

MovieRec

Personalised Movie Recommendation Web Application

Individual Literature Review and Project Design and Implementation

May 2021

Abstract

As a result of e-commerce, social media, and overall enterprise digitisation, data has grown exponentially in the last decade. Consequently, it leads to information overload, making it difficult to interpret the data and make good decisions. Recommendation systems have risen in popularity in recent years as a solution to the problem of information overload. They do this by recommending items of interest to users by applying filtering and clustering approaches. However, recommender algorithms have several limitations, such as scalability and data sparsity, which increases as the number of users and items increases. This project aims to build a fully functioning web application using a suitable recommendation technique. The web application should recommend movies to users based on user preferences obtained through ratings. This report presents the design, implementation, testing and evaluation of the web application while also stating the reasons behind the chosen technique. The application uses ReactJS for the frontend, Django Rest for the backend and Google Firestore and Authentication for storing data. The application calls the TMDB API to retrieve data on all existing movies to display to the user. This report concludes by evaluating the functional and non-functional requirements by using user feedback obtained through testing, and it discusses the criteria for a successful project.



I certify that all material in this dissertation which is not my own work has been identified

Table of Contents

1 Introduction	1
1.1 Aims of the project	1
1.2 Report Overview	1
2 Summary of Literature Review and Specification	2
2.1 Summary of Literature Review	2
2.1.1 Content-Based Filtering	2
2.1.2 Collaborative Filtering (CF)	2
2.2 Related Applications	3
2.2.1 Movielens	3
2.2.2 Taste App	3
2.3 Summary of Project Specification	4
2.3.1 Functional Requirements	4
2.3.2 Non-functional Requirements	4
3 Design	4
3.1 Machine Learning Model	4
3.1.1 The idea behind the model and dataset	5
3.2 The Web Application	5
3.2.1 Process	6
4 Implementation	6
4.1 Item-based CF Model Implementation	6
4.2 The Web Application Implementation	8
4.2.1 Project Technologies	8
4.2.2 Project Architecture	8
4.2.3 Initialisation	9
4.2.4 Folder Structure	9
4.2.5 Backend (Django) Implementation	9
4.2.6 Frontend (ReactJS) Implementation	10
5 Testing	14
5.1 Model Testing	14
5.2 Website Testing	14
5.2.1 Unit Testing	14
5.2.2 User Testing	15
6 Evaluation	15

6.1 Requirements Evaluation.....	16
6.2 User Testing Results Evaluation.....	16
6.2.1 User Interface (UI) Feedback.....	16
6.2.2 Functionality Feedback	17
6.3 Comparison with other similar apps.....	18
6.3.1 Movielens.....	18
6.3.2 Taste App.....	18
7 Project Critical Assessment	18
7.1 Future work	19
8 Conclusion	20
9 Bibliography.....	21
10 Appendix	24
A. DEMO EMAIL BY FIREBASE SENT TO USERS FOR AUTHENTICATION.....	24
B. SURVEY SENT FOR USER TESTING	25
C. ADMIN PAGE	29
D. TOKENS WITHIN THE ADMIN PAGE	30
E. PICTURES OF THE USER INTERFACE.....	31

1 Introduction

In recent years, there has been tremendous growth in the use of the internet. It has become a widespread information infrastructure with more than 4.6 billion users [1] around the world as of 2019. With an increasing number of users, it results in a huge amount of data distributed across the digital universe. Scavenging through the immense data as a regular user can be difficult which results in the problem of information overload. People become overwhelmed not knowing what the right decision to make is. We use many different strategies to determine what items to buy, what movie to watch, or even what food to eat [2]. This could be through speaking to peers or exploring social media for inspiration. Successful companies like Amazon, Facebook, YouTube, and Netflix make use of recommender engines to help us discover new and relevant items which improve the user experience while also driving incremental revenue. Their demand has rapidly increased over the last few years.

The goal of a recommender engine is to generate meaningful recommendations to a collection of users for items they might be interested in [3]. Nowadays, recommender engines are being used everywhere without the knowledge of the public. For example, an application that gained popularity during the pandemic known as TikTok has a specific 'For You' page, which grasps the user's attention by generating videos based on the user's ethnicity, gender and likes and dislikes. It is personalized for each user thereby increasing the engagement between the app and the user. Recommender systems collect information about the user's preferences in two ways, either by implicit or explicit feedback [4]. Implicit feedback involves the application monitoring user behaviour to get information on watched movies, purchased products and more. On the other hand, explicit feedback is accomplished by directly asking the users to give feedback through ratings or reviews.

Some challenges that are faced when developing recommender engines include privacy issues and sparsity of data [5]. To produce reliable personalised recommendations, the systems need to gather as much data on the user as possible which can cause issues as users feel that the system knows too much about them. This is usually avoided by using techniques that sensibly and meticulously gain user data without making it accessible to malevolent users [6]. Additionally, the majority of users do not rate most of the items which lead to the rating matrix becoming very sparse. This decreases the chances of finding a set of similar users. Therefore, it is becoming harder to produce accurate recommendations.

This project aims to bring together machine learning and web development with the ultimate goal of implementing a personalised web application that recommends movies to the user using explicit feedback through ratings. This project comprises two main components. The first is the generation of a model, which follows a recommendation technique responsible for providing users with pertinent movies based on input. While the second is the implementation of the web application which should provide an attractive, responsive, and user-friendly user interface (UI). It must be understandable to others without any prior information.

1.1 Aims of the project

In summary, my project aim can be split into three:

- i. Create a well-suited machine learning model which generates movie recommendations based on user's explicit feedback.
- ii. Develop a web application that is an interface for users to rate movies and get recommendations based on their ratings called 'MovieRec'.
- iii. Learn and become more confident with web development as well as machine learning.

1.2 Report Overview

The report focuses on the Software Development Lifecycle (SDLC) of the Movie Rec Application.

1. First, a summary of the literature review focusing on the different recommendation algorithms along with other similar applications is presented.
2. Then a specification describing the functional and non-functional requirements is listed.
3. Next, a design to illustrate how my project is approached, detailing the abstract architecture of my project.
4. Following on from this, the development section demonstrates how each part of my project was implemented and justifies any modifications from the original specification.
5. The testing section outlines the processes used to assess the quality of my application and the recommendations generated – generally split into unit and user testing.
6. This is followed by an evaluation of results obtained from user testing and the subsequent design modifications that contributed to my project.
7. Then the next section discusses the critical analysis of my project, detailing any future work that may be undertaken.
8. Finally, a conclusion reflecting the entire project is given.

2 Summary of Literature Review and Specification

This section describes some existing research into recommender algorithms and illustrates some applications based on them.

2.1 Summary of Literature Review

Although the main purpose of this project is to build a web application, the model behind the application also needs to be addressed. A correct machine learning technique is important for a system to ensure recommendations are catered to the user. Recommendation algorithms are generally classified into two techniques: content-based and collaborative filtering. Modern recommenders also combine both approaches for better predictions – this is known as hybrid filtering.

2.1.1 Content-Based Filtering

Content-based filtering matches users interests by selecting items based on their content and matching it to user preferences [7]. The predominant focus in this area has been centred around recommending items with associated textual content such as movies and books where data like user reviews and descriptions are available [8]. For example, if a user has seen and enjoyed a Chris Pratt movie, then the system proposes other Chris Pratt movies to the user. Many methods have handled content-based filtering as an Information Retrieval (IR) task where the content associated with user preferences is viewed as a query and the unrated documents are scored with relevance or similarity to the query [8]. PRES (Personalized Recommender System) is an example of a content-based recommender system that generates dynamic hyperlinks for a website [9]. It makes recommendations by comparing user profiles with the content of each document in the collection [9]. In PRES, “the similarity between a user profile vector and a document is determined by the cosine similarity measurement” [9]. Cosine similarity measures the difference between two vectors of an inner product space. Similar vectors have a smaller angle with a cosine similarity value closer to 1. Other classification algorithms used for content-based recommender systems are Neural Networks, Decision Trees and K-Nearest [10].

2.1.2 Collaborative Filtering (CF)

Collaborative filtering operates by looking at a user’s historical data and collecting feedback in the form of ratings to compare them with other users to identify similarities in user behaviour. There are two approaches to CF. Neighbourhood-based (also known as memory based) and model-based approaches.

The neighbourhood-based approach observes the similarities between users based on ratings. Preferences can be calculated in two ways: item-based and user-based collaborative filtering. User-based CF predicts the items a user may like based on the ratings that similar users have given to that item. In this implementation of a movie recommender system, Bei-Bei [11] used the K-nearest neighbours’ algorithm to select the k number of users

with the highest similarity as the active user. This similarity was assessed using cosine measurement. Another way to calculate similarity is by Pearson's correlation which is a measurement of the linear relationship between two variables. Item-based CF uses the statement that two items are similar if a user gives similar ratings to both. This implies that if a user rated both 'Avengers' and 'Black Panther' a 4 out of 5 then this classifier assumes they are similar. Amazon [12] employs this method, they compare a user's purchased items to other similar items and displays the recommendation list to the user. They also use the cosine similarity measure to determine the similarity between the products.

The model-based methodology seeks to produce a model that can predict a user's rating of an unrated item. Methods in this category include the matrix factorization model, which has become one of the popular models among the model-based approaches due to its efficiency in dealing with large datasets. Netflix, which challenged the data mining communities to build systems that could outperform Cinematch's accuracy by 10%, has taken matrix factorization strategies to the forefront [8,13,14]. Although the final winning model was a complex collection of different algorithms, they discovered that accuracy improved when the factor model's dimensionality increased and when the descriptions involved a more distinct set of parameters [8, 13].

2.2 Related Applications

There are currently a few applications on the market that use these recommender algorithms to produce recommendations for users. This section introduces two of these applications, one of which is a website, while the second is a mobile application. I summarise their features and delineate the algorithms used.

2.2.1 MovieLens

MovieLens is a web-based recommendation application created in 1997 and is run by Grouplens, a research lab at the University of Minnesota. It recommends movies to its users based on their film preferences using explicit feedback which is obtained when the user provides direct relevance feedback [7]. MovieLens generates its recommendations using a variety of collaborative filtering algorithms, including item-based CF, user-based CF, and regularized singular vector decomposition, as part of the matrix factorization model.

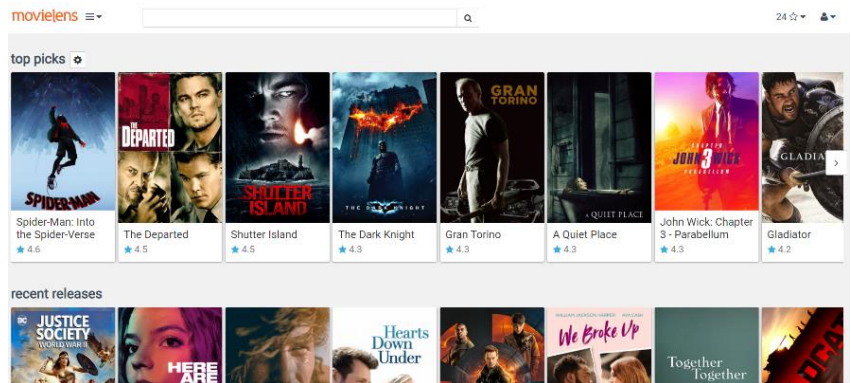


Figure 1: MovieLens Website User Interface

It has an easy-to-use frontend that is visually appealing, as seen in Figure 1. It categorises movies by genres and has a search bar that allows users to easily locate movies of their choice. It has predictive ratings of each movie, and it provides you with details about your ratings. These details include the frequency distribution of your ratings, average rating per genre and a comparison of your ratings to what other people have rated. MovieLens also addresses the cold-start problem by using preference elicitation methods.

2.2.2 Taste App

Taste [15] is a mobile application that can be downloaded on Android and iOS devices. It is very similar to MovieLens in that it lets you build your taste profile and gives you recommendations for movies and TV shows. The main difference between the applications is that the user interface is entirely different since it is a mobile app.

Taste App uses multiple collaborative filtering algorithms to build its recommendation functionality. The app has a match percentage for each movie, which is a measurement of how much they think you will enjoy the movie. It also factors in recency, serendipity, and rarity so that users can discover the hidden recommendations.

It has a swiping motion where you swipe right to save the movie and left if you do not want to watch it. However, I found that when I swipe continuously, it repeats movies because it ran out of recommendations. Moreover, the match percentage changes when I click on the movie, which means it is not consistent and the algorithm is not trustworthy.

2.3 Summary of Project Specification

My project specification has been altered over the year, which is mainly due to time and complexity constraints. My project's main goal has not changed, but the idea for the user interface has been improved. This includes adding a 'watch later' tab, and the ability to add friends so that you can also view your friends' recommendations. The updated specification is listed below.

2.3.1 Functional Requirements

- A user must be authenticated using an email and password to use the web application.
- User must be able to rate movies of their choice out of 5.
- Reliable recommendations should be provided to the user based on their movie ratings.
- A user can search for movies of their choice and the movie should be displayed with its name and an image.
- Each movie's details should be displayed, including the cast, synopsis, and genres of the movie.
- Movies should be split into genres so that they are easy to find by users and they can be added to 'watch later'.
- User should be able to add friends to view their recommendations.

2.3.2 Non-functional Requirements

- Once the user rates a movie, the recommendation list must be updated based on the user ratings instantly. I.e., the response time should be fast (within roughly 2/3 seconds). Recommendations should be done continuously.
- User details must be securely stored in a database along with encrypted passwords.
- The application should be scalable so that other items such as TV shows and books may be added to the website therefore it should have the capability to handle them.
- The web application will have an easy to use and simple design that is easily understandable by users.

3 Design

During the design phase, developers start the high-level construction of the system architecture along with discussing the technical details and risks associated with each requirement. The design is split into two stages: construction of a machine learning model and web application. In this section, I will also describe the abstract architecture of my project.

3.1 Machine Learning Model

As stated in the literature review, a recommender engine can be built using many different techniques. In this section, I will be discussing the positives and negatives of each method to support my choice of technique.

Content-based filtering is a simple technique to implement. It is scalable as it does not require any historical data on other users (since recommendations are specific to the user). Nevertheless, this approach only produces suggestions based on current user preferences; it lacks the ability to extend, which means it would be unreliable if a user's interests shift over time. It also necessitates a great deal of domain knowledge. The collaborative filtering strategy, on the other hand, has the benefit of requiring no domain knowledge since the embeddings are automatically learned. Nonetheless, it suffers from a cold start problem, which means that when new items are added to the system, there is insufficient data to provide optimal recommendations. A user-based CF matrix is easy to implement, but it is computationally expensive due to the dynamic personalities of users. Item-based CF, on the other hand, is not dependent on the users. Furthermore, item-based CF allows scalability and performance

since it creates the expensive similar items matrix offline [12]. The model-based approach identifies more interpretable low-dimensional representatives for users and items, therefore improving recommendation performance. However, this method has the disadvantage of producing a sparse matrix because users would not rate all items, resulting in imprecise recommendations. An advantage of CF over content-based is that it helps users discover new items since other similar users could have rated it highly. Finally, the benefit of using the hybrid method compared to a single approach is that it leverages the strengths of both by making predictions separately and then combining them. This results in better and more accurate recommendations.

Reflecting on the literature and shortcomings of recommender engines, the item-based neighbourhood CF approach seems like a good fit for this project. It is selected due to the minimal amount of time available to complete this project. A collaborative filtering algorithm aims to recommend new movies or to estimate the probability of a particular user enjoying a certain item based on the user's previous likings, and the opinions of other like-minded users [16]. I will be implementing this technique using the KNN classifier and cosine similarity. KNN is regarded as a lazy classifier due to that it does not learn when it is trained. It instead memorizes the dataset. There is no training phase or testing phase to evaluate the accuracy of the classifier. All the data is used to generate the model and once it is generated, it can be used to make predictions. This is beneficial for my project because the item-based CF model does not depend on the users therefore not requiring a training and testing dataset of users. It should instead accept a movie title or Id as input and return a list of similar movies.

3.1.1 The idea behind the model and dataset

In a typical CF scenario, we have a list of n users $U = \{u_1, u_2, \dots, u_n\}$ and a list of m items $I = \{i_1, i_2, \dots, i_m\}$ (in this project, these items are movies). Each user u_i for $0 < i \leq n$ has a list of items I_{u_i} about which the user has expressed their opinion. These opinions are given explicitly through ratings. CF techniques represent an $n \times m$ user-item matrix as a rating matrix, A [16]. Each entry a_{ij} in A signifies the rating score of the i^{th} user on the j^{th} movie. Now, for an item-based CF, we isolate the users that have rated movies j and k and then apply the cosine similarity measurement to determine the similarity s_{ij} . This similarity can be computed using definition 1 [16].

$sim(i, j)$ is given by

$$sim(i, j) = \cos(\vec{i}, \vec{j}) = \frac{\vec{i} \cdot \vec{j}}{\|\vec{i}\|_2 * \|\vec{j}\|_2}$$

where “.” denotes the dot-product of the two vectors.

Definition 1: Cosine similarity calculation [16]

applied to 9,000 movies by 600 users and the large containing 27,000,000 ratings applied to 58,000 movies by 280,000 users.

To generate the $n \times m$ matrix, I need a dataset of user ratings of movies. This dataset can be acquired from the Grouplens [17] website. It has many datasets for various purposes that are freely accessible for developers to use. The specific dataset I will be using is extracted from Movielens and is recommended for education and development purposes. It contains two datasets – the small containing 100,000 ratings

Some of the useful libraries I will be using to generate my model include:

- NumPy – a library containing mathematical functions.
- Pandas – data analysis library that offers data structures and operations to manipulate them.
- SciPy – a library that provides efficient numerical routines.
- Seaborn – to generate graphs to show the relation between users and movies.

3.2 The Web Application

In this section, I will be noting the abstract architecture of my project along with the features my application should have.

Figure 2 depicts the web application's theoretical architecture. As previously indicated, I will need a web server that incorporates the KNN model to generate movie recommendations based on a given movie Id. This web

server will interact with the web client to display those recommendations to the user. The movies should be presented in a nice format to provide a user-friendly user interface. Therefore, the web client should communicate with an API that returns details about individual movies, including a poster image of the movie so that they can be displayed on the screen for the user. The web client will also require some form of authentication before being accessed by users and the user information which includes their ratings will be stored in a database. The web client should be able to query this database to obtain the required information when needed.

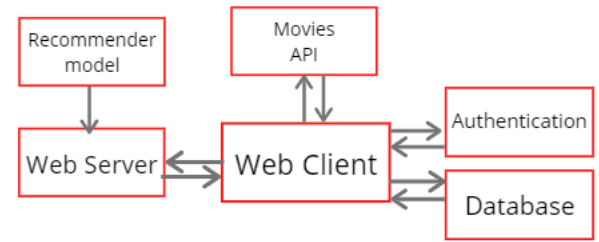


Figure 2: Abstract Project Architecture

The following are the application’s specifications. After the user has been authenticated, they should be taken to a preference elicitation page where they can rate a selection of given movies before proceeding to the main webpage. This is done to mitigate the cold start problem that occurs for new users when not enough data is collected on them to produce recommendations. Thereafter, a user should be able to rate and add to ‘watch later’ movies of their choice. These movies can then be accessed by clicking on the respective buttons at the top of the page. Movies can also be found by genre or by searching for them by title. These films should be displayed with their poster images and titles. Furthermore, a user should be able to add another user as a friend and their recommendations should be available to each other.

3.2.1 Process

Before starting implementation, a development process needs to be identified so that my project can be established in a structured manner. A very popular methodology which many big organizations use is Agile development [18]. This is an umbrella term for many practices which involve evaluating requirements and acquiring solutions through a collaborative effort of self-organizing and cross functional teams. Rapid Application Development (RAD) is an agile project management strategy where it uses prototyping as a mechanism for iterative development [19]. RAD emphasizes the use of software and user criticism over stringent planning and documentation standards. The user is given the opportunity to provide input to the developers at the end of each development cycle [20]. RAD is intended for short-term projects that will be completed in a few months.

RAD is a suitable methodology for my project because it involves building a web application that requires a lot of client interaction. RAD allows quick implementations where the client can check to make sure everything operates appropriately before producing new features. In this case, I will be acting as the client to constantly test the application to ensure all features function properly. Furthermore, weekly meetings with my supervisor allow for further discussions of the development plans as well as regular testing by my supervisor.

4 Implementation

Implementation is the process of turning a design into working software. This section focuses on how the design was made a reality along with explaining the technological choices that were made during the process and the architecture of my project. As this is a web application, this section also discusses and justifies the user interface design choices. It describes how each page was implemented along with any challenges that were encountered.

4.1 Item-based CF Model Implementation

Before developing any model, a dataset must be chosen to work on. In the original specification, I planned to use the small dataset from the Movielens website [17]. However, while implementing this model, I realized that the dataset was too trivial to produce accurate recommendations. This was because after data cleansing, the dataset became much smaller. Additionally, the dataset only contained 9,000 movies, which I considered inadequate to reflect a movie recommendation site. For these reasons, I generated the model using the larger dataset from Movielens, which has over 55,000 movies. The files I used from the dataset are:

- Ratings.csv file – has 27,000,000 ratings by users to different movies and was used to generate the user x item matrix
- Movies.csv file – has the movies information such as Id, title, and genres.
- Links.csv file – this file contains a list of movie Ids to TMDB Ids to IMDB Ids and is used to translate between TMDB Ids and movie Ids.

After I obtained the dataset, data cleansing had to be done to ensure any outliers are removed. To carry out data cleansing, I observed that there is a gap in the dataset where some movies had not been rated by any user and some users had not rated any movie. I first began by removing such users and movies as well as movies that did not have a TMDB Id, since displaying these movies to the users would be difficult. Then I plotted the number of votes cast by each user and the number of votes each movie received in two graphs represented in figures 3a and 3b. As seen on the graphs, I decided to set a bottom threshold of 150 for the number of votes a movie received and 200 for the number of votes a user has given. I also set an upper threshold of 7500 for the number of votes a user has given to avoid those who could have just voted randomly for the sake of voting. The bottom threshold was set to ensure each film received enough votes before comparing them. These thresholds were finalised by first displaying various thresholds on graphs and selecting those that I felt best represented the dataset.

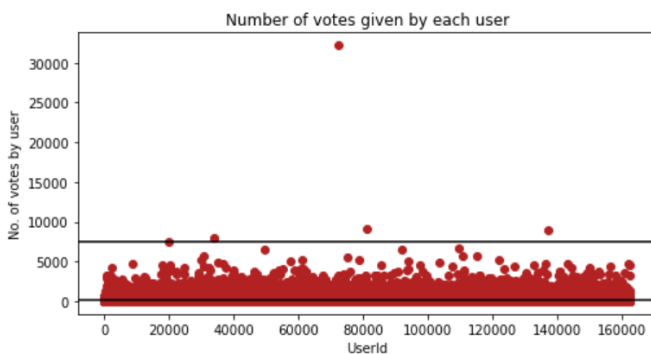


Figure 3a: A scatter graph showing the number of votes given by each user

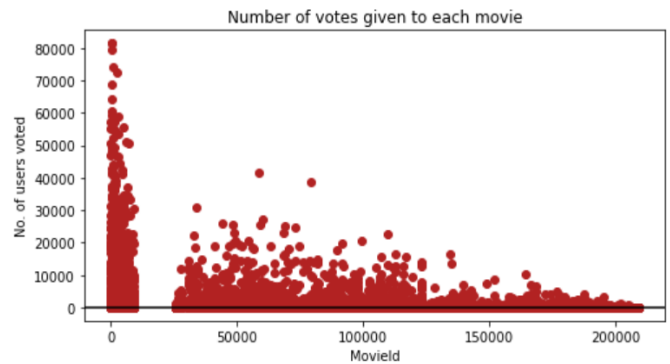


Figure 3b: A scatter graph showing the number of votes given to each movie

After pre-processing the rating dataset, I created the final user x movie matrix. This matrix was very sparse since there were many movies that the user had not rated, resulting in a lot of 0s in the matrix. To avoid this, I used the ‘csr_matrix’ from the Scipy library to generate a compressed sparse row matrix that can be used to fit the model. For the classifier, I used the ‘NearestNeighbors’ method which incorporates unsupervised nearest neighbour learning and serves as a uniform interface for three main nearest neighbour algorithms: BallTree, KDTree and Brute-Force [21]. As mentioned before, it does not require a training and testing dataset since it instead memories the dataset to make predictions using the metric given. I used the cosine similarity metric with a neighbours’ value of 20 and the algorithm as ‘brute’. The code used to generate the model is presented in figure 4.

```
In [13]: # Generating the model
from sklearn.neighbors import NearestNeighbors
knn = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=20, n_jobs=-1)
knn.fit(user_movie_matrix)

Out[13]: NearestNeighbors(algorithm='brute', metric='cosine', n_jobs=-1, n_neighbors=20)
```

Figure 4: KNN implementation

Having generated the classifier, it needs to be used within Django to produce recommendations. Therefore, multiple files had to be saved so that they can be accessed within Django. These files include:

- knnModel.pkl – this is the model saved as a pickle file using the ‘joblib’ library. Joblib efficiently serializes Python objects with NumPy arrays. It is significantly faster than the ‘pickle’ library on large arrays, hence it is used to generate the pickle file.

- `sparse_matrix.npz` – to use the model, the `.kneighbors` method is called, which requires a query point to generate k-neighbours of that specific point. This query point can be accessed using the sparse matrix. Therefore, it must also be saved and is saved using the NumPy library's `'save'` method.
- `movie_tmdb_map.npy` – a hashmap of {movie Ids to TMDB Ids} is also needed to convert between the two Ids. The hashmap is saved using a NumPy method `'save'`.

4.2 The Web Application Implementation

4.2.1 Project Technologies

ReactJS Framework is used to build the frontend of the application. ReactJS is a JavaScript library useful for creating user interfaces. It was created by Facebook to "enable the creation of interactive, stateful, and reusable UI components" [22]. ReactJS has many advantages over other front-end frameworks like Angular. It has a Virtual Document Object Model (DOM) which can be rendered on either the client or server side, making it a highly efficient performer. For a change to be reflected on a webpage, ReactJS first changes the Virtual DOM instead of instantly updating the browser DOM, which results in fast-performing applications [23]. It also has a shallower learning curve compared to Angular, which was desirable due to my lack of experience with any of these technologies.

Python framework Django Rest is used to build the backend of the application. Django Rest is a robust and adaptable toolkit for creating Web APIs. It enables the building of web applications with very few lines of code [24] and minimal overhead. Furthermore, Django employs an MVT (Model-View-Template) approach [25] in which the views are replaced by templates and the controllers are replaced by views from the traditional MVC architecture. Django was chosen since the backend needs to run the machine learning model to provide recommendations to the users. Machine learning models are predominantly implemented using Python due to the numerous useful in-built libraries such as SciPy, and Scikit-learn, therefore making the transition between Jupyter Notebook and Django seamless.

In my application, user ratings must be stored in a database. Initially, I decided to use the Google Firebase Realtime Database. However, I later discovered that the advantages of using Google Cloud Firestore outweighed those of using Firebase Realtime. Some of the benefits include better querying of more structured data as data is stored hierarchically, it is designed to scale, and it facilitates flexible web development [26]. For user authentication, I used Google Firebase authentication, which stores the users' passwords in a hash, making it very simple to implement. It also integrates nicely with the other Firebase services.

The web application requires an API to fetch film data to be displayed on the UI for the user. I decided to use 'The Movie Database (TMDB)' [27] API which lets us find and search for movies free of charge. TMDB was selected because of its extensive archive of movies and TV shows which is regularly updated. It also has helpful documentation on the developer's page [27] which demonstrates how to use the TMDB API. The API provides the details (title, genres, cast and more) and the images for a movie.

4.2.2 Project Architecture

Figure 5 illustrates how the different technologies interact with one another to produce a functional web application. The frontend communicates with Firebase Authentication to log in users using an email and a password. This email must be verified, which is done using Firebase Authentication. It sends an email with a link to verify the account before proceeding onto the website. The frontend also communicates with the Cloud Firestore database to store, delete, update, and retrieve data about the users and movies. The frontend also communicates with the TMDB API to obtain data about the

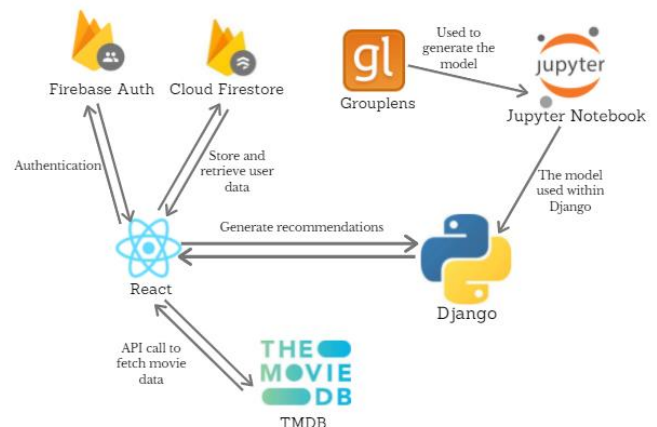


Figure 5: Project Architecture

movies that will be displayed on the user interface. To generate the recommendations, the frontend sends a POST request to the backend with a list of TMDB IDs. The rating model is then used by the backend to generate a list of similar movies and the TMDB IDs of the movies are sent back to the frontend to be displayed on the screen. To produce recommendations, the backend uses the model generated by the Grouplens dataset.

4.2.3 Initialisation

To start the implementation of the website, I created a virtual environment for my project. A virtual environment helps keep all your dependencies in one location, preventing other applications from interfering with my project. The environment was created using *'pipenv shell'* and the packages required for the development were installed using *pip*. The *pip* function manages all the packages and dependencies by confining them into the virtual environment. Having a virtual environment also helps us to easily monitor dependencies, and we can get all the packages installed by running the command *'pip freeze > requirements.txt'*. With the virtual environment created, I was able to start building the web application.

As previously stated, my project's two major frameworks are ReactJS and Django. Therefore, I began by installing the two and creating two separate projects. I then connected them using the *settings.py* and *urls.py* files in Django. ReactJS uses *'npm install'* which is distributed with *node.js* to download React bootstraps while Django uses *'pip'* to import its dependencies. To start the ReactJS project, I used *'npm start'* which launches the development server. Then running *'npm run build'* bundles the app into static files for testing or production. I then set up a Google Firebase account to connect to the Cloud Firestore and Firebase Authentication. The database is connected to the frontend using an API. Finally, to ensure that my project is backed up, I connected my code to GitHub.

This completes my project's initial setup.

4.2.4 Folder Structure

My project folder structure is shown in figure 6. Django acts as a server while ReactJS acts as a client. When a Django project is created, the root of the project is the folder that contains the *'manage.py'* file. The root of my project is the *'backend'* directory. The inner *'backend'* folder is the actual Python package for my project. The files inside this directory are those that set up the configuration of my project. The *'RecommenderApp'* directory contains the python files required for the backend logic. The *'movie-recommendation-react'* folder contains my project's frontend code, which includes the HTML and JavaScript for the web pages that are rendered when an endpoint is defined.

4.2.5 Backend (Django) Implementation

This section describes the implementation of the backend using the Django tools.

Admin and Views

The most powerful part of Django is its' automatic admin interface [28]. When a new Django project is created, the framework produces a Django admin page, which is a built-in feature that generates a user interface for performing create, read, update, and delete operations. The admin page can be accessed locally by using the weblink *'http://127.0.0.1:8000/admin'* and is shown in Appendix C. Since the admin page is intended for administrative purposes, logging in requires the creation of a superuser via the command line. For this project, I used the admin page to save an authentication token for each account

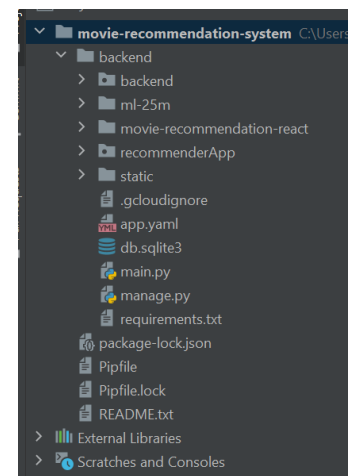


Figure 6: Folder Structure of my project

```
# Url routing
urlpatterns = [
    url(r'^backend/auth/login/$',
        CustomAuthToken.as_view(),
        name='auth_user_login'),
    url(r'^backend/auth/register/$',
        CreateUserAPIView.as_view(),
        name='auth_user_create'),
    url(r'^backend/auth/logout/$',
        LogoutUserAPIView.as_view(),
        name='auth_user_logout'),

    url('backend/suggestions', Recommendations.as_view(), name="recommendations"),

    # Anything else is redirected to frontend
    url(r'^$', FrontendAppView.as_view()),
]
```

Figure 7: URL endpoints in Django *"recommenderApp/urls.py"*

created on the frontend. This token is stored in the backend SQLite3 database, which is installed by default when we launch Django. This establishes a secure method of transferring data from the backend to the frontend.

The `urls.py` file, seen in Figure 7, is used for routing in Django. It sets up an endpoint that calls the appropriate function within the `views.py` file [29]. Django views are an essential component of the framework that accepts a web request and returns a response. Views are mainly used to fetch and modify objects, as well as to render information and return it as a JSON object. I created class-based views that take a request from the frontend and return a response so that it can be displayed. I implemented 5 view functions that are called from the frontend using the endpoints as an HTTP process.

Token Authentication

The endpoint `'backend/auth/register'` calls the view function `'CreateUserAPIView'` as a POST request. When a user signs up for the first time, TokenAuthentication [30] is used to generate a new user and a token, which is then saved in the SQLite3 database. This authentication system is suitable for client-server setups and employs basic token-based HTTP Authentication. These tokens are saved in the 'Tokens' section of the admin page, as seen in Appendix D. The endpoint `'backend/auth/login'` invokes the view function `'CustomAuthToken,'` which returns a response containing the previously generated token, which is then stored in the frontend user's local storage for future usage.

Recommendations

Finally, the endpoint `'backend/suggestions'` calls the `'Recommendations'` view function. The frontend sends a POST request with TMDB Ids, and the `'Recommendations'` function produces a response with recommended movie Ids. When the `'Recommendations'` view receives a web request from the frontend, it invokes the `'recommendations.py'` file, which contains a `'get_recommendations'` function. This function uses the hashmap to convert the TMDB Ids to movie Ids, which are then fed into the item-based CF model to produce recommendations. The model's list of Ids is then converted back into TMDB Ids using the hashmap before returning the list of unique Ids to the frontend so that the movies can be displayed to the user. To ensure that only a limited number of movies are displayed to the user, I set a threshold of 25 movies, so the `'get_recommendations'` function only returns the 25 most similar movies to the frontend. To access this endpoint as an API call, the token key must be provided in the Authorization HTTP Header.

To run Django, we use the command `'python manage.py runserver'`.

4.2.6 Frontend (ReactJS) Implementation

This section illustrates the implementation of the user interface along with justifications for any UI choices that have been made.

React Folder Structure

As shown in figure 8, the `movie-recommendation-react` folder contains subfolders called `'pages'`, `'components'` and `'services'`. The `'pages'` folder comprises of web pages that are mapped to a URL and are displayed to the user. The elements that make up the web pages and are replicated through various pages are stored in the `components` folder. Examples of these include the header and the rating component. Since it adheres to the 'Don't Repeat Yourself' principle, this folder layout creates an optimal programming environment. The database API configuration, as well as the functions for logging in and out of the programme, are located in the `services` folder. The `index.html` file, which is stored in the `public` folder, is the main file that is called and rendered into different pages. This gets called from within the `app.js` file. The pages get rendered using URLs. Any URL that is not in the Django `urls.py` file is forwarded to the `urls.py` file within ReactJS, which then routes it to the correct page.

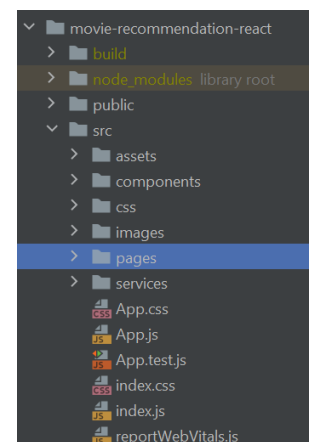


Figure 8: React Folder Structure

React Routing

Routing allows movement between different parts of the application. ReactJS has an inbuilt router package that uses dynamic routing to help navigate between pages. This is achieved in the app.js file and happens to be a standard switch statement displayed in figure 9. It renders the various pages based on the URL given. For e.g., ‘.../main’ renders the main page, while ‘.../recommendations’ displays the recommendations. It accomplishes this by determining whether or not the user is authenticated. There are two options for routing: public or private. The pages that need authentication go down the private path, while the login and signup pages go down the public path. This is because the application is only accessible to authorised users.

```
render() {
  return this.state.loading === true ? <h2>Loading...</h2> : (
    <div>
      <Router>
        <Switch>
          <Route exact path="/" component={Home}/>
          <PublicRoute path="/signup" authenticated={this.state.authenticated} component={Signup}/>
          <PublicRoute path="/login" authenticated={this.state.authenticated} component={Login}/>
          <PublicRoute path="/forgotpassword" authenticated={this.state.authenticated} component={ForgotPassword}/>
          <PrivateRoute path="/main" authenticated={this.state.authenticated} component={Main}/>
          <PrivateRoute path="/profile/settings" authenticated={this.state.authenticated} component={Settings}/>
          <PrivateRoute path="/profile/friends" authenticated={this.state.authenticated} component={FriendsPage}/>
          <PrivateRoute path="/profile/ratings" authenticated={this.state.authenticated} component={RatingCharts}/>
          <PrivateRoute path="/movie_details" authenticated={this.state.authenticated} component={MovieDetails}/>
          <PrivateRoute path="/cast" authenticated={this.state.authenticated} component={CastMovies}/>
          <PrivateRoute path="/recommendations" authenticated={this.state.authenticated} component={Recommendations}/>
          <PrivateRoute path="/watchlater" authenticated={this.state.authenticated} component={WatchLater}/>
          <PrivateRoute path="/ratedmovies" authenticated={this.state.authenticated} component={MoviesRated}/>
          <PrivateRoute path="/genres" authenticated={this.state.authenticated} component={Genres}/>
        </Switch>
      </Router>
      <Footer/>
    </div>
  );
}
```

Figure 9: React Routing to different pages

React State and Lifecycle

Components in ReactJS must be defined as either classes or functions. When you create class components, you must provide a render() method that returns the page's HTML. ReactJS components have a state object that, when modified, re-renders the component. Each component also has lifecycle methods that are called at various stages of displaying the HTML. The main lifecycle methods used are as follows [31]:

- Constructor() – This method is called before the component is mounted. It initializes the local state by assigning an object to ‘this.state’. The props are passed to the constructor and can be used for initializing the state.
- ComponentWillMount() – This method is also invoked before mounting occurs and any changes you want to make before rendering the component is done in this method.
- Render() – Renders the HTML and displays it
- ComponentDidMount() – Invoked immediately after a component is mounted.
- ComponentDidUpdate() – Invoked after an update has occurred or state has changed. Mainly used to make network requests to compare current props to previous props.

React pages Design & Implementation

To access the application, a user is required to log in or sign up using a valid email address. To validate the account, it uses Firebase Authentication, which sends a verification email to the email address the user used to sign up. A forget password button is also available on the login page, and when pressed sends an email with a link to change the password. These draft emails can be set up in my Google Firebase account. A demo email is shown in appendix A.

After successfully signing up, the user is asked to rate a number of films in order to receive initial suggestions. This is referred to as preference elicitation, and it lasts until the user has rated enough films for the model to generate recommendations. My specification assumes that if a user gives a movie a 4 or 5 out of 5-star rating, they enjoyed it. I used this idea to decide which TMDb IDs should be sent to the backend as the model query point. The films that appear on the preference elicitation page were selected by browsing websites such as Rotten Tomatoes and other similar websites for popular movies of each genre. I chose this method over showing random movies from the ‘movies.csv’ file because it includes many unpopular films, meaning users would take a long time searching to find movies to rate. This is simply to ensure that the initial ratings can be completed

quickly. This eliminates the cold start issue that collaborative filtering strategies have. It solves the issue by requiring the new user to rate certain items explicitly in order to instantly construct a profile for her/him [32].

After completing their initial ratings, the user is routed to the main page. The main page is made up of three movie carousels obtained through a call to the TMDB API, as seen in figure 10. The first two carousels display the most popular and recent films, while the third displays the top grossing films of all time. I chose to implement the third carousel because top-grossing films are popular with the majority of users thereby increasing the number of ratings an individual completes. The main page also has a search bar that generates search results using the TMDB API.

```
// Gets the popular movies from API
getPopularMovies = () => {
  fetch( input: 'https://api.themoviedb.org/3/movie/popular?api_key='
    +this.apiKey+'&language=en-US&page=1') Promise<Response>
    .then(data => data.json()) Promise<any>
    .then(data =>
      {
        this.setState( state: {popular:[...data.results]})
      })
    return this.state.popular;
}
```

Figure 10: TMDB API Call

Each movie is displayed with its image as the main identifying feature. When the user hovers over the image, it displays details such as the title, synopsis, and rating scale for that movie. I chose to use a hovering feature rather than showing the information on the screen because it saves space and creates a clutter-free UI. When the user clicks on the 'View Details' link on the hover feature, they are redirected to a page with detailed information about that film. The genres, cast, and a carousel of similar films are all located on this page. The cast and genres within this page can also be clicked. It makes use of the TMDB API to generate a list of movies in which the actor appears in or movies of a particular genre. Furthermore, when the user hovers over a movie, a 'watch later' button appears. It enables them to save movies so that they can be conveniently accessed in the future.

The 'Watch Later' and 'Movies Rated' tabs on the header display a list of movies in a similar format to when a user searches for a movie. In the watch later tab, the button inside the hovering feature switches to 'remove movie,' and once pressed, it is no longer clickable to ensure it cannot be clicked twice, and the movie is deleted from the list. The 'Genres' tab has a list of dropdown options, and when a certain genre is selected, the TMDB API is called to generate a list of movies of that genre. For example, when the 'action' genre is selected, the URL changes to '.../genres#action'. When a different genre is chosen, the `componentDidUpdate()` function is called, which checks the text after the hash using `'props.location.hash'` and, if they differ, the state is updated and the component is re-rendered. This makes for a smooth transition between genres.

The page's header includes a profile tab with a dropdown list of 'Settings,' 'Your Ratings Data,' and 'Your Friends' buttons. The settings page contains a form that, when submitted, calls the 'handleSubmit' function. This form allows the user to add additional information about themselves and/or change their password if necessary. The 'handleSubmit' function process the updates by querying the database and calling a Firebase Authentication API method to update the user password. Moreover, the user has the option of deleting their account. When they press this button, a warning emerges asking them to confirm their decision. If they press yes, the account will be deleted permanently, and the user will be returned to the login page. When a user is deleted, they are removed from the Firebase Authentication system and are no longer able to login to the website. The user profile, as well as the subcollections containing the user's ratings and friends list, is then removed from the Cloud Firestore Database. The token that was created in the backend is also deleted. Next, the ratings data page displays charts created with Canvas JS, a charting library that generates fast graphs in ReactJS. I have created three charts that display the distribution of ratings per day, over movie release years, and the distribution of rating values. Finally, the friends page shows the user's friend list as well as their friend requests. A user can add a friend by entering their email address. Once the 'add' button is pressed, it checks to see if the email address is valid and sends a request to that user. If an error occurs when performing this operation, the request is terminated, and a relevant error message is shown to the user. To see their friends' recommendations, they must accept the request first. Currently, I set a limit of 5 friends as it can get too complicated with too many people.

The key objective of my project was to show the user movie recommendations. When you click on the 'Recommended Movies' tab in the header, recommendations are generated in real time. It sends a fetch request to the backend and returns a list of TMDB IDs, which are then used in the TMDB API request. The initial selection in which TMDB IDs from the database are sent to the backend is accomplished by a random selection of all movies that have been scored 4/5 out of 5, with the presumption that any scores of 4 or higher indicate that the user enjoyed the movie. I made an implementation choice to randomize the IDs because it allows for new suggestions to be generated each time the user refreshes the tab. To restrict the number of random IDs sent to the backend, I used an if statement to pick a certain number of IDs based on the total number of movies that scored 4/5 out of 5. At the top of the page, there is a dropdown button that contains the email addresses of friends that the user has added and accepted. When they click on another email, the page is immediately loaded with their friends' recommendations.

Screenshots of the User Interface

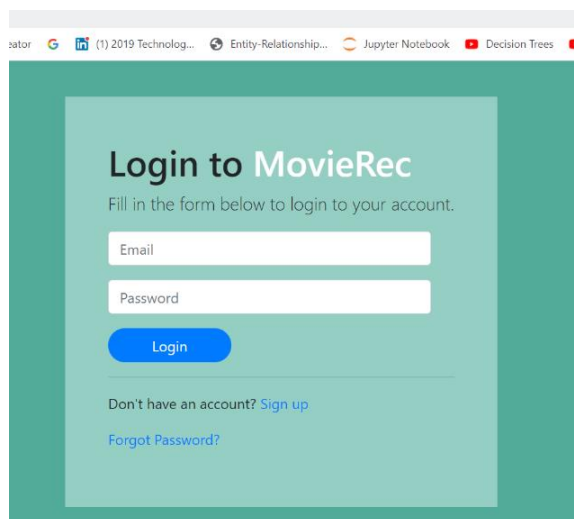


Figure 11: Login Page

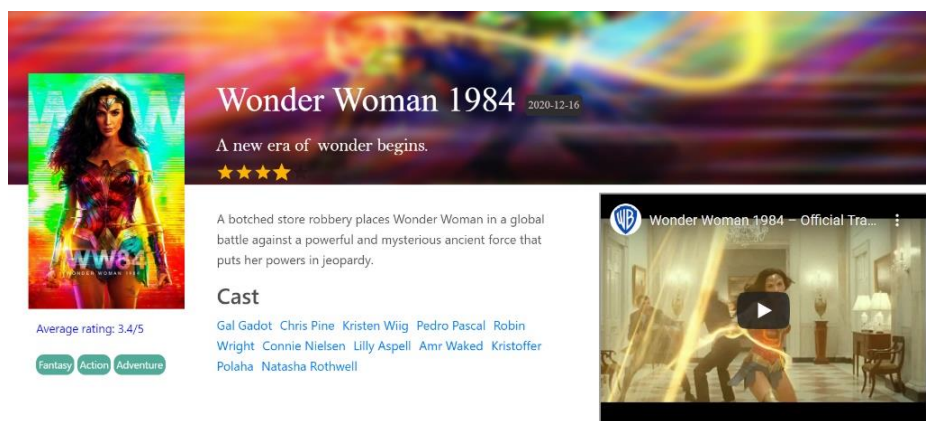


Figure 12: Wonder Woman movie details page

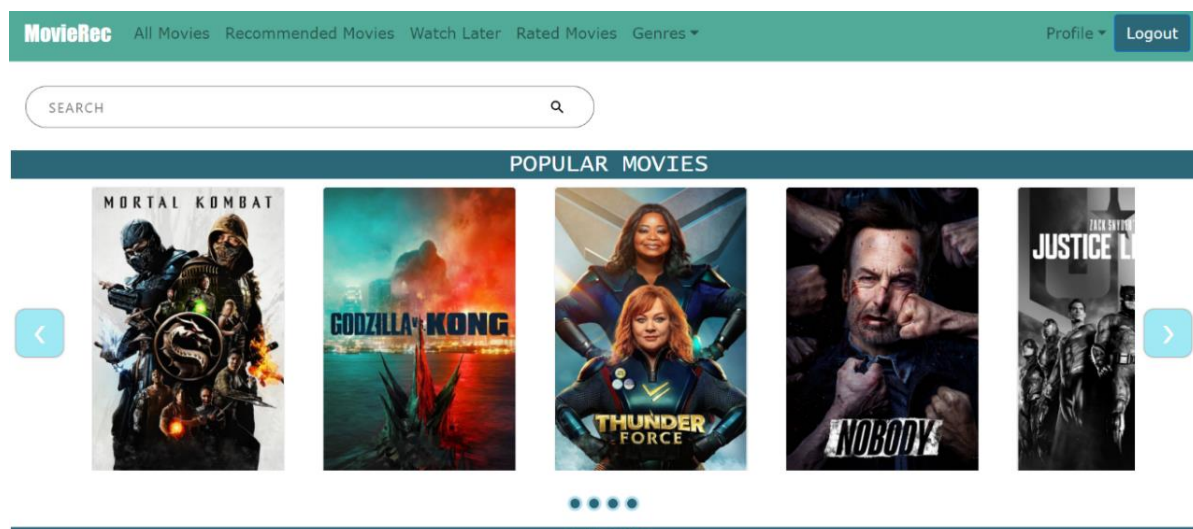


Figure 13: Main Page

5 Testing

Software testing is an essential part of the SDLC process. It is the process of evaluating the functionality of the code to ensure that the software application meets the requirements. My project's testing is divided into two parts: testing the model and testing the web application.

5.1 Model Testing

Traditionally, when evaluating a model, the dataset is split into training and testing data. In contrast, the NearestNeighbors model I generated trains itself by memorising the entire dataset. As a result, no training-testing split is required. Consequently, the accuracy of the model cannot be mathematically tested. Instead, to test the model produced, it needs to be analysed with real situational cases. I created a method that takes a movie title as input and returns a list of movies that are similar using the classifier. Figures 14a and 14b show examples of movies that were tested using this method.

Recommendations for: Avengers: Age of Ultron (2015)

Out[15]:

	Movie Title	Distance
10	Captain America: The Winter Soldier (2014)	0.212176
9	Captain America: Civil War (2016)	0.257727
8	Ant-Man (2015)	0.257986
7	Iron Man 3 (2013)	0.266673
6	Avengers, The (2012)	0.269243
5	Captain America: The First Avenger (2011)	0.283726
4	X-Men: Days of Future Past (2014)	0.287655
3	Guardians of the Galaxy (2014)	0.288036
2	Thor: The Dark World (2013)	0.305880
1	Thor (2011)	0.320097

Figure 14a: Avengers similar movies

Recommendations for: Maze Runner, The (2014)

Out[30]:

	Movie Title	Distance
10	Maze Runner: Scorch Trials (2015)	0.401590
9	Divergent (2014)	0.411792
8	The Hunger Games: Mockingjay - Part 1 (2014)	0.446800
7	The Hunger Games: Catching Fire (2013)	0.456245
6	The Hunger Games (2012)	0.476726
5	Jurassic World (2015)	0.476906
4	Guardians of the Galaxy (2014)	0.498873
3	Lucy (2014)	0.499008
2	Now You See Me (2013)	0.502581
1	World War Z (2013)	0.503335

Figure 14b: Maze Runner similar movies

One can see that the model generated very similar movies to that of the input (most similar movies at the top of the list). To further prove this statement, I compared the output with similar movies listed on Google's 'People also search for...' section. It was shown that those that I highlighted for the film 'The Maze Runner' did appear on that list, demonstrating that the model is indeed functioning as it should and producing adequate recommendations. However, Google does have a much larger database of movies and it prioritises showing users new movies. As a result, most of the list retrieved from Google included movies from recent years, which my model lacked.

5.2 Website Testing

Using the RAD framework for developing my project resulted in regular weekly testing. Website testing is further subdivided into unit testing and user testing. Unit testing is the lowest level of software testing. It tests a unit of software which is the smallest testable piece of software and is often called 'module' or 'component' interchangeably [33]. User testing, on the other hand, is performed by end-users. It promotes user-developer collaboration while also lowering the cost of potential fixes because resolving errors during UAT is much cheaper than fixing them after release [34].

5.2.1 Unit Testing

Django includes a robust unit-testing framework that encourages you to write tests while developing [35]. When you create a project in Django, it automatically generates a tests.py file that uses the Python standard library module 'unittest'. It defines tests using a class-based approach. I tested the various views, such as creating a token and generating recommendations, with a test Client that functions as a mock Web browser by simulating GET and POST requests on the URL endpoints [36]. The tests invoke the intended view, which returns an HTTP

response. Using the 'assertEquals' function, I compared the predicted and actual responses. The command 'python manage.py test' can be used to execute these Django unit tests from the command line.

ReactJS testing is performed in the app.test.js file, which is triggered by the command 'npm run test.' The tests include various scenarios for user authentication. I aimed to test the Firebase Authentication functions which allow a user to sign up, log in or delete their account. The tests include failure scenarios such as when a user types in the incorrect password or when an account with the same email address already exists.

5.2.2 User Testing

User testing is carried out to obtain user feedback on the web application. My supervisor and I tested the application regular during our weekly meetings as new features were implemented. Initially, user testing was conducted in a group supervisor meeting where they reviewed the website and filled out a survey that included feedback on the user interface and how the recommendations were produced. A few key changes made during this session included adding a forgot password button, allowing the user to delete their account, and changing the length of the username and password textboxes. I implemented these initial changes before sending a survey to friends and family.

During UAT, the system should be validated in a production-like environment, providing a clear demonstration of what the product would look like once released. To carry out further end-users testing, I had to deploy my website temporarily. Many problems arose during deployment because many free hosting websites, such as Netlify or Heroku [37], did not permit the deployment of multiple frameworks. Nonetheless, I was able to provisionally deploy the website on Google Cloud to test the application. I created a survey shown in Appendix B and distributed it to around 20 people of various ages. The survey includes questions about the user interface as well as the application's functionalities. I wanted to know whether a user who has no idea what this application is about could use it without any background knowledge. I received a lot of positive feedback, including several user-requested changes. My survey respondents were mainly aged 18 to 25, with 15% under the age of 18, 50% between the ages of 18 and 21, 20% between the ages of 22 and 25, and the rest aged 25 and above. Approximately 10% of the testers were perplexed as to how to access the login page, while 68 percent scored the user interface a 9 or 10 out of 10 as seen in figure 15. At the end of the survey, I asked if they planned to use this app again in the future. Figure 16 shows that almost 90% of users said yes, which is encouraging. I will be further discussing the user feedback in the evaluation section of the report.

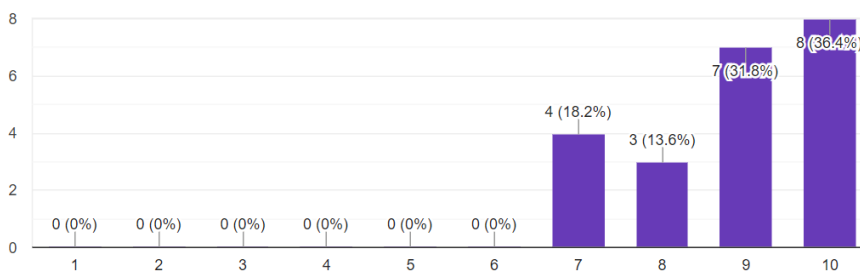


Figure 15 : Overall UI rating

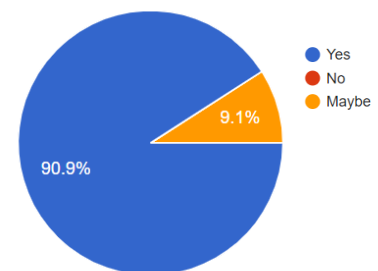


Figure 16 : Would you use this website in the future?

6 Evaluation

This section seeks to evaluate the product in two ways. Firstly, by looking at the functional and non-functional requirements stated in the literature review. Secondly, by reviewing the user feedback attained by user testing. To conclude the evaluation, I will compare and assess my application to the existing applications mentioned in the literature review. The success of this project centres around the achievement of the requirements set and the overall feedback given by users.

6.1 Requirements Evaluation

The primary aim of this project was to build a fully functioning website that achieved the functional and non-functional requirements set, while also becoming more confident in my web development skills. Prior to this project, I had never used any framework, so I decided to challenge myself to learn more about web development tools and machine learning algorithms. All the functional and non-functional requirements were met and extended throughout my project.

The functional requirements were achieved as follows. To use the application, a user must first sign up with a real, verified email address and a password. This uses Firebase Authentication to hash account passwords internally and stores the hashed variants in the database. It ensures secure storage of user information, including encrypted passwords. The user can then rate movies on a scale of 1 to 5, with the results stored in the Cloud Firestore Database. They can search for a movie on the main page, and the results are shown almost instantly with a list of movie images. Clicking on the 'view details' link on the hovering element of a movie directs the user to a page that displays additional information about the film, such as cast, synopsis, and genres. The movies are further divided into genres and can be found in the header genre dropdown list, enabling users to browse for movies based on a particular genre. Finally, on the profile tab, a user can add a friend, and if the friend approves the request, they can view each other's recommendations.

While carrying out my project, a few additional functional requirements were added to the list as new features were suggested by users and incorporated. Some additional improvements include a settings page where the user can update their password if necessary, the creation of charts that display the distribution of ratings across various attributes, and the addition of a forgot password button. Further features along with the final requirement of producing reliable recommendations will be discussed in the section evaluating the testing results.

For my project to be deemed a success, the non-functional requirements must also be satisfied. One of the requirements included the scalability of the application to add new features in the future. Since this project was developed using ReactJS, adding new components to the frontend is as simple as adding another page and routing it to the correct endpoint. React-router enables quick routing between pages, making it simple to raise the load on endpoints. Furthermore, the generated recommendations are tailored to a user's profile - when the user continuously rates new movies, they are added to the database and used to generate recommendations. These recommendations are generated instantly when the 'Recommended Movies' tab is clicked. Refreshing the recommendations page several times displays new movies, enabling a wider range of movies to be shown to the user, satisfying yet another requirement. Overall, the application was kept simple with an 'easy to use' interface as stated by end-users.

6.2 User Testing Results Evaluation

As described in section 5.2.2, the application was deployed temporarily, and a survey was given to roughly 20 users of different age groups to test the website. The feedback received was predominantly positive with a few possible improvements that could have been made to enhance the application. This section gathers the results from user testing and evaluates the feedback against the user interface and functionality of the application.

6.2.1 User Interface (UI) Feedback

In the survey, the first 14 questions were about the user interface. According to feedback, the most popular UI feature was the carousel with the hovering effect because it allowed for a 'clear and organised interface', making it 'easy to read'. Another page that users rated highly was the movie details page, which received a rating of 10/10 from more than 55 percent of users. Users admired the webpage featuring charts illustrating the distribution of user ratings because it helped them to see how their ratings ranged across various attributes. Nonetheless, one user stated that they would like to see 'ratings distributed across genres' so that they could determine the genres they preferred. This was an implementation change I decided to carry out. I added a chart that queries the database for an array of genres for each movie they've rated, calculates the percentage of each genre, and shows the results as a pie chart.

Another feature suggested by users was to add a ‘link to the movie on Netflix or Prime to stream’. This is not possible with the TMDB API; however, I was able to query the API to obtain a list of streaming providers by country and show their icons on the movie details page. This informs users about the availability of a given movie. While the query does not return a direct link to the movie, it does return a TMDB URL containing the actual deep links to the content. When any of the icons is pressed, the user is redirected to the TMDB URL, which contains links to the actual location where they can watch the movie.

The colour scheme of the website was disputed between the users as displayed in figure 17. It received a mixed range of feedback. Some users praised the colour relationship between the header and the rest of the website, while others questioned whether it was appropriate for the application. Nonetheless, I believe it should be left to individual preference, so a potential addition might be to encourage users to choose their own colour scheme or include a light/dark mode on the settings tab, allowing users more flexibility with the UI. Furthermore, some users had difficulty navigating from the home page to the login page, so I had to make it more evident. Moreover, a user suggested that more pictures and text detailing the website should be added to the home page so that visitors are more knowledgeable of the application's features. This was yet another change I decided to make to the implementation.

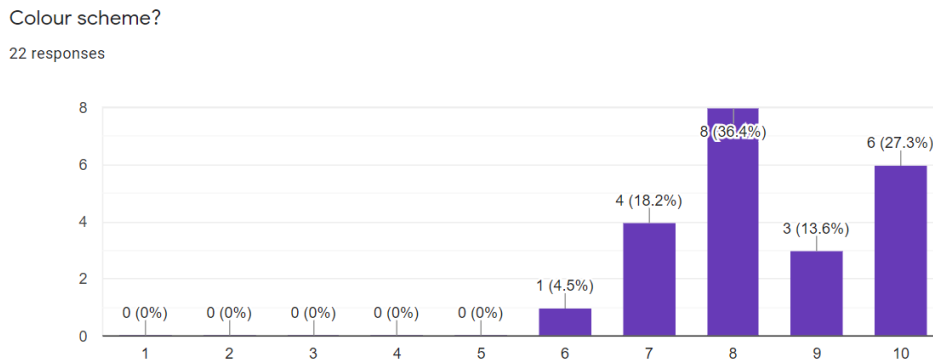


Figure 17: User ratings for the colour scheme

Overall, the UI received positive reviews, with one user stating that ‘it's close to that of other popular streaming services such as Netflix,’ making it simple to use and adapt to. Any user-provided improvements that could be applied under the time constraints were introduced to my project.

6.2.2 Functionality Feedback

My survey's final nine questions were about the website's functionality. All survey participants were able to update their password, search for a movie, navigate to the movie details page to see the trailer, add a movie to watch later tab, and get movie recommendations. All of the website's basic functionality seems to have worked well without creating any problems for the user. Many users appreciated the friend's functionality because it ‘makes it more interactive’ and improves the likelihood of people visiting the website because many ‘want to see what their friends' recommendations are’. However, it was proposed that a privacy button be implemented so that the user can shield their recommendations from other people if they do not want others to see their recommendations. I was unable to enforce this feature due to time constraints. However, this may be implemented in the future if the application is developed further.

Furthermore, another functional requirement was to produce reliable recommendations. This was fulfilled because the users were extremely pleased with the recommendations they received. According to one user, the website created ‘strong recommendations based on what I rated’. However, another person discovered a flaw in which movies they had previously rated were showing as suggestions. To resolve this issue, I queried the database for all movie Ids they had rated and removed them in the backend so they would no longer appear on the recommendations page. This ensures that only new movies that the user has not seen or rated are shown to them. Another notable suggestion was to have ‘worded feedback’ in which the user would write reviews for

movies. This could be a method of developing a more accurate model for generating recommendations, or it could simply be a means for users to see what other people think of the movies. In either case, as a future implementation, I could use sentiment analysis to categorise the movies based on their reviews, making for more accurate user recommendations.

In general, the results seem promising and useful, as they allowed me to gain insight into what end-users might like to see on the application and whether or not they liked it. Based on the testing results, I think this project was an overall success because, as previously stated, more than 90% of users enjoyed the website and will use it again in the future.

6.3 Comparison with other similar apps

In this section, I will be comparing the web application I implemented to the applications described in the literature review.

6.3.1 MovieLens

To begin, my web application functions similarly to MovieLens in that it allows users to rate movies, search for movies, and view movies based on their genres. The user interface, on the other hand, is entirely different. MovieLens colour scheme is orange and white, while my application is blue-green and white. Furthermore, MovieLens lacks an 'adding friend' function, which users claim is an 'innovative feature' that improves the application's 'social aspect.' The hovering feature was also unique to my application, and the ability to find movie streaming services enhances the likelihood of users using the application.

Nonetheless, a feature that would be beneficial to implement in the future is a personalized movie rating prediction. MovieLens provides this functionality because it already predicts the user's rating of a movie. End users would benefit greatly from this and it would make it easier for them to find movies that they would enjoy because those movies would be predicted a 5/5.

6.3.2 Taste App

My application is very different from the Taste app. This is predominantly because this is a mobile app that has different facilities compared to a website. This application is very unique in that it has a sliding left or right feature to discard or save the movie, respectively. This user interface is well-designed for a mobile app. A website, on the other hand, would have been impractical with this UI. As a result, I preferred a more simple, user-friendly interface in which movies are easily found and the user can navigate without difficulty. The Taste app also displays tv shows which could be an addition to my application in the future. Furthermore, when using the app, it appears that the match percentage, which indicates how likely you are to like the movie, is quite unreliable. My application's recommendations seem to be better and more accurate than the Taste App's. Finally, another aspect of the Taste app that I believe could have been a successful addition to my application is its division of movies into streaming providers. This is since, in recent years, websites such as Netflix and Prime have grown in popularity, and more people continue to find movies from those subscription services.

7 Project Critical Assessment

Each stage of my project, design, implementation, and testing was completed successfully. Using Rapid Application Development as a software development methodology was appropriate for my project as it prioritizes rapid prototype releases and iterations. Implementing small changes or adding a new feature then quickly testing it out was the best way I could have chosen to approach my project. However, with the lack of experience in machine learning and web development, I was not able to estimate the time taken to build the application. A better grasp of time frames would have made it easier to plan out my project.

The implementation was divided into two parts: the model and the web application. The web application was further divided into frontend and backend. This division was necessary and beneficial during development because it helped me to understand and learn one component at a time without being overwhelmed. Separating

the frontend and backend into separate frameworks allowed me to work on them separately and integrate them when necessary. The advantages of this include modularity and easier testing. Keeping the frontend separate from the backend made it easier to work on one module while keeping the other untouched. Furthermore, having two frameworks meant that testing for each framework was completely separate, which made it easy to find bugs because any conflicts that arose were either part of the frontend or the backend. Nonetheless, using two frameworks posed an issue while attempting to deploy my application, and I believe that using one (Django) could have resulted in a better code setup. It would have also made scaling much easier because, in the future, new developers can easily adapt to a single framework rather than two.

As I mentioned previously, I encountered a few problems when attempting to temporarily deploy my application for testing purposes. This was mainly due to the size of the pickle file generated for the classifier. It was too large to commit to GitHub, and many of these hosting platforms used GitHub repositories to deploy applications. This was resolved by deploying my application to Google Cloud via the command line. Nonetheless, I had to upload my pickle file to GitHub as a Zipped document. As a result, if the application is being replicated on a local computer, the file must first be unzipped into the appropriate folder.

Furthermore, due to a lack of Django knowledge prior to this project, I was unable to take advantage of Django's Model-View-Template architecture. The model manages the data, the view displays the data to the user through an interface and the templates interpret user inputs and communicate with the model [38]. I made good use of Django's admin functionality but was unable to use the models for development. This was due to the fact that this project is primarily a frontend application, which meant that models were not required for the development.

If I were to repeat my project, I would meticulously plan it out, with strict time frames, to guarantee a completed product within the time constraint. In addition, I would start end-user testing earlier in the development process so that I have more feedback to work from when trying to improve the application. Moreover, I would use only one framework because it would result in better construction and set up of the code while also making it easier to deploy the application. Additionally, this would allow me to take advantage of Django features such as the MVT architecture.

On reflection, I believe my project was a complete success. The regular supervisor meetings and group meetings kept me on top of my work because I would set a target each week and make sure I accomplished it by the time I met with my supervisor. My project achieves its primary goal of building a web application that recommends movies to the user. I also believe it was a success due to the user feedback received. It was mainly positive, and many were amazed at how creative the website was. Although I had some technical problems with setting up the frameworks and GitHub, I was able to resiliently fix them and move ahead with my project without it stopping me for too long. It was a great learning experience and I feel more confident in creating something from scratch.

7.1 Future work

Throughout the paper, I have mentioned some improvements I would like to make in the future. In summary, as a direct result of user feedback, these involve modifying the colour scheme, incorporating worded reviews, introducing a privacy setting, and a predictive rating of movies personalised to the user. Predictive rating may be accomplished by using a model-based technique to generate ratings based on the user profile. A model which could have been used is the Matrix Factorization model which characterizes users and movies by their feature vectors of low dimension inferred from the rating patterns of users. In addition, to make the website more functional, I will increase the number of friends that can be added in the future. Currently, a user may only add 5 friends; however, increasing this number and making it more user-friendly can result in more visitors to the website. Furthermore, if I had more time to complete my project, I would like to create a more accurate model. According to the literature review, a hybrid approach may have resulted in stronger recommendations. They overcome the limitations of neighbourhood-based methods in terms of space and scalability [39]. This can be achieved by implementing a content-based system and merging it with the existing system. The content-based

system would balance the recommendations for less popular movies because it categorises them based on features like genres rather than other similar users.

Another way to enhance the model that was generated could be by adding to the dataset that I have used. Currently, the KNN model was built using a fixed dataset from Grouplens. Once the web application has users, I will be able to use their information to refine the model in the future. This can be accomplished by appending user ratings of movies to the ratings.csv file obtained from Grouplens. Through doing so, newer movies would be added to the dataset, providing more data for a more accurate model. This could be implemented with the code seen in figure 18.

```
# This is called if 'isUpdate' is set to true.
# It adds the rating to the ratings csv file
def update_dataset(self, data):
    movieId = self.get_movie_index(data[1])
    with open(self.path_ratings_src, 'a') as fd:
        fd.write(str(data[0]) + 'A') + ','
            + str(movieId) + ','
            + str(data[2]) + ','
            + str(time()) + '\n')
```

Figure 18: Code for updating the ratings.csv file

8 Conclusion

In conclusion, the objective of this project has been to create a fully functional web application that generates recommendations of movies to users based on explicit feedback obtained from users through ratings. The literature review summary has introduced the fundamentals of machine learning techniques used in recommender engines and other similar applications that exist within the field. In addition, the summary included the functional and non-functional requirements that are taken into consideration when evaluating the product. Functional requirements consist of the features of the application while non-functional requirements consist of criteria for the operation of the website.

This aim has been achieved through the software development lifecycle phases of Design, Implementation and Testing. After deliberating amongst the various techniques, I decided to implement an item-based collaborative filtering system with a KNN algorithm that calculates the similarity between movies using cosine measurement. This was selected because of its simplicity and efficiency in producing reliable recommendations. The design divided my project into two components: the web application and the KNN classifier. The application's frontend was developed with ReactJS, and the backend was built with the Django Rest framework. The website took advantage of the easy integration between the two frameworks. The database chosen to store user information was Cloud Firestore. The classifier was built using the well-known Grouplens dataset of movie and rating data. The RAD approach was used to carry out the implementation, which was again divided into two parts. Each feature of the web application was built and tested concurrently, allowing vulnerabilities to be identified early and prevented from propagating to the next stages.

The complete product was evaluated through the various unit and user testing. The success of this project depended on the achievement of the functional and non-functional requirements set and the feedback obtained from user testing. All requirements were met and described in the evaluation section of this report. The results of user testing revealed that people were optimistic about the website, with many stating that it was 'very impressive' and had 'good functionality'. Some enhancements were also reviewed in the evaluation section, and those that could be introduced within the time frame were implemented. Other improvements, such as incorporating a privacy setting or a predictive rating of movies, were addressed in the section on future work.

Overall, this project has been successful in building a web application with a visually appealing user interface that generates reliable recommendations. This project has unquestionably strengthened my coding abilities and broadened my understanding of machine learning. It provided me with the opportunity to undertake work that is not far from the new frontiers of computer science. It also has the potential for future work, which may provide an interesting challenge given the growing demand for recommender engines.

9 Bibliography

- [1] "How much data is on the internet? | The Big Data Facts Update 2020", *NodeGraph*, 2021. [Online]. Available: <https://www.nodegraph.se/how-much-data-is-on-the-internet/>. [Accessed: 08-April-2021].
- [2] P. Melville and V. Sindhvani, "Recommender Systems", IBM T.J. Watson Research Center.
- [3] G. Rodríguez, "Introduction to Recommender Systems", Tyrolab, 2019.
- [4] J. Bobadilla, F. Ortega, A. Hernando and A. Gutiérrez, "Recommender systems survey", *Knowledge-Based Systems*, vol. 46, pp. 109-132, 2013. Available: 10.1016/j.knosys.2013.03.012.
- [5] B. Kumar and N. Sharma, "Approaches, Issues and Challenges in Recommender Systems: A Systematic Review", *Indian Journal of Science and Technology*, vol. 9, no. 47, 2016. Available: 10.17485/ijst/2015/v8i1/94892.
- [6] Shyong K, Frankowski D, Riedl J. Do you trust your recommendations? An exploration of security and privacy issues in recommender systems. *Emerging Trends in Information and Communication Security*. Springer Berlin Heidelberg; 2006. p. 14–29.
- [7] K. Funakoshi and T. Ohguro, "A content-based collaborative recommender system with detailed use of evaluations", *Fourth International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies*. Vol. 1, pp. 253-256. 2000
- [8] P. Melville and V. Sindhvani, "Recommender Systems", IBM T.J. Watson Research Center
- [9] R. Meteren. "Using Content-Based Filtering for Recommendation." 2000.
- [10] M. Pazzani and D. Billsus, "Learning and Revising User Profiles: The Identification of Interesting Web Sites", pp. 313–331. 1997
- [11] B. Cui, "Design and Implementation of Movie Recommendation System Based on Knn Collaborative Filtering Algorithm", *ITM Web of Conferences*, vol. 12, p. 04008, 2017
- [12] G. Linden, B. Smith and J. York, "Amazon.com recommendations: item-to-item collaborative filtering," in *IEEE Internet Computing*, vol. 7, no. 1, pp. 76-80, Jan.-Feb. 2003
- [13] Y. Koren, R. Bell and C. Volinsky, "Matrix Factorization Techniques for Recommender Systems", *Computer*, vol. 42, no. 8, pp. 30-37, 2009.
- [14] J. Bennett and S. Lanning, "The Netflix Prize", *KDD Cup and Workshop*. 2007.
- [15] "Taste - Movie & TV recommendations based on your taste.", *Taste.io*, 2021. [Online]. Available: <https://www.taste.io/>. [Accessed: 12-April-2021].
- [16] B. Sarwar, G. Karypis, J. Konstan and J. Reidl, "Item-based collaborative filtering recommendation algorithms", *Proceedings of the tenth international conference on World Wide Web - WWW '01*, 2001. Available: 10.1145/371920.372071
- [17] "MovieLens", GroupLens, 2020. [Online]. Available: <https://grouplens.org/datasets/movielens/> [Accessed: 15-April-2021].
- [18] "Manifesto for Agile Software Development", *Agilemanifesto.org*, 2021. [Online]. Available: <https://agilemanifesto.org/>. [Accessed: 15-April-2021].
- [19] N. Ruparelia, "Software development lifecycle models", *ACM SIGSOFT Software Engineering Notes*, vol. 35, no. 3, pp. 8-13, 2010.

- [20] P. Beynon-Davies, C. Carne, H. Mackay and D. Tudhope, "Rapid application development (RAD): an empirical review", *European Journal of Information Systems*, vol. 8, no. 3, pp. 211-223, 1999
- [21] "sklearn.neighbors.NearestNeighbors — scikit-learn 0.24.2 documentation", *Scikit-learn.org*, 2021. [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestNeighbors.html>. [Accessed: 19-April-2021].
- [22] A. Kumar and R. Singh, "Comparative analysis of angularjs and reactjs", *International Journal of Latest Trends in Engineering and Technology*, Vol. 7, Issue 4, pp. 225-227.
- [23] S. Aggarwal, "Modern Web-Development using ReactJS", *International Journal of Recent Research Aspects* ISSN: 2349-7688, Vol. 5, Issue 1, pp. 133-137. March 2018
- [24] J. Forcier, P. Bissex and W. Chun, "Python Web development with Django." Upper Saddle River, NJ: Addison-Wesley, 2009.
- [25] S. Dauzon, A. Bendoraitis and A. Ravindran, *Django: Web Development with Python*. 2016.
- [26] "Cloud Firestore vs the Realtime Database: Which one do I use?", *The Firebase Blog*, 2021. [Online]. Available: <https://firebase.googleblog.com/2017/10/cloud-firestore-for-rtdb-developers.html>. [Accessed: 19-April-2021].
- [27] "API Docs", *Developers.themoviedb.org*, 2020. [Online]. Available: <https://developers.themoviedb.org/3> [Accessed: 19-April-2021].
- [28] "The Django admin site | Django documentation | Django", *Docs.djangoproject.com*, 2021. [Online]. Available: <https://docs.djangoproject.com/en/3.1/ref/contrib/admin/>. [Accessed: 19-April-2021].
- [29] "Writing views | Django documentation | Django", *Docs.djangoproject.com*, 2021. [Online]. Available: <https://docs.djangoproject.com/en/3.1/topics/http/views/>. [Accessed: 19-April-2021].
- [30] T. Christie, "Authentication - Django REST framework", *Django-rest-framework.org*, 2021. [Online]. Available: <https://www.django-rest-framework.org/api-guide/authentication/>. [Accessed: 19-April-2021].
- [31] "React.Component – React", *Reactjs.org*, 2021. [Online]. Available: <https://reactjs.org/docs/react-component.html>. [Accessed: 19-April-2021].
- [32] M. Bahadourpour, N. Behzad, N.S. Mohammad H. Determining optimal number of neighbors in item-based kNN collaborative filtering algorithm for learning preferences of new users. *Journal of Telecommunication*, pp. 163-167, 2017.
- [33] L. Luo, "Software Testing Techniques", *Software Testing and Continuous Quality Improvement*, pp. 525-591, 2000.
- [34] "What are the main problems facing in User Acceptance Testing (UAT) and its solutions?", *Software Testing Class*, 2015. [Online]. Available: <https://www.softwaretestingclass.com/what-are-the-main-problems-facing-in-user-acceptance-testing-uat-and-its-solutions/> [Accessed: 21-April-2021].
- [35] J. Bennett, *Practical django projects*. Berkeley, CA: Apress, 2009.
- [36] [13]"Testing tools | Django documentation | Django", *Docs.djangoproject.com*, 2021. [Online]. Available: <https://docs.djangoproject.com/en/3.2/topics/testing/tools/>. [Accessed: 21-April-2021].
- [37] "Cloud Application Platform | Heroku", *Heroku.com*, 2021. [Online]. Available: <https://www.heroku.com/>. [Accessed: 21-April-2021].


[38] <https://ibit.temple.edu/wp-content/uploads/2011/03/IBITWebframeworks.pdf> - Evaluating web development frameworks: Django, Ruby on Rails and CakePHP


[39] R. Sharma, D. Gopalani and Y. Meena, "Collaborative filtering-based recommender system: Approaches and research challenges," *2017 3rd International Conference on Computational Intelligence & Communication Technology (CICT)*, 2017, pp. 1-6, doi: 10.1109/CICT.2017.7977363.


10 Appendix


A. DEMO EMAIL BY FIREBASE SENT TO USERS FOR AUTHENTICATION

movie-recommender ▾ Authentication


Go to docs 

 Password reset

 Email address change

 SMTP settings


SMS

 SMS verification

Sender name

not provided

From

noreply@movie-recommender-49dcc.firebaseio.com 

Reply to

noreply

Subject

Reset your password for %APP_NAME%

Message

Hello,

Follow this link to reset your %APP_NAME% password for your %EMAIL% account.

https://movie-recommender-49dcc.firebaseio.com/_/auth/action?mode=action&oobCode=code

If you didn't ask to reset your password, you can ignore this email.

Thanks,

Your %APP_NAME% team

B. SURVEY SENT FOR USER TESTING

PART 1

What's your age group?

- ☐ Under 18
- ☐ 18 - 21
- ☐ 22 - 25
- ☐ 26 - 31
- ☐ 32+

Login/Sign up page

1 2 3 4 5 6 7 8 9 10

Don't like it ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ Like it

Login/Sign up page - how can it be improves?

Your answer

Movie carousel on main page

1 2 3 4 5 6 7 8 9 10

Don't like it ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ Like it

Movie carousel on main page - feedback

Your answer

Search movie layout

1 2 3 4 5 6 7 8 9 10

Don't like it ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ Like it

PART 2

Profile ratings data layout - written feedback - what other charts would you like to see? How was the layout?

Your answer

Hovering on movie image to get their details

- ☐ Liked it!
- ☐ Could be better
- ☐ Hated it!

If for hovering, you put 'could be better' or 'hated it', why?

Your answer

Movie details page - when clicked on the movie details link when you hover on a movie

1 2 3 4 5 6 7 8 9 10

Don't like it ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ Like it

Colour scheme?

1 2 3 4 5 6 7 8 9 10

Don't like it ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ Like it

Overall UI? What did you think of it?

1 2 3 4 5 6 7 8 9 10

Don't like it ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ Like it

PART 3

Did the movies recommended to you seem reasonable? - Were they similar to those you've rated?

- ☐ Yes
- ☐ No

Did your watch later movies show up in watch later tab?

- ☐ Yes
- ☐ No

What did you think about being able to see your friends recommendations? - you can add "123456@live.com" as a friend and try it out

Your answer _____

Were you able to change your password?

- ☐ Yes
- ☐ No

Searching for a movie - found what you're looking for?

- ☐ Yes
- ☐ No

Were you able to watch the trailer in the movie details page?

- ☐ Yes
- ☐ No

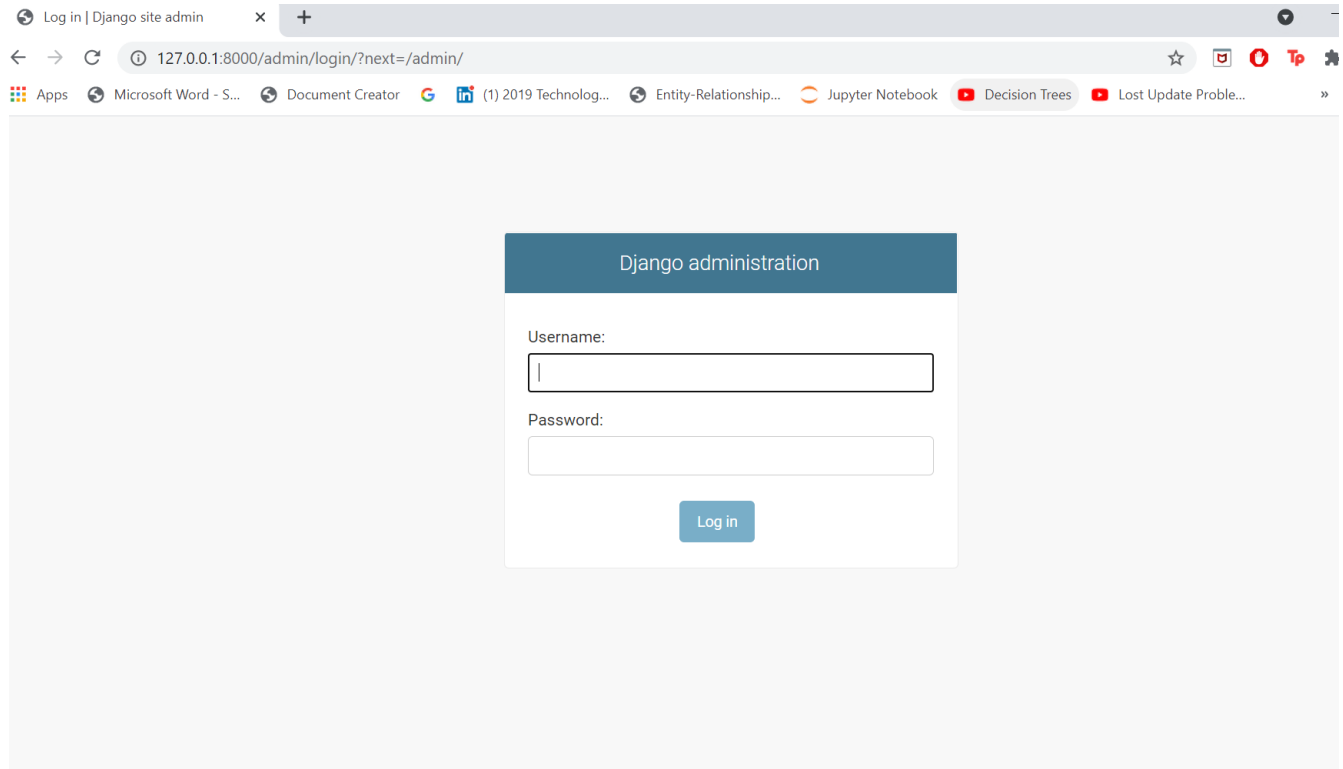
Overall what did you think of the functionality? - Includes: providing you recommendations, showing you charts on your ratings, allowing you to rate and add to watch later etc What other charts would you want to see?

Your answer

Would you use this type of website to find movies in the future?

- ☐ Yes
- ☐ No
- ☐ Maybe

C. ADMIN PAGE



The image shows a web browser window with the Django administration login page. The browser's address bar displays the URL `127.0.0.1:8000/admin/login/?next=/admin/`. The page features a central login form with a dark blue header bar containing the text "Django administration". Below this, the form has two input fields: "Username:" and "Password:". A blue "Log in" button is positioned at the bottom of the form. The browser's taskbar at the top shows several open applications, including "Log in | Django site admin", "Microsoft Word - S...", "Document Creator", "(1) 2019 Technolog...", "Entity-Relationship...", "Jupyter Notebook", "Decision Trees", and "Lost Update Proble...".

Log in | Django site admin

127.0.0.1:8000/admin/login/?next=/admin/

Apps Microsoft Word - S... Document Creator (1) 2019 Technolog... Entity-Relationship... Jupyter Notebook Decision Trees Lost Update Proble...

Django administration

Username:

Password:

Log in

D. TOKENS WITHIN THE ADMIN PAGE

Django administration

WELCOME, **NIHAR**. [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

Home > Auth Token > Tokens

AUTH TOKEN

Tokens + Add

AUTHENTICATION AND AUTHORIZATION

Groups + Add

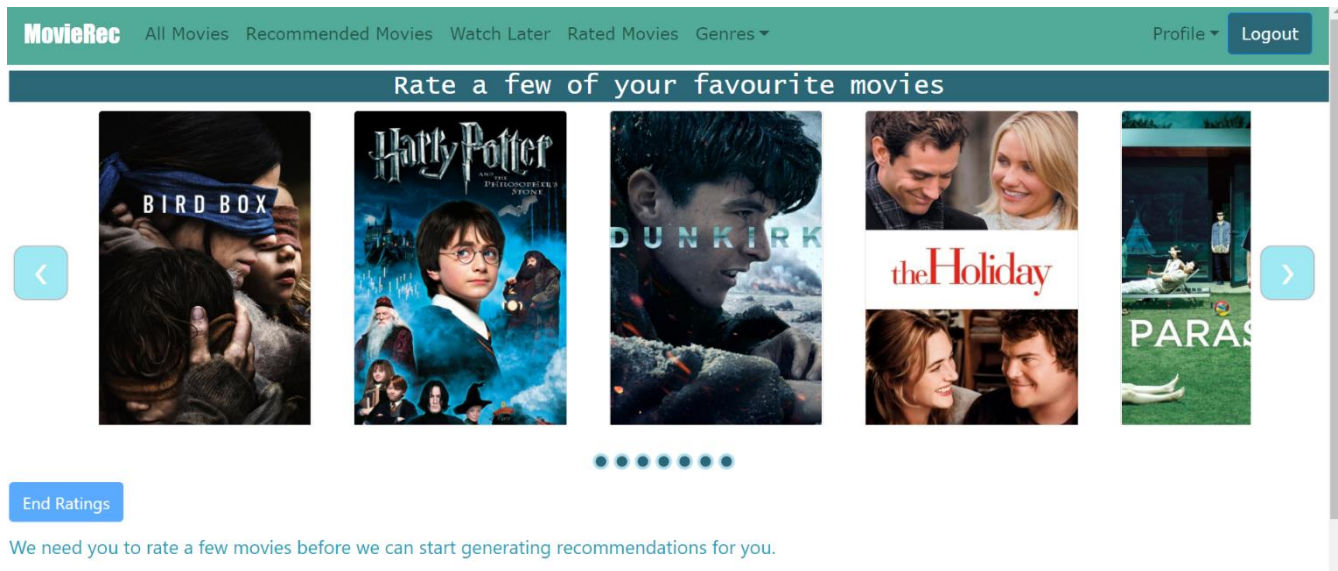
Users + Add

Select token to change

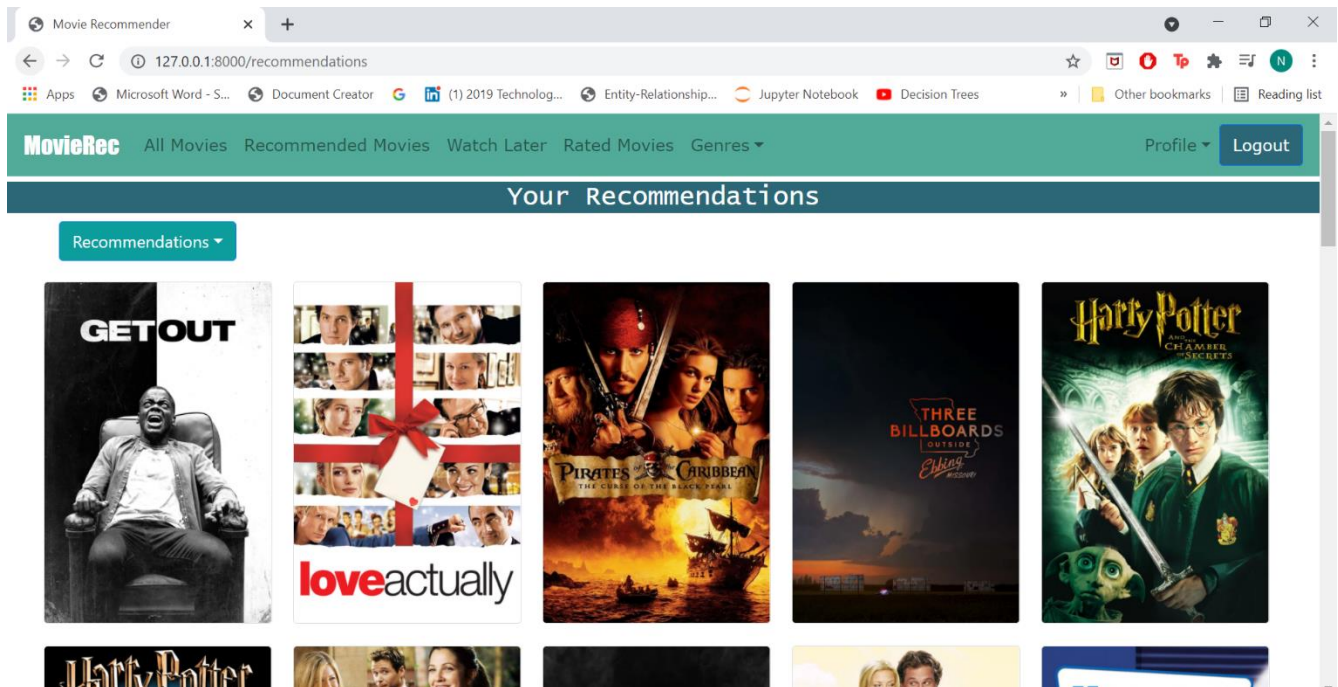
ADD TOKEN +

KEY	USER	CREATED
0d5e2e47de039bab0392e7abc3ce558e0d5bd554	nihareddy31@live.co.uk	April 23, 2021, 1:38 p.m.
aa35fe68973e46d06005fee596598f8a281fb3f8	654321@live.com	April 23, 2021, 1:21 p.m.
b08a5ea162894c2eeaeffa6f819893229b0a00af	ng342@exeter.ac.uk	April 20, 2021, 4:40 p.m.
b52851b33dbf4b067b5586ec1daa91763a2ef52a	niha1	April 9, 2021, 11:53 a.m.

E. PICTURES OF THE USER INTERFACE

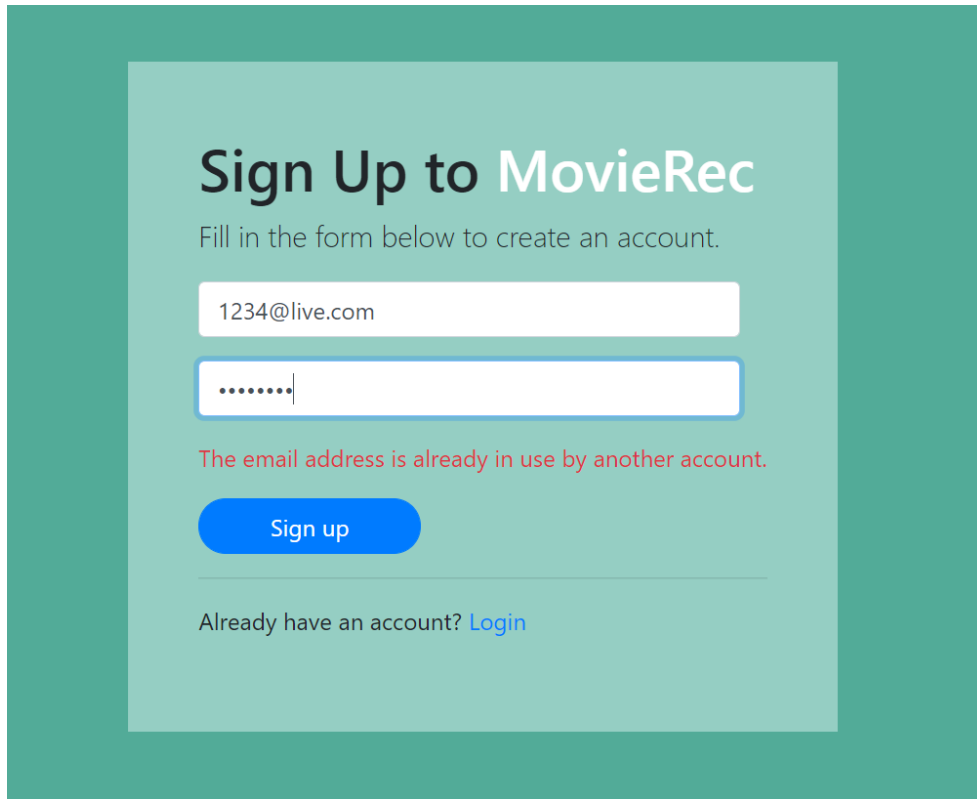


Preference Elicitation Page



Recommendations Page

Wrong input in the sign up page. (Email already exists and short password)



Sign Up to MovieRec

Fill in the form below to create an account.

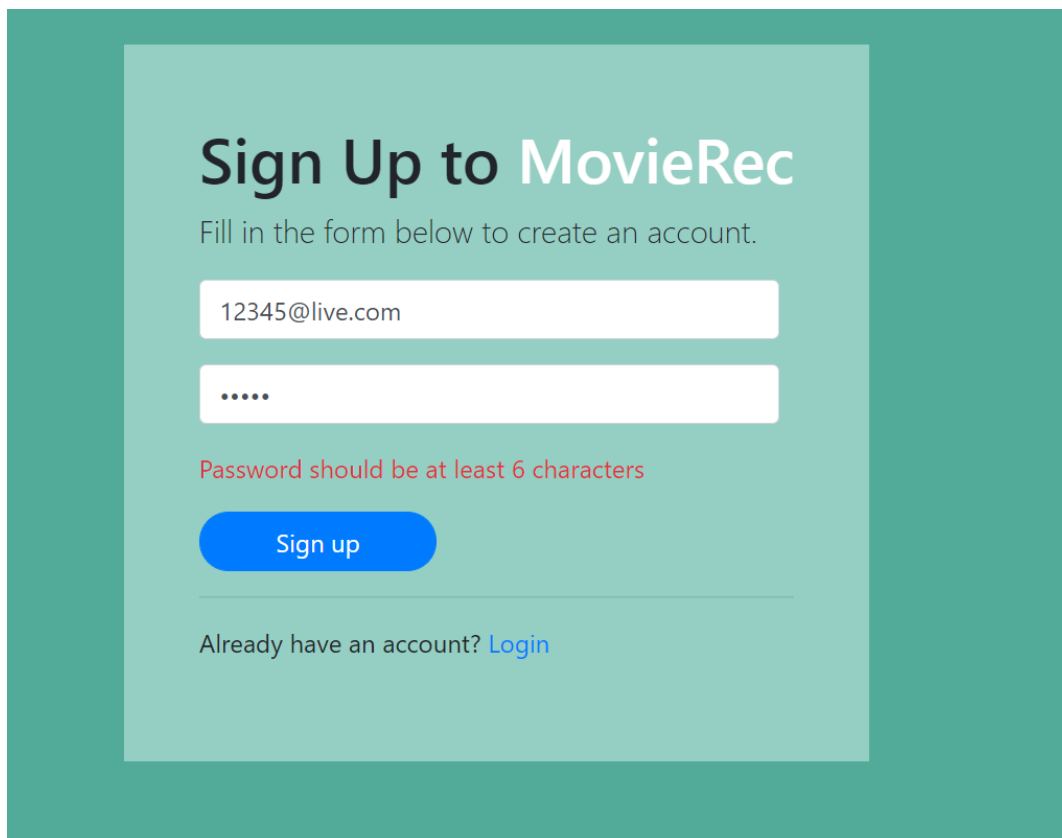
1234@live.com

.....

The email address is already in use by another account.

Sign up

Already have an account? [Login](#)



Sign Up to MovieRec

Fill in the form below to create an account.

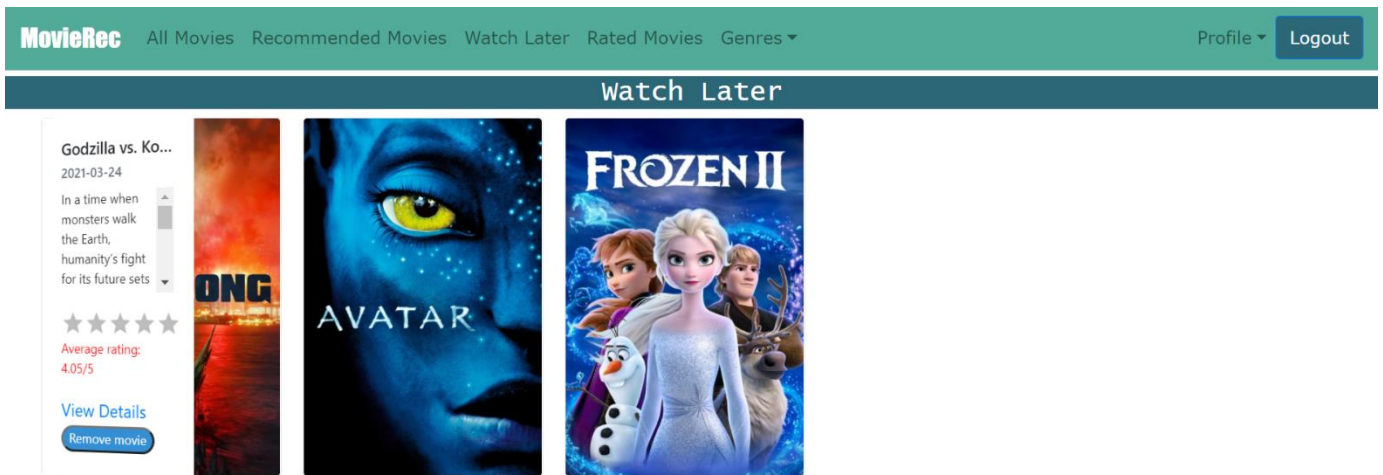
12345@live.com

.....

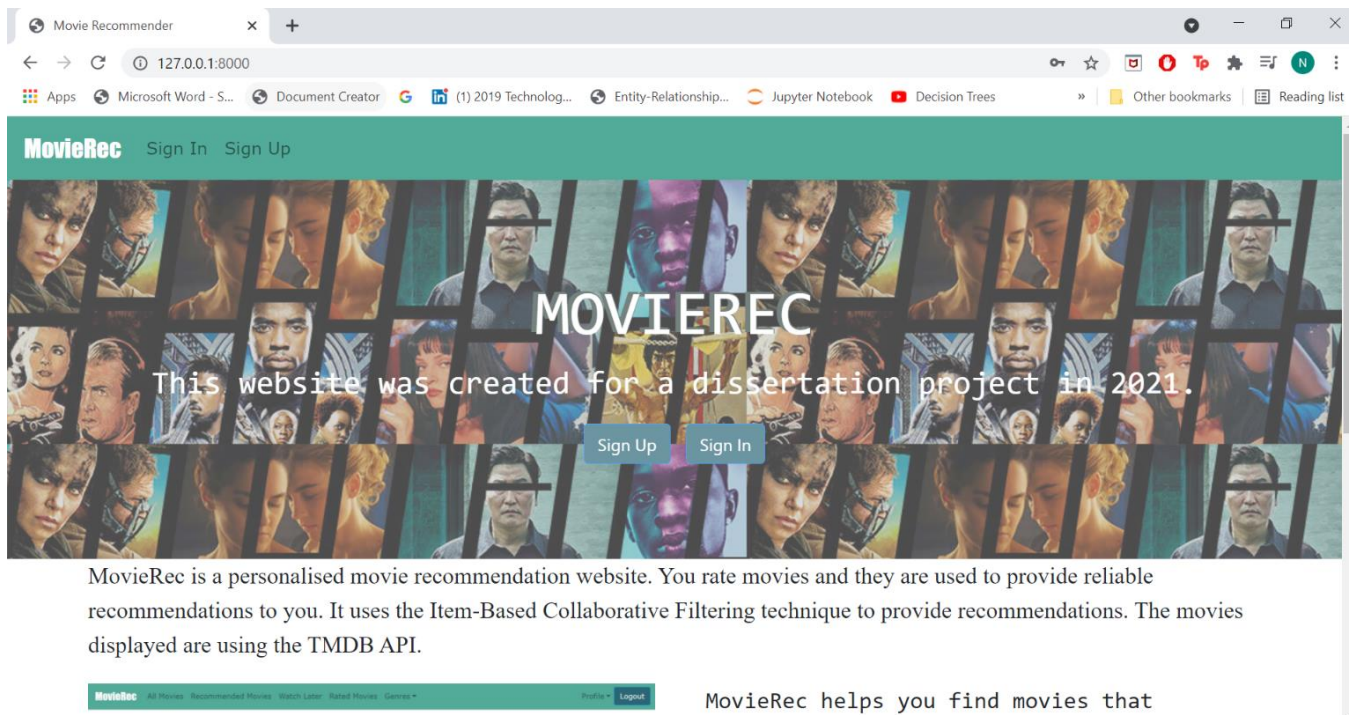
Password should be at least 6 characters

Sign up

Already have an account? [Login](#)



Watch later page with one movie showing the hovering feature



Home page that is visible to users that haven't logged in yet.