COMP9900 Project Report

Submitted by,

Team TROJAN

<u>Group Members</u>

Rafael Formoso (z5220888)

Sai Sree Kavya Musty (z5240707)

Chirag Panikkasseril Unni (z5241855)

Longjuan Sun (z5228211)

Table of Contents

Overview

1. Introduction

People who are passionate about movie and are cinephiles are constantly browsing for movies and reviewing the ones they've watched. A web-based movie recommendation site can help readers keep track of all movies they want to watch, browse for new movies, and get personalised recommendations to find the next movie to watch. It also aims to help users find community by exploring the wishlist of reviewers who leave a review for a movie and might possibly find movies that they could possibly like. Though this is the primary aim of FilmFinder, it is not limited to this. Users can create wishlist, search and filter searches, view other users' wishlists, leave review, comment on other reviews, like and dislike any review and browse random movies. Our team aims to provide all these services in an efficient and user-friendly way to the users.

2. Project Description

The online website should be user driven where each FilmFinder can create a profile and login. This should enable FilmFinders to use other functionalities. The platform should allow them to search and find movies they are interested in and find other FilmFinders with similar interest. The search option should enable the user to search using keywords which should bring up results based on the movie name director name or actor name and genre as well as descriptions. The generated result page should have list of Movies and their average ratings provided by other FilmFinders. One should be able to view the detailed page for the selected movie from the list populated. The detailed page should provide name of the movie, a small description, genre associated, film director, cast, latest average rating given by FilmFinders, it should also show the reviews that other FilmFinders have posted about the movie. The logged in FilmFinder should be able to add or change their rating for the movie, from a scale of 0 to 5 and be able to add, edit or remove their review to the movie. The FilmFinder should be able to add or remove these movies which they are interested into their profiles Wishlist of movies they would like to watch. Additionally, a FilmFinder should be able to view a list of similar movies related to the selected one from search list. A similar movie suggestion list should be populated based on latest review history and other attributes which the user selects like director, genre, etc. The landing page should allow FilmFinders to browse through the platform's movie database based on director, genre or movies based on popularity where it should be sorted on descending order of rating and then with alphabetically for movies with same ratings. The website should have additional feature which enable a user to add another reviewer to blacklist. Once added the FilmFinder should not be seeing their reviews for the movies and should not influence their ratings for the movie's hereafter. The detailed page should allow the FilmFinder to like or dislike a review which another FilmFinder has put for the Movie and should be able to sort these reviews based on popularity or date. The page also should have option to share the page via a link which can be copied to use it to share on other social media platforms. Like Wishlist, there should be collections of movies which a user can create, which any other FilmFinder in the platform should be able to view so that they can find like-minded people in the platform.

3. System architecture

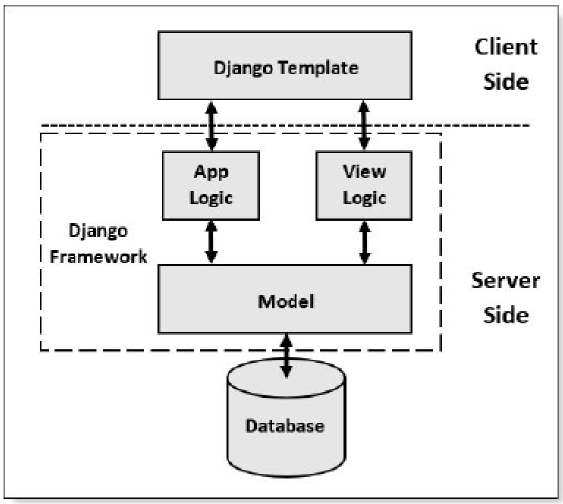


Fig. 1. A Holistic view of Django's Architecture

This project is heavily laid on Django architecture. All displaying web pages come from Django Template. The database is also a built in sqlite3 provided by the Django migrations command. The server the project is hosted on is the local server of the system it is run on.

3.1 Presentation layer

The team has elected to develop the website's user interface using HTML, CSS, and JavaScript, making full use of Django's templates to develop a rapid prototype of the website. Our direct use of Django's templates implies that the application will not have the same level of responsiveness provided by front end frameworks which support the development of single-page applications. However, we note that the nature of our website does not require or place any emphasis on real-time screen updates, and will not be facing the same level of user interaction as a fully featured social media website such as Facebook. Bootstrap is used as our website's CSS framework, providing design templates for many common design elements, such as fonts and user input components. This allowed us to standardize elements of our user interface across web pages without the need to rewrite style sheets for many common components to achieve an aesthetically pleasing look for both mobile and desktop devices.

3.2 Business layer

Django + Python Script: Django is a powerful frontend web tool with python. It can provide highly interactive web experience and comprehensive python libraries. By using python script, it can separate the front end and backend. It can handle the model and achieve data extraction, data storage, and recommendation systems. It connects presentation layer and data layer. Models and views are created using the Django template.

3.3 Data layer

In this project, the team use sqlite3 which is one of Django's default database to store, analyze and extract users', wishlist, banned list, movies, reviews, and recommender's data.

In addition, the IMDb API is used to get movies' posters, imdb rating, rotten tomato score, etc. The data layer is connected with business layer.

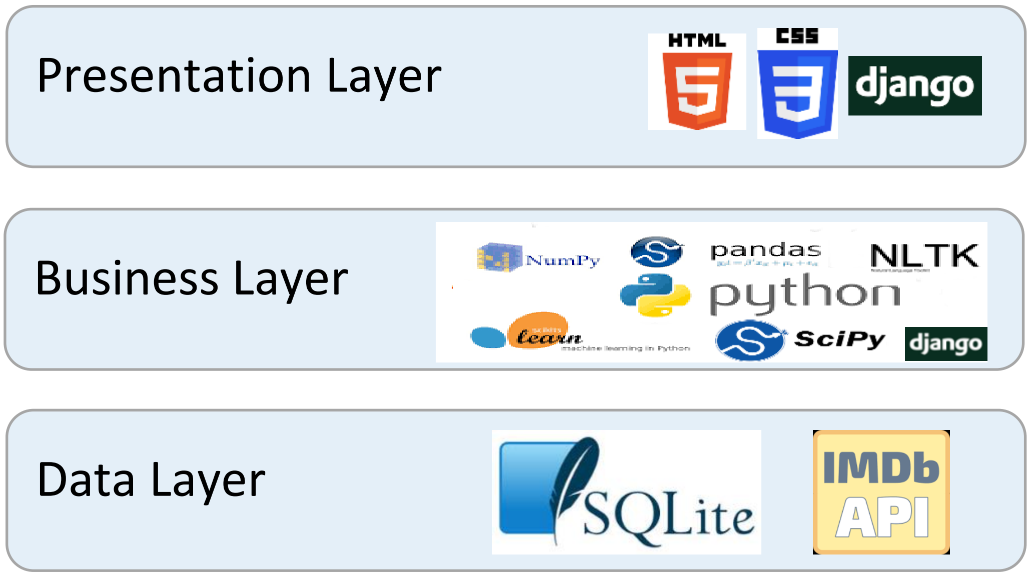


Fig. 2. Layers view of System Architecture

Functionalities and Implementation Challenges

1. SPRINT-1:

1.1 CWT-3: As a user, I want to register an account so that I can use the services on the website.

Fig. 3. The registration option on the home page

In this user story the users can register themselves to use functionalities inside the webpage. User is first taken to home page from there he could click on the nav bar link on top right corner "Sign Up". This takes the user to register page.

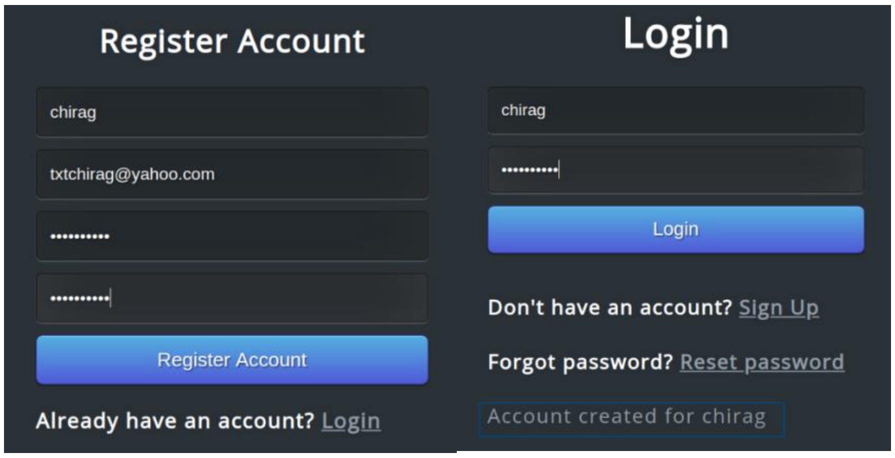

Register Account
chirag
txtchirag@yahoo.com
••••••••••
••••••••••
Register Account
Already have an account? Login
Login
chirag
••••••••••
Login
Don't have an account? Sign Up
Forgot password? Reset password
Account created for chirag

Fig. 4. Registration page and Login page

In register page the user is asked to fill-up the details as in the picture. Where the user will be providing a unique username to the database and an email id which can be used to reset their password. Password option must satisfy certain conditions which uses standard Django password form, and the user must confirm the same password on the subsequent field. Once filled, the user can click on "Register Account" button to create their unique profile in the database. This will in turn allow you to use the websites full functionality. The user is also having option to go to Login page by clicking on the "Login" link as well.

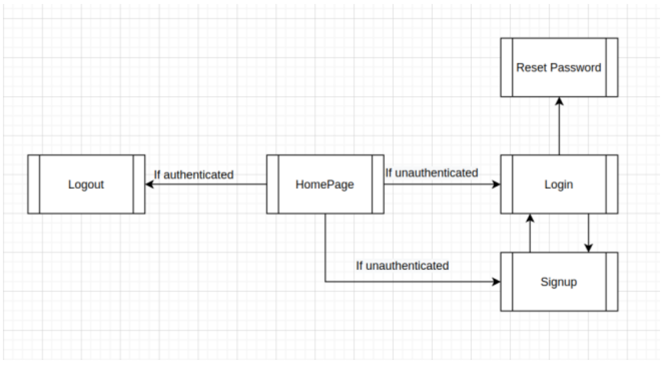
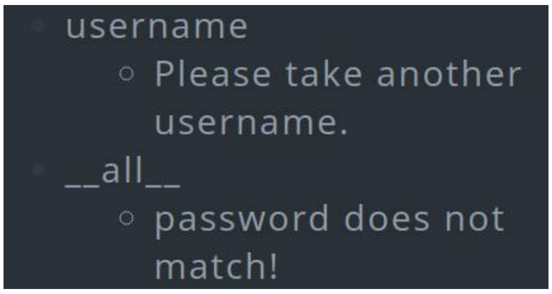


Fig. 5. Registration view function work flow

The page then redirects you to Login page with a message "Account created for <username>". One can access the same page from the home page Navbar as well.
The following error message will be prompted if any of the condition fails.



- username
  - Please take another username.
- __all__
  - password does not match!

Fig. 6. Error message prompt while registration

The challenges faced were to setup username and email ID as unique in the database. This provides the opportunity to send email for resetting the password. We took care of this by using Django's "CreateUserForm" and expanding its logic also applied frontend check to verify the user enter data appropriately. This was done by Django Forms where the integrity of unique user, password match and email verification were handled.

1.2 CWT-4: As a user, I must be able to log in into my FilmFinder account so that I could use the services.

In this user story, the user can make use of his registered username and corresponding password to login to the unique account. For this he can navigate to the Login Page by clicking on the Navbar link on homepage on top right corner "Log In". This redirects the user to Login page.

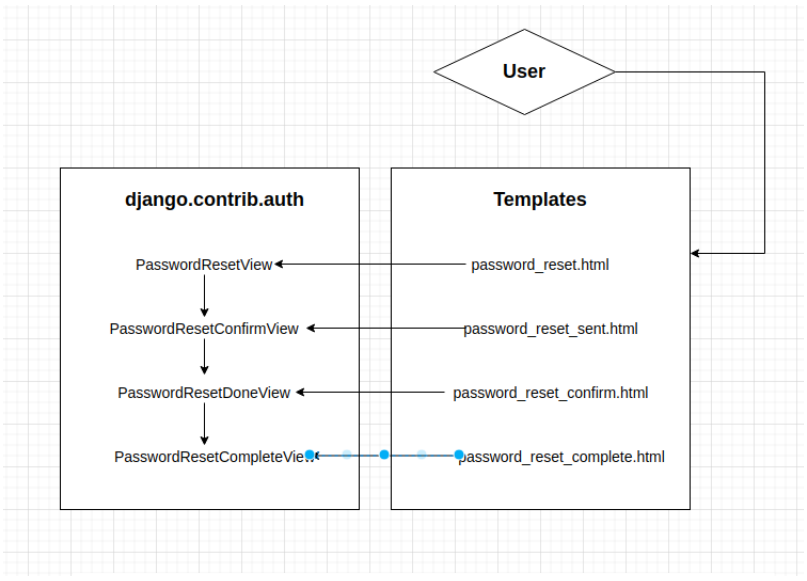


Fig. 7. Login work flow

Here the user is provided with a form to be filled, first takes the registered username and the next the corresponding password as shown in pic in the above user story. The user can fill the same and click on the "Login" button which will log the user in to the website and then redirect them to homepage. If any error in password or username the page displays appropriate error message and asks the user to try again.

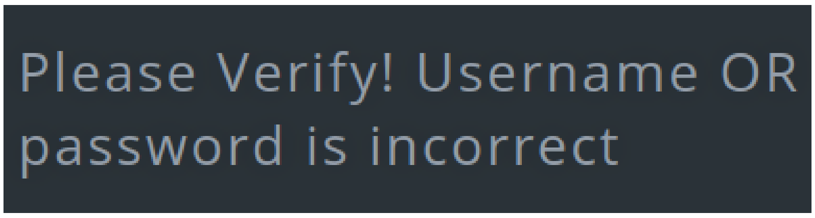

Please Verify! Username OR password is incorrect

Fig. 8. Error message prompt if username or password is incorrect while logging in

Once logged in the user can see additional links in the navbar such as the signed "username" and "Logout" at the top right corner and links for "Reset Password ", "Wishlist" and "Banned List".

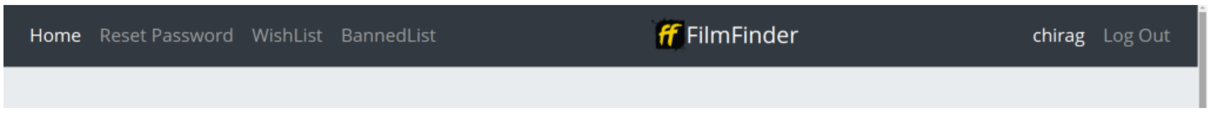
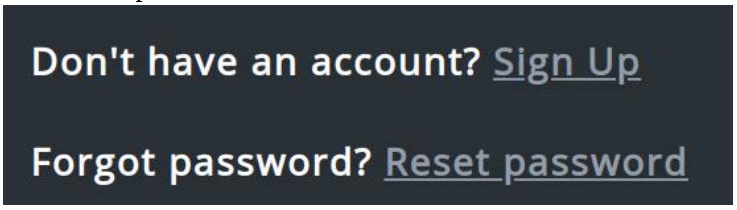


Fig. 9. Homepage navigation bar after login

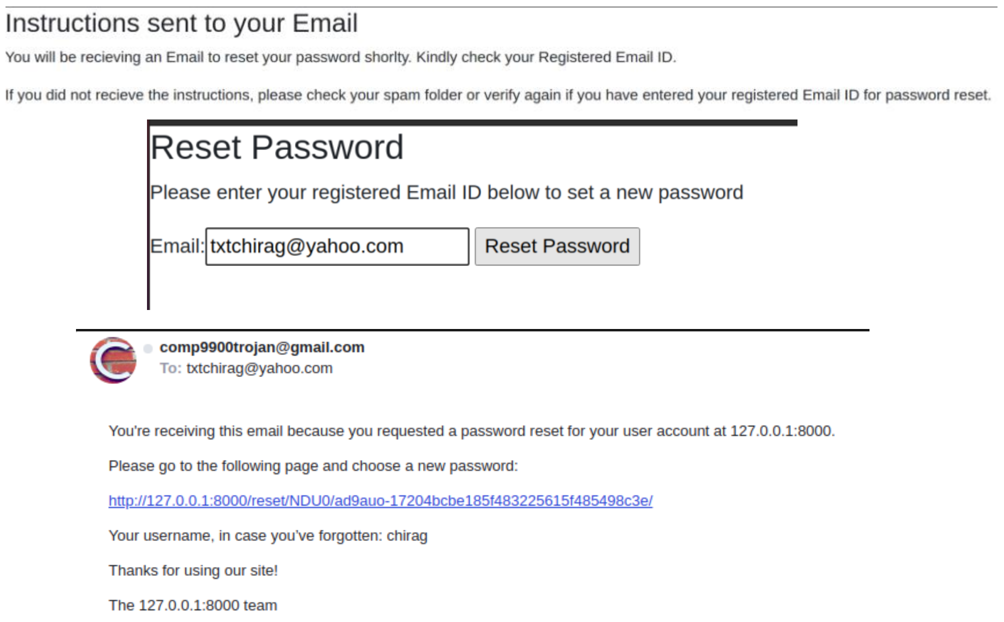
Additionally, a user has option to click on "Sign Up" link to do the registration which will take the user to register page. The user can also click on the "Reset password" link, if they forgot their password for their account and want to reset the password.


Don't have an account? Sign Up

Forgot password? Reset password

Fig. 10. Prompts for signing up or resetting password on login page

The reset password link redirects the user to another template page “reset_password”. Here the user will be asked to enter the registered Email ID into the form field. Once the email Id is entered click on the “Reset Password” button. This will redirect the user to “reset_password_sent” template page. The user must have received a mail from the website with instruction to reset the password.



Instructions sent to your Email

You will be recieving an Email to reset your password shorlty. Kindly check your Registered Email ID.

If you did not recieve the instructions, please check your spam folder or verify again if you have entered your registered Email ID for password reset.

Reset Password

Please enter your registered Email ID below to set a new password

Email: txtchirag@yahoo.com

comp9900trojan@gmail.com
To: txtchirag@yahoo.com

You're receiving this email because you requested a password reset for your user account at 127.0.0.1:8000.

Please go to the following page and choose a new password:

http://127.0.0.1:8000/reset/NDU0/ad9auo-17204bcbe185f483225615f485498c3e/

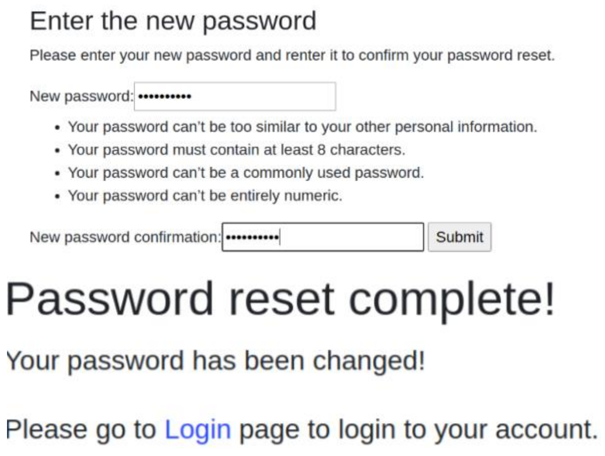
Your username, in case you've forgotten: chirag

Thanks for using our site!

The 127.0.0.1:8000 team

Fig. 11 a. Reset password functionality

If you click on the link provided you will be redirected to the page “set-password” where you can reset your password. It asks you to follow the requirements for the new password as per Django's default password form and once it is filled you can click on submit button. The next redirected page “reset_password_complete” will give message as “Password rest complete!” and will be asked to go to “Login” again.



Enter the new password

Please enter your new password and renter it to confirm your password reset.

New password: ••••••••••

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

New password confirmation: ••••••••••

Password reset complete!

Your password has been changed!

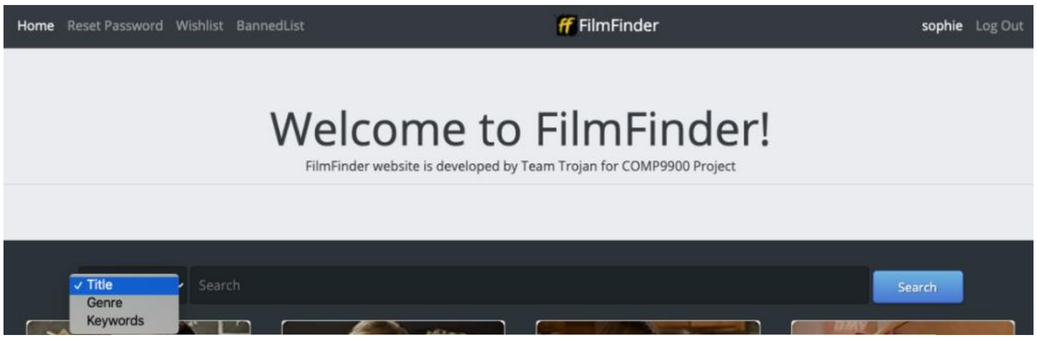
Please go to Login page to login to your account.

Fig. 11 b. Reset password functionality contd.,

For implementation of Mail feature, we used Django backend email feature. In settings.py we configured the email id and password for the application which in turn will be using smtp server configuration to send password reset mail to register users. The email account we used was comp9900trojan@gmail.com.
Setting up function for password reset was easily done thanks to Django's built-in password reset functionality, only addition was to setup appropriate template html to redirect the user on each step. The

functionalities like login, authenticate, and logout are satisfied by the built-in django.contrib.auth module. For login into the website the Django Forms satisfied the conditions check for username and password check.

1.3 CWT-5: As a user, I must be able to use the search box so that I can find the movie based on name, description or genre.



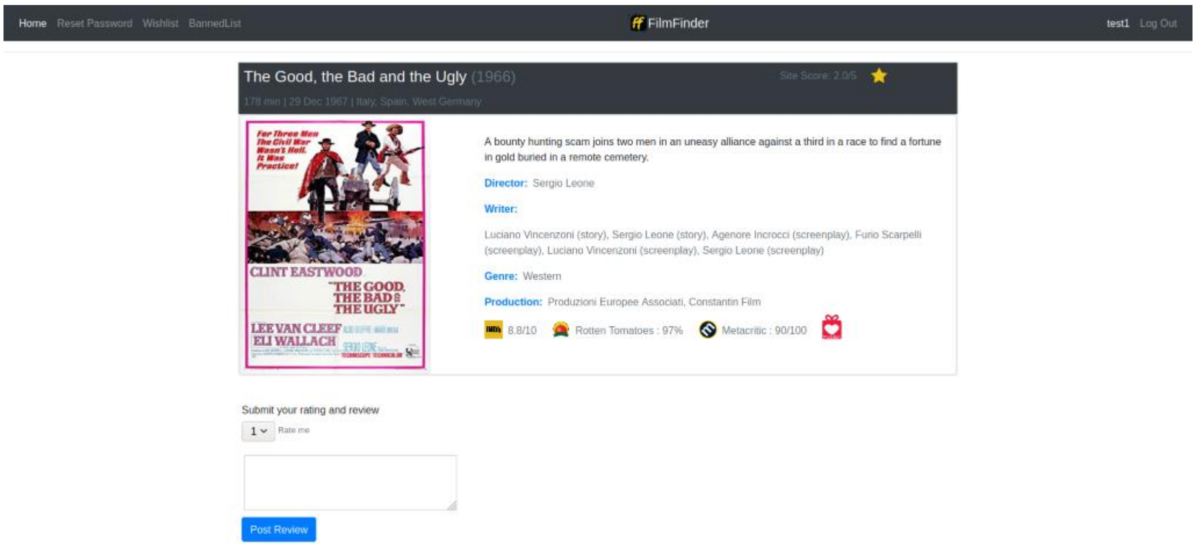
Welcome to FilmFinder!

FilmFinder website is developed by Team Trojan for COMP9900 Project

Fig. 12. Search box on the home page

In this user story, a user who is registered can search movie by tile or genre or keywords. This function is similar to ***CWT-30***, only that this is for a registered user.

1.4 CWT-21: As a user, I must be able to view the full details of the movie



The Good, the Bad and the Ugly (1966) Site Score: 2.0/5

178 min | 29 Dec 1967 | Italy, Spain, West Germany

A bounty hunting scam joins two men in an uneasy alliance against a third in a race to find a fortune in gold buried in a remote cemetery.

Director: Sergio Leone

Writer:

Luciano Vincenzoni (story), Sergio Leone (story), Agenore Incrocci (screenplay), Furio Scarpelli (screenplay), Luciano Vincenzoni (screenplay), Sergio Leone (screenplay)

Genre: Western

Production: Produzioni Europee Associati, Constantin Film

8.8/10 Rotten Tomatoes : 97% Metacritic : 90/100

Submit your rating and review

1 Rate me

Fig. 13. Full details movie page

This functionality describes the ability to open a webpage for a specific movie and view details regarding that movie. These include items such as:

- A movie poster
- Release Year
- Runtime
- Movie Name
- Director
- Actors

- Producer
- Ratings from our site and other sites
- Movie reviews from our site
- Like and dislikes for each review

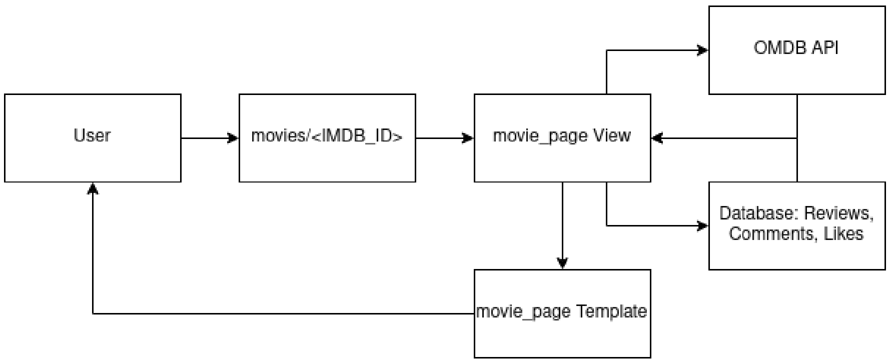


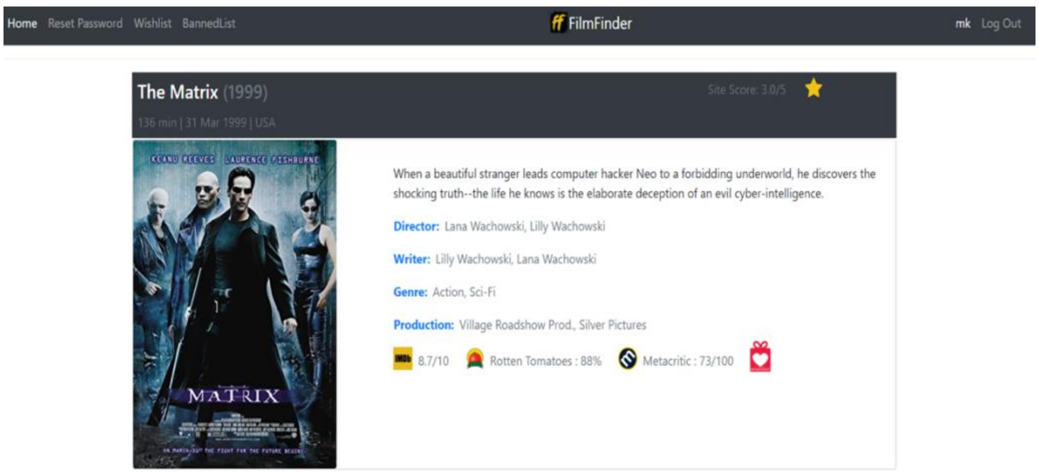
Fig. 14. Full details movie workflow

The webpage for a particular movie can be accessed through its IMDB (Internet Movie Database) ID, which is a unique ID assigned by the IMDB to each movie and used across a variety of APIs. Once the request has been received by the server, it calls the OMDB (Open Movie Database) API using the given ID, loads the JSON response, and passes that information to the corresponding template for movie webpages. Reviews, likes, and comments are also retrieved from our own database, but are discussed in their own sections.

One of the primary issues we had to address when developing our site was how to populate the page with information. While we considered a few alternatives, we ultimately opted to retrieve the data from the OMDB API. The OMDB API is a REST API that is entirely maintained by users. While it carries the advantage of being accessible at a very low cost, its licensing prevents any of its data from being used in a commercial manner, preventing us from monetizing the site in any form. Noting this, we consider it as a drop-in replacement for the IMDB API, which does provide licensing options for commercial use, but requires us to directly contact IMDB with a proposal, which is an arrangement that we do not believe is fit for this proof-of-concept project.

2. SPRINT-2:

2.1 CWT-7: As a user, I must be able to add movies into the wishlist



The Matrix (1999)

Site Score: 3.0/5

136 min | 31 Mar 1999 | USA

When a beautiful stranger leads computer hacker Neo to a forbidding underworld, he discovers the shocking truth--the life he knows is the elaborate deception of an evil cyber-intelligence.

Director: Lana Wachowski, Lilly Wachowski

Writer: Lilly Wachowski, Lana Wachowski

Genre: Action, Sci-Fi

Production: Village Roadshow Prod., Silver Pictures

IMDb 8.7/10 Rotten Tomatoes : 88% Metacritic : 73/100

Fig. 15. Wishlist option on the full details page (pink icon)

The user story provides the functionality of adding a movie to the wishlist page which allows the user to browse movies that the user is interested in. It maps to one of the project objectives, where the user must be able to add the movies that he comes across in the platform into a wishlist of their account. The wishlist page will consist of the titles of the movies added into it which is a hyperlink which when pressed on directs to the full detail's movie page of the specific movie.

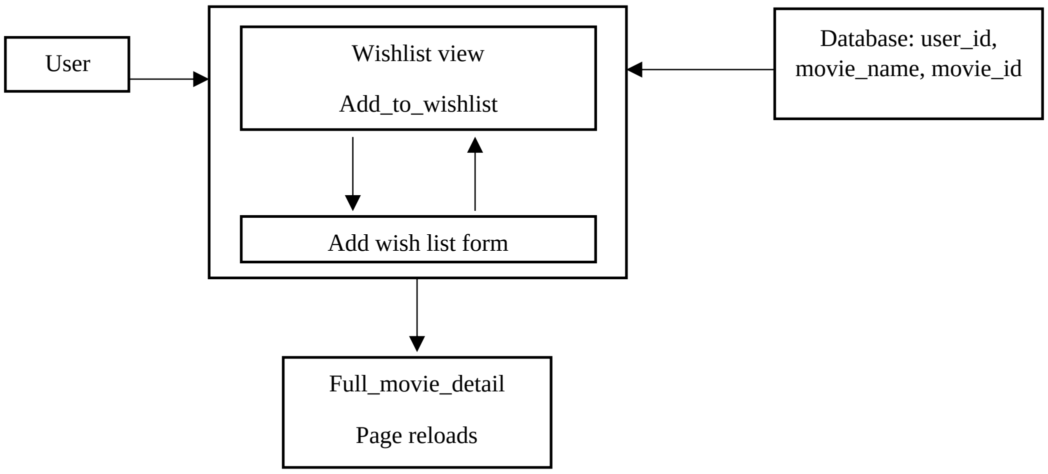


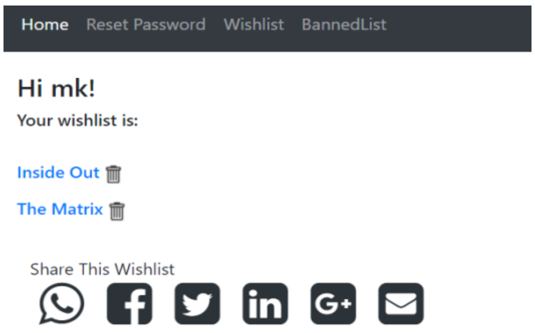
Fig. 16. Wishlist functionality workflow

This functionality is developed using the Model-View design pattern in a new Django application called 'wishlist' as follows:

- Model: The model which is a representation of the database consists of 'user_id' which is a foreign key from the base model which is created by Django, 'movie_id' which stores the IMDB id scraped from the API and 'movie_name' which will store the title of the movie name that is added into the wishlist.
- View: There is a view function called add_to_wishlist, which will accept the request and store the data into the database using the help of a form called 'AddWishlistForm' which will store the clean and store the movie name from the request. After a user adds movie into the wishlist, the page will reload itself which is the return action of this function.

The challenge of this user story is about inserting the data into the database, which will be used in future user stories. Also, the decision of viewing the wishlist immediately after adding into the wishlist was initially thought of but we decided to put that functionality into another user story.

2.2 CWT-8: As a user, I must be able to remove the movies from wishlist



Hi mk!

Your wishlist is:

Inside Out

The Matrix

Fig. 17. Bin icon representing delete option on wishlist page

This user story provides the functionality of removing the movies from the wishlist of the user. This is one of the project objects that the user must be able to add or remove movies from their wishlist. The user can remove movies from their wishlist at any time by clicking on the 'trash icon' which is the delete button.

This functionality is implemented using the Model-Template-View design provided by Django as follows:

- Template: It is the 'view_wishlist_page.html' page which receives the instructions from the view which alters the page to remove the items deleted.
- Model: The wishlist model which is stated in CWT-7. In this user_id associated with the items are getting deleted from the database,
- View: There is a view function called 'delete_wishlist', which will accept the request, movie_id and user_id. It will find the movie referenced with the user_id and delete it from the wishlist using the delete() function of Django. After that it will jump to the view_wishlist.

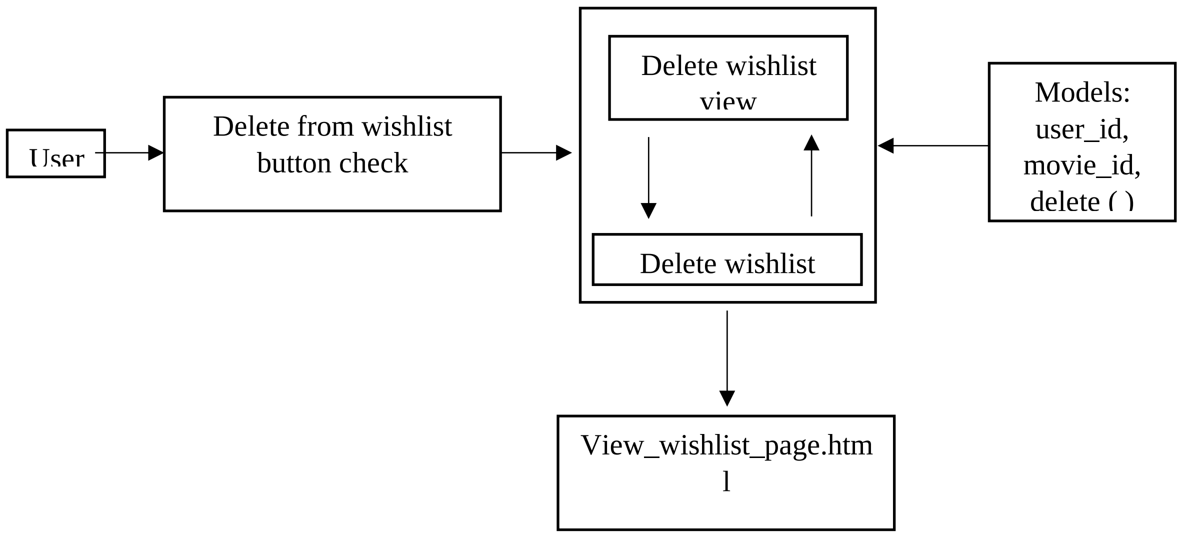

User
Delete from wishlist button check
Delete wishlist view
Delete wishlist
Models: user_id, movie_id, delete ( )
View_wishlist_page.html

Fig. 18. Delete option on wishlist page work flow

Initial challenges were that we were not familiar with dynamic web content about how to ask the database to delete the record from the database. This was solved using the forms and checking whether it is valid before fetching the values from the form using Django's built-in function.

2.3 CWT-14 & CWT-15: As a user, I must be able to write a review for a movie. As a user, I must be able to give a rating for a movie

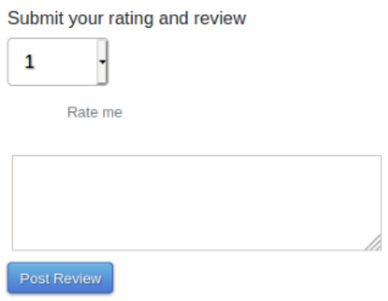


Fig. 19. Review and rating option on full movie detail page

A core feature of our site is the ability to post reviews of movies with ratings. These ratings serve as the backbone of the website, as they provide the primary means of user engagement and comprise a core component of our recommendation algorithm. Ratings and reviews consist of mandatory numerical ratings and text reviews.

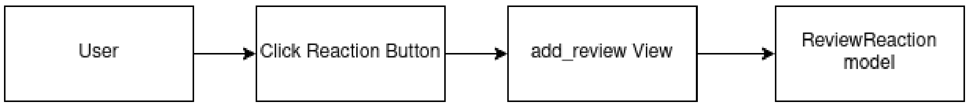
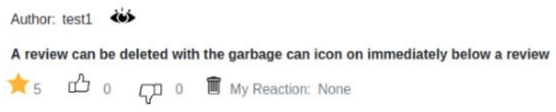


Fig. 20. Review and rating option work flow

Ratings and reviews are represented together as entries in a review table in our database. Users may submit only one review per movie, and reviews are indexed uniquely by movie IDs and user IDs. If a user has already submitted a review for a given movie, then submitting another review will overwrite the previous review. Submitting a review writes the review directly to the database and refreshes the page.

The design considerations regarding the review model revolve mainly around the additional features that are specific to one review. While it may be possible to store data such as reactions and comments within a JSON field tied to the review entry, we found that this does not adhere well with Django's development patterns and would leave a large amount of technical debt. We ultimately opted to use separate tables to manage any additional features.

2.4 CWT-16 & CWT-17: As a user, I must be able to remove the rating I have given for a movie. As a user, I must be able to remove any review that I gave to a movie


Author: test1

A review can be deleted with the garbage can icon on immediately below a review

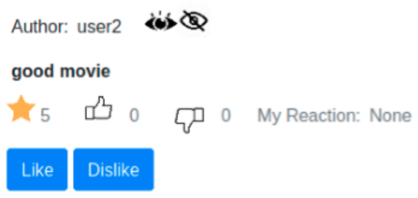
Fig. 21. Removing rating and review option on full movie detail page

Aside from posting reviews, users must be able to delete reviews and ratings that they have given a movie. This may be done for a variety of reasons but may be used when users feel that their past reviews no longer reflect their current opinions regarding a movie.

Deletion works similarly to writing new reviews. Users are given the option to delete a review only if they are the user that submitted that review. When a user clicks the delete button, the review is simply removed from the database.

The only major design consideration for managing reviews involved how comments should be handled. As it would not affect the site's core functionality, we opted to simply leave comments for reviews that have been deleted, as they will not be displayed if the review's ID is never called.

2.5 CWT-19: As a user, I must be able to like or dislike any review of other users


Author: user2

good movie

Fig. 22. Like and dislike button for a review in full movie detail page

We have included reaction buttons as a novelty feature in our website that aren't present in other existing alternatives. Reaction buttons serve as a good way for users to examine the quality of a review and reinforce the sentiment behind a given review. They also allow users to feel more engaged with the website's community as it grows and develops by offering an added means of interaction. Aside from this, reactions can also be used to analyse patterns amongst users and act as another form of feedback for the site owners to examine overall user behaviour.

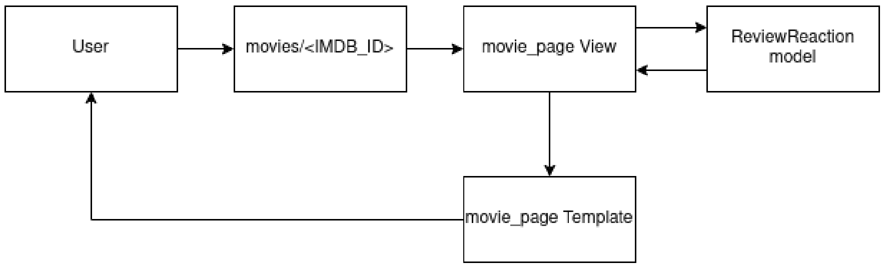


Fig. 23. User viewing movie page workflow

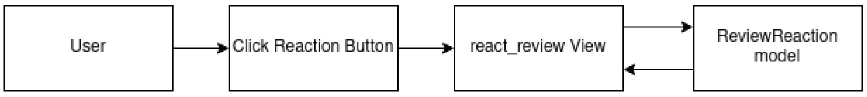


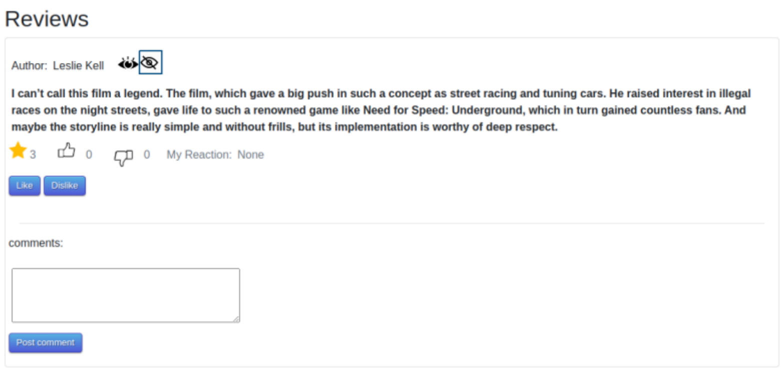
Fig. 24. User clicking reaction button workflow

Reactions are stored in a table indexed by user IDs and review IDs. The reactions themselves are stored in text format as "Like" or "Dislike", which leave the option for adding other reaction types in the future. When a user clicks on the reaction button, a new entry is generated in the table. Clicking the user's current reaction will delete that reaction, while clicking a reaction different from the current reaction will overwrite the previous reaction for that review.

A major design consideration involved how reactions would be stored. While we ultimately opted to store them in a table to align with Django's Model-View-Template architecture, this method will not scale well to large numbers of reactions. Because reactions are much more common than reviews or comments, and because each reaction is linked to only one review, it may be preferable to store them in a separate JSON field linked to each review instead. However, while this would be preferable as a long-term solution, it would add a large amount of technical debt to our implementation.

2.6 CWT-27: As a user, I must be able to add any review writers into my banned list

In this user story a logged in user will be able to add movie reviewer to their "BannedList". Once user is in a detailed page of a particular movie, they will be able to see reviews under it if any. If the user wishes to add a reviewer to their "BannedList" then they can click on the "striked eye" icon against the review.



Reviews

Author: Leslie Kell

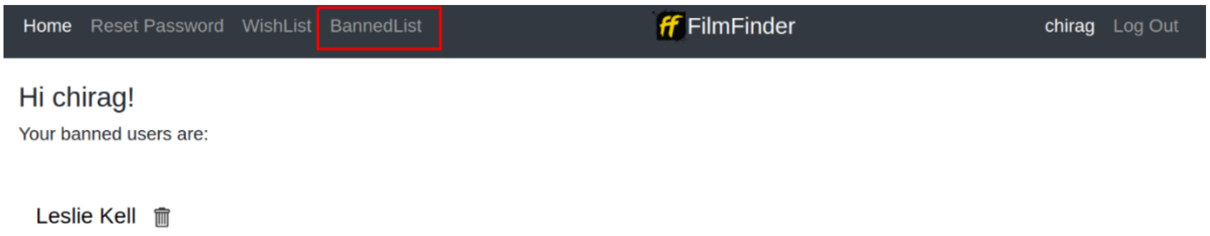
I can't call this film a legend. The film, which gave a big push in such a concept as street racing and tuning cars. He raised interest in illegal races on the night streets, gave life to such a renowned game like Need for Speed: Underground, which in turn gained countless fans. And maybe the storyline is really simple and without frills, but its implementation is worthy of deep respect.

3 0 0 My Reaction: None

comments:

Fig. 25. Adding reviewer to banned list

We can navigate to "BannedList" user on the Nav bar link and the user will be able to see the reviewers he has added to his "BannedList".



Hi chirag!

Your banned users are:

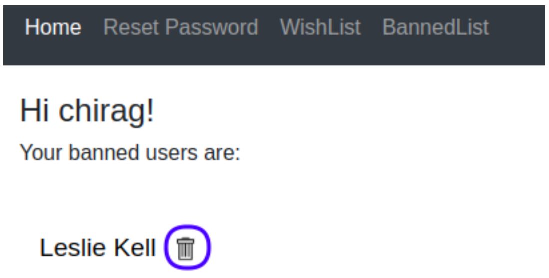
Leslie Kell

Fig. 26. Banned list page of a user "Chirag"

In order to set up a BannedList feature, the team created a Django app "BannedList" and created a Django model with user_id and banned_id to identify which user has banned which reviewer. Such unique occurrences will be added to database upon user interaction on the detailed page. The add feature was handled in the details.html which represent detailed info of the movie and enabled add to "BannedList" if the user is authenticated.

2.7 CWT-28: As a user, I must be able to remove any reviewers from the banned list

In this user story the logged in user can delete a previously added reviewer from their "BannedList". A user should navigate to their "BannedList" from Navbar and will be able to see their list of Banned users. Against each previously added reviewer there will be a "trash can" icon. The user can click the icon to remove that reviewer from their banned List. Once done They will be able to view the reviewers review and the rating provided for the films on site. The rating will be used up for the calculation of site score average and get updated for the corresponding movie's site score.



Hi chirag!

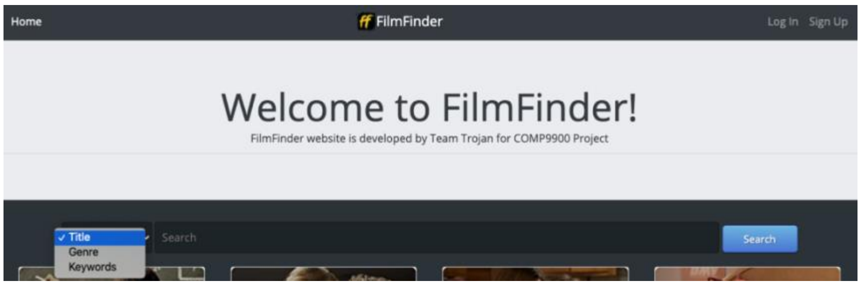
Your banned users are:

Leslie Kell

Fig. 27. Deleting user from banned list

The Delete functionality had to be implemented partially through front end user interaction and backend function, where the selected reviewer id is filtered for logged in user and particular object are removed. The filter is done using Django Query set on the database.

2.8 CWT-30: As a user, I must be able to search for the movie from the search box even if I don't have an account.



Welcome to FilmFinder!

FilmFinder website is developed by Team Trojan for COMP9900 Project

Fig. 28. Homepage view when the user is logged out

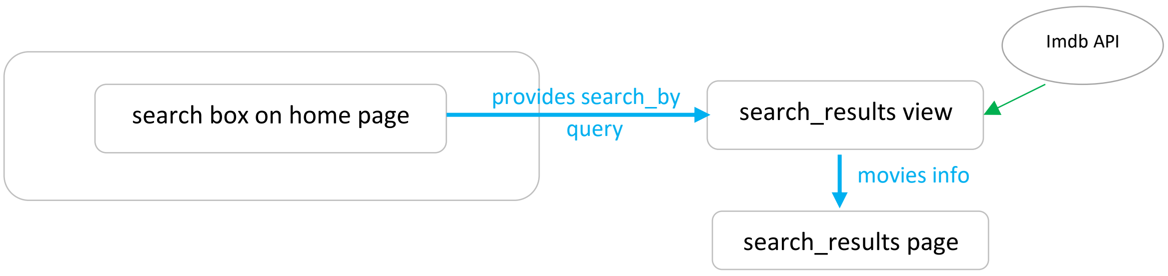
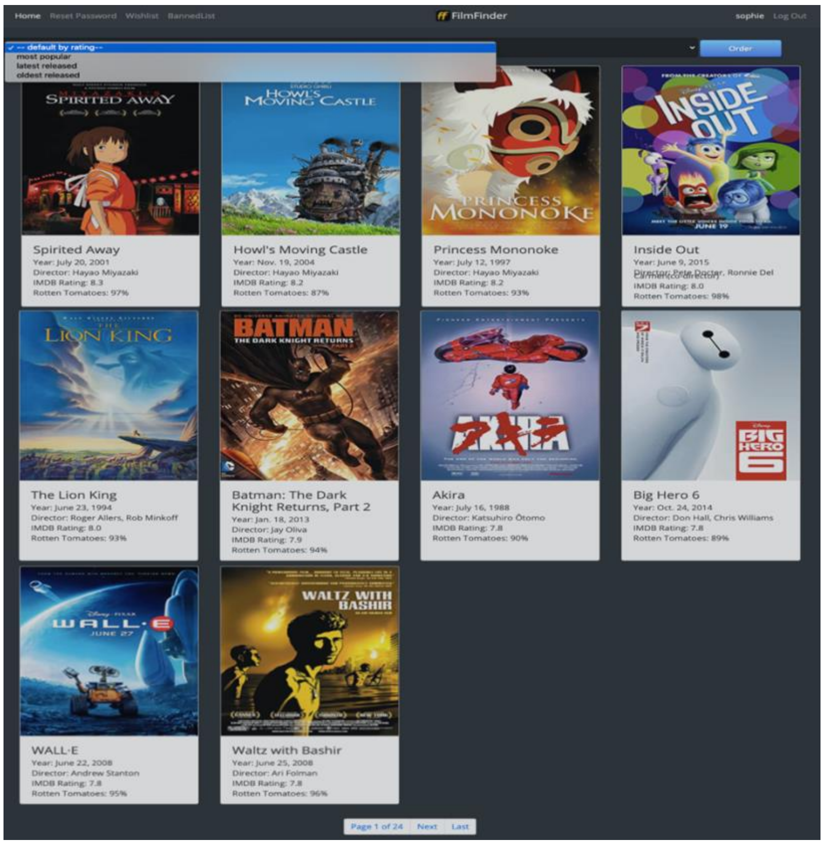


Fig. 29. Search button workflow

In this user story, a user who is not registered can still search movie by tile or genre or keywords. A form in home page will pass 'search_by' and query which contains search content to search_results view. The view will query the database based on 'search_by' and query content. After querying the database, imdbID will be obtained and passed to the imdb API so that we can have all needed information for display later on. All movies which match the search criteria are displayed in cards' form on the search_result page.

At this point, the team didn't notice that once thousands of movies' imdbID are obtained, API will be called thousands of times. Because of that reason, the webpage will be unreasonably slow. Then the team implemented pagination by 10 for searched movies in ***CWT-42***. Since there are only 10 movies per time needed to call the API, the search is faster.

2.9 CWT-42: Improve on the search engine from sprint -1 by adding add order function on search result page and use pagination to have a better user experience.



Spirited Away
Year: July 20, 2001
Director: Hayao Miyazaki
IMDB Rating: 8.3
Rotten Tomatoes: 97%

Howl's Moving Castle
Year: Nov. 19, 2004
Director: Hayao Miyazaki
IMDB Rating: 8.2
Rotten Tomatoes: 87%

Princess Mononoke
Year: July 12, 1997
Director: Hayao Miyazaki
IMDB Rating: 8.2
Rotten Tomatoes: 93%

Inside Out
Year: June 9, 2015
Director: Pete Docter, Ronnie Del Carmen(co-director)
IMDB Rating: 8.0
Rotten Tomatoes: 98%

The Lion King
Year: June 23, 1994
Director: Roger Allers, Rob Minkoff
IMDB Rating: 8.0
Rotten Tomatoes: 93%

Batman: The Dark Knight Returns, Part 2
Year: Jan. 18, 2013
Director: Jay Oliva
IMDB Rating: 7.9
Rotten Tomatoes: 94%

Akira
Year: July 16, 1988
Director: Katsuhiro Ôtomo
IMDB Rating: 7.8
Rotten Tomatoes: 90%

Big Hero 6
Year: Oct. 24, 2014
Director: Don Hall, Chris Williams
IMDB Rating: 7.8
Rotten Tomatoes: 89%

WALL·E
Year: June 22, 2008
Director: Andrew Stanton
IMDB Rating: 7.8
Rotten Tomatoes: 95%

Waltz with Bashir
Year: June 25, 2008
Director: Ari Folman
IMDB Rating: 7.8
Rotten Tomatoes: 96%

Fig. 30. Sorting options on search results

In this user story, a user should be able to order by rating or popularity or released date. In addition, the user should be able to get this result page fast and don't get all movies with infinite scrolling down.
A list option was created to give users options by what to order. The pagination was implemented so that only 10 movies are on one page. On the bottom of the page, 'next' and 'previous' were given to link users to next or previous search result page.

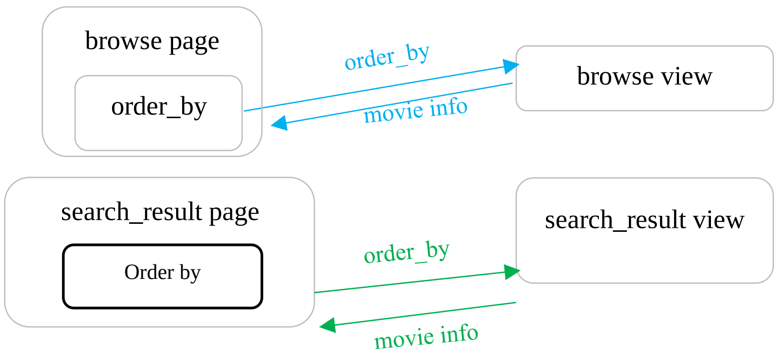
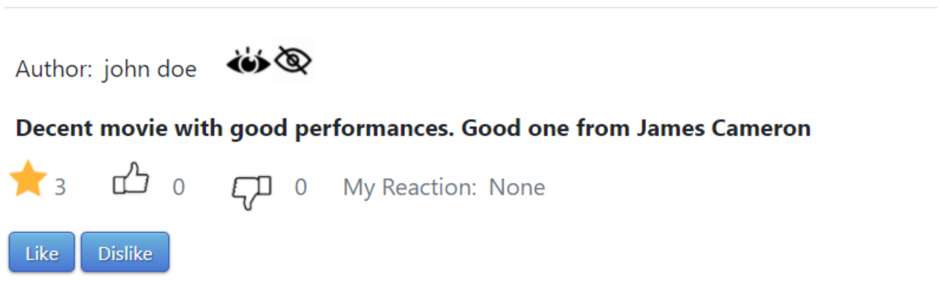


Fig. 31. Sorting options workflow

In this part, the team did not have too much difficulty since the team was more familiar with Django and other related knowledge.

3. SPRINT-3:

3.1 CWT-9: As a user, I must be able to view the wishlist of another user who is a reviewer

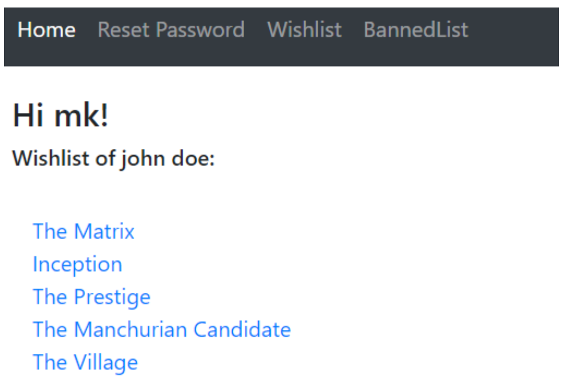


Author: john doe

Decent movie with good performances. Good one from James Cameron

3 0 0 My Reaction: None

Fig. 32. Button to view reviewers wishlist called 'johndoe' in full movie detail page (eye)



Hi mk!

Wishlist of john doe:

The Matrix
Inception
The Prestige
The Manchurian Candidate
The Village

Fig. 33. John doe's wishlist after clicking the button

This functionality is also one of the project objectives which is to help the user of one account access the wishlist of another user who is a reviewer. So, whenever a user is currently is viewing the full details movie page of a movie, he can access the wishlist of the users who reviewed that movie irrespective of the current user being logged in or not.

This is implemented from the templates in 'review.html' where a link to the wishlist of the user is triggered when the current user clicks on that button which in turn calls the view_wishlist() function in views.py of wishlist application.

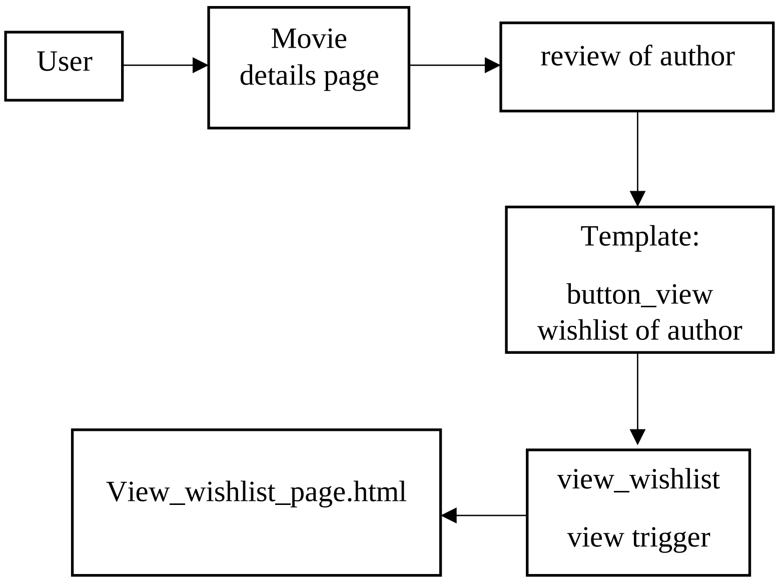
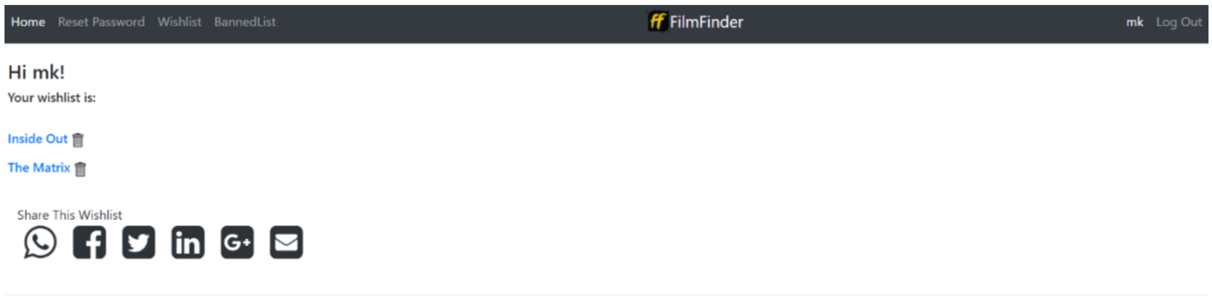

User
Movie details page
review of author
Template: button_view wishlist of author
view_wishlist view trigger
View_wishlist_page.html

Fig. 34. View reviewers wishlist workflow

The challenge while implementing this was coming up with the logic of viewing others wishlist. Initially after implementing the functionality there was a bug recognised when a user who is not the owner of that wishlist was able to delete the movie from that list. This was rectified by checking the user authentication with the list.

3.2 CWT-10: As a user, I must be able to view my wishlist of movies



Hi mk!

Your wishlist is:

Inside Out

The Matrix

Fig. 35. User 'mk' wishlist view from the wishlist option on the navigation bar

This functionality aides to the project objective of viewing the wishlist of the current user after he adds or removes movies from it. The user can view his wishlist from the homepage of the website in the navigation bar. When the button is clicked the user is directed to a new page where the titles of the movies which the user added into his wishlist are displayed in the order in which they are added.

This is also implemented using the views and templates structure of Django as follows:

- Template: This implements the front-end part of displaying the wishlist items created by fetching the final data from views.py which would be a dictionary consisting of ‘user_id’ for authenticating the user and titles of the movies for displaying.
- Views: There is a function called ‘view_wishlist’ in the wishlist application which gets the request and filters out the user_id and wishlist object using the user_id and sends that data to the template for displaying the page.

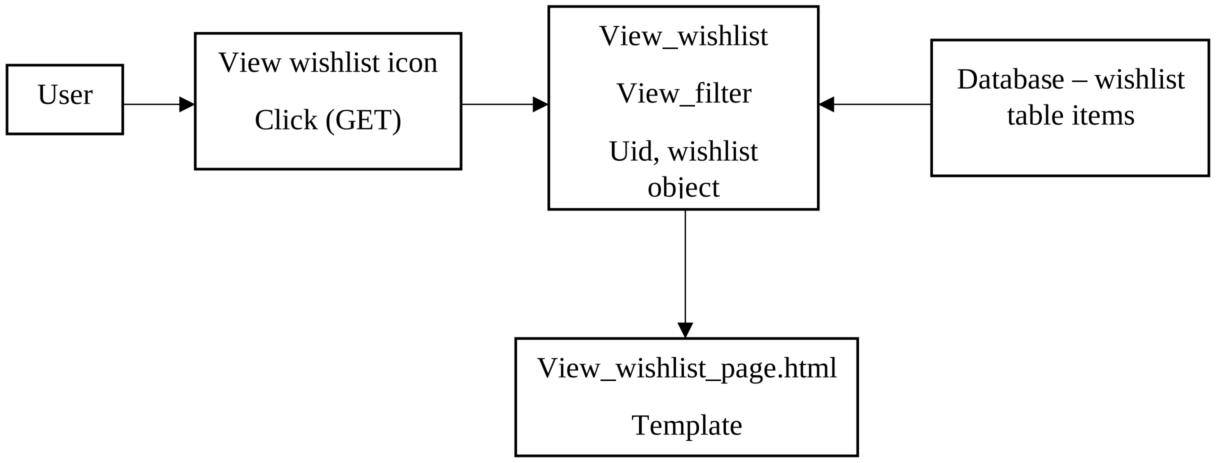


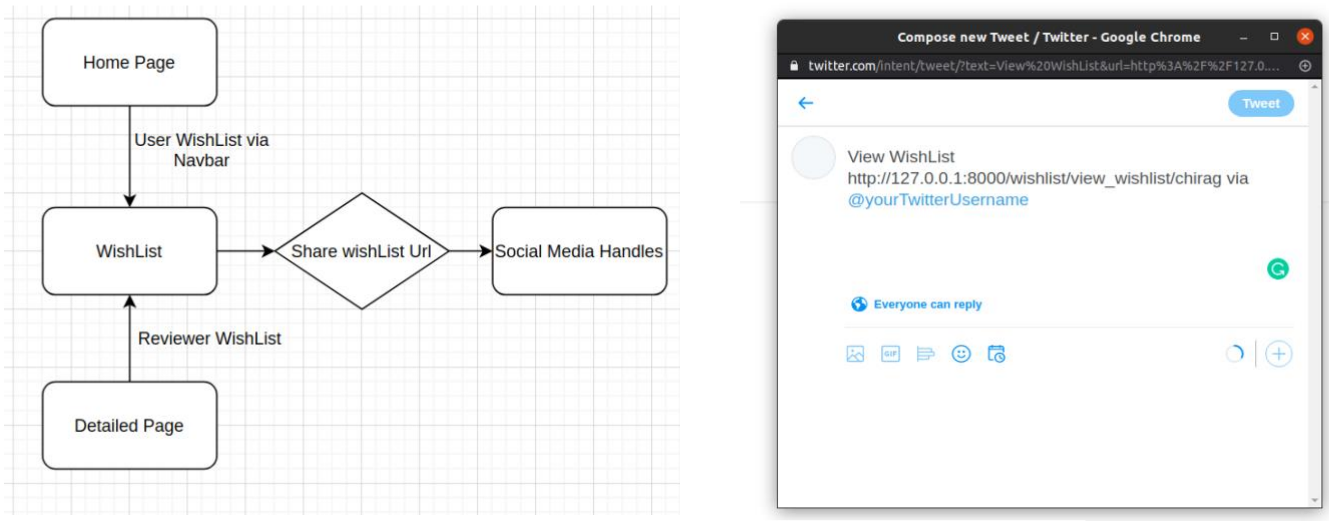
Fig. 36. View wishlist option workflow

This was one of the easy functionalities to implement after figuring out the flow of the website and linking all the templates to the right functions in the view.

3.3 CWT-11: As a user, I must be able to share my collection with other users or in social media

In this user story the user can share their wishlist to a social media site if they need to. The user can navigate to “WishList” page and in the bottom the user will have multiple social media icons for Whatsapp, Facebook, twitter, LinkedIn, google plus, and mail respectively. Once clicked the user is redirected to particular social media page to post or share the Wishlist page URL eg: “http://127.0.0.1:8000/wishlist/view_wishlist/chirag”

Share This Wishlist

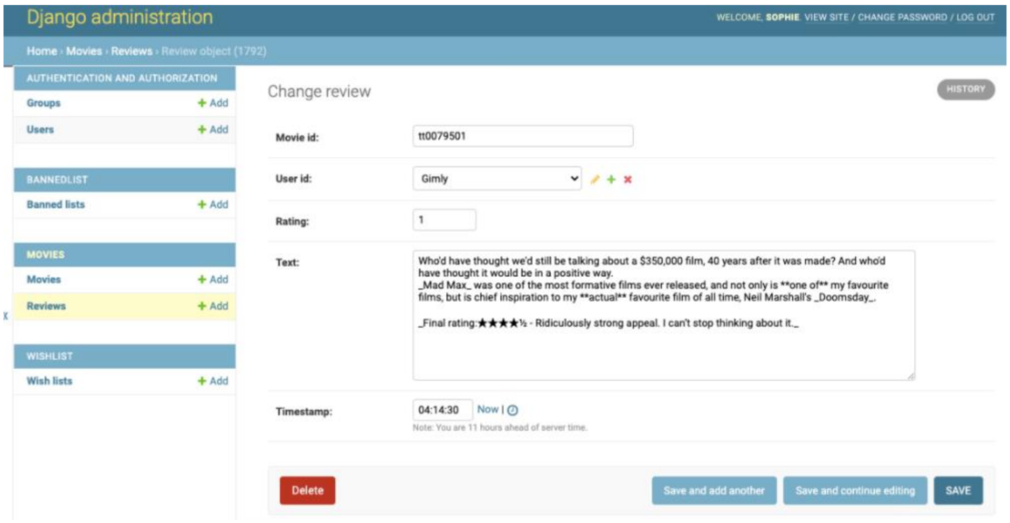


View WishList
http://127.0.0.1:8000/wishlist/view_wishlist/chirag via @yourTwitterUsername

Fig. 37. Sharing the wishlist on Twitter and workflow of the functionality

The functionality is handled entirely by a JavaScript at the front end. Which process the URL and parse it over to each of the APIs for the social media sites. Here the data_url field is used to share the WishList page URL to other social media handles. The JavaScript snippet is Sharer.js is maintained by ellisonleao.

3.4 CWT-18: As an admin of the website, I should be able to remove any offensive reviews from the review list for any movie.



Movie id: tt0079501

User id: Gimly

Rating: 1

Text: Who'd have thought we'd still be talking about a $350,000 film, 40 years after it was made? And who'd have thought it would be in a positive way.
Mad Max was one of the most formative films ever released, and not only is **one of** my favourite films, but is chief inspiration to my **actual** favourite film of all time, Neil Marshall's _Doomsday_.

Final rating:★★★★½ - Ridiculously strong appeal. I can't stop thinking about it.

Timestamp: 04:14:30 Now |
Note: You are 11 hours ahead of server time.

Fig. 38. Admin functionality to remove offensive review

In this user story, an admin can delete any review for any movie. Django administration is used in record all reviews for any movie. And review information can help recommendation system to predict users' possible like movies in the later function.

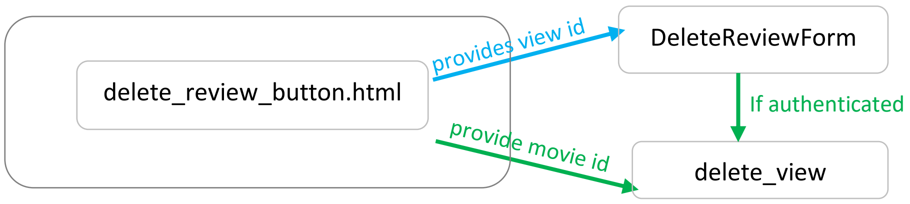
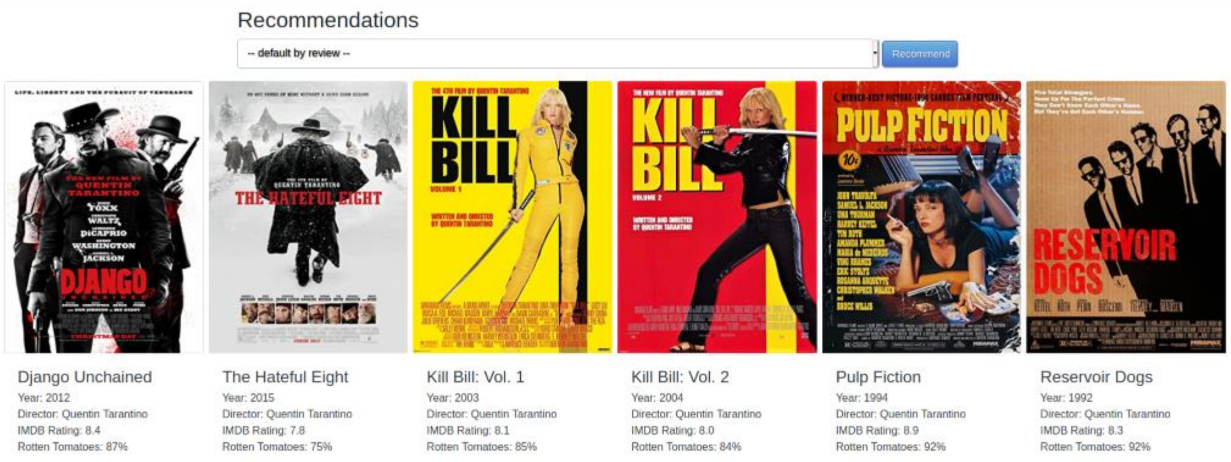


Fig. 39. Removing offensive reviews workflow

At this stage, the team was not familiar about how to delete content in the sqlite3.
The problem then was split into: first, the movie and review information need to be passed; second, reviews need to be updated. The team created a button within a form which contains a hidden input value: review_id and also passes movie_id. When a user clicks the button, review_id and movie_id will be passed to delete_view in movie/views.py.
The view will check and delete the review for the movie.
To avoid anyone deleting any review, only an admin has the interface to delete any review of any movie.

3.5 CWT-24: As a user, I must be able to view movies which are similar to the review history of a movie



Django Unchained
Year: 2012
Director: Quentin Tarantino
IMDB Rating: 8.4
Rotten Tomatoes: 87%

The Hateful Eight
Year: 2015
Director: Quentin Tarantino
IMDB Rating: 7.8
Rotten Tomatoes: 75%

Kill Bill: Vol. 1
Year: 2003
Director: Quentin Tarantino
IMDB Rating: 8.1
Rotten Tomatoes: 85%

Kill Bill: Vol. 2
Year: 2004
Director: Quentin Tarantino
IMDB Rating: 8.0
Rotten Tomatoes: 84%

Pulp Fiction
Year: 1994
Director: Quentin Tarantino
IMDB Rating: 8.9
Rotten Tomatoes: 92%

Reservoir Dogs
Year: 1992
Director: Quentin Tarantino
IMDB Rating: 8.3
Rotten Tomatoes: 92%

Fig. 40. Default recommendations based on review on full movie details page

In this feature, users must be able to view a list of recommendations for each movie from the details page of any movie. This also forms one of the core features of our site. In our implementation, we make full use of the objective features stored in our database for each movie.

The recommendations system we developed makes use of the bag-of-words approach, heavily utilizing the sklearn libraries to both develop our models and speed up our computations. Initially, we retrieve the data we have stored for our entire database of movies. Our database stores information for movie titles, directors, genres, keywords, and synopses. A separate vectorizer is trained on each field, such that given a string, a vector is returned containing the number of occurrences of each word. For each movie in our database, we then concatenate the vectorized representation of each field for each movie

Following our calculation of each movie's feature vector, we then train a ball tree on this data. A ball tree is a data structure that represents data as a tree of hyperspheres denoting the distance to each data point. We chose to utilize this data structure to minimize the number of comparisons performed to calculate the nearest neighbours of each movie. Utilizing a ball tree over brute force effectively reduces the complexity of our search from O(n^2) to O(logn), but it does incur a substantial storage cost of approximately 130MB for each model trained.

Once the vectorizers and ball tree have been trained on our data, they are then stored as pickle files for quick retrieval. While we considered storing these in the database directly, we observed empirically that writing directly to the file system resulted in faster load times.

To generate a list of recommendations given a particular movie, we load the entire model into memory, and run the vectorizers on the current movie. The ball tree is then queried for the 30 nearest neighbours.

Once we have obtained a list of recommendations, we then run them through our custom ranking system. When developing our algorithm, we came up with the following objectives:

- The ranking must incorporate ratings from our website
- A popular, highly rated movie must be ranked above highly rated movies with few reviews
- Highly rated movies must always be ranked above poorly rated movies
- Poorly rated movies with many reviews must be ranked below poorly rated movies with few reviews

- Suggestions must generally tend towards those that are more closely related to the current movie

Given these objectives, we then developed the following formulae:

- $vote\ factor = 2 * (sigmoid(average\ internal\ and\ external\ movie\ ratings) - 0.5)$
- $popularity\ factor = \frac{1}{3}\frac{\log(external\ count)}{\max(\log(external\ count))} + \frac{1}{3}\frac{popularity}{\max(popularity)} + \frac{1}{3}\frac{internal\ count}{\max(internal\ count)}$
- $distance\ factor = \exp\left(1 - \frac{distance}{\max(distance)}\right)$
- $ranking = vote\ factor * popularity\ factor * distance\ factor$

We note that the vote factor represents a number ranging from –1 to 1, depending on the average score of that movie. The popularity factor then scales that value, placing emphasis on movies that are popular internally, then on those with a high popularity metric on IMDB. Finally, the distance factor scales these values from 1 to e, which provides an overall greater contribution than the popularity factor.

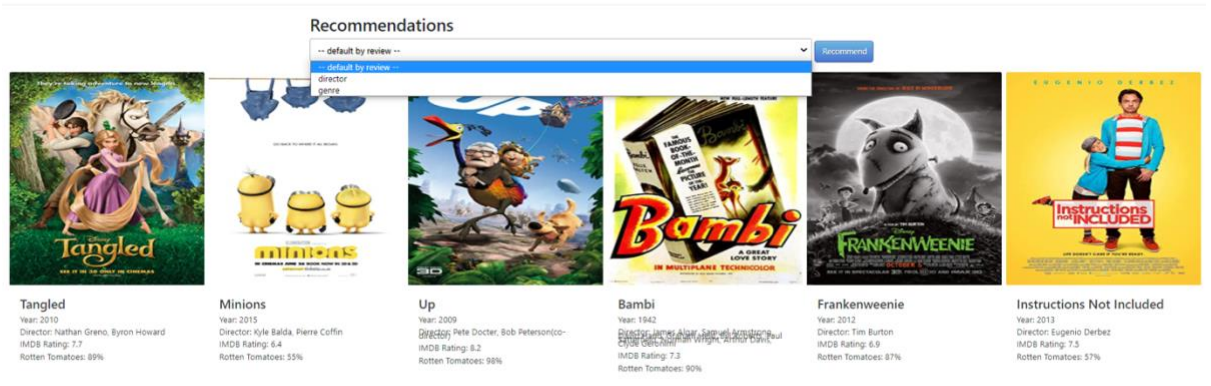
Following these calculations, the movies are then sorted by ranking in descending order of the product of the three factors. These are then displayed on the website.

Initially, our team had intended to use collaborative filtering to develop our recommendation system. However, while we do acknowledge its strengths, we note that it has several drawbacks compared to the approach which we ultimately decided to use. These include:

- Larger numbers of parameters, due to the necessity of storing parameters for each user
- Requires a larger number of reviews and samples to develop a coherent model. This is infeasible for a newly launched website, and is geared more towards those with larger userbases
- Does not fully utilize the features we have at our disposal. Collaborative filtering is particularly useful in cases in which we do not have access to objective features for each movie. However, the APIs from which we source our data readily provide these
- Requires frequent retraining, particularly when new movies are added. The model does not scale particularly well to growth in both the userbase and the number of movies.

Given these drawbacks, we ultimately decided to develop our own custom bag-of-words, nearest-neighbours, ranking pipeline.

3.6 CWT-25: As a user, I must be able to view recommendations by the attribute of the movie that I'm currently viewing



Tangled
Year: 2010
Director: Nathan Greno, Byron Howard
IMDB Rating: 7.7
Rotten Tomatoes: 89%

Minions
Year: 2015
Director: Kyle Balda, Pierre Coffin
IMDB Rating: 6.4
Rotten Tomatoes: 55%

Up
Year: 2009
Director: Pete Docter, Bob Peterson(co-director)
IMDB Rating: 8.2
Rotten Tomatoes: 98%

Bambi
Year: 1942
Director: James Algar, Samuel Armstrong, [illegible] Norman Wright, Arthur Davis, Clyde Geronimi
IMDB Rating: 7.3
Rotten Tomatoes: 90%

Frankenweenie
Year: 2012
Director: Tim Burton
IMDB Rating: 6.9
Rotten Tomatoes: 87%

Instructions Not Included
Year: 2013
Director: Eugenio Derbez
IMDB Rating: 7.5
Rotten Tomatoes: 57%

Fig. 41. Attributes the user can select to get recommendations on full movie detail page

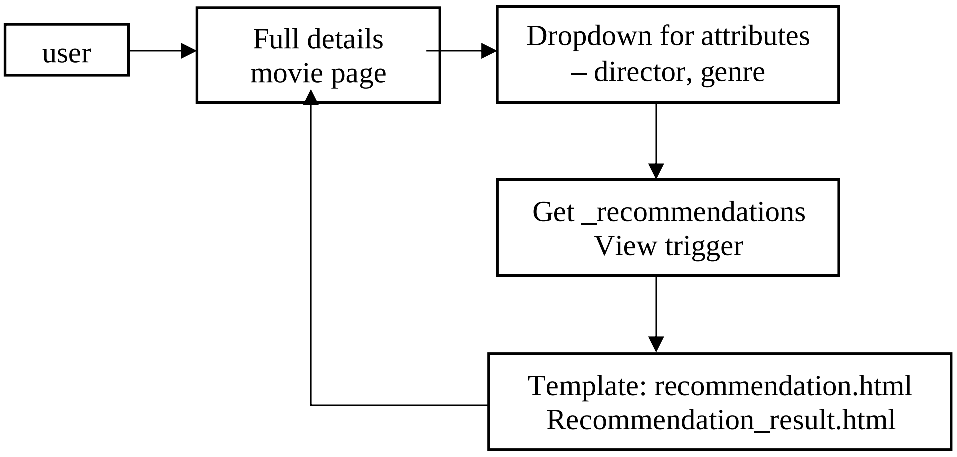

user
Full details movie page
Dropdown for attributes – director, genre
Get _recommendations View trigger
Template: recommendation.html Recommendation_result.html

Fig. 42. Recommendations by attribute workflow

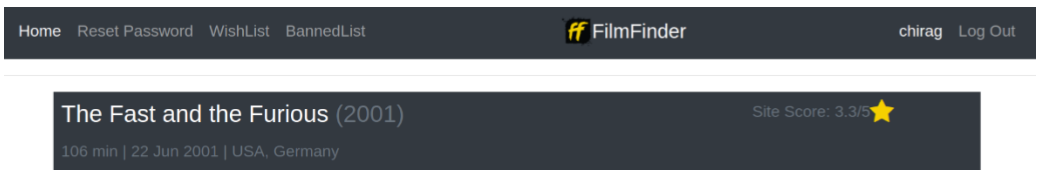
This is one of the main project objectives to provide the user an option to get recommendations based on the attribute he selects. The attributes the teams have design the recommendation system are genre and director apart from the default review recommendation.

The logic of the recommendation system is similar to the previous functionality of building a recommendation system keeping in view of reviews but additionally the 'filter' in 'get_recommendations' function in 'utils.py' of the recommendation application changes as the user selects the attribute on which he would like to be recommended on. A list is returned by the function which consists of 6 best ranked movies which is sent to the 'movie_page' in views of movies application which is then directed to 'recommendations.html' which uses the imdb id of the movie to fetch the poster, title, rating, year and director of the movies to display on the full movie details page itself.

The most challenging part was to figure out the recommendation system as mentioned in the above section and how to return values to the page. There were many debates on how to display the information- either on a new page or on the movie details page, how many items to display, how to give user the option to select, should we display all the attributes recommendations without using the dropdown option and many more. But the team finally felt that it was efficient to implement the recommendation on the movie page with dropdown button to select the attributes as shown above.

3.7 CWT-29: As a developer, I shouldn't let the average rating be influenced by the users in the banned list

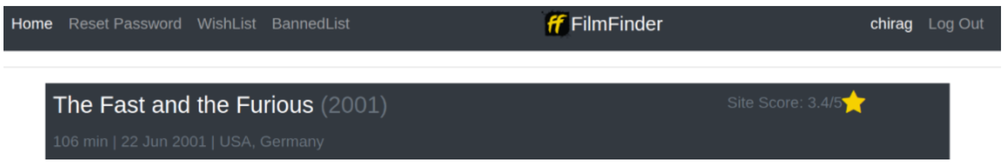
In this user story the Site score and Review list in detailed page will be updated as per the logged in users Banned List. If a User is added to Banned List i.e. by clicking the "striked eye" icon, the page refreshes with exclusion of that reviewer and it will also change the site score accordingly. This functionality enables a logged in user not to get influenced by banned list reviewers review and their ratings on the movies. Here is the site score for movie "The Fast and Furious" before and after adding the reviewer "Leslie Kell" to banned list. And you can view the rest of the reviews in the page including the logged in user.



The Fast and the Furious (2001)

Site Score: 3.3/5

106 min | 22 Jun 2001 | USA, Germany

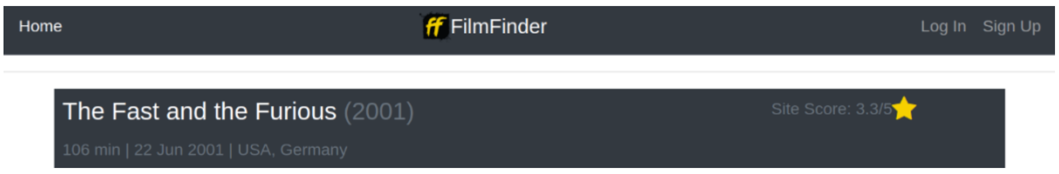

Home Reset Password WishList BannedList ff FilmFinder chirag Log Out

The Fast and the Furious (2001) Site Score: 3.4/5

106 min | 22 Jun 2001 | USA, Germany

Fig. 43. Site score after user 'chirag' added a reviewer into his banned list

This feature only excludes the reviews for the logged in user and if a user is not logged in or a different user is logged in to the website the earlier added banned users review "Leslie Kell" will still be visible and the site score will be same as before.


Home ff FilmFinder Log In Sign Up

The Fast and the Furious (2001) Site Score: 3.3/5

106 min | 22 Jun 2001 | USA, Germany

Fig. 46. Site score of the same movie when viewed from a user logged out – remains unaffected

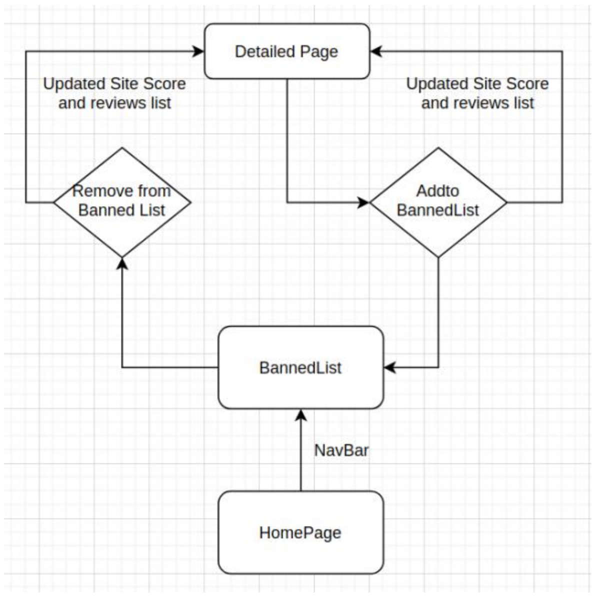
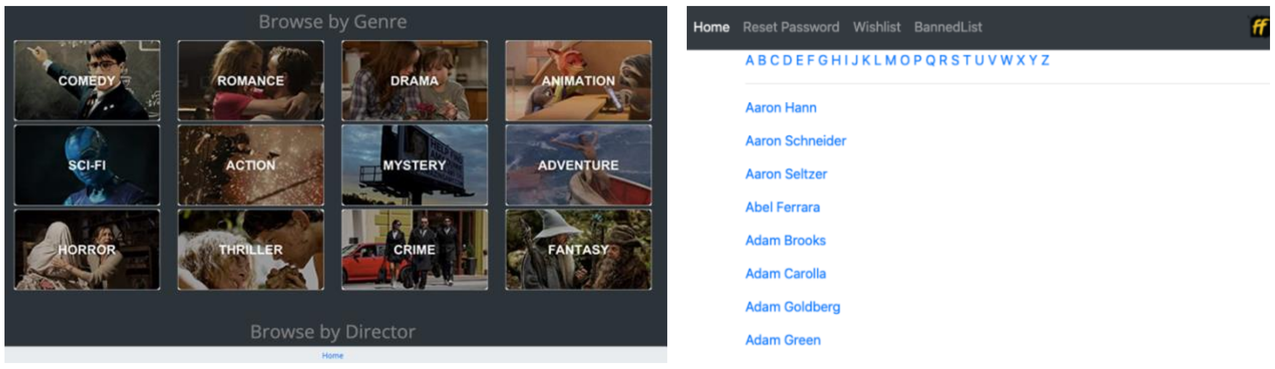


Fig. 44. Workflow of unaffecting site score

The challenges in setting up BannedList were to have check on whether the user is logged in or not and then if the reviewer is added/removed to or from the banned list the site score and review must get updated accordingly. The functional aspects were done partially on the front end details.html page template and on the back end views.py under BannedList app. For creating an object under BannedList we made use of Django's Forms and for viewing or deleting the object was done using the Django's query set and filter by logged in user

3.8 CWT-46: As a user, I want to be able to browse by genre on home page.



Aaron Hann
Aaron Schneider
Aaron Seltzer
Abel Ferrara
Adam Brooks
Adam Carolla
Adam Goldberg
Adam Green

Fig. 45. View of home page browsing by genre and director

In this user story, a user should be able to browse by genre or by director on home page.

The page has different 'cards' for different genre. For each genre, a link is created to pages that show all the specific genre movies in cards form with 10 movies per page.

On the bottom of the home page, a link for 'Browse by Director' is created which will leads to new page. On that page, there is an alphabet index on top and all directors whose name starts with same letter will be classified on one page.

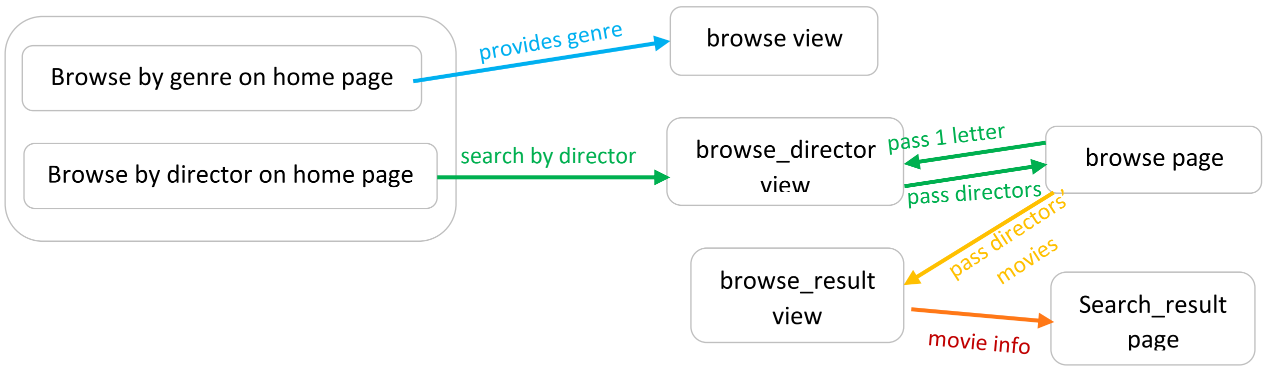


Fig. 46. Workflow of the browsing

Directors are stored for movies in the movie form, at this stage, the team found it was difficult to get directors. Because it is always a movie object that the team got. Then the team used query filter to get all movies with that director and use 'values_list' by 'Director' with 'flat' to get directors. Finally, 'distinct' was included to remove duplicate names.

Engineering Practices

SCRUM which is one method of AGILE is used to manage the project. The team implemented the project over three sprints where each one lasted about two to three weeks.

JIRA was used to record all user stories and related description as well as accepting criteria. In addition, JIRA was used to assign tasks to team members and keep track of members' progress.

Daily scrum meeting allowed every member know well about everyone’s progress so that project efficiency was guaranteed.

There was always a retrospective meeting after every sprint. The retrospective meeting allowed the team focus on what the target was. In addition, the retrospective meeting helped every member knew well about what have done, what could be improved and how we could improve it.

User Document / Manual

Environment setup

Setting up the environment, installing dependency modules and launching the launching the server can be done by the below steps

Method one

To run the server please run the below shell command

./setup.sh

Then launch website on a web browser by hitting the following URL
http://127.0.0.1:8000/

Method two

Alternatively, we can setup the website by following the below steps on a Linux machine

1. Save the project root and go to home directory by running the following

PROJECT_ROOT="$(pwd)"

cd ~

2. Make a new directory for the virtual environments

mkdir -p venv

3. Using python3 venv module setup a virtual environment for the project FilmFinder

python3 -m venv venv/FilmFinder-env

4. Activate the created environment

source ~/venv/FilmFinder-env/bin/activate

5. Change directory to Project root

cd $PROJECT_ROOT

6. The project dependency requirements are to be installed in the environment using pip

Following modules and their corresponding versions are required to run the webserver.

asgiref==3.3.1

click==7.1.2

Django==3.1.3

```
joblib==0.17.0

nltk==3.5

numpy==1.19.4

pandas==1.1.4

pkg-resources==0.0.0

python-dateutil==2.8.1

pytz==2020.4

regex==2020.10.28

scikit-learn==0.23.2

scipy==1.5.4

six==1.15.0

sklearn==0.0

sqlparse==0.4.1

threadpoolctl==2.1.0

tqdm==4.51.0
```

We can install them by running the below pip install command and passing the requirements file

pip3 install -r requirements.txt

7. Install NLTK stopwords for the recommendation utility

python3 -m nltk.downloader stopwords

8. Change directory to Django Project folder

cd FilmFinder-project

9. Run initial manage commands to make migrations and migrate for setting up the database.

python3 manage.py makemigrations
python3 manage.py migrate

10. Run below command to start up the server in local host port 8000

python3 manage.py runserver

11. Launch website on a web browser by hitting the following URL

http://127.0.0.1:8000/

Project Backend and database

The webserver uses Django backend which in turn integrates with an inbuilt SQLite database server to store the data. Django's built in backend modules create and update schemas and insert data as the webserver is used.

In addition to Webserver usage public data has been sourced to import movie data of 5000 popular movies from TMDB database. We have also used certain scraping of the TMDb API to retrieve data for corresponding data on reviewers and users who reviewed them. All the data are imported into the project database

./FilmFinder-project/db.sqlite3
We can find the resource files in csv format in the following directory

support_files/

tmdb_5000_credits.csv and ***tmdb_5000_movies.csv*** are the original data sourced from Kaggle ***https://www.kaggle.com/tmdb/tmdb-movie-metadatareign*** key dependency for use This dataset was generated from The Movie Database API. This product uses the TMDb API but is not endorsed or certified by TMDb. Their API also provides access to data on many additional movies, actors and actresses, crew members, and TV shows. We have additionally modified the data as per our use case and stored information is the following file ***movies2.csv***. For review data we separately scraped the reviews from the movies using TMDb API and the reviews are stored in ***tmdbReviews.csv*** same has been modified and updated to the database as ***movies_review.csv***. The reviewer accounts have also been additionally populated as the model in the webserver has for users. This is updated into ***auth_user.csv***

Additional dependency used for website functionalities

Generated trained models are stored in following directory. This will be trained and used upon initial invocation of the recommender utility.

./FilmFinder-project/recommender/pickle

For search engine and search result to populate data on detailed page, an API is called to retrieve the respective data from OMDB API API key is configured into the project for its execution. same can be found in support_files directory. The OMDb API is a RESTful web service to obtain movie information, all content and images on the site are contributed and maintained by our users.

apikey=d21f1d0&

The front-end design uses Bootstrap 4 for style cascade sheets and majority of the code snippets has been sourced from public cdn of the bootstrap. This is included in templates ***base.html*** file.

```
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/css/bootstrap.min.css"
integrity="sha384-
TX8t27EcRE3e/ihU7zmQxVncDAy5uIKz4rEkgIXeMed4M0jlfIDPvg6uqKI2xXr2"
crossorigin="anonymous">
```

Conclusion and Future Work

Given all of our previous discussion, we believe that the project in its current state fulfils all the objectives outlined in our initial proposal. This includes all of the base required features, as well as novel features in the form of a reaction system and a comment system. Furthermore, in its current state, the website should serve as a good proof-of-concept which is capable of handling a small userbase. We note that there are many areas of improvement that future maintainers can choose to undertake, in particular to support a growing community and future improvements in computational resources. We list these improvements, dividing them into those geared towards making improvements to the user experience, the website's scalability, and the quality of the recommendation system.

User Experience

A number of features that are present in most modern websites can be used to enhance the overall user experience of our website. Improvements that can be made to the website's aesthetics include:

- Direct manipulation of the DOM instead of using page reloads, particularly when interacting with the review page
- A persistent header in between page loads
- Infinite scrolling for the search page
- Improvements to element rescaling to better adapt to smaller screens and mobile devices
- A "last edited" timestamp, and the ability to view previous versions of a review

In addition, we also include a list of features that may be included in the future. These features may often be observed in other social media websites, such as Twitter or Reddit. In general, these are designed to provide added opportunities for users to interact with the site.

- User-catered recommendations directly on the front page
- User-tailored search results
- Named movie collections and bookmarks beyond user wishlists
- A page listing all of a given user's activity, including reviews submitted, reactions given, and comments sent
- Nested comments
- Sorting options for comments and reviews
- A minimal private messaging system
- The ability to follow other users' reviews and view these on a user-specific feed page

Finally, we note that all of the features stated in the project objectives revolve around those available to users of the website. We also propose the development of a number of tools to assist non-admin moderators to manage the website and its community. Note that most of these features should be accessible only to moderators.

- The ability to add movies and edit the details of existing movies
- The ability to edit and delete the comments of other users
- The ability to place users into a "banned" or "suspended" state, without outright deleting their accounts
- The ability to lock reviews from being submitted to a movie page
- The ability to lock comment threads
- A Terms of Use for account creation
- A report system available to users

Scalability

Many of the decisions made during the development of the website were made on the assumption that the website's initial community would be small, and development speed was prioritized over scalability. However, as the website scales, our team has identified a number of areas in which the back end can be improved.

- The website can make use of a larger, more fully-featured API like that of IMDB. Though this would be more costly, it would improve the overall size of our database of movies
- The website can be made to cache the results of each API query. This would help in expanding our database of movies, as well as in the development of our recommendation system
- Operations on objects with a one-to-many relationship with other objects can be improved by maintaining and storing lists in a JSON field. For example, the IDs of reviews for a particular movie can be stored in a field within the table of the movie page itself. This could potentially result in massive improvements to lookup as the site's community becomes larger
- The website can make use of a more sophisticated DBMS, such as PostgreSQL. While it was initially planned that we would use PostgreSQL, we opted to switch to SQLite to ensure consistency between our test and production environments

Other features which can be adopted include making use of a CDN, and the development of tools to manage comment and e-mail spam. However, we believe that such features are not immediately necessary, and that development on these features can begin only when problems begin to arise that necessitate their inclusion.

Recommendation System

The recommendation system developed for this project was created to strike a balance between offering recommendations which are relevant to the current movie, and offering recommendations which users are likely to enjoy watching. The recommendation system can be further expanded to provide content more custom-tailored to each user, but would require a more fully-developed community with plenty of review samples from a wide variety of movies.

Improvements that can be made to the recommendation system include:

- Finding ways to include features such as release dates in the algorithm
- Modifications to the rating weighting system to incorporate likes, dislikes, and comments
- Incorporating user clicks and searches into the algorithm
- Using gradient descent or similar approaches to modify the feature coefficients based on page visits and the click-through rate for recommendations
- Decoupling recommendation rankings from external sites such as Rotten Tomatoes and prioritizing internal ratings and reviews
- Incorporating the text content of user reviews into the recommendation system through sentiment analysis
- Switching to collaborative filtering instead of using a nearest neighbours' approach on the bag-of-words feature vector

Other improvements can be made to expand the website's general machine learning systems, and website owners should constantly monitor the userbase's activity to prioritize development items.

References

Version 3 of The Movie Database (TMDb) API
< https://developers.themoviedb.org/3/getting-started/introduction>

The OMDb API
< https://www.omdbapi.com/>

YouTube
<https://www.youtube.com/watch?v=1wi0AHxjcn8&list=PLmolQff5pz3HEdLTVG5w7yG0lYLsfOnuQ&index=2&t=2s>
<https://www.youtube.com/watch?v=YH-ipgxlJzs&list=PLmolQff5pz3HEdLTVG5w7yG0lYLsfOnuQ&index=9&t=19s>

pandas - Python Data Analysis Library n.d.,
<https://pandas.pydata.org/about/index.html >

Django documentation,
<https://docs.djangoproject.com/en/3.1/ >

sklearn | balltree
< https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.BallTree.html >

Numpy Reference, Release 1.19, June 29, 2020
< *https://numpy.org/doc/stable/reference/index.html* >

Scipy.org, Release 1.5.4, November 04, 2020
< *https://numpy.org/doc/stable/reference/index.html* >