CIS 550 Course Project Report

Group 11: Yangfei Li, Haocheng Wu, Jiadi Xiong, Yinchuan Xu

1. Project Goal

The aim of this final project is to build a movie based social network -- MovieBook -- for movie fans. MovieBook has following features:

Movie Information: MovieBook provides detailed movie information, including directors, actors, duration, reviews, and ratings. Especially, MovieBook combines ratings from IMDB, Fandango, RottenTomatoes, and Metacritic, so that ratings on these four world's most popular movie sites can be easily obtained from MovieBook, instead of searching on each site.

Social Community: MovieBook enables users to easily evaluate movies by writing reviews. Besides, users on MovieBook can also interact with other users by the "Following" button. On one user's profile page, public personal information, movies that this user liked, scores that this user rated, and reviews that this user wrote are summarized, so that users can find friends having the same taste.

Personalized Recommendation System: MovieBook will recommend similar movies that one user has not watched based on movies that this user watched and liked.

2. Basic architecture

This is a full-stack website development project. We separated the system into three different layers, which are client side, server side and database. The client side will interact with users and send users' requests to the server side. Receiving the requests, server side will handle them and ask for data which are stored in database.

On the client side, to build a beautiful and interactive web page interface, we wrote HTML, CSS and JavaScript in a complex frontend framework, called Semantic-UI, and adopted the jQuery library. On the server side, we used Node.js and Express.js as the RESTful web API framework. We used MySQL and MongoDB to store the CSV and JSON format data respectively. We hosted the mySQL database on Amazon AWS and mongoDB on mLab.

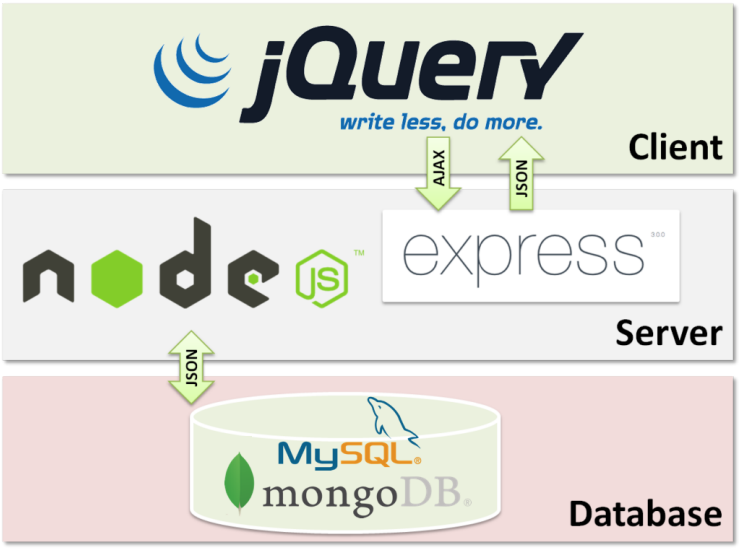


Fig. 1. Architecture of our application

3. Description of Datasets

The project uses four overlapping datasets from multi sources, where two of them are large datasets. These four datasets have two different formats: CSV and TXT. The summary of these four datasets is shown in Table 1.

	Dataset Name		Resource	File Format	Size	Important Variables
1	MovieLens Datasets	Links	GroupLens	CSV	100,000 ratings applied to 9,000 movies by 700 users	movie ID, IMDB ID
		Movies		CSV		movie ID, title, genres
		Ratings		CSV		user ID, movie ID, rating
2	Large Movie Review Dataset		Stanford University	TXT	12,500 reviews	IMDB link, review
3	Fandango Dataset		Kaggle	CSV	100 movies	title, ratings on IMDB, Fandango, RottenTomatoes, Metacritic
4	IMDB 5000 Movie Dataset		Data.world	CSV	5043 movies	IMDB link, duration, director's name, actors' name, country, publication year

Table 1. Summary of four datasets used in the project

3.1 MovieLens Datasets

MovieLens datasets are from MovieLens website (http://movielens.org), which contains 100,000 ratings applied to 9,000 movies by 700 users between January 09, 1995 and March 31, 2015. The dataset was generated on October 17, 2016, including three CSV files: links.csv, movies.csv, and ratings.csv.

This Dataset is our main dataset, because it contains IMDB ID, which is the key to connect all these four datasets, and movie titles. In addition, the dataset also contains users' information, which can be used to establish a primary users database. Meanwhile, users and their rating information can be used for training recommendation system.

3.2 Large Movie Review Dataset

Large Movie Review Dataset, which is from Stanford University (http://ai.stanford.edu/~amaas/data/sentiment/), contains 50,000 movie reviews in TXT format. Large Movie Review Dataset has three variables, movie ID, IMDB link, and review in IMDB. Movie ID is a localized ID for this dataset, so movie ID is eliminated during data integration.

3.3 Fandango Dataset

The third dataset is from Kaggle called Fandango Dataset (https://www.kaggle.com/jwilsey/pandas-exploration-fandango-dataset) in CSV format, containing a set of 100 movies with their ratings from four movie critics websites: IMDB, Fandango, Rotten Tomatoes, and Metacritic.

3.4 IMDB 5000 Movie Dataset

The fourth dataset is IMDB 5000 Movie Dataset from Data.world (https://data.world/popculture/imdb-5000-movie-dataset), containing 5043 movies. The dataset includes total 28 descriptive variables, including duration, director's name, actors' names, country, publication year, and IMDB link.

4. Data Cleaning and Integration

In relational part, these four datasets are cleaned, connected, and then separated into seven tables.

Although these four datasets are all about movies, it's still a hard task to connect them. Most important reason is that these four datasets use different keys: some datasets use IMDB ID as their key, while other use their own movie IDs as their key. After carefully observing these datasets, we come up with several solutions and use IMDB ID or movie titles to connect these four datasets.

Although Large Movie Review Dataset and IMDB 5000 Movie Dataset does not contain IMDB ID, they do contain IMDB URL. IMDB IDs are taken out from IMDB URLs by Python (*Fig.* 1). In this way, MovieLens Datasets, Large Movie Review Dataset, and IMDB 5000 Movie Dataset are connected by IMDB ID.

IMDB URL: http://www.imdb.com/title/tt0499549/?ref_=fn_tt_tt_1

↓

IMDB ID: 499549

Fig. 2. Extract IMDB ID from IMDB URL

Fandango Dataset does not contain any information about IMDB ID or IMDB URL. As a result, we choose to join this dataset with other datasets by movies' titles. However, some movies' names contain non-alphabetic character, such as Ã and Â, and some contain improper punctuation, such as 'Appl??e'. Before joining datasets by movie titles, these characters should be cleaned. So we use regular expression to clean these data and render them in a correct form before integrating.

In addition, we notice that the format of "Genres" column in Movies.csv for each movie is a list of genres separating by vertical bars. To solve the problem, we separated various genres for one movie into different rows in Python (*Fig.* 2).

title	genres
Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy

↓

title	genres
Toy Story (1995)	Adventure
Toy Story (1995)	Animation
Toy Story (1995)	Children
Toy Story (1995)	Comedy
Toy Story (1995)	Fantasy

Fig. 3. Revision of "Genres" column in movies.csv

All detailed steps about connecting these four datasets and separating to seven tables are explained below:

1. Movie table contains two columns, IMDB ID and movie title, which is obtained by inner joining Links.csv and Movies.csv on movieId.
2. Movie_rate table has six columns, IMDB ID, movie title, and ratings on IMDB, Fandango, Rotten Tomatoes and Metacritic websites, which is obtained by inner joining Movie.csv and Fandango Dataset on movie title.
3. Movie_desc table is achieved directly from IMDB 5000 Movie Dataset, where IMDB ID is obtained from IMDB URL.
4. Genres table comprises two columns, IMDB ID and genres. Information about movie genres are in movies.csv. Then, genres table is obtained by inner joining Links.csv and Movies.csv on movieId.
5. User table contains user ID, username, password and email. User ID is obtained from Ratings.csv.
6. User_like table has user ID, IMDB ID, and rating, which is achieved by inner joining Links.csv and Ratings.csv on movieId.
7. Following table comprises two columns -- user ID and user ID, used to store information that one user is following another one.

Next, we automatically create SQL queries using Python and imported these queries into MySQL Database.

In non relational part, we utilize Large Movie Review Dataset, where each review is in one TXT format file. We first read TXT files, which include IMDB ID (extracted from urls), user ID, and review content by pandas library and Json library in Python. Then, we transform data format and store them as JSON file in order to build Mongo Database.

5. Relational Schema and ER Diagram

Movie(imdbId, Title)

Movie_rate(imdbId, RottenTomatoes, Metacritic, IMDB, Fandango_Stars)

Movie_desc(imdbId, Title_Year, Country, Duration, Director_Name, Actor_1_Name, Actor_2_Name, Actor_3_Name, Movie_IMDB_Link)

Genres (imdbId, genres)

User(userId, email, User_name, Password)

User_like(userId, imdbId, rating)

Following(userId, userId)

Fig. 4. Relational schema of data

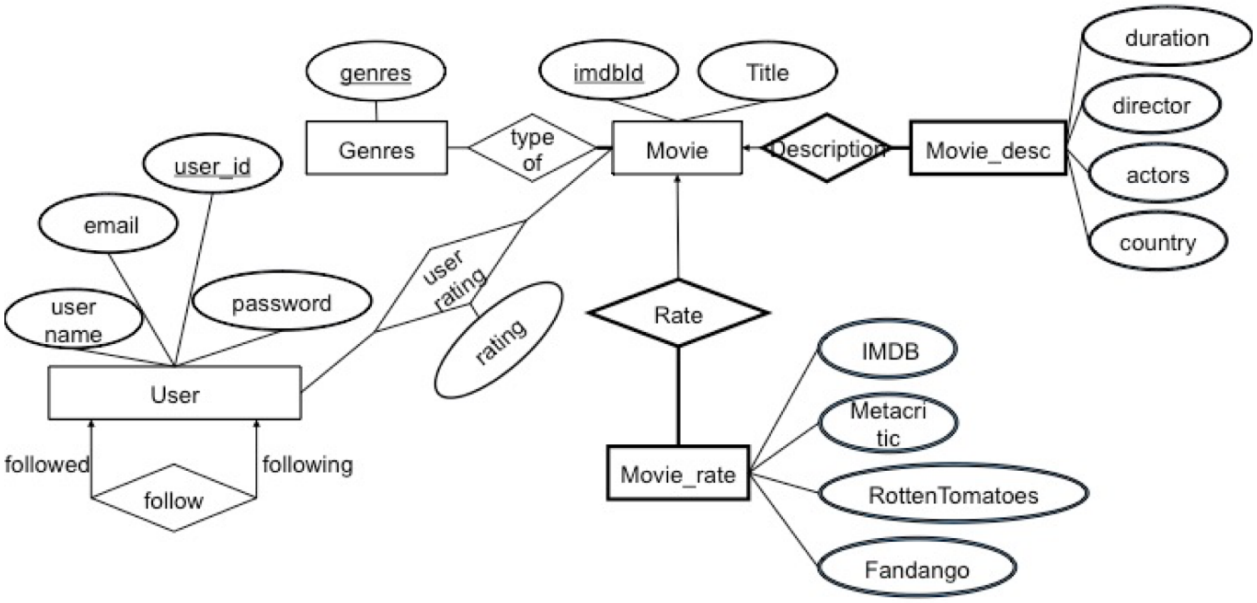


Fig. 5. ER diagram of data

6. Databases used and why

MySQL and MongoDB are used as our databases. MySQL is used for supporting SQL query and relational data storage. MongoDB is for NoSQL part.

Most data are stored in MySQL, including movie information (IMDB ID, movie title, publication year) and user information (user ID, username, following), because relational schema can properly handle these information and the schema will not change. Besides, MySQL owns internal indexing mechanism, which enables users to add more indices for looking up, deleting, updating as well as retrieving.

We store our user reviews in MongoDB. Since NoSQL provides a more flexible schema for this large scale text information. In other words, it's more convenient to change the schema of review, such as adding one attribute of review date.

7. Complex queries

Here are some examples of complex queries we used:

Find users who like all movies liked by one specific user:

```
SELECT DISTINCT u.userId
FROM user_like ul JOIN User u ON ul.User_ID = u.userId
WHERE ul.User_ID NOT IN (
        SELECT User_ID FROM user_like WHERE imdbId NOT IN
                (SELECT imdbId FROM user_like WHERE User_ID = "specific id")  AND ul.User_ID
                != "specific id")
```

Three movie genres mostly liked by one specific user:

```
SELECT g.genres
FROM user_like u INNER JOIN Genres g ON u.imdbId = g.imdbId
WHERE u.User_ID = 'specific user id'
ORDER BY u.rating DESC
LIMIT 3
```

Query for movies recommendation before login

```
(SELECT m.title, m.imdbId AS id, (mr.RottenTomatoes / 10 + mr.Metacritic /9.4 + mr.IMDB / 0.8 +
mr.Fandango_Stars / 0.5) / 4 AS rating
FROM Movie m INNER JOIN  Movie_rate mr ON m.imdbId=mr.imdbId
ORDER BY rating  DESC
LIMIT 5)
```

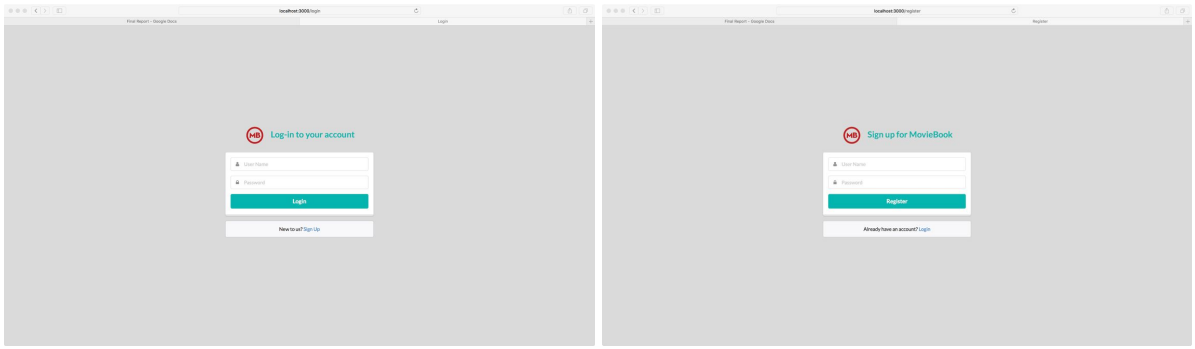
```
UNION All
(SELECT m.title, m.imdbId AS id, COUNT(*) AS rating
FROM user_like ul NATURAL JOIN Movie m
WHERE ul.rating > 3
GROUP BY ul.imdbId
ORDER BY rating  DESC
LIMIT 5)
```

Movie with highest average weighted rate:

```
SELECT m.title, m.imdbId AS id, (mr.RottenTomatoes / 10 + mr.Metacritic /9.4 + mr.IMDB / 0.8 +
mr.Fandango_Stars / 0.5) / 4 AS rating
FROM Movie m INNER JOIN  Movie_rate mr ON m.imdbId=mr.imdbId
ORDER BY rating  DESC
LIMIT 5
```

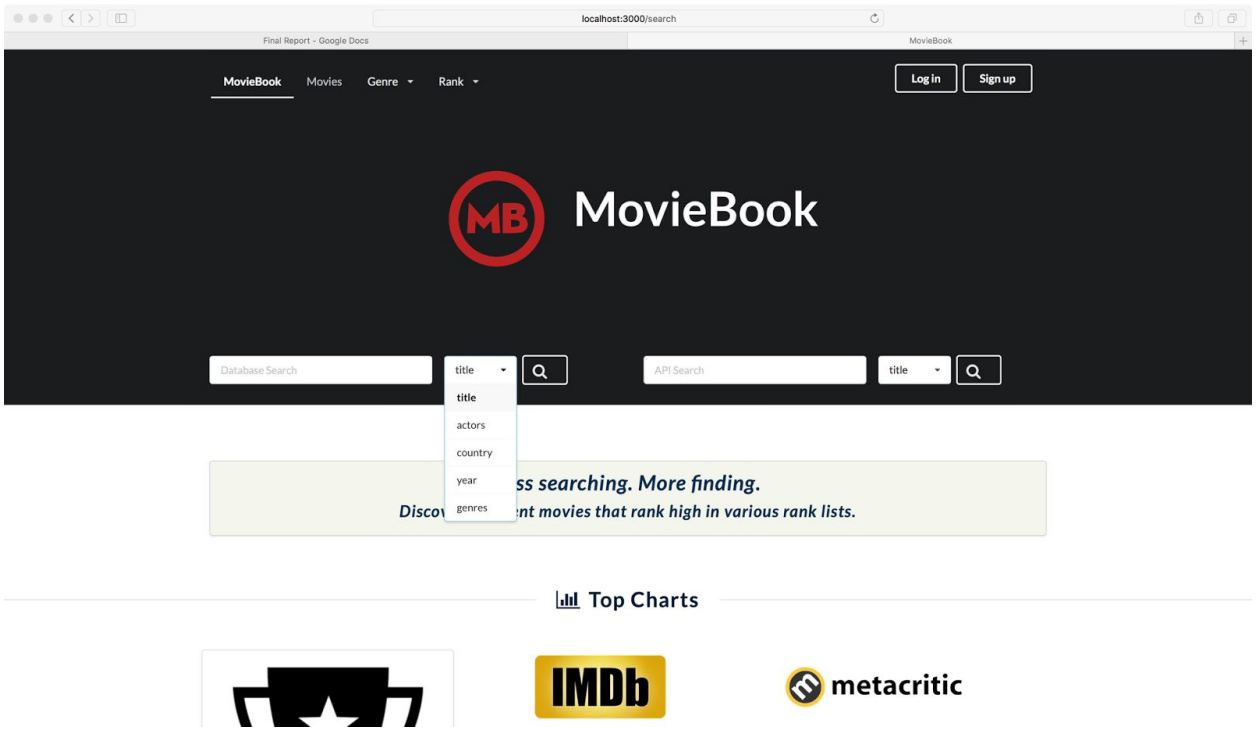
8. Key features

8.1 Login and Signup



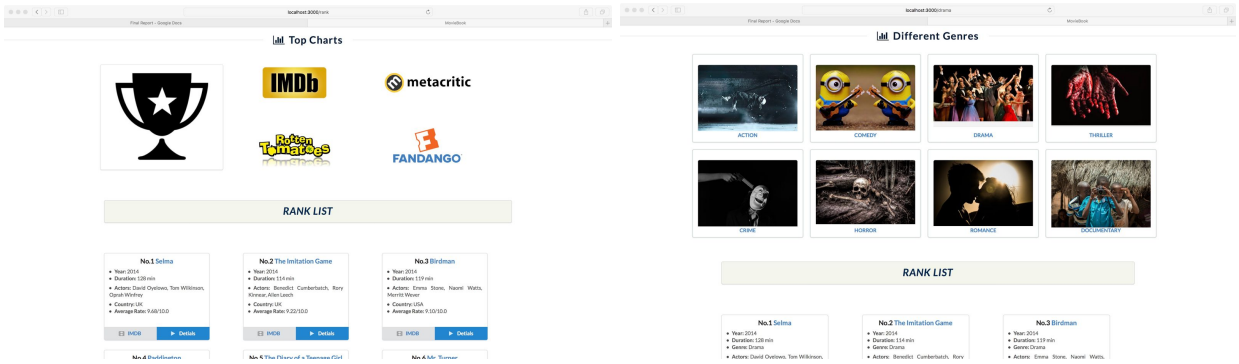
Login page is the first page users will see when they come to MovieBook. When users use their usernames and passwords to login to MovieBook, the server will recognize these users to make other functions of website possible.

8.2 Searching Movies



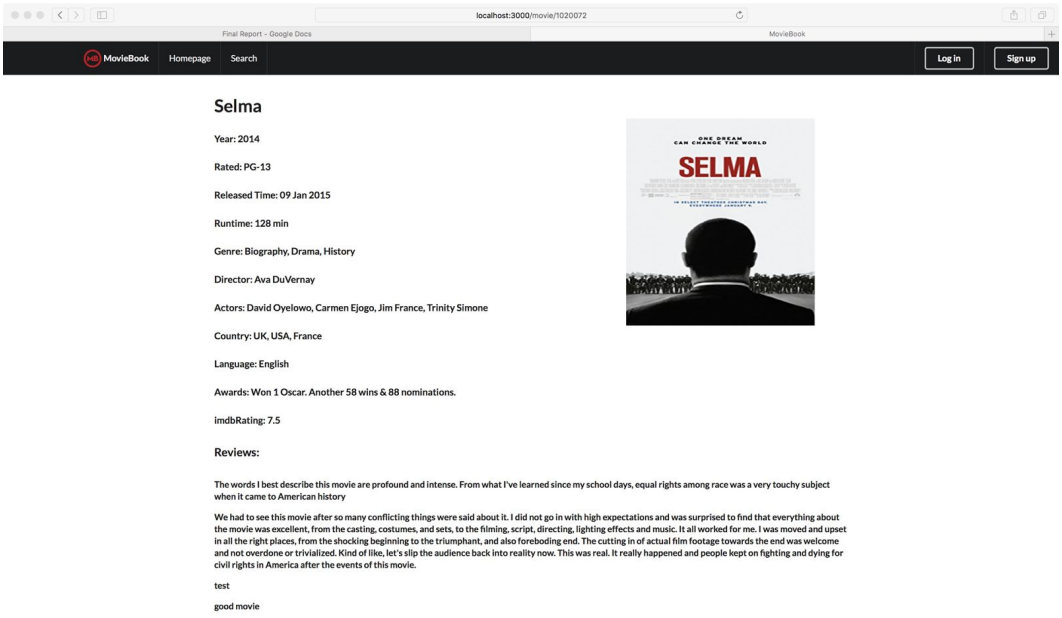
Searching is one of the key functions of our applications. We can search movies by title, actor, country, year, or genre.

8.3 Movies Ranking



There are one main ranking, four rankings according to ratings on four different movie critics websites -- RottenTomatoes, Metacritic, IMDB and Fandango, and eight rankings for different genres. The main ranking is based on the normalized average values of ratings on these four websites.

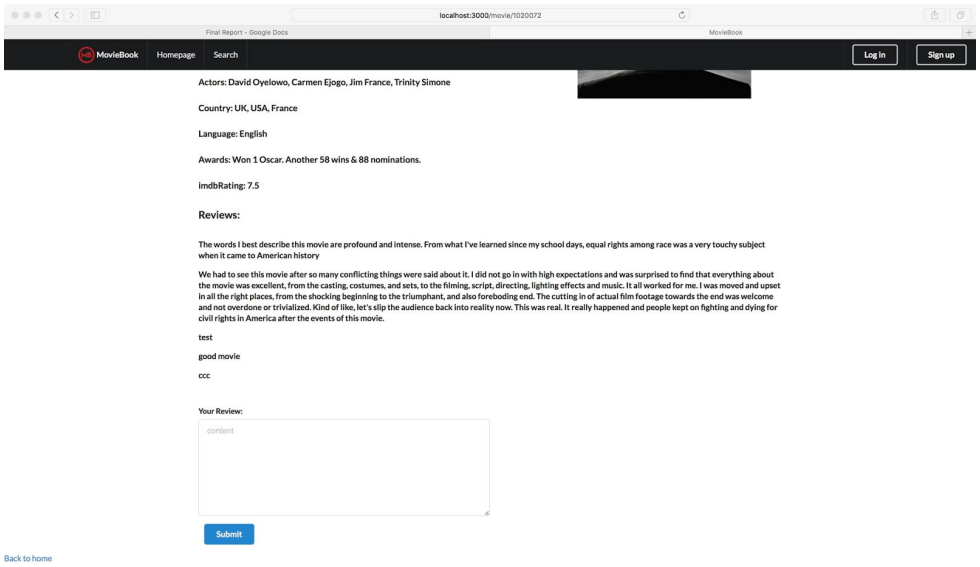
8.4 Movie Page



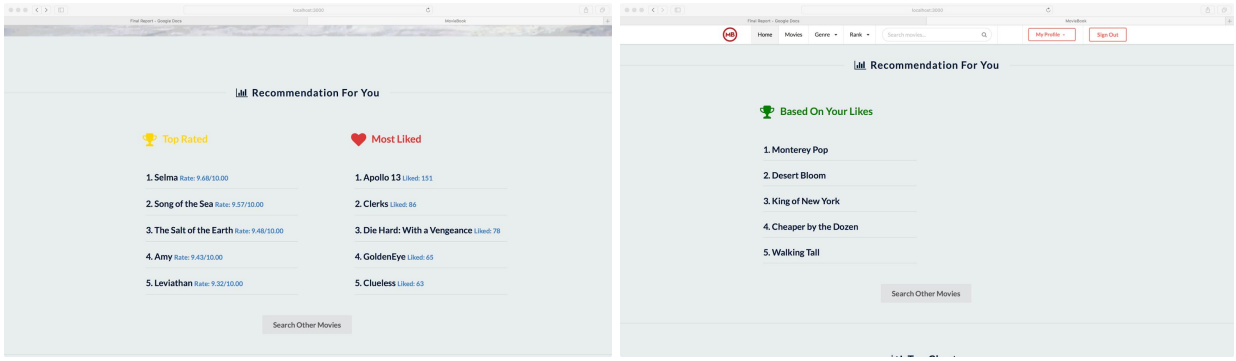
Each movie has a page. Users can find some basic information about the movies, such as their issue year, cast, directors and even their post. What's more,users can read the review of the movie, as well as write reviews.

In addition, users can like or dislike the movie. System can recommend movies to users according to their likes and dislikes.

8.5 Read and Write Movie Reviews

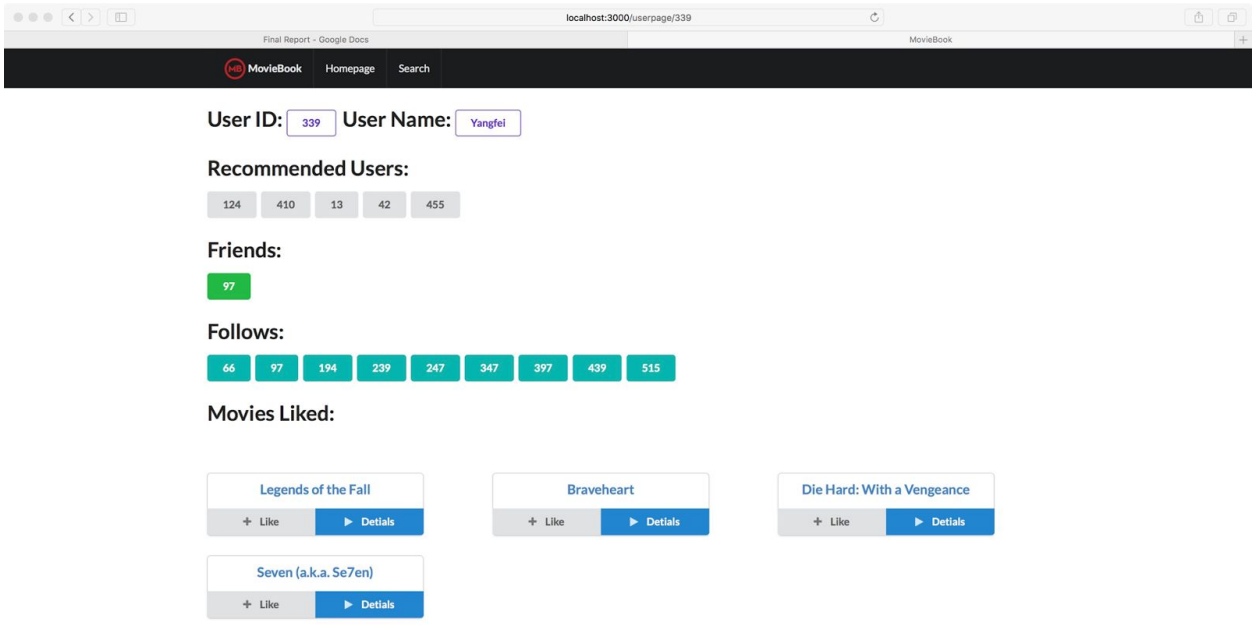


8.6 Movies Recommendation



Before login, recommended movies are based on highest rated movies and on most loved movies; after login, recommendation model is used for personalized movies recommendation.

8.7 User Page



Each user also has their own page. On one user's profile page, public personal information, movies that this user liked, scores that this user rated, friends who the user are followed and following are presented.

8.8 Friend Recommendation

We recommend one user friends whose loved movies are also loved by this user. The first five friends will be recommended by ranking the number of their loved movies. For the example (*Fig*. 5), User 2 and User 3 will be recommended to User 1, and User 2 has a higher priority.

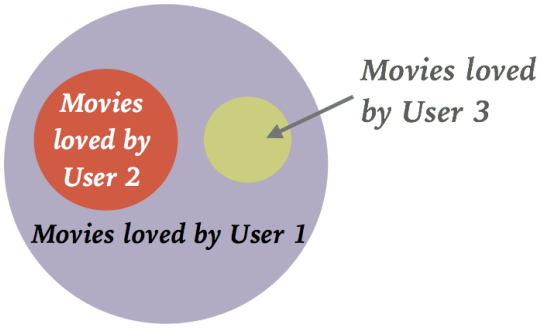


Fig. 6. Example of friends recommendation

9. Recommendation system

The recommendation system will recommend users movies they may like based on movies they have watched and liked.

The core recommendation algorithm is Item-based filtering. The general technique is to precompute the most similar movies for each movie. Then, when you wish to make recommendations to a user, you look at his top-rated movie and create a weighted list of the movies most similar to those.

We build the model in Python, and then serialize the recommendation model and data we need. So when we need to use the model, we don't have to train this model again. In node.js, we use the child_process standard library to spawn a python process and to call the python model.

10. Query optimization and performance evaluation

Firstly we use view to help us optimize the query. The utilization of views can help us allows us to push projections and selections down to the basic tables. The merge of smaller tables helps to optimize the query performance, and our results agrees with this improvement.

We also use index to fasten our query. For example, we have a complex query:

SELECT DISTINCT m.title, d.title_year, d.duration, group_concat(g.genres Separator ',') as genres, d.actor_1_name, d.actor_2_name, d.actor_3_name, d.movie_imdb_link, r.RottenTomatoes, r.Metacritic, r.IMDB, r.Fandango_Stars
FROM Movie m Left JOIN movie_desc d ON m.imdbId = d.imdbId left JOIN Movie_rate r ON m.imdbId = r.imdbId INNER JOIN Genres g ON m.imdbId = g.imdbId
WHERE m.title LIKE '%t%'
Group by m.title
LIMIT 50

This query uses both LIKE and GROUP BY on title. To improve processing speed, we add index to title. Then the query time reduced from 0.136 second to 0.098 second, improving the processing speed by around 40%.

11. Special Features And Extra Credits

1. Semantic UI for better website vision: the front-end development framework Semantic UI is introduced in our application to produce intuitive and beautiful user interface.
2. OMDB API: the OMDB API is implemented in our movie search engine to provide complementary movie information such as movie posters, newly-released movie updates, and to achieve higher completeness of the movie database.
3. Login and personal data: the website is capable of online login by username and password. The password input field is masked for privacy concerns. Every user will have an unique homepage with personalized movie recommendation and customized user page.
4. Personalized recommendation: the recommendation system will recommend users movies they may like based on movies they have watched and liked. The core recommendation algorithm is Item-based filtering.
5. Interaction with movies
    a. Like: User can like or dislike a movie.

   b. Comments: Some movie comments have already been extracted from IMDB website, and stored in the MongoDB. Users are also allowed to add movie comments to any movie, and these new comments information will be updated in MongoDB.

6. Social network service: our application position is a community, which requires us to implement a SNS for our website. Users of our application can follow other users, and when two users follow each other they will establish a friend relation between each other.

12. Technical challenges and how overcome

- Challenge: confused about connecting node.js with python model

How to overcome: we build the model in Python, and then serialize the recommendation model and data we need. So when we need to use the model, we don't have to train this model again. In node.js, we use the child_process standard library to spawn a python process and to call the python model.

- Challenge: have no idea about transferring a folder of TXT files into json file

How to overcome: we used Json package and Pandas package to successfully read all txt file, transformed the data format and stored them as json file.

13. Potential future extensions

MovieBook is community of movie. The potential of this community is enormous, for the idea of interest-based social networking can be applied beyond movie to books, music, and etc. In other words, we add dataset of books, music, sports to this application.

14. Division of work

Yangfei Li is our team leader. He is mainly responsible for front-end building, server side, and back-end building. Haocheng Wu is mainly responsible for server side and back-end building. Yangfei Li and Haocheng Wu also support our database building. Yinchuan Xu and Jiadi Xiong are mainly responsible for recommendation system and database building.