this website is to create a more personalised recommendation system, so the user can gather recommendations and find films which they will like better. This will be more personalised for the user and will be less determined on other popular films which other people like.

# Objectives

In this section, I will list everything which I want my website to do, these are all of my ideas and my end goal of what I want it to do. There are 7 main objectives broken down into smaller parts, and those smaller parts broken down again and again. The main 7 objective categories are; A fast cross platform Website, Display everything on a website, User information stored, Searching for films actors and directors, user can rate films, a recommendation for films are created and finally the chatroom.

1. **Platform to use**
    a. Website based so that can be cross platform and can access anywhere with an internet connection, and any device.
    b. Will be using the python flask module, for the web framework for the code of the website, as it is lightweight and fast
    c. Fast and accurate searching for films, actors and directors the user wants to find information about.

2. **Website**
    a. Display everything on a website
    b. Have multiple pages which have different links for the user to find information about
        i. Register
        ii. Login
        iii. Forgot password
        iv. Change password
        v. Search for film
        vi. Search for actor
        vii. Search for director
        viii. Favourites
        ix. Recommendations
        x. Friends
        xi. Chatroom
            1. Completely random
            2. Random with interest
            3. Chosen by user
    c. The website must be easy to use, easy to understand
    d. It must also be as fast as possible loading pages so that people do not get bored waiting
        i. Waiting image appears if the page does not load instantly
        ii. All loads between pages are below 3 seconds

3. **User accounts**
    a. User can Register if haven't before.
        i. Will tell the user if the username or email entered already exists
        ii. Adds information to the database with the password and security questions encrypted, including;
            1. Password

2. Security questions
3. Username
4. Name
5. Email
6. Date Registered
7. Date of birth

iii. Send confirm nation email of the account being created to the user.

b. User can Login if they have registered.
   i. Once the user has logged in they will be able to access all their information for;
      1. Favourite films saved
      2. History of the past 5 films they have looked at
      3. Friends list
      4. Recommended films
c. User can change their password
   i. If the user wishes to change their password they may do so
   ii. If the has forgot their password they can request it by entering their email address and it will sent them an email of their current password

**4. Searching**

a. Once the user has logged in they will be able to search for a film, actor or director
   i. Display all information about films for either when the user searches for a film, looks at their favourited list or is looking at their recommendations are;
      1. Display release date
      2. Display title of film
      3. Display genres related to film
      4. Display languages related to film
      5. Display runtime of film
      6. Display colour of film
      7. Display actors in film
      8. Display directors who directed film
      9. Allow the user to favourite and rate any film displayed to them.
   ii. Display all information about actors and directors (information displayed for both are in the same format)
      1. Name of person
      2. Date of birth
      3. Death date (if dead)
      4. Place of birth
      5. Spouses and children
      6. Height
      7. Films acted in or directed

b. When the user clicks on their favourite list it will display all of their favourited films only.
c. The user can click on their recommendations page to see all of their recommendations
d. Search for friends they wish to add to their friends list

5. **User can rate films**
    a. The user can rate any film they see, it will display a rating system consisting of 4 categories from 1 to 10 where 10 is the highest rating and N/A is for unknowns (the user did not rate in this category), the categories are;
        i. (1-10) Comedy value, how funny was the film
        ii. (1-10) Overall, the user's overall opinion on the film
        iii. (1-10) Quality, how well was the film produced
        iv. (1-10) Actors, how well did the actors act in the film
    b. This will be used in the recommendation process, where it will try and find films, based off of what the user has favourited films and their favourited films,
        i. If a film has a rating of "N/A" then the rating will be set to 5 as it is the middle value. more based upon that specific rating of that film.
        ii. Using the values which the user has rated, it will use these to try and balance each film depending on its attributes such as; genre, actors and directors
    c. All of this information will be stored in a database, but the user will not be able to see the ratings once they have rated the film

6. **Recommendation algorithm**
    a. Identify the films which the user will like based upon their favourited and rated films. As long as the user has enough rated and favourited films
    b. The algorithm will try and predict 30 films for the user at once
        i. The user will be able to click on a button to get new recommendations whenever they want
        ii. There will be an option to say if the recommendation is good or bad to help predict the recommendation algorithm
    c. Use the user's ratings if there are any when calculating the recommendation for the user, it will also filter out the films which the user does not want if the rating is too low. Will prioritize films with a higher rating over others, and discriminate against films which have a lower rating.
        i. I will use overall rating for the genre
        ii. The actors rating for the actors
        iii. Quality rating for the directors
        iv. The comedy weighting will be used for the genres if there is a "Comedy" genre in there
    d. The algorithm will predict films for the user by finding;
        i. All the films which have the most common and liked actors and directors in the user's currently rated and favourited films
        ii. Once those films have been found it will look at the details about each film to try and narrow down the recommendation set by;
            1. Giving films with the most common genre combination in the user's favourited and rated films a higher weighting over other films.
            2. Finding the average runtime to try and remove films which the user will find too long or too short.
            3. Find the most common language in the user's favourited and rated films, and remove all of the films which are not this language as the user will not be able to understand it otherwise.

        a. Will potentially try and find second and third most common language as the user may be bilingual
    4. See if the user likes black and white films or just films in colour, if it is a mixed set of film colours, then will create a proportion of which should be in colour and which should be in black and white
e. I will design a machine learning algorithm to effectively use all of this information and to predict the best films to the user for them to watch.

7. **Chatroom**
    a. A chatroom for users with similar interests to talk about films they like
    b. Allow users logged in or not to enter a chatroom
    c. If the user is logged in give them an option to join a chatroom with people with similar interests as them
        i. It will find people with similar interests as them, by finding the films they both have highly rated and favourited.
        ii. Save the chat and who sent them in a JSON file with a timestamp, so that the user's can see the history of the chat when they are offline.
    d. Allow the user to join a chatroom with the people they
        i. They can add the people from their friend list by clicking on their name or by searching for a user's user identification number.

# Background research

## Current systems looked at

For my secondary research for my project, I decided to look at 3 large companies which do the same sort of the thing which I will be designing and creating; Netflix, YouTube and IMDB. In this section I will explain, some off the cons and pros of those systems which those companies use and how I want to improve on it.

With Netflix, it is a paid subscription and user's may not want to actually want to watch the films on there, since they may be signed up to another system which has a larger portfolio of films to watch. Also, some people may just want to use a system to get a quick recommendation and then go to the store and buy the film as a hard copy instead of just having it digitally, as well as some people may not be able to afford the subscription. For all of these reasons I wanted to design a free website where users can get quick recommendations without having any ties to it in terms of subscriptions and money.

Another system is YouTube, there recommendation system is good and is free, however I have found after some while the recommendations which they give you are repetitive and the same. As well as less personalized to the user and based off what other people like. Another thing with YouTube is that they are mainly focussed on their own videos and not films made by directors in the mainstream. Since they have a good recommendation algorithm this is a drawback to them that they lack content. For all of these reasons I want to make my system more random and less repetitive and more personalized to the user, as well as having a plethora of films for the user to view, favourite, rate and gain recommendations about.

The final system which I looked at was IMDB, I will be using their system to gather all of the information in my database, I will web-scrape their website using a python script and store the information in a database on my machine, I will not gather all of the information for all of the films on their website as there is too much to store. This is a positive of IMDB is that they have a large collection of data for a plethora of films. However, their recommendations for the user's are not as personalized, they rely on other users, however they do become more personalized, once the user has interacted with their system more. With my system I want the recommendations to instantly be personalized to the user. As well as making my system simpler to use for the user, as IMDB is very cluttered and has a lot of useless features in my opinion.

## Website tutorial.

The first thing I have decided to watch some Flask tutorials on YouTube, on how to make a website. These will help me get an understanding of how I want to layout and use my website. I have watched 4 of the episodes already on how to create your website and have started to use it to design mine. I will show what I have done for far in my Modal section. I downloaded the code which they had available on there to view, and I tried to work my website off of their code. The main layout template (I have renamed to home) contains some of the person's code, as well as the CSS file. I will probably will also be using the FlaskForm module they used in the tutorial as it seems to make the process a lot easier.

In the description he has a link to the code which he wrote in the video, and there I downloaded it and worked from it.

Here is a link to the first video:- https://www.youtube.com/watch?v=MwZwr5Tvyxo

Here is a link to the playlist:- https://www.youtube.com/playlist?list=PL-osiE80TeTs4UjLw5MM6OjgkjFeUxCYH

A link to the code on github:- https://github.com/CoreyMSchafer/code_snippets/tree/master/Python/Flask_Blog

## Best searching algorithm article

I needed to find a searching algorithm, so I decided to look online to find some of the easiest and fastest ones to implement. I stumbled across the exponential search which built off a binary search, so I decided to search it more. I went onto stack overflow to find information comparing the two algorithms, but I was only able to find information comparing the fundamentals of the algorithms, such as the time and space complexity. I wanted to see time comparisons, so I wrote my own program to do so.

Article which I read:- https://stackoverflow.com/questions/52844293/exponential-search-vs-binary-search

## Code wrote

## Results

**First run**

```
Time to gather all films: 0.45302581787109375
Si Darna at ang Planetman
Time taken when searching from database: 0.03800177574157715... and position is (13217, 'Si Darna at ang Planetman')
Time taken for Linear search is: 0.023001432418823242... and position is 197438
Time taken for regular binary search is: 0.0009999275207519531... and position is 197438
Time taken for recurive binary seasrch is: 0.0... and position is 197438
Time taken for exponential search is: 0.0... and position is 197438
Name of the film is:  Si Darna at ang Planetman, and ID is: 13217

Roommates
Time taken when searching from database: 0.016000986099243164... and position is (19948, 'Roommates')
Time taken for Linear search is: 0.022001266479492188... and position is 184270
Time taken for regular binary search is: 0.0... and position is 184271
Time taken for recurive binary seasrch is: 0.0... and position is 184271
Time taken for exponential search is: 0.0... and position is 184271
Name of the film is:  Roommates, and ID is: 100374

Bardi
Time taken when searching from database: 0.040002346038818360... and position is (259113, 'Bardi')
Time taken for Linear search is: 0.003000020980834961... and position is 23902
Time taken for regular binary search is: 0.0... and position is 23902
Time taken for recurive binary seasrch is: 0.0... and position is 23902
Time taken for exponential search is: 0.0... and position is 23902
Name of the film is:  Bardi, and ID is: 259113

Aandal
Time taken when searching from database: 0.041002511978149414... and position is (139627, 'Aandal')
Time taken for Linear search is: 0.0009999275207519531... and position is 6403
Time taken for regular binary search is: 0.0... and position is 6403
Time taken for recurive binary seasrch is: 0.0... and position is 6403
Time taken for exponential search is: 0.0009999275207519531... and position is 6403
Name of the film is:  Aandal, and ID is: 139627

Darmiyaan: In Between
Time taken when searching from database: 0.03800225257873535... and position is (108551, 'Darmiyaan: In Between')
Time taken for Linear search is: 0.006000041961669922... and position is 49518
Time taken for regular binary search is: 0.0... and position is 49518
Time taken for recurive binary seasrch is: 0.0... and position is 49518
Time taken for exponential search is: 0.0... and position is 49518
Name of the film is:  Darmiyaan: In Between, and ID is: 108551
```

**Second run**

```
Time to gather all films: 0.47302699089050293
Ibok 3 hyeongje
Time taken when searching from database: 0.041002511978149414... and position is (20407, 'Ibok 3 hyeongje')
Time taken for Linear search is: 0.014000892639160156... and position is 96127
Time taken for regular binary search is: 0.0... and position is 96127
Time taken for recurive binary seasrch is: 0.0... and position is 96127
Time taken for exponential search is: 0.0010001659393310547... and position is 96127
Name of the film is:  Ibok 3 hyeongje, and ID is: 20407

Cuatro piernas
Time taken when searching from database: 0.04400277137756348... and position is (129939, 'Cuatro piernas')
Time taken for Linear search is: 0.006000041961669922... and position is 46505
Time taken for regular binary search is: 0.0... and position is 46505
Time taken for recurive binary seasrch is: 0.0... and position is 46505
Time taken for exponential search is: 0.0009999275207519531... and position is 46505
Name of the film is:  Cuatro piernas, and ID is: 129939

Gumnaam Hai Koi
Time taken when searching from database: 0.04100227355957031... and position is (58896, 'Gumnaam Hai Koi')
Time taken for Linear search is: 0.011000633239746094... and position is 84031
Time taken for regular binary search is: 0.0... and position is 84031
Time taken for recurive binary seasrch is: 0.0... and position is 84031
Time taken for exponential search is: 0.0... and position is 84031
Name of the film is:  Gumnaam Hai Koi, and ID is: 58896

Gurgaon
Time taken when searching from database: 0.04300236701965332... and position is (224307, 'Gurgaon')
Time taken for Linear search is: 0.011000633239746094... and position is 84262
Time taken for regular binary search is: 0.0... and position is 84262
Time taken for recurive binary seasrch is: 0.0... and position is 84262
Time taken for exponential search is: 0.0... and position is 84262
Name of the film is:  Gurgaon, and ID is: 224307

Mig og min lillebror og Bølle
Time taken when searching from database: 0.04100227355957031... and position is (13817, 'Mig og min lillebror og Bølle')
Time taken for Linear search is: 0.02000117301940918... and position is 143345
Time taken for regular binary search is: 0.0... and position is 143345
Time taken for recurive binary seasrch is: 0.0... and position is 143345
Time taken for exponential search is: 0.0... and position is 143345
Name of the film is:  Mig og min lillebror og Bølle, and ID is: 13817
```

**Third run**

```
Time to gather all films: 0.472027063369751
Acilar içinde
Time taken when searching from database: 0.037001848220825195... and position is (85476, 'Acilar içinde')
Time taken for Linear search is: 0.0010001659393310547... and position is 7526
Time taken for regular binary search is: 0.0... and position is 7526
Time taken for recurive binary seasrch is: 0.0... and position is 7526
Time taken for exponential search is: 0.0... and position is 7526
Name of the film is:  Acilar içinde, and ID is: 85476

Evanukku Engeyo Matcham Irukku
Time taken when searching from database: 0.04000210762023926... and position is (249707, 'Evanukku Engeyo Matcham Irukku')
Time taken for Linear search is: 0.009000539779663086... and position is 69303
Time taken for regular binary search is: 0.0... and position is 69303
Time taken for recurive binary seasrch is: 0.0... and position is 69303
Time taken for exponential search is: 0.0... and position is 69303
Name of the film is:  Evanukku Engeyo Matcham Irukku, and ID is: 249707

The Bruise on the Olive
Time taken when searching from database: 0.039002180099487305... and position is (138216, 'The Bruise on the Olive')
Time taken for Linear search is: 0.027001380920410156... and position is 215498
Time taken for regular binary search is: 0.0... and position is 215498
Time taken for recurive binary seasrch is: 0.0... and position is 215498
Time taken for exponential search is: 0.0... and position is 215498
Name of the film is:  The Bruise on the Olive, and ID is: 138216

The Wife's Letter
Time taken when searching from database: 0.040002346038818360... and position is (227489, "The Wife's Letter")
Time taken for Linear search is: 0.029001712799072266... and position is 229955
Time taken for regular binary search is: 0.0... and position is 229955
Time taken for recurive binary seasrch is: 0.0... and position is 229955
Time taken for exponential search is: 0.0... and position is 229955
Name of the film is:  The Wife's Letter, and ID is: 227489

Take Off
Time taken when searching from database: 0.042002439498901370... and position is (231492, 'Take Off')
Time taken for Linear search is: 0.027001142501831055... and position is 209969
Time taken for regular binary search is: 0.0... and position is 209969
Time taken for recurive binary seasrch is: 0.0... and position is 209969
Time taken for exponential search is: 0.0... and position is 209969
Name of the film is:  Take Off, and ID is: 231492
```

From these results I decided to use a binary search, as it is overall the best for what I want it to do, (look in a list of film titles and find the one which matches the one specified). It is the fastest and the most consistent. The time complexity of the binary search is OLog(n) which is more efficient than the linear search which is O(n) meaning it looks at every single element in the list. The different is barely noticeable between exponential and binary searches, so it is better to use the binary searhc.

### Finding best way to create recommendations

I had to find a way to create customised recommendations for users and while looking into this I found the terms “content based filtering” and “collaborative based filtering”. I compared these to decide which one would be more suitable for my project. “Collaborative based filtering” is used to bring together information about similar users and suggests something to them, in this instance it would be a film. As I want to create a more personalised system, this will not be the best method to use. I will be using “content based filtering” and decided to look more deeply into it.

I read an article about “content based filtering” for a “movie recommendation” system:- https://medium.com/data-science-101/movie-recommendation-system-content-filtering-7ba425ca0920.

This discusses generic topics, including current systems and how they work, and it links to a data set with forty-five-thousand films and twenty-six-million ratings. I did not want to use data which has already been found so I gathered my own and asked the current users for their ratings. It uses an import which generates the recommendations and is just working off a module, but I want to work from the ground up, designing it and using all the information which users will actually like. It then goes on to create a website. But as this currently uses pre-rated films, I do not consider this to be truly “content based”. For this reason, I decided to look elsewhere.

I investigated further and tried finding other articles. What I found was that they were much the same sort of principal of: talking about large companies which currently do this, then using an import with currently rated films, and then saying here is a recommendation. I decided to utilise my time in a different way and to ask potential users of my system what they would want, and instead use real people as my basis for my recommendation algorithm.

## End users

My system is made for anyone who enjoys films, they will be able to use it on any device with an internet connection as it is a website, they will be able to access it anywhere as well, the website will be open 24/7 so they can access it whenever they want, but I am guessing a typical user will only use it for 5-20 minutes at a time to get a quick recommendation on what to watch. The amount of users which can use my website at any given time is dependent on the hardware which I get, if I am able to get powerful enough hardware and enough traffic, then I do not know the exact number as it all depends on the hardware.

The website will be designed in a way that anyone can use it for all ages, it will be simple enough so the less technically able people can understand what to do and will be sophisticated enough that advanced users do not think that it is amateur.

My next step is to survey 6 people who I think could potential be interested in using my website. I tried to get people from all ages, but with a small sample it was hard.

### Questionnaire which was asked

1. What does the title of "film recommendation system" suggest?

2. Do you use a current system which recommends films to the user?

3. What would a new system have to have for you to swap systems. if no: what would a system have to have for u to use it?

4. Is a chatroom or friend system a fundamental feature ?

5. What colour scheme would you want the website to be if you were designing it?

## Response from person 1

Teenage male

1. To me it suggests that the website is created to suggest films to me that I would be interested in after I view some other films

2. No I don't use any current system that recommends films to a user

3. A system would have to have a very pleasant UI for the user that is simple to use and navigate

4. A chatroom is not a fundamental feature for a film recommendation system

5. The colour scheme for would be having to be a nice light blue

## Response from person 2

Teenage male

1. The title suggests that it is a website that I can enter my preferences into, and it will then recommend a list of films I might enjoy based upon those preferences.

2. I normally watch films that are generally popular with people or within my friend group. I also make heavy use of YouTube reviewers to find new and obscure films I might not otherwise know about.

3. Ideally the system would recommend me films I may not otherwise find about, whether they are simply too niche or for some other reason.

4. A chatroom may be a good idea as it would allow people to discuss the film, however a friend feature isn't fundamental as I would likely speak to me friends in person about films regardless.

5. I like the colour blue, so likely white and blue. However, the website could utilise multiple different colours, for example an different shades of blue.

## Response from person 3

Teenage male

1. That I'd be given recommendations for films I should watch, probably based on films I've already watched or like some form which I have to fill with all the different things I like watching

2. Not particularly, I'd guess the Netflix recommendation system would be enough but otherwise I don't have a specific service which allows me to find films I might like

3. It would have to give me good results that I'm likely to watch, like if I give it a genre like animated films it should first give me animated films but also films around the idea that I might like: I'd want to have a good variety of films to watch and not just films specific to a genre

4. A chatroom or friend system wouldn't be absolutely necessary: it could be nice to have but it should probably be out of my way and not a primary feature. I'm not going on the system to make friends but to find films I'd like

5. probably something light but not very bright - a bit low-key and not very striking or painful for my eyes. A darker theme could work but I'd like to use the website at any point of day, and a dark theme would really be hard to see in the daytime.

## Response from person 4

Teenage male

1. The title suggests that the system provides recommended films for me to watch based on my current likes and dislikes.

2. I currently do not use a system.

3. I would look for a system that takes my recently watched films, recording my opinion of them. Using these, I would expect a system to recommend similar films to ones I like.

4. I think it would be an interesting tool for people to talk about current films, however I would not consider it a critical aspect.

5. I would like an interesting colour scheme to differentiate itself from current systems.

## Response from person 5

Middle aged female

1. A film recommendation system suggests that it would contain details of films that have been recommended based on my preferences from films that I have previously watched and liked.

2. I don't currently use a system that recommends films. If I want to check films that may suit what I like to watch I would use a search engine such as Google or possibly  through the Smart TV app.

3. Good recommendations of films that I would actually watch. An image and short description of the film to see if it is something that I would like to watch.

4. It is not something that I had thought about, but it is a good feature to create a small community of people who like the same kind of films who could recommend other films to each other in a centralised place.

5. If I was designing a website for film recommendations, I would like a neutral colour that would not create too much interference with the film image and descriptions. Possibly a pastel colour, a blue or green background.

## Response from person 6

Elderly female

1. It suggests putting the film titles in order based up on alphabetical order, for example if I want to find a film with a specific subheading then I can look under a certain letter such as "w" for weather.

2. no as I do not watch many films these days,

3. To be able to determine my ratings and use it for new films.

4. No, not really, I would not want to use a chatroom to talk about films or even talk to my friends in person about films before I have seen them. I would prefer to have the to have a professional's suggestion on what to watch.

5. I would want a mixture of colours, going from a light to a dark, would have like a blue but with other colours mixed together with it

## My opinion from the research carried out

From all of the people which I survey, it seems to be that the name of my website is relevant and describes to them what it does without them being baffled.

The majority of the people I interviewed do not use a system at the moment, to recommend films to them, this was surprising as I would have imagined my younger set of people interviewed would have been more interested.

All of the potential users' responses differed slightly for question 3, however they all have important features suggested. When I am designing and creating my system, I will try and make sure that the UI is simple and easy to use so that all ages can use it without confusion. I will make sure that the recommendations use the films which the user has currently liked and favourited, and will try and find films with a similar genre around them and have a wide selection of films recommended, so that the user will be able to get recommendations which they may not of been able to get elsewhere. When I am gathering all of the information for the films, I will make sure to get the image of the film as well as the description so that the user is able to see it.

All of the people who I interviewed said that a chatroom is not necessary, but some said it may be a good idea, some said they would not use it as it is somewhat out of place in a film recommendation system,  as the purpose is to find films to watch. Person 2 said they would talk to their friends in person regardless about films. And others said it would be good to get recommendations, but as I am designing an algorithm to do this, I do not think I will add this feature. As the user's are not too keen on this chatroom feature, I will not be adding this onto my website, and since the friends feature was an add-on for the chatroom that will not be added either.

For question 5 I will go into more detail about it in the next section, as all of the people interviewed suggested the same colours/mixture. I will show the images which I will be using on my website.

## Colours and images used

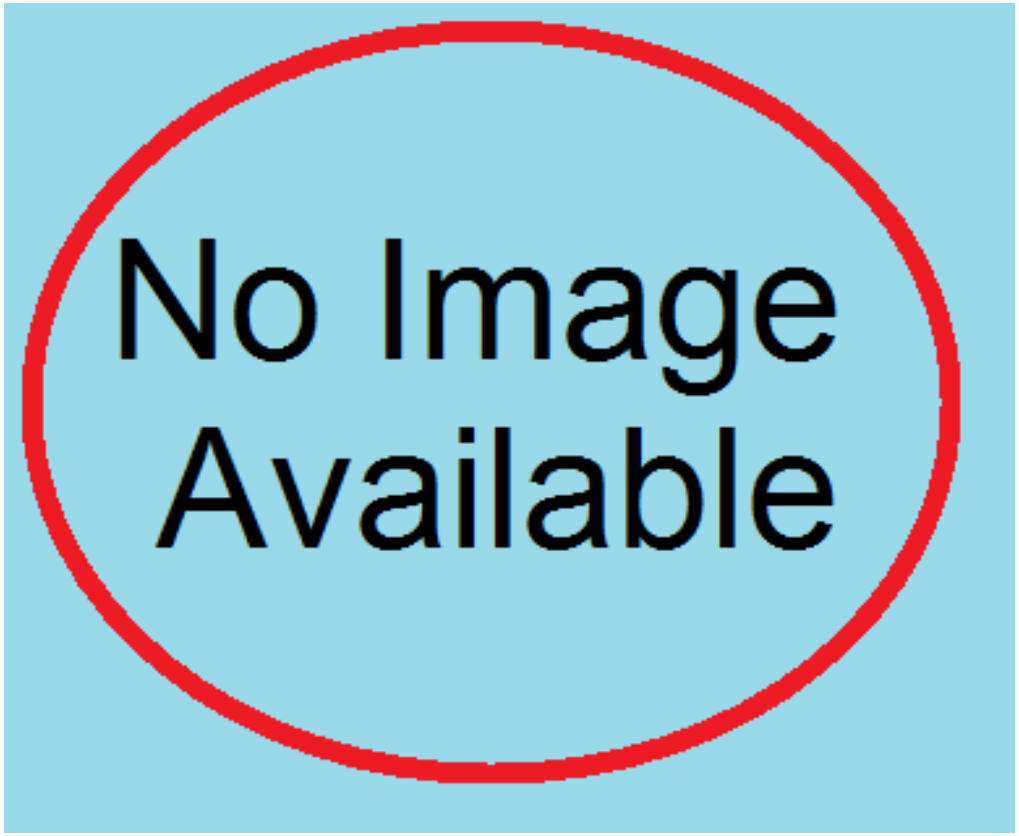
Since the potential users want something blue-ish but not too bright that it isn't blinding to them I decided on this shade of blue/green as a lot of them wanted a mixture of colours I decided to mix these too together, as person 2 suggested a dark theme and this is  nice contrast between a dark

and a light theme, as it is able to be seen at night and day no matter how bright it is where the user is viewing the website.

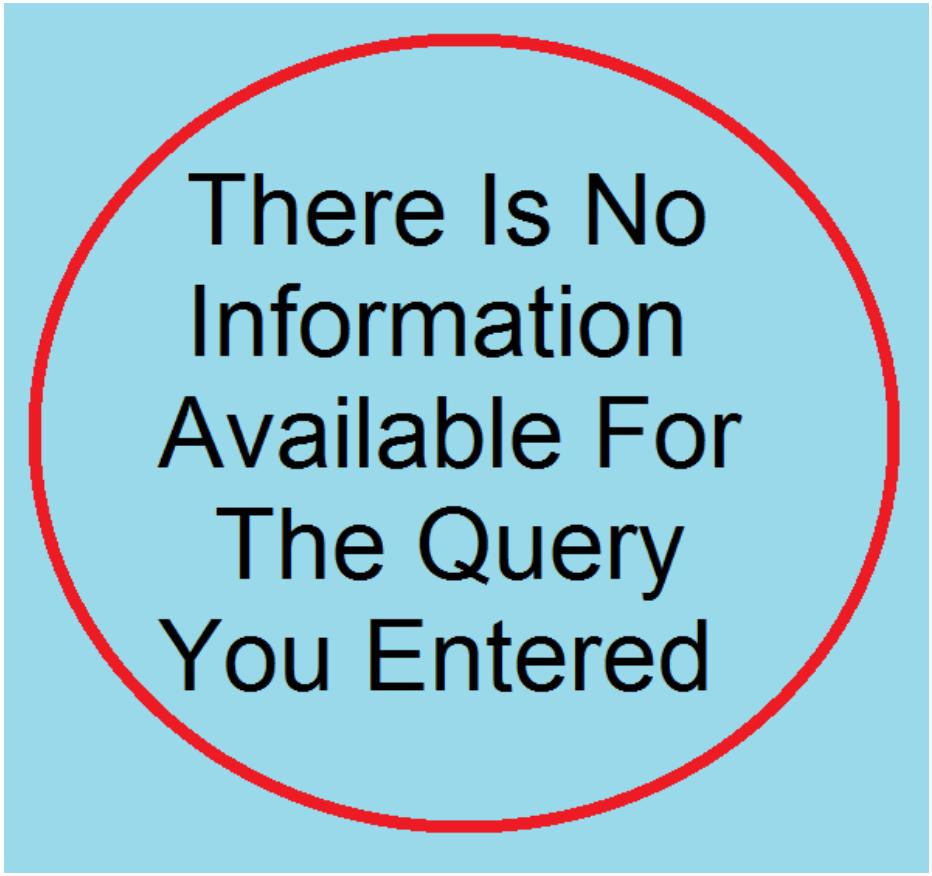
**This is the colour I decided on**

I have decided to use the theme of blue all throughout my website and have created these images.

**When there is no image to display the user will see this as a place holder**



**When the user is has entered a page but there is no information this image will appear**




**When the user is waiting for a page to load or information to appear, then they will see a gif, of this but it will be moving**

**Finally you have my original logo (seen on the title page)**

## Limitations

From my research it seems that the older generation is unlikely to use my website. For this reason, I will not gather films before 1965 when I am web-scraping for the information about films. As I am guessing that the younger generation will not be interested in watching films which were made before they were born.

I will be gathering all of the information of the films off IMDB, I will limit the films which I get, as I do not want to have a lot of unneeded data in my database. For example, not so popular films in other countries such as China and Turkey will not be included as I do not think my current potential users will be interested in those as it is an English website. But I will gather information about less popular films in English speaking countries, such as Australia and Canada.

Some films which have an alternative name may not be able to be found with the other name.

My system will not automatically update to the newest films, as I do not have a server and a machine to run it all the time to do so. For this reason, the information available will be updated until around mid-2019. In the future if I am able to get a server to run the website on 24/7 I will write an algorithm to gather new films every week or two.

A large amount of actors and directors will have unknown images, as IMDB does not have a picture for them

# Modal

This section is my modal, I will show examples of code which I was testing to make sure that the principal will work when I try and code it later on, and so that when I try and design it I can work off from this.

## Encryption

Here I tested to see if I was able to encrypt the user's password

```
from cryptography.fernet import Fernet

key = Fernet.generate_key() # randon key generated in binary 64 bits

word = input("Enter a word to encrypt: ")
print(f"This is your original word: {word}")
encrypted_word = Fernet(key).encrypt(word.encode())
print(f"This the word entered has been encrypted to: {encrypted_word}")

decrypted = Fernet(key).decrypt(encrypted_word).decode()
print(decrypted)
```

When this code was run, and I entered information this is what I got.

```
Enter a word to encrypt: banana
This is your original word: banana
This the word entered has been encrypted to: b'gAAAAABeMvmoFqtJ1RZGVtJuMzi3ftsen
LSEKVvqn39gj8wBBdMMryge5nUWc3x-ZdwBdYc4SUwaqgy1XZ7-dlKnJdS4g2geaA=='
banana
```

This worked perfectly and did exactly what I wanted, It encrypted the information which was entered so that the user cannot understand what it says, I will use this later on.

## Web scraping for titles and IMDB link

Here I tried to gather the title of the film as well as its link on IMDB, this is the code which I wrot

After the code was run the information was added to the database these are the first 10 records in the database.

| | FilmID | Title | Link |
|---|---|---|---|
| | Filter | Filter | Filter |
| 1 | 1 | A Quiet Place | /title/tt6644200/ |
| 2 | 2 | Hereditary | /title/tt7784604/ |
| 3 | 3 | Avengers: Infi... | /title/tt4154756/ |
| 4 | 4 | Solo: A Star ... | /title/tt3778644/ |
| 5 | 5 | Black Panther | /title/tt1825683/ |
| 6 | 6 | Ready Player ... | /title/tt1677720/ |
| 7 | 7 | Deadpool 2 | /title/tt5463162/ |
| 8 | 8 | BlacKkKlansman | /title/tt7349662/ |
| 9 | 9 | The Man Who... | /title/tt1318517/ |
| 10 | 10 | Jurassic Worl... | /title/tt4881806/ |

As you can see it worked perfectly. I can now use this information later on to look into the link which I was gathered for each film and find all of the specific information I want about them and add that information to my database.

### Web scraping for descriptions from the IDMB link of film

The next test which I did was to make sure that I was able to use the link to gather information about the film, so I wrote a little program to find the description of a film given the link of it.

```
# this is for the rampage film
When three different animals become infected with a dangerous pathogen, a primat
ologist and a geneticist team up to stop them from destroying Chicago.
>>>
```

It worked perfectly so I was fine to use this code, and the links to find the description of the film, I can use this link later on to find other information about the film including the actors, directors, genres and languages, ect…

### Tested mini search algorithm on mini database

With the code which I wrote before, that added the information to a testing database, I ran an SQL statement on it to make sure that it would work correctly when I write my full searching algorithm later on.

The statement which I ran `SELECT * FROM Films WHERE Title like '%page%'` is:-

When I queried my database with this statement I got these fields back in return.

| | FilmID | Title | Link |
|---|---|---|---|
| 1 | 31 | Rampage | /title/tt2231461/ |
| 2 | 1608 | Zombie Beauty Pageant: Drop Dead Gorgeous | /title/tt8016004/ |
| 3 | 3137 | The Seventh Page | /title/tt8503240/ |

These are all related to the word "page" which is the word inside of the percentage signs, this is the word which is trying to be found with SQLs fuzzy search algorithm.

This works well and gets the information which I want, I will build off of this SQL statement later on when I create my full searching algorithm.

### Tested to make sure flask displayed everything nicely

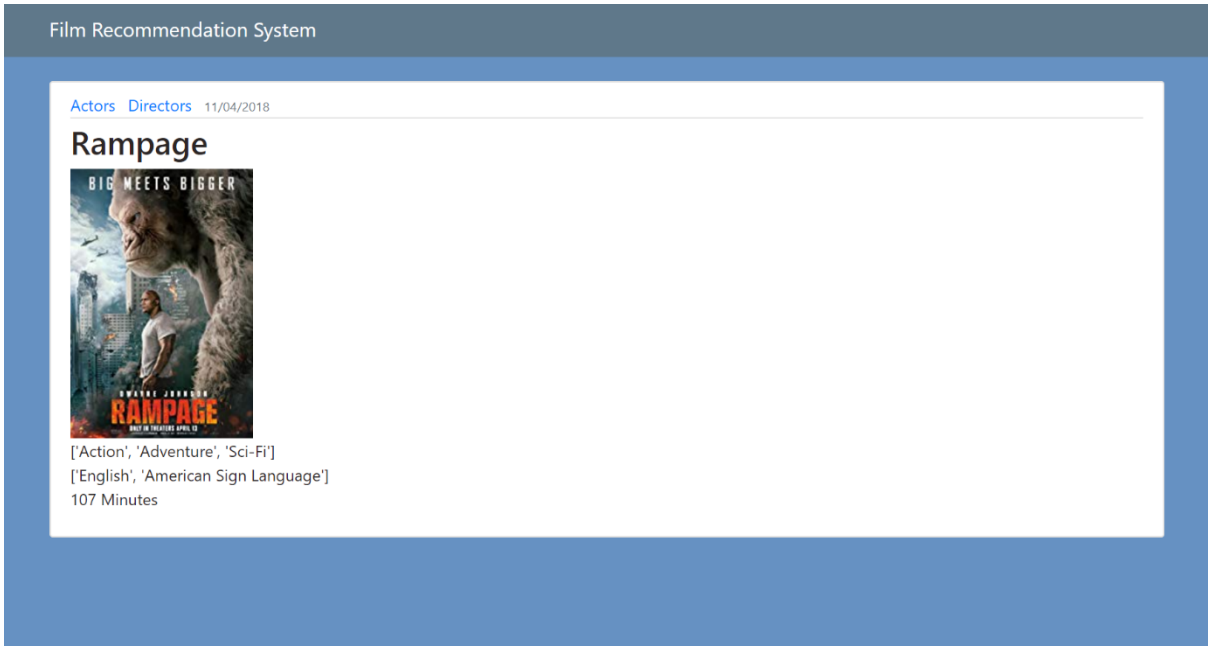
My next test was my user interface, where I designed a small website to test to make sure it would display the information in a format which I would like, later on when I improve on it and add all of the other features which I want. This was also used to test the principal of the website to make sure I was confident and comfortable using it.

**Code which I wrote for the main python file (app.py)**

It is used to store all of the information about the film which I wanted to display.

**Code for the HTML document which was used to display all of the information (home.html)**

Once I ran the website it worked as expected and displayed all of the information which I wanted.




# Design

## Modules used

| Import | How it is used and how it works |
|---|---|
| flask.flask | To initiate the instance of the app variable so that the website can access all the routes, sessions, and configurations. |
| flask.render_template | Returned from a route on the website to render a html template, redirecting the user to the page to view from the html template. Also takes in parameters so that they can be accessed in the html template using jinja2. The variables can be manipulated in the template, such as a "for loop" iterating over the data to display to the user. |
| flask.url_for | Returned from a route on the website to identify a route as a URL. These will also be used in the html templates, so that the name of the route (the function in the python file), can be converted into a URL and that the html document can read it as a directory to go to next. It will look like |

| | |
|---|---|
| | {{"home_page"}} where home_page is the name of the route in the python file, if this information is inspecting on the browser by the user then it will look like "/home-page" as it is the URL to the page, and not just the name of the function for the route. |
| flask.flash | This is stated before a flash.redirect and is used to save a message to the session which will be deleted, once that page has been accessed. The first time the page is rendered, the message will be in the user's session, but if they refresh the page it will be deleted. This is used for when the user signs up for the website and are informed that their account has been created correctly. |
| flash.redirect | Returned from a route on the website to redirect its path to another route. This will utilise the "flask.redirect" module so that the website knows which exact route to go to next, instead of just the name of a python function. |
| flask.jsonify | A route is called from a JavaScript function, inside of the route, inside of the python script, data will be computated and once the route's set of instructions has been completed it will return a jsonifed dictionary with information that can be accessed inside of the JavaScript function. The flask.jsonify module is only used when returning a response to a JavaScript function. |
| flask.session | Can be accessed in any route and is unique to the user's browser identifier and will store all the necessary information required by the website. If the user is logged in then it will store the user id, as well as the security question for the user to change their password if they wish. If the user accesses a route which has a flash message before its return statement, then it will store the message in the session. If the user logs out of the session then the session will be cleared. The sessions are stored as cookies on the user's browser, but the user cannot see the content of the cookie. |
| sqlite3.connect | Used to initiate the connection to the database, so that the python script can interact with the database by; gathering (selecting), adding (inserting), deleting, and updating data to the database. The system only interacts with one database, and all the information for the Films, Actors, Directors and Users are stored in it. |
| spellchecker.SpellChecker | Used to check the spelling of the user's input. Takes in an integer parameter for the Levenshtein distance. The default distance has been set to 2, the higher the value the more thorough the algorithm will be to find incorrectly spelt words. The distance of 1 is used when the length of the word is greater than 4, since if a distance of 2 is used on a longer word it will use a significant amount of time and drastically deteriorate the speed of my algorithm. |
| cryptography.fernet.Fernet | This is used to encrypt data by specifying a salt. A salt can be made by using the method ".generate_key()". This is a bytes object, so the method ".decode()" has to be applied to convert it to a string to be stored in the database. To encrypt data so that it cannot be understood, give it the parameter of the salt as a bytes object, to convert a string back to a bytes object use the method ".encode()" on the string.<br>The data you wish to encrypt, needs to be a bytes object, so you will need to use the ".encode()" method on it. Once it has been applied to the string you can use the ".encrypt()" method on "cryptography.fernet.Fernet", this method takes in the parameter of the |

| | |
|---|---|
| | data the user has requested to encrypt as a bytes object. This will then return the data encrypted as a bytes object.<br>To undo this process you will want to use the ".decrypt()" method on "cryptography.fernet.Fernet" giving it the encrypted data as a parameter of the encrypted data as a bytes object. It will return the original data as a bytes object so the ".decode()" method will need to be applied to convert it back to a string. |
| smtplib.SMTP | This is used to initiate the connection to the email port, allowing the system to automatically send emails to the user, when they register, change their password or request their current password. An instance of this module will be assigned to a variable, allowing an encrypted link to be established. The method of ".login()" can be applied to login with the email details which the email will be sent from, this is just the email address and the password. The first parameter is the email address, and the second one is the password. The method of ".sendmail()" is applied to send the content of the email to the user, the first parameter is the email of the sender, the second parameter is the email of the receiver and the third one is the contents of the email which is to be sent. all of the contents are stored in a text file, which will be changed depending on the user's data. |
| requests.get | Takes in a parameter of the URL of a webpage and returns the HTML code of the page. This works on any webpage on the internet since all HTML code is publically available. This was used in my gathering of all the information of the database, and is actively used to get the descriptions of the films. |
| bs4.BeautifulSoup | Once the webpage has been gathered as text using the "requests.get" module, an instance of this module is assigned to a variable, with the first parameter being the webpage, and the second being a "parser", I have used the "html.parser" parser. Once the instance has been created, methods such as ".find()" and ".find_all()" can be used to find certain pieces of data and tags, with specific names, classes and ids. Once one of the find methods have been applied it will return a string of the information which has been found, ".text" can be applied to remove all html code and just have plain text returned. |
| Imdb.IMDb | This class is used to gather information off IMDB, instead of me needing to find every HTML tag which relates to a piece of data on IMDB,<br>then having to write the code which works with this. It is also significantly faster at gathering the information instead of writing code using beautiful soup.<br>Unfortunately this module does not have every single piece of data which I need off IMDB so I will have to still write some beautiful soup code. This is used by initiating a class of itself then I can call methods on itself to gather information about a person or a film. It will be very useful when I use it to create my database. |
| currency_converter.CurrencyConverter | This module will be used when I am creating my database to convert all of the currencies which are not in USD, to USD.<br>When I web-scrape the information off IMDB some films may not be in USD, so I will need to convert them.<br>Firstly I will use it by checking if the currency which the film's budget or revenue is in, is available in the module, I will do this by checking in the ".currency" attribute on the class to see if that conversion is in there. If it |

| | |
|---|---|
| | is then I will find the first date and the last date which this conversion rate was used, by indexing the ".bound" attribute, with the currency I wish to convert from. Once I have checked all of this information and gathered the bounds, I can call the ".convert()" method to convert the currency to USD, I will give it the parameters of the upper bound, new currency and the old currency. The first date (lower bound) is not used, so that it looks at all of the currencies before that date. The upper bound is used so that it knows when to convert the currency to. |
| threading.Thread | This class allows multiple classes and threads to run in the same CPU cycle as each other, so multiple processes can happen at the same time. It takes in parameters, where target="name of the function", attr="a list of all parameters"(if there are any). On this class the method of ".start()" can be applied to start running the class or function. |
| threading.active_count | Counts the amount of "threading.Thread" currently alive. It is a function and will return an integer of the amount currently active. Does not take in a parameter. |
| json.dumps | This takes in a parameter of a list or dictionary and returns it as a string, so that it can be stored as a JSON type in a different location, in a database or a variable. |
| json.loads | Takes in a parameter of a string and converts it to a dictionary or a string, depending on its format. This is normally used in opposition to the "json.dumps" module, to convert it back to its original format. |
| json.load | This takes in a parameter of a file and returns it as a dictionary or a list depending on how it is stored in the file. The parameter which is a file will be an object and converted. |
| random.choices | It is a function and the first parameter is a list and there are 3 other parameters that are optional, but the list is compulsory. This is used in the recommendations to randomly select a certain amount of films from the list. One of the optional parameters which is k="a number" is used to state the amount of elements randomly selected from the list. |
| collections.counter | Takes in a parameter which is a list of values and returns an object of itself, which can be converted into a dictionary, and will look like '{ "cat": 10, "dog": 5, "rat": 3}'. It counts the amount of each element in the list. |
| math.ceil | Takes in a float parameter and returns the smallest number rounded down to the nearest whole number. |
| math.sqrt | Takes in a float parameter and returns the square root of the number. |
| datetime.datetime | This is used to access the calendar and time and convert strings into time values that can be understood by the python script. When the user registers, this is used to state the date and time the user registers to the website. |
| datetime.timedelta | It is a Class and is used to determine how long a user's session is active before it is deleted. The time has been set to 4 hours. So the parameter will look like "hours = 4". But it can take in a parameter for, seconds, minutes, hours, days, weeks, months and years. |
| datetime.date | This class is used in the form of "datetime.date.today()". This gets the current time and date for today, the method of ".strftime()" can be applied to it to format the date<br>to the way i wish. "strftime" takes in a singular string parameter to format the date, when the user registers for the website the format specified is "%Y-%m-%d" and will look like "2019-09-25".<br>It can also be used to find the amount of days between dates, by |

| | |
|---|---|
| | specifying the format which the strings are in. It can also reformat a date so that it uses "November" instead of 11 for the 11th month of the year. |
| time.time | When it is stated it will return the time stamp of its current time. |
| time.sleep | Takes in an integer parameter which will be in seconds. Once this has been stated it will set a delay for the amount of seconds specified. This is useful inside of "while loops" when a script is to be run periodically for certain intervals. |

## flask.session extra information

I did a test to try and store the information about the films, actors and directors in the user's session, instead of in the HTML document, however it would not of worked as the "flask.session" module has a limit on how large the session can be, storing the information in the session would of made it over 4093 bytes which is the max size of a singular session.

```
s\base_response.py:481: UserWarning: The "b'session'" cookie is too large: the value was 8948 bytes but the header required 2
6 extra bytes. The final size was 8974 bytes but the limit is 4093 bytes. Browsers may silently ignore cookies larger than this.
  samesite=samesite,
```

For example, when the user requests to try and reorder a film by a certain attribute it will need to store all of the information of all of the films on the page in the session, this will lead to the session being too large. This has led me to store the information which is required in the value of a button the HTML template, so that when the user clicks on the button to reorder the page the information is sent to the function to reorder the page.

```
<button class="btn btn-primary"
  value="{{films}}|||{{actors}}|||{{directors}}|||{{genres}}|||{{languages}}|||{{title}}|||{{results}}"
  name="order-films-data-button"> Order </button>
```

The 3 pipe symbols "|||", are used to separate the data once it has been requested from the form in the python script. I can use the '.split("|||")' method on it to identify where each specific piece of data is in the list.

## Why is Flask used?

It is a lightweight and simple web framework for python, when I was researching which framework to use, flask popped out to me, as it did not require a bunch of other modules to be used a long side it, and was self sufficient. It was fast and effective and had all the features I require to complete this project.

## Why Jinja2 is being used for template code?

Jinja2 is being used, as it is a straightforward templating language, and very similar to python code, so there is not much difference between it. It is well optimised for use with flask so there is unlikely going to be any problems when using it. I will be able to access all the variables I need inside of the HTML template by passing in the parameters into the flask function "flask.render_template". This allows me to loop through all of the information from a python list in the HTML so I can display everything to the user in a nice format.

## Why Bootstrap is being used for styling?

It has a lot of features which will come in handy when creating my website, since these features have already been created and optimized by a set of highly trained developers for the use in websites such as mine, it was more practical to use Bootstrap than to create my own styles and features. For example creating a navigation bar would of taken a lot of time and would of been an inefficient use of my time, since when undergoing a project as large as this you're very short on time.

## Why is JavaScript used?

It will be very useful, when I want to update the page without having it reload, this is the main purpose of me using JavaScript. For example, when the user requests to see the description of a film I will have a JavaScript function called on the click of that button to display the text description of that film. As well as when the user is waiting for a page to load such as when they are searching for a film I can make a loading image appear. Another way I am going to be using JavaScript is by having it receive information from the HTML template, and send the information to the database, like when the user is rating a film.

# Gathering Information For Database

These are all the steps which I need to take to get all of the information for my main database.

In this section I will show the functions and classes involved in this process, and the database structures as well, to show the evolution of my main database.

I will have 3 databases which I will work from.

1. FilmInformation – This will just be one table and store all of the information about the films.
2. PeopleInformation – This will be four tables, and will store the information about the actors and directors
3. MainDB – This is my fully normalized database with the information from the previous two organized. As well as the tables for the users, favourites, recommendations and top rated films.

## Step 1

This step is done in the file name "link_gathe.py"

The first thing which I need to do is write a python script to gather all of the titles of films and the link to them for IMDB, I will only be gathering films which were released after 1965.

I will store each year in its own JSON film, with the name of the file being the release date year. This script may take awhile to run so I will need to keep  the machine on overnight which is gathering the information.

### Imports used

from bs4 import BeautifulSoup

from requests import get

from json import dump

from threading import Thread

### Functions defined in this file

| Function Name | Parameters: Type | Returns: Type | What does it do |
|---|---|---|---|
| webpage_links_going_to_scrape_from | :param: year: int | :return: all_links_first: list<br><br>:return: all_links_second: list | Used to gather all of the page URLs which need to be web-scrape on later on |
| gahtering_data | :param: year: int | | Used to gather all of the links of the films and the title for the year specified |

## Step 2

This step is done in the file name "film_info_off_imdb.py"

Use the links which were gathered for all films, and gather the remained of the information about them off IMDB and insert it into the database. Will leave their image until later, as I may not have enough time to gather it as well.

Will use the module "imdb.IMDb" to gather most of the information about the films, as it is optimized to do so. But from my research into the module has shown that it does not have some of the information so I will need to write my own script to gather the information.

I will need to write a web scraping script which will gather the; budget, revenue and colour of the film.

### Database structure

This is the structure of what the database will look like in the beginning. It will be a database named "FilmInformation.db" and it will only have one table this will be changed later on.

FIlms(FilmID, Year, FilmName, Link, Length, Colour, ReleaseDate, Budget, GrossRevenue, Genres, Languages, Actors, Directors)

It will have all of the information about the films, the fields Genres and Languages, will be a list of all of the genres and languages which are in the film, but this will be changed later and put into a new table. Also the fields Actors and Directors will be a dictionary of all of the actors which acted in the field and all of the directors which directed. But again later on they will be in their own table instead so that data is not repeated in the database. I will also gather more information about them later on so that if the user wants to they can see more information about the actors or directors. The budget and the revenue will be put into their own table later on as well as not a lot of films have this data known.

### Imports used

```
from imdb import IMDb

from bs4 import BeautifulSoup

from sqlite3 import connect

from requests import get

from threading import Thread, active_count

from json import load, dumps
```

### Functions defined in this file

| Function Name | Parameters: Type | Returns: Type | What does it do |
|---|---|---|---|
| opens_and_gets_information_from_file | | :param: release_year: int | Used to gather all of the information from the json files, and then start of thread for each year so that the inforamtion about each film can be found |
| sends_the_information_to_the_database | :param: release_year: | | Used to send all of the information gathered |

| | | | |
|---|---|---|---|
| | int<br><br>:param: film_title: str<br><br>:param: link: str | | to the database. |
| imdb_object_creator_film | :param: link: str | :return: TYPE: imdb.IMDb | Used to reformat the link of the film so that it is the IMDB film id of the film and then it is turned into an object of the IMDb class |

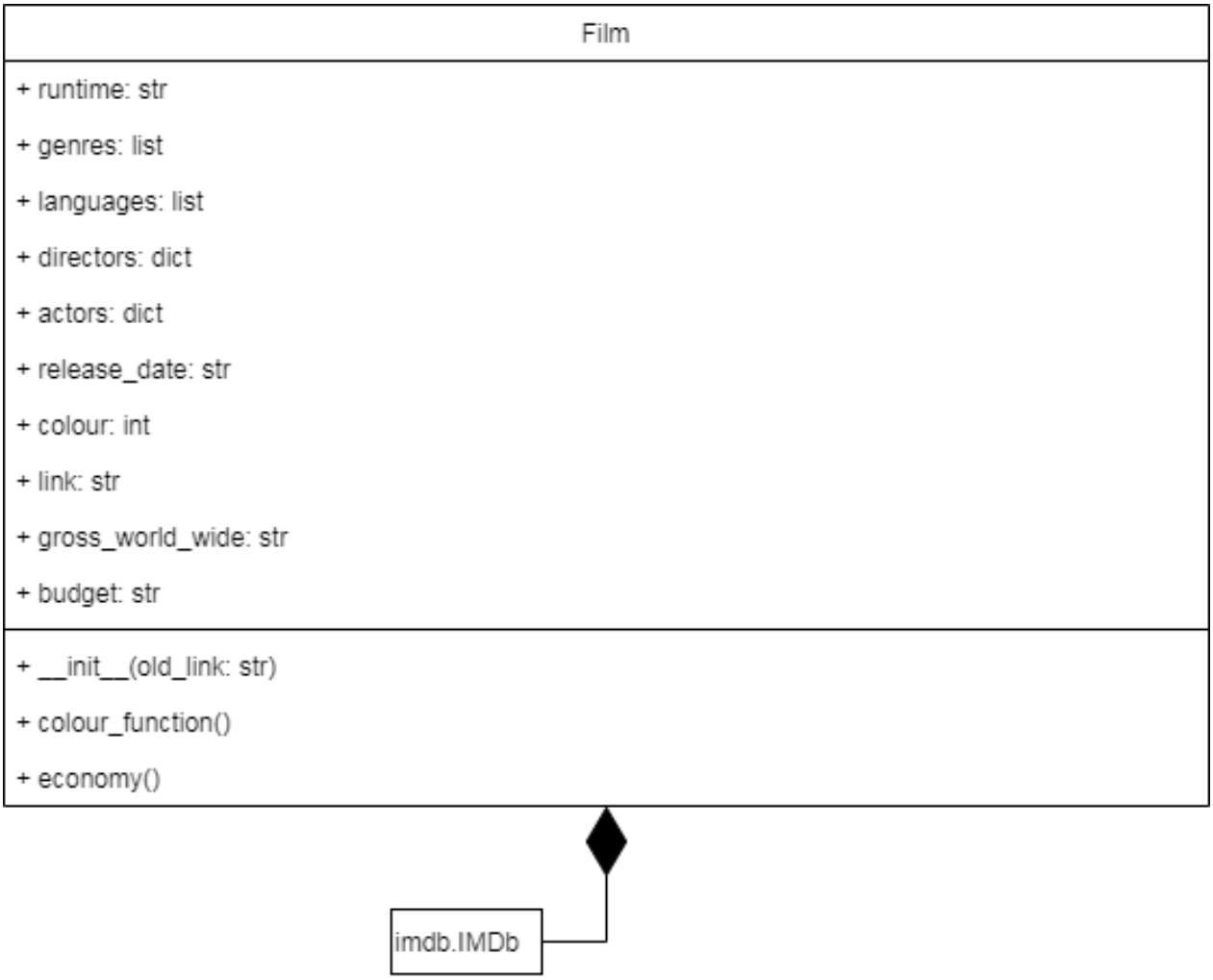
## Film Class

This file will have one class in it which is used to help organize information about the films so it can be accessed nicely later on.

This is used to house all of the information about each film so that it can be accessed in a nice format later on.

### Class diagram

The class imdb.IMDb which is from a module import is called from inside, which is assigned to a variable, this is so that the information about the films can be found using the module which was imported




## Methods for the class

These are all of the methods which are defined in the class.

| Function Name | Parameters: Type<br>Return: Type | What does it do |
|---|---|---|
| __init__ | :param: old_link: str | Main method of the class which initiates all of the attributes |
| economy | | As the IMDb module does not have the budget and revenue of the film. I will have to web scrape the data myself off the website, this method will gather the budget and the revenue of the film |
| colour_function | | As the IMDb module does not have the colour information of the film. I will have to web scrape the data myself off the website, this method will gather the colour of the film |

## Step 3

This step is done in the file name "neatens_database.py "

Now that all of the information about the films have been gathered, I will need to reformat my database. I am going to have to put all of the information into a format which I want and where the database is normalized. It will split the Genres and Languages into their own table with a link table between them. The budget and the revenue table will be done later on near the end.

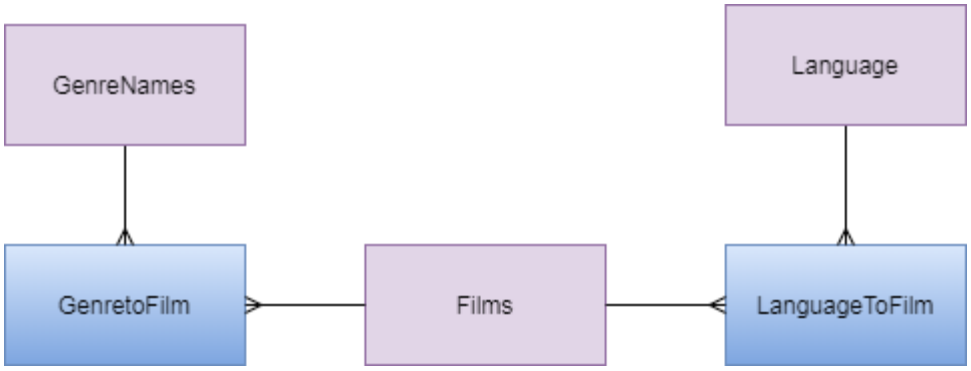
I will however need to change the way to which the link of the film is stored in the database later on if I decide to gather the image of the film, the reason for not doing this now, is that I will need to use the link to gather the image and it is simpler and easier to do it them. I will also need to bring the information about the actors and directors into one table as well.

Finally, once all of the code has finished running, I will need to change the GenreID and the LanguageID in the main GenreNames and Language table. The IDs must be reduced by one.

### Database structure

This is the updated database structure once it has been semi normalized, this is the "MainDB.db" database

The information about the actors and directors which was gathered will not yet be added to the database, as they will be stored in a separate table which I will do in the next step.




FIlms(FilmID, Year, FilmName, Link, Length, Colour, ReleaseDate, Budget, GrossRevenue)

GenreNames(GenreID, Genre)

GenreToFilms(*FilmID , GenreID*)

Language(LanguageID, Language)

LanguageToFilm(*FilmID , LanguageID*)

### Imports used

from sqlite3 import connect

from json import loads

from threading import Thread, active_count

from time import sleep

## Functions defined in this file

| Function Name | Parameters: Type | Returns: Type | What does it do |
|---|---|---|---|
| adds_information_to_database | :param: film_id: int<br><br>:param: year: int<br><br>:param: name: str<br><br>:param: link: str<br><br>:param: length: int<br><br>:param: colour: int<br><br>:param: release: str,<br><br>:param: budget: str<br><br>:param: revenue: str | | Once all of the current information in the old database has been formatted and changed, then it will add it to a new database,<br>all of the information will be in a more user friendly format for the user to read |
| genre_or_language_table_information | :param: field: str | :return: categories: list | Gets everything from the regular GenreNames/Language table |
| adds_information_to_the_linking_table | :param: film_id: int<br><br>:param: genre_or_language_id: int<br><br>:param: table: str | | Adds the information to the linking table between the film table and the language table or the genre table |
| links_film_to_genre_or_language | :param: table: str<br><br>:param: list_genres_or_languages: list<br><br>:param: film_id: int<br><br>:param: genres_or_langauges_currently_in_database: list | | Used to call the function which is used to add the information to the database about the genres and languages, which are related to a film as a thread |
| initiates_function_calls | | | This function is used to house the code which is run in this file.<br>It will call all the necessary functions, so that the |

| | | | |
|---|---|---|---|
| | | | information can be added to the database to the right table, record and field |

## Step 4

This step is done in the file name " people_information_off_imdb.py "

Once the information about the films have been gathered along with the actors and directors which relate to them. And the information about the films have been moved into the main database, I will gather the information about the actors and directors and link them to the relating film. The links of the actors and directors are in the original database which is not normalized, “FilmInformation.db”. Once the information has been gathered it is added to the “PeopleInformation.db” database. This will be added to the fully normalized database later on, “MainDB.db”. The Film IDs are stored in this table with the actor or director IDs but have no purpose as they cannot be used as there is no information about the films in this database.

### Imports used

from imdb import IMDb

from sqlite3 import connect

from json import dumps, loads

from threading import Thread, active_count

from datetime import datetime

from time import sleep

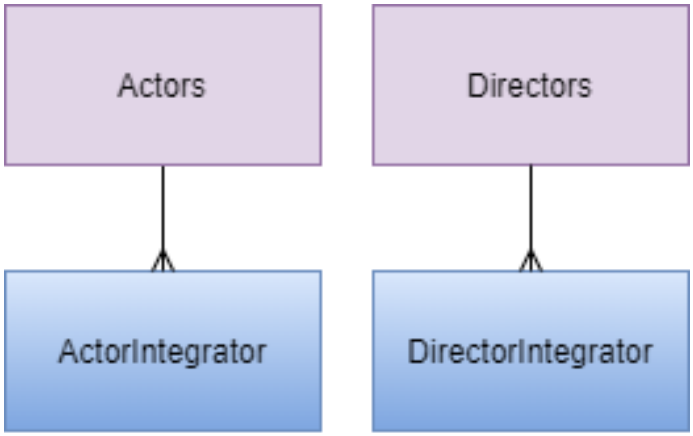
### Functions defined in this file

| Function Name | Parameters: Type | Returns: Type | What does it do |
|---|---|---|---|
| imdb_object_creator_people | :param: link: str | :return: TYPE: imdb.IMDb | Used to reformat the link of the actor or director<br>so that it is the IMDB people id<br>and then it is turned into an object of the IMDb class |
| adds_person_information_to_database | :param: link: str<br><br>:param: table: str | | Used to insert the person information into the database, if their information is not already inthere.<br>It will gather all of the information off IMDB using the imdb.IMDb module. |
| adds_person_to_linking_table | :param: film_id: int<br><br>:param: person_id: int<br><br>:param: table: str | | Used to link the data about an actor or director to a film, by adding their information to the linking integrator table |
| gather_all_people_from_datab | :param: table: str | :return: | This function gets the |

| | | | |
|---|---|---|---|
| ase | | all_people_ids: list<br><br>:return: links: list | id, name and imdb link of an actor or director from the database |
| link_actor_and_director_to_film | :param: table: str<br><br>:param: film_id: int<br><br>:param: people_related_to_film : dict | | Looks at all of the actors or directors currently in the database, checks to see if information about the actor or director which is related to the film has already been added to the database, if it has not then the information will be added to the database and linked to the film, if it is already in the database then it will be linked to the actor or director id which is already in the database. |
| no_actors_or_directors_for_film | :param: table: str<br><br>:param: film_id: int | | If the film has no known actors or directors then will add the unknown id linking it to that film to the database |
| gathers_people_information_and_links_to_film | | | This is the main file,, which calls all of the functions, it will gather all of the films which are in the database, along with their actors and directors. If the information about the actor or director is not in the database then that information is found and added, once that is done it will link the film to the actor or director. |

## Database structure

Here is a diagram of the “PeopleInformtion.db” database.




Actors(ActorID, ActorName, Link, DOB, PlaceOfBirth, DeathDate, Height, Spouse)

ActorIntegrator(*FilmID, ActorID*)

Directors(DirectorID, DirectorName, Link, DOB, PlaceOfBirth, DeathDate, Height, Spouse)

DirectorIntegrator(*FilmID , DirectorID*)

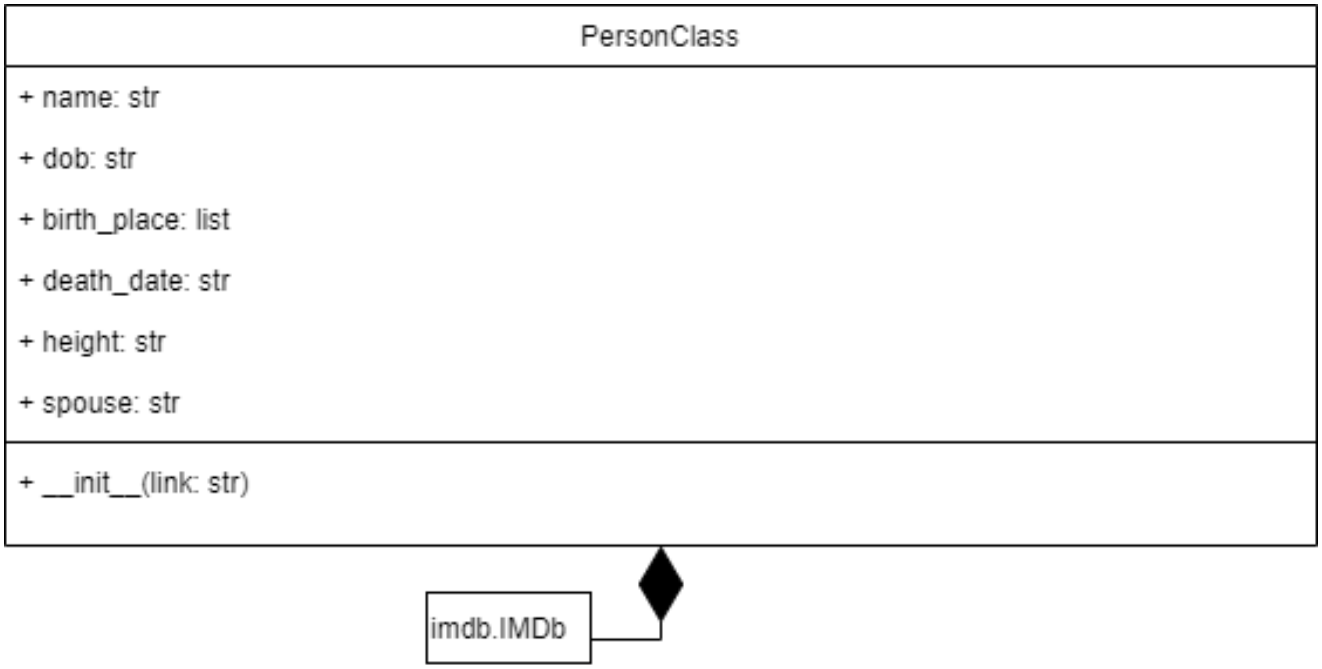
## PersonClass Class

This file will have one class in it which is used to help organize information about the actors or directors so it can be accessed nicely later on.

This is used to house all of the information about each actor or director, so that it can be accessed in a nice format later on.

### Class diagram

The class imdb.IMDb which is from a module import is called from inside, which is assigned to a variable, this is so that the information about the people can be found using the module which was imported.



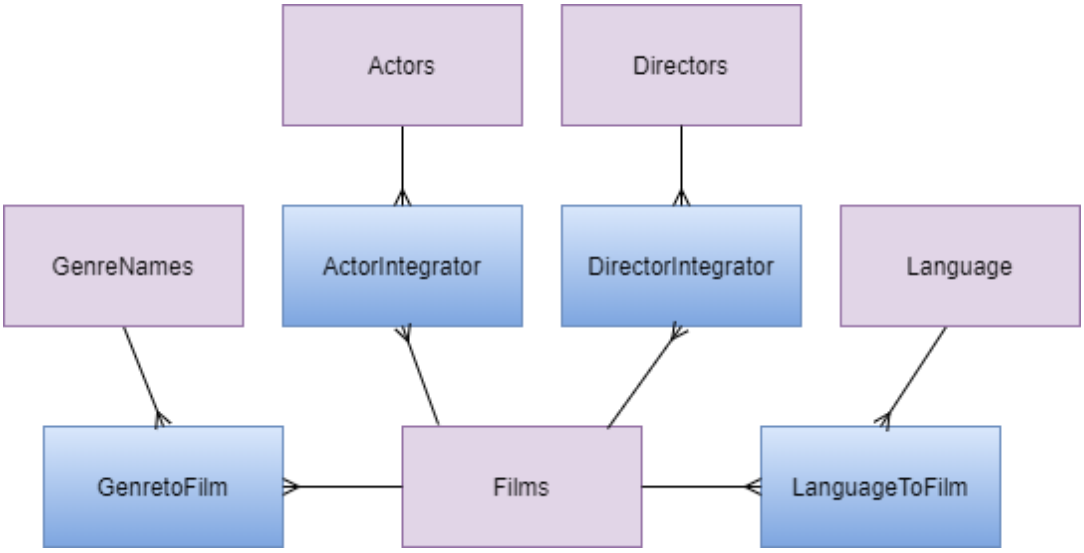
## Step 5

This step is done in the file name “moves_all_data_to_one_database.py”.

I will move all of the information from the “PeopleInformation” database to the normalized database “MainDB”, this means I am now about to use the foreign keys for the actors and directors linking to the film IDs.

### Database structure

Diagram of the new database structure of the database "MainDB.db".




FIlms(FilmID, Year, FilmName, Link, Length, Colour, ReleaseDate, Budget, GrossRevenue)

Actors(ActorID, ActorName, Link, DOB, PlaceOfBirth, DeathDate, Height, Spouse, ImageURL)

ActorIntegrator(*FilmID, ActorID*)

Directors(DirectorID, DirectorName, Link, DOB, PlaceOfBirth, DeathDate, Height, Spouse, ImageURL)

DirectorIntegrator(*FilmID , DirectorID*)

GenreNames(GenreID, Genre)

GenreToFilms(*FilmID , GenreID*)

Language(LanguageID, Language)

LanguageToFilm(*FilmID , LanguageID*)

### *Imports used*

from sqlite3 import connect

### Functions defined in this file

This file only has one function, and does not return any information

| Function Name | Parameters: Type | What does it do |
|---|---|---|
| adds_people_to_main_database | :param: table: str | Moves the information from the PeopleInformation database to the MainDB database. |

## Step 6

This step is done in the file name “images_for_films.py”.

In this stage of my database creation, I will first add a new table of "ImageURL" to the database. And then I will edit the format of the release date of the film so that it stores its date in one field not having the year in a separate one, I will also change it so that the date is in a nicer format. Once that is done I will gather the images for the films using custom coded web scraping code, using beautiful soup. Finally before adding all of this information to the database for the film I will change the format of the Link which is in the "Link" field in the "Films" table, so that if there are any repeated parts in it they will be removed. It will just store the IMDB ID in the database instead. This being said I will not store the full link of the image in the database as it has repeated parts in it again, so I will only store the parts of the link which are not repeated.

Once all this information has been added to the database I will delete the "Year" field as it is no longer needed, since this information is now stored in the "ReleaseDate" field as well.

### Database structure

This is the new structure of the Films table, it has been updated in the database "MainDB.db", some fields have been removed and added to the Films table. The rest of the database is the same structure as in "Step 5".

FIlms(FilmID, FilmName, Link, Length, Colour, ReleaseDate, Budget, GrossRevenue, ImageURL)

### Imports used

```
from bs4 import BeautifulSoup

from requests import get

from sqlite3 import connect

from threading import Thread, active_count

from time import sleep

from json import dumps
```

### Functions defined in this file

The two functions in this file, do not have return any information from them.

| Function Name | Parameters: Type | What does it do |
|---|---|---|
| gathers_films_and_calls_functions_films | | This function is used to call the function which is used to gather the images of the films and change any piece of data which still needs to be changed. It will make the function calls threads so that multiple can run at the same time to reduce the amount of time needed to run this program. |
| image_gather_and_data_formatte | :param: film_ids: int | Changes the format of the date and |

| | | |
|---|---|---|
| r_for_films | :param: link: str<br><br>:param: date: str<br><br>:param: year: int | the year, so that they are combined into one.<br>It will also change the link to imdb in the database so that it is shorter so it uses less space.<br>Once all of that is done it will find the URL for the image of the film |

## Step 7

This step is done in the file name “images_for_people.py”.

This is the same sort of process as in "Step 6" but is for actors and directors. I will gather the image of them off IMDB using custom web scraping code using beautiful soup. But I will not be editing any of the current fields in the database. I will just add a new field called "ImageURL" so I can store the link to the user's image.

### Database structure

The database structure is the same again like in "Step 5" and "Step 6" except for the Actors and Directors table having the extra field added to them "ImageURL"

#### *Imports used*

from bs4 import BeautifulSoup

from requests import get

from sqlite3 import connect

from threading import Thread, active_count

from time import sleep

### Functions defined in this file

The two functions in this file, do not have return any information from them.

| Function Name | Parameters: Type | What does it do |
|---|---|---|
| image_gatherer_for_people | :param: person_id: int<br><br>:param: person_link: str<br><br>:param: table: str | Finds and adds the image of the actor or the director to the database |
| gathers_films_and_calls_functions_people | :param: table: str | This function is used to call the function which is used to gather the images of the actors or directors<br>It will make the function calls threads so that multiple<br>can run at the same time to reduce the amount of time needed to run this program. |

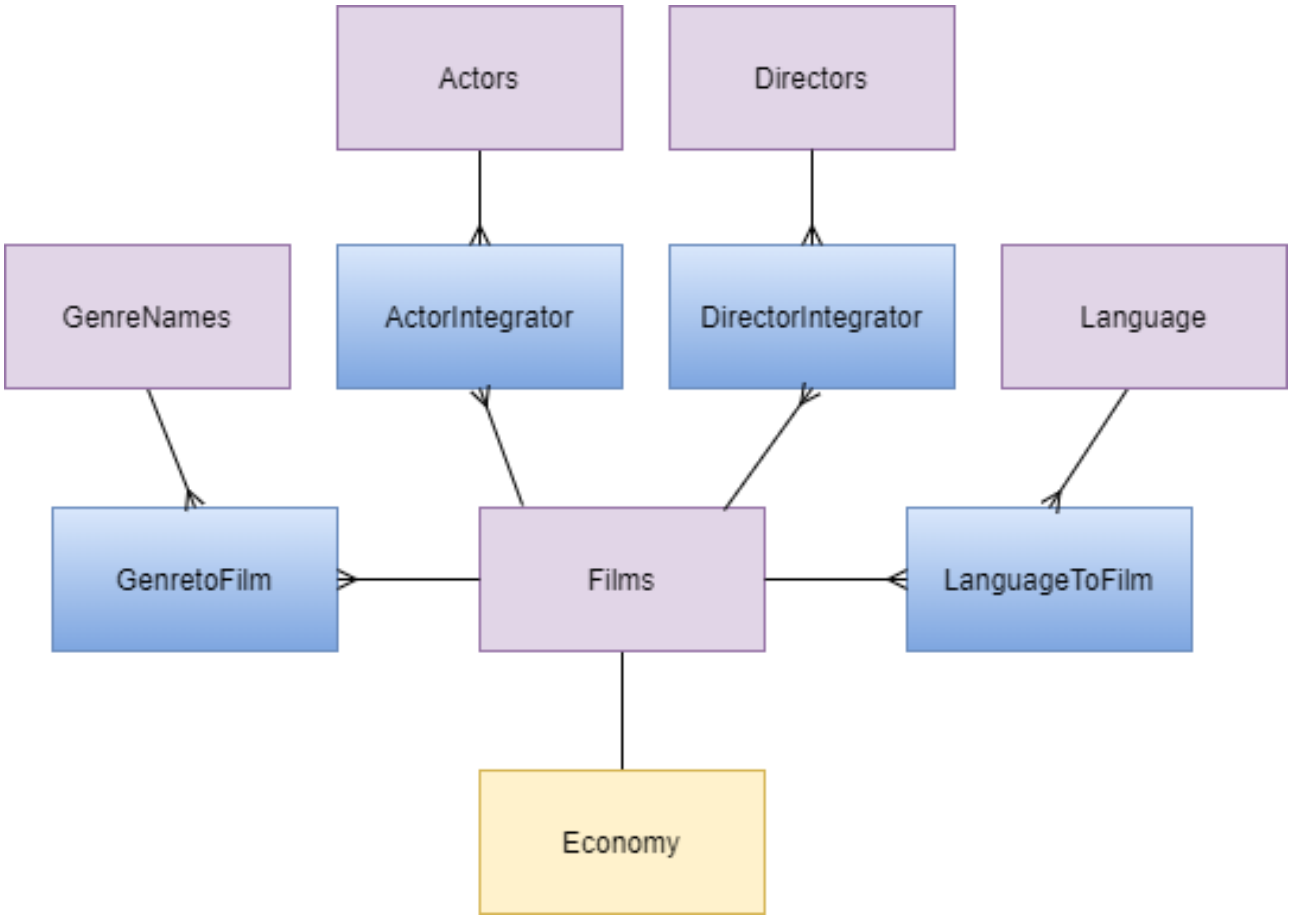
## Step 8

This step is done in the file name “changes_currency.py”.

As some of the currencies in the database are in a currency type, this file will convert them all into the same currency which is USD. Firstly I will create a new table called Economy and it will have 3 fields; "FilmID", "Budget" and "GrossRevenue". I will have to access the information about the budget and the revenue from the old "FilmInformation.db" database as the currency stored in the new normalized one will not have the currency type eg "GBP". I can then use the information from that file and convert the currency and add it to the new normalized database "MainDB.db". Once all of the currencies have been added I will delete the fields "Budget" and "GrossRevenue" from the "Films" table, as it is not needed anymore. In the new normalized database, it will not store films which have both an unknown budget and revenue in the "Economy" field as the majority of the films have this information unknown and it will increase the size of the database significantly if I am just storing unknown values.

### Database structure

Here is the diagram of the new database structure of the "MainDB.db" database, with the "Economy" field added. The fields "Budget" and "GrossRevenue" have been deleted from the "Films" table.




FIlms(FilmID, FilmName, Link, Length, Colour, ReleaseDate, ImageURL)

Economy(*FilmID*, Budget, GrossRevenue)

Actors(ActorID, ActorName, Link, DOB, PlaceOfBirth, DeathDate, Height, Spouse, ImageURL)

ActorIntegrator(*FilmID, ActorID*)

Directors(DirectorID, DirectorName, Link, DOB, PlaceOfBirth, DeathDate, Height, Spouse, ImageURL)

DirectorIntegrator(*FilmID , DirectorID*)

GenreNames(GenreID, Genre)

GenreToFilms(*FilmID , GenreID*)

Language(LanguageID, Language)

LanguageToFilm(*FilmID , LanguageID*)

### *Imports used*

```
from sqlite3 import connect

from threading import Thread, active_count

from time import sleep

from currency_converter import CurrencyConverter
```

### Functions defined in this file

The two functions in this file, do not have return any information from them.

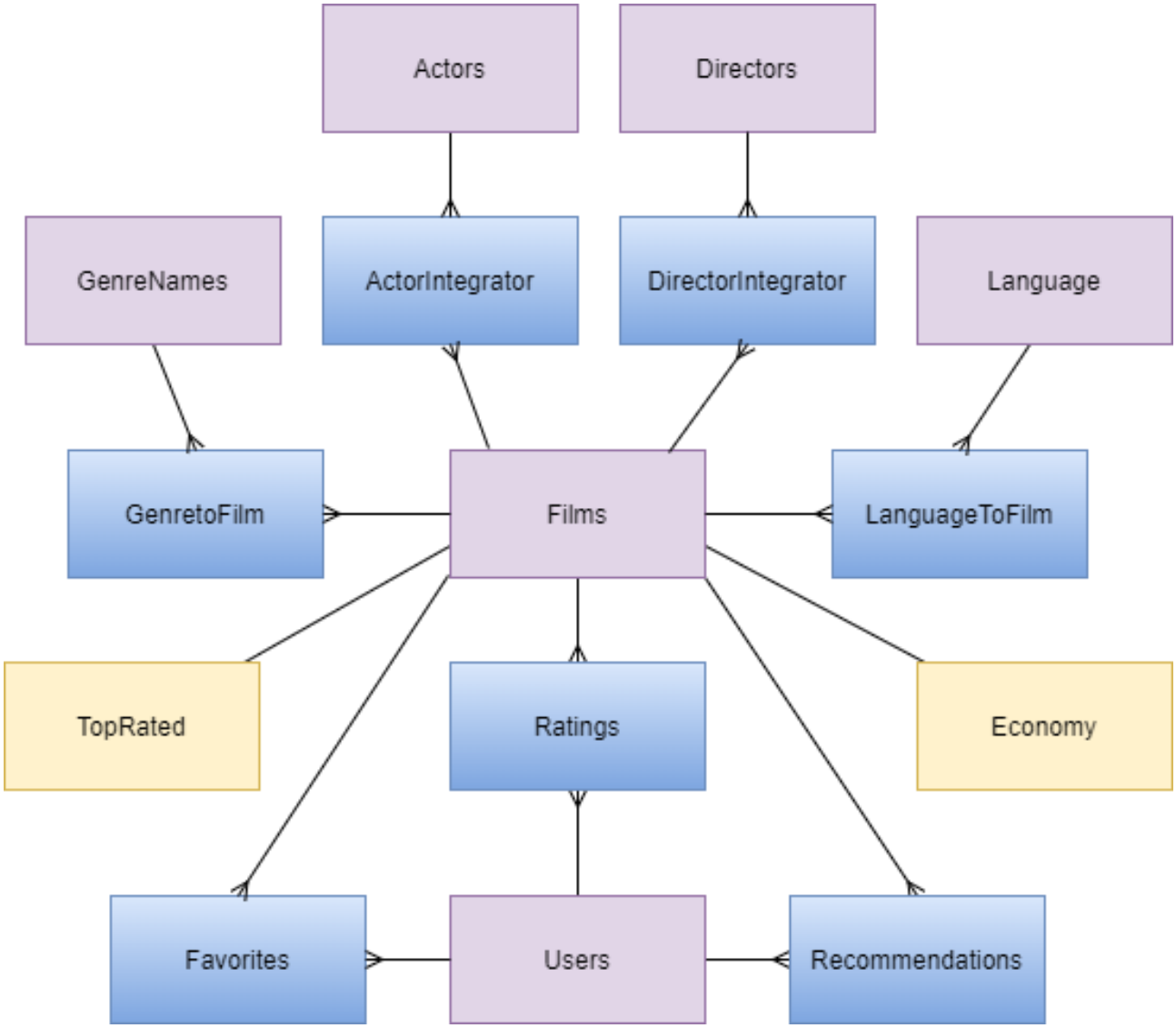
| Function Name | Parameters: Type | What does it do |
|---|---|---|
| converts_money | :param: film_id: int<br><br>:param: budget: str<br><br>:param: revenue: str | This function is used to convert all of the currencies in the database to USD, if there are any for that film<br>If there is not a budget or revenue then it is not added to the economy field.<br>If there is only one budget for one of them then the unknown one will be stored in the database as "U" |
| main_function_to_change_currencies | | The main function which controls which functions are called in this file, it will thread the functions which are called<br>so that multiple film information can be converted at once<br>It will first gather the film id, budget and revenue of the films which are in the old database. As the films here store<br>the currency type which needs they are stored in, and this will be needed to figure out what to currency to convert<br>the budget or revenue from. |

## Step 9

Once the database has had all of its information added to it I can then add the other tables to it.

Kian

Here is a diagram of my finished database




FIlms(FilmID, FilmName, Link, Length, Colour, ReleaseDate, ImageURL)

Economy(*FilmID*, Budget, GrossRevenue)

Actors(ActorID, ActorName, Link, DOB, PlaceOfBirth, DeathDate, Height, Spouse, ImageURL)

ActorIntegrator(*FilmID, ActorID*)

Directors(DirectorID, DirectorName, Link, DOB, PlaceOfBirth, DeathDate, Height, Spouse, ImageURL)

DirectorIntegrator(*FilmID , DirectorID*)

GenreNames(GenreID, Genre)

GenreToFilms(*FilmID , GenreID*)

Language(LanguageID, Language)

LanguageToFilm(*FilmID , LanguageID*)

Users(UserID, Name, Email, DOB, Username, Password, DateRegistered, SecurityQuestion1, SecurityQuestion2, SecurityQuestion3, Salt)

Favourites(*FilmID*, *UserID*, DateAdded)

Ratings(RatingID, *FilmID*, *UserID*, Overall, Comedy, Actors, Quality)

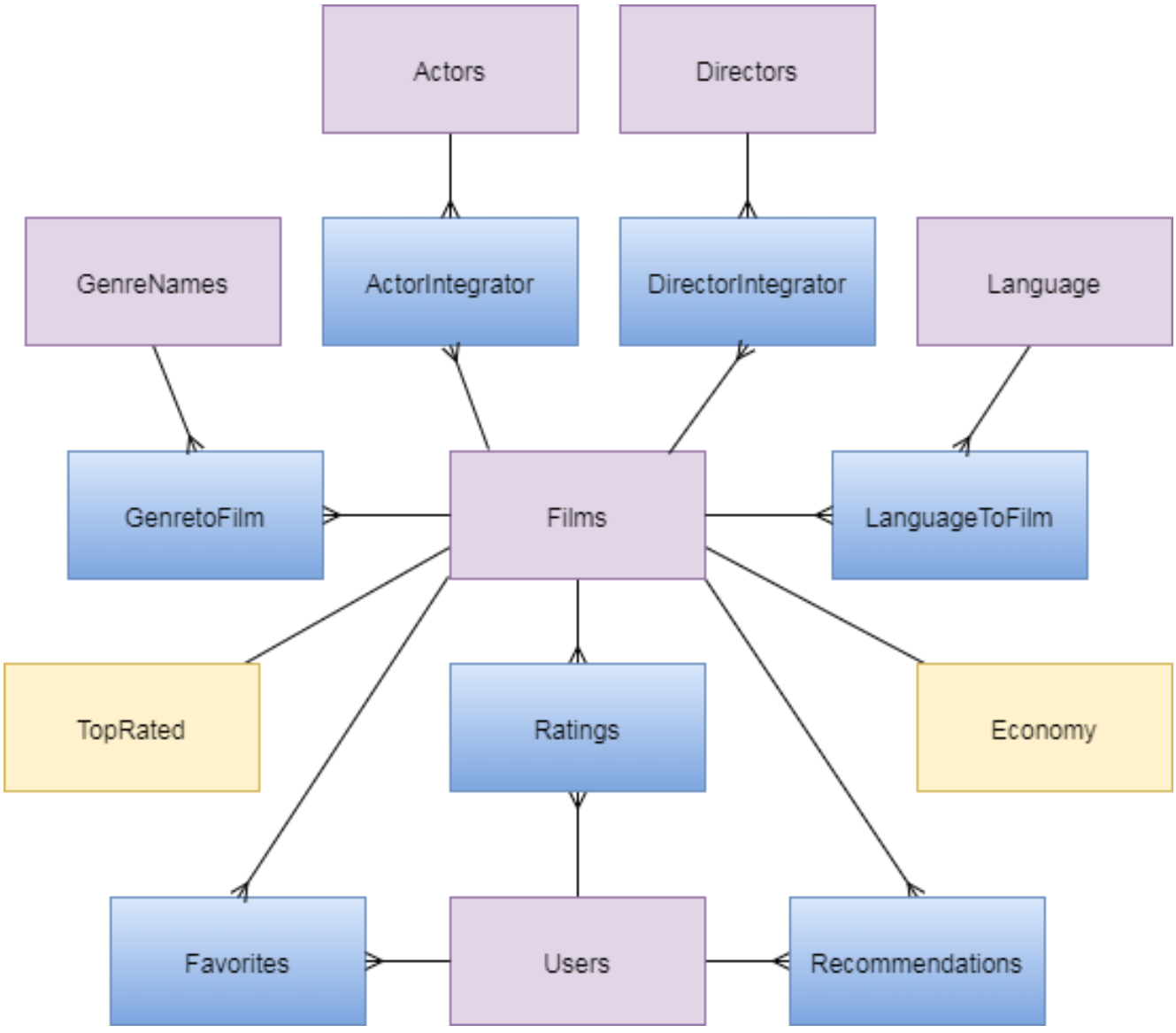
TopRated(*FilmID*, AverageRating, Category, Priority)

Recommendations(*FilmID*, *UserID*, Liked)

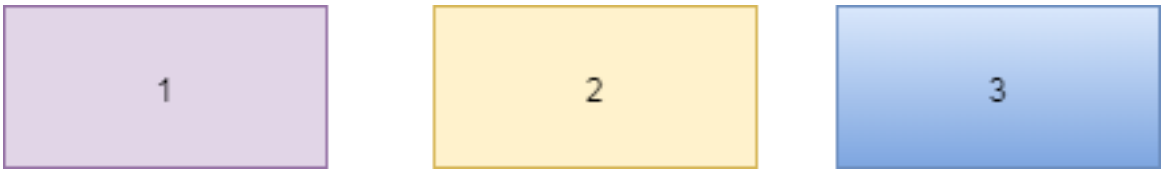
# Database

## Entity Relationship Diagram (ERD)

This is my entity relationship diagram for my database this is all the tables which will be in it (15 tables)




## Box meanings



1. Main tables, which have important information about films, users, genres, languages, actors or directors. They will be linked to other tables which are have additional information, so that everything works together nicely.

2. Extra information for the film table, these are 1 to 1 tables, as not all films will be in there but that film will only appear once in there so it is best to have a separate table for it instead of storing it in the same one, as there will be lots of unknown data in that field, as not many films have this information.

3. Used to link tables together which have important information about films, users, genres, languages, actors or directors. This is to eliminate duplicate data so that the database is a smaller size and that it is fully normalized.

## Information about each field

Unknown data in my database is defined by -1 when an integer or U when a string.

For each table I will have a table like this explaining all the fields and their special features

| Field name | Data type | Extra Information |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

**This Information applies to the Films, Actors and Directors table**

Furthermore, all the links in the ImageURL fields in my database will need the beginning of https://m.media-amazon.com/images, to work properly, I did not store this in my database because it is repeated data which is not needed.

A regular link will look like this /M/MV5BMTZhYzczY2ItNmEyZC00Yjk2LWFkOGMtYWYwNTNiZTE3YTNjXkEyXkFqcGdeQXVyMzYxOTQ3MDg@._V1_UY268_CR30,0,182,268_AL_.jpg but will have the "https://m.media-amazon.com/images" at the begging as stated before.

For each Link field in the tables, the purpose of it is so that I access the information on IMDB, it is just an integer value, which is the id of the person or the film on IMDB.

## These are all my tables and table fields

FIlms(FilmID, FilmName, Link, Length, Colour, ReleaseDate, ImageURL)

Economy(*FilmID*, Budget, GrossRevenue)

Actors(ActorID, ActorName, Link, DOB, PlaceOfBirth, DeathDate, Height, Spouse, ImageURL)

ActorIntegrator(*FilmID, ActorID*)

Directors(DirectorID, DirectorName, Link, DOB, PlaceOfBirth, DeathDate, Height, Spouse, ImageURL)

DirectorIntegrator(*FilmID , DirectorID*)

GenreNames(GenreID, Genre)

GenreToFilms(*FilmID , GenreID*)

Language(LanguageID, Language)

LanguageToFilm(*FilmID , LanguageID*)

Users(UserID, Name, Email, DOB, Username, Password, DateRegistered,
SecurityQuestion1, SecurityQuestion2, SecurityQuestion3, Salt)

Favourites(*FilmID*, *UserID*, DateAdded)

Ratings(RatingID, *FilmID*, *UserID*, Overall, Comedy, Actors, Quality)

TopRated(*FilmID*, AverageRating, Category, Priority)

Recommendations(*FilmID*, *UserID*, Liked)

## Film table

This will store most of the information about the films.

There will be around 260,000 records in this table.

| Field name | Data type | Extra Information |
|---|---|---|
| Film ID | Integer | Auto increments<br>Primary key |
| FilmName | String | |
| Link | Integer | The IMDB ID of the film |
| Length | Integer | |
| Colour | Integer | Black and White = 1<br>Colour = 2<br>Unknown = -1 |
| ReleaseDate | String | The year when the film was released, or the month as well or the exact day as well.<br>If YYYY-00-00 then the month and day is unknown<br>If YYYY-MM-00 then the exact day is unknown |
| ImageURL | String | |

**Example data for the Films table**

| FilmID | FilmName | Link | Length | Colour | ReleaseDate | ImageURL |
|---|---|---|---|---|---|---|
| Filter | Filter | Filter | Filter | Filter | Filter | Filter |
| 1 | Der letzte Mo... | 59385 | 92 | 2 | 1965-00-00 | /M/MV5BMTY... |
| 2 | The Intelligen... | 59315 | 103 | 2 | 1965-12-04 | /M/MV5BMjE5... |
| 3 | Crack in the ... | 59065 | 96 | 2 | 1965-10-11 | /M/MV5BMTY... |
| 4 | The Hill | 59274 | 123 | 1 | 1965-06-27 | /M/MV5BMTM... |
| 5 | Devils of Dark... | 59100 | 88 | 2 | 1965-09-00 | /M/MV5BNzA5... |
| 6 | Hysteria | 58216 | 85 | 1 | 1965-06-27 | /M/MV5BOTY... |
| 7 | Kuroi yuki | 223618 | 89 | 1 | 1965-06-09 | /M/MV5BNGQ... |
| 8 | Mara of the ... | 138593 | 90 | 2 | 1965-06-23 | /M/MV5BMTM... |
| 9 | The Spy with ... | 58610 | 88 | 2 | 1965-08-16 | /M/MV5BMjE4... |
| 10 | Pyramid of th... | 59621 | 102 | 2 | 1965-04-17 | /M/MV5BYjBiN... |

## Economy table

This table store the budget and the gross revenue for a film, it is a 1 to 1 table. Since not that many films have both known budgets and revenues so if it was in the Films table it will have a lot of blank data in the fields. If both are unknown then the film id will not appear in this table. However if only one of them is unknown then it will be a "-1" in the field which is unknown and the actual value in the other table.

There will be around 50,000 records in this table.

| Field name | Data type | Extra Information |
|---|---|---|
| FilmID | Integer | Foreign key linking to the Films table |
| Budget | Integer | |
| GrossRevenue | Integer | |

**Example data for the Economy table**

| FilmID | Budget | GrossRevenue |
|---|---|---|
| Filter | Filter | Filter |
| 3 | 873000 | -1 |
| 4 | 2500000 | -1 |
| 5 | 137041 | -1 |
| 11 | 60000 | -1 |
| 16 | 300000 | -1 |
| 19 | 33000 | -1 |
| 35 | 1893325 | -1 |
| 41 | 900000 | -1 |
| 43 | 354 | -1 |
| 47 | 90000 | -1 |

## Actors table

This table stores all the information about the actors.

There will be around 1,800,000 records in this table.

| Field name | Data type | Extra Information |
|---|---|---|
| ActorID | Integer | Auto increments<br>Primary key |
| ActorName | String | |
| Link | Integer | The IMDB ID of the actor |
| DOB | String | The year when the actor was born, or the month as well or both the month and the exact day as well.<br>If YYYY-00-00 then the month and day is unknown<br>If YYYY-MM-00 then the exact day is unknown |
| DeathDate | String | If YYYY-00-00 then the month and day is unknown<br>If YYYY-MM-00 then the exact day is unknown |
| PlaceOfBirth | String | This is a JSON list and will need to be loaded once the data has been received from the database as long as it is not unknown.<br>It is in ascending order by which is the nearest to the exact location, where list[-1] is the country (In python code) |
| Height | String | This is in metres as a decimal |
| Spouse | String | This is a list which will need to be loaded with JSON Stores information about their previous and current spouses and if they have any children and how many |
| ImageURL | String | |

**Example data for the Actors table**

| ActorID | ActorName | Link | DOB | PlaceOfBirth | DeathDate | Height | Spouse | ImageURL |
|---|---|---|---|---|---|---|---|---|
| Filter | Filter | Filter | Filter | Filter | Filter | Filter | Filter | Filter |
| 1 | Karin Dor | 233312 | 1938-02-22 | ["Wiesbaden, ... | 2017-11-06 | 1.65 | ["George Rob... | /M/MV5BMTc... |
| 2 | Marie France | 289828 | U | U | U | 1.63 | U | U |
| 3 | Ricardo Rodrí... | 736084 | U | U | U | U | U | U |
| 4 | Carl Lange | 486089 | 1909-10-30 | ["Flensburg, ... | 1999-06-23 | U | U | U |
| 5 | Anthony Steffen | 211922 | 1930-07-21 | ["Rome, Lazio... | 2004-06-04 | 1.9 | U | U |
| 6 | Kurt Großkurth | 344152 | 1909-05-11 | ["Langenselbo... | 1975-05-29 | U | ["Martel Gro\\... | U |
| 7 | Joachim Fuch... | 297293 | 1927-03-11 | ["Stuttgart, G... | 2014-09-11 | U | ["Gundula Kor... | /M/MV5BN2Q... |
| 8 | Daniel Martín | 554607 | 1935-05-12 | ["Cartagena, ... | 2009-09-28 | U | U | U |
| 9 | Víctor Bayo | 63175 | U | U | 2012-00-00 | U | U | U |
| 10 | Mariano Alcón | 17369 | U | U | U | U | U | U |

## ActorIntegrator table

This table links together the Film and Actors table, removing the need of redundant data which would have been stored if everything was in a single table. As an actor is highly likely to be in more than just one film.

There will be around 4,500,000 records in this table.

| Field name | Data type | Extra Information |
|---|---|---|
| FilmID | Integer | Foreign key linking to the Films table |
| ActorID | Integer | Foreign key linking to the Actors table |

If the film does not have any known actors (at least 1 known), then it will link the film to the unknown ActorID

| ActorID | ActorName | Link | Dob | PlaceOfBirth | DeathDate | Height | Spouse | ImageURL |
|---|---|---|---|---|---|---|---|---|
| 8888888888 | Filter | Filter | Filter | Filter | Filter | Filter | Filter | Filter |
| 888888888888 | UNKNOWN | 888888888888 | U | U | U | U | U | U |

This is stored in the Actors table. Unknown = 888888888888, which Is 12 sets of 8s. I did this instead of using a different value like "-1" since it is easier to spot 12 sets of 8s and than a -1 when you're looking through the database trying to find films with unknown Actors, also the reason for using such a large number is that there will never be eight hundred billion people, so I do not need to worry about it having any collisions.

**Example data for the ActorIntegrator table**

| FilmID | ActorID |
|---|---|
| Filter | Filter |
| 1 | 1 |
| 1 | 2 |
| 1 | 3 |
| 1 | 4 |
| 1 | 5 |
| 1 | 6 |
| 1 | 7 |
| 1 | 8 |
| 1 | 9 |
| 1 | 10 |

## Director table

This table is very similar to the actor information table but is just for the directors.

There will be around 120,000 records in this table.

| Field name | Data type | Extra Information |
|---|---|---|
| DirectorID | Integer | Auto increments<br>Primary key |
| DirectorName | String | |
| Link | Integer | The IMDB ID of the director |
| Dob | String | If YYYY-00-00 then the month and day is unknown<br>If YYYY-MM-00 then the exact day is unknown |
| DeathDate | String | If YYYY-00-00 then the month and day is unknown<br>If YYYY-MM-00 then the exact day is unknown |
| PlaceOfBirth | String | This is a JSON list and will need to be loaded once the data has been received from the database as long as it is not unknown.<br>It is in ascending order by which is the nearest to the exact location, where list[-1] is the country (In python code) |
| Height | String | This is in metres as a decimal |
| Spouse | String | This is a list which will need to be loaded with JSON Stores information about their previous and current spouses and if they have any children and how many |
| ImageURL | String | |

**Example data for the Directors table**

| DirectorID | DirectorName | Link | DOB | PlaceOfBirth | DeathDate | Height | Spouse | ImageURL |
|---|---|---|---|---|---|---|---|---|
| Filter | Filter | Filter | Filter | Filter | Filter | Filter | Filter | Filter |
| 1 | Harald Reinl | 718243 | 1908-07-09 | ["Bad Ischl, A... | 1986-10-09 | U | ["Daniela Deli... | U |
| 2 | Robert Asher | 38895 | 1915-00-00 | ["London, Eng... | 1979-00-00 | U | U | U |
| 3 | Andrew Marton | 554249 | 1904-01-26 | ["Budapest, A... | 1992-01-07 | 1.8 | ["Jarmila Mart... | U |
| 4 | Sidney Lumet | 1486 | 1924-06-25 | ["Philadelphia... | 2011-04-09 | 1.65 | ["Mary Gimbel... | U |
| 5 | Lance Comfort | 173806 | 1908-08-11 | ["Harrow, Lon... | 1966-08-25 | U | ["Peggy Mary ... | U |
| 6 | Freddie Francis | 5711 | 1917-12-22 | ["Islington, Lo... | 2007-03-17 | U | ["Pamela Man... | U |
| 7 | Tetsuji Takechi | 847463 | 1912-12-10 | ["Osaka, Japa... | 1988-07-26 | U | ["Hideko Kaw... | U |
| 8 | Frank McDonald | 567757 | 1899-11-09 | ["Baltimore, ... | 1980-03-08 | 1.75 | ["Goodee Mon... | U |
| 9 | John Newland | 627950 | 1917-11-23 | ["Cincinnati, ... | 2000-01-10 | U | ["Areta ?::(19... | /M/MV5BOGM... |
| 10 | Robert Siodmak | 802563 | 1900-08-08 | ["Dresden, Sa... | 1973-03-10 | U | ["Bertha Oden... | /M/MV5BZTR... |

## DirectorIntegrator table

This table is very similar to the ActorIntegrator table but is used to link the Film and Director table, removing the need of redundant data which would have been stored if everything was in a single table. As a director is highly likely to director more than just one film.
There will be around 300,000 records in this table.

| Field name | Data type | Extra Information |
|---|---|---|
| FilmID | Integer | Foreign key links to the Films table |
| DirectorID | Integer | Foreign key links to the Directors tale |

If the film does not have any known directors (at least 1 known), then it will link the film to the unknown DirectorID

| DirectorID | DirectorName | Link | Dob | PlaceOfBirth | DeathDate | Height | Spouse | ImageURL |
|---|---|---|---|---|---|---|---|---|
| 88888888 | Filter | Filter | Filter | Filter | Filter | Filter | Filter | Filter |
| 888888888888 | UNKNOWN | 888888888888 | U | U | U | U | U | U |

This is stored in the Directors table. Unknown = 888888888888, which Is 12 sets of 8s. I did this instead of using a different value like "-1" since it is easier to spot 12 sets of 8s and than a -1 when you're looking through the database trying to find films with unknown Director, also the reason for using such a large number is that there will never be eight hundred billion people, so I do not need to worry about it having any collisions.

**Example data for the DirectorIntegrator table**

| FilmID | DirectorID |
|---|---|
| Filter | Filter |
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
| 6 | 6 |
| 7 | 7 |
| 8 | 8 |
| 9 | 8 |
| 10 | 10 |

## GenreNames table

This table will store all the names of the possible films genres. The reason for having this table instead of just having one table linking to the Film table and storing the same name of the genre repeatedly, is to use up less storage space, and that my database is fully rationalised.

There will be around 30 records in this table.

| Field name | Data type | Extra Information |
|---|---|---|
| GenreID | Integer | Auto increments<br>Primary key |
| GenreNames | String | |

**Example data for the GenreNames table**

| GenreID | Genre |
|---|---|
| Filter | Filter |
| 0 | Adult |
| 1 | Unknown |
| 2 | Family |
| 3 | Adventure |
| 4 | Mystery |
| 5 | Documentary |
| 6 | Sport |
| 7 | Game-Show |
| 8 | War |
| 9 | News |

## GenreToFilm Table

This table will allow the database to link all the names of the genres to the corresponding film, without having to store all the genres for the films as a list inside of the Film table, or in a separate table with the genre name repeated on different lines linking to a different FilmID.

There will be around 430,000 records in this table.

| Field name | Data type | Extra Information |
|---|---|---|
| FilmID | Integer | Foreign key linking to the Films table |
| GenreID | String | Foreign key linking to the GenreNames table |

A film with an unknown genre, does not have at least 1 known genre, will link to the GenreID 1

| GenreID | Genre |
|---|---|
| Filter | Unknown |
| 1 | Unknown |

**Example data for the GenreToFilm table**

| FilmID | GenreID |
|---|---|
| Filter | Filter |
| 1 | 26 |
| 2 | 24 |
| 2 | 4 |
| 3 | 10 |
| 4 | 23 |
| 4 | 8 |
| 5 | 25 |
| 6 | 19 |
| 6 | 4 |
| 7 | 11 |

## Language table

This table is like the GenreNames table, the difference is that it stores the information about the languages, it will store all the possible languages for any film. The reason for having this table instead of just having one table linking to the Film table and storing the same language repeatedly, is to use up less storage space, and that my database is fully rationalised.

There will be around 300 records in this table.

| Field name | Data type | Extra Information |
|---|---|---|
| GenreID | Integer | Auto increments<br>Primary key |
| GenreNames | String | |

**Example data for the Language table**

| LanguageID | Language |
|---|---|
| Filter | Filter |
| 0 | North Americ... |
| 1 | Dutch |
| 2 | Syriac |
| 3 | Shanxi |
| 4 | Georgian |
| 5 | Khasi |
| 6 | Awadhi |
| 7 | Icelandic Sign... |
| 8 | Hokkien |
| 9 | Bosnian |

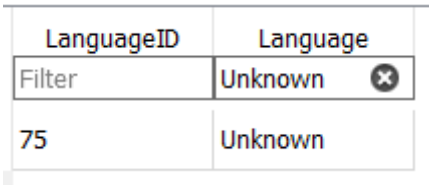
## LanguageToFilms table

This table is like the GenresToFilms, the difference is that it stores the links between the Films table and the Language table for the corresponding film, without having to store all the different languages for the films as a list inside of the Film table, or in a separate table with the language repeated on different lines linking to a different FilmID.

There will be around 300,000 records in this table

| Field name | Data type | Extra Information |
|---|---|---|
| FilmID | Integer | Foreign key linking to the Films table |
| LanguageID | String | Foreign key linking to the Language table |

A film with unknown languages, does not have at least 1 known language, will link to the LanguageID 75.



| LanguageID | Language |
|---|---|
| Filter | Unknown |
| 75 | Unknown |

**Example data for the LanguageToFilms table**

| FilmID | LanguageID |
|---|---|
| Filter | Filter |
| 1 | 153 |
| 2 | 149 |
| 3 | 149 |
| 4 | 149 |
| 5 | 149 |
| 6 | 201 |
| 6 | 149 |
| 7 | 200 |
| 7 | 149 |
| 8 | 149 |
| 9 | 149 |
| 10 | 153 |

## Users table

This table will store all of the important information about the user.

It can have any number of records depending on how many users sign up to the website.

| Field name | Data type | Extra Information |
|---|---|---|
| UserID | Integer | Auto increments<br>Primary key |
| Name | String | |
| Email | String | - I need to store this information because I will be sending them update information to them directly informing them about important news about their account or if a sequel to the film which is in their favourites is coming out.<br>- I will also need their email address to help them do account recovery if they forget their password |
| DOB | String | - This data will be used in the future to help the recommendation algorithm predict what type of films the user will like. This can help tailor the recommendation to the user better. |
| Username | String | Needed so that the user can login |
| Password | String | Stores the user's password encrypted using the salt which is stored in the same table. If an unauthorized person was able to get hold of the salt they will not know how to apply it to get the encrypted password to get the original one. Since they do not know the algorithm so it is safe storing them in the same place. |
| DateRegistered | String | |
| SecurityQuestion1 | String | This is a JSON list and will need to be loaded once the data has been received from the database. The first index of the list will be the id of the question and the second index will be the encrypted answer which the user has entered. |
| SecurityQuestion2 | String | Same principal as the SecurityQuestion1 |
| SecurityQuestion3 | String | Same principal as the SecurityQuestion1 |
| Salt | String | The salt used to encrypt the password and the security question in the database for the user. |

**Example data for the Users table**

## Favourites table

This table links the Users table to the Films table allowing the user to favourite a films and have access to all the information about it.

It can have any number of records depending on how many films each user rates and how many users are signed up to the website.

| Field name | Data type | Extra Information |
|---|---|---|
| FilmID | Integer | Foreign key linking to the Films table |
| UserID | Integer | Foreign key linking to the Users table |
| DateAdded | String | |

**Example data for the Favorites table**

| FilmID | UserID | DateAdded |
|---|---|---|
| Filter | Filter | Filter |
| 134618 | 1 | 2020-01-11 |
| 210072 | 1 | 2020-01-11 |
| 231105 | 1 | 2020-01-11 |
| 175129 | 1 | 2020-01-11 |
| 160605 | 1 | 2020-01-11 |
| 121992 | 1 | 2020-01-11 |
| 143729 | 1 | 2020-01-11 |
| 130292 | 1 | 2020-01-11 |
| 235669 | 1 | 2020-01-11 |
| 154365 | 1 | 2020-01-11 |

## Ratings table

This table links the Users table to the Films table allowing the user to rate a films, and when they are looking at their favourites or searching for a film they will be able to see if that film has been rated or not. The ratings are also used to create the recommendations for the user, as well as creating the Overall and Comedy leaderboard.

It can have any number of records depending on how many films each user rates and how many users are signed up to the website.

| Field name | Data type | Extra Information |
|---|---|---|
| RatingID | Integer | Auto increments<br>Primary key<br>Is not actually used for anything important |
| FilmID | Integer | Foreign key linking to the Films table |
| UserID | Integer | Foreign key linking to the Users table |
| Overall | Integer | |
| Comedy | Integer | |
| Actors | Integer | |
| Directors | Integer | |

**Example data for the Ratings table**

| RatingID | FilmID | UserID | Overall | Comedy | Actors | Quality |
|---|---|---|---|---|---|---|
| Filter | Filter | Filter | Filter | Filter | Filter | Filter |
| 41 | 134618 | 1 | 6 | 6 | 5 | 7 |
| 42 | 210072 | 1 | -1 | -1 | 9 | -1 |
| 43 | 160605 | 1 | 8 | -1 | 8 | 9 |
| 44 | 235669 | 1 | -1 | 10 | -1 | -1 |
| 45 | 167507 | 1 | -1 | 10 | -1 | -1 |

## TopRated table

Information calculated and formulated from the leaderboard algorithm is added to this table, for the films with the highest average ratings top 100 in each category of Overall and Comedy.

It can will have between 0 and 200 records.

| Field name | Data type | Extra Information |
|---|---|---|
| FilmID | Integer | Foreign key linking to the Films table |
| AverageRating | String | A sum of all of the ratings for that FilmID divided by the amount of ratings it has |
| Chategory | Integer | 1 = Overall<br>2 = Comedy |
| Priority | Integer | 2 = above 2 standard deviations of mean rating for films<br>1 = above 1 standard deviations of mean rating for films<br>0 = above the mean rating for films |

**Example data for the TopRated table**

| FilmID | AverageRating | Chategory | Priority |
|---|---|---|---|
| Filter | Filter | Filter | Filter |
| 148956 | 10.0 | 1 | 2 |
| 178007 | 10.0 | 1 | 2 |
| 148934 | 10.0 | 1 | 2 |
| 209996 | 9.0 | 1 | 1 |
| 167694 | 9.0 | 1 | 1 |
| 209996 | 9.0 | 1 | 0 |
| 167694 | 9.0 | 1 | 0 |
| 148956 | 10.0 | 2 | 2 |
| 194713 | 10.0 | 2 | 2 |
| 235669 | 10.0 | 2 | 2 |

## Recommendations table

This table links the Users table to the Films table allowing the database to store the recommendations for the users. The information is added from the recommendation algorithm, each user will have between 0 and 30 recommendations, however the algorithm tries to get 30 recommendations

It can have any number of records depending on how many users request recommendations and how many users are signed up to the website.

| Field name | Data type | Extra Information |
|---|---|---|
| FilmID | Integer | Foreign key linking to the Films table |
| UserID | Integer | Foreign key linking to the Users table |
| Liked | Integer | These are the weightings for the films, the ones with the higher weighting will appear first when the user clicks on their recommendations.<br>0 = If they do not have a specific weighting (The majority of the values)<br>7 = Monochrome films<br>8 = Third highest weighting<br>9 = Second highest weighting<br>10 = Highest weighting |

**Example data for the Recommendations table**

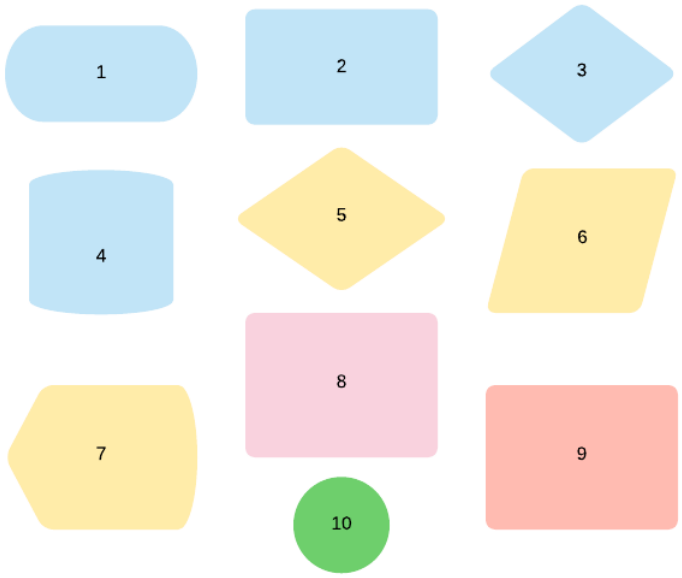
| FilmID | UserID | Liked |
|---|---|---|
| Filter | Filter | Filter |
| 172128 | 1 | 0 |
| 203898 | 1 | 0 |
| 256199 | 1 | 0 |
| 235760 | 1 | 0 |
| 122497 | 1 | 9 |
| 122507 | 1 | 9 |
| 126095 | 1 | 9 |
| 128147 | 1 | 9 |
| 191634 | 1 | 9 |
| 144023 | 1 | 9 |

# Flowcharts

## Box colours and meaning

**Light blue boxes are processes done on the backend (1, 2, 3, 4)**

**Light yellow boxes are things which interact with the user (5, 6, 7)**



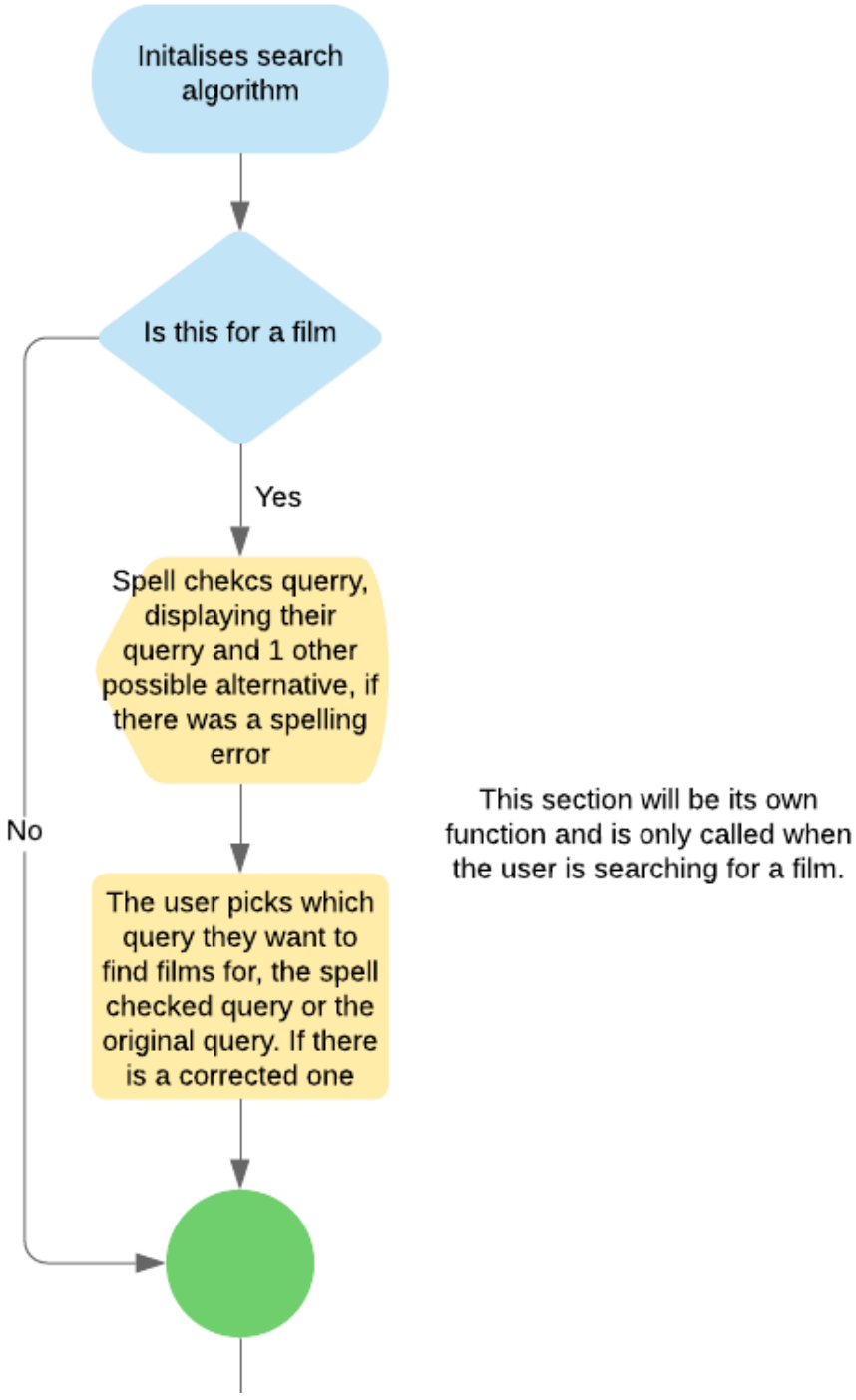

1. Starts and terminates the algorithm, does not interact with user.
2. Processes a task on the backend, this does calculations, as well as instructing the algorithm what to do.
3. The system makes a decision on the backend which will determine the path the algorithm will take.
4. Accesses the database and interacts with it, gathering or sending data to it.
5. The user can make a decision, may have 1 possible output or multiple. If 1 possible output then the option is not required, and the user did not enter anything.
6. The user can input data.
7. Information is displayed to the user.
8. The process box is used with this pinkish colour in the recommendation flowchart, to suggest it is moving to the next major section of code.
9. This process box with a light reddish colour is used in the recommendation flowchart, to suggest that the information inside of it will be updated at the end of the algorithm.
10. A connector, so that when numerous sections of the algorithm join together they can be joined in a neat easy to read manner.

## Searching algorithm

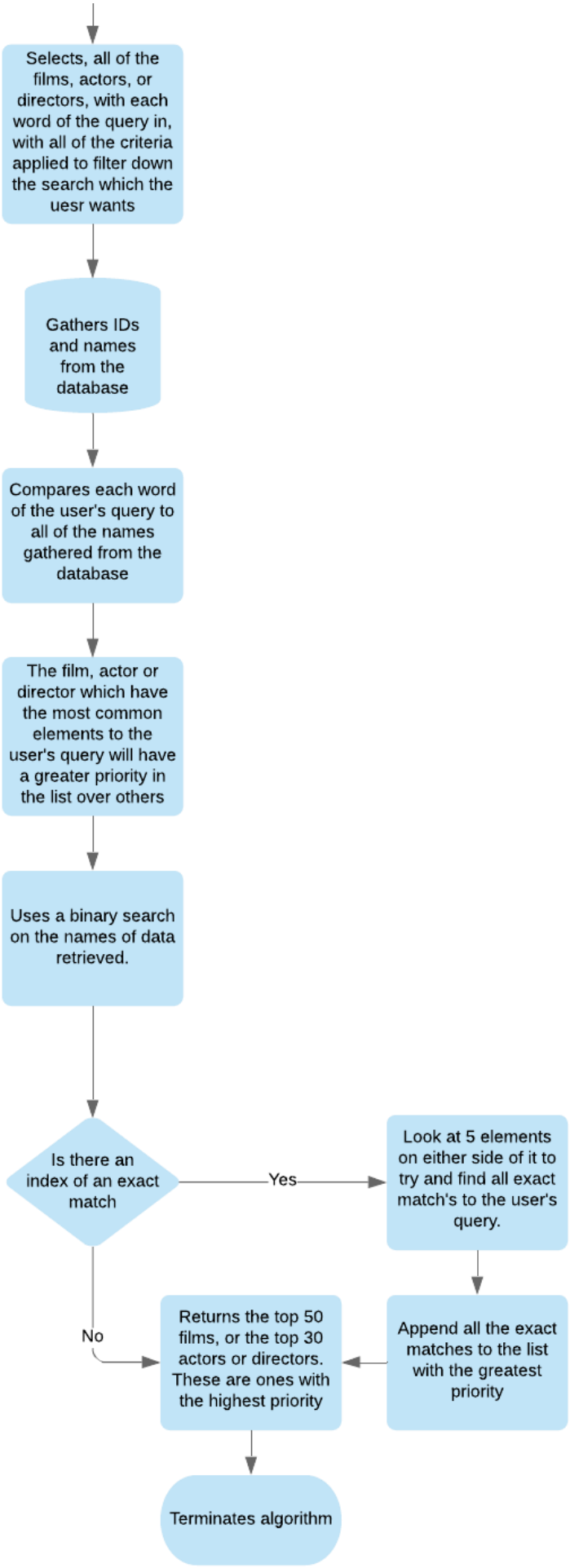
This is the algorithm which is run when the user searches for either an actor, director or film.

If the user searches for a film, then the user's query will be spell checked, identified in the image below. Once it is spell checked it will return the corrected queries if there are any and ask the user which one they want, this will not happen if it is an actor or director.

Once the user decides which query they want to find films for the rest of the algorithm is run.



Kian


Selects, all of the films, actors, or directors, with each word of the query in, with all of the criteria applied to filter down the search which the uesr wants
Gathers IDs and names from the database
Compares each word of the user's query to all of the names gathered from the database
The film, actor or director which have the most common elements to the user's query will have a greater priority in the list over others
Uses a binary search on the names of data retrieved.
Is there an index of an exact match
Yes
Look at 5 elements on either side of it to try and find all exact match's to the user's query.
No
Returns the top 50 films, or the top 30 actors or directors. These are ones with the highest priority
Append all the exact matches to the list with the greatest priority
Terminates algorithm

## Searching for films, actors and directors page

This algorithm has 2 entrance points, 1 if the user is searching for a film and another if they are searching for an actor or a director.

They are all in the same algorithm, as it uses the same principals to find the match what the user is looking for.

When the user is searching for a film it will display all the information about the film including (will only be displayed if the information is known);
- Release date
- Colour type (black and white or just colour)
- Genres
- Languages
- Actors
- Directors

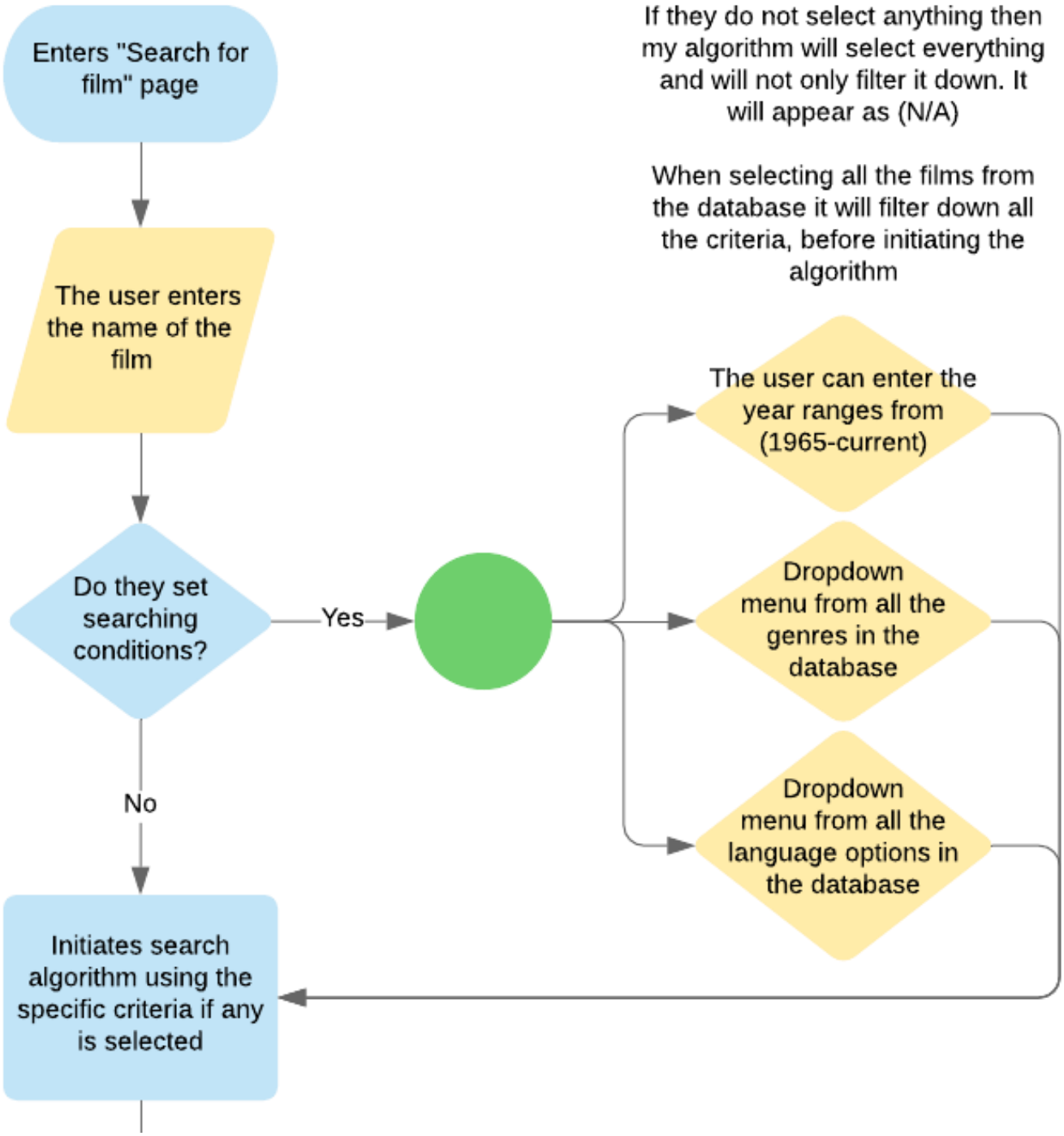
Also the ability to;
- View description of the film
- Rate
- Favourite
- Unfavourite (if favourited)

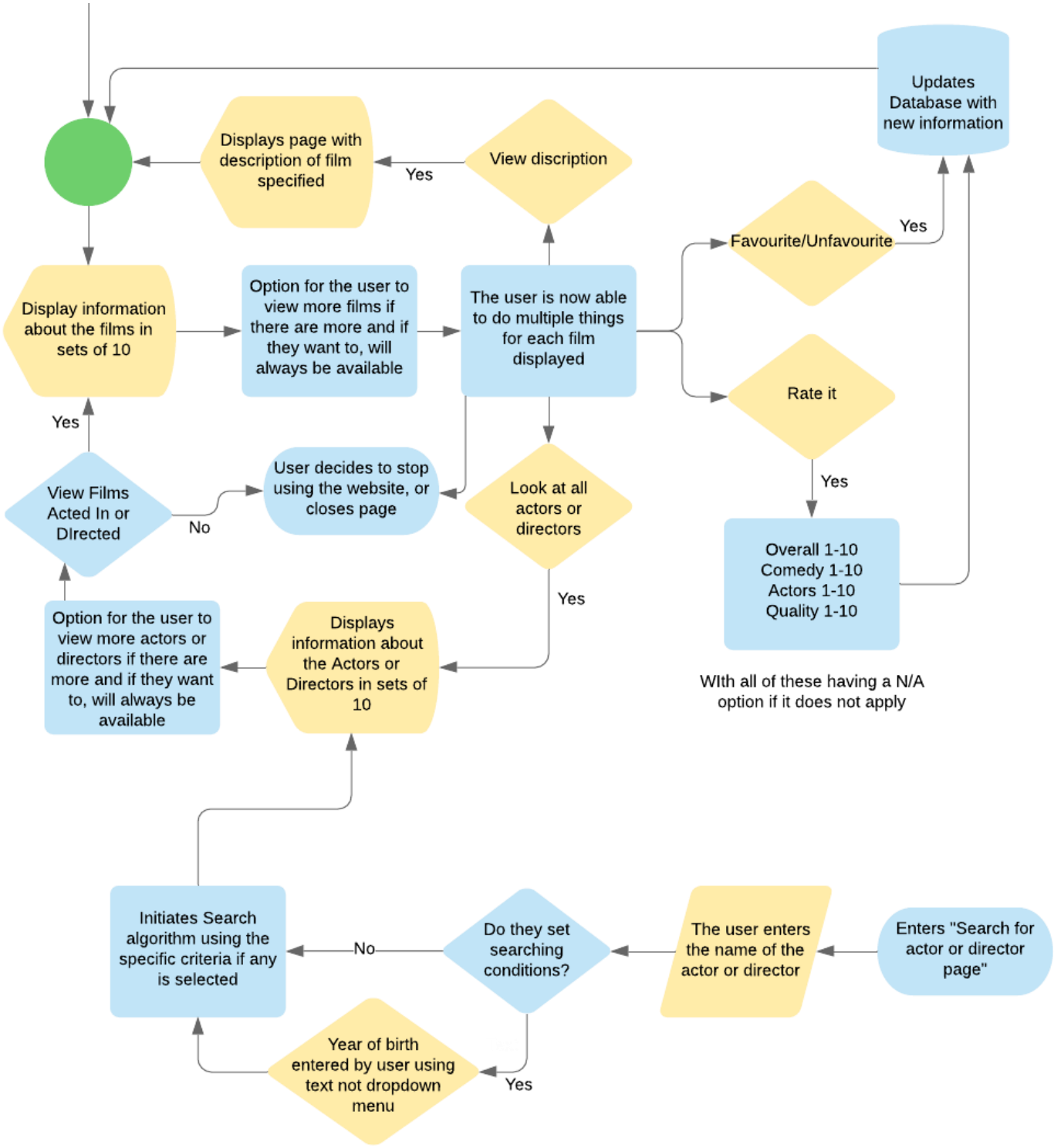
The user is able to click on the actors and directors and a modal will be displayed with all of them, and the user can click on which one they want to see more information about

When the user is searching for an actor or director then it runs the same algorithm with the same filters it is just looking in a different section of the database. The information which will be displayed once the algorithm has found all of the possible matches are (will only be displayed if the information is known);
- Name
- Date of birth
- Death date (if applies)
- Spouses and children
- Location of birth
- Films in or directed

If the user clicks to see the film in or directed then it will display all of the films which are associated with that person.

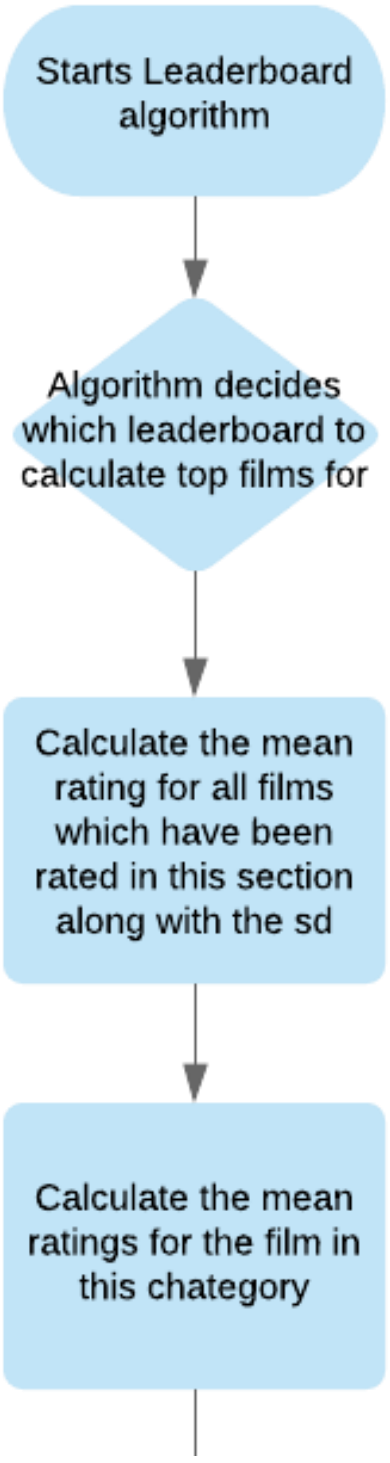
The search algorithm has more details in how the information is regulated to display the correct data.

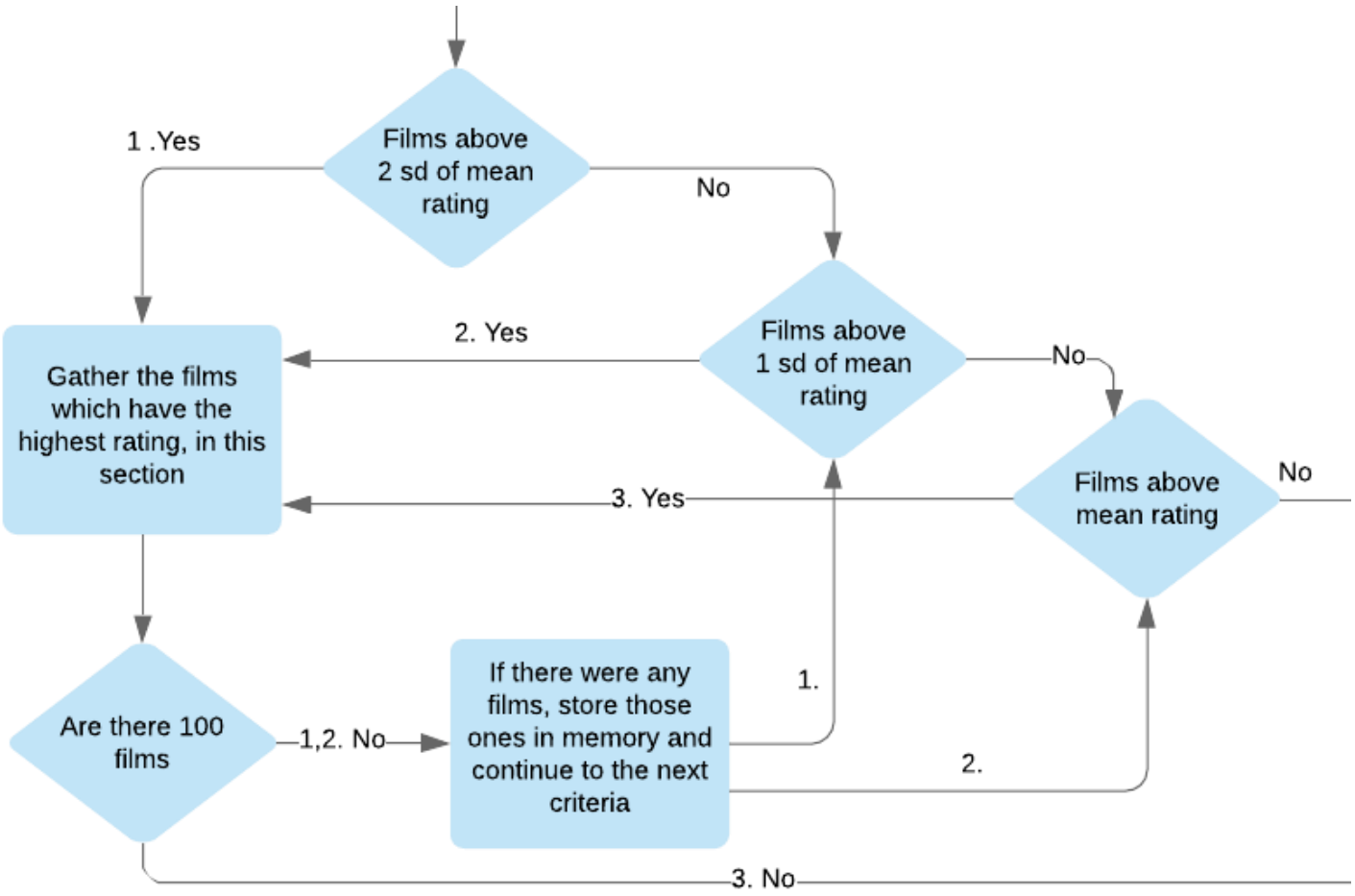

Enters "Search for film" page
The user enters the name of the film
Do they set searching conditions?
Yes
No
Initiates search algorithm using the specific criteria if any is selected
The user can enter the year ranges from (1965-current)
Dropdown menu from all the genres in the database
Dropdown menu from all the language options in the database
If they do not select anything then my algorithm will select everything and will not only filter it down. It will appear as (N/A)
When selecting all the films from the database it will filter down all the criteria, before initiating the algorithm

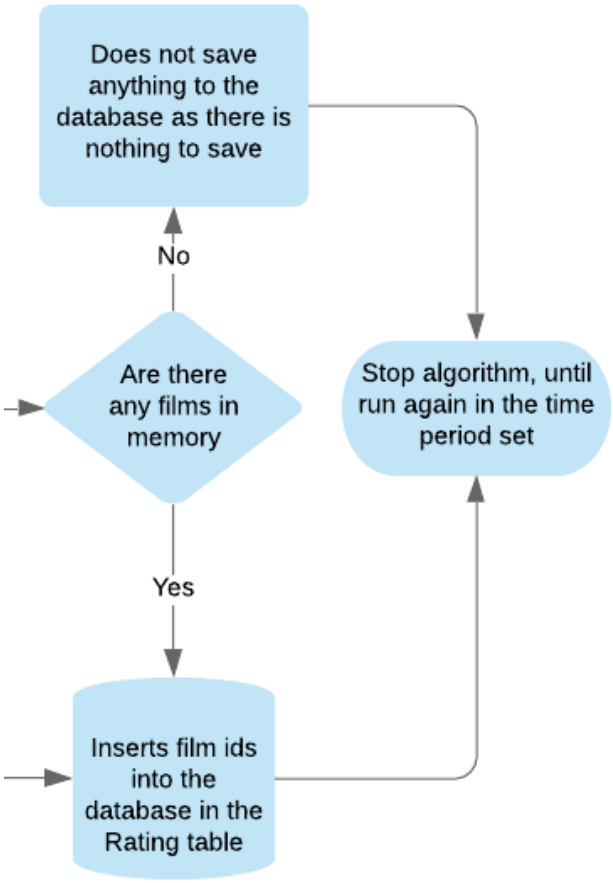

Updates Database with new information
Displays page with description of film specified
Yes
View discription
Favourite/Unfavourite
Yes
Display information about the films in sets of 10
Option for the user to view more films if there are more and if they want to, will always be available
The user is now able to do multiple things for each film displayed
Rate it
Yes
View Films Acted In or DIrected
No
User decides to stop using the website, or closes page
Look at all actors or directors
Overall 1-10
Comedy 1-10
Actors 1-10
Quality 1-10
Option for the user to view more actors or directors if there are more and if they want to, will always be available
Displays information about the Actors or Directors in sets of 10
WIth all of these having a N/A option if it does not apply
Initiates Search algorithm using the specific criteria if any is selected
No
Do they set searching conditions?
The user enters the name of the actor or director
Enters "Search for actor or director page"
Year of birth entered by user using text not dropdown menu
Yes

## Leaderboard algorithm

The input into this algorithm will be either "Overall" or "Comedy", it will calculate the average ratings for all films and store the top 100 films in the database to access when the user enters the specific page. It is then run periodically and updates the ratings so that it is the newest ratings in the database without causing performance issues.



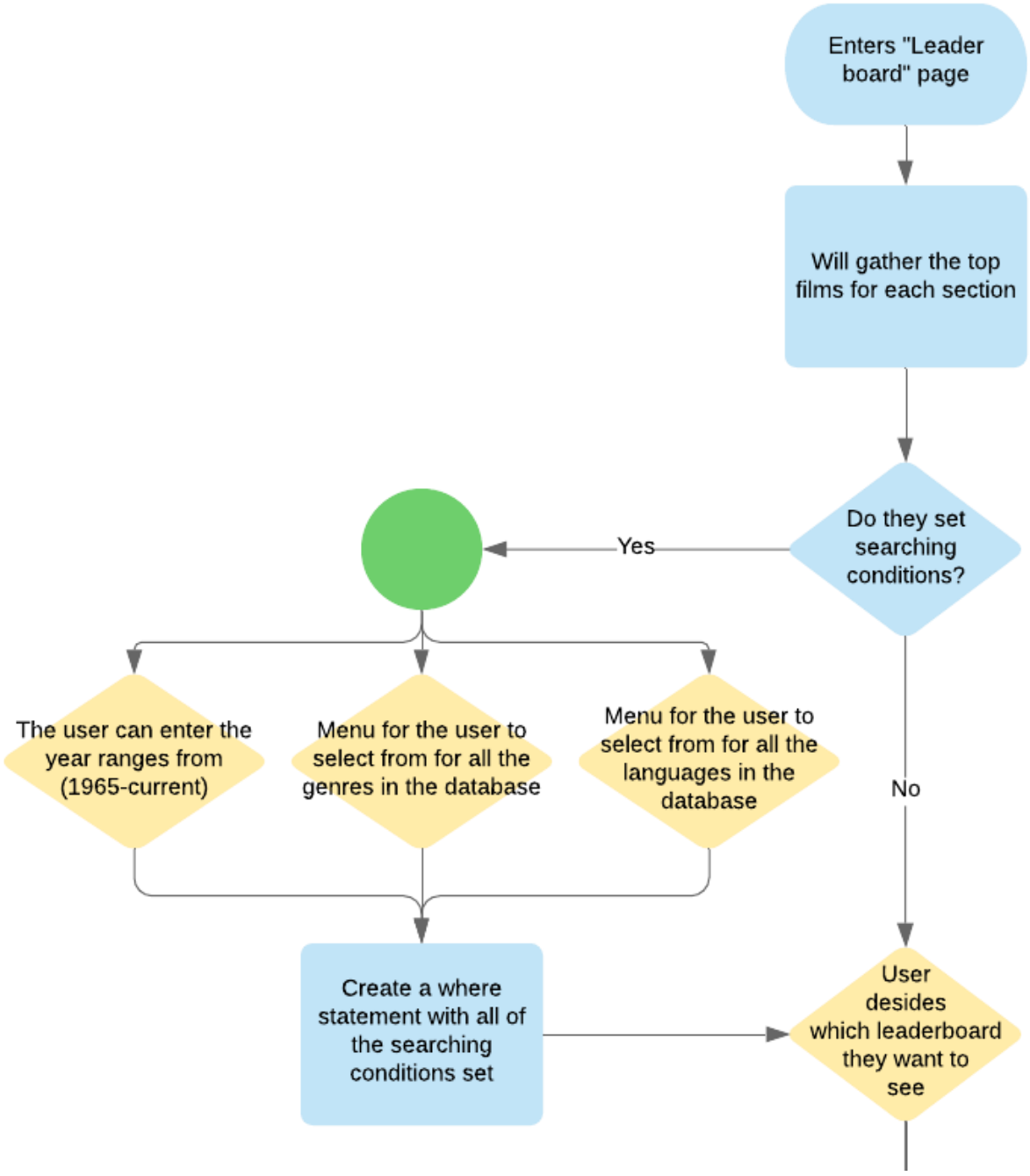

Films above 2 sd of mean rating
1 .Yes
No
Films above 1 sd of mean rating
2. Yes
No
Gather the films which have the highest rating, in this section
Films above mean rating
No
3. Yes
Are there 100 films
1,2. No
If there were any films, store those ones in memory and continue to the next criteria
1.
2.
3. No

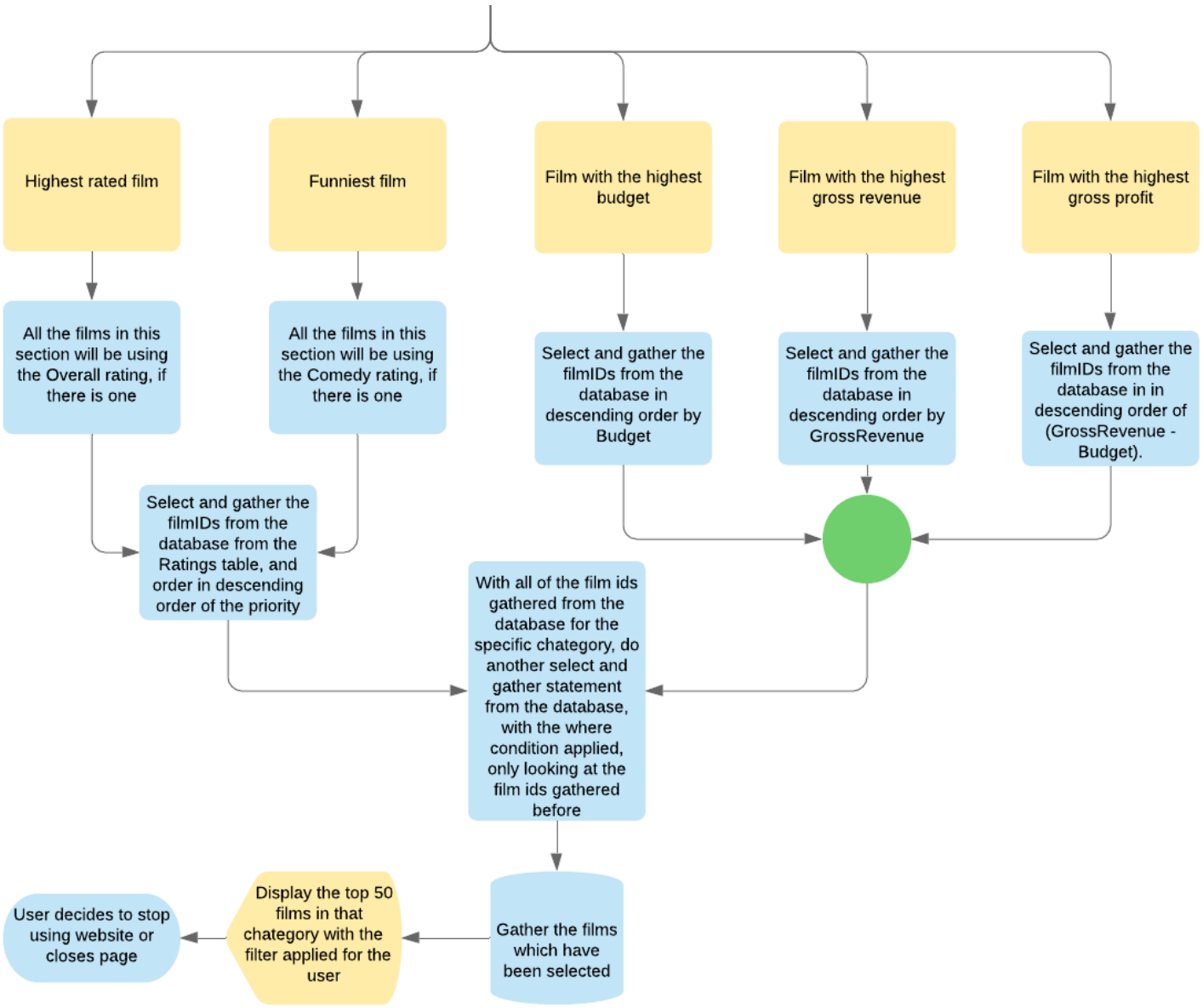

Does not save anything to the database as there is nothing to save
No
Are there any films in memory
Stop algorithm, until run again in the time period set
Yes
Inserts film ids into the database in the Rating table

## Leaderboard page

Since if the Budget or GrossRevenue is Unknown then the value is "-1", meaning the "Profit" will be negative. Both of them will not be "-1" as the Economy table only stores Budget and GrossRevenue for films with a known value, if one of them is known then the film id will appear in the table. When the user requests to see the leaderboard the negative ones will be at the bottom and will not be in the top 100 which are added to the database (top 50 which are filtered down by the criteria).

When the films are displayed (for the top 50) it will be the same sort of display model described in the searching for films, actors and directors page.




Highest rated film
Funniest film
Film with the highest budget
Film with the highest gross revenue
Film with the highest gross profit
All the films in this section will be using the Overall rating, if there is one
All the films in this section will be using the Comedy rating, if there is one
Select and gather the filmIDs from the database in descending order by Budget
Select and gather the filmIDs from the database in descending order by GrossRevenue
Select and gather the filmIDs from the database in in descending order of (GrossRevenue - Budget).
Select and gather the filmIDs from the database from the Ratings table, and order in descending order of the priority
With all of the film ids gathered from the database for the specific chategory, do another select and gather statement from the database, with the where condition applied, only looking at the film ids gathered before
Gather the films which have been selected
Display the top 50 films in that chategory with the filter applied for the user
User decides to stop using website or closes page

## Recommendation algorithm

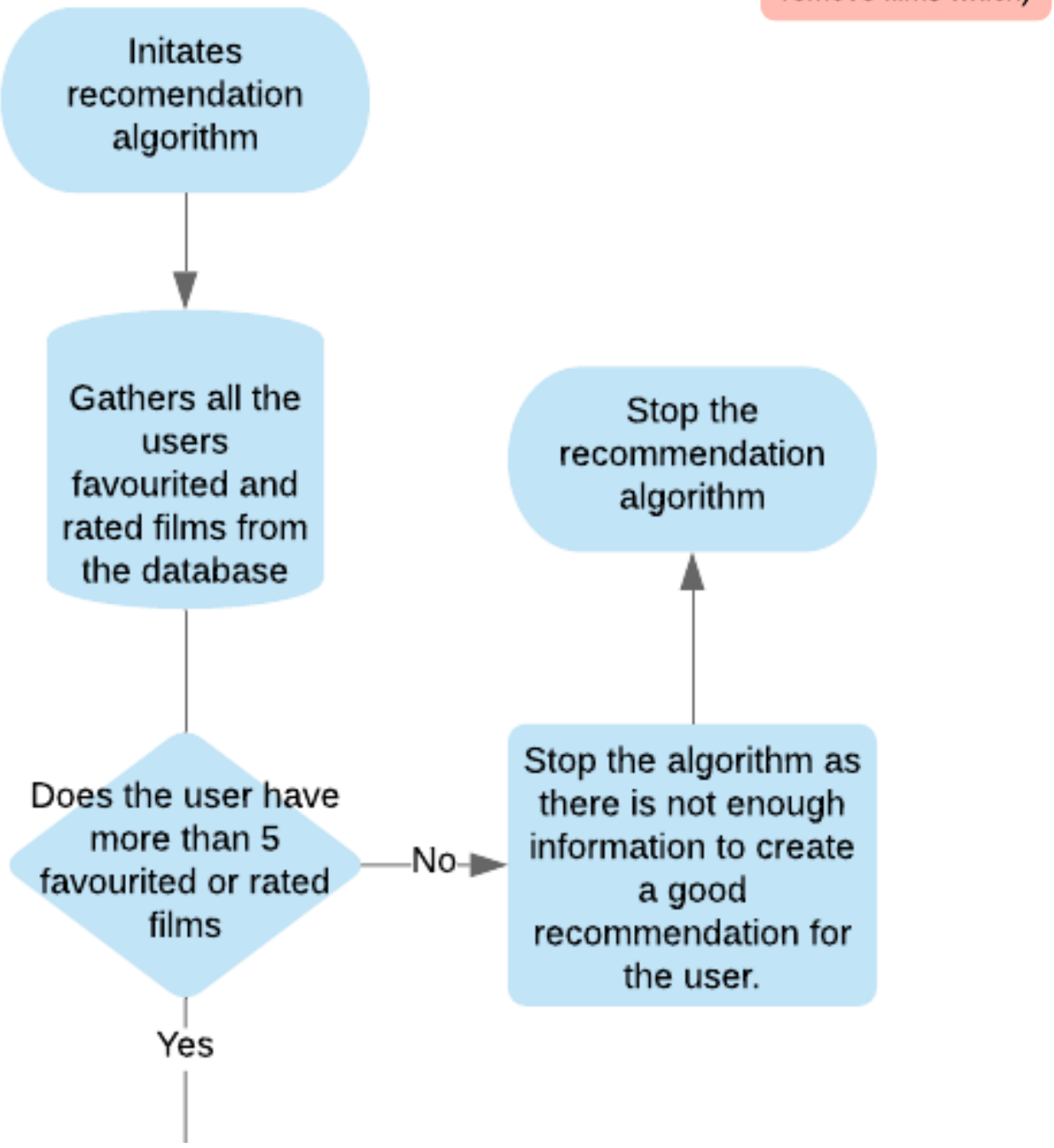
There are 3 lists;
first_weighting, second_weighting and third_weighting

sd : Standard Deviation

Will be looking at the rated and favourited films from the user. If a certain film does not have a rating in a specific category, then the algorithm will set the rating as 5 since it is in the middle value of all the possible ratings.

Continue with algorithm until the end and remove all the films which have = **CWATERFW** (continue with algorithm til end remove films which)

### Starts the algorithm

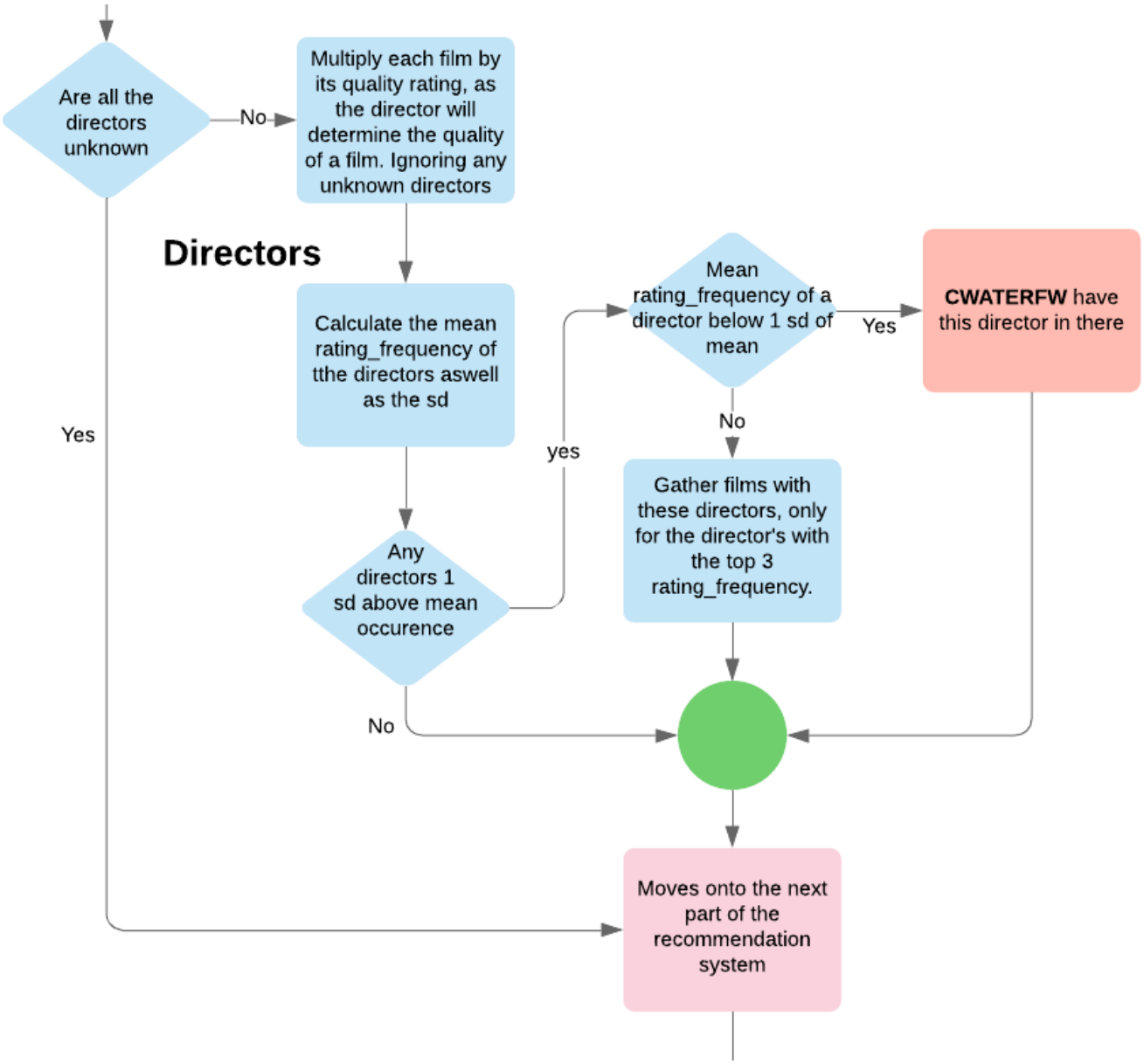


## Directors

The first section which the algorithm goes to is the director section, where it looks at all of the rated and favourited films from the user and tries to find the directors which correlate the most and appear the most. Since the quality rating determines how well the film is produced, and the director is responsible for that, which film's occurrence will be multiplied by the integer of the quality rating for that film. If there is no rating given then the quality rating will automatically be 5.

I then used the multiplied ratings, to calculate the mean and the standard deviation, and this was used to find which films were above 1 or 2 standard deviations of the mean

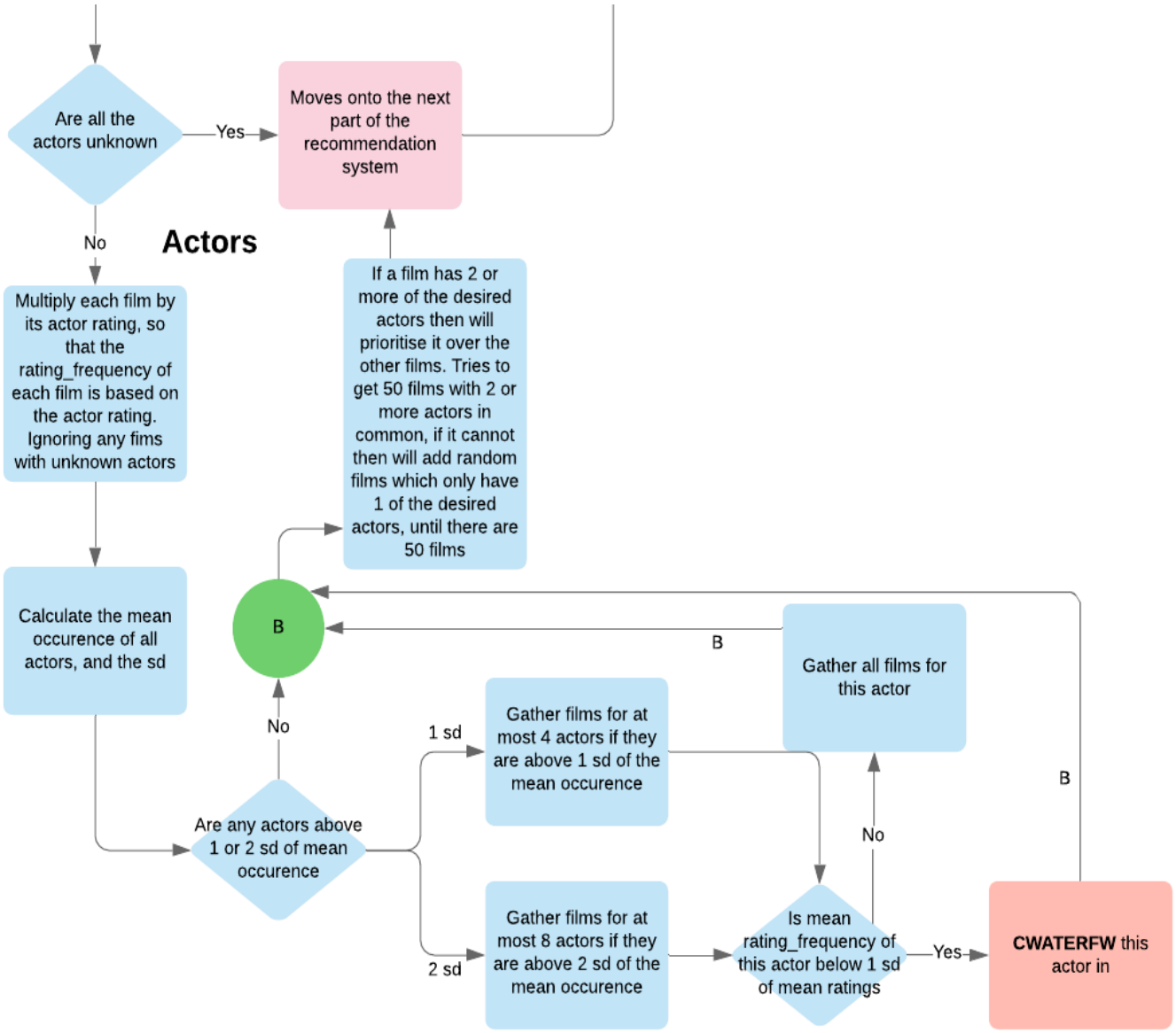
I did the same thing with the occurrence, but this was not multiplied by the rating and was just a raw value. Where I tallied how often each director directed a film and divided it by the total amount of films with a director (if I multiplied the directors occurence by the quality rating then it woulbe inaccurate).



## Actors

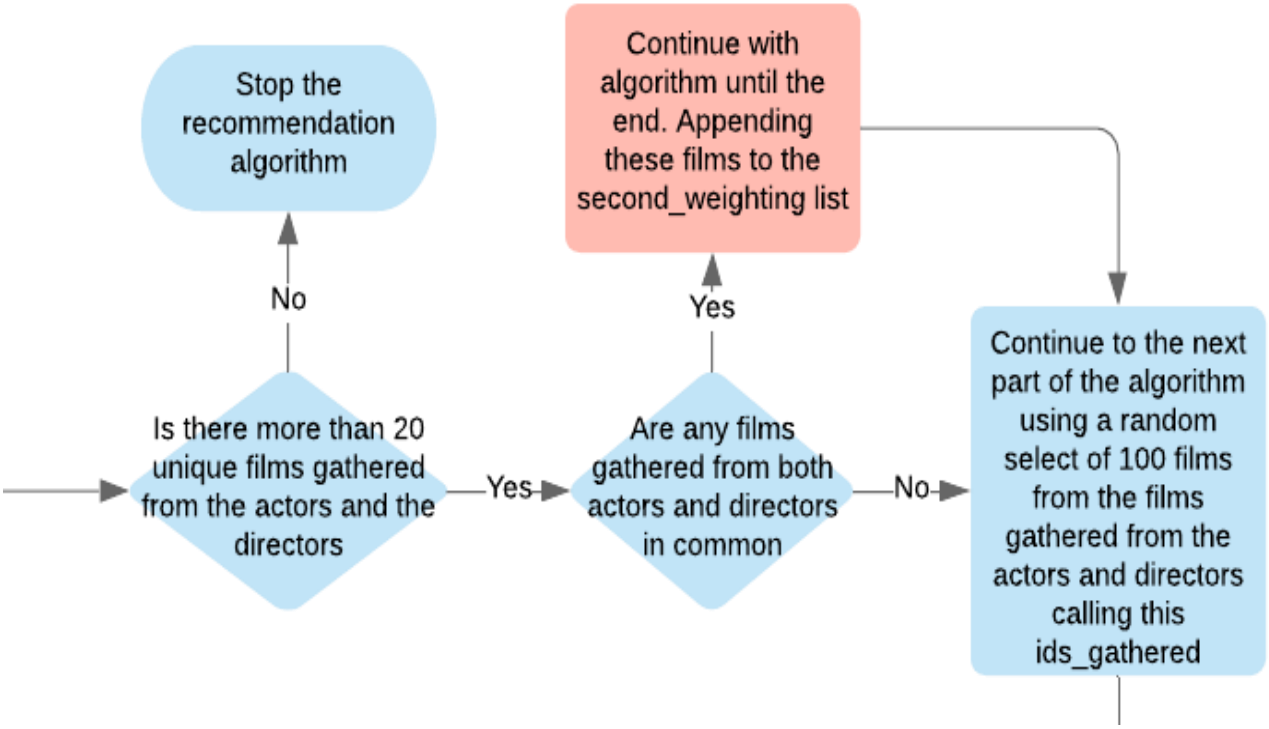
Once the director section is completed then this section is run.

The same idea of using the ratings and mean occurrence is applied to the actor section as seen in the director section but it is done for the actor rating instead of the quality rating.



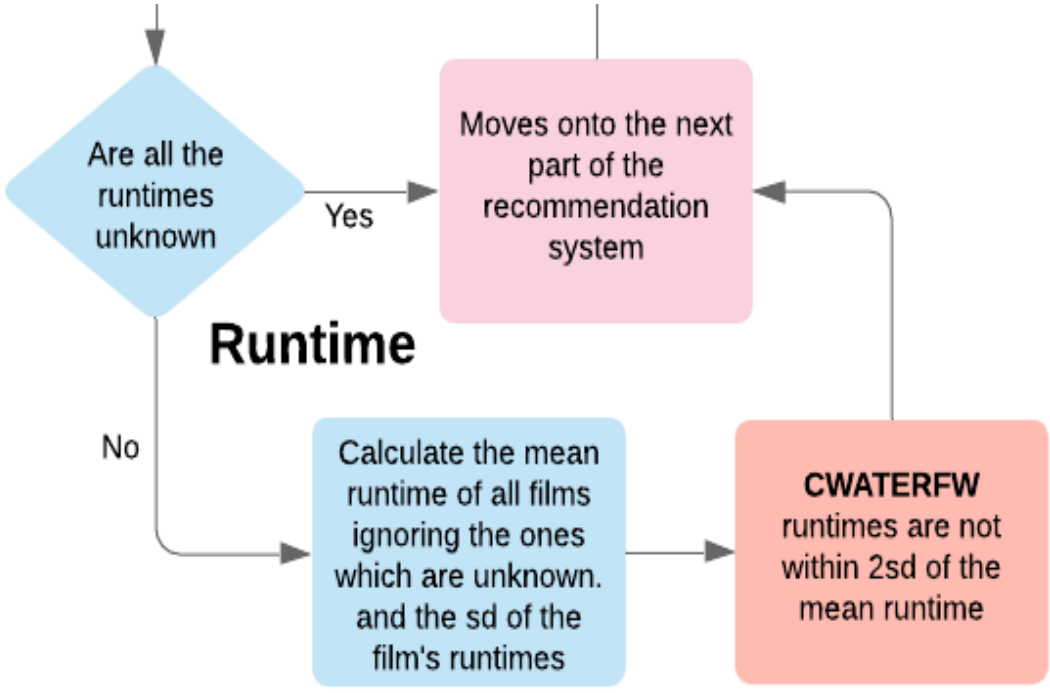
### Checks amount of films gathered

Makes sure that there are enough films gathered from the 2 previous sections of the algorithm, to create a good recommendation for the user.



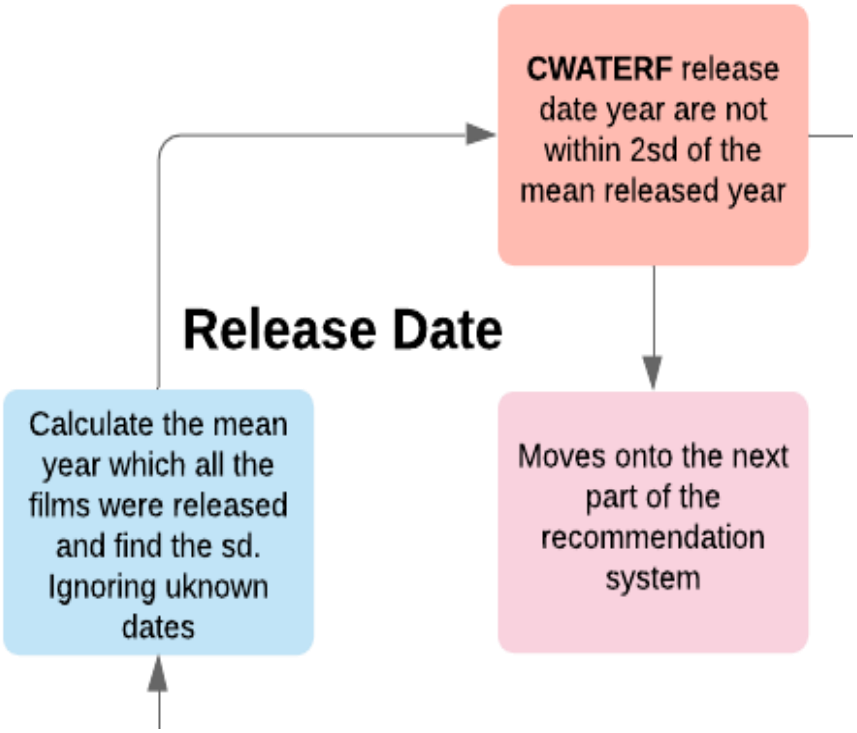

### Runtime

The next part of the algorithm is the runtimes, which will remove all of the films which are too long or too short.



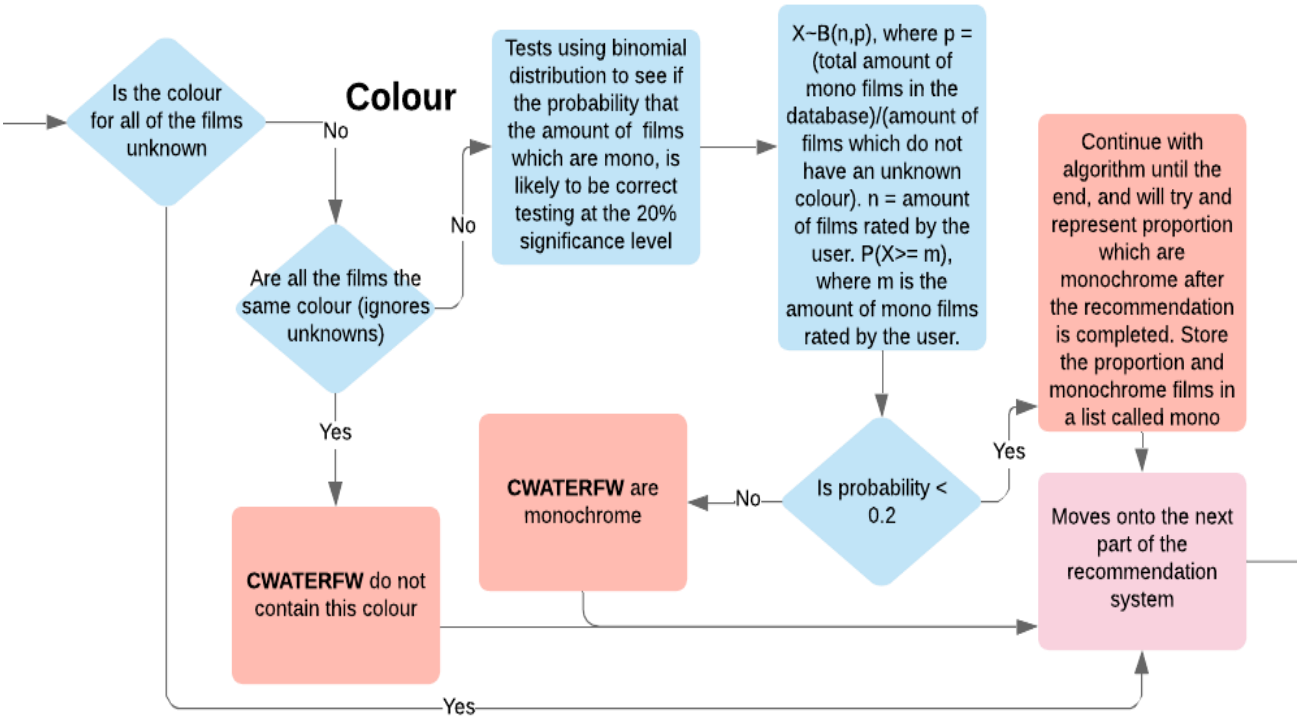
## Release date

Once runtimes is finished the release date section is run.



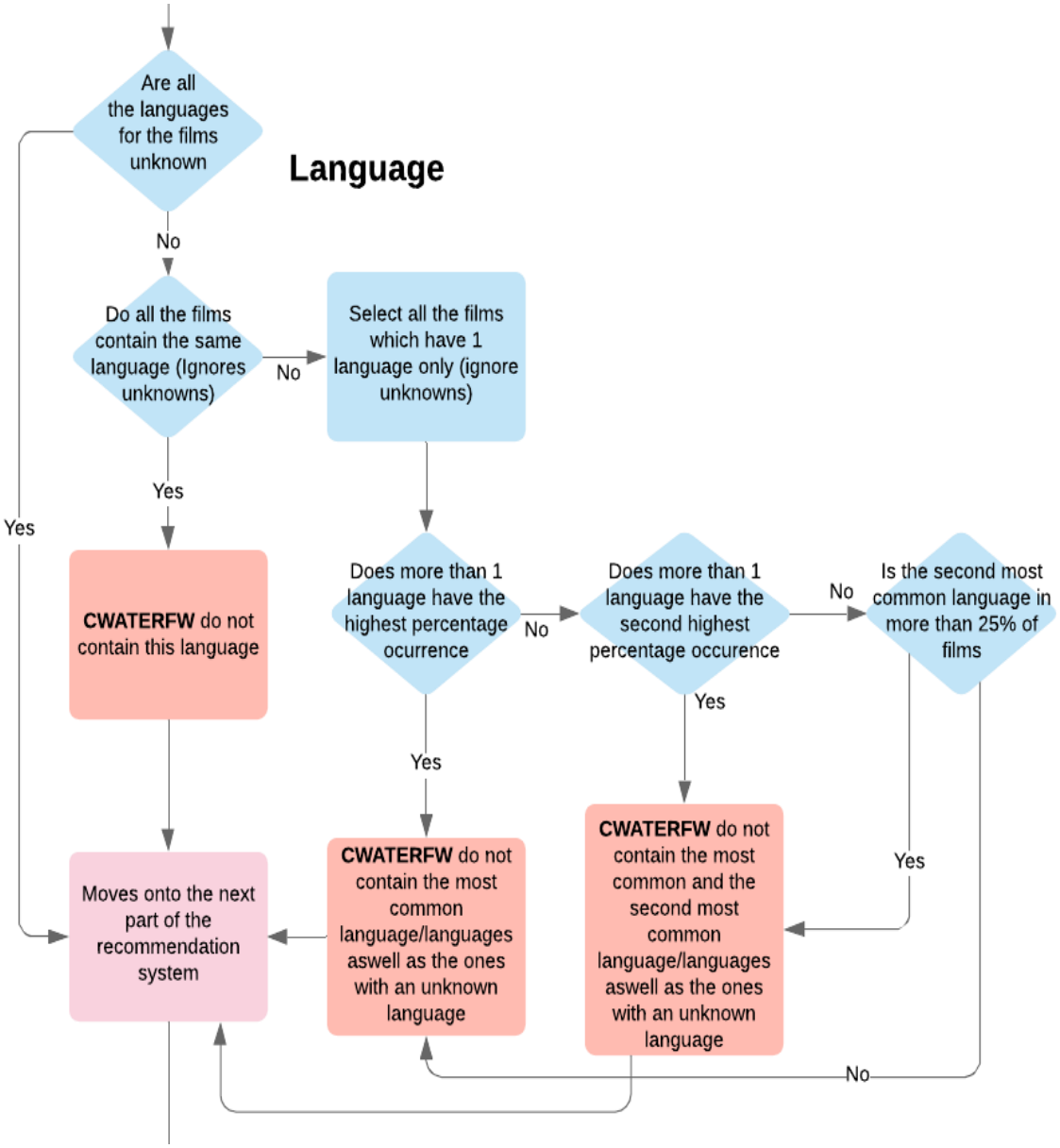

## Colour

Once release dates has finished, the next section is colour. It will remove all of the films which are not equal to all the other colours (black and white or just colour), if they are all the same



## Language

The next section is the languages, where it will find all of the languages which the user could potentially want in their recommendations.



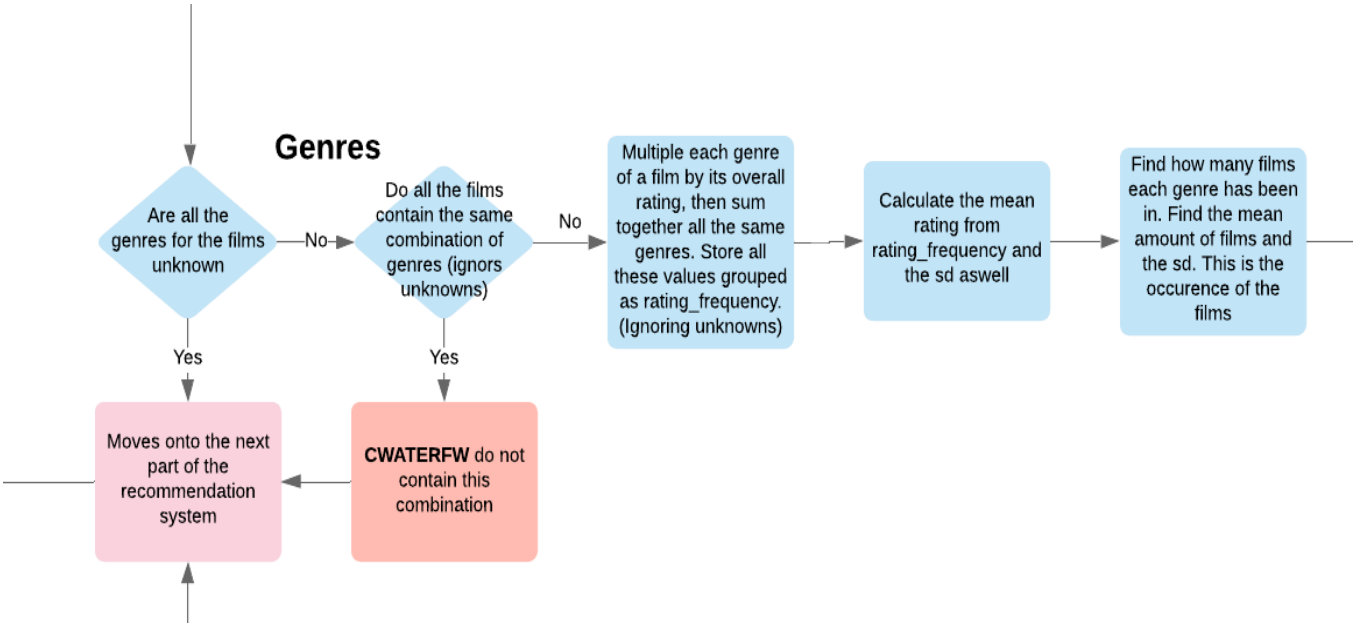
## Genre

The final section of finding the film ids which are not wanted is the genre section, which is the largest of them all, and has a lot of possible combinations to consider, this is done last as it will take the longest to complete.

The same idea as the director and actor section is applied here with the mean occurrence and the mean rating.

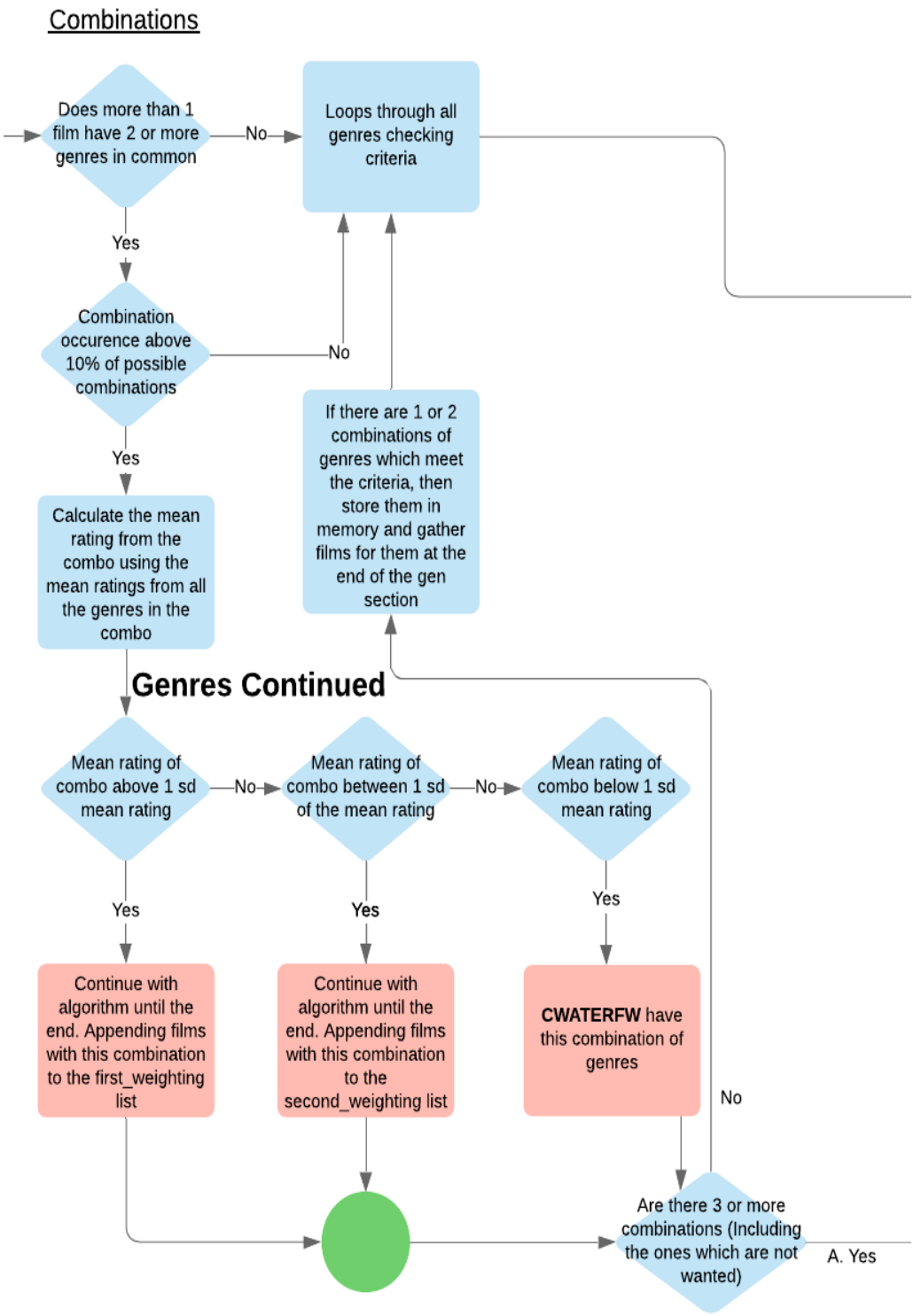
If overall rating is "N/A" (-1) then will let it equal 5 since it is the middle value, and it is likely that if the user did not rate the films, they did not have any strong feelings about it, which the rating value of 5 suggests.

This section is split up into 3 sections because it is too large to fit onto one screen shot. On the process box which "moves to the next part of the algorithm" it has an arrow each connected at the bottom, which is coming from the "A" connector.

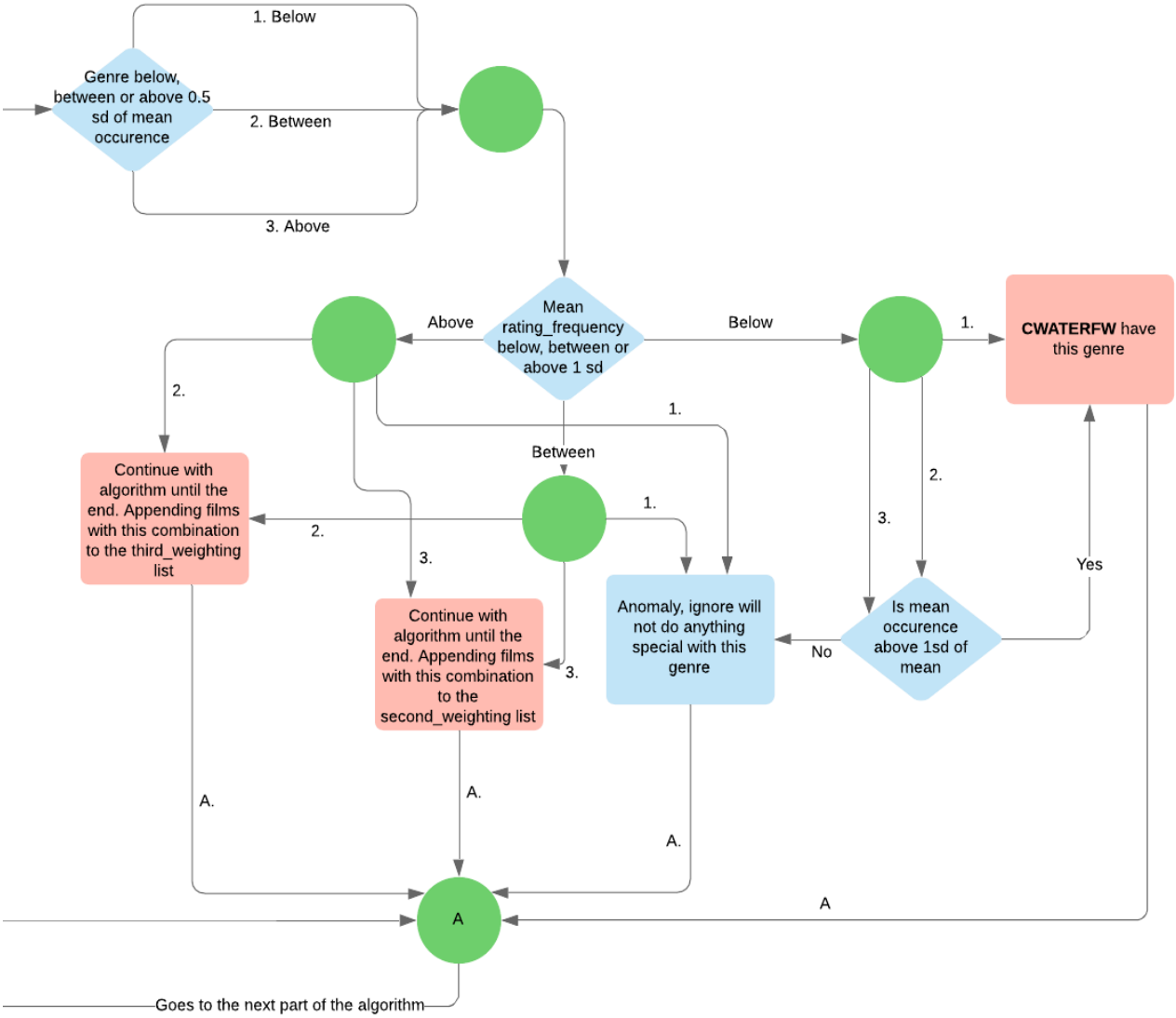


The diagram below is the diagram to the right, of the one above. The combinations are calculated in this section of the genre section of the algorithm. Once the combinations have been calculated if there are not already enough combinations then it will look at the individual genres else it will go to the "A" connector. Both sections are to the right of this diagram and will be shown in the next section.

The line at the bottom of this section of the flowchart, is an arrow leading to the process box which tells the algorithm to continue to the next section, which is the final section.

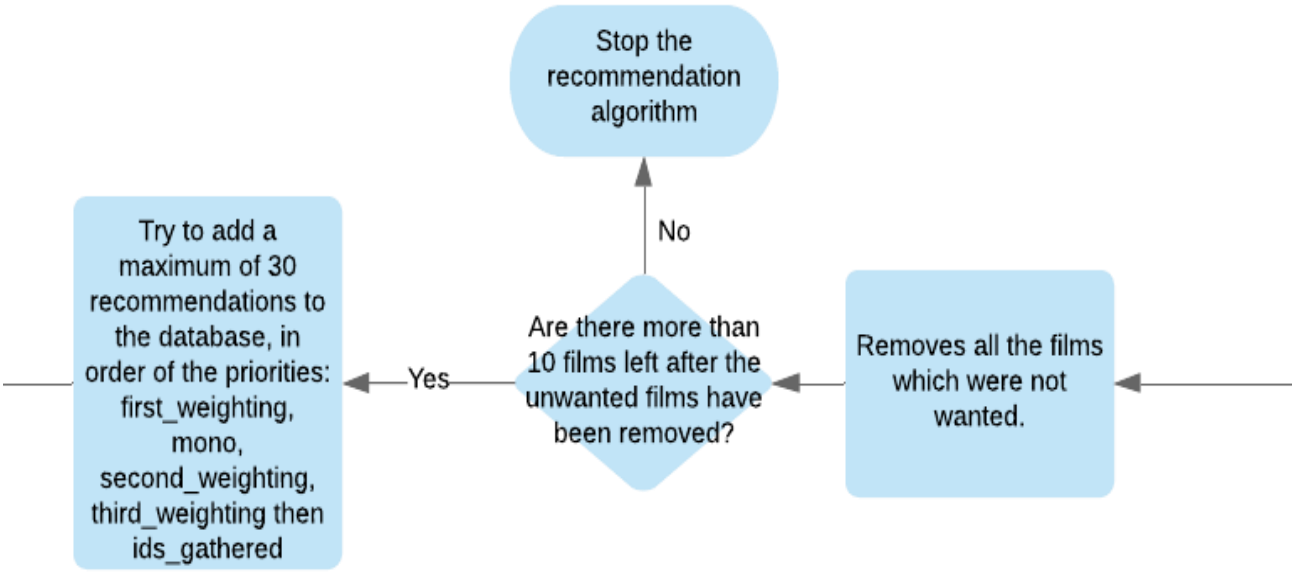


This part looks at the individual genres not the sections.

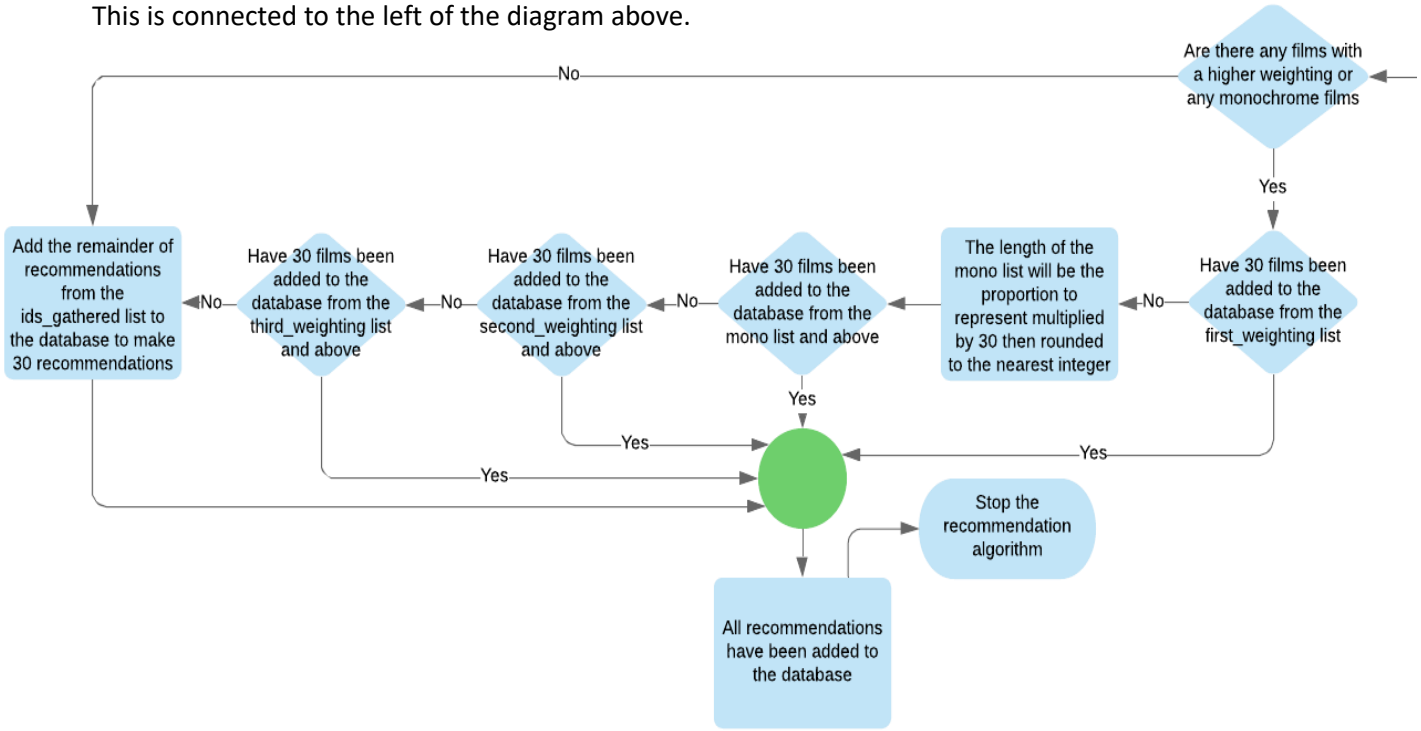


## Adding the recommendations to the database

This section is the final stage, and adds all of the 30 recommendations to the database for the user. It is run after the genre section. If there are not more than 10 films left after the amount of not wanted films has been deducted, then the algorithm will stop and not save anything to the database. It will save all of the films to the database in priority order and make sure that there is a correct proportion of black and white films added to the database, if the user wants black and white films.

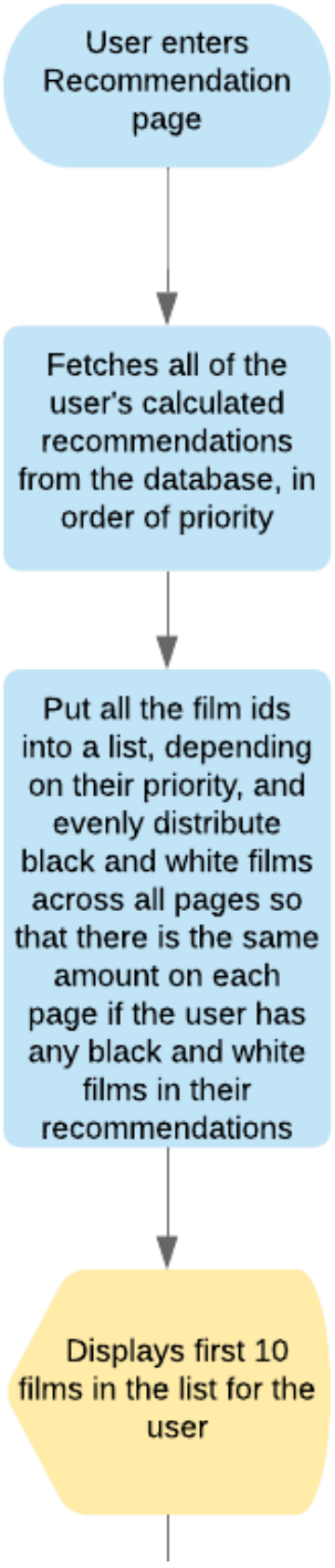


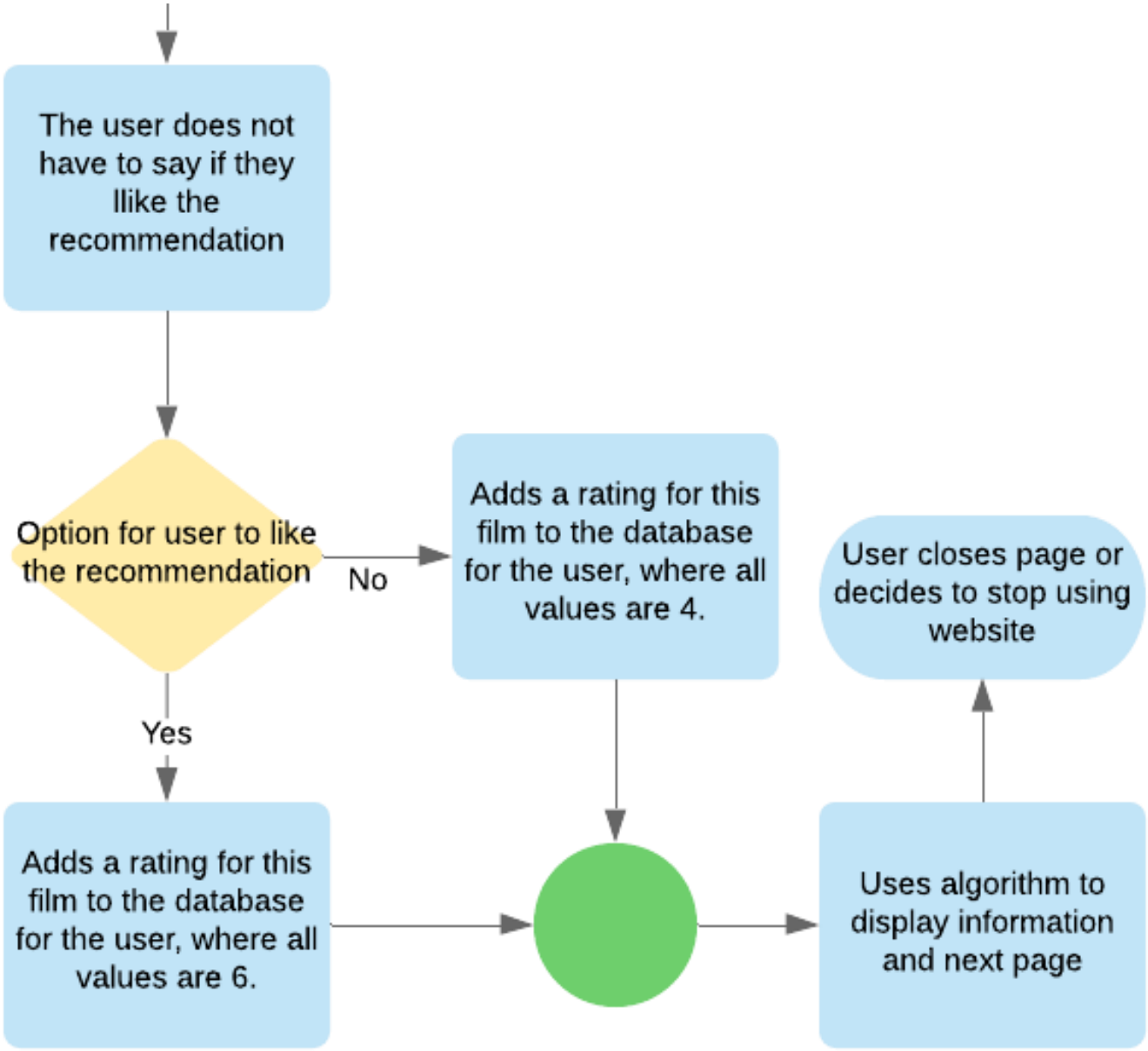

This is connected to the left of the diagram above.



## Recommendation page

The "Use algorithm to display information and next page" is the same model described in the searching for films, actors and directors page and will display each page in sets of 10 with the option to view more films if the user wants to do so




The user does not have to say if they llike the recommendation
Option for user to like the recommendation
No
Adds a rating for this film to the database for the user, where all values are 4.
User closes page or decides to stop using website
Yes
Adds a rating for this film to the database for the user, where all values are 6.
Uses algorithm to display information and next page

# SQL Statements used

## Information about functions used

I will have 4 main function for; selecting, inserting, deleting and updating to and from the database. These will have a default set of parameters so that I can just send all of the values to these functions, and it can interact with the database, instead of creating connections in each place where data needs to be interacted with the database, instead I can just call one of these 4 functions.

In all functions except “inserts_info_to_database” there is a “where_statement” parameter. If there is not any need to specify a “where” statement then the parameter will be set to “1”, which will just check to see if 1 is true and since 1 is true it will not do anything and does not affect the speed of the algorithm.

Each function connects to the database as the variable "db"  and "db.cursor()" is assigned to the variable "pointer".  The pointer the executes the SQL code. Once the code has been executed, then "db.commit()" is run to save the data to the database. In all of the functions the SQL statement is edited using "f strings". However in the "inserts_info_to_database" and "updates_database" functions, the information to add to the database is inputted using a parameter of the "Sqlite3" module, as taking in the data as a variable into the SQL statement would not be possible.

| **Function name** | **Parameters** | **Sqlite3 statement used.** | **Extra information** |
|---|---|---|---|
| selects_info_from_database | fields: str<br>table: str<br>where_statemen: str | pointer.execute(f"SELECT {fields} FROM {table} WHERE {where_statement}") | It will fetch all of the information from the database and return it as a list, if there is no information in the tables requested then it will just return an empty list. The list returned will be a list of tuples. The function allows for cross table selecting, as the join statement will be added to the table parameter. |
| inserts_info_to_database | field: str<br>table: str<br>data: list - 2d list of tuples | pointer.executemany(f"INSERT INTO {table} {fields} VALUES (?{question_marks})", data) | The fields parameter will look like fields are like “(Field1, Field2,…)” or just (Field). It is a tuple but is a converted into a string, beforehand.<br>question_marks = ",?"*fields.count(",")<br>This is used to determine how many question marks are needed in the statement depending on how many fields have been specified.<br><br>Takes a list of tuples to insert into the database, so that it is the same every time, however "executeman " can work with lists of lists. |

| | | | |
|---|---|---|---|
| deletes_from_database | table: str<br>where_statement: str | pointer.execute(f"DELETE FROM {table} WHERE {where_statement}") | |
| updates_database | field: str<br>table: str<br>where_statemen: str<br>data: tuple | pointer.execute(f"UPDATE {table} SET {fields_formatted} WHERE {where_statement}", data) | The fields parameter will look like fields are like “Field1, Field2,…” or just (Field). It is a list, without the square brackets before and after, but is a converted into a string, beforehand.<br>fields_formatted = fields.replace(",", "=?,")+"=?"<br>This is used, so that the Sqlite3 statement knows in which fields to insert the data into. |

## select_info_from_database function

SQL statements which have the "where_statement" in the form "FieldName=Value" or "FieldName IN value_list" and may also have multiple field names with an "AND" statement connecting them so that both values in each field must match, are not included here as they are self explanatory.

### searching_algorithm.py

This SQL code is run when trying to find a specific actor, director or film. This like statement is SQLite's built in fuzzy search, this will find any records which are similar to the "element" specified in the Field stated.

**Code added to where statement**

```
where_statement += f" {FIELD}Name like '%{element}%'"
```

### *Searching for a film*

#### Release date filter

This is the SQL code which is run when the user has entered an option to find a film using the release date filter, to find films between 2 specific dates. Here the SQL will find records which release dates are above "year_criteria[0]", which is the lower bound, and below "year_criteria[1]" which is the upper bound.

**Code added to where statement**

```
where_statement += f" AND ReleaseDate > {year_criteria[0]} AND ReleaseDate < {year_criteria[1]+1}"
```

#### Genre filter

This is the SQL code which is run when the user has entered an option to find a film with a specific genre,  it will find all of the records which have the "GenreID" which the user has requested. Since the genres are stored in a separate table to the rest of the information about the films, a "JOIN" statement must be used so that the information can be gathered. This is selecting information across 2 tables, so a statement to check that the film ids in both tables are the same so that it does not select that same record multiple times.

It will add to the "join_statement" so that the script knows which table to join and gather the extra information from.

**Code added to where statement**

```
where_statement += f" AND GenreID IN {'('+str(genre_criteria)[1:-1]+')'} AND GenreToFilm.FilmID = Films.FilmID"
```

**Code added to join statement**

```
join_statement += " JOIN GenreToFilm"
```

#### Language filter

This SQL code is the same principal as the genres, as the tables are structured the same.

**Code added to where statement**

```
where_statement += f" AND LanguageID IN {'('+str(language_criteria)[1:-1]+')'} AND LanguageToFilm.FilmID = Films.FilmID"
```

**Code added to join statement**

```
join_statement += " JOIN LanguageToFilm"
```

**Code which is run**

This is the code which is run with the where_statements and the join_statements applied, when searching for a film.

```
selects_info_from_database("Films.FilmID, Films.FilmName", "Films"+join_statement, where_statement)
```

### *Searching for an Actor or Director.*

When searching for an actor or director there is only a filter for the date of birth, which is a similar principal as the release date filter for the films.

#### Date of birth filter

This SQL code is the same principal as the release date filter for the films, as the fields are structured the same way and so is the format of the date.

**Code added to the where statement**

```
where_statement += f" AND DOB > {year_criteria[0]} AND DOB < {year_criteria[1]+1}"
```

**Code which is run**

This is the code which is run with the where_statements applied, when searching for an actor or director. The parameter for the function to state which field and table are using a variable to state which one to select from as, it will be different for both actors and directors.

```
selects_info_from_database(f"{FIELD}ID, {FIELD}Name", f"{FIELD}s", where_statement)
```

### *Find top films to display*

The SQL code here is used to reorder the films by the revenue, so that the most successful films will appear at the top of the list when displayed to the user.

It will use an “ORDER” statement to change the order which the films are in and in based on the “GrossRevenue” field and in descending order by using the “DESC” statement.

**Code which is run**

```
selects_info_from_database("FilmID", "Economy", f"FilmID IN {all_ids} ORDER BY GrossRevenue DESC")
```

## leaderboard.py

Used to gather the top information from the database for the films, to display to the user on the leaderboard page.

### *Gathering most profitable films from the database*

The SQL code here will gather, all of the films which are in the "Economy" table in order of their profit. With the most lucrative appearing at the top of the list, so that when the user selects this option to see this leaderboard they will get the results which they are wanting.

As the profit is not stored in the database, the SQL code will calculate it by deducting the budget from the revenue (GrossRevenue - Budget). It will not select records where the "Budget" does not equal -1 and 0 and the "GrossRevemue" is not equal to 1, the reason for this is that if these values were chosen then the actual value of the profit will be wrong and the films displayed on the leaderboards would be incorrect, as the most lucrative films. It then orders all of the films in descending order using a "DESC" statement.

**Code which is run**

```
selects_info_from_database("FilmID", "Economy", f"Budget NOT IN (-1, 0) AND GrossRevenue <> -1 ORDER BY (GrossRevenue-Budget) DESC")
```

### *Gathering highest rated films in the Overall and Comedy chategory from the database from the database*

The SQL code here is used to reorder the films by the priority, so that the films with the highest priority will appear at the top of the list when displayed to the user.

It will use an "ORDER" statement to change the order which the films are in and in based on the "Prioity" field and in descending order by using the "DESC" statement. And will also make sure that the chategory selected from is the same one as the parameter of the function.

**Code which is run**

```
selects_info_from_database("FilmID", "TopRated", f"Chategory = '{chategory_number}' ORDER BY Priority DESC")
```

### *Gathering the highest profit and revenue from the database.*

This gathers the remaining films from the database, the only other categories left are "Budget" and "GrossRevenue". As it does not have a where statement it will have the value of "1" as its where statement, which will check to see if "1" is true, and since "1" is always true it will just ignore it and go to the "ORDER" statement. It is going to order the films by the chategory, either the Revenue or the Budget, using the "DESC" statement so that they are in descending order, with the largest value at the top of the list for the user to see first.

**Code which is run**

```
selects_info_from_database("FilmID", "Economy", f"1 ORDER BY {chategory} DESC")
```

### *Filtering films with criteria*

If the user has set filters when they are searching for a film, it will filter the films down here. This section is very similar to the “search for film” section in the “searching_algorithm.py” file.

The only part which is different is the beginning, when the “like” statement is appended to the where_statement. Instead it will just select all of the film ids which are in the list of the film ids gathered from the section above, once the films have been selected from the specific chategroy. The rest of the filtering down is the same and will not be shown, as you can see the section above.

**Code which is run**

```
where_statement = f"Films.FilmID IN {all_film_ids}"
```

### *Gathers ratings from the database to calculate leaderboard*

Here it will gather all of the ratings from the users, from the “Ratings” table, it will not select the ratings where it is unknown “-1”, as this will not create a true average rating.

**Code which is run**

```
selects_info_from_database(f"FilmID, {chategory}", "Ratings", f"{chategory} <> -1")
```

### gathering_types.py

Used to filter down all of the films which have an unknown id for the  field specified, as when calculating recommendations, I do not want to favour unknown information over known data, as known data will give the user a better recommendation. It will select all of the film ids which are in the list as long as the value in the field is not unknown. Field will be a value such as language or genre.

**Code which is run**

```
selects_info_from_database(f"FilmID, {field}", table, f"FilmID IN {film_ids} AND {field} <> {UNKNOWN_ID}")
```

## formulating_recommendations_from_types.py

### *Gathering films which are related to directors*

Used to gather all of the film and director ids from the database to calculate the recommendation for the user, where all of the director ids wanted are in the tuple “director_ids_wanted”. It also runs another “SELECT” statement inside to find all of the film ids which are not wanted, by using the directors which are not wanted from the tuple “director_ids_not_wanted” and finding all of the films which they are in, as long as they are not in the user’s currently rated and favourited films.

**Code which is run**

```
selects_info_from_database("FilmID, DirectorID", "DirectorIntegrator", f"DirectorID IN {director_ids_wanted} AND FilmID NOT IN (SELECT FilmID FROM DirectorIntegrator WHERE DirectorID IN {director_ids_not_wanted}) AND FilmID NOT IN {favourited_and_rated_film_ids})")
```

### *Gathering films which are related to actors*

Used to gather all of the film and director ids from the database to calculate the recommendation for the user, where all of the actor ids wanted are gathered and all of the film ids which are not wanted are ignored, these include; the user’s favourited and rated films and the film ids not wanted from the previous filtering process.

**Code which is run**

```
selects_info_from_database("FilmID, ActorID", "ActorIntegrator", f"ActorID IN {actor_ids_wanted} AND FilmID NOT IN {favourited_and_rated_films + film_ids_not_wanted}")
```

## formats_and_creates_dictionary_for_films.py

### *Gathers genre, language, actor and director ids for specific film*

This statement below is used to gather all of the genre, language, actor and director ids for a specific film, it is written to find the ids for the Genres but works to find all information for, languages, actors and directors as well, the field name and the table name just needs to be changed to the correct information which corresponds to that information. When selecting the field, it is using a “CAST” statement to change the type of value which will be returned, as the ids are stored as integers in the database but are wanted as strings. “VARCHAR(11)” is the value to convert a variable to a string in SQL.

**Code which is run**

```
selects_info_from_database("CAST(GenreID AS VARCHAR(11))", "GenreToFilm", f"FilmID = {film_id}")
```

### *Gathers all of the information for genres, languages, actors and directors to display*

The section below is for actors but is the same principal for the directors, it will gather the id of the actor or director as a string, using the “CAST” statement which was mentioned in the section above, as well as their name and their picture from the database, as long as they are in the list specified.

**Code which is run**

```
selects_info_from_database("CAST(ActorID AS VARCHAR(11)), ActorName, ImageURL","Actors", f"ActorID IN {actor_ids}")
```

This SQL code is similar to the section above but is for genres and languages so there is not an image/picture to display so that field is not selected.

**Code which is run**

```
selects_info_from_database("CAST(GenreID AS VARCHAR(11))", "GenreToFilm", f"FilmID = {film_id}")
```

## inserts_info_to_database function

As this function does not take a "where_statement", all uses of it are self explanatory. It can insert both a list containing a single tuple and a list containing multiple tuples.

However as all of SQL statements run for this function are self explanatory, I still included one so that the format can be understood.

### user_interactions_with_database.py

Single record being added to the database, when the user rates a film, then the rating is added to the database

**Code which is run**

```
inserts_info_to_database("(Overall, Comedy, Actors, Quality, FilmID, UserID)", "Ratings",
[tuple(ratings + [film_id, user_id])])
```

## deletes_from_database function

SQL statements which have the "where_statement" in the form "FieldName=Value" and may also have multiple field names with an "AND" statement connecting them so that both values in each field must match, are not included here as they are self explanatory.

However as all of SQL statements run for this function are in the form stated above , I included one which is in that form so that the format can be understood.

app.py

Used to remove the user's favourited film from the database when they request to do so.
Where the UserID equals the user's id in the session.

**Code which is run**

```
deletes_from_database("Favourites", f"FilmID = {film_id} AND UserID = {session['user_id']}")
```

## updates_database function

SQL statements which have the "where_statement" in the form "FieldName=?" and may also have multiple field names with an "AND" statement connecting them so that both values in each field must match, are not included here as they are self explanatory.

However as all of SQL statements run for this function are in the form stated above , I included one which is in that form so that the format can be understood.

user_interaction_with_database.py

When the user rates a film, if they have already rated the film it will update their current rating. Where the variable "ratings" has all of the integer values of the ratings which needs to be added to the database.

**Code which is run**

```
updates_database("Overall, Comedy, Actors, Quality", "Ratings", "FilmID =? AND UserID =?", tuple(ratings + [film_id, user_id]))
```

## Navigation

In this section I will talk about how you can navigate around my website and where each different page leads, as well as any special features which the page has. There will be 2 sections with one of them being broken down into 2 smaller ones as it would be too large to explain in one segment.

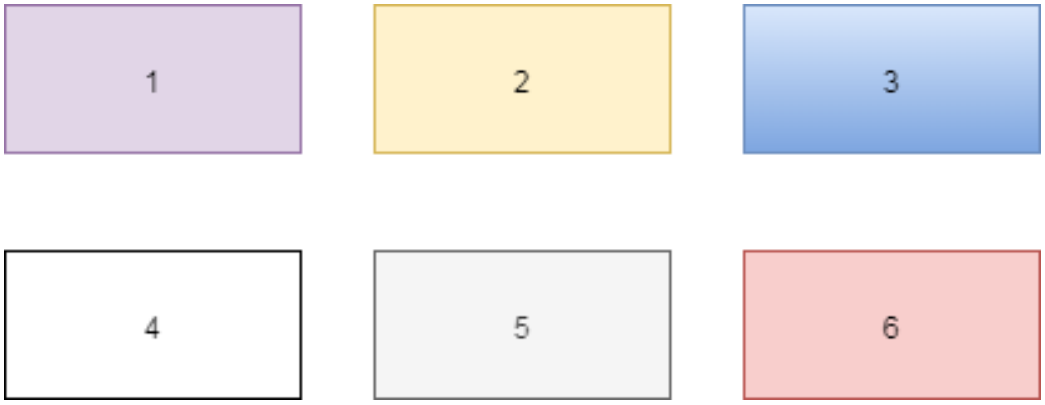
There is a section on how to navigate the website when the user is not logged in, and a section on how to navigate the website when the user is logged in.

Each of the 3 sections will have this table, describing the reason to have each page and what it does.

Each section will also have screenshots of how it will look "roughly" on the website to get the best understanding on what to expect.

| Page name | Purpose and function |
|---|---|
| | |
| | |
| | |

## Meaning of boxes




1. A main page with important information about data, this will have lots of features inside of it with lots of possible directions it can lead to. For Film, Actor and Director information pages.

2. If the page is a dropdown menu

3. A link to a page, once the user clicks on it a page will appear with information the user can enter or view.

4. A redirect page, or a none major page which cannot be accessed through the navigation bar, these are used to send data to the database to be checked or for information to be requested and sent back to the website by the user.

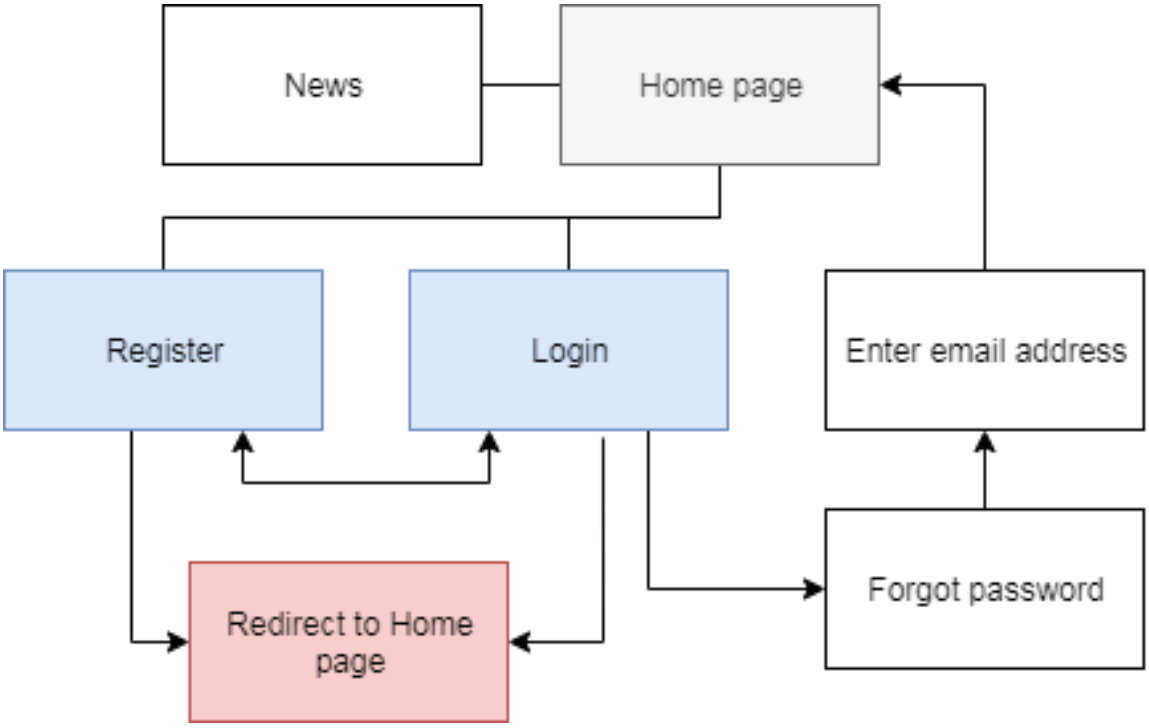
5. The Home page

6. A redirect box, so that the navigation diagram knows its next location to go to, used so that there are not lots of arrows going everywhere making the diagram messy.

## Before the user logs in

These are all the pages the user can access before they are logged into the website.

If the user is not logged and they try to access one of the other pages such as "Search for film" then they will be redirected to the login page telling them to login before they can access it, once they login they will be redirected back to the page they tried to access.

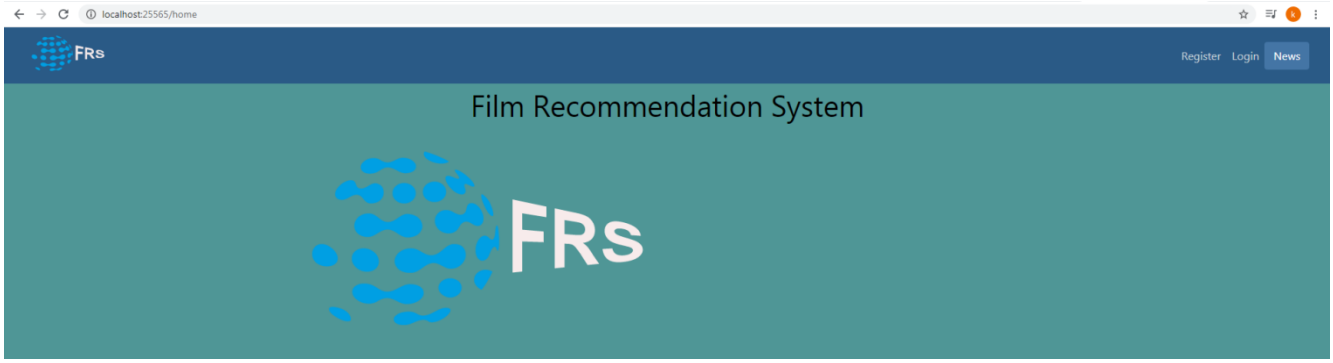



| Page name | Purpose and function |
|---|---|
| Home page | The main page of the website, there will be an image which will link to the "Search for film" page but the user will be redirected to the login page if they click on it as they are not currently logged in and cannot access that page yet. This gives the user the incentive to sign up to the website to view the information, as they will be curious about what this route does.<br>Located on the navigation bar so that the user can access this from any page they are on. |
| News | A modal which will display news about the website, important information which the user may want to know about. Located on the navigation bar so that the user can access this from any page they are on. |
| Register | Where the user can register to the website. Once they are registered they will be receive an email saying they have signed up. If the user already has an account then there will be another link on this page linking them to the login page.<br>Located on the navigation bar so that the user can access this from any page they are on. |
| Login | Once the user has created an account they will be able to login to the website. If the user does not have an account there will be another link on this page linking them to the register page, as well as a link linking them to the forgot password page where they can request their password to be emailed to them.<br>Located on the navigation bar so that the user can access this from any page they are on. |
| Redirect to Home page | Sends the user back to the home page telling them information, if there account has been created successfully and if they have logged in successfully, it will also display information such as if the email address or |

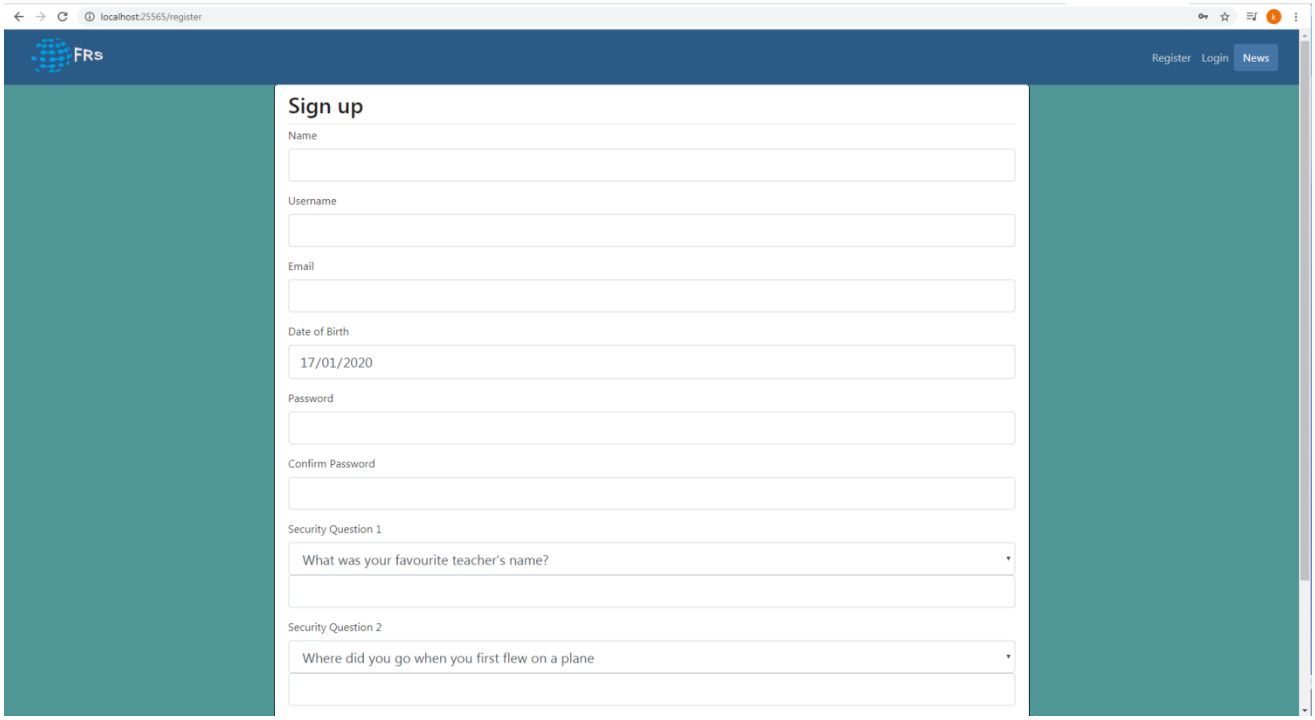
| | |
|---|---|
| | username already exists when the user tries to register. Or if there password is wrong when they login. |
| Forgot password | Name of the page where the user can request their current password. this is used to redirect to the Enter email address page. |
| Enter email address | The user can enters their email address, and their current password will be emailed to them, with information telling them that it is a good idea to change their password. Once it has been emailed the user will be redirected back to the homepage |

## How it will look on the website

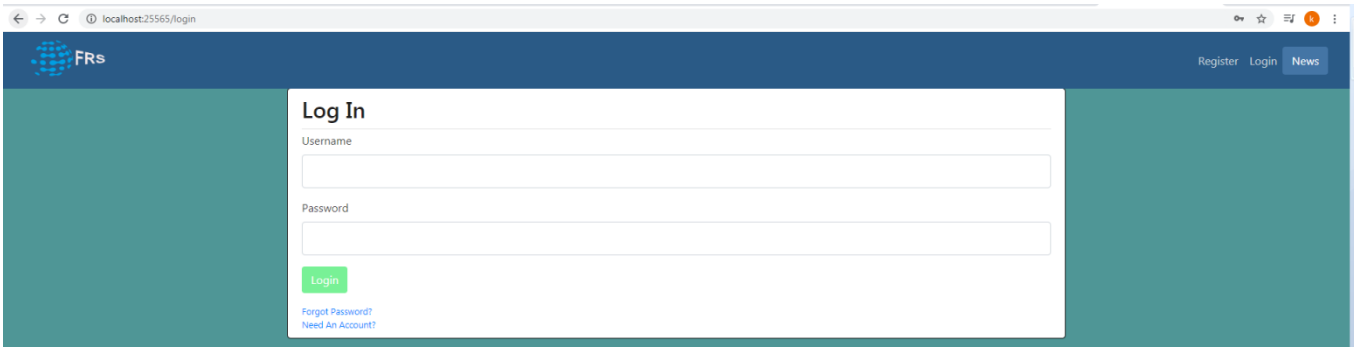
### *Home page*



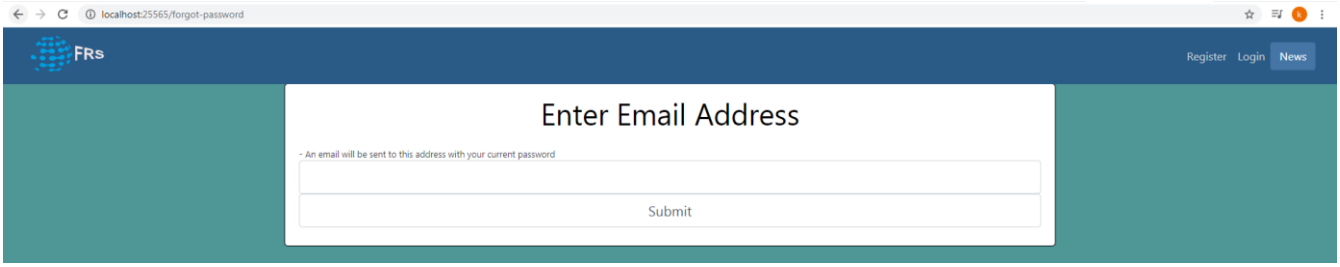

### *Register page*



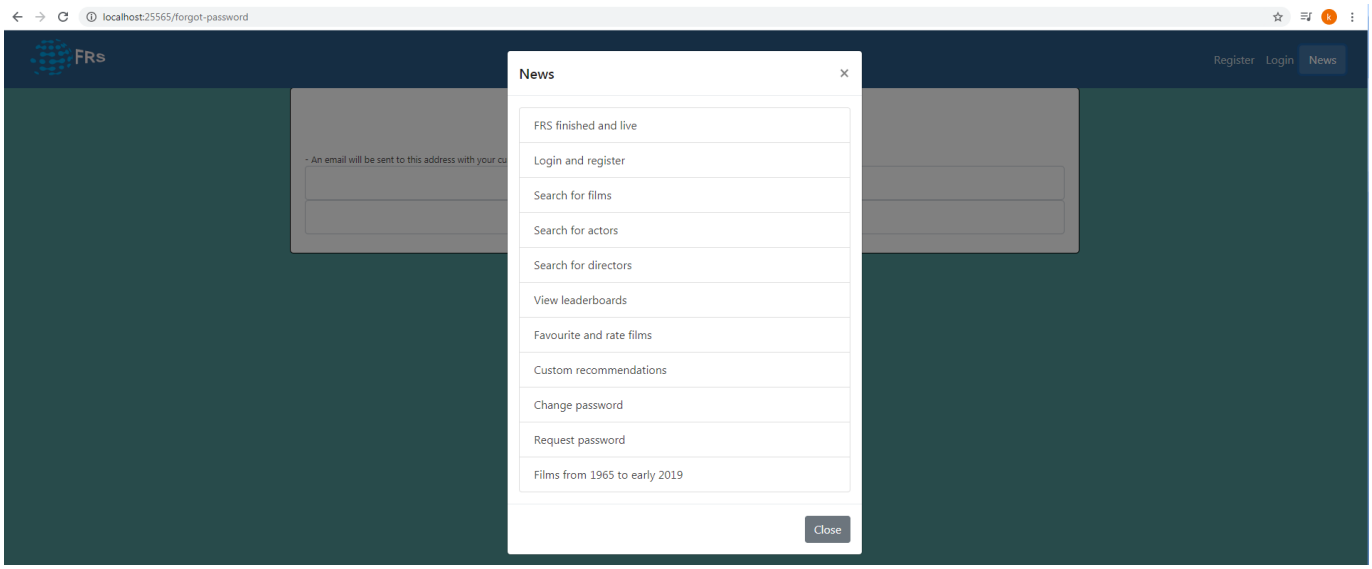
## *Login page*




## *Forgot password page*



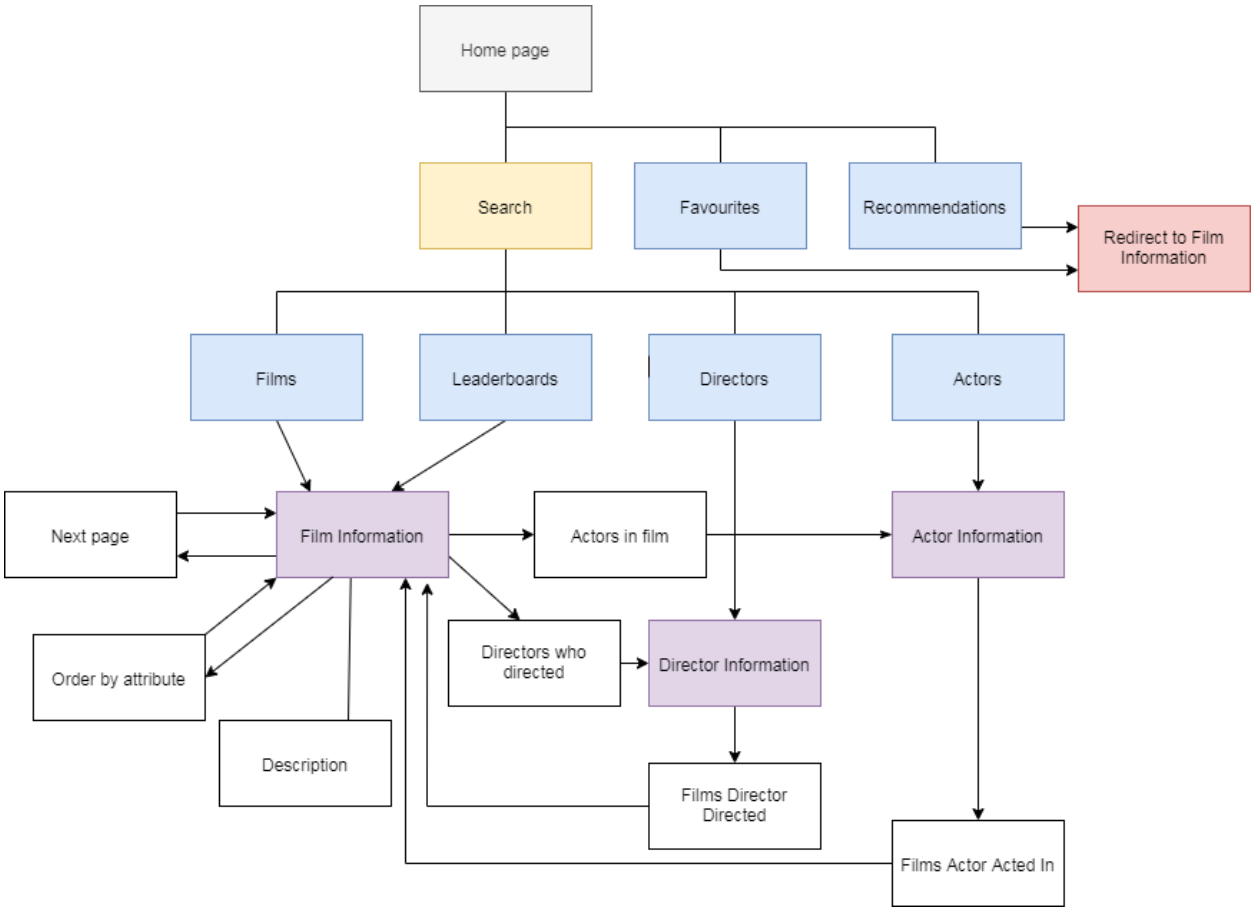

## *News modal*



## Once the user has logged in

Once the user has logged into the website the navigation bar will change, this section is split up into 2 parts the left and side and the right hand side, it is done like that so that it is easier to read and understand. The left hand side of the navigation bar is for films, actors and directors. The right hand side of the navigation bar is for user information such as logging in and changing the password.

### Left hand side




| Page name | Purpose and function |
|---|---|
| Home page | The main page of the website, there will be an image which will link to the "Search for film". The user will be redirected to this page if any error occurs, or if they do not have any films in their favourite list, as well as when the user clicks on the new recommendation button on the recommendation page, it will bring them back to this page telling them if a recommendation has been created or not. Located on the navigation bar so that the user can access this from any page they are on. |
| Favourites | The page which fetches all of the user's favourited films, it then is redirected to the Film Information page to display them all to the user. Located on the navigation bar so that the user can access this from any page they are on. |
| Recommendations | The page which fetches or creates all of the recommendations for the user, it will be redirected to the Film Information page to display them all to the user. However when the user is looking at a recommendation page then there will be extra information such as if the user wishes to create a new recommendation, which will refresh all of the current ones to |

| | |
|---|---|
| | something which is more up to date based on their favourited and rated films.<br>Located on the navigation bar so that the user can access this from any page they are on. |
| Redirect Film Information | Sends the user Film Information page, so that the information about the Favourite films or the recommended films can be displayed to the user. |
| Search | A dropdown menu where the user can select from; Films, Leaderboards, Actors or Directors.<br>Located on the navigation bar so that the user can access this from any page they are on. |
| Films | The page where the user can enter a query for a film they want to search for. It will have filters; languages, genres and the year it was released.<br>Located on the navigation bar so that the user can access this from any page they are on. |
| Leaderboard | The page where the user can pick which leaderboard they want to see films for, there will be 5 possible options; Highest budget, Largest revenue, biggest profit, best overall rating and most funny. It will also have filters; languages, genres and the year it was released.<br>Located on the navigation bar so that the user can access this from any page they are on. |
| Actors | The page where the user can enter a query for an actor they want to search for. It will have a filter for the date of birth of the actor.<br>Located on the navigation bar so that the user can access this from any page they are on. |
| Actor Information | Displays all of the information about the actor including their; name, age, date of birth, death date, place of birth, spouses. This page can only be accessed through the Actors page or the link from the Film information page looking at information about the actor. |
| Films Actor Acted In | Button/ link available from the Film information page for each actor, allowing the user to view details about the actor in more detail, once the user clicks on this they will be redirected to the Actor Information page, just for one actor. |
| Directors | Similar to the "Actors" page but just for directors. The page where the user can enter a query for a director they want to search for. It will have a filter for the date of birth of the director.<br>Located on the navigation bar so that the user can access this from any page they are on. |
| Director Information | Similar to the "Actor Information" page but just for directors. Displays all of the information about the director including their; name, age, date of birth, death date, place of birth, spouses. This page can only be accessed through the Directors page or the link from the Film information page looking at information about the directors. |
| Films Director Directed | Similar to the " Films Actor Acted In" page but just for directors. Button/ link available from the Film information page for each director, allowing the user to view details about the director in more depth, once the user clicks on this they will be redirected to the Director Information page, just for one director. |
| Film Information | The main page for the films, with all of its information will be displayed, it will display films in sets of 10. The information displayed about each film (if that information is known) are; Film name, Languages, Genres, Budget, Revenue, Colour as well as the image of the film (title picture), also |

| | |
|---|---|
| | includes the modal for the Actors and Directors where the user can click on it and see all of the actors and directors, a button to see the description of the film, a rate it button, a favourite button if the user has not already favourited the film and if they have then there will be an unfavourite button. If the page was accessed through the "Recommendation" route then it will have 2 extra buttons; "Liked Recommendation", "Disliked Recommendation".<br>Located on the navigation bar so that the user can access this from any page they are on. |
| Next page | This is a button and can only be accessed through the "Film Information" page, these will display films in sets of 10, it will redirect the user back to the "Film Information" page and display the films with the indexes which they have requested. |
| Order by attribute | A button which is only accessible on the "Film Information" page and the user can reorder the films from one of 4 attributes; Revenue, Budget, Release date, Runtime. Each attribute will have both a descending and an ascending order. |
| Description | A button which is only available through the "Film Information" page where the user can request to see the description about a film, this does not redirect anywhere, but will call a JavaScript function which will send the information to the user's screen of the content of the description of the film. |
| Actors in film | This button can only be accessed through the "Film Information" page, and will redirect the user to the "Actor Information" page, and then a modal will appear with all of the actors if there are any, the user can then click on a specific actor then will be redirected to the "Actor Information" page where the user can view the information about this one actor. |
| Directors who directed | Similar to the "Actors in film" page but just for directors. This button can only be accessed through the "Film Information" page, and then a modal will appear with all of the directors if there are any, the user can then click on a specific director then will be redirected to the user to the "Director Information" page. Sends the director id to the " Director Information" page where the user can view the information about this one director. |

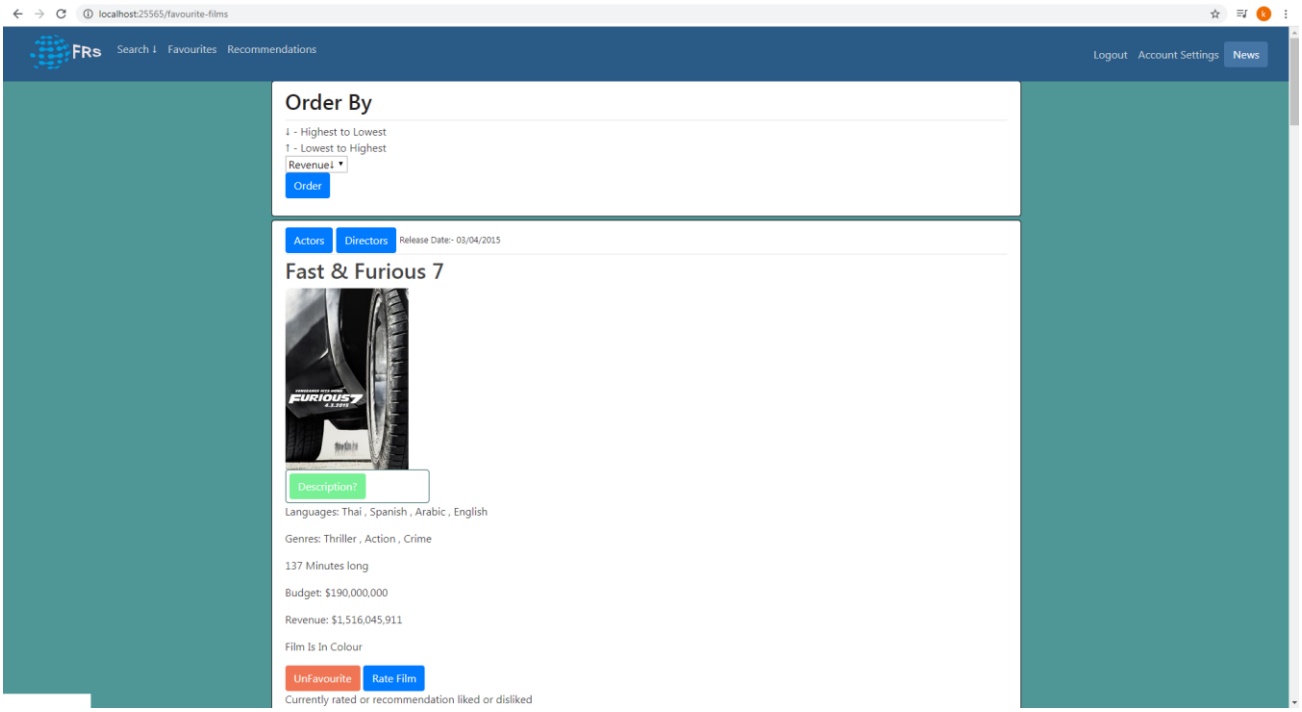
## How it will look on the website

### *Home page*

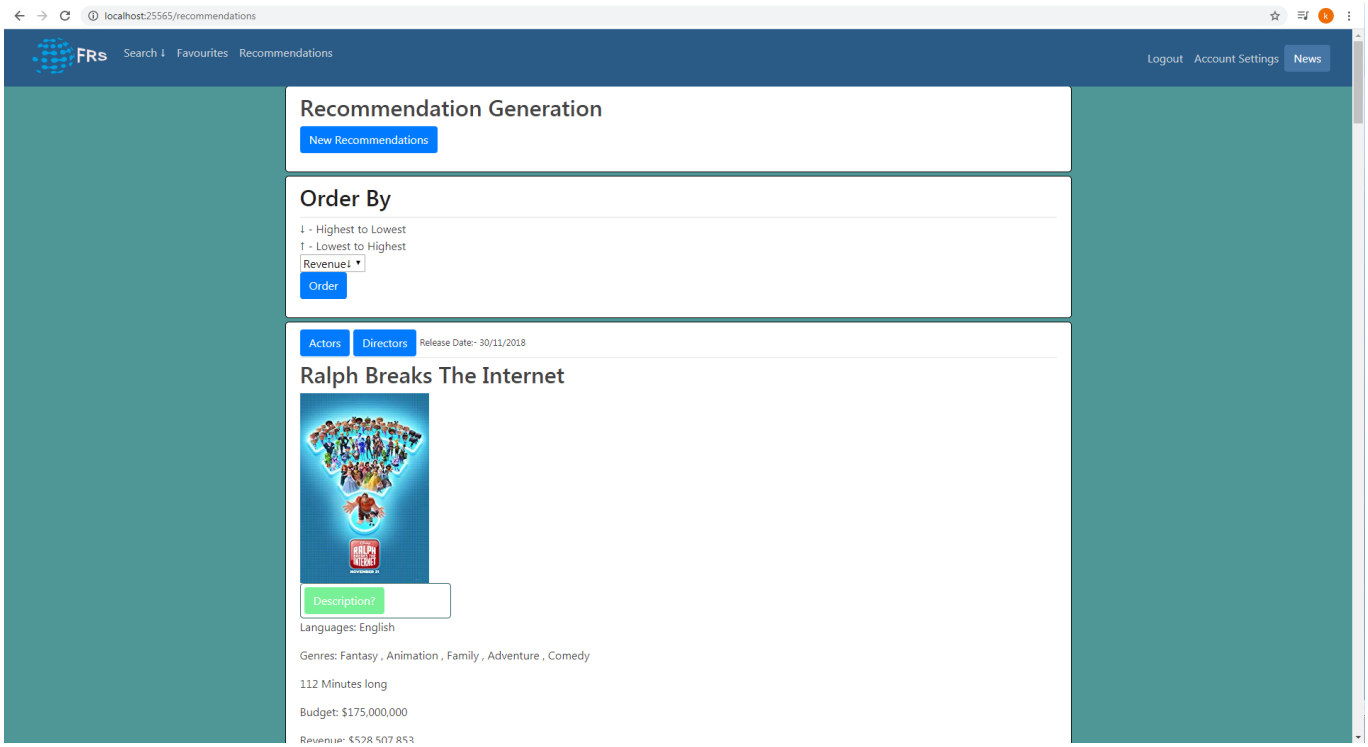
This is once the user has logged into the website

Kian

## *Favourites page*

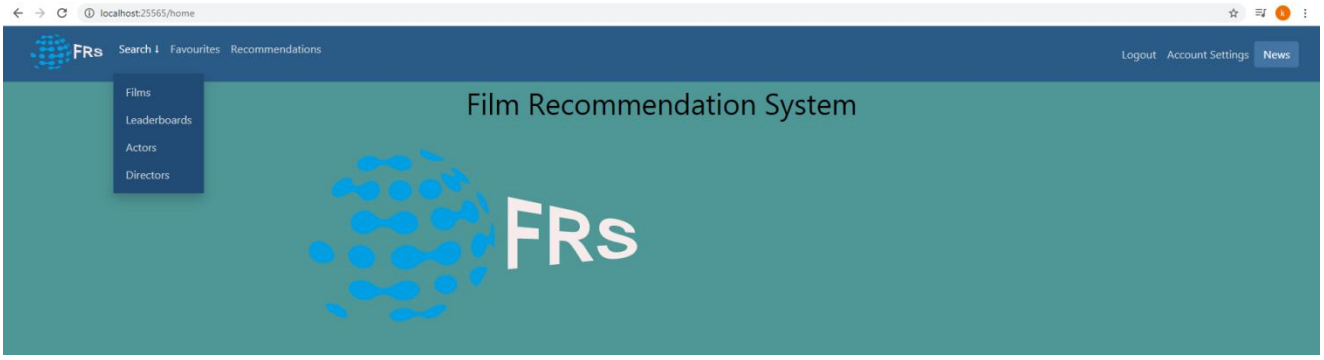



## *Recommendations*

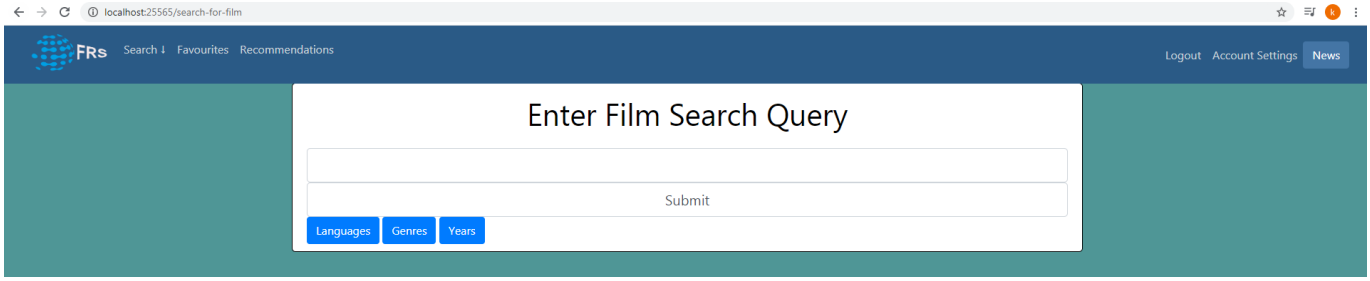



If the user clicks on the "New Recommendation" button and they have enough films rated and favourited to create a good recommendation then they will see this appear on their redirect to the home page.

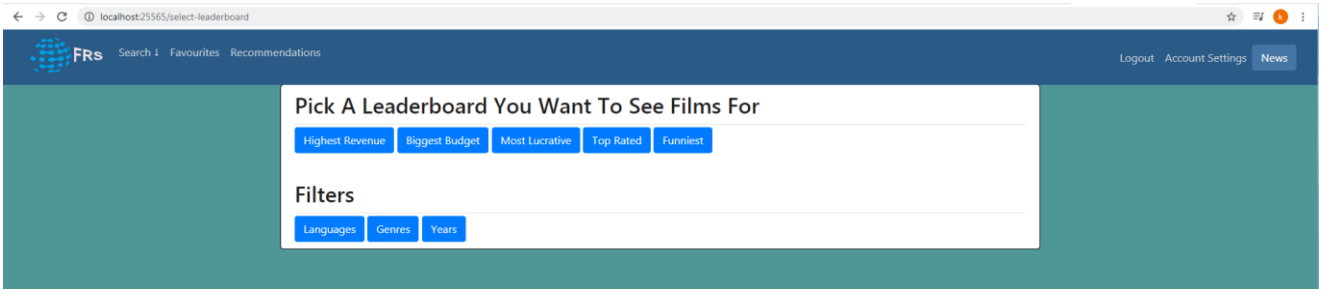

*Search dropdown menu*




*Films page*

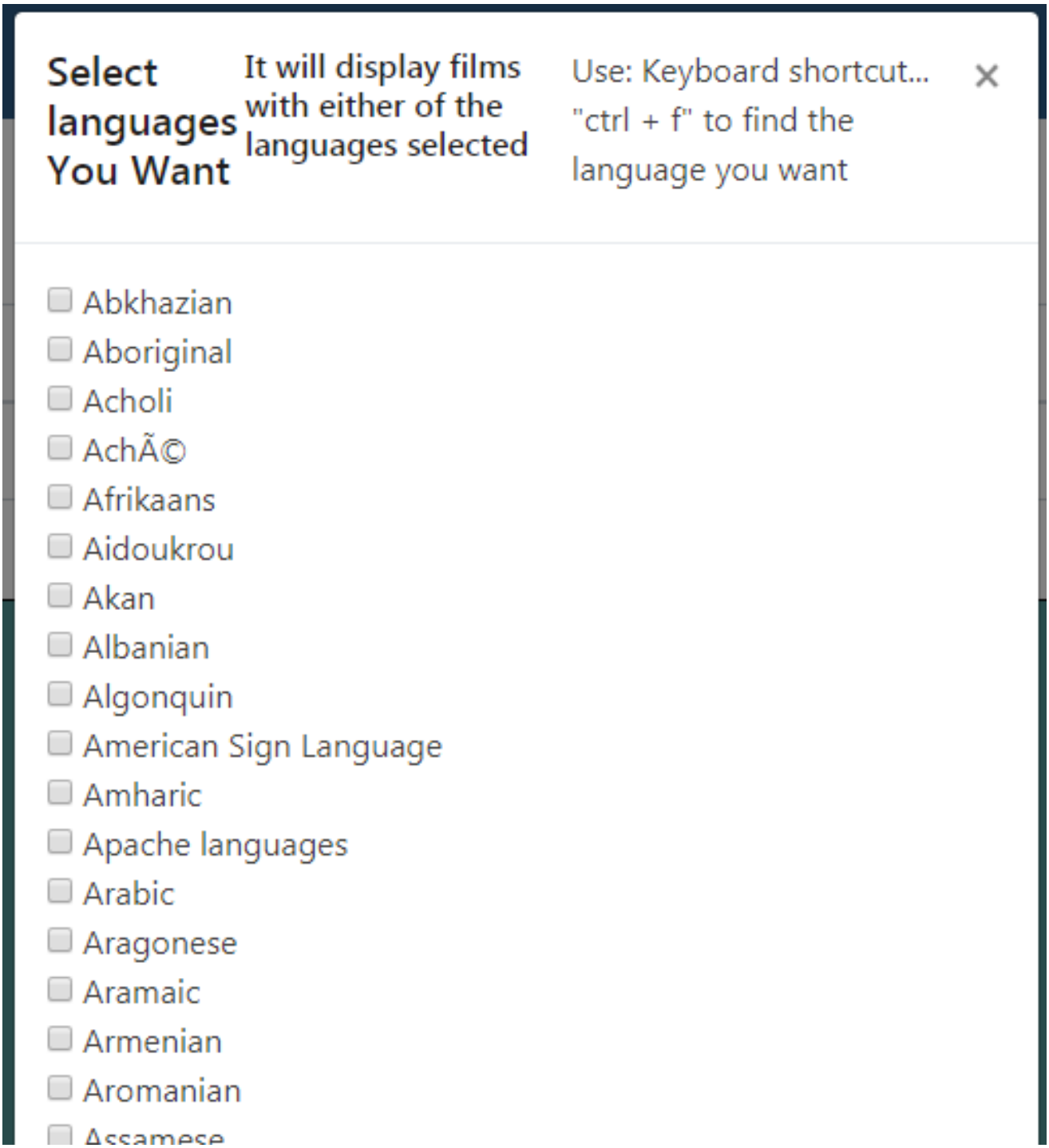


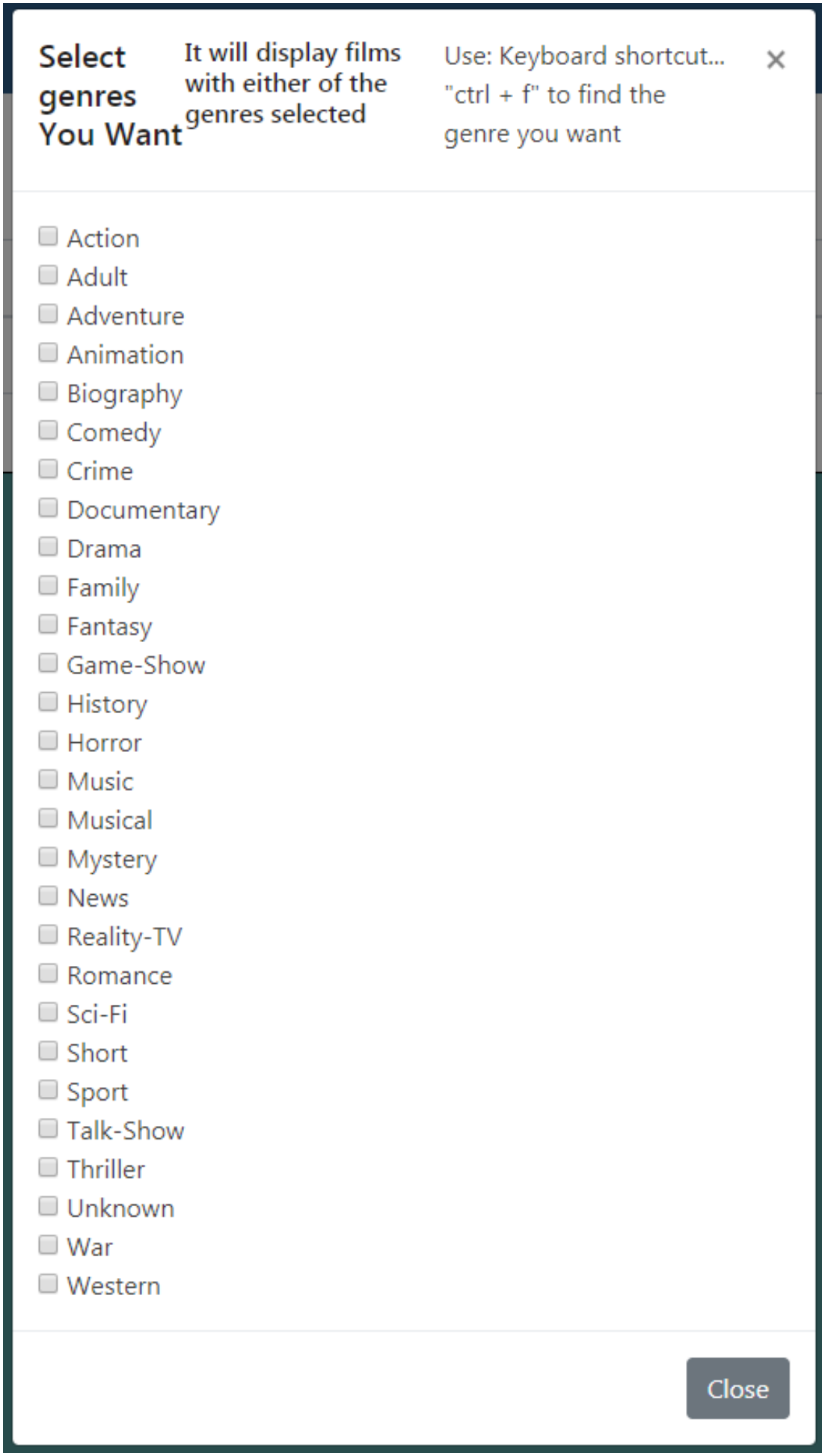

*Leaderboard page*

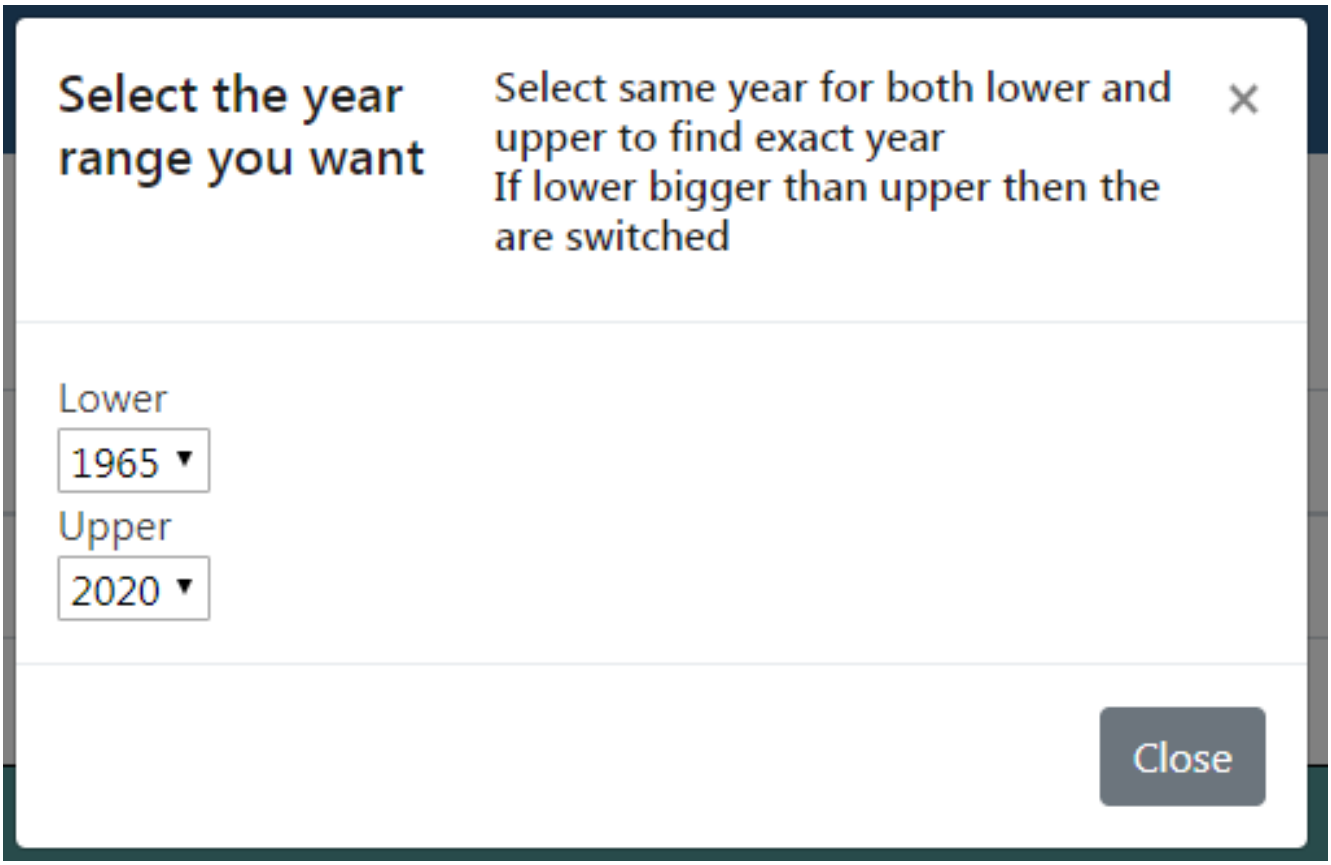


*Languages, Genres and Years Filter modals*

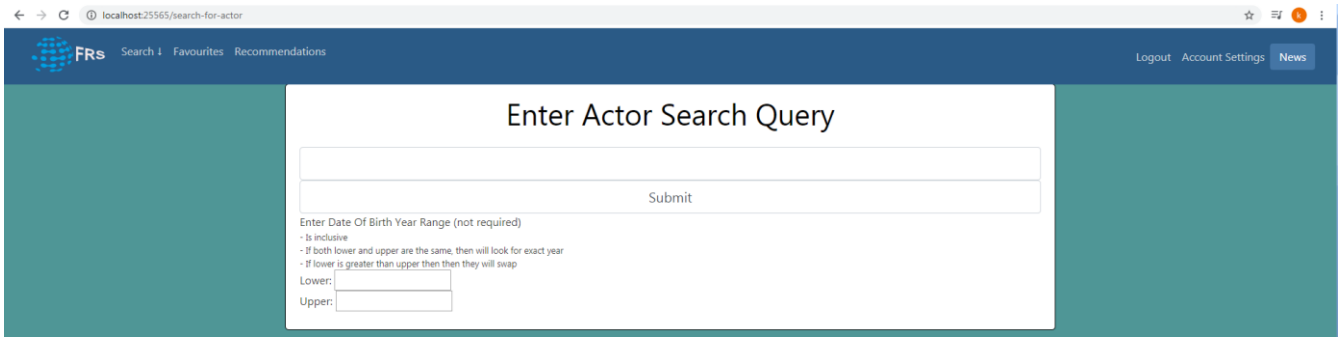
These modals appear on the Search for film page and the leaderboard page.



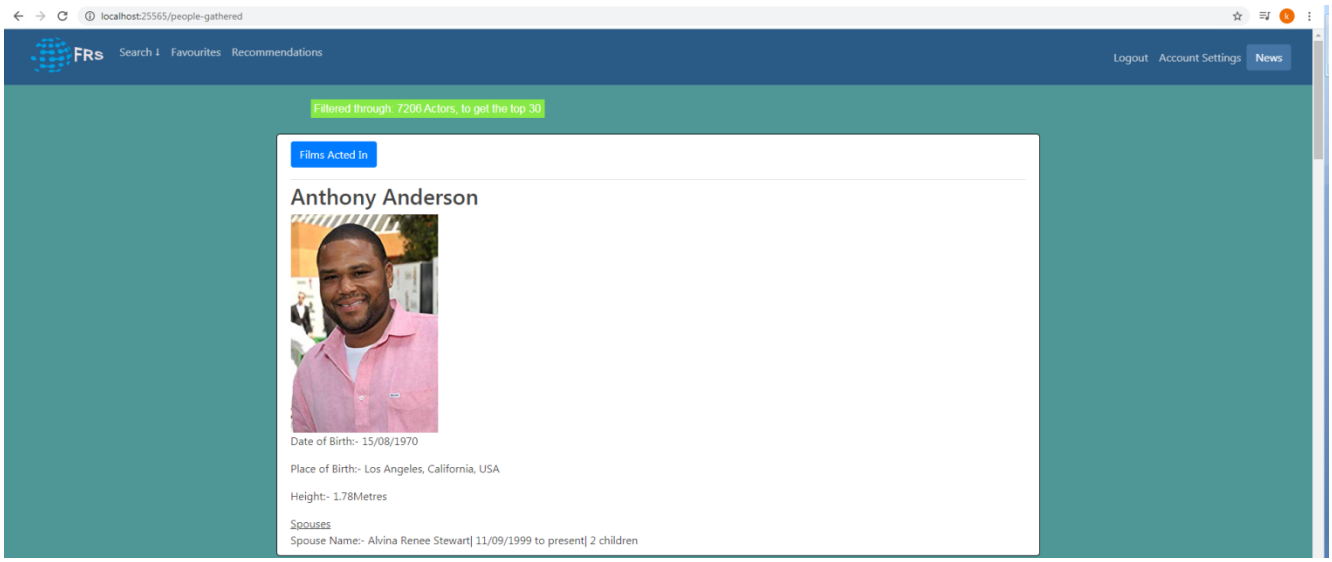

Select genres You Want
It will display films with either of the genres selected
Use: Keyboard shortcut... "ctrl + f" to find the genre you want
Action
Adult
Adventure
Animation
Biography
Comedy
Crime
Documentary
Drama
Family
Fantasy
Game-Show
History
Horror
Music
Musical
Mystery
News
Reality-TV
Romance
Sci-Fi
Short
Sport
Talk-Show
Thriller
Unknown
War
Western
Close


Select the year range you want
Select same year for both lower and upper to find exact year
If lower bigger than upper then the are switched
Lower
1965
Upper
2020
Close

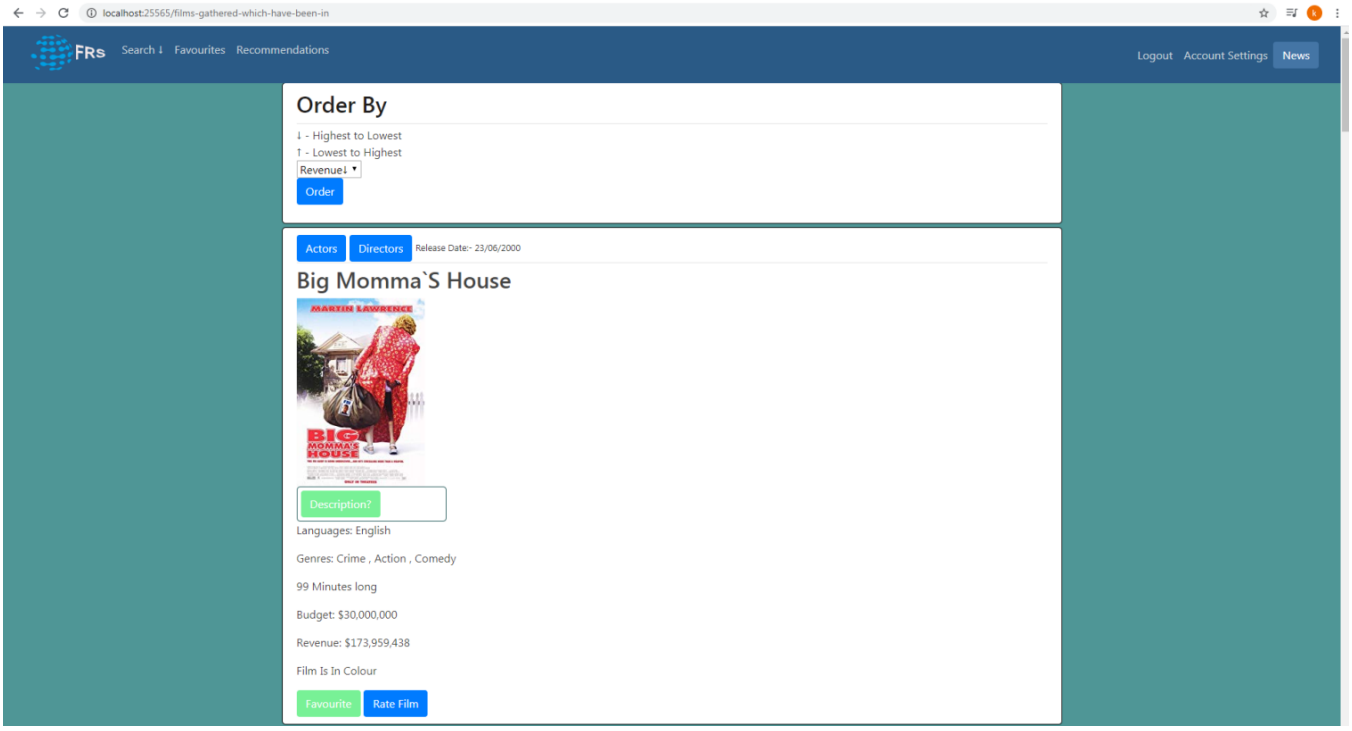
## *Actors page*



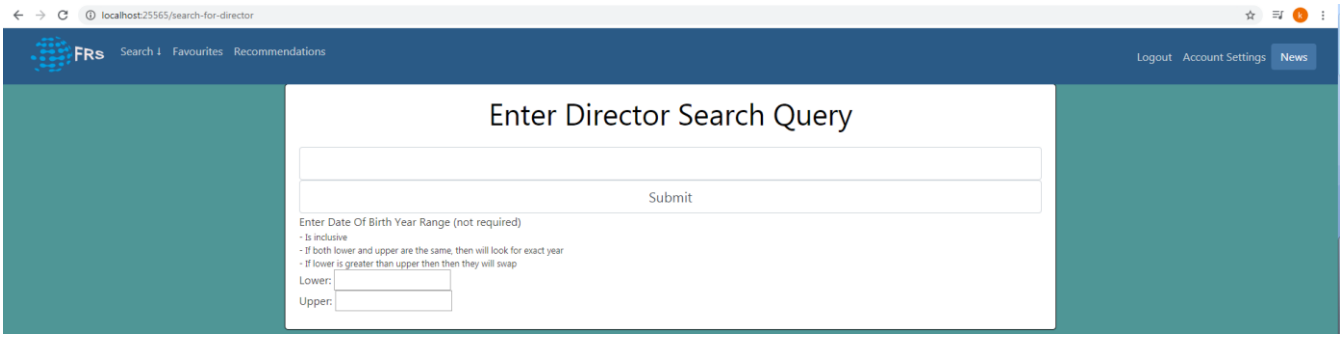

## *Actor Information Page*



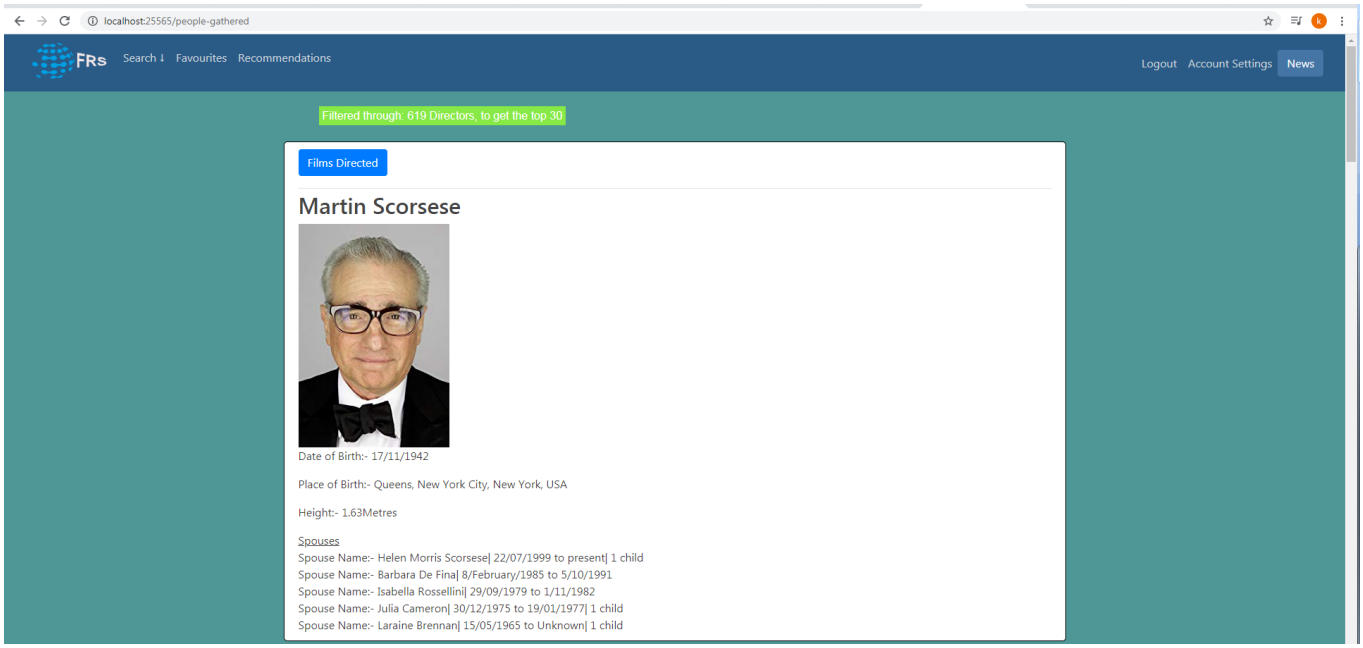

## *Films Actor Acted In*



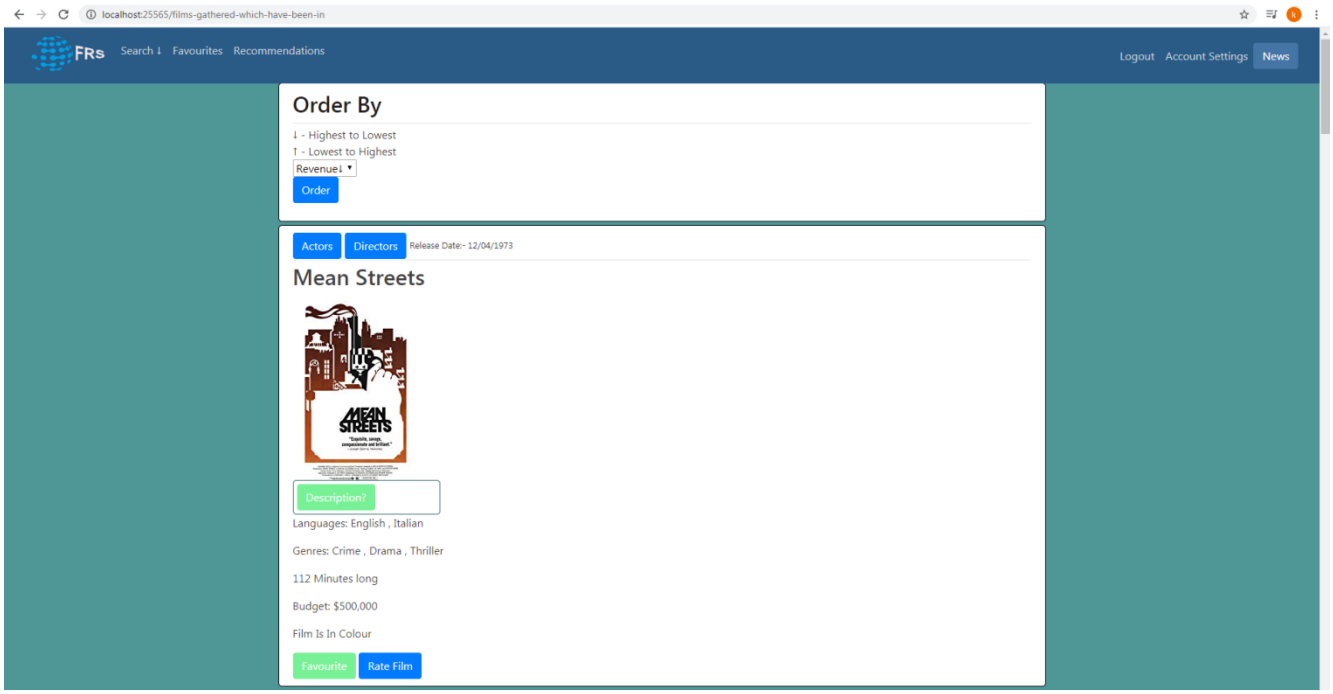
## *Directors Page*




## *Director Information Page*



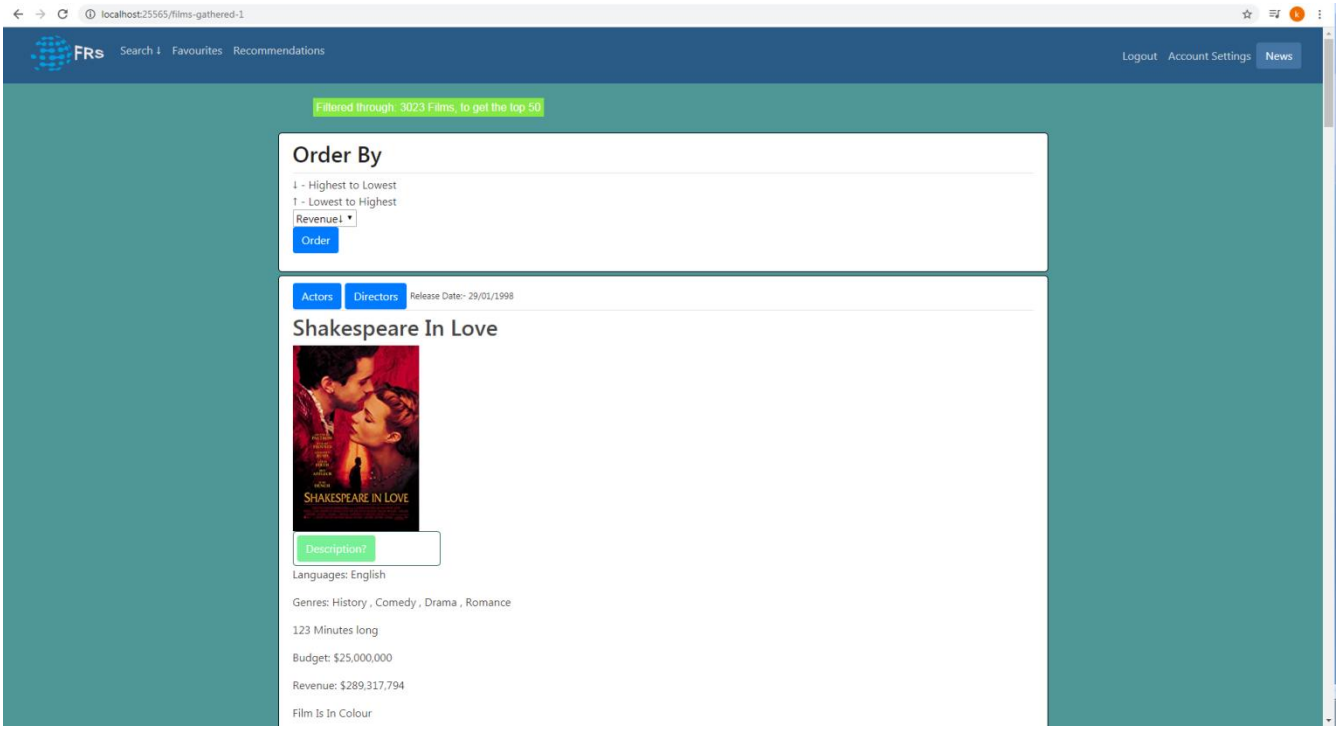

## *Films Director Directed*



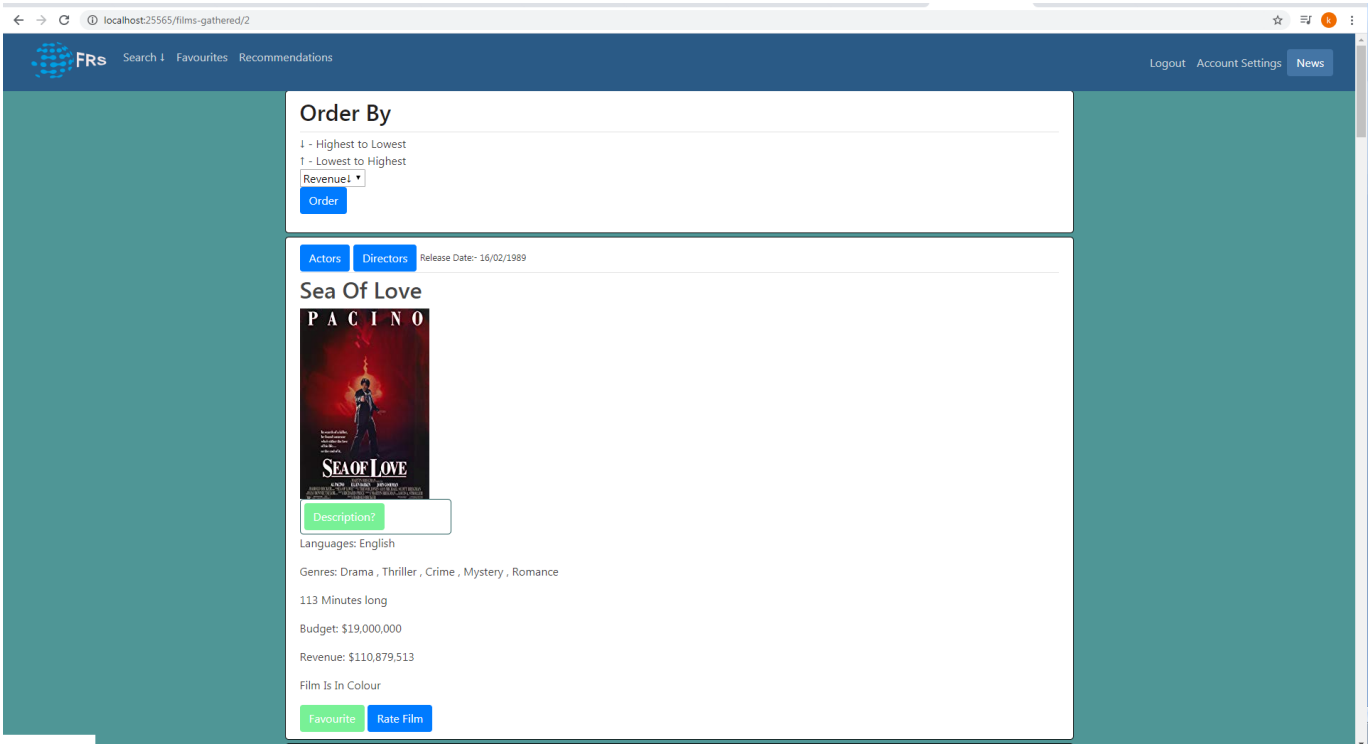
Kian

*Film Information Page*

Searched for "love"

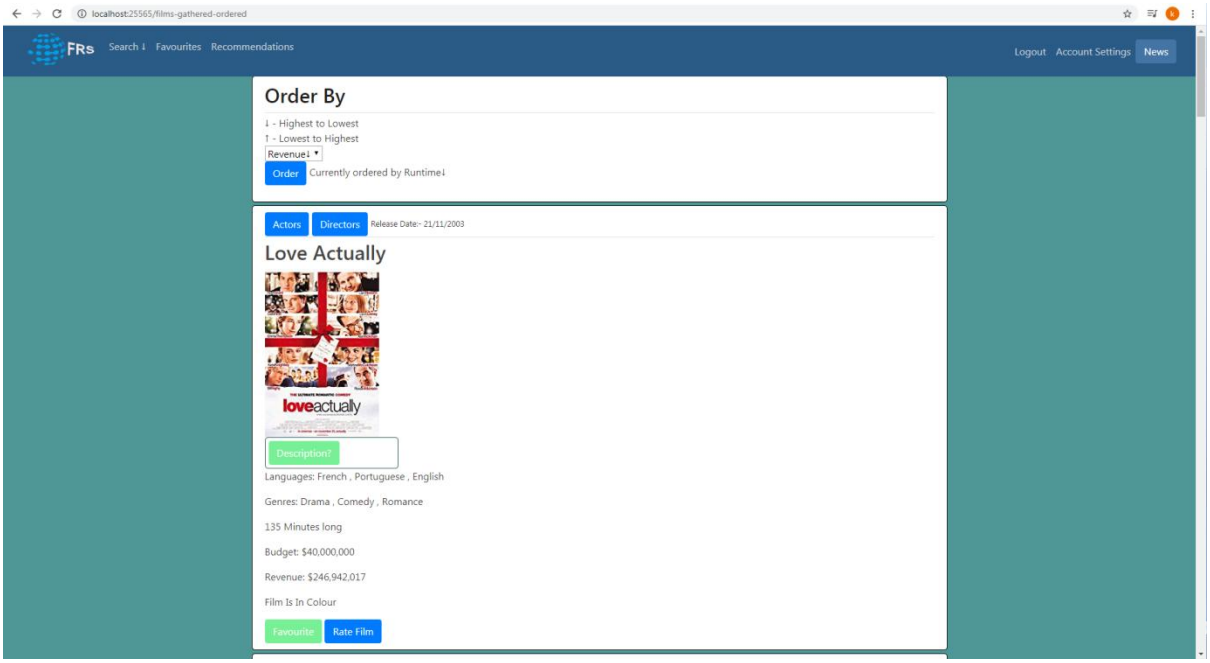



*Next page redirect*

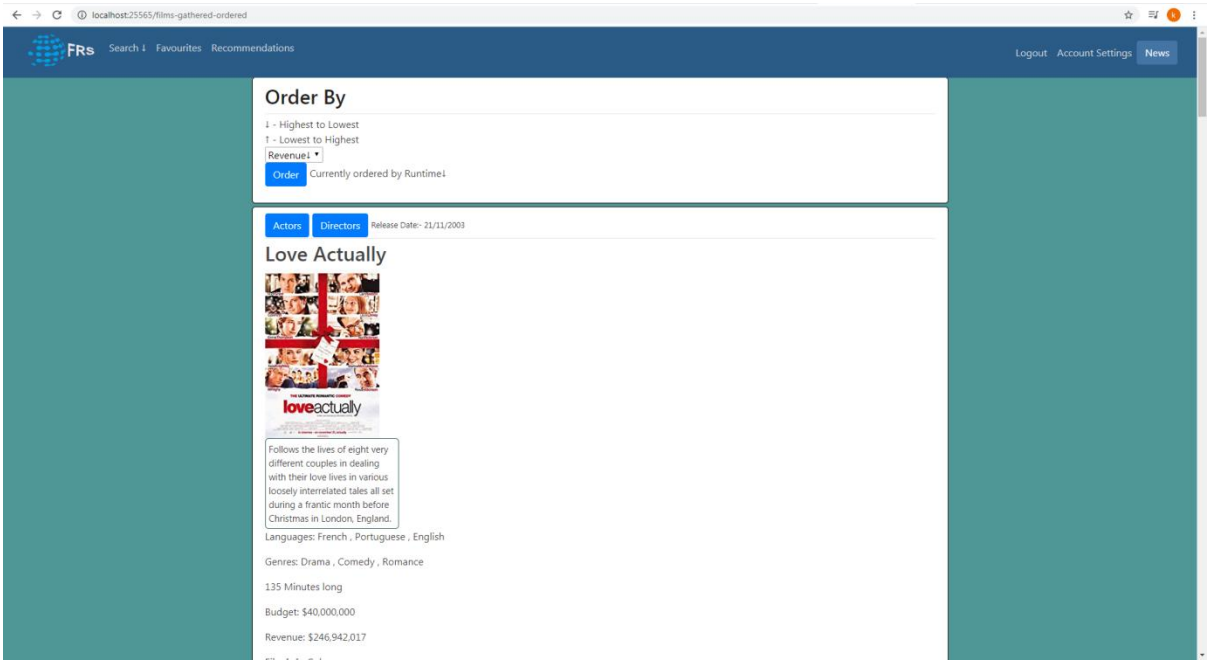


## *Order by attribute*

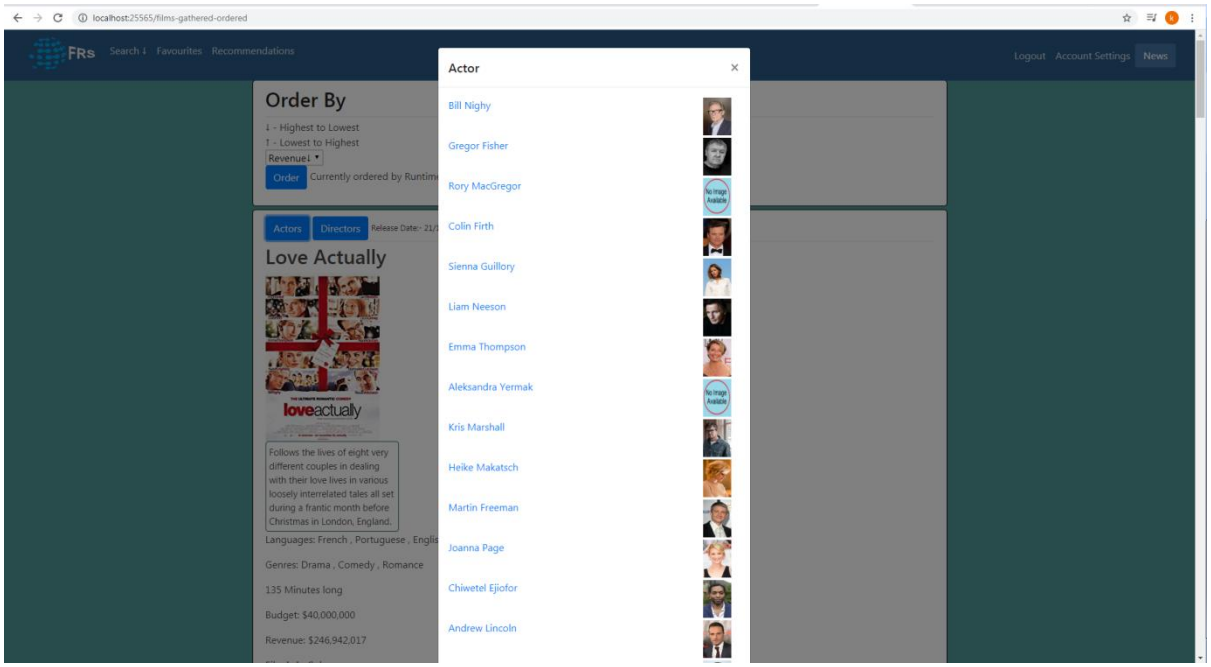
This is the first page for the search "love" ordered by the films with the longest runtime at the top



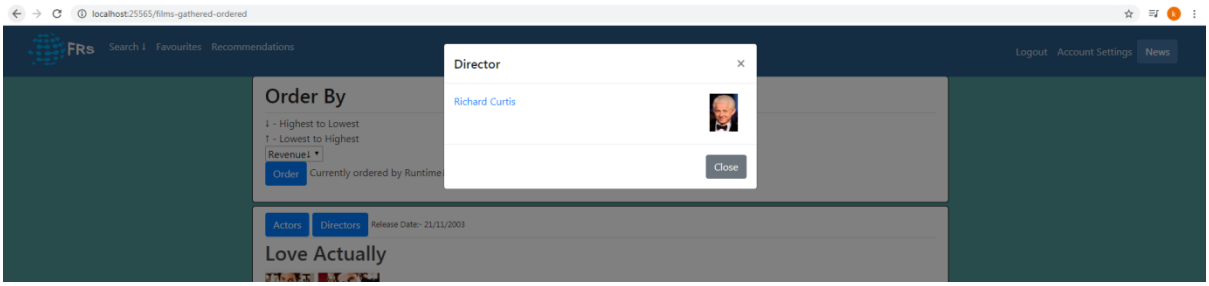

## *Description*



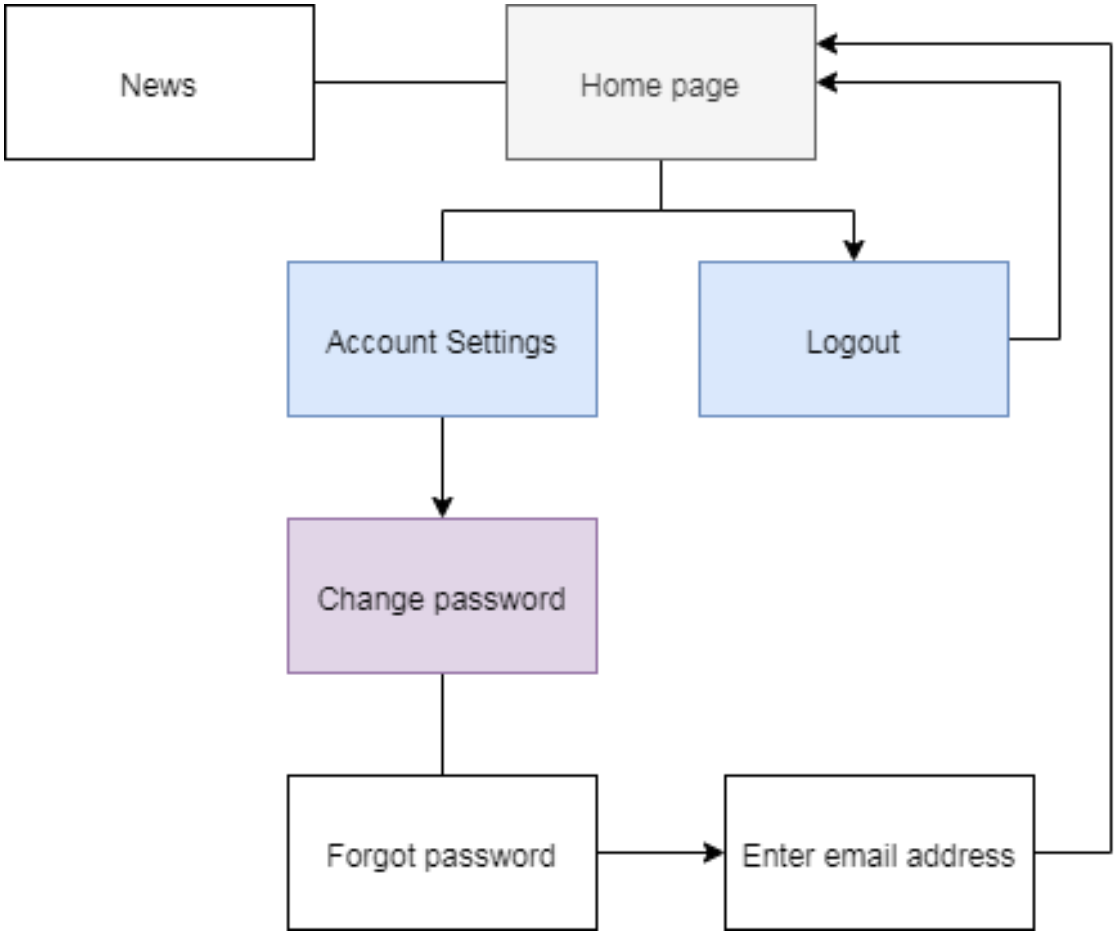
## *Actors in film Modal*




## *Directors who directed Modal*



## Right hand side




| Page name | Purpose and function |
|---|---|
| Home page | Same homepage as before, for the left hand side. If an error occurs then the user will be redirected to the homepage, as well as when they enter their email address to request their forgotten password. Located on the navigation bar so that the user can access this from any page they are on. |
| News | Same modal as when the user is not logged in. A modal which will display news about the website, important information which the user may want to know about. Located on the navigation bar so that the user can access this from any page they are on. |
| Logout | The user logouts of the website, their session and cookies will be clear (deleted). When they log back in a new recommendation for them will run in the background. Once they have logged out they will be redirected back to the homepage, telling them they have logged out successfully.<br>Located on the navigation bar so that the user can access this from any page they are on. |
| Account Settings | The name of the page where the user can change their password or if they have forgotten it then they can request their current one to be emailed to them. This is just to redirect to the change password page. |

| | |
|---|---|
| | Located on the navigation bar so that the user can access this from any page they are on. |
| Change password | The user has to enter a random one of their security question answer (the question is decided by the system), their current password, if they wish for their password to be changed. An email will be sent to the user telling them that there password has been changed. |
| Forgot password | Same as when the user is not logged in. Name of the page where the user can request their current password. this is used to redirect to the Enter email address page. |
| Enter email address | Same as when the user is not logged in. The user can enters their email address, and their current password will be emailed to them, with information telling them that it is a good idea to change their password. Once it has been emailed the user will be redirected back to the homepage |

## How it will look on the website

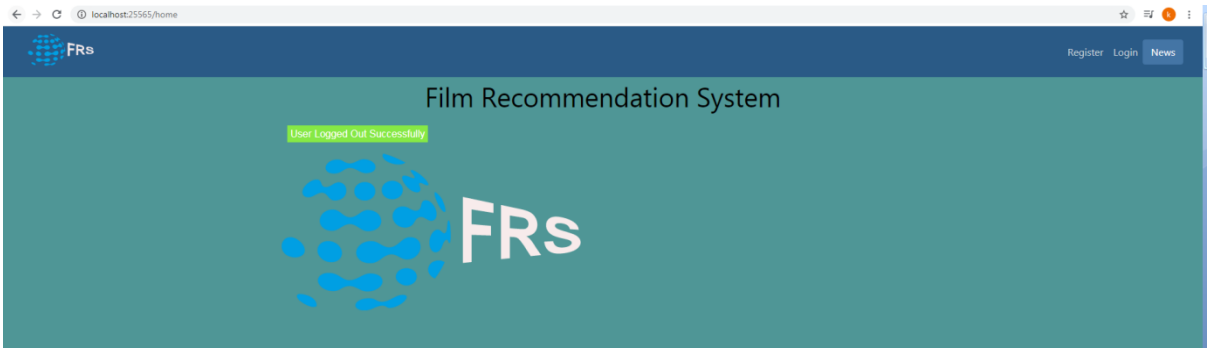
### Home page

Same as the left hand side

### News modal

Same as when the user is not logged in, but the navigation bar will look like the navigation bar when the user is logged in.

### Logout




### Account Settings

Kian

## Change password

This is when the button is pressed to change the password


## Forgot password

Same as when the user is not logged in, however the navigation bar will just look different.

## Enter email address

Same as when the user is not logged in, however the navigation bar will just look different.

## Folders, Files, Classes and Functions

In this section, I will state all of the folders, files, classes and functions which will be used in my program, along with any extra information. Each Folder and file will have a heading and a description of what they do, the folders will be separated using the files which they contain, all file types will be included in this section.

If a file is a python file and it has functions there will be a function table which will state what the function is used for the parameters it has and the data it will return, it will look like the table below.

| Function Name | Parameters: Type | Returns: Type | What does it do |
|---|---|---|---|
| | | | |

When there are classes in a file then there will be class diagrams, unless it is for the user's form which is accessed on the website and then a class diagram will not be appropriate to describe how it works.

The information below is everything which is inside of the main folder.

## app.py

This is the main python file, where the website is run from. It sets all of routes which are needed on the website for the user to access the page which they want. It calls all of the necessary functions to formulate the data which is required for the page which is being displayed. Everything is run from this file.

This table is different to the other ones as all the functions in this file have a route which is the URL how the user will access the information on the page, for this reason there is additional information in the "Parameter: Type" section, which is "Route" and "Methods" where the "Route" is the URL and the "Methods" are the methods used if the page is accessed through a form tag "<form>".

In addition, all route functions except for “home_page”, “page_does_not_excist” and “forgot_password” are wrapped and structured in try and except statements. This is since these routes could potentially cause an error and this is to remove the default error page of flask. If an error is to occur, then this code is run. It is used to redirect the user to the homepage telling them an error has occurred.

```
except:
    flash(["An error has occured, please try again.", "unsuccessful"])
return redirect(url_for("home_page"))
```

### Imports used

from flask import Flask, render_template, url_for, request, flash, redirect, jsonify, session

from datetime import datetime, timedelta

from json import load, loads

from time import time, sleep

from math import ceil

from threading import Thread

#### *Custom imports from project*

from formulas import selects_info_from_database, deletes_from_database, inserts_info_to_database

from searching.searching_algorithm import gathers_potential_spellchecked_query, searching_algorithm_gathers_film_ids_to_display

from searching.leaderboards import gathers_top_films_from_database_on_request, updates_database_with_top_rated_films, creating_and_updating_top_ratings_for_leaderboards

from searching.description_of_films import gathering_description

from recommend.formulating_recommendation_from_types import RecommendedFilms

from dictionary_creator_for_html_pages.formats_and_creates_dictionary_for_films import turning_film_information_into_dictionary

from dictionary_creator_for_html_pages.formats_and_creates_dictionary_for_people import turning_people_information_into_dictionary

from user_information.user_interactions_with_database import user_register, user_login, user_change_password, user_adds_ratings, user_security_question_to_ask

from user_information.login_register_changepassword_form import RegisterForm, LoginForm, ChangePasswordForm

from user_information.sending_emails import email_template

Functions

| Function Name | - Route<br>- Methods<br>- Parameters: Type | Returns: Type | What does it do |
|---|---|---|---|
| home_page | /<br>/home | :return: html-document: flask.render_template | The homepage of the website, the main page.<br>Everything will be redirected here if information needs to be displayed to the user.<br>It has an image (of the logo) which is linked to the "search_for_specific_film" page,<br>and will be redirected to that function/page |
| page_does_not_excist | /<page><br>param: page: str | :return: html-document: flask.render_template | When the user enters a URl which does not excist they will be redirected here<br>and warned that the page does not excist.<br>If another page has the same beginning part of the URL, but the parameter section is invalid, then page will not be displayed,<br>and instead the user will be redirected to the main "home_page" route<br>It will render the homepage html with a parameter telling the user that "x" page does not excist. |
| search_for_specific_film | /search-for-film | :return: html-document: flask.render_templat | Page where the can user search for a film |
| spellchecks_users_film_query | /unused-url-1 | :return: javascript-response: flask.jsonify | Spell checks the user's query, aslong as it is not too long.<br>Based on what the user has entered will show potential spell checked results<br>This function is called from within a javascript function so the URL route is never used,<br>and will just be used to run code |
| gathering_film_search_query_information | /unused-url-2 | :return: javascript-response: flask.jsonify - no information in it | This function is called from a javascript function, once the user has clicked on the spell checked<br>query, it will gather the criteria, and format it, then will set the values equal to the user's session |

| | | | |
|---|---|---|---|
| | | | so that it can be accessed in the "films_gathered" route.<br>If there is no corrected query then it will instantly go to this function without the user needing to click anything,<br>Here it will edit and check the user's query to make sure it does not contain unwanted characters,<br>which could cause trouble looking up the query in the database, it will also check the length aswell |
| films_gathered | /films-gathered-1 | :return: html-document: flask.render_template | Gathers all of the top 50 film ids for the query the user has entered, with the filters applied.<br>for the top 10 films from the algorithm, it will fetch the information about those and display<br>it on the website for the user to view.<br>This page can only be accessed through the route "search_for_specific_film" by entering a valid query<br>if it is invalid then the user will be redirected to the home page telling them an error has occured. |
| displaying_10_films | /films-gathered/<page_number><br><br>Method = POST<br><br>param: page_number: str | :return: html-document: flask.render_template | Renders the template for the next 10 films for the user,<br>Similar to the "films_gathered" route but the film ids have already been gathered<br>so just finding the extra information about the films |
| order_films_from_requested_attribute | /films-gathered-ordered<br><br>Method = POST | :return: html-document: flask.render_template | This route will sort the films by the attribute requested by the user, and returns the page reordered.<br>The information is gathered from a html "form" tag |
| description | /unused-url-4 | :return: javascript-response: flask.jsonify | Gathers the description of the film id specified, it is called from within a javascript function |
| favourite_a_films | /unused-url-5 | :return: javascript-response: flask.jsonify - no information in it | Adds the film id to the database with the user id, in the Favourites table,<br>this function is called from within a javascript functions |
| remove_a_favourited_films | /unused-url-6 | :return: javascript-response: flask.jsonify - no information in it | Removes the favourited film from the database for the user, in the Favourites table,<br>this function is called from within a javascript functions |
| adds_ratings | /unused-url-7 | :return: javascript-response: flask.jsonify | Adds or updates the ratings of a film<br>this function is called from within a javascript functions |
| favourite_films | /favourite-films | :return: html-document: flask.render_templ | Displays the user's favourited films, only the first 10, they can select to see the next page if they wish, |

| | | | |
|---|---|---|---|
| | | ate | and they will be redirected to the "displaying_10_films" route |
| recommendations | /recommendations | :return: html-document: flask.render_template | Displays the recommendations for the user,<br>it will order the film_ids for that, the highest priority films are nearer the front of the list,<br>it will also make sure that if there are any black and white recommendations,<br>then they are evenly distrubuted between each page of the recommendations.<br>to go to the next page of the recommendation the route "displaying_10_films" will be called |
| recommendation_liked_or_not_liked | /unused-url-8 | :return: javascript-response: flask.jsonify - no information in it | The user comments on if they like the recommendation or not,<br>then adds the information to the database<br>this function is called from within a javascript functions |
| saving_the_recommendations_to_database | /unused-url-9 | :return: flask-redirect: flask.redirect | Creates new recommendations for the user by calling the class "RecommendedFilms",<br>then it redirects back to the "recommendations" route to display all of the new recommendations |
| leaderboard | /select-leaderboard | :return: html-document: flask.render_template | The user can, select which leaderboard they want to see,<br>once they select the leaderboard they want then they will be redirected<br>to the "displaying_the_leaderboard_requestion" route, where the films will be displayed |
| displaying_the_leaderboard_requestion | /displaying-leaderboard<br><br>Method = POST | :return: html-document: flask.render_template | Displays the information about the leaderboard requested by the user,<br>with all of the filters applied, it will gather all of the film ids,<br>in order of the most relevent to the leaderboard.<br>The user can access the "displaying_10_films" route,<br>to see more of the films for the current leaderboard they are looking at |
| films_person_has_been_associated_with | /films-gathered-which-have-been-in<br><br>Method = POST | :return: html-document: flask.render_template | This gets all of the films which the actors and directors have been in or directed,<br>Gathers all of the film ids and displays the first 10<br>The user can access the "displaying_10_films" route,<br>to see more of the films which the actor has been in or the director has directed |
| information_for_single_person | /person-information/<is_actor>/<person_id> | :return: html-document: flask.render_template | This route is called when the user is looking at the actor or director modal, and clicks on a person<br>This function is called from the html film "display_film_information.html"<br>and will display all of the information about an |

| | | | |
|---|---|---|---|
| | is_actor: str<br><br>person_id: str | | actor or a director requested by the user.<br>It gathers all of the important information from the database for the person, and displays it<br>in a nice format for the user to read. |
| search_for_person | /search-for-<is_actor><br><br>param: is_actor: str | :return: html-document: flask.render_template | Loads the page where, the user can search for either an actor or director,<br>if the is_actor == actor then it is an actor if it is director then its a director.<br>if it does not equal either actor or director then it will redirect to the home page saying that option does not exist. |
| people_gathered | /people-gathered<br><br>Method = POST | :return: html-document: flask.render_template | Gathers all of the top 30 actor or director ids for the query the user has entered, with the filters applied.<br>for the top 10 actor or directors from the algorithm (most relevent), it will fetch the information about those and display<br>it on the website for the user to view.<br>This page can only be accessed through the route "search_for_person" by entering a valid query<br>if it is invalid then the user will be redirected to the home page telling them an error has occured. |
| displaying_10_people | /people-gathered-more<br><br>Method = POST | :return: html-document: flask.render_template | Renders the template for the next 10 actors or directors for the user to view,<br>Similar to the "people_gathered" route but the people ids have already been gathered<br>so just finding the extra information about the actors or directors |
| register | /register<br><br>Method = GET, POST | :return: html-document: flask.render_template | The html form page where the user can register to the website,<br>they will enter their information into a form from flask_wtf.FlaskForm,<br>and that information once the submit button has been pressed, can be accessed in this route,<br>once the submit button has been pressed the information can be processed to the database,<br>and the user will be registered on the website.<br>Once they have registered an email will be sent to them giving them a discloser of important<br>information they must know before using the website. |
| login | /login<br><br>Method = GET, POST | :return: html-document: flask.render_template | The html form page where the user can login to the website,<br>they will enter their information into a form from flask_wtf.FlaskForm,<br>and that information once the submit button has been pressed, can be accessed in this route,<br>once the submit button has been pressed the information can be processed to the database,<br>and checked to see if it matches the current |

| | | | |
|---|---|---|---|
| | | | information if it does then the user can login to the website.<br>If the user was on a page which is restricted to only users which are logged in and they get redirected<br>here then once they login they will be redirected back to that page but will be logged in |
| logout | /unused-url-10 | :return: flask-redirect: flask.redirect | Logs out the user from the website if they are already logged in |
| account_settings | /account-settings<br><br>Method = GET, POST | :return: html-document: flask.render_template | The html form page where the user can change their password,<br>they will enter their information into a form from flask_wtf.FlaskForm,<br>and that information once the submit button has been pressed can be accessed in this route,<br>once the submit button has been pressed the information can be processed to the database,<br>to make sure that it matched the current information, if it does then the user's password will be changed<br>and an email will be sent to them informing them, of this change |
| forgot_password | /forgot_password | :return: html-document: flask.render_template | Renders the page where the user can enter their email address to get their current password<br>emailed to them, once they click submit the "emails_the_user_current_password" function will run<br>and email the user their password.<br>No validation is required since the html page does not take in any jinja2 parameters as variables.<br>It is just loading a plain html page. |
| emails_the_user_current_password | /unused-url-11<br><br>Method = POST | :return: flask-redirect: flask.redirect | Once the user has entered their email on the "forgot_password" page, the program,<br>will redirect here and an email will be sent to the user<br>depending on if the email excists or not |

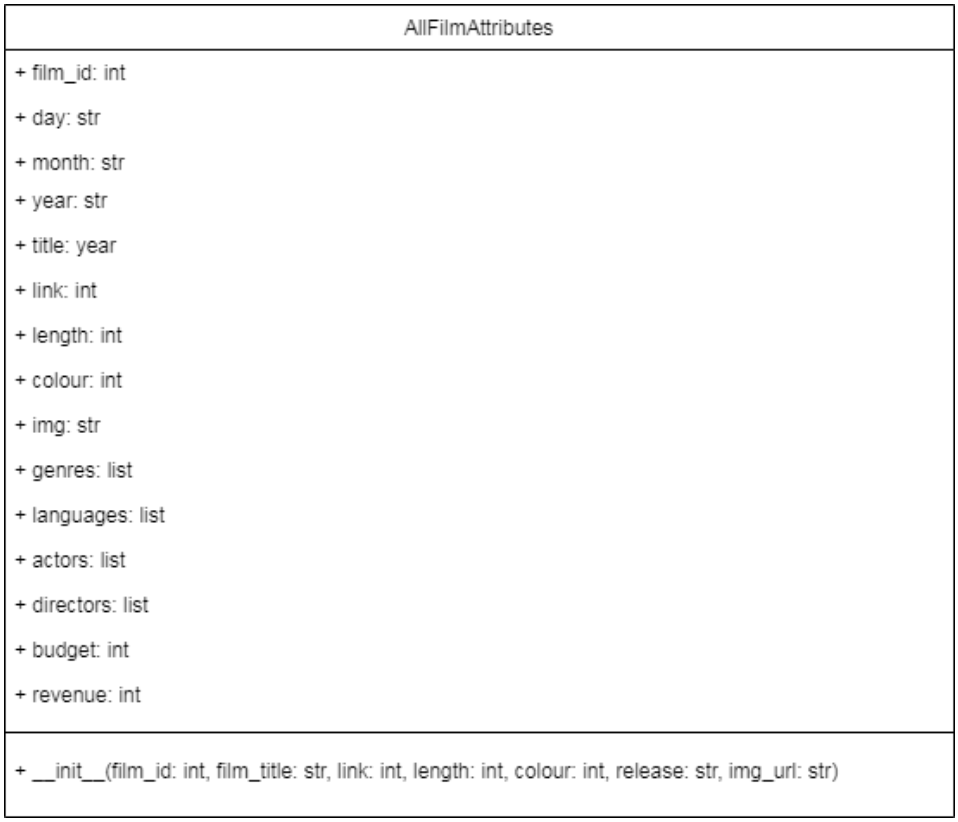
## classes.py

There are 3 classes in this file and no functions, the classes are used just to assign parameters to attributes in the function, so that it will be easier to access and organise information in other files. This section will only have 3 class diagrams as there are only 3 classes. This file does not have any imports

### Classes

#### *AllFilmAttributes*

Parameters are passed into the class and assigned to the attributes.
If the attribute, isn't assigned a variable from the parameters then it is given a default value of an empty list or a 0 is given



### *AllFilmAttributesForRecommendation*

Parameters are passed into the class and assigned to the attributes.
This class is used only in the recommendations, as some of the attributes from the "AllFilmAttributes" are not neded, and languages and genres are already gathered when the class is called, so they can be initially added

| FilmAttributesForRecommendation |
| --- |
| + film_id: int<br>+ length: int<br>+ colour: int<br>+ release: str<br>+ languages: list<br>+ genres: list |
| + __init__(film_id: int, length: int, colour: int, release: str, language_list: list, genre_list: list) |

### *AllFilmAttributes*

Parameters are passed into the class and assigned to the attributes.

| AllActorAndDirectorAttributes |
| --- |
| + person_id: int<br>+ name: str<br>+ link: int<br>+ height: str<br>+ dob: str<br>+ death: str<br>+ birth_place: str<br>+ spouse: str<br>+ img: str |
| + __init__(person_id: int, name: str, link: int, dob: str, birth_place: str, death_date: str, height: str, spouse: str, img_url: str) |

## formulas.py

In this python file, the most frequently used functions (modules) are created. The majority of the functions defined in this file are used across all of the files in the project directory. Only in this file is a connection to the database created, as when the program wants to interact with the database, a function from inside of this file is called.

### Imports used

from math import sqrt

from sqlite3 import connect

### Functions

| Function Name | Parameters: Type | Returns: Type | What does it do |
|---|---|---|---|
| mean_and_sd | :param: list_of_values: list | :return: mean: float : return: sd: float | Calculates the variance using the formula SUM(X*X)/n - mean*mean, where X is each individual element in the array of data Once varience is found square roots it for the standard deviation (sd) |
| factorial | :param: val: int | :return: factorial: int | Calculates the factorial of the paramater |
| combination_formula | :param: total: int :param: position: int | :return: events: int | nCr, combination formula, used to calculate permutation, the different combinations which can occure. n is the number of items (total) r is the number of items being chosen (position) this uses the pascal triangle, and n would be the line number and r would be the positon along that line |
| binomial_distribution | :param: lower: int :param: upper: int :param: trial: int :param: probability: float | :return: value: float | Calculates the probability of getting a value between the lower and upper value based on the amount of trials if lower is the same as upper then it will work out the proability of getting 1 value. the lower and upper bounds are inclusive Additional info, trial * probability, is the mean value, the probability is worked out around this value. |
| binary_search | :param: values: list :param: target: str :param: startpoint: int :param: endpoint: int | :returns: False, if the target is not in the values list, but otherwise it will return itself, and go through the algorithm | Rular binary search OLog(n), justing using a recursive algorithm, just using recursion. |

| | | | |
|---|---|---|---|
| | | again, once the target has been located then it will return the position | |
| selects_info_from_database | :param: fields: str<br>:param: table: str<br>:param: where_statement: str | :return: information: list | Gets the information from the database for the fields entered based on the where condition specified. |
| inserts_info_to_database | :param: field: str<br>:param: table: str<br>:param: data: list - 2d list of tuples | | Inserts all of the data specified into the database |
| deletes_from_database | :param: table: str<br>:param: where_statement: str | | Deletes data from table where it meets the where condition |
| updates_database | :param: field: str<br>:param: table: str<br>:param: where_statement: str<br>:param: data: tuple | | Updates the fields specified where it meets the where criteria.<br>The where statement will look like "Field=? AND Field2=?", and data with all all the items to add to the database |
| groupings | :param: data_to_group: list | :return: grouped_data: dict | Groups together all of the information which is given as a paramater, where data_to_group is a 2d list of tuples with each tuple having 2 values,<br>the first value in the tuple is the film id and may occur more than once so this will group them together in a dictionary |

## database folder

This folder is used just to house the database, it has no other purpose except to separate the database from the rest of the files.

### MainDB.db

This is the where all of the information about the films, actors, directors and users are stored. Along with the user's recommendations, ratings and favourites. The information for the leaderboards are also stored in here.

# user_information folder

All of the code which is needed for the user to interact with the database is housed in this folder, along with the code which is needed to do anything related to the user. There are also ".txt" and ".json" files stored in this folder, and these contain information which will be played to the user when they need it.

## login_register_changepassword_form.py

A file of classes for the user to enter information in on the website when they register, login or change their password.

These are all inherited from the "flask_wtf.FlaskForm" module.

### *Imports used*

from datetime import date

from json import loads

from flask_wtf import FlaskForm

from wtforms import StringField, PasswordField, SubmitField, BooleanField, SelectField

from wtforms.validators import DataRequired, Length, Email, EqualTo

from wtforms.fields.html5 import DateField

### *Form field - Types*

StringField - Used when the user is entering a string, does not do anything special just allows the user to have an error message if an error applies if validation was set.

PasswordField - Used when the user is entering a password, so that the data which they are entering does not appear.

SelectField - A dropdown menu for the user to select from.

DateField - A calendar for the user to select the date from which they want.

SubmitField - Once the user is happy with all of the information which they have entered then they can press this button which will send all of the information to the python function, so that the information can be sent to the database or checked against the database to give the user the response which they want.

### *Form field- Validators*

DataRequired() - The user must enter something, if they do not then it will tell them to enter information

Email() - The information which the user has entered, must be in the format of an email address, if it is not then it will tell the user to enter an email.

Lenght(min=?, max=?) - The information which the user has entered must be between a certain range, they are inclusive

EqualTo() - Has to be the same as another piece of information stated.

*RegisterForm*

The instance of the class FlaskForm is being passed in,

so  all of it's information can be accessed from inside this class.

This class "RegisterForm" is then passed into a html file and using Jinja2 the information is accessed

It will be able to take user inputs and store them so that they can be used later on and added to the database

it has inbuilt validation up to a certain point to check if the user's input meets the certain criteria wanted.

Once the user is happy with the information they have entered they can hit submit and the form is sent to a

python function where the data is processed and added to the database

**security_questions -** Is a JSON file with all of the security questions to ask the user, there are 15 and split up into 3 categories, 5 for each question.

| Variable name | Field Type | Validators<br>Extra Information(EI) |
|---|---|---|
| name | StringField | DataRequired() |
| email | StringField | DataRequired()<br>Email(message="Please enter a valid email format") |
| username | StringField | DataRequired()<br>Length(min=3, max=12) |
| password | PasswordField | DataRequired(), Length(min=6, max=128) |
| confirm | PasswordField | DataRequired()<br>EqualTo(<br>"password", message= "Please make sure you re-enter the exact same password as above") |
| security_question_answer_box1 | StringField | DataRequired() |
| security_question_answer_box2 | StringField | DataRequired() |
| security_question_answer_box3 | StringField | DataRequired() |
| security_question_options1 | SelectField | EI - choices = security_questions["SecurityyQuestion1"] |
| security_question_options2 | SelectField | EI - choices = security_questions["SecurityyQuestion2"] |
| security_question_options3 | SelectField | EI - choices = security_questions["SecurityyQuestion3"] |
| dob | DateField | DataRequired()<br>EI - default = date.today (which is the current date of today) |
| submit | SubmitField | |

*LoginForm*

The instance of the class FlaskForm is being passed in,

so  all of its information can be accessed from inside this class.

This class "LoginForm" is then passed into a html file and using Jinja2 the information is accessed

It will be able to take user inputs, do that they can be compared to information already in the database.

If the information is the same then the user can login, else they will be told their information is wrong,

this validation is not done inside of this class but instead with python then the user is redirected to the home page.

it has inbuilt validation up to a certain point to check if the user's input meets the certain criteria wanted.

Once the user is happy with the information they have entered they can hit submit and the form is sent to a

python function where the data is processed and added to the database

| Variable name | Field Type | Validators |
|---|---|---|
| username | StringField | DataRequired |
| password | PasswordField | DataRequired |
| submit | SubmitField | |

*ChangePasswordForm*

The instance of the class FlaskForm is being passed in,

so all of it's information can be accessed from inside this class.

This class "ChangePasswordForm" is then passed into a html file and using Jinja2 the information is accessed

It will be able to take user inputs, do that they can be compared to information already in the database.

If the information is the same as the current info, then the user can continue and their password can be changed,

else it will redirect the user to the home page telling them that the information which they have entered is incorrect,

the comparison is done inside of the pythons script, as some of the inbuilt validiation does not support this.

Once the user is happy with the information they have entered they can hit submit and the form is sent to a

python function where the data is processed and added to the database

| Variable name | Field Type | Validators |
|---|---|---|
| security_question | StringField | DataRequired |
| password | PasswordField | DataRequired |
| new_password | PasswordField | DataRequired<br>Length(min=6, max=128) |
| submit | SubmitField | |

### encryption.py

The purpose of this file is to house the functions which are used to encrypt and decrypt the user's password.

#### *Imports used*

from cryptography.fernet import Fernet

#### *Functions*

| Function Name | Parameters: Type | Returns: Type | What does it do |
|---|---|---|---|
| encrypt | :param: data: str<br>:param: key: str | :return: encrypted_data: str | Takes in the data which is wanted to be encrypted and the key (salt) used to encrypt it.<br>Using the cryptography module to encrypt the data |
| decrypt | :param: encrypted_data: str<br>:param: key: str | :return: decrypted_data: str | Takes in the encrypted data which is wanted to be decrypted and the key (salt) used to encrypt it originally,<br>so that it can be decrypted to its original form.<br>Using the cryptography module to dencrypt the data |

## sending_emails.py

The purpose of this file is to house the function which is used to send emails to the user, it will gather the contents of the email to send from a file and fill in the missing pieces of data with specifics from the user.

### *Imports used*

from smtplib import SMTP

from json import load

#### Custom imports from project

from formulas import selects_info_from_database

from user_information.encryption import decrypt

### *Functions*

| Function Name | Parameters: Type | Returns: Type | What does it do |
|---|---|---|---|
| email_template | :param: user_data: int OR login_register_changepassword_form.RegisterForm<br>:param: email_type_to_send: str | :return: correct_email_for_forgot_password: bool<br>- this is only used when the user has forgot their password to check if the user has entered an email address which excists or not | Sends emails to the user depending on if they have created their account, changed their password or forgot their password.<br>It will give a description of what action has occured and additonal information if required in the email.<br>if email_type_to_send == created; then user_data type is an object of login_register_changepassword_form.RegisterForm<br>or if email_type_to_send == changed; then user_data is an integer with the user's id<br>else which is when email_type_to_send == forgot; it will it will be a string of the user's email |

### user_interactions_with_database.py

When the user enters information on the website it will be sent to one of the functions in this file, which will request or send data to or from the database, depending on the action which the user took.

#### *Imports used*

from cryptography.fernet import Fernet

from json import dumps, loads, load

from datetime import date

from random import choice

#### Custom imports from project

from formulas import selects_info_from_database, updates_database, inserts_info_to_database

from user_information.encryption import encrypt, decrypt

#### *Functions*

| Function Name | Parameters: Type | Returns: Type | What does it do |
|---|---|---|---|
| user_register | :param: form: login_register_changepassword_form.RegisterForm | :return: False: bool - If the username or the email already excists | Takes in the parameter form which is the user has entered all their information in<br>and it is added to the database, encrypting the password and the security questions |
| user_login | :param: form: login_register_changepassword_form.LoginForm | :return: TYPE: str - If the username or the email already excists | Takes in the parameter form which is the user has entered all their information in<br>and it is checked to the information in the database to see if they are the same, if they are the same<br>then the user can be logged into the system. |
| user_change_password | :param: form: login_register_changepassword_form.ChangePasswordForm<br>:param: user_id: int<br>:param: security_field: str - The name of the field of the database to get the security question | | Takes in the parameter form which is the user has entered all their information in<br>and it is checked to the information in the database to see if they are the same, if they are the same<br>then the user can change their password to the one which they have entered. |
| user_adds_ratings | :param: film_id: int<br>:param: user_id: int<br>:param: ratings: list | :return: False: bool - If the username has already rated the film | Adds the rating to the database if it is new or updates it if the user has already rated that film |
| user_security | :param: user_id: int | :return: | Gets the security question which will be |

| | | | |
|---|---|---|---|
| _question_to_ask | | security_field: str<br>:return: security_question_to_ask: str | asked if the user wants to change their password,<br>it is generated when the user logs in and is stored in the cookies |

email_details.json

The email address and the password of the email which will send the information to the user, this is stored in a dictionary in a JSON file, so that it can be accessed in the python script, if there is any need to change the password for the email it is in a known location without needed to search in the code.

Will have the keys; "email" and "password".

rating_questions.json

Stores all of the rating questions which will be asked to the user.

Will have the keys; "overall", "comedy", "actors" and "quality"

security_questions.json

Stores all of the security question options which will be available for the user, there will be 3 keys and each key will have a list of 5 elements each element is a 2d list with the first index being a number 1 to 5 and the second index being the question which will be asked to the user.

Will have the keys;  "SecurityQuestion1", "SecurityQuestion2" and "SecurityQuestion3"

email_contents folder

Houses the email templates which will be sent to the user if an email needs to be sent.

*changed.txt*

The contents of the email if the user requests to change their password, it will inform the user of the changes to their account.

*created.txt*

The contents of the email if the user has created an account, it will inform the user of all the information which they will need to know about the website and security along with when they can change their password and any of their information.

*forgot.txt*

The contents of the email if the user requests to know their current password, it will email the user their password and tell them any information they need to known.

# searching folder

This folder houses all of the python scripts which are used to find information.

## searching_algorithm.py

The purpose of this file is to house all of the functions which are used to find the film, actor or person which the user has searched for. If the user has searched for a film then it will spell check the query before starting the algorithm.

### *Imports used*

from spellchecker import SpellChecker

from datetime import datetime

from json import load

from collections import Counter

#### Custom imports from project

from formulas import binary_search, selects_info_from_database

### *Functions*

| Function Name | Parameters: Type | Returns: Type | What does it do |
|---|---|---|---|
| spell_checks_query | :param: query: list | :return: TYPE: list - 2d array of the orginal query and potentially the possible correction if there was an error | Spell checks the query which the user has enetered, splitting it into a list at spaces and *dahses* "-", checking the spelling of these words.<br>If any of these words in the query have an error then it will update the query with the corrections<br>If there are no spelling errors or if there are more than 5 spelling errors,<br>aswell as if the amount of words needed to check is the same amount as the words in the query (but not 1) and the original amount of words in the query is less than 7<br>then it will return the original query as a 2d array of 1 element.<br>else it will return a 2d array with 2 elements the first being the original query and the second being the spell checked one |
| removing_common_words | :param: query: list | :return: word_to_gather: list | Takes the users spellchecked query as a list (may be changed may not be changed)<br>Gathers the top 30 most common words from the "common_words.json" file which has been created by scraping the |

| | | | |
|---|---|---|---|
| | | | database<br>Loops through the query list and removes all of the common words; but will not remove more than half of the words<br>Will start by removing the most common words in the database to the least |
| gathering_data | :param: corrected_querried: list<br>:param: year_criteria: list<br>:param: genre_criteria: list<br>:param: language_criteria: list<br>:param: film_actor_director: int | :return: TYPE: list - list of tuples (id, name) where id and name is for the film, actor or director | The user has entered a query this has been spellchecked and correct with the common words removed, if it is a film.<br>The function, goes through all of the elements in the list and creates a where statement to select all of the, films, actors or directors,<br>which the words in the query in.<br>Adds the criteria to the where statement if the user has entered any.<br>Excecutes the sql statement and gathers the data from the database. |
| data_which_have_each_word | :param: data: list<br>:param: original_query: str<br>:param: query: list :param: film_actor_or_director: list | :return: TYPE: Counter - from the collection module, all of the data tallied up with frequencies | Takes the ids and the names of the films, actors or directors, splits them into seperate lists (all_ids, all_names).<br>The query is split into a list at each word interval so that the algorithm can find the films which have each word in<br>If a film, actor or director has more than 1 of the words in the query then it's title, then it will<br>be favoured over the ones which do not to find the best result. |
| finding_top_data | :param: tallied_data: Counter or list. If is_one is False then tallied_data will be a Counter object (ID, OCCURENCE), where OCCURENCE is the amount of words in the user's query But if is_one is true then tallied_data will | :return: to_display: list | Gathers the top 50 films if the user is searching for a film or the top 30 actors or directors if the user is searching for an actor or director<br>It returns a list in order of the most relevant, with the first index being the most relevant to the user's search<br>If the user's query only has 1 word in it then it will perform a binary search to find the films which have only that word in it (as an exact match)<br>and append it to the front of the list, then fill in the remaining amount of space with data ordered by the |

| | | | |
|---|---|---|---|
| | be a list of tuples like (ID, NAME)<br>:param: original_query: str<br>:param: film_actor_or_director: int<br>:param: is_one: bool | | revenue if it is a film (highest revenue first)<br>and if it isn't a film then it will just select a random id to append.<br>If the revenue is unknown then it will just append a random id to fill in the space required.<br>If the query has more than 1 word, then it will look at the films (if it is a film) which have the most amount of words from the query in it and<br>append those to the front of the list in descending order of the revenue, and so on until the 50 films are reached.<br>For actors and directors though it will just append the ids with the highest occurence of different words at the front of the list until 30 actors or directors have been added<br>The data gathered from here will be displayed in sets of 10s |
| gathers_potential_spellchecked_query | :param: query: str | :return: more_possible_query_results: list | When the user is searching for a film, it will spell check the query, and return the potential options<br>Will return none if the user did not enter a query, the query would be "". Will return False if there are no potential<br>spell checked options, and if there are options it will return more_possible_query_results, which is a list of the original query<br>and the potential option |
| searching_algorithm_gathers_film_ids_to_display | :param: query: str<br>:param: year_criteria: list<br>:param: genre_criteria: list<br>:param: language_criteria: list<br>:param: film_actor_or_director: int | :return: film_actor_or_director_ids_to_gather_information_about_to_display: list<br>:return: amount_of_films_or_people_filtered_through: int | The main function of finding the search query which the user has entered for a film, actor or director.<br>It will first remove all of the most common words if it is a film, where the most common words are the ones<br>which appear the most in the database.<br>It splits the user's query into a list at all spaces " " and finds all of the individual parts to the query in from the database<br>so it will select all of the films, actors or directors which have atleast 1 of the parts of the uesr's query. |

| | | | |
|---|---|---|---|
| | | | It will then create a hierarchy of the films, actors or directors which have the most words/parts in compared to the user's query<br>I will then order all of the ids to be in order of the hierarchy and return the results to display on the website.<br>if the param "film_actor_or_director" is equal to 0 then it is a film, equal to 1 then an actor, equal to 2 then director |

### leaderboards.py

The purpose of this python script is to create the calculate the top ratings for the "Overall" and "Comedy" rating and store them in the database, as well as when the user is searching for a leaderboard it will return the information which is related to the leaderboard stated (5 possible options). Finally it houses the function which is used to periodically call the function which updates the top rated films function, and this function is inside a thread with a sleep statement so it doesn't always run.

#### *Imports used*

from time import time, sleep

#### Custom imports from project

from formulas import selects_info_from_database, mean_and_sd, groupings, inserts_info_to_database, deletes_from_database

#### *Functions*

| Function Name | Parameters: Type | Returns: Type | What does it do |
|---|---|---|---|
| gathers_top_films_from_database_on_request | :param: chategory: str<br>:param: langauge_criteria: list<br>:param: genre_criteria: list<br>:param: year_criteria: list | :return: TYPE: list - a list of film ids if there are any | When the user requests to see a leaderboard this function will be called, it will, return the top 100 film ids which relate to it<br>and which have the criteria specified on it applied.<br>It will be in order of priority and relevance<br>There are 5 options the user can enterm Budget and Revenue, which are already stored in the database<br>Profit which is the Budget Subtracted from the revenue<br>Then finally top rated and funniest, which have been calculated using the user's ratings |
| updates_databas | :param: chategory: | | The function is called when the |

| | | | |
|---|---|---|---|
| e_with_top_rated_films | str | | website is loaded up, and is run periodically, updating the top rated and funniest films in the database,<br>so that when the user requests to see the leaderboard for them it is the most up to date it can be without causing performance issues<br><br>The chategory of "Overall" which is the top ratings has the value of 1<br>and the chategory of "Comedy" has the value of 2 |
| creating_and_updating_top_ratings_for_leaderboards | | | Runs when the webiste loads up, then every 30 minuets after that until it shuts down<br>this function will be in a thread so it will be running in the background |

### description_of_films.py

This python script has a single function, which is used to gather the description of the film id given as a parameter and returns the description if there is one to be found.

#### *Imports used*

from bs4 import BeautifulSoup

from requests import get

#### Custom imports from project

from formulas import selects_info_from_database

#### *Functions*

| Function Name | Parameters: Type | Returns: Type | What does it do |
|---|---|---|---|
| gathering_description | :param: film_id: int | :return: desc: str OR False: bool - if there is no description | Gathers the description of the film from imdb, using BeautifulSoup4 |

# recommend folder

Stores the python scripts which are needed to create the recommendation for the user.

## gathering_types.py

The functions from inside of this file are called from the python script " formulating_recommendation_from_types.py" which is used to create the recommendation for the user. This file is used to find all of the attributes to look for when creating the recommendation for the user.

### *Imports used*

from threading import Thread, active_count

from collections import Counter

### Custom imports from project

from formulas import selects_info_from_database, mean_and_sd, combination_formula, binomial_distribution, groupings

### *Functions*

| Function Name | Parameters: Type | Returns: Type | What does it do |
|---|---|---|---|
| language | :param: films: list - all the films which the user has favourited | :return: TYPE: list - the language ids which the user may like | Gathers all the users favourited and rated films, and determines which language they are most likely going to want the films to be in |
| colour | :param: films: list - all the films which the user has favourited | :return: TYPE: float - the ID of the type of colour, or the percentage of films which should be monochrome | Gathers all the users favourited and rated films, and determines which colour type the user is most likely going to want the film to be in |
| runtime | :param: films: list - all the films which the user has favourited | :return: lower bound: int<br>:return: upper bound: int | Gathers all the users favourited and rated films, and determines the length of the film the user is most likely going to prefer |
| release_date | :param: films: list - all the films which the user has favourited | :return: lower bound: int<br>:return: upper bound: int | Gathers all the users favourited and rated films, and determines which range of years the user is most likely going to want a film from |
| statistics_for_actors_directors_genres | :param: films: list - all the films which the user has favourited<br>:param: field: str<br>:param: table: str<br>:param: rating_chategory: str<br>:param: UNKNOWN_ID: int - if not specified then unknown id for actor or | :return: mean_rating: float<br>:return: sd_of_rating: float<br>:return: mean_occurence: float<br>:return: | The directors, actors and genres algorithm are very similar, so can use the same function for a large part of it.<br>Quality rating is for the directors, Actors rating is for the actors and Overall rating is for the genres.<br>The quality is determined by how well the film is produced and that is based on the director.<br>Takes in all the users favourited |

| | | | |
|---|---|---|---|
| | director as they are the same. When specified it will be the unknown id for genre | sd_of_occurence: float<br>:return: one_of_each_rating: list<br>:return: average_ratings: dict<br>:return: director_actor_occurence: dict<br>:return: without_unknowns: list | films, and information for the sql query as paramaters. Then returns data which can be used in either the actors, directors or genre<br>function to determine which actors, directors or genres the user is likely going to want. |
| directors | :param: films: list - all the films which the user has favourited | :return: TYPE: list - 2d list, containing 2 elements; the first list are the ids of the directors which films are<br>wanted for and the second list are the ids of the directors which are not wanted | Gathers all the users rated and favourited and rated films, and determines which directors they are likely going to want,<br>and the ones which they are likely not to want, calling the "statistics_for_actors_directors_genres" function |
| actors | :param: films: list - all the films which the user has favourited | :return: TYPE: list - 2d list, containing 2 elements. 1st - Singular actors to gather. 2nd - Actors which should not be in the films gathered. | Gathers all the users rated and favourited and rated films, and determines which actors they are likely going to want,<br>and the ones which they are likely not to want, calling the "statistics_for_actors_directors_genres" function<br>It calculates the most popular actors, aswell as the actors which should not be in the recommended films. |
| genres | :param: films: list - all the films which the user has favourited | :return: TYPE: list - 2d list, containing 2 elements. 1st - Combinations of genres to gather, 2nd - List of genres to gather, with the weighing order to present in the recommended | Gathers all the users rated and favourited films and determines which genres they are likely going to want,<br>and the ones which they are likely not to want, calling the "statistics_for_actors_directors_genres" function<br>It calculates the most common combinations of genres in films which have the highest rating. Aswell as the most popular genres,<br>if there are not more than 3 |

| | | | |
|---|---|---|---|
| | | films. | combinations, aswell as returning the genres which should not be in the recommended films<br>Combinations and singular genres have weightings, where if a film has that combination of genres or singular genre it will be added to a different weighting class,<br>Where 3 is the highest weighting which will be favoured, followed by 2 then 1. If the weighting is 0 then that genre or combination is not wanted |

## formulating_recommendation_from_types.py

This python script has no stand alone functions, it only has one class in it, which is used to calculate the recommendation for the user, the purpose of this file is to house the class which is used to create the recommendation for the user specified. The class will take a parameter of the user id and looks at all of the films that user has currently rated and favourited.

### *Imports used*

from random import choices

#### Custom imports from project

from formulas import inserts_info_to_database, selects_info_from_database, deletes_from_database

from classes import FilmAttributesForRecommendation

from recommend.gathering_types import groupings, language, colour, runtime, release_date, directors, actors, genres

### *RecommendedFilms Class*

Gathers all of the users favourited and rated films from the database

Checks to see if the user has enough films to make a good recommendation (atleast 5)

If they do then it will gather all films which the wanted actors and directors have been in if there are atleast 20 films from these 2, then it will

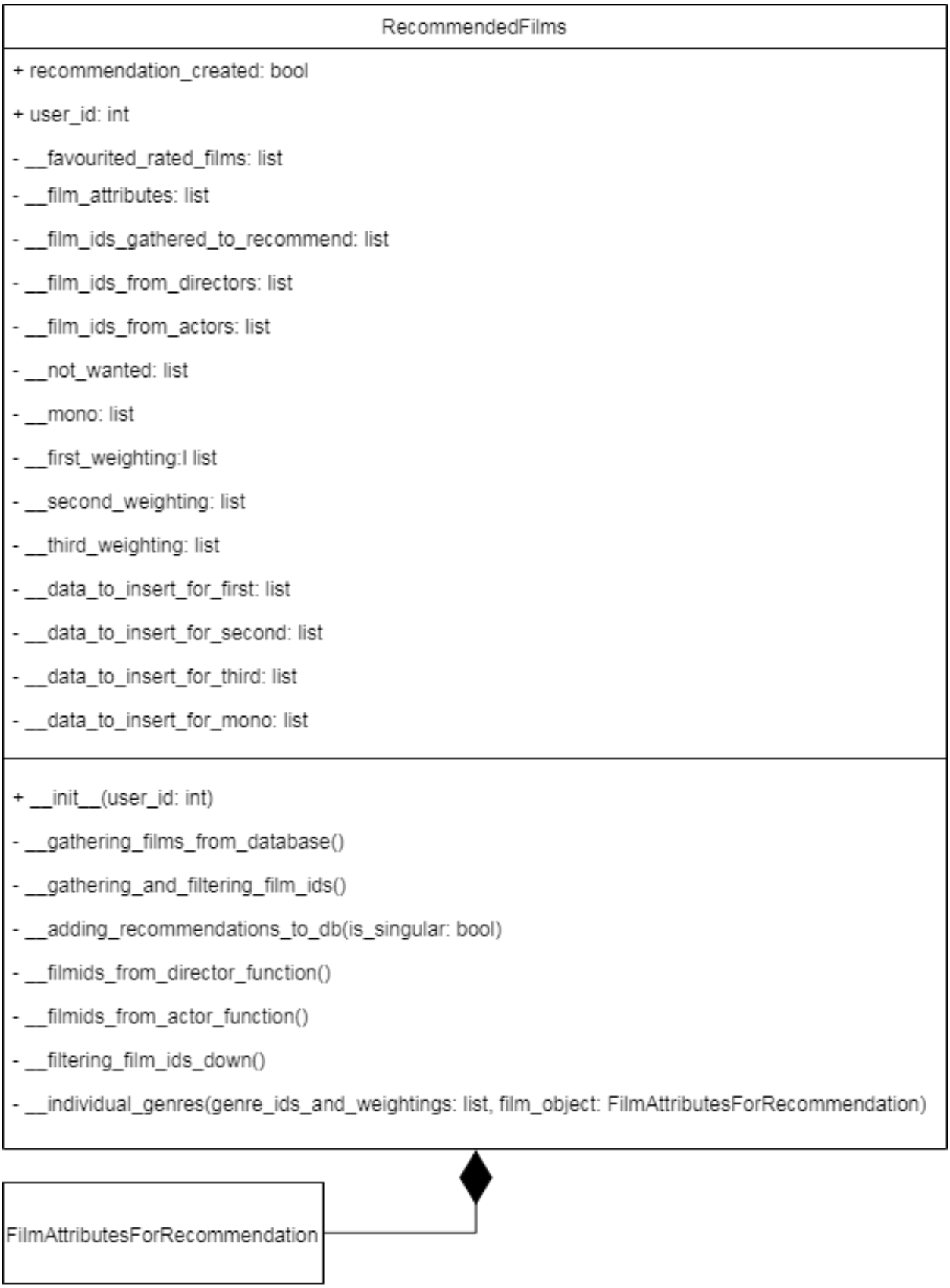
Filter down (a random set of 100 films from the actors and directors)

all of the films from the actors and directors in the following chategories runtime, release date, language, colour of the film and the genres.

If there are more than 10 filtered recommendations then they are added to the database.

## Class diagram

The class of "FilmAttributesForRecommendation" is called from inside, which is needed to assign variables to the attributes in the class, so that when the information about the films are used later on they are in an organised form.



### Methods for the class

These are all of the methods which are defined in the class.

| Function Name | Parameters: Type<br>Return: Type | What does it do |
|---|---|---|
| __init__ | :param: user_id: int | Calls all the functions, and creates the attributes |
| __gathering_films_from_database | | Gathers all of the film ids from the database what are going to be used to create a good set recommendations |
| __gathering_and_filtering_film_ids | :return: True or False: bool - depends on if a recommendation is created or not | Gathers all of the films which the actors wanted and directors wanted have acted in and directed.<br>Checks to make sure that there are enough film ids to generate a decent recommendation, if there is not will return False, so that the user knows to favourite or rate more films<br>If there are enough films, then it will use all of the film ids from the actors and the directors to create a recommendation, by first;<br>gathering all attributes for each film and creating this as a list of objects. Comparing each value of each film and filtering it down.<br>Putting higher influenced films in a different chategory so that they are favoured when adding the 30 films to the database<br>Once all of this has been done the information is gathered together and is added to the database. |
| __adding_recommendations_to_db | :param: is_singular: bool | Adds the top 30 highest weighted films, is done randomly in the sense that the lists are not in any particular order, and is just selecting the first 30 from the lists<br>If none of the films have a speicifc weighting then will just add the first 30 of the "self.__film_ids_gathered_to_recommend" list to the database<br>This is the order of priority --- self.__first_weighting, self.__mono, self.__second_weighting, self.__third_weighting<br>for self.__mono, it will be added to the database, but when the user is viewing the films it will try and only select a third in each segment of 10 (using integer division). |

| | | |
|---|---|---|
| __film_ids_from_director_function | | Gathers all of the film ids based on the director ids which were wanted and not wanted from the "directors" function<br>The function is called and the film ids are assigned to the __film_ids_from_directors list,<br>this is done so that threading can be used in the main class to speed up the data processing speed, so tasks can be preformed simultaneously |
| __film_ids_from_actor_function | | Gathers all of the film ids based on the actor ids which were wanted and not wanted from the "actors" function,<br>It groups together all of the films where it has multiple actors from the ones which are wanted<br>(the only actor ids which are selected are the ones which were from the "actors" function)<br>It adds all of the film ids where it has 2 or more actors (from the selected actors from the function) to the "__film_ids_from_actors" list.<br>If there is not 100 films already in the list, then it will randomly select films where it only has 1 actor using the random.choice method<br>Until the length of the list is equal to 100 |
| __filtering_film_ids_down | | 5 SECTIONS<br><br>RUNTIME<br>Find films which are longer or shorted than the wanted runtimes, and ignores films which have an unknown runtime<br><br>RELEASE DATE<br>Finds the films which are not in the bounds for the dates which the user will like, and adds it to the not_wanted list<br><br>COLOUR<br>Checks to see if any of the films do not have the colour which is wanted, and if they do have the colour which is not wanted then it is added to the not_wanted list.<br>If a probability is given then gathers all of the monochrome films to represent later at the final stage |

| | | |
|---|---|---|
| | | LANGUAGE<br>Finds films which do not have any of the languages which are wanted and adds them to the not_wanted list if they do not have any<br><br>GENRE<br>Films which have the highest desired combination are added to the "__first_weighting" list<br>Films which have a desired combination will be added to the "__second_weighting" list aslong with singular genres which have the highest desirability<br>Films which have desirable singular genres will be added to the "__third_weighting" list.<br>Films which have combinations which are not wanted or genres which are not wanted are added to the not_wanted list<br>If the genre is unknown, or does not contain one of the genres specified, then it will be just left and not added to any lists<br><br>SUMMARY<br>All of these 5 sections are done in 1 functions is since they are all done using a loop, important information is assigned and stated before the loop begins<br>Each section is labled with its name<br>Finds all of the unwanted film ids |
| __individual_genres | :param: genre_ids_and_weightings: list<br><br>:param: film_object: FilmAttributesForRecommendation - which is an object | Looks at all of the genres and its weightings, compares it to the data of that specific film (film_object)<br>updates the variables which are stored inside of the class and is only called 2 times |

# dictionary_creator_for_html_pages folder

Used to house the python scripts which are used to find the information to display on the website for the film, actor or director ids specified as a parameter to the functions, there are 2 files and they both follow the same principal of taking a list as a parameter of the ids specified and returning a list of the information (or multiple lists and dictionaries if there is extra information required). This information is then sent to the html template using "Jinja2" which will loop through the list and display it to the user.

## formats_and_creates_dictionary_for_films.py

Used to store the functions which are used to find all of the necessary information about the list of film ids given, so that it can be displayed to the user in a nice form and a legible fashion.

### *Imports used*

from threading import Thread, active_count

from time import time

#### Custom imports from project

from formulas import selects_info_from_database

from classes import AllFilmAttributes

### Functions

| Function Name | Parameters: Type | Returns: Type | What does it do |
|---|---|---|---|
| information_about_films_as_objects | :param: film_ids: list | :return: all_objects_of_film_information: list | Creates objects for each film |
| adds_to_information_to_display_dictionary_for_films | :param: information_about_films: dict<br>:param: film_object: classes.AllFilmAttributes<br>:param: all_actor_ids_related_to_films: list<br>:param: all_director_ids_related_to_films: list<br>:param: all_genre_ids_related_to_films: list<br>:param: all_language_ids_related_to_films: list | | Takes in the dictionary as a param which will be updated, and does not need to be returned since it is a global inside of the function,<br>the function may be called in the same cpu cycle since it is being threaded and the dictionary can be updated in each thread<br>seperately. film_object is an object created from the classes.AllFilmAttributes class. |
| turning_film_information_into_dictionary | :param: film_ids: list<br>:param: ordered: bool | :return: gathered_and_formatted_information_about_films: list | Gathers all of the information about all of the films from the database.<br>Puts it into a nice format which the user can read easily, and gathers all of the extra information |

| | | | |
|---|---|---|---|
| | | :return: actors: dict<br>:return: directors: dict<br>:return: genres: dict<br>:return: languages: dict | about the actors and directors which will be used to display in a modal in the website.<br>it then returns a list of dictionaries with all the information about the films,<br>which can be looped through on the website using jinja2 and all of the information is displayed |

### formats_and_creates_dictionary_for_people.py

Used to store the functions which are used to find all of the necessary information about the list of actor or director ids given, so that it can be displayed to the user in a nice form and a legible fashion. It will also format the data if required as some pieces of information stored in the database are not easy to be read at a glance.

#### *Imports used*

from threading import Thread, active_count

from json import loads

from time import time

#### Custom imports from project

from formulas import selects_info_from_database

from classes import AllActorAndDirectorAttributes

#### *Functions*

| Function Name | Parameters: Type | Returns: Type | What does it do |
|---|---|---|---|
| information_about_people_as_objects | :param: people_ids: list<br>:param: actor_or_director: str | :return: all_objects_of_people_information: list | Creates an objects for each person<br>The param actor_or_director will have the value of either "Actor" or "Director", it is used<br>when deciding which table to select from, from the database |
| adds_to_information_to_display_dictionary_for_people | :param: information_about_people: dict<br>:param: person_object: classes.AllActorAndDirectorAttributes | | Takes in the dictionary as a param which will be updated, and does not need to be returned since it is a global inside of the function,<br>the function may be called in the same cpu cycle since it is being threaded and the dictionary can be updated in each thread<br>seperately. person_object is an object created from the classes.AllActorAndDirectorAttributes class. |
| turning_peopl | :param: | :return: | Gathers all of the information about all of |

| | | | |
|---|---|---|---|
| e_information_into_dictionary | people_ids: list<br>:param:<br>actor_or_director: str | gathered_and_formatted_information_about_people: list | the actors or the directors from the database.<br>Puts it into a nice format which the user can read easily and returns all of the data as a list of dictionaries<br>which can be looped through on the website using jinja2 and all of the information is displayed |

## templates folder

Used to house all of the html templates which are accessed by the user, the python script will render them and send in data using "Jinja2", this can be displayed in a nice format for the user, so they can get the best possible experience.

### home.html

The page which the user will first see when they go onto the website and when they first login. If any errors occur when the user is using the website then they will be redirected to this page, as well as if system needs to tell the user important information, such as; their account has been created, an email has been sent or if they do not have any films currently favourited

### layout.html

This page is never accessed by the user, but everything inside of it is inherited onto the other html pages, it has the layout of how the website will look as well as the information which is needed on the navigation bar. All of the "JavaScript" code is stored here as when I tried to store it in a separate ".js" file it the website would not pick up its contents, so all of the code is stored at the bottom of the file, with each "JavaScript" function being in its own script tag "<script>". The majority of the functions which have been defined use "ajax" from the Google API as well as code from "jquery".

### search/films folder

This is a folder inside of another folder, and houses all of the html files which are related to information revolving around films, this includes; if the user wants to search for a film or a leaderboard and if the user wants to see information about a set of films, then it will access the html template which has the layout for the films and passes in all of the information to display to the user using "Jinja2" for the films specified.

#### *search_for_films.html*

When the user is entering a query to search for a film they will access this html template, which has the option to filter the language, genre and year.

#### *leaderboard.html*

When the user is decided what leaderboard to look at, they will access this page, which has the option for them to view 5 leaderboards, as well as the same filtering option as the "search_for_films.html" template, so the user can filter the language, genre and year to what they would like if they choose to.

*display_film_information.html*

The layout of how all of the information about a set of films will be presented to the user, the information will be passed in as a variable and accessed using "Jinja2" the information will also be looped over using a "for" loop in "Jinja2" to display the information for the user to view in a nice format. If any extra formatting is required then it will be done inside of this file. All pages which require information to be displayed about a film are displayed using this html template.

## search/people folder

This is a folder inside of another folder, and houses all of the html files which are related to information revolving around actors or directors, this includes; if the user wants to search for am actor or a director and if the user wants to see information about a set of actors or directors, then it will access the html template which has the layout for the people (actors or directors) and passes in all of the information to display to the user using "Jinja2" for the actor or director specified.

*search_for_person.html*

When the user is entering a query to search for an actor or director they will access this html template, which has the option to filter the date of birth for the year range only, so that it will only display actors or directors born before, between or after a specific year.

*display_people_information.html*

The layout of how all of the information about a set of actors or directors will be presented to the user, the information will be passed in as a variable and accessed using "Jinja2" the information will also be looped over using a "for" loop in "Jinja2" to display the information for the user to view in a nice format. If any extra formatting is required then it will be done inside of this file. All pages which require information to be displayed about a film are displayed using this html template.

## user_information folder

Any html template which is accessed by the user and they pass in information specifically about them, is stored in this folder.

*register.html*

When the user signs up for website they will have to enter information into a form on this page, the form is rendered to the user and they can enter in the information which they wish, the layout of the form is displayed using the base of the "flask_wtf.FlaskForm" module, a class will have been created earlier on with all of the specific information which will need to be displayed as well as the validation which is required.

*login.html*

When the user wants to log into the website, then they will be redirected to this page, which has the option for them to enter their "Username" and "Password", the user can enter in this information if they wish. The layout of the form is displayed using the base of the "flask_wtf.FlaskForm" module, a class will have been created earlier on with all of the specific information which will need to be displayed as well as the validation which is required.

### *account_settings.html*

If when the user is using the website and they decide to change their password they will be redirected to this page, where they have to enter one of their security questions (randomly chosen by the system), their current password and their new password which they have chosen into the form on the page. The layout of the form is displayed using the base of  the "flask_wtf.FlaskForm" module, a class will have been created earlier on with all of the specific information which will need to be displayed as well as the validation which is required.

### *forgot_password.html*

If the user does not know their password when they try and login then they can request their current password using this page, where they will need to enter in their email address and their current password will be sent to them. The text box to enter the user's email is inside of a form tag "<form>" and when the user hits enter they will be redirected to the home page telling them that an email has been sent if the email address entered is in the database.

## static folder

This folder houses one file, and this is used for some of the styling on the website.

### main.css

Any extra styling which I decide on will be stored in this file, as the majority of the design and layout of the website will be used using "Bootstrap". Mainly the colouring and positioning of text will be stored in this file as an attribute to a "class" or an "id".

# Technical Solution

## Python code to create database

When creating my database, I wrote 8 python scripts to gather the information, and I had 3 databases in total used which I spoke more about before in my design section. These are the 8 python scripts which I coded

### These are the python scripts used in order

- link_gatherer.py
- film_information_off_imdb.py
- neatens_database.py
- people_information_off_imdb.py
- moves_all_data_to_one_database.py
- images_for_films.py
- images_for_people.py
- changes_currency

### Python code

- app.py
- formulas.py
- classes.py
- encryption.py
- sending_emails.py
- login_register_changepassword_form.py
- user_interactions_with_database.py
- description_of_films.py
- searching_algorithm.py
- leaderboards.py
- gathering_types.py
- formulating_recommendation_from_types.py
- formats_and_creates_dictionary_for_films.py
- formats_and_creates_dictionary_for_people.py

## Folder and file structure

All of these files are inside a main folder, which is used to run my website.

- app.py
- formulas.py
- classes.py
- user_information
    - encryption.py
    - sending_emails.py
    - login_register_changepassword_form.py
    - user_interactions_with_database.py
    - email_details.json
    - security_questions.json
    - rating_questions.json
    - email_contents
        - created.txt
        - changed.txt
        - forgot.txt
- searching
    - description_of_films.py
    - searching_algorithm.py
    - leaderboards.py
    - all_genres.json
    - all_languages.json
    - common_words.json
- recommended
    - gathering_types.py
    - formulating_recommendation_from_types.py
- dictionary_creator_for_html_pages
    - formats_and_creates_dictionary_for_films.py
    - formats_and_creates_dictionary_for_people.py
- templates
    - layout.html
    - home.html
    - user_information
        - register.html
        - login.html
        - account_settings.html
        - forgot_password.html
    - search/films
        - search_for_films.html
        - leaderboards.html
        - display_film_information.html
    - search/people
        - search_for_people.html
        - display_people_information.html

# HTML templates

## layout.html

All of the code in this file is inherited by the other files, so that all pages look the same.

The code which is in the head tag “<head>” of this file, is used to gather the CSS code, which is used in this website, including the bootstrap. It also sets the title of the page on the website. The next part of the code in this file is used to  display the navigation bar, it will change how the navigation bar looks depending on if the user is logged in or not. The next part are the scripts which is the JavaScript used. As I was unable to get the project to recognize the separate JavaScript file I stored it at the bottom of the layout file. I stored each function in its own script tag “<script>”

The first script tag, is used to listen for when the user hits the enter key, this is used when the user is searching for a film, as when they enter information into the search box they have to click this button, this enables them to press the enter key instead and it emulates the search button being pressed.
The rest of the code which are in script tags are in full functions and are not listeners

The next piece of code which is in the script tag is the “spellChecksUserQuery” function which is used to gather the query which the user has entered into the search box when they are searching for a film, and sends the information to a python script to check if the information which the user has entered is spelt correctly if it is then it will script the next part. If it is incorrect then it will display two possible options to the user, one which is the query they originally answered and the other is the query which they might of meant instead.

The next piece of code which is in a script tag is the “userQueryInformationGatherer” function, this is called after the “spellChecksUserQuery” function has finished running and the user has clicked on the query which they want to find films about. If there is no corrected spelling, then this function is run without the user needing to press anything. It gathers all of the filters from the HTML template where the user is searching for a film and sends it to a python function for the information to be calculated. Once all of the information has been sent to the python function, the python function will find all of the information to the films relating to the query which the user has entered, and it will return the page which needs to be displayed to the user. At the end of this function it will render the page for the user of all of the information about the films for the user to see.

The next piece of code which is in a script tag is the “moreFilms” function, this is used to display the next set of 10 films to the user if they have clicked the option of “Results x-y”.

The next piece of code which is in a script tag is the “descriptionOfFim” function, this is used to gather and display the description of the film to the user, when they click on the button to do so.

The next piece of code which is in a script tag is the “favouritingButton” function, this is used to add the film to the database as a favourite for the user when it is clicked.

The next piece of code which is in a script tag is the “unfavouriteButton” function, this is used to remove a favourited film from the database when the button is clicked.

The next piece of code which is in a script tag is the “addsRatings” function, this is used to add the ratings to the database for a film if the user tries to rate a film.

The next piece of code which is in a script tag is the “waitingImage” function, this is used to display an image when the user clicks on a button which may take a long time to gather the information. This is for aesthetics  so that the user does not feel as the wait is as long as it actually is

The next piece of code which is in a script tag is the “recommendationWaitingImage” function, this is the same as the function about but is only used in the recommendations page, as when the user clicks on the “New Recommendation” button it will display an additional message as well, as this takes longer to calculate so I am informing the user.

The next piece of code which is in a script tag is the “userLikedRecommendation” function, this used to add a rating to the database at a value of 6 for all categories, this is run when the user is on the recommendations page and clicks on the button to of “Good recommendation”, this will help improve the recommendation algorithm.

The next piece of code which is in a script tag is the “userNotLikedRecommendation” function, this is similar to the function above, but the ratings which are adding to the database are 4 in each category instead of 6. This helps improve the recommendation algorithm by disfavouring films which are like this one.

## home.html

The main page, if any errors occur or if the user needs to know information then they will be redirected to this page.

## register.html

When the user is registering to the website, they will be on this page, they will be able to enter their information into the form on this page. The validation and the layout of this form is controlled by the parameter of “form”, this is using the “flask_wtf.FlaskForm” module

## login.html

This is similar to the register form but is for the login, this also uses “flask_wtf.FlaskForm” as its base.

## account_settings.html

This page is again similar to the register page but is used when the user wants to change their password, it uses the base of “flask_wtf.FlaskForm”

## forgot_password.html

This template is used when the user wishes to request their password to be emailed to them, it is a regular HTML form and has one piece of information which the user can enter

## search_for_films.html

This template is used when the user is searching for a film which they want, it has one text field where the user can enter the name of the field, and 3 modals which are used to filter the; languages, genres and release dates of the film. This information is gathered using a JavaScript function, and it will spell check the user’s query if there is a spelling error.

### leaderboard.html

This template is used when the user is trying to see a leaderboard, it has 5 buttons which the user can choose from to see a leaderboard about, and they are also about to filter the; languages, genres and release dates similarly to the search for film page.

### display_film_information.html

This template is used to display all of the information about the films, it takes in multiple parameters and that information is accessed here, it loops through all of the film information and accesses any extra information about actors and directors through a separate dictionary parameter.

This template is used in multiple pages on the website including; results for films when the user searches, favourites, recommendations, next page when the user clicks on more results, all of the leaderboards and all of the films an actor has acted in or a director has directed.

### search_form_people.html

This template is used when the user is searching for an actor or a director, they are able to enter the query for the actor or director they are trying to find, they are also able to enter a number for the date of birth lower and upper, so that they can decide the year range of the date of birth of the directors or actors they want to see.

### display_people_information.html

Used to display all of the information about an actor or director, this page is used for both. It is used when the user decides to view more information about an actor or director from the actor or director modal on the page which displays all of the information about a film, or when the user is searching for an actor or director.

## CSS

### main.css

These are all my custom styles which are used in my project

## JSON files

### all_genres.json

All of the genres which are in the database, where the key is the GenreID in the database. This is used when displaying the filtering option.

### all_languages.json

All of the languages which are in the database, where the key is the LangugeID in the database. This is used when displaying the filtering option.

### common_words.json

All of the most common words in the title of films in the database, where the key is the word.

## email_details.json

The email and password which emails are sent from

## security_questions.json

The security questions which are asked when the user is registering their website or changing their password.

```
{"SecurityQuestion1":
  [["1", "What was your favourite teacher's name?"],
    ["2", "What is your favourite food?"],
    ["3", "What is your favourite website?"],
    ["4", "What is your favourite colour?"],
    ["5", "What is the name of your favourite pet?"]
  ],

  "SecurityQuestion2":
  [["1", "Where did you go when you first flew on a plane?"],
    ["2", "What was the last 3 digits of your first phone number?"],
    ["3", "What was your first phone?"],
    ["4", "What is the name of the first thing you learned to cook?"],
    ["5", "What was the name of your first pet?"]
  ],

  "SecurityQuestion3":
  [["1", "Name of your school when you were 10 years old?"],
    ["2", "What hour were you born at?"],
    ["3", "Name of the car your parents drove you around in as a child?
"],
    ["4", "What street did you grow up on?"],
    ["5", "What was the name of the hospital where you were born?"]
]
}
```

## rating_questions.json

The rating questions which the user are asked when they open the ratings modal, when looking at a film.

```
{
    "overall" : "Overall rating?",
    "comedy" : "Was it funny?",
    "actors" : "Did you like the actors?",
    "quality" : "Was it produced well?"
}
```

## txt files

These are all of the templates, which are used when sending the user an email.

- created.txt
- changed.txt
- forgot.txt

# Testing

This is my testing which I performed on my code, I will test the majority of the possible flawed areas in my website, and where the average user may stumble across a problem.

# Test plan

| Test No. | Purpose of test | Timestamp | Data tested | Expected outcome | Actual outcome |
|---|---|---|---|---|---|
| 1.0 | To make sure the website can load up and the user can access it | 0:00:00 | Button click | The homepage is displayed | Same as expected |
| 2.0 | Do the page buttons work on the website before the user logs in | 00:10 | Button click | All the page buttons work and sends the user to the corresponding page | Same as expected |
| 2.1 | Test homepage button | 00:18 | Button click | Displays the homepage | Same as expected |
| 2.1.1 | Test homepage search for film image button | 00:23 | Button click | Redirected to login page for the user to login | Same as expected |
| 2.2 | Test Register button | 00:33 | Button click | Displays the register page with a form | Same as expected |
| 2.2.1 | The user can click "Already Have an Account?" | 00:38 | Button click | Redirected to login page to login | Same as expected |
| 2.3 | Test Login button | 00:42 | Button click | Displays the login form | Same as expected |
| 2.3.1 | The user can click on the "Need An Account?" button | 00:50 | Button click | Redirect to the register page for the user to login | Same as expected |
| 2.3.2 | The user can click on the browser back | 00:56 | Button click | Redirects the user to their previous page which will be the login page | Same as expected |
| 2.3.3 | User can click on the "Forgot password?" button | 00.58 | Button click | Will redirect the user to a page where they can enter their email address, so that their password can be emailed to them | Same as expected |
| 2.4 | Test news modal | 1:04 | Button click | A window will appear with the news about the website | Same as expected |
| 2.4.1 | Test that the "Close" button works | 1:13 | Button click | The menu will close | Same as expected |
| 3.0 | Can the user register to the website | 1:20 | Button click | The user successfully registers to the website and is redirected to the | Same as expected |

| | | | | | |
|---|---|---|---|---|---|
| | | | | homepage telling them they have registered | |
| 3.1 | Go back to the register page | 1:25 | Button click | The register page loads | Same as expected |
| 3.2 | User clicks on the "Register" button | 1:30 | Button click | They are told to enter information | Same as expected |
| 3.3 | Make sure that the user is informed to enter information into all of fields if they click enter "Register" before they all contain data. | 1:32 | Button click | They are told to enter information into the unfilled fields | Same as expected |
| 3.3.1 | To test the principal of data needing to be entered into a field. See if the user does not enter information into the "Name" field they are told to enter information. | 1:40 | **Information entered into each field**<br>Name:- ""<br><br>Username:- "Kian123"<br><br>Email:- "xxx@gmail.com"<br><br>Date of Birth:- "xxx"<br><br>Password:- "Password333"<br><br>Confirm Password:- " Password333"<br><br>Security Question 1 dropdown menu:- " What was your favourite teacher's name?"<br><br>Security Question 2 dropdown menu:- " What was your first phone?"<br><br>Security Question 3 dropdown menu:- " What street did you grow up on?<br><br>Security Question 1 answer box:- "bob" | The "Name" field is highlighted so that the user knows to enter information. | Same as expected |

| | | | | | |
|---|---|---|---|---|---|
| | | | Security Question 2 answer box:- "nokia"<br><br>Security Question 3 answer box:- "lane" | | |
| 3.4 | To test to see if there is validation for the length of the information which the user enters into the "Username" field | 03:22 | Same information which was entered in the test "3.3.1", except for the field "Username" is changed to "a"<br>Username:- "a" | The user is told that the length of the name which they should enter is between 3 and 12 characters | Same as expected |
| 3.5 | To test to see if there is validation for the format of the email address entered in the "Email" field. The format should be, STRING@STRING.STRING | 04:12 | Same information which was entered in the test "3.3.1", except for the field "Email" is changed to "a"<br>Email:- "a" | The user is told that the email must be in the correct form if it is not in a email address format | Same as expected |
| 3.6 | To test to see if the user can only enter numbers into the "Date of Birth" field | 05:04 | Going to spam random characters into the date of birth field | If the user types in a character it should not be entered in, can only type in numbers | Same as expected |
| 3.7 | A calendar will appear when the user clicks the "Date of Birth" field and clicks the calendar icon | 05:40 | Button click | A calendar will appear | Same as expected |
| 3.8 | To test to see if when the user enters a password into the "Password" field and "Confirm Password" field it blacks out the password. | 06:09 | The password which was entered in the test "3.3.1" will be blacked out | The password will appear as black circles | Same as expected |
| 3.9 | To test to see if there is validation for the length of the information which the user enters into the "Password" field | 06:22 | Same information which was entered in the test "3.3.1", except for the field "Password" is changed to "a"<br>And field "Confirm Password" changed to "a" as well<br><br>Password :- "a" | The user is told that the length of the password which they should enter is between 6 and 128 characters | Same as expected |

| | | | | | |
|---|---|---|---|---|---|
| | | | Confirm Password:- "a" | | |
| 3.10 | To test to see if there is validation for when the user enters their password again into the "Confirm Password" field, the password entered must be the same as the password in the "Password" field. | 06:45 | Same information which was entered in the test "3.3.1", except for the field "Confirm Password" is changed to "a"<br><br>Confirm Password:- "a" | The user will told to " Please make sure you re-enter the exact same password as above" | Same as expected |
| 3.11 | The user is able to click on the dropdown menu of the "Security Question" field... It is the same for all security questions just different questions | 07:00 | Button click | A menu of 5 options will appear | Same as expected |
| 3.12 | The user is able to enter anything into the "Security Questions" field answer box as well as the "Name" field | 07:22 | Will spam a bunch of characters into the 4 fields, but will change the "Username" field so that it gives an error so that an account is not created. | No error will appear, for the 3 "Security Question" fields nor an error for the "Name" field. But an error will occur for the "Username" field as we do not want an account created | Same as expected |
| 3.13 | The user can then enter all valid information and register onto the system | 08:56 | Same information which was entered in the test "3.3.1", except for the field "Name" is changed to "Kian"<br>Name:- "Kian" | The user will be redirected to the login page telling them that their account has been created, and an email has been sent. | Same as expected |
| 3.14 | Checks email inbox to see if an email has been sent | 10:04 | Checks email inbox | The inbox has an email saying that the account has been created, for the username "Kian123" | Same as expected |
| 4.0 | Tries to create an account for another user, the purpose is to see if it recognises if there is a duplicate username in the database. | 10:20 | **Information entered into each field**<br>Name:- "xxx"<br><br>Username:- "Kian123" | The user will be told that that username is already in the database, as we have just created an account for the username "Kian123" | Same as expected |

| | | | | | |
|---|---|---|---|---|---|
| | | | Email:- "xxx@gmail.com"<br><br>Date of Birth:- "xxx”<br><br>Password:- "Password333"<br><br>Confirm Password:- " Password333"<br><br>Security Question 1 dropdown menu:- " What was your favourite teacher's name?"<br><br>Security Question 2 dropdown menu:- " Where did you go when you first flew on a plane?"<br><br>Security Question 3 dropdown menu:- " Name of your school when you were 10 years old?"<br><br>Security Question 1 answer box:- "bob"<br><br>Security Question 2 answer box:- "country"<br><br>Security Question 3 answer box:- "school" | | |
| 4.1 | To show that when the user tries to create an account for an email address which is already in the database it will inform them that it already exists | 11:59 | All the same information which was entered in the "4.0" test except that the "Username" field has been changed to "xxx" and the Email "Field" has been changed to "xxx@gmail.com" | The user will be told that the email address has already been used. | Same as expected |

| | | | | | |
|---|---|---|---|---|---|
| | | | Username:- "xxx"<br><br>Email:- "xxx@gmail.com" | | |
| 4.3 | Enters the correct information into the register form, so that an account will be created for the user. | 13:01 | All the same information which was entered in the "4.0" test except that the "Username" field has been changed to "xxx"<br><br>Username:- "xxx" | They will be redirected back to the login page telling them that their account has been created. | Same as expected |
| 5.0 | Go to the login page, to test the forgot password feature | 14:01 | Button click | The login page will load | Same as expected |
| 5.1 | Click on the "Forgot Password?" button | 14:07 | Button click | Will redirect the user to the page where they can enter their email address | Same as expected |
| 5.1.1 | To show that when the user enters an email address which is not in the database then it will inform the user that the email does not exist | 14:12 | Email:- "xxx@gmail.com" | The user will be informed that the email address is not in the database | Same as expected |
| 5.2 | To see if the user is emailed when they enter their actual email address | 14:36 | Email:- "xxx@gmail.com" | The user will have an email in their inbox waiting with their current email, they will also be redirected back to the home page telling them that they have been emailed | Same as expected |
| 6.0 | Go back to the login page, so that the user can log into the website | 15:18 | Button click | The login page will load | Same as expected |
| 6.1 | To show their information must be entered before the user can login, the "Login" button will be test without any information entered into the form | 15:22 | Button click | The fields will be highlighted telling the user to enter information into them | Same as expected |
| 6.2 | To show that the user | 15:35 | Username:- "xxx" | The user will be | Same as |

| | | | | | |
|---|---|---|---|---|---|
| | must enter information which is current in the database. | | Password:- "xxx" | informed that the username is wrong, as it is the first thing which the user has entered | expected |
| 6.2.1 | To show that the user must enter both pieces of information correct before they can login. | 16:05 | Username:- "xxx"<br>Password:- "xxx" | The user will be informed that the password is wrong | Same as expected |
| 6.2.2 | To show that the user is informed that their username is wrong, even though the password is in the database, but is for a different username | 16:30 | Username:- "xxx"<br>Password:- "xxx" | The user will be informed that the username is wrong | Same as expected |
| 6.2.3 | To show that the user can login successfully | 16:52 | Username:- "xxx"<br>Password:- "xxx" | The user will be redirected to the "Search for film" page as earlier they tried to go to that page but did not succeed as they were not logged in | Same as expected |
| 7.0 | Do the page buttons work on the website now that the user is logged in | 17:07 | Button click | All the page buttons work and sends the user to the corresponding page | Same as expected |
| 7.1 | To test the home page button | 17:27 | Button click | Displays the home page | Same as expected |
| 7.1.1 | To test home page "Search for film" image button | 17:30 | Button click | Displays "Search for film" page | Same as expected |
| 7.2 | Test that the drop down feature works when the user hovers over the "Search" button | 18:05 | Hover over button | A small menu will appear with options to select; Films, Leaderboards, Actors, and Directors | Same as expected |
| 7.2.1 | Press the "Films" option from the drop down menu | 18:08 | Button click | Page will reload because the user was already on the "Search for films" page | Same as expected |
| 7.2.1.1 | Test modals work | 17:36 | Button click | Menus will appear with options | Same as expected |
| 7.2.1.2 | Test that the "Languages" button/modal work | 17:42 | Button click | A menu will appear with all the language options | Same as expected |
| 7.2.1.3 | Test that the cross "X" | 17:45 | Button click | The menu will close | Same as |

| | | | | | |
|---|---|---|---|---|---|
| | button works | | | | expected |
| 7.2.1.4 | Test that the "Genres" button/modal work | 17:53 | Button click | A menu will appear with all the genre options | Same as expected |
| 7.2.1.5 | Test that the "Years" button/modal work | 18:00 | Button click | A menu will appear with 2 options, lower bound and upper bound | Same as expected |
| 7.2.2 | Press the "Leaderboards" option from the drop down menu | 18:16 | Button click | The user will be redirected to the "Leaderboards" page | Same as expected |
| 7.2.2.1 | Show that the filters are the same as the filters on the "Search for films" page | 18:19 | Button click | The filters will look at the same as the ones on the "Search for films" page | Same as expected |
| 7.2.2.2 | Click on the "Highest Revenue" button | 18:31 | Button click | The user will be redirected to a page with all of the films with the highest revenue in the database | Same as expected |
| 7.2.2.3 | Click on the "Biggest Budget " button | 18:49 | Button click | The user will be redirected to a page with all of the films with the largest budget in the database | Same as expected |
| 7.2.2.4 | Click on the "Most Lucrative" button | 19:03 | Button click | The user will be redirected to a page with all of the films which have been the most profitable in the database | Same as expected |
| 7.2.2.5 | Click on the "Top Rated " button | 19:16 | Button click | The user will be redirected to the top rated page, however an image will appear stating that "there is no information available for the query that you have entered". This is because the database is currently empty as no users have rated any films | Same as expected |
| 7.2.2.6 | Click on the "Funniest" button | 19:40 | Button click | The user will be redirected to the funniest page, | Same as expected |

| | | | | | |
|---|---|---|---|---|---|
| | | | | however an image will appear stating that "there is no information available for the query that you have entered". This is because the database is currently empty as no users have rated any films | |
| 7.2.3 | Press the "Actors" option from the drop down menu | 20:00 | Button click | The user will be redirected to the "Actors" page | Same as expected |
| 7.2.4 | Press the "Directors" option from the drop down menu | 20:10 | Button click | The user will be redirected to the "Directors" page | Same as expected |
| 7.3 | To test the "Favourites" page button | 20:17 | Button click | Redirects the user to the home page telling them they do not have any favourited films currently | Same as expected |
| 7.4 | To test the "Recommendations" button | 20:28 | Button click | Redirects the user to the "Recommendations" page telling them that "There is no information available for the query you have entered" | Same as expected |
| 7.4.1 | To test the "New recommendations" button | 20:44 | Button click | Redirects the user to the home page telling them they do not have enough favourited and rated films to create a recommendation currently | Same as expected |
| 7.5 | Test news modal | 21:04 | Button click | A window will appear with the news about the website, same as when the user was not logged in | Same as expected |
| 7.6 | To test the "Account Settings" page button | 21:15 | Button click | Displays the "Account settings" page | Same as expected |
| 7.6.1 | Click on the "Forgot Password?" button | 21:24 | Button click | Will redirect the user to the page | Same as expected |

| | | | | | |
|---|---|---|---|---|---|
| | | | | where they can enter their email address, same as when the user was not logged in | |
| 7.7 | To test that the user is able to log out and can only log out once, will duplicate the current page and try and logout on both pages. | 21:32 | Button click | The first page which the user clicks logs out on will tell them that they have logged out successfully and the second page will tell them that they are not currently logged in | Same as expected |
| 8.0 | To show that the user will be redirected to any page they try and access before they are logged in. | 22:05 | Will edit the URL, so that it's route after the port is "/search-for-actor" | The user will be redirected to the login page so that they can login, and will not be shown the "Search for actors" page | Same as expected |
| 8.1 | To show that the user can login and will be redirected to that page | 22:30 | Username:- "Kian123"<br>Password:- "Password333" | The user will be redirected to the "Search for actors" page since they are now logged in | Same as expected |
| 9.0 | Send the user back to the "Search for film" page using the dropdown menu | 22:49 | Button click | The user will be redirected to the "Search for film" page | Same as expected |
| 9.1 | Show that the spellchecker works when searching for a film | 22:59 | Query:- "fwst and frious" | A menu will appear with the user's original query and what the spellchecker predicts. The spellchecker should predict it is "Fast-And-Furious" | Same as expected |
| 9.2 | Show that it will display films for the spellchecked option, once the user has clicked on "Fast-And-Furious" | 23:14 | Button click | A page will load with 10 films which are related to the search which the user has entered | Same as expected |
| 9.3 | Show that the order by feature works on the page | 23:52 | Menu option:- "Release↓"<br>Button click of button "Order" | The page will reload with the film with the newest release date by year at the top of the page and | Same as expected |

| | | | | | |
|---|---|---|---|---|---|
| | | | | the oldest film will be at the bottom, it will say at the top that the page is "Currently ordered by Release↓" | |
| 9.4 | Show all features available when looking at a specific film | 24:39 | Film used:- "Fast & Furious Presents: Hobbs & Shaw" | All features will work correctly | Same as expected |
| 9.4.1 | Show that the user can view the description of the film | 25:00 | Button click | The description button will disappear and the description of the film will appear | Same as expected |
| 9.4.2 | Show that the favourites button works | 24:46 | Button click | The favourite button will turn red and say unfavourite, as the film is now unfavourited | Same as expected |
| 9.4.3 | Show that the film can be unfavourited | 24:51 | Button click | The button will turn back green and say favourite again | Same as expected |
| 9.4.4 | Show that the user can view the rate menu | 25:10 | Button click | A menu will appear where the user can select the rating for the film | Same as expected |
| 9.4.4.1 | Show that the user can rate the film for each category | 25:18 | **Data entered into each section**<br>Overall:- 10<br>Comedy:- N/A<br>Actors:- 10<br>Quality:- 10 | The menu will update saying that the rating has been added | Same as expected |
| 9.4.4.2 | Refresh the page to show that it will tell the user that the film which was rated is currently rated or the recommendation has been liked or not | 25:41 | Page refresh | Will tell the user that the film has is "Currently Rated or the recommendation has been liked or disliked" | Same as expected |
| 9.4.4.3 | Show that the rating can be updated, will rate the film again | 25:59 | Same data which was entered in the "9.2.4.1" test but the Comedy is now "7" | The menu will update saying that the rating has been updated | Same as expected |
| 9.4.5 | To show that the directors button works and will display all of the directors for the film | 26:34 | Button click | A menu will appear with the director which directed the film | Same as expected |
| 9.4.5.1 | Will click on the | 26:40 | Button click | A menu will appear | Same as |

| | | | | | |
|---|---|---|---|---|---|
| | director which directed the film | | | with all off the information about this director | expected |
| 9.4.5.2 | Will click on the films which this director has directed | 26:50 | Button click | The user will be redirected to the page with all of the films which the director has directed in sets of 10s | Same as expected |
| 9.4.5.3 | Find the "Fast & Furious Presents: Hobbs & Shaw" film on the page as the director has directed it | 27:01 | Scrolling through page | Will locate the film on the page which the user was trying to find | Same as expected |
| 9.4.6 | For the "Fast & Furious Presents: Hobbs & Shaw" film show that the Actors button works, and will display all of the actors for the film | 27:10 | Button click | A menu will appear with all of the actors for the film | Same as expected |
| 9.4.6.1 | Will click on an actor to show the information about them | 27:18 | **Actor used:-** "Jason Statham" | Will display all of the information about this actor | Same as expected |
| 9.4.6.2 | Show all of the films which this actor has been in | 27:29 | Button click | Will display all of the films which this actor has been in sets of 10s | Same as expected |
| 9.5 | Show that the user is able to view more than just 10 films, by clicking on more results | 27:56 | Button click of "Results 11-20" | Will try and display another set of 10 films to the user if there are 10 films to display | Same as expected |
| 9.5.1 | Will find and the "Gnomeo & Juliet" film on the page | 28:06 | Scrolling through page | Will locate the film on the page which the user was trying to find | Same as expected |
| 10.0 | Send the user back to the "Search for film" page using the dropdown menu | 28:30 | Button click | The user will be redirected to the "Search for film" page | Same as expected |
| 10.1 | Open the year modal to filter the films, will try and find films released in a specific year | 28:35 | Query:- "fast and furious"<br><br>Lower:- 2017<br>Upper:- 2017 | Will redirect the user to the films page only displaying the films released in the year 2017 | Same as expected |
| 10.2 | Open the year modal to filter the films, will try and find films | 29:28 | Query:- " fast and furious " | Will redirect the user to the films page only displaying | Same as expected |

| | | | | | |
|---|---|---|---|---|---|
| | between a specific date | | Lower:- 2010<br>Upper:- 2017 | the films released between 2010 and 2017 | |
| 10.3 | To prove that when you set the lower bound of the year larger than the upper bound they will be swapped around | 30:31 | Query:- " fast and furious "<br><br>Lower:- 2015<br>Upper:- 2013 | Will redirect the user to the films page only displaying the films released between 2013 and 2015 | Same as expected |
| 11.0 | To open the language modal to filter the films for 1 specific language, this works in the same principal as the genre filter so will not need to test it | 31:24 | Query:- " fast and furious "<br><br>Language option selected:- "Russian" | Will display all of the films which are related to the search and have the language of Russian | Same as expected |
| 12.0 | Will filter the films which have either one of multiple genres, this works in the same principal as the language filter so will not need to test it | 31:58 | Query:- " fast and furious "<br><br>Genre option:- "Action", "Thriller" | Will display all of the films which are related to the search and have the genre of Action or Thriller | Same as expected |
| 12.1 | To show that the user cannot refresh the first page which is displayed and they will redirected to the homepage and told to not do this | 33:15 | Page refresh | The user will be redirected to the homepage and told to not try and refresh the page and to re-enter the query again | Same as expected |
| 12.2 | Re-enter the information to show that the user can refresh the first page as long as it is accessed through the "Results 1-10" option button at the bottom of the page | 33:53 | Page refresh | The page will load first with all the information, the page will then reload with a different URL, then once the user tries to refresh the page it will refresh. | Same as expected |
| 13.0 | Send the user to the "Search for Actors" page using the dropdown menu | 34:32 | Button click | The user will be redirected to the "Search for Actors" page | Same as expected |
| 13.1 | To show that when the user does not enter a query it will tell them to enter a query. The same principal is used on the "Search for Film" page but is not | 34:38 | Query:- "" | Will redirect the user to the homepage telling them to enter a different query, as the one they entered was too long or too short | Same as expected |

| | | | | | |
|---|---|---|---|---|---|
| | needed to be shown as shown here | | | | |
| 13.2 | To show that when the user enters a query with symbols which could be used in an SQL injection, they are rejected from the system, to prevent from attackers. The same principal is used on the all pages which take a user input but is not needed to be shown as shown here. It will try and delete everything from the Films table in the database, but will not succeed. | 34:54 | Query:- "1;DELETE * FROM FILMS WHERE FILMID<999999999" | Will redirect the user to the homepage telling them to not enter a query which contains unauthorized characters | Same as expected |
| 13.3 | Will search for an actor, and show that all of the best possible results for that actor are displayed, this feature works in the same principal as the directors so will not need to test for the directors | 36:25 | Query:- morgan freeman | The user will be redirected to the page with all of the actor which they are looking for | Same as expected |
| 13.4 | Will search for the actor again but will filter their date of birth to narrow down the search, this filter of the date of birth works the same as when the user is filtering the release date of a film. Will try and find actors which are born in a specific year only as it's the same principal as what was saw before | 36:52 | Query:- morgan freeman<br>Lower:- 1937<br>Upper:- 1937 | Will only display actors which were born in the year 1937 | Same as expected |
| 13.5 | Show that the page can be refreshed once the user has found the actor which they | 37:50 | Button click | The page will be refreshed with all of the information exactly the same | Same as expected |

| | | | | | |
|---|---|---|---|---|---|
| | want | | | | |
| 13.6 | Will click on the films which the actor has been in, to show all of the films | 38:03 | Button click | Will display all of the films which the actor has been in, in sets of 10, same layout as before | Same as expected |
| 14.0 | Send the user to the "Search for Directors" page using the dropdown menu | 38:34 | Button click | The user will be redirected to the "Search for Directors" page | Same as expected |
| 14.1 | To show that the user can search for directors, and filter their date of birth | 38:45 | Query:- "Steven Spielberg"<br>Lower:- 1940<br>Upper:- 1960 | Will display the director which the user is looking for | Same as expected |
| 14.2 | Will show that the user is able to view more than just 10 directors, if they use the more results option at the bottom of the page | Error - 39:18<br><br>Fixed - 39:49 | Button click of "Results 11-20" | The user will be redirected to the next page displaying another set of 10 films. | The user was redirected to the homepage as an error occurred |
| 15.0 | Click on the "Account Settings" page on the navigation bar | 40:40 | Button Click | The user will be redirected to the Account settings page, where it will tell them to change their password | Same as expected |
| 15.1 | To show that the user must enter information into the fields before they can attempt to change their password | 40:45 | Button click | The user will be informed that they must enter information into the fields | Same as expected |
| 15.2 | Tries to change the password for the user, the purpose is to show the validation for each field, and that the password gets changed | 40:52 | **Information entered into each field**<br>Security Question:- (Correct answer){as it is random I do not know what the security question will be yet}<br>Current password:- "Password333"<br><br>New password:- "a" | The user is told that the length of the new password must be between 6 and 128 characters | Same as expected |
| 15.2.1 | To show that the user must enter the correct security question | 41:36 | **Information entered into each field**<br>Security Question:- | The user is told that the password or the security question is incorrect | Same as expected |

| | | | | | |
|---|---|---|---|---|---|
| | | | (Incorrect answer)<br><br>Current password:- "Password333"<br><br>New password:- " Newpassword12345 " | | |
| 15.2.2 | To show that the user must enter the correct password | 42:13 | **Information entered into each field**<br>Security Question:- (Correct answer)<br><br>Current password:- "Elephants"<br><br>New password:- " Newpassword12345 " | The user is told that the password or the security question is incorrect | Same as expected |
| 15.3 | To show that the user's password is changed, and that they have received an email telling them it has been changed | 42:42 | **Information entered into each field**<br>Security Question:- (Correct answer)<br><br>Current password:- "Password333"<br><br>New password:- "Newpassword12345" | The user will be redirected to the homepage telling them that they have had their password changed. An email will also be sent to them to tell them it has changed. | Same as expected |
| 16.0 | Will logout and log back in to show that the new password works and that it has been changed | 43:45 | **Information entered into the login screen**<br>Username:- "Kian123"<br><br>Password:- "Newpassword12345" | The user will be redirected to the homepage and told that they have logged into the website successfully and welcomes their username | Same as expected |
| 17.0 | To show that the user can search for 2 films at the same time and click the next page without them interfering with each other. | 44:27 | Screen1 Query:- "Titanic"<br><br>Screen2 Query:- "Karate" | Both screens are able to look at all next pages, without them interfering with each other | Same as expected |
| 18.0 | Show that the Favourites list is empty | 46:18 | Button click | There will be no films in the Favourites page | Same as expected |

| | | | | | |
|---|---|---|---|---|---|
| 18.1 | Will rate and favourite a total of 20 films altogether | 46:27 | **Films favourited -**<br>A Beautiful Mind<br>Rain Man<br>Iron Man<br>The Fate of the Furious<br>Ice Age<br>Ice Age: Continental Drift<br><br>**Films rated -**<br>Terminator Genesis<br>Interstellar<br>Frozen<br>Star Wars: Return of the Jedi<br><br>**Films favourited and rated -**<br>Avengers: Age of Ultron<br>Jumanji<br>Central Intelligence<br>Spider-Man<br>Undercover Brother | All the films will be rated and favourited successfully | Same as expected |
| 18.2 | Will show the films in the favourites list | 47:56 | Button click | The films favourited will appear in the list | Same as expected |
| 19.0 | Will go to the "Recommendation" page, and click the "New Recommenations" button | 48:35 | Button click | The user will be redirected to the home page and told that their recommendations have been created | Same as expected |
| 19.1 | To show that the recommendations have been created for the user, the user will go back to the "Recommendation" page | 48:57 | Button click | The recommendation page will load with all of the recommendations which have been created | Same as expected |
| 19.2 | To show that the user is able to rate, favourite, like and dislike the recommendations | 49:11 | Button click | All the buttons work correctly and the information is updated | Same as expected |
| 19.2.1 | To show that the rate button/modal works | 49:35 | Button click | The rate window will appear, same as before | Same as expected |
| 19.2.2 | To show that the | 49:24 | Button click | The button will turn | Same as |

| | | | | | |
|---|---|---|---|---|---|
| | favourite button works | | | red and say unfavourite, same as before | expected |
| 19.2.3 | To show that the like recommendation button works | 49:13 | Button click | The button will disappear as the user has commented saying they like the recommendation, but the dislike recommendation button will still bet there | Same as expected |
| 19.2.4 | To show that the dislike recommendation button works | 49:46 | Button click | The button will disappear as the user has commented saying they dislike the recommendation, but the like recommendation button will still bet there | Same as expected |
| 19.3 | To show that when the user refreshes the page, the films which the user has commented on the recommendation, will tell the user that they have done so and that the button to comment again on it will appear again. | 50:09 | Page fresh | The buttons to comment on the recommendation for both good and bad will appear and the films which have had their recommendations commented on will have a message stating it has. | Same as expected |
| 20.0 | To show that the website works on mobile devices and that are not on the same network, will be using 4G, will go onto the website on a different device to what it is hosted on and which is a mobile device not on the network, and login | 50:46 | Username:- "Kian123"<br><br>Password:- "Newpassword12345" | The user will be redirected to the homepage saying that they have logged in | Same as expected |
| 20.1 | To show that all of the favourited films are still there from before. Will go to the "Favourites" page | 51:32 | Button click | All of the favourited films from before will load and appear | Same as expected |

| | | | | | |
|---|---|---|---|---|---|
| 20.2 | To show that when the user logs in their recommendations update automatically. Will go to the "Recommendations" page | 52:04 | Button click | There will be new recommendations on the recommendation page for the user. | Same as expected |
| 21.0 | To show that when I go to the "Leaderboards" page through the dropdown menu, and click on the "Top rated" films, there is now films as I have rated films, and they are now in the database | 53:20 | Button click | The films which the user has rated with the best overall rating will appear | Same as expected |
| 21.1 | To redirect the user back to the "Leaderboards" page, will press the back button | 54:11 | Button click | The user will be redirected back to the "Leaderboards" page | Same as expected |
| 21.2 | To show that when I click on the "Funniest" option, there are now rated films there which have the highest Comedy rating. | 54:16 | Button click | The films which the user has rated with the best comedy rating will appear | Same as expected |
| 22.0 | To show that the system allows multiple users on it at the same time, will log onto the other account created earlier on. | 55:29 | Login form information<br>Username:- "xxx"<br><br>Password:- "Password333" | The user will be redirected to the homepage and told welcome | Same as expected |
| 22.1 | Show that the user has an empty favourited list, to prove that the accounts are not linked together with the other people | 56:26 | Button click | Redirects the user to the home page telling them they do not have any favourited films currently | Same as expected |
| 22.2 | To show that the user has no recommendations, and that the accounts are not linked and are individual. | 56:37 | Button click | Redirects the user to the "Recommendations" page telling them that "There is no information available for the | Same as expected |

| | | | | | |
|---|---|---|---|---|---|
| | | | | query you have entered" | |
| 22.3 | To show that the top rated films are for everyone and not just an individual user, will click on the "Leaderboards" page through the "Search" dropdown menu | 56:44 | Button click | Will redirect the user to the "Leaderboards" page | Same as expected |
| 22.3.1 | Will click on the "Funniest" option, to show that the films with the highest comedy rating are the same as the ones on the other account | 56:55 | Button click | Display the page with all of the films with the best comedy rating | Same as expected |
| 22.3.2 | Have both accounts open on screen to show that they are the same films | 57:17 | Split screen | The films will be exactly the same, on both sides of the screen | Same as expected |
| 23.0 | Will redirect both user's to the homepage, so that they can click on the "Search for films" image, once on the homepage will click on the "Search for films" image | 57:24 | Button click x 2 | Will redirect the user to the homepage and then redirect the user to the "Search for films" page | Same as expected |
| 23.1 | Will search for films at the same time on both accounts, to show that there are no interference between them | 57:32 | Films searched for<br><br>Account1:- "Forever After"<br><br>Account2:- "Hop" | Both films will appear on each page for the user | Same as expected |
| 23.1.1 | Will favourite the film which was being looked for both users | 58:04 | Button click | The favourite button on both accounts will turn red and say unfavourite | Same as expected |
| 23.1.2 | Will click on the "Results 21-30", to show that it is correct for both user's and no interference. | 58:13 | Button click | The correct films will be displayed on both sides of the screen | Same as expected |
| 23.2 | Will redirect both users to the "Favourites" page and show that the favourited films are still unique for the | 58:47 | Button click | All of the correct favourite films will be displayed to each user. | Same as expected |

| | | | | | |
|---|---|---|---|---|---|
| | user, and that the user on Account2 only has 1 favourited film | | | | |

## Test 12.1

The reason that the user cannot refresh the page specifically here, is because in the JavaScript function to the python function, the data which is used is deleted, this cannot be stored in a cookie as if the user has multiple tabs open at once then it will override some of the other data. Using the timestamp on the cookie also does not work as if the website lags and the user is searching for 2 films at the same time then it will cause the data to intertwine. This is the best solution unless the user clicks on the "Results 1-11" page.

## Test 14.2

The problem was that when I had the title of the page, I tried to add an integer to a string, this caused a concatenation problem. I needed to wrap the string around an "int" function.

**Code with the problem**

```
return render_template("search/people/display_people_information.html" , actor_director = information_about_people,
    person_type = is_actor, results = all_people_ids, title= f"{is_actor} Information: {page_number+1}")
```

**Code fixed**

```
return render_template("search/people/display_people_information.html" , actor_director = information_about_people,
    person_type = is_actor, results = all_people_ids, title= f"{is_actor} Information: {int(page_number)+1}")
```

# Evaluation

Here I will evaluate my final product of the system which I have designed and made.

## Objectives met

These are all of my objectives which I originally defined in my "Analysis" section of my documentation, and here I state if I have met them or not and if I have not or have not fully met them then I will describe why I have not.

| Objective | Has objective been met | Additional information |
|---|---|---|
| **8. Platform to use** | | |
| a. Website based so that can be cross platform and can access anywhere with an internet connection, and any device. | Yes | |
| b. Will be using the python flask module, for the web framework for the code of the website, as it is lightweight and fast | Yes | |
| c. Fast and accurate searching for films, actors and directors the user wants to find information about. | Yes | |
| | | |
| **9. Website** | | |
| a. Display everything on a website | Yes | |
| b. Have multiple pages which have different links for the user to find information about | Mostly | The friends and chatroom feature were not added so there is no link to it for the user |
| i.Register | Yes | |
| i.Login | Yes | |
| i.Forgot password | Yes | |
| .Change password | Yes | |
| .Search for film | Yes | |
| i.Search for actor | Yes | |
| i.Search for director | Yes | |
| i.Favourites | Yes | |
| .Recommendations | Yes | |
| .Friends | No | This feature was not added to the website I will talk more about this later on |
| i.Chatroom | No | This feature was not added to the website I will talk more about this later on |
| 1. Completely random | No | |
| 2. Random with interest | No | |

| | | |
|---|---|---|
| 3. Chosen by user | No | |
| c. The website must be easy to use, easy to understand | Yes | |
| d. It must also be as fast as possible loading pages so that people do not get bored waiting | Yes | |
| i.Waiting image appears if the page does not load instantly | Yes | |
| i.All loads between pages are below 3 seconds | Yes | |
| | | |
| **10. User accounts** | | |
| a. User can Register if haven't before. | Yes | |
| i.Will tell the user if the username or email entered already exists | Yes | |
| i.Adds information to the database with the password and security questions encrypted, including; | Yes | |
| 1. Password | Yes | |
| 2. Security questions | Yes | |
| 3. Username | Yes | |
| 4. Name | Yes | |
| 5. Email | Yes | |
| 6. Date Registered | Yes | |
| 7. Date of birth | Yes | |
| i.Send confirm nation email of the account being created to the user. | Yes | |
| b. User can Login if they have registered. | Yes | |
| i.Once the user has logged in they will be able to access all their information for; | Mostly | The friends, chatroom and history feature were not added so there is no information for that |
| 1. Favourite films saved | Yes | |
| 2. History of the past 5 films they have looked at | No | I did not add this to the website, as if loads of user's sign up to the website, then this table will be filled with a lot of data and the database will become full meaning I will run out of storage space, and as I am not using a remote server or cloud storage to store this information I cannot afford to do so. |
| 3. Friends list | No | This feature was not added to the website I will talk more about this later on |
| 4. Recommended films | Yes | |
| c. User can change their password | Yes | |
| i.If the user wishes to change their password they may do so | Yes | |

| | | |
|---|---|---|
| i.If the user has forgot their password they can request it by entering their email address and it will sent them an email of their current password | Yes | |
| | | |
| **11. Searching** | | |
| a. Once the user has logged in they will be able to search for a film, actor or director | Yes | |
| i.Display all information about films for either when the user searches for a film, looks at their favourited list or is looking at their recommendations are; | Yes | |
| 1. Display release date | Yes | |
| 2. Display title of film | Yes | |
| 3. Display genres related to film | Yes | |
| 4. Display languages related to film | Yes | |
| 5. Display runtime of film | Yes | |
| 6. Display colour of film | Yes | |
| 7. Display actors in film | Yes | |
| 8. Display directors who directed film | Yes | |
| 9. Allow the user to favourite and rate any film displayed to them. | Yes | |
| i.Display all information about actors and directors (information displayed for both are in the same format) | Yes | |
| 1. Name of person | Yes | |
| 2. Date of birth | Yes | |
| 3. Death date (if dead) | Yes | |
| 4. Place of birth | Yes | |
| 5. Spouses and children | Yes | |
| 6. Height | Yes | |
| 7. Films acted in or directed | Yes | |
| | | |
| b. When the user clicks on their favourite list it will display all of their favourited films only. | Yes | |
| c. The user can click on their recommendations page to see all of their recommendations | Yes | |
| d. Search for friends they wish to add to their friends list | No | This feature was not added to the website I will talk more about this later on |
| | | |
| | | |
| **12. User can rate films** | | |
| a. The user can rate any film they see, it will display a rating system consisting of 4 categories from 1 to 10 where 10 is the highest rating and N/A is | Yes | |

| | | |
|---|---|---|
| for unknowns (the user did not rate in this category), the categories are; | | |
| i. (1-10) Comedy value, how funny was the film | Yes | |
| i.(1-10) Overall, the user's overall opinion on the film | Yes | |
| i.(1-10) Quality, how well was the film produced | Yes | |
| v.(1-10) Actors, how well did the actors act in the film | Yes | |
| b. This will be used in the recommendation process, where it will try and find films, based off of what the user has favourited films and their favourited films, | Yes | |
| i.If a film has a rating of "N/A" then the rating will be set to 5 as it is the middle value. more based upon that specific rating of that film. | Yes | |
| i.Using the values which the user has rated, it will use these to try and balance each film depending on its attributes such as; genre, actors and directors | Yes | |
| c. All of this information will be stored in a database, but the user will not be able to see the ratings once they have rated the film | Yes | |
| | | |
| **13. Recommendation algorithm** | | |
| a. Identify the films which the user will like based upon their favourited and rated films. As long as the user has enough rated and favourited films | Yes | |
| b. The algorithm will try and predict 30 films for the user at once | Yes | |
| i.The user will be able to click on a button to get new recommendations whenever they want | Yes | |
| i.There will be an option to say if the recommendation is good or bad to help predict the recommendation algorithm | Yes | |
| c. Use the user's ratings if there are any when calculating the recommendation for the user, it will also filter out the films which the user does not want if the rating is too low. Will prioritize films with a higher rating over others, and discriminate against films which have a lower rating. | Yes | |
| i.I will use overall rating for the genre | Yes | |
| i.The actors rating for the actors | Yes | |
| i.Quality rating for the directors | Yes | |

| | | |
|---|---|---|
| v.The comedy weighting will be used for the genres if there is a "Comedy" genre in there | No | This rating was not used for the genre, as it would be too biased towards comedy films, instead this was used in the leaderboards page and used to display all of the films with the highest comedy rating. |
| d. The algorithm will predict films for the user by finding; | Yes | |
| i.All the films which have the most common and liked actors and directors in the user's currently rated and favourited films | Yes | |
| i.Once those films have been found it will look at the details about each film to try and narrow down the recommendation set by; | Yes | |
| 1. Giving films with the most common genre combination in the user's favourited and rated films a higher weighting over other films. | Yes | |
| 2. Finding the average runtime to try and remove films which the user will find too long or too short. | Yes | |
| 3. Find the most common language in the user's favourited and rated films, and remove all of the films which are not this language as the user will not be able to understand it otherwise. | Yes | |
| a. Will potentially try and find second and third most common language as the user may be bilingual | Yes | |
| 4. See if the user likes black and white films or just films in colour, if it is a mixed set of film colours, then will create a proportion of which should be in colour and which should be in black and white | Yes | |
| e. I will design a machine learning algorithm to effectively use all of this information and to predict the best films to the user for them to watch. | Yes | |
| | | |
| 14. **Chatroom** | | This whole section was not added to the website I will talk more about this later on, in its own separate section. |
| a. A chatroom for users with similar interests to talk about films they like | No | |
| b. Allow users logged in or not to enter a chatroom | No | |
| c. If the user is logged in give them an option to join a chatroom with people with similar interests as them | No | |

| | | |
|---|---|---|
| i.It will find people with similar interests as them, by finding the films they both have highly rated and favourited. | No | |
| i.Save the chat and who sent them in a JSON file with a timestamp, so that the user's can see the history of the chat when they are offline. | No | |
| d. Allow the user to join a chatroom with the people they | No | |
| i.They can add the people from their friend list by clicking on their name or by searching for a user's user identification number. | No | |

## Reason for not adding chatroom

One of the main reasons why I didn't add the chatroom to my system, is because when I was first designing my project and I was asking potential users, they told me that this was not necessary but would be nice, for this reason I decided to leave this feature out of my design, and if later on in the future I decide to come back to this project I will add this on.

Another reason why I did not add this on is since that if I first launch my website, I will not have enough user traffic for users to fully utilise this feature and it would just be sitting there taking up extra space.

The final reason why I did not add this is that I would need to store every single chat message which was sent by every user in the database or a JSON file so that if a user is offline and someone sends a message, then they will be able to see the messages when they log back in. This means that if somehow, I started to gain a large amount of traffic my server would not be able to handle this as it does not have enough storage space to store all of these messages and timestamps.

## Reason for not adding friend feature

Since the chatroom feature was not added to the system, there was not really a need for the friends feature as users would not be able to fully utilise it as they could not message other users.

When I was surveying different potential users at the beginning of my project, the majority told me that this feature as well as the chatroom was not necessary, this is the main reason as to why I did not go any further with this idea.

If later on I decide to improve on this project, then I will definitely add this feature as one of the first things I do.

# Timeline

This is how I spent my time when I was doing my project, the gathering of the information for the database took the longest time out of everything unfortunately.

| What I did | Date started | Date finished |
|---|---|---|
| Wrote the code to gather all of the titles of films and the link to them on IMDB | 11/03/2019 | 17/03/2019 |
| Ran the code which I previously written and stored all of the information in JSON files, a file for each year. Had to leave the machine on overnight which was running the script to gather the links | 18/03/2019 | 24/03/2019 |
| Wrote the code which was used to gather all of the information about the films. | 22/03/2019 | 30/03/2019 |
| Ran the code which was previously written and stored all of the information in a database, in one table which was not normalized. Had to leave the machine which was running this on again. This took significantly longer to run than I originally thought, as I did not consider the speed of my internet with web scraping each individual page into the equation. | 31/03/2019 | 06/05/2019 |
| Wrote the code which was used to normalize the database and store all similar information in a linking table linking it to the necessary films, this was for genres and languages. I also reformatted the way some of the information was stored in the database so that it was more user friendly. | 04/04/2019 | 10/04/2019 |
| Ran the code which was previously written and stored it in a new database, which was fully normalized. This ran quite fast as it was all done locally and did not require the internet. | 08/05/2019 | 10/05/2019 |
| Wrote the code which was used to gather all of the information about the actors and directors. | 15/04/2019 | 22/04/2019 |
| Ran the code which was previously written and stored it in a new database, with tables linking it to the information about the films, even though the films were not in that database. For this reason I moved this information into the main fully normalized database next. This took significantly longer than I thought, and even longer than the code to find the information about the films. | 12/05/2019 | 17/07/2019 |
| As I stored all of the information for the Films, Actors and Directors in separate database, I needed to bring everything together into one database. And fully normalize my database. I wrote a script which would do this for me. | 1/05/2019 | 5/05/2019 |
| Ran the code which was previously written. | 18/07/2019 | 22/07/2019 |
| Wrote the code which was used to gather the images of the films, actors and the directors | 24/04/2019 | 27/04/2019 |
| Ran the code which was previously written, this surprisingly ran a lot faster than what I thought as the code to find the information about the films, actors | 23/07/2019 | 28/07/2019 |

| | | |
|---|---|---|
| and directors took significantly longer. Inserted this information into the tables which stored the information about the films, actors and directors in a new field call “imgURL” | | |
| Realised that the currency for the “Budget” and the “GrossRevenue” was not all converted to the same currency, I wanted to store everything as USD so I had to research how to convert all of the values in the database and update them, and know what the current conversion rate was. Then I had to write the code for this. | 02/08/2019 | 9/08/2019 |
| Ran the code above, which converted all of the currencies in the database to the same one. | 11/08/2019 | 15/08/2019 |
| Finally, as the database was created and all of the information in the correct format and normalized, I was able to start coding the website. | 17/08/2019 | 27/12/2019 |
| Designed the template layout of the website, of how it will look, and the logo, and all of the images which will appear | 17/08/2019 | 20/08/2019 |
| Wrote the search algorithm for the films, actors and directors | 21/08/2019 | 04/09/2019 |
| Designed and wrote the spell-checking algorithm for the search for films | 06/09/2019 | 08/09/2019 |
| Designed the layout and organized the HTML file on how it will take the information in and display it to the user, this includes all of the extra features such as ordering and the next page | 15/09/2019 | 23/09/2019 |
| Designed the layout and organized the HTML file which will display the information about an actor or director to the user | 25/09/2019 | 01/10/2019 |
| Wrote the python code which is used to take in the query of the user, when they are searching for a film, actor or director, and send it to the searching algorithm to display it, using the filters | 01/10/2019 | 07/10/2019 |
| Wrote the python code to go to the next page for the user, for both films, actors and directors | 07/10/2019 | 07/10/2019 |
| Wrote the python code to reorder the page by a specific attribute | 08/10/2019 | 10/10/2019 |
| Linked the film page to the actor and director page, so that when the user clicks on a specific actor or director, they will be redirected to that page | 11/10/2019 | 13/10/2019 |
| Designed and wrote the algorithm to display leaderboards to the user, this is done systematically and added to the database for the top films which users have rated | 15/10/2019 | 28/10/2019 |
| Added the features to the film page which is displayed and wrote the python code which links to it so that the user can, rate, favourite and view the description of the films. | 30/10/2019 | 04/11/2019 |
| Added the feature for the user to view their favourited films | 05/11/2019 | 05/11/2019 |

| | | |
|---|---|---|
| Wrote the recommendation algorithm. | 06/11/2019 | 27/11/2019 |
| Added the recommendations to the website, as well as the feature for the user to get a new recommendation, and comment on the recommendation | 27/11/2019 | 30/11/2019 |
| Designed the registration process for the user, and implemented the encryption algorithm into it, so that the user can register securely | 02/12/2019 | 07/12/2019 |
| Added the register feature onto the website so that they can register their account | 07/12/2019 | 10/12/2019 |
| Added the feature for the user to login to the website using their registered account. | 11/12/2019 | 12/12/2019 |
| Linked together all of the recommendations, favourited films and rated films to that specific user logged in using the sessions module from flask. | 12/12/2019 | 15/12/2019 |
| Added the feature so that the user automatically logs out after a certain amount of time and as well as the ability for the user to logout whenever they want. | 15/12/2019 | 16/12/2019 |
| Added the feature for the user change their password if they wish | 16/12/2019 | 18/12/2019 |
| Added the feature where the user can enter their email and request their current password to their email address if they forgot it | 18/12/2019 | 18/12/2019 |
| Wrote all of the email templates for the user and linked them to the action which the user makes on the website, so that an email is sent to them, depending on their action. | 19/12/2019 | 21/12/2019 |
| Read through all of my code, and HTML documents and check my comments to make sure that everything is correct and accurate. Made any changes which were needed and spot and removed all error with a bit of light testing | 22/12/2019 | 27/12/2019 |

# User's opinions on the outcome

I decided to ask two of the people who I originally surveyed in my analysis about their opinion on the final product.

## Questions asked

1. Are you happy with how the website looks?

2. Are all the features added useful and you would use them?

3. Would you use this system instead of your current system?

## Response from person 1

This is the same person 1 from the analysis section.
Teenage male

1. Yes, I am quite happy with how the website looks

2. From the features I have seen they have all been very useful for what I needed/wanted to do.

3. I would use this system. It's fast and simple to navigate will the information already present rather than searching deep into it.

### Response

I am happy that I was able to design and create a system which caters for this user's needs, and that if they wish to use it in the future when its live it has all the features which they wanted.

## Response from person 3

This is the same person 3 from the analysis section.
Teenage male

1. Yes, I'm quite happy with how it looks, the system is quite intuitive and easy to work with. the greenish blue colour was a good choice and doesn't hurt my eyes.

2. Yes, most if not all of the features (such as the fast search utility, the login system and the favouriting options) are quite useful and not out of place for your system.

3. I could see myself using this as a component in my consumption of films and other media. The search facility is very well made, and the recommendations are surprisingly good while also providing a variety for which films I should watch.

### Response

I am pleased that I was able to live up to this user's expectation with my final product designed and created, and that their eyes are okay. I am glad that most of the features which I added will be fully utilised by this user if this website is to go live properly in the future and that they are impressed by my recommendation algorithm. As they mentioned "other media" has given me the idea to add tv shows to the website later on in the future, and potentially rename it so that it is more than just a "Film Recommendation System".

# My opinion on the outcome

I am honestly, really happy of the outcome of my project, and am proud of what I have accomplished and that the user's which I was able to interview again were impressed by the outcome as well. However I am somewhat annoyed it took me so long to complete it, but I am still

proud of my outcome. It has made me feel sad about it at some points but seeing the end result has made it worth it, I am likely going to work on and improve on this project more in the future and potentially make it available to the public if I can get a domain and a server to run it from.

## What I want to add later

As I am really impressed with the outcome of this project, I have decided to work on it again in the future. Some of the features I want to add on in the future are these in order of importance;

The chatroom and friend feature, as I wanted to add this on before but as it was not that popular with the potential user's I interviewed I decided to hold off on it.

Periodically update the database with films, actors and directors, for instance once every week I will do a search for new information available and update my database with it. This also has led me into wanted to increase the range of films which are stored in my database, so that it is more international.

I also want to add on the ability for premium users (paid users) to add a film, actor or director to the database, but first the information would be checked by moderators.

The ability to play clips of the film through the website if the user wants to see the trailer.

Add the ability for the user to view more than just films on the website, adding tv shows as well in the future. This is what the third person I originally survey stated in their second interview as a nice feature they would use.

Finally, I want to add the feature where the user can view their history but instead, it is of all films ever searched for and actors and directors and friends. This will be more challenging as I will need to make sure my server's storage is large enough to store all of the information depending of the traffic of the website.