COMP214 Design Analysis Report

Cong Bao, Hongchi Chen, Jinke He, Xiang Li, Zekun Wang, Yuan Zhu
Artificial Intelligence Group Project
Team 1

ABSTRACT

Content-based Filtering and Collaborative Filtering algorithms are commonly used in recommender systems. Nevertheless, in order to offer an appropriate recommendation to users, a large library which contains the features of recommended objects are required, such as categories and tags. In our project, we will introduce a user-based Collaborative Filtering algorithm, which uses the experiences of other users and recommend independently of the features of recommended objects (movies in our project). K-Nearest Neighbour algorithm and Cosine Similarity algorithm will be used to find a list of relevant users. This report will firstly give a description of our system. After that, the algorithms will be introduced in detail. The design of the server system, database, user interface, and testing will also be included. In the end of report, a review of the project process will be given.

CONTENTS

1 PROJECT DESCRIPTION

1.1 BACKGROUND

As data is being an important part of people's life, using "Big data" to help people with decision making is on trend. For instance, recommending the song with the same style by some music player applications, filtrating restaurants with the user's flavour. Our project is to design an algorithm-based movie recommender system which uses artificial intelligence method including machine learning to speculate what kind of movies the user is interested in, which will reduce the user's time of browsing many movies but only find few movies the user preferred.

As planned in the requirement phase, the movie recommender system will focus on the Collaborative Filtering (CF) algorithm instead to provide recommendations based on users' ratings. More specifically, rather than using an average value of ratings as a reference, CF tries to find users who share similar interests by analysing the previous ratings of a user [1]. After comparing several different AI machine learning algorithms, we choose k-Nearest Neighbours (k-NN) algorithm to find users with similar interests with higher performance of data analysing and less mistakes. By calculating the similarity with others, k-NN will find some users who have similar interests in movies [2]. Based on the ratings made by the users that found by k-NN, the system will predict the ratings of a user on the movies he/she has not seen yet. In brief, the movie recommendation system uses a combination of CF algorithm and K-NN algorithm to calculate the result. After that, a list of recommended movies will be generated according to the predicted ratings. This report will lay emphasis on the algorithm design, and the database and interface design as well, but also an introduction of testing design.

1.2 OBJECTIVES

The main purpose of our project is to develop a movie recommender system based on CF algorithm and k-NN algorithm. A new algorithm to predict movie ratings will be introduced by combining the two algorithms. The target customer we designed for will be the following three groups:

- People who love to watch movies but have no idea what movies he/she may like.
- People who have interests on current most popular or top rated movies.
- People who would like to know information and rating of a specific movie.

Additionally, to support the recommender system, a database system and a web server system will be established as well. Eventually, customers will be able to interact with the movie recommender system through a web site, at which user will be provided with a personal recommended movies list according to their ratings. Some additional functions, such as searching for a movie or a user, comments under movies, adding movies to favourites, following or unfollowing other users, will also be added in the system to ensure its integrity.

1.3 ORIGINAL PROPOSAL

The original proposal of this project is to practice using artificial intelligence method in reality, with designing database and software testing as well. The project is aimed at making us acquainted with the whole process of a software project, from the requirement phase to design phase, then to implementation phase and testing phase.

In our project, we focus on the artificial intelligence machine learning method. At the beginning of designing the project we attend to only use K-NN algorithm, however we decided a multiple-based algorithm should be designed more suitable for our movie recommender system, and K-NN is just a simple classifier for lazy learning. Then we choose CF and K-NN algorithm and combine them as the new method.

1.4 RESEARCHES

Till now, we have done several researches which are shown below.

- Algorithm: CF & K-NN's implementation and corresponding pseudo code.
- Database: Interface design, Data dictionary, Entity-relation diagram, Map-Reduce model.
- User-interface (UI): Website Map Diagrams, jQuery 3.1.1, Bootstrap 3.3.7 UI framework
- Applications: Apache 2.4, Nginx 1.10, Apache Tomcat 8.0, Django 1.11, Struts 2.3, Hibernate 4.3, Spring 4.0, MySQL 5.7, MQTT, PuTTY, Git.

2 PROJECT DESIGN

2.1 INTRODUCTION

2.1.1 System Description

The main form of this project is designed as a website. After user login to the website, the user will have his own movie list calculated by the database according to the user's rating on the movie displayed on the website. The rating of the user will be processed by the database system with CF and K-NN algorithm. The database will store the information of movies and the accounts as well. An account can interact with another by the "Follow user" function.

The database of the system will contain an original movie database and the user account database. The data stored in the movie database will be combined with the "tags" which is the rating made by the users. According to the "tags", the clustered data can be easily classified for speculating the user preference of the current user by calculating and displaying k numbers of nearest neighbours.

2.1.2 Functional Description

- All registered users could delete their account if they want, after which they will become a non-registered user.
- All registered users could reset their password if they forgot them or want to have a new one.
- All registered users could rate movies and add movies to their favourites, which will affect the speculation of the recommendation.
- The website should provide registered users with new recommended movies lists each time the user rated a new movie.
- All registered users could set preferences to decide which kind of movies will be recommended.
- The website should provide registered users with a list of recommended user accounts. The algorithm will not only calculate the movie for recommendation, but also conversely speculate accounts with similar movie ratings.
- All registered users can follow or unfollow another registered user.
- The website should provide the function to search a movie.
- The website should provide the function to search a registered user.
- The website should have a good compatibility with mobile browsers.
- A friendly error page with help messages should be given if the website runs into problems.

2.1.3 Use Cases

The user case diagram is shown in Figure. 1.

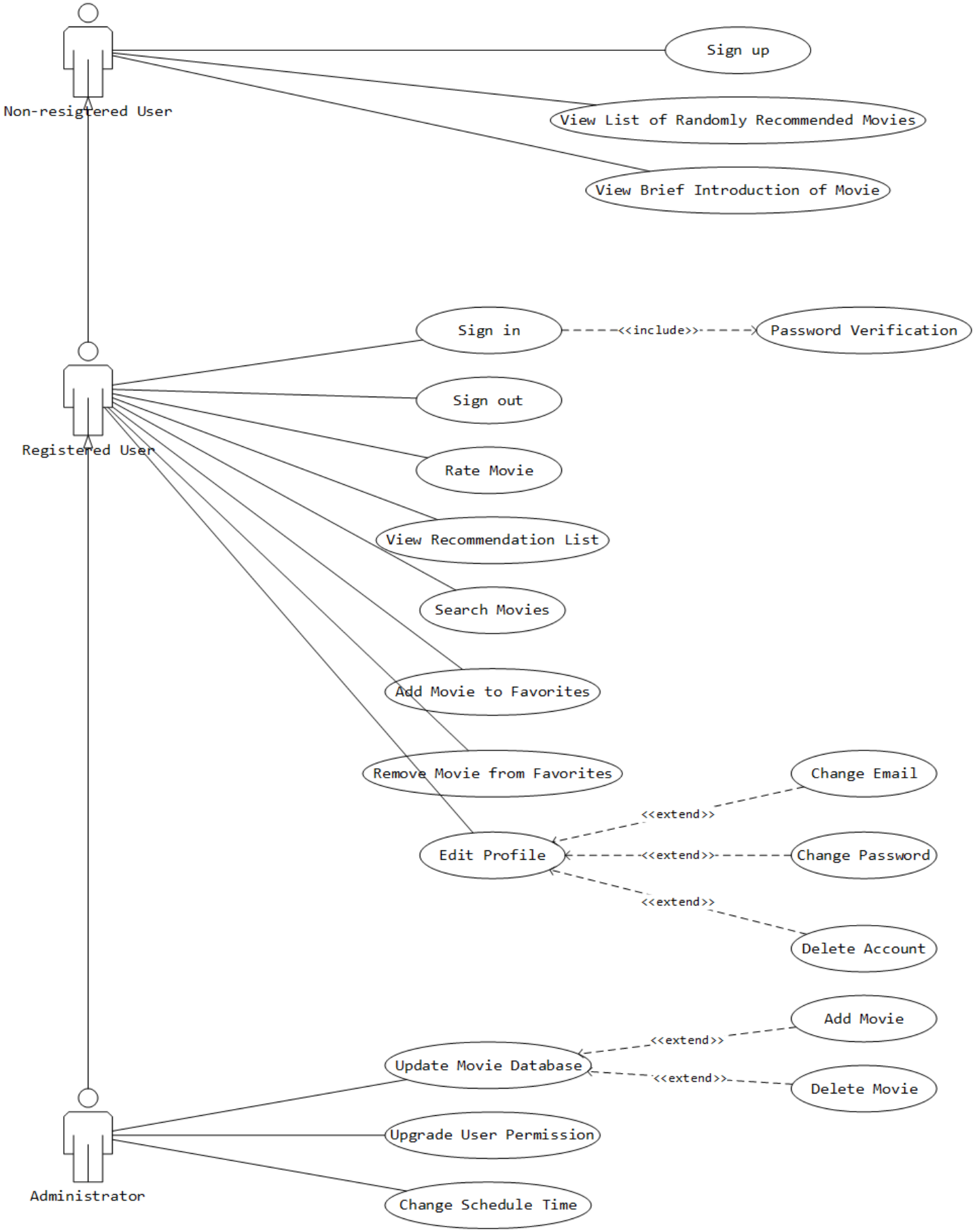


Figure. 1. Use Case Diagram

2.2 ALGORITHM DESIGN

2.2.1 Background

In general, there are two common algorithms to implement a movie recommender system – Content-based Filtering and Collaborative Filtering. The algorithm used by the system is User-based Collaborative Filtering.

Content-based Filtering uses the features of movies, which are the so called "tags". For example, a recent movie "Logan" has the tags of "Action", "x men", "based on marvel comic", etc. The tags of movies are typically their genres but they can also be their keywords, such as "based on a book", "talking animals" and even the name of a star. To generate a recommendation list, Content-based Filtering firstly needs to create a huge library consisting of all the possible tags of movies. Then, by analysing the movies that a user liked in the past using learning techniques such as neural networks, clustering analysis and decision tree, it can derive the weights which indicate the importance of each tag to the user. The weights of tags to a user are commonly called the "user preference". According to the derived user preference, the probability of whether a user will like a movie can be estimated so the system can generate a recommendation list by picking out the movies with highest probability of being liked by the user [3].

Nevertheless, there are several limitations of Content-based Filtering. Firstly, the system using Content-based Filtering needs the data of the tags of movies, which is hard to collect by a small group. Secondly, for the system using Content-based Filtering, before training the algorithm using machine learning technique, the data needs to be cleaned because for some movies, they may have multiple tags with the same meaning. For example, if there are two tags, "zombie" and "zombies", they will be treated differently in the learning process so the duplicates must be removed before the training process. It is true that for data source, it can be guaranteed that this situation will never happen but if the data is collected from different sources, some algorithms must be used to eliminate duplicates. It is quite hard for us, a group of six people to deal with this problem in a short time. Therefore, our project uses another well-known recommendation algorithm, User-based Collaborative Filtering, which uses the experiences of other users and recommend independently of the features of movies.

2.2.2 Algorithm Specification

- *User-based Collaborative Filtering*
  Basically, User-based Collaborative Filtering is to find the users who are similar to the target user and predict the ratings of the target user on the movies that he/she hasn't seen based on the ratings of the similar users [3].
  According to our design, the recommender system will receive an id number when a user sends a request to the web server. Then, the system will read all the rating information from the database and each rating consists of the user id, the movie id which is rated and the rating value which is a number between 0 and 5. Using K-Nearest Neighbour and Cosine Similarity algorithm, the system can find K users who are most similar to the target user and predict the potential ratings of the target user on each movie based on the ratings of the K neighbours and the similarities between them. In this way, the final recommendation list will be made up of movie ids which have the highest predicted rating values.
- *K-Nearest Neighbour Algorithm*
  K-Nearest Neighbour Algorithm is to predict a value or classify a thing based on its k nearest cases. The output of an input can be the average of the values from the nearest neighbours. It can also be useful to assign weights to the nearest neighbours and then, the output will be the sum of the weighted values. In our recommender system, to predict the target user's rating on a movie, the value can be computed as the sum of weighted ratings from the nearest neighbours and the weights are the similarities between the nearest neighbours and the target user.
  In our movie recommender system, the neighbours are the users who have rated at least one same movie with the target user and the nearest neighbours are the neighbours who have the highest similarities with the target user. The similarity between two users is decided by their previous ratings using Cosine Similarity Algorithm, which will be discussed later. As mentioned earlier, each movie that the target user

hasn't seen before will be given an estimated value, which indicates how likely the target user may like the movie. The estimated value of a movie is the sum of ratings from the nearest neighbours which are greater than a threshold times the similarity between the nearest neighbour who gave this rating and the target user. The threshold is called the recommendation threshold. Only movies which were given ratings greater than this threshold will be recommended by the ratters. By default, the recommendation threshold is set to 3 but the value may change for better performance of the system. After traversing all the nearest neighbours to compute the estimated values for all the movies, the system can know how much the target user may like each movie. Finally, the recommendation list will be made up of movies with highest predicted values. Notice that the estimated value of each movie is not really the estimated rating of the target user on the movie because the value may be larger than 5 or very small. However, since the system only needs to know which movies have the highest estimated values, the real value of each movie does not really matter.

- *Cosine Similarity*

An algorithm is needed to compute the similarity between two users based on their previous ratings. However, there are various similarity algorithms available. For example, Euclidean Distance, which is a well-known similarity algorithm to compute the distance between two points in a multi-dimensional space. In mathematics, the distance between two points using Euclidean Distance is

$$d_{12} = \sqrt{\sum_{k=1}^{n}(x_{1k} - x_{2k})^2} \quad (1)$$

However, if the system uses Euclidean Distance, there is a serious problem that according to Euclidean Distance, the more ratings on same movies the two users have made, the higher similarity they will have. This is not desirable because in the situation that two users have rated many same movies but their ratings are very different, Euclidean Distance will give a very high similarity. Therefore, Euclidean Distance is not a suitable similarity algorithm to our recommender system because the similarity algorithm used in this system should focus on how similarly the two users made ratings on the same movies and it should be independent of the number of same movies they have rated on. Thus, the system uses another similarity algorithm, the Cosine Similarity Algorithm.

Basically, Cosine Similarity is a measure of the similarity between the directions of two vectors. The value it uses is the cosine value of the angle between the two vectors so it is independent of the magnitudes of the vectors. According to Euclidean dot product formula, in a two-dimensional space, given two vectors $\boldsymbol{A}, \boldsymbol{B}$ and the angle between the vectors θ, there is $\boldsymbol{A} \cdot \boldsymbol{B} = ||\boldsymbol{A}|| \, ||\boldsymbol{B}|| \, \cos\boldsymbol{\theta}$. Then, in coordinates, the cosine value of θ can be computed as

$$\cos\theta = \frac{x_1 x_2 + y_1 y_2}{\sqrt{x_1^2 + y_1^2} \times \sqrt{x_2^2 + y_2^2}} \quad (2)$$

The greater $\cos\theta$is, the smaller the angle between the vectors is. Therefore, a larger $\cos\theta$ indicates a higher similarity. It has also been proved that the formula $\boldsymbol{A} \cdot \boldsymbol{B} = ||\boldsymbol{A}|| \, ||\boldsymbol{B}|| \, \cos\boldsymbol{\theta}$ also holds in multi-dimensional space so similarly, the similarity of two multi-dimensional vector can be computed as [4]

$$similarity = \cos\theta = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2} \times \sqrt{\sum_{i=1}^{n}(B_i)^2}} \quad (3)$$

In our recommender system, the ratings of each user are a multi-dimensional vector while each movie id the user rated is a dimension and the rating on the movie is the projection on that dimension. If the user did not rate a movie, then the project on that dimension is set to zero. Therefore, using the formula given above to compute the similarity between two multi-dimensional vectors, the system can generate the similarity between two users based on their previous ratings. Each similarity will have a value between 0 and 1. Since Cosine Similarity Algorithm focuses on the orientation of two vectors, the similarity between two users will also focus on the difference in their orientations of ratings, which means the similarity will be independent of the number of same movies they have rated. It is thus that Cosine Similarity Algorithm is very suitable for this system.

2.2.3 Interface Design

In this section, the interfaces are focus on algorithms.

Interface	`List<Movie> Recommend(User target_user, int k);`
Description	Generate a recommendation list of movie ids to the target user.
Parameters	`target_user` – the user who needs a recommendation list `k` – the number of nearest neighbours the system needs to look at
Return Value	A list of movie ids which are recommended to the target user

Interface	`Dictionary CreateUserMovieDic(List<rating> ratings);`
Description	Create two dictionaries based on all the user ratings.
Parameters	`ratings` – the whole list of ratings from the users. Each rating entity contains user_id, movie_id and the rating.
Return Value	`user_to_movie_dic` – a dictionary in which each user id is a key and the value of this key is the ratings of the user (List<Movie_id, rating>) `movie_to_user_dic` – a dictionary in which each movie id is a key and the value of this key is the list of user ids who have rated it (List<User_id>)

Interface	`Dictionary GetAllNeighborsSimilarity(target_user_id, movie_to_user_dic, user_to_movie_dic);`
Description	Generate a dictionary which records the similarities between the target user and other users.
Parameters	`target_user_id` – the id of the user who needs a recommendation list `movie_to_user_dic` – the dictionary in which each movie id is a key and the value of this key is the list of user ids who have rated it (List<User_id>) `user_to_movie_dic` – the dictionary in which each user id is a key and the value of this key is the ratings of the user (List<Movie_id, rating>)
Return Value	`all_neighbors_similarity_dictionary` – the dictionary in which the similarity between a user and the target user is the key and the value is the id of this user. The dictionary is sorted in the order of similarity(key).

Interface	`CalculateCosinDistance(ratings_of_user1, ratings_of_user2);`
Description	Calculate the Cosine Similarity between two users based on their previous ratings. The computation is similar to the computation of the Cosine Similarity between two multi-dimensional vectors.
Parameters	`ratings_of_user1 (List<Movie_id, rating>)` – the ratings of user1, each of which contains the movie id and the rating value. `ratings_of_user2 (List<Movie_id, rating>)` – the ratings of user2, each of while contains the movie and the rating value.
Return Value	`similarity` – a float number which represents the similarity between user 1 and user2.

2.2.4 Pseudo-Code

The pseudo-codes of the interfaces are shown in below.

Main Function

```
Function Main
    Set target_user_id = Get user_id From server
    // recommendation_list only consists of movie_ids of the recommended movies
    Set recommendation_list = Recommend(target_user_id, k)
    Set result = new empty list
    For Each movie_id In recommendation_list Do:
        Set movie_info = Get information of Movie (movie_id) From Database
        Append movie_info To reuslt
    End For
    Send result To server
End Function
```

Recommend Function

```
Function Recommend(target_user_id, k)
    Set recommendation_dictionary = new empty dictionary
    Set ratings [Global] = Read All ratings (List of [user_id, movie_id, rating]) From database
    Set movie_to_user_dic, user_to_movie_dic = CreateUserMovieDic()
    Set all_neighbors_similarity_dictionary = GetAllNeighborsSimilarity(target_user_id,
                                              movie_to_user_dic, user_to_movie_dic)
    Set k_nearest_neighbors = Select the top k entities([similarity, neighbro_id]) From
                              all_neighbors_similarity_dictionary // find k nearest neighbors
    For Each neighbor([similarity, neighbor_id]) In k_nearest_neighbors Do:
        Set neighbor_movie_ratings(List of [movie_id, rating]) = user_to_movie_dic[neighbor_id]
        For Each movie_rating([movie_id, rating]) In neighbor_movie_ratings Do:
            If rating > recommend_threshold Do:
                If movie_id Is Not In recommendation_dictionary Do:
                    Create a new key movie_id
                    Set recommendation_dictionary[movie_id] = rating * similarity
                Else Do:
                    Add rating * similarity To recommendation_dictionary[movie_id]
                End If
        End For
    End For
    Sort recommendation_dictionary from high value to low value
    Return list of keys in recommendation_dictionary
End Function
```

CreateUserMovieDic Function

```
Function CreateUserMovieDic(ratings)
    Set user_to_movie_dic = new empty dictionary
    Set movie_to_user_dic = new empty dictionary
    For Each rating([user_id, movie_id, rating]) In ratings Do:
        Set movie_rating = [movie_id, rating]
        If user_id is already a key in user_to_movie_dic Do:
            Append movie_rating To user_to_movie_dic[user_id]
        Else Do:
            Create a new key user_id
            Set user_to_movie_dic[user_id] = new empty list
            Append movie_rating to user_to_movie_dic[user_id]
        End If
        If movie_id is already a key in movie_to_user_dic Do:
            Append user_id To movie_to_user_dic[movie_id]
```

```
        Else Do:
            Create a new key movie_id
            Set movie_to_user_dic[movie_id] = new empty list
            Append user_id To movie_to_user_dic[movie_id]
        End If
    End For
    Return user_to_movie_dic, movie_to_user_dic // return two dictionaries
End Function
```

GetAllNeighborsSimilarity Function

```
Function GetAllNeighborsSimilarity(target_user_id, movie_to_user_dic, user_to_movie_dic)
    Set all_neighbor_ids = new empty list
    For Each movie_rating([movie_id, rating]) In user_to_movie_dic[target_user_id] Do:
        Set movie_id = movie_rating[0]
        For Each user_id In movie_to_user_dic[movie_id] Do:
            If user_id is not a key in all_neighbor_ids And user_id != target_user_id Do:
                Append user_id To all_neighbor_ids
            End If
        End For
    End For
    Set all_neighbors_similarity_dictionary = new empty dictionary
    For Each neighbor_id In all_neighbor_ids Do:
        Set similarity = calculateCosinDistance(user_to_movie_dic[target_user_id],
                         user_to_movie_dic[neighbro_id]
        Append Key-value Pair(similarity, neighbro_id) To all_neighbors_similarity_dictionary
    End For
    Sort all_neighbors_similarity_dictionary from low to high according to the key
    Return all_neighbors_similarity_dictionary
End Function
```

CalculateCosinDistance Function

```
Function CalculateCosinDistance(ratings_of_user1(List of [movie_id, rating]),
         ratings_of_user2(List of [movie_id, rating]))
    Set sum_x = 0.0
    Set sum_y = 0.0
    Set sum_xy = 0.0
    For user1_movie([movie_id, rating]) In ratings_of_user1(List of [move_id, rating]) Do:
        For user2_movie([movie_id, rating]) In ratings_of_user2(List of [move_id, rating]) Do:
            If user1_movie[0] == user2_movie[0] Do: // if both of them have rated the movie
                // userx_movie[0] is the movie_id, usex_movie[1] is the rating value
                Add user1_movie[1] * user2_movie[1] To sum_xy
                Add user1_movie[1] * user1_movie[1] To sum_x
                Add user2_movie[1] * user2_movie[1] To sum_y
            End If
        End For
    End For
    If sum_xy == 0.0 Do:
        Return 0
    End If
    Set sx_sy = the square root of sum_x * sum_y
    Set similarity = sum_xy / (sum_x * sum_y)
    Return similarity
End Function
```

2.3 SERVER DESIGN

2.3.1 Background

Server is playing the role to support the whole system, managing each part and making response to users' requests. Therefore, it is necessary to build a stable and flexible server which can bear large amounts of requests and easy to add new features into it.

The goal of resisting high concurrency is usually depending on the web server software, such as Apache and Nginx. These web servers have their own distinctive techniques to handle large number of requests. However, in the other side, it is also hard to change the web server software to improve server's performance. Instead, we concentrate on the improvement of efficiency of handling each request. As shown in the formula below:

$$k = \frac{R_{proc}}{R_{recv}} \tag{4}$$

In which the variable R_{proc} means the number of requests processed in a time slot, and R_{recv} means the number of requests received in a time slot. If the ratio k is much greater than 1, the server is possible in an idle state, because the amount of request is quite small or the computing speed of server is quite large. If the ratio k is approaching to 0, the server is possible in a busy state, because there are large number of requests or the processing speed of server is quite low. Ideally, the ratio k should approaching to or equal to 1, which means the server is fully utilized. In our design, we will improve the variable R_{proc} to a reasonable level to balance with the requests received. To achieve this, other than using faster CPUs and other high performance hardware, we will also focus on the algorithm, which has introduced in the algorithm design section, and the using of some efficient framework, which will be described in the next section.

To achieve the goal of flexibility, some software architectural patterns will be used. The most common pattern is the Model-View-Controller (MVC) framework. By separating different components from the whole system, MVC reduces the coupling and improves the cohesion of a system. In addition, developers can work in parallel on different components without impacting others. This feature makes it easy to add new functions into a system as each function is a new component and it couples a little with other components. In our system, we will make use of three famous open source frameworks, which are Struts, Spring, and Hibernate, and integrate them together. A brief introduction to these frameworks and detailed system structure description will be covered in next section.

2.3.2 Structure and Framework

The structure of our web server can be described as five layers, which are data source layer, data access layer, service layer, web layer, and display layer. Algorithm Server can be described as two layers. It is intuitive from the diagram below (Figure. 2).

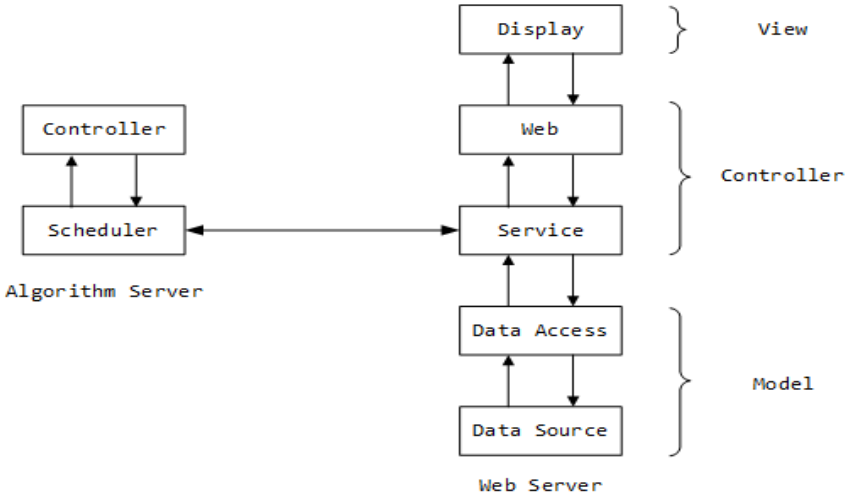


Figure. 2. System Structure

As shown in the diagram, the data source layer and data access layer formulate the model layer in MVC. The service layer and web layer act as the controller, and the display layer acts as the view. The following texts will introduce each layer from the bottom to the top.

The data source layer is a representation of database. This layer will not include any create, read, update, and delete (CRUD) operation, but translate data in relational database to an object-oriented class, which is known as object-relational mapping (ORM). After this converting, higher layers can access the data in database more easily as the system is realized in an object-oriented language. The realization of ORM will depend on the Hibernate framework, which is an open source software written in Java. By using this library, much work will be saved as it is easy to map a class field to a database attribute using the Java annotations [5].

The data access layer provides a serial of CRUD operations to higher layers. In other words, this layer will encapsulate query languages of DBMS as some abstract interfaces. Higher layers can easily access data from database by calling functions provided by data access layer rather than writing SQL to perform CRUD operations. Also, after performing CRUD operations, data access layer will return instances of data source layer to higher layers, which avoid the controller access database directly.

The service layer is a part of the controller layer in MVC. Service layer will focus on business logic rather than data accessing. It will receive the requests from higher layers and return results after processing. The requests sent to this layer is more detailed than lower layers. It focusses on some detailed businesses such as user login, user registration, movie recommendation, and so on. The realization of data access layer and service layer will depend on Spring framework. Spring framework uses inversion of control (IoC) to manage interfaces and implementation objects. It utilizes Java reflecting technique to create instances from configuration files such as XML file, reduce the coupling between different objects [6]. This pattern is also known as dependency injection. Another feature Spring framework provides is the aspect-oriented programming (AOP). Different from object-oriented programming, AOP focuses on aspects of objects, which makes it easy to add features to similar objects with less programming [7]. In this project, AOP will be used to listen for changes in database.

The web layer is another part of controller. Different from service layer, web layer will not process business logic directly. Instead, it will transmit requests from higher layers to appropriate functions in service layer, and return processing results to higher layers. Additionally, web layer has the mission to verify requests from higher layers to make sure all requests are valid and safe. To realize this layer, the Struts framework will be used. Struts encapsulates the Java Servlet and is powerful in URL analysis [8]. It is easy to use a XML file to show the mapping of processing actions and results.

The display layer represents the view layer in MVC. In this layer, no business logic should be processed. The only thing this layer will do is to display a graphical user interface using techniques such as HTML, CSS, JavaScript, and JSP. If there are some user requests sent, such as a form is submitted, the display layer should pack parameters and send it to lower layers, and wait for the processing results.

In the algorithm server, there are two layers, which are controller layer and scheduler layer. The scheduler layer contains a scheduler that will launch at a fix time to exchange data with web server. The controller is used to manage the scheduler, for example, changing the schedule time. The protocol used to exchange data between servers is Message Queuing Telemetry Transport (MQTT) protocol. MQTT is a lightweight Pub-Sub protocol that usually used in Internet of Things (IoT) [9]. In this project, MQTT is used for that it is easy to manage and will not increase too much network flow during scheduling.

2.3.3 Interface Design

In this section, the interfaces will focus on the service layer.

The user manager interfaces:

Interface	`int validLogin(User user);`
Description	Check if a login request is valid.
Parameters	`user` – The user to be checked.
Return Value	A flag of result, where • `LOGIN_FAIL = -1` • `LOGIN_USER = 0` • `LOGIN_ADMIN = 1`

Interface	`int validRegister(User user);`
Description	Check if a register request is valid.
Parameters	`user` – The user to be checked.
Return Value	A flat of result, where • `REGISTER_FAIL = -1` • `REGISTER_USER = 0` • `REGISTER_ADMIN = 1`

Interface	`boolean addFavorite(Favorite favorite);`
Description	Add a favorite record to database.
Parameters	`favorite` – the favorite record to be added
Return Value	If the operation succeeds

Interface	`boolean isUserExist(User user);`
Description	Check if a user is existing in database.
Parameters	`user` – the user to be checked
Return Value	If the user is existing in database.

Interface	`User updateAccount(User origin, String account);`
Description	Update the account record of a user.
Parameters	`origin` – the user to be updated `account` – the new account of user
Return Value	The user record after updating.

Interface	`User updatePassword(User origin, String password);`
Description	Update the password record of a user.
Parameters	`origin` – the user to be updated `password` – the new password of user
Return Value	The user record after updating.

Interface	`User updateEmail(User origin, String email);`
Description	Update the email record of a user.
Parameters	`origin` – the user to be updated `email` – the new email of user
Return Value	The user record after updating.

Interface	`User updateMailVerifyState(User origin, boolean verified);`
Description	Update the state of mail verification of a user.
Parameters	`origin` – the user to be updated `verified` – the new mail verifies state of user
Return Value	The user record after updating.

Interface	`User updateRole(User origin, int roleId);`
Description	Update the role of a user.
Parameters	`origin` – the user to be updated `verified` – the new role id of user
Return Value	The user record after updating.

Interface	`User getUserByAccount(String account);`
Description	Obtain a user record by the user's account.
Parameters	`account` – the account of user
Return Value	The user record obtained.

Interface	`Role getRoleByAccount(String account);`
Description	Obtain a role record by the user's account.
Parameters	`account` – the account of user
Return Value	The role record obtained.

Interface	`UserBean getUserBeanByAccount(String account);`
Description	Obtain a user bean instance by the user's account.
Parameters	`account` – the account of user
Return Value	The user bean instance obtained.

Interface	`RoleBean getRoleBeanByAccount(String account);`
Description	Obtain a role bean instance by the user's account.
Parameters	`account` – the account of user
Return Value	The role bean instance obtained.

Interface	`List<Favorite> getFavoritesByUser(User user);`
Description	Obtain a list of favorite records of a user.
Parameters	`user` – the user to be queried
Return Value	A list of favorite records obtained.

Interface	`void deleteUser(User user);`
Description	Delete a user record in database.
Parameters	`user` – the user to be deleted

Interface	`void deleteFavorite(Favorite favorite);`
Description	Delete a favorite record in database.
Parameters	`favorite` – the favorite record to be deleted

The movie manager interfaces:

Interface	`boolean addMovie(Movie movie);`
Description	Add a movie record to database.
Parameters	`movie` – the movie record to be added
Return Value	If the operation succeeds

Interface	`boolean addRating(Rating rating);`
Description	Add a rating record to database.
Parameters	`rating` – the rating record to be added
Return Value	If the operation succeeds

Interface	`boolean isMovieExist(Movie movie);`
Description	Check if a movie is existing in database.
Parameters	`movie` – the movie to be checked
Return Value	If the movie is existing in database.

Interface	`boolean isRatingExist(Rating rating);`
Description	Check if a rating record is existing in database.
Parameters	`rating` – the rating record to be checked
Return Value	If the rating record is existing in database.

Interface	`Movie updateYear(Movie origin, int year);`
Description	Update the year record of a movie.
Parameters	`origin` – the movie to be updated `year` – the new year of movie
Return Value	The movie record after updating.

Interface	`Movie updateTitle(Movie origin, String title);`
Description	Update the title record of a movie.
Parameters	`origin` – the movie to be updated `title` – the new title of movie
Return Value	The movie record after updating.

Interface	`Rating updateRating(Rating origin, float rating);`
Description	Update the rating value of a rating record.
Parameters	`origin` – the rating record to be updated `rating` – the new rating value of the rating record
Return Value	The rating record after updating.

Interface	`Movie getMovieById(int id);`
Description	Obtain a movie record by the movie's id.
Parameters	`id` – the id of movie
Return Value	The movie record obtained.

Interface	`MovieBean getMovieBeanById(int id);`
Description	Obtain a movie bean instance by the movie's id.
Parameters	`id` – the id of movie

Return Value	The movie bean instance obtained.

Interface	`List<Movie> getMoviesByYear(int year);`
Description	Obtain a list of movie records by year.
Parameters	`year` – the year of movie
Return Value	A list of movie records obtained.

Interface	`List<Movie> getMoviesByTitle(String title);`
Description	Obtain a list of movie records by title.
Parameters	`title` – the title of movie
Return Value	A list of movie records obtained.

Interface	`List<Movie> recommendMoviesToUser(User user);`
Description	Recommend a list of movies to a user.
Parameters	`user` – the user to be recommended
Return Value	A list of recommended movies.

Interface	`List<Rating> getRatingsByUser(User user);`
Description	Obtain a list of rating records of a user.
Parameters	`user` – the user to be queried
Return Value	A list of rating records obtained.

Interface	`List<Rating> getRatingsByMovie(Movie movie);`
Description	Obtain a list of rating records of a movie.
Parameters	`movie` – the movie to be queried
Return Value	A list of rating records obtained.

Interface	`void deleteMovie(Movie movie);`
Description	Delete a movie record in database.
Parameters	`movie` – the movie record to be deleted

Interface	`void deleteRating(Rating rating);`
Description	Delete a rating record in database.
Parameters	`rating` – the rating record to be deleted

2.3.4 Class Diagram

The class diagram of the whole system is shown in Appendix I.

2.3.5 Interaction Chart

The interactions between system's five layers is shown in the following figures (Figure. 3 to Figure. 8).

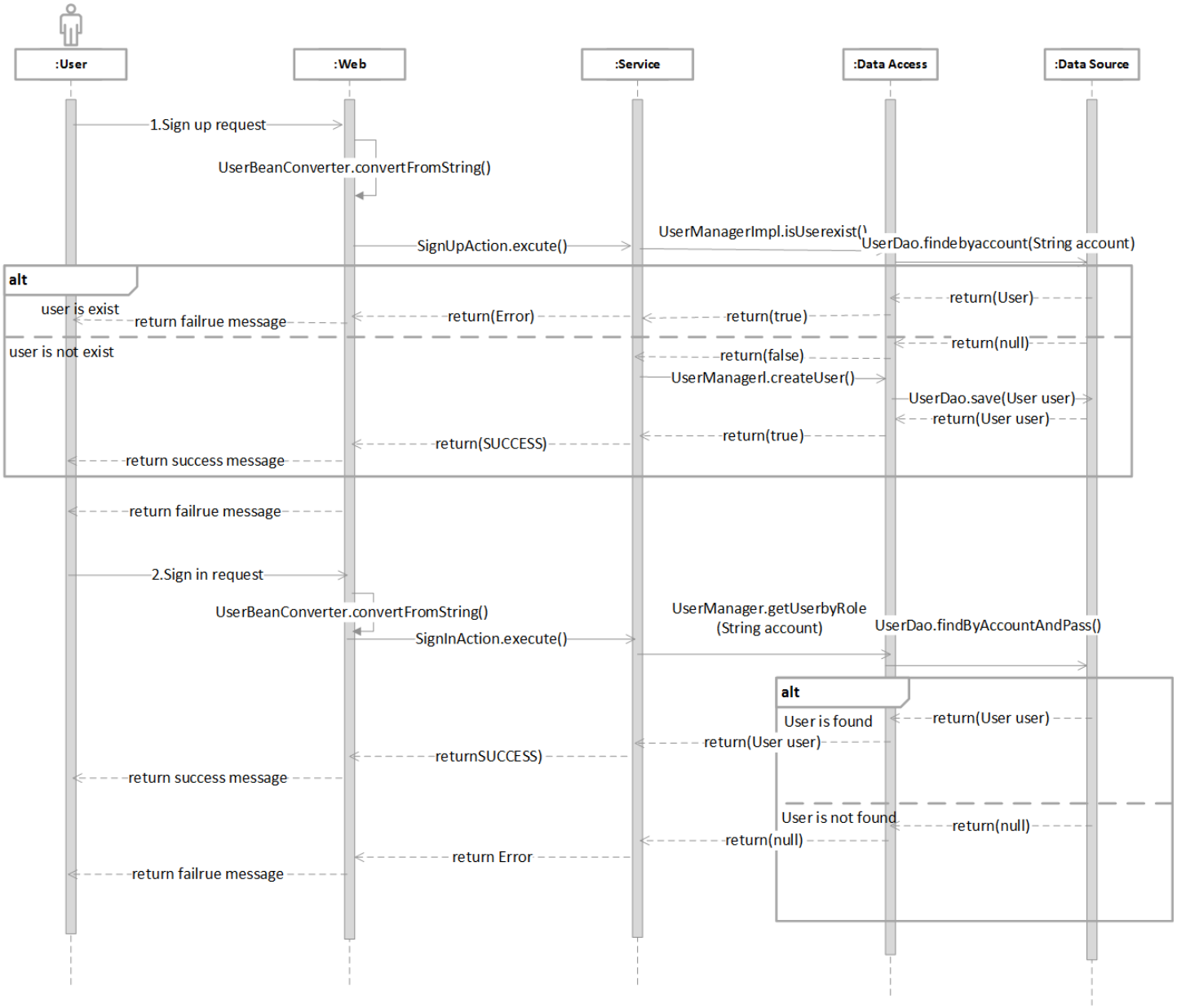


Figure. 3. Sign up/Sign in Interaction Diagram

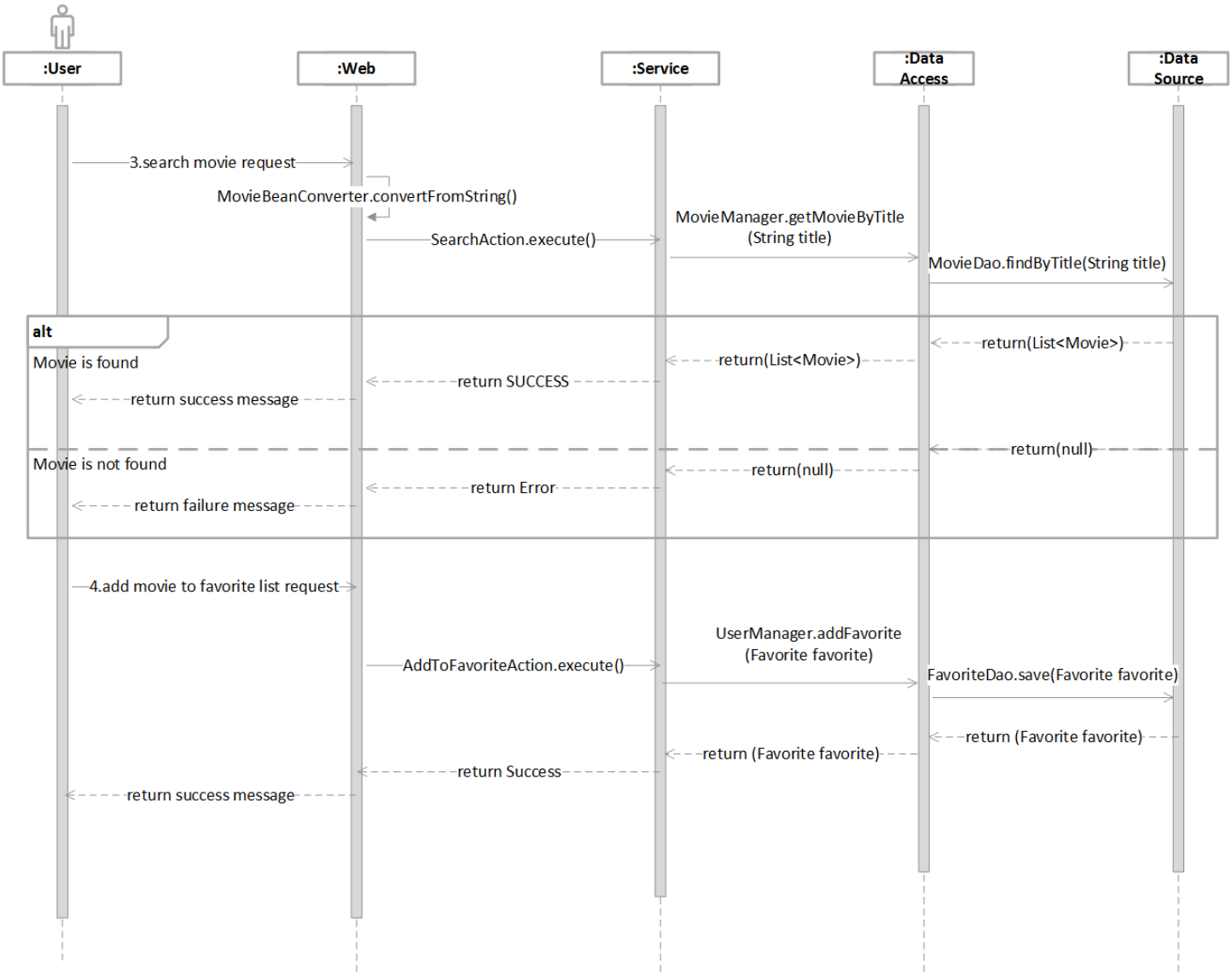


Figure. 4. Search movie/Add Favourite Interaction Diagram

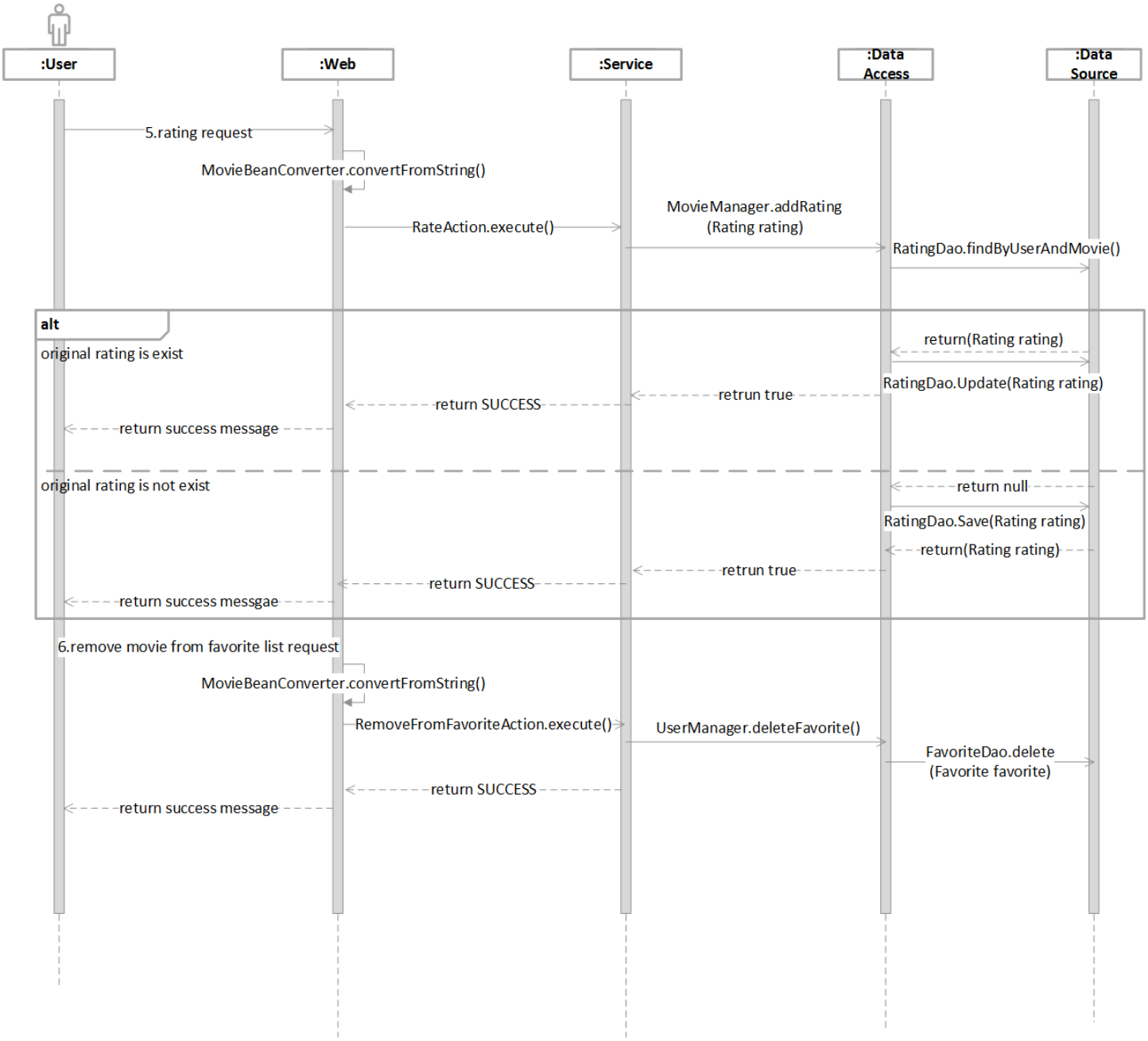


Figure. 5. Rating/Remove favourite Interaction Diagram

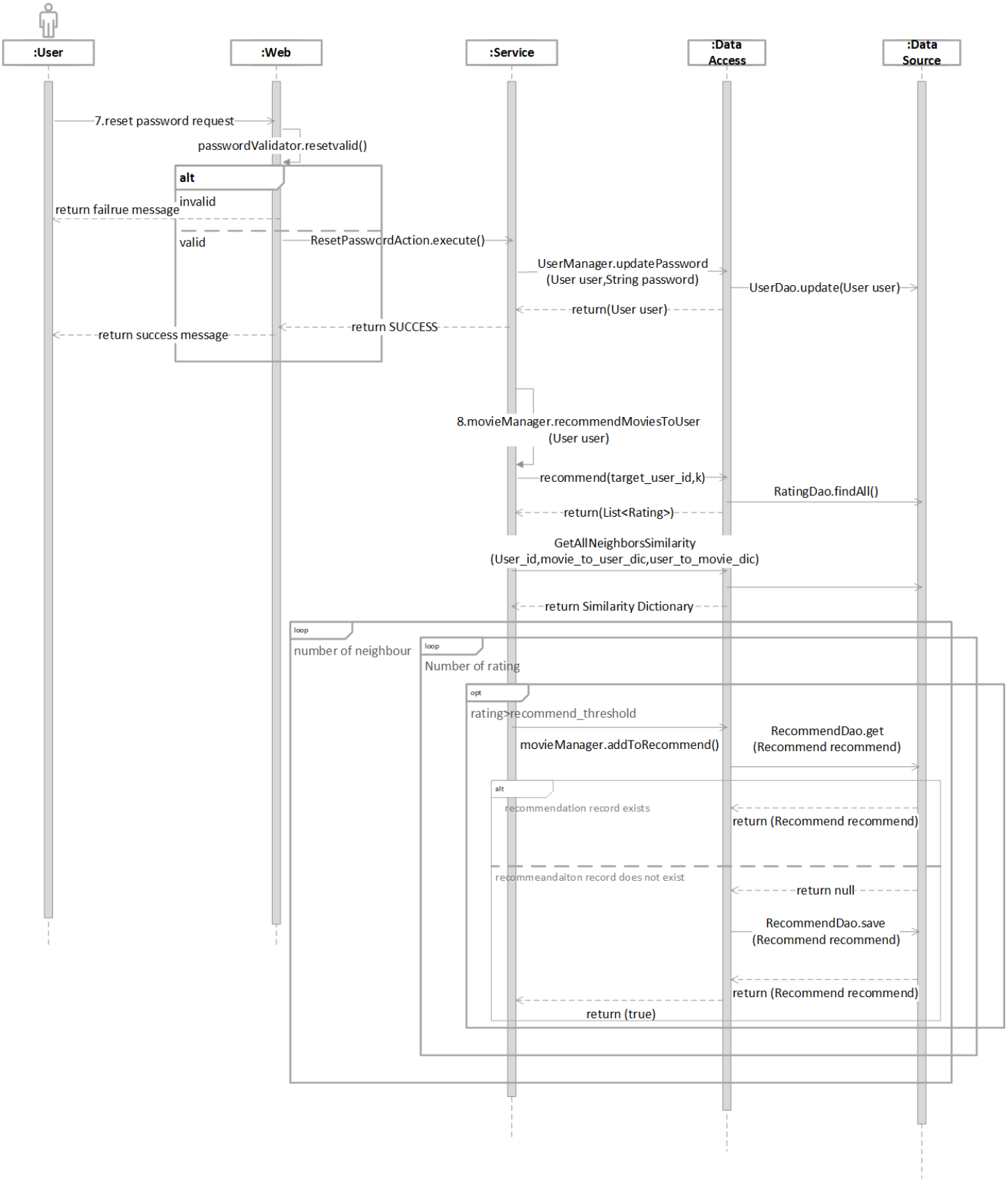


Figure. 6. Reset password/Recommend movie Interaction Diagram

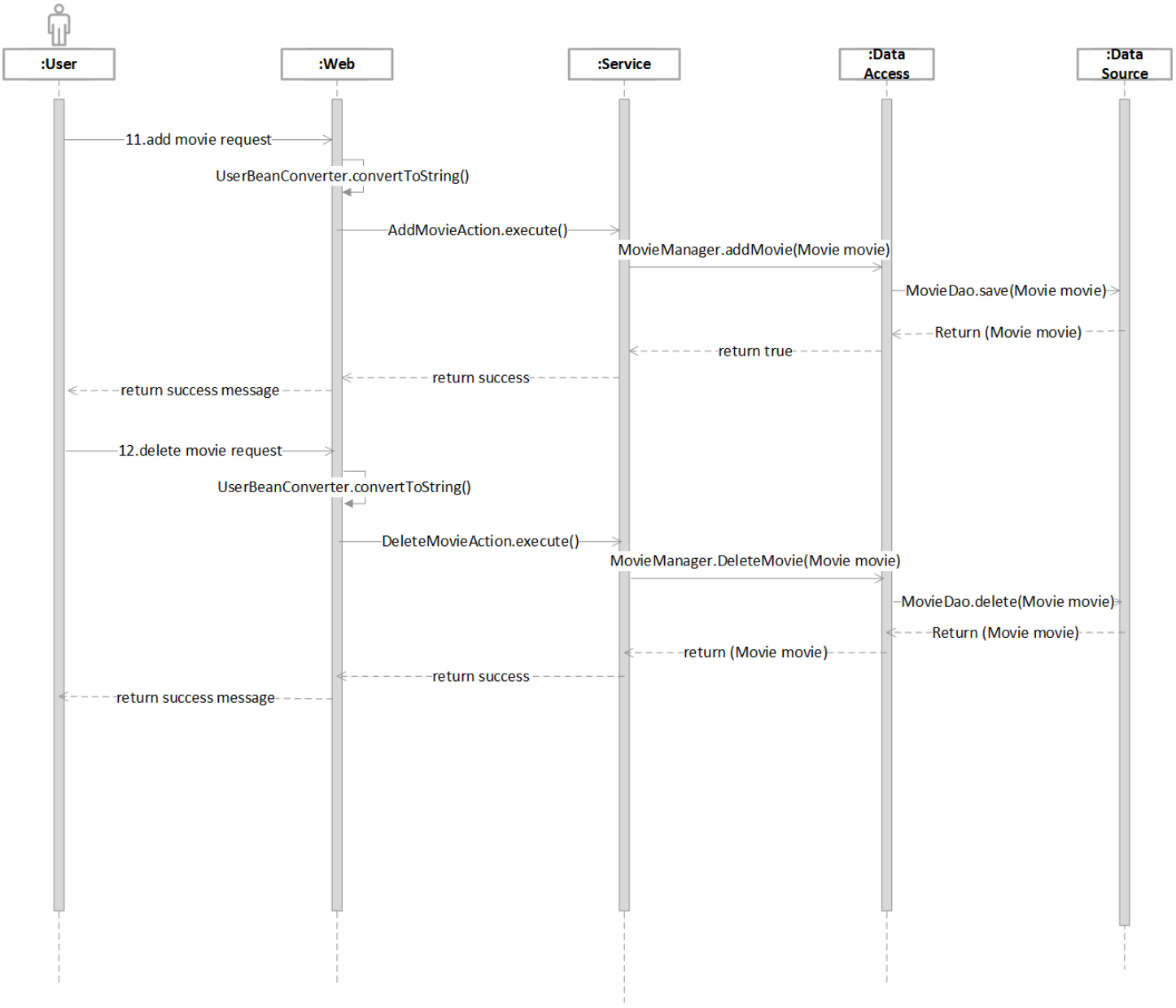


Figure. 7. Add/Delete Movie Interaction Diagram

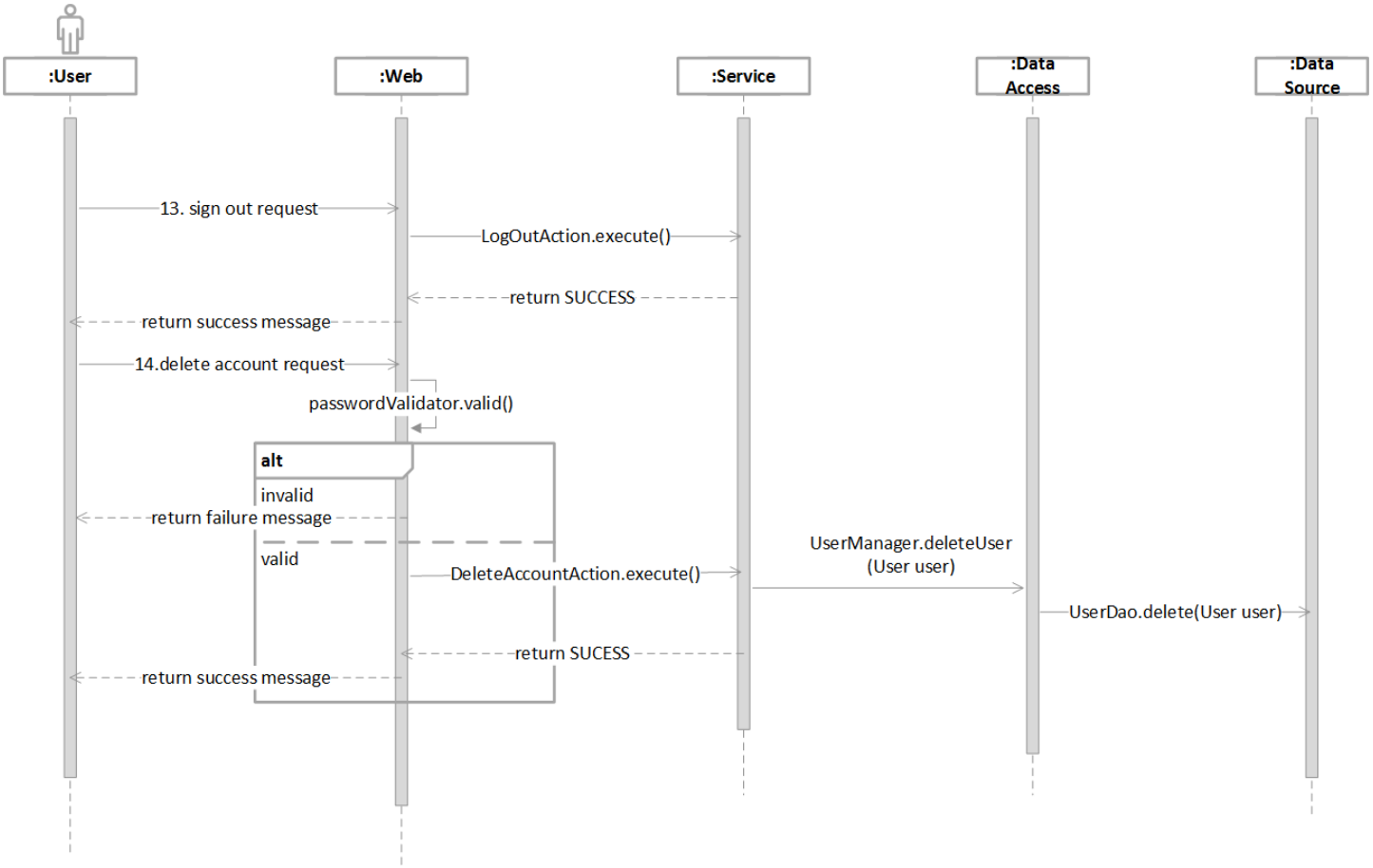


Figure. 8. Sign out/Delete account Interaction Diagram

2.4 DATABASE DESIGN

2.4.1 Background

In contemporary network design, it is important and common to use external storage device as database carrier. The data stored in a database is easy to be invoked, processed and stored. In a system, the storage and partition of the database usually correspond to objects, which improve the efficiency, in the meantime extruding construction of system. The database quantity is large and well-organized, therefore easy to be used in statistic model and well-prepared to be used in forward research or display. The Retrieve-Process-Update model is becoming more essential when needing to process a medium or large quantity of data [10].

The structure of the system can be described by certain diagrams, being represented by data dictionary and entity relation model. Each attribute of the table will be assigned few properties to set constraint, including data type and relational index with other table, in order to create relationship between entities to generate a more specific and efficiency data [11]. The properties and relationship of data are the key point of database design.

In the Movie Recommendation System, the main objects and data associated with system are divided into five basic parts: the user who can interact with system, the movie to be recommended and reviewed, the favourite relationship record with each existing user and movie pair, the rate score graded by user to a movie, and a simple user confidential level, has been named as ‘role’, to control the authorization and authentication issue. Each of these objects are stored as an individual table in the database, can be retrieved by SQL.

2.4.2 Data Dictionaries

The centralized data dictionary stores all data information and their structures of movie recommendation system, including hardware and software. In the meantime, the specific information relevant to system administration is also represented in the data dictionary [12].

Entity Description Dictionary

The dictionary table describe every entity in the movie recommendation system, including the main description, their possible aliases called and understand, and their usage in the system.

Entity	Description	Aliases	Occurance
User	A person who use the system	registered user	allow interacting with the system and execute specific operation, including rating and favouriting, etc.
Movie	Describe the basic information of a system	movie	Contains the title, year and other informations of a movie
Favourite	A function to record a user has marked and favourited a specific movie	registered movie	Intersaction between user and movie to let user add a specific movie to the favourite list
Rating	A user's grade to a specific movie	grade	Record each user's grade of a movie to calculate the specific average grade of that movie
Role	The accessibility of the user to the system	accessibility, authority	To control the accessibility of user and prevent unauthorized behaviour

Relation Description Dictionary

The table describes every relationship between system, including name of two entity, their multiplicity, and the association between them.

Entity	Multiplicity	Relationship	Multiplicity	Entity
User	0..*	favourite	0..*	Movie
User	1..1	rate	0..*	Rating
Movie	1..1	has rating	0..*	Rating
Role	1..1	authorize	0..*	User

Attribute Description Dictionary

The table describes every attribute of each entity, including their description, data type with length, if possible to be NULL, if could be multivalve, and the role played in their entity table.

Entity	Attributes	Description	Data Type and Length	Nulls	Multivalued	Key
User	user_id	a specific id created for each user	INT, 11	No	No	Primary Key
User	role_id	the authorization type of a user	INT, 11	No	No	Foreign Key
User	account	the account name created and entered by user	VARCHAR, 64	No	No	NULL
User	email	the register email of user	VARCHAR, 64	No	No	NULL
User	mail_verified	if the email used to register has or has not been verified by clicking email link	CHAR, 1	No	No	NULL
User	password	a string contains alphabetical and decimal characters created by user to authenticate user	VARCHAR, 64	No	No	NULL
User	recommendation	a list of recommended movie ids	BLOB	Yes	No	NULL
Movie	movie_id	a specific id created for each movie	INT, 11	No	No	Primary Key
Movie	title	the name of movie	VARCHAR, 256	No	No	NULL
Movie	year	publish year of movie	INT, 11	Yes	No	NULL
Movie	imdb	the id of movie in imdb system	INT, 11	No	No	NULL
Role	role_id	a specific id created for each role type	INT, 11	No	No	Primary Key
Role	role	the accessibility type of user	VARCHAR, 32	No	No	NULL
Rating	rating_id	a specific id created for each rating record	INT, 11	No	No	Primary Key
Rating	user_id	the user id of who rated the movie	INT, 11	No	No	Foreign Key

Rating	movie_id	the movie id of which movie was rated	INT, 11	No	No	Foreign Key
Rating	rating	the numeric value of rating	FLOAT	No	No	NULL
Favourite	user_id	the user id of who favourited the movie	INT, 11	No	No	Primary Key
Favourite	movie_id	the movie id of which movie was favourited	INT, 11	No	No	Primary Key

2.4.3 Entity-Relationship Diagrams

The ER diagram (Figure. 9) indicates entities and relationships with movie recommendation system objects.

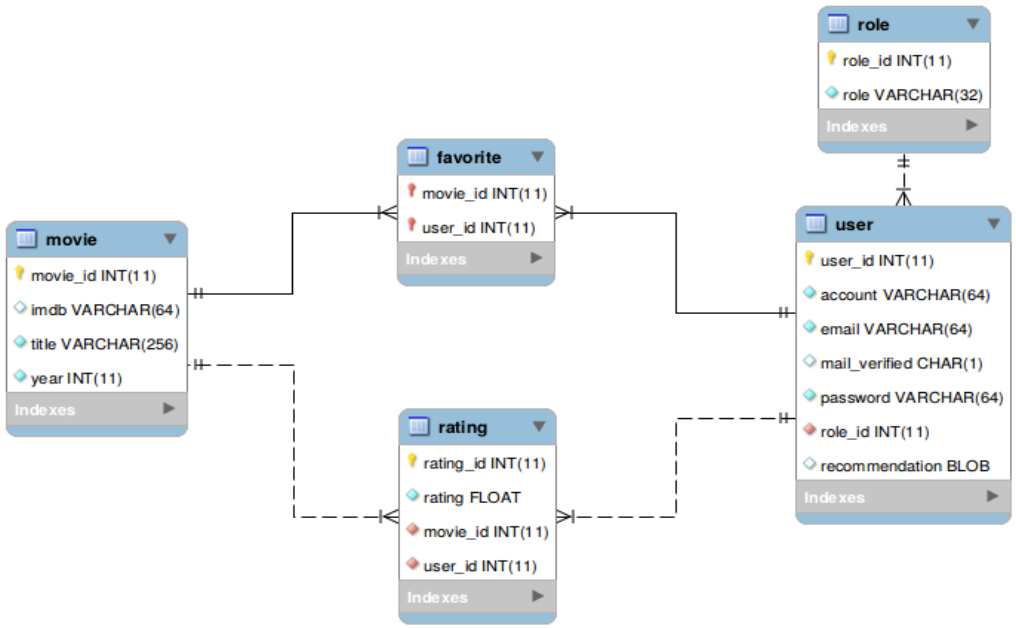


Figure. 9. Entity-Relationship Diagram

Entities:

- User Table:
  The primary table of the system stores user information.
  *User_ID: Integer with length of 11, not NULL, primary key
  *Role_ID: Integer with length of 11, not NULL, foreign key from 'Role Table'
  *Account: Varchar with maximum length of 256, not NULL, unique attribute
  *Email: Varchar with maximum length of 64, not NULL, unique attribute
  *Mail_Verified: Boolean value, not NULL
  *Recommendation: Byte value
- Movie Table:
  The table stores all movie information
  *Movie_ID: Integer with length of 11, not NULL, primary key
  *Title: Varchar with maximum length of 64, not NULL
  *Year: Integer with length of 11, not NULL
- Rating Table:
  The table stores all users' single rating grades to movie
  *Rating_ID: Integer with length of 11, not NULL, primary key
  *User_ID: Integer with length of 11, not NULL, foreign key from 'User Table'
  *Movie_ID: Integer with length of 11, not NULL, foreign key from 'Movie Table'

*Rating: Float value (to store fractional part), not NULL

- Role Table:
  The table stores the authorization level and code correspondence
  *Role_ID: Integer with length of 11, not NULL, primary key
  *Role: Characters with length of 32, not NULL
- Favourite Table:
  The linked list of User and Movie table stores the movie favorite by user relationship
  *Movie_ID: Integer with length of 11, not NULL, foreign key from 'Movie Table'
  *User_ID: Integer with length of 11, not NULL, foreign key from 'User Table'

Relationships

- Rate:
  Between 'User Table' and 'Rating Table', that ONE user can give N rating grades, with each corresponds to a movie
- Has rating:
  Between 'Movie Table' and 'Rating Table', that ONE movie can have N rating grades, with each correspond to a user
- Authorize:
  Between 'Role Table' and 'User Table', that ONE user can has only one role, meanwhile one authorization role can correspond to N users.
- Favourite:
  Between 'User Table' and 'Favourite Table', that ONE user can has N favourite record and each one related to a specific movie, while a favourite record can only have one user.
- Be favourited:
  Between '' and '', that ONE movie can has N favourite record and each one related to a specific user, while a favourite record can only have one movie.

2.4.4 Interface Design

In this section, the interface design will focus on the data access layer.

The base data access object interfaces:

Interface	`T get(Class<T> entityClazz, Serializable id);`
Description	Load an entity via id.
Parameters	`entityClazz` - the class of entity `id` - a serializable entity instance contains id
Return Value	the entity got

Interface	`Serializable save(T entity);`
Description	Save an entity to database.
Parameters	`entity` - the entity to save
Return Value	the entity saved

Interface	`void update(T entity);`
Description	Update an entity record.
Parameters	`entity` - the entity to update
Return Value	

Interface	`void delete(T entity);`
Description	Delete an entity record.
Parameters	`entity` - the entity to delete
Return Value	

Interface	`void delete(Class<T> entityClazz, Serializable id);`
Description	Delete en entity via id
Parameters	`entityClazz` - the class of entity `id` - a serializable entity instance contains id
Return Value	

Interface	`List<T> findAll(Class<T> entityClazz);`
Description	Obtain all entities.
Parameters	`entityClazz` - the class of entity
Return Value	a list of all entities obtained

Interface	`long findCount(Class<T> entityClazz);`
Description	Obtain the total number of entity.
Parameters	`entityClazz` - the class of entity
Return Value	the total number of entity

The user data access object interfaces:

Interface	`List<User> findByAccountAndPass(User user);`
Description	Find a list of users by account and password.
Parameters	`user` - the user instance contains account and password
Return Value	a list of users found

Interface	`User findByAccount(String account);`
Description	Find a user record by account.
Parameters	`account` – The account of user
Return Value	return the user record found

The movie data access object interfaces:

Interface	`Movie findById(Integer id);`
Description	Find a movie record by id.
Parameters	`id` - the id of movie
Return Value	the movie record found

Interface	`List<Movie> findByYear(Integer year);`
Description	Find a list of movie records by year.
Parameters	`year` - the year of movie
Return Value	a list of movie records found

Interface	`List<Movie> findByTitle(String title);`
Description	Find a list of movie records by title.
Parameters	`title` - the title of movie
Return Value	a list of movie records found

The favourite data access object interfaces:

Interface	`List<Favorite> findByUser(User user);`
Description	Find a list of favourite records by user.
Parameters	`user` - the user to be queried
Return Value	a list of favourite records found

Interface	`List<Favorite> findByMovie(Movie movie);`
Description	Find a list of favorite records by movie.
Parameters	`movie` - the movie to be queried
Return Value	a list of favorite records found

The rating data access object interfaces:

Interface	`List<Rating> findByUser(User user);`
Description	Find a list of rating records by user.
Parameters	`user` - the user to be queried
Return Value	a list of rating records found

Interface	`List<Rating> findByMovie(Movie movie);`
Description	Find a list of rating records by movie.
Parameters	`movie` - the movie to be queried
Return Value	a list of rating records found

The role data access object interfaces:

Interface	`Role findById(Integer id);`
Description	Find a role record by id.
Parameters	`id` - the id of role
Return Value	the role record found

2.5 USER INTERFACE DESIGN

2.5.1 Background

User interface design contains the design of the website map, which presents the structure of website. All subsites will be covered by the website map, and navigation of website will also be clearly shown on the map. Developers will benefit from the user interface design, which can make the structure of website become visible. The user interface design is a necessary work before develop a website with several subsites connected.

2.5.2 Website Map diagram

The website map diagram is shown in Figure. 10.

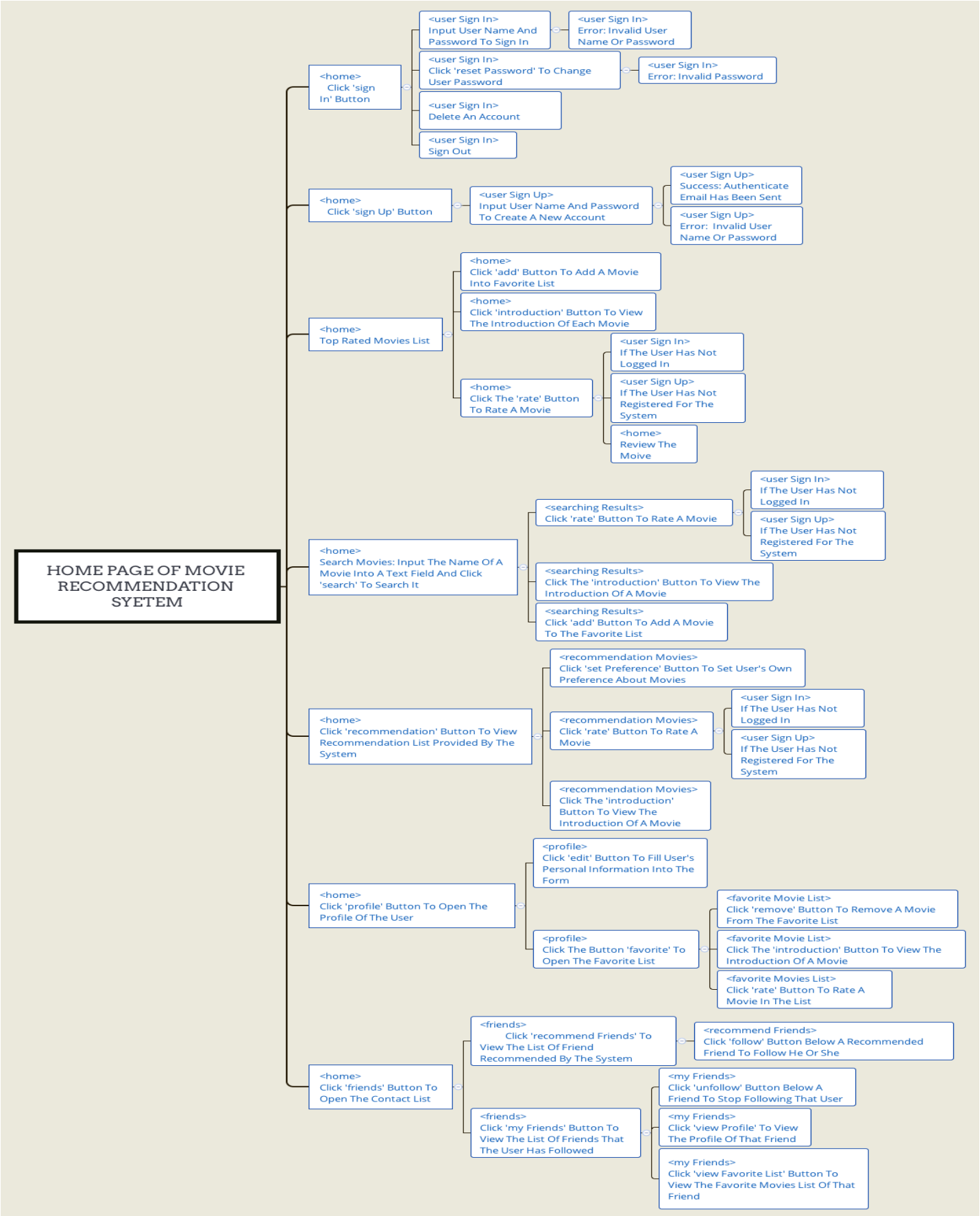


Figure. 10. Website Map Diagram

Several main function areas will be displayed on the homepage of the website.

- Sign in and sign up function:
  Sign up function will require a non-registered user to create a new account for the system, and the sign in function will require a registered user to input the user name and password to log in. Both of these functions will lead to a page which will show the information about whether the operation has succeeded or not.
- Top Rated Movies List:
  A list of movies with top scores or with top box office will be displayed on the home page of the website. There will be several function areas below each movie, such as add movie to the favourite list, view the introduction, rate the movie and so on.
- Search movies function:
  A text field will be put on the home page. After a user inputting a movie name and click 'search' button, a subsite about the information of the movie will be displayed. Function areas will be the same as ones on top rated movies list.
- View the recommendation list:
  There will be a 'recommendation movies' button on the home page. It will lead to the subsite about movies recommended by the system.
- Edit profile function:
  After a user clicking the 'profile' button, the profile of that user will be present in a new page. The user can edit it with personal information. The favourite list can also be viewed by clicking a 'my favourite' button on the profile.
- Search friend function:
  A user can click 'my friends' button on the homepage the open his or her address list. For each friend on the list, the user can choose whether he or she want to follow that friend.

2.6 TESTING DESIGN

2.6.1 Background

The testing Design will contain the evaluation criteria for different parts of the system and testing methods of testing. Testing design is used to select testing methods and set requirements for the system. Executing the methods in the testing design and evaluating the results by criteria set in the testing design will help developers eliminate defects and make system more useable and reliable.

2.6.2 Evaluation Criteria

- Algorithm:
  Criterion:
  There should be less coupling among each algorithm.
  Method:
  After an algorithm is implemented, the use of parameters, important variables, main functions etc. will be checked whether they are related to other algorithm blocks.
  Criterion:
  Accuracy and usability.
  Method:
  Unit testing will be used to check whether there is any bug or defect in every functional unit. Integration testing will also be used to test the whole algorithm. Different situations of inputs, including extreme ones will be set and outputs from algorithm will checked whether they are the same as anticipated ones.
- Database:
  Criterion:
  Accuracy of the querying results

Method:

The database will first be tested by creating query. Different situations for the testing will be set and different MySQL query statements will be used to query the database. Responses of database will be checked whether there is any mistake to ensure the accuracy of database.

Criterion:

Reliability of the use of database.

Methods:

Some testing tools like 'JMeter' will also be applied to do some pressure testing for the database.

- User interface:

  Criterion:

  Usability of each page and accuracy of responding and navigating

  Method:

  Static testing will be executed. The layout, style, pictures, fonts etc. of the user interface will be checked whether they meet requirements.

  Dynamic testing will also be executed. Dynamic elements of the user interface like buttons will checked for their responses.

  Navigation testing will be applied. The website will be checked whether each subsite is usable and leaded correctly according to the website map.

  Criterion:

  the interface should be user-friendly

  Method:

  Some volunteers will be invited to experience the use of user interface and they will be required to evaluate the interface whether it is efficient to find each function and whether it is convenient to use.

- Whole system:

  Criterion:

  The system should run as anticipated, each function developed should work normally.

  Method:

  Students who have not participated the project will be invited to experience the use of system. They will be required to create accounts, rate movies, searching for other users and use other functions of the system to check whether the system will run normally.

  Criterion:

  Compatibility of the system

  Method:

  The system will also be displayed on different environment including Windows, OSX and Linux. Different web browsers such as IE11, Firefox and Chrome will also be selected to display the system to test its compatibility.

3 PROJECT PLANNING

3.1 PROGRESS DESCRIPTION

3.1.1 Original Planning

- The database designing is supposed to be accomplished by structuring ER-diagram, data dictionary, and several tables.
- In this designing phase the pseudo code of the key algorithm should be provided.
- The use cases should be provided as a use-case diagram.
- The data structure of the movie information in the database should be designed.
- Pseudo code for the key algorithm and key methods should be provided.
- The website map diagrams should be designed to show the user interface.

3.1.2 Progress Review

- By now our project has finished designing the main part of the algorithm part with pseudo code and a brief frame of the website database with ER-diagram (See 2.4.3) and some interfaces which can be checked in the database design part (See 2.4.4), and other tasks mentioned above as well. With the building of main frame and the Gantt Chart, the direction of what should be implemented next is clear.
- In the next phase, all team members will focus on learning some corresponding background knowledges for the realization phase, and the development of database, interface, ORM as well.
- After reviewing the feedback of the requirement phase, we slightly changed some functions of the movie recommender system which may focus more on the algorithm which will be discussed in 3.2.

3.2 PLANNING CHANGES

After reviewing the progress, we made the following changes to the project:

- The function of the e-mail authentication has been removed.
  The realization of e-mail authentication is complex and we should lay emphasis on algorithm part. The time we planned for the e-mail authentication function has been changed for comparing different artificial intelligence methods used for machine learning and big data implementation.
- The time and frequency of movie recommendation has been changed.
  "The website should provide registered users with new recommended movies lists each time the user rated a new movie." has been changed to "The recommendation movie list is updated at a fixed time every day" for reducing the load of the database in case many users are using the function concurrently.
- The encryption method has been changed from AES to md5 + salt.
  Though AES encryption algorithm is a quick and US-standard method, compared with md5 or SALT encryption, is still not safe enough when facing with the side-channel attacks, especially timing attacks. They attack implementations of the cipher on hardware or software systems that inadvertently leak data. As the database got a huge quantity of user information, using SALT algorithm is more suitable for multiple-hashed data. Usually, the attacker can test the attack to all the accounts with each attempt to see whether there is an account "meet" the attack. However, using SALT, the data can be protected from the precomputed table of hashes. Once the attacker trying his method to all item in database, what he tried is only one item due to other items in the database have different "salt", which is vital to the weakness of some popular encryption algorithms —-rainbow table attacks. In other words, if each password got different salt value, the pre-computation would include each entry for each different salt value. If the salt is large, then attacking from pre-computing hashes is impossible, which is suitable for the database.

4 REFERENCES

[1] X. Su and T. M. Khoshgoftaar, "A survey of collaborative filtering techniques," Adv. in Artif. Intell., vol. 2009, pp. 2-2, 200

[2] D. A. Adeniyi, Z. Wei, and Y. Yongquan, "Automated web usage data mining and recommendation system using K-Nearest Neighbor (KNN) classification method," Applied Computing and Informatics, vol. 12, pp. 90-108, 1// 2016.

[3] Balabanović, M. and Shoham, Y., 1997. Fab: content-based, collaborative recommendation. *Communications of the ACM*, *40*(3), pp.66-72.

[4] Cremonesi, Paolo, Yehuda Koren, and Roberto Turrin. "Performance of recommender algorithms on top-n recommendation tasks." *Proceedings of the fourth ACM conference on Recommender systems*. ACM, 2010.

[5] J. Linwood and D. Minter, Beginning Hibernate, Apress, 2010, p. 118.

[6] D. Rubio, J. Long, G. Mak and M. Deinum, Spring Recipes: A Problem-Solution Approach, Apress, 2014, p. 48.

[7] L. Seinturier and R. Pawlak, Foundations of AOP for J2EE Development, Apress, 2006, p. 100.

[8] K. S. Inc., Struts 2 Black Book, 2Nd Ed (With Cd), Dreamtech Press, 2008, p. 717.

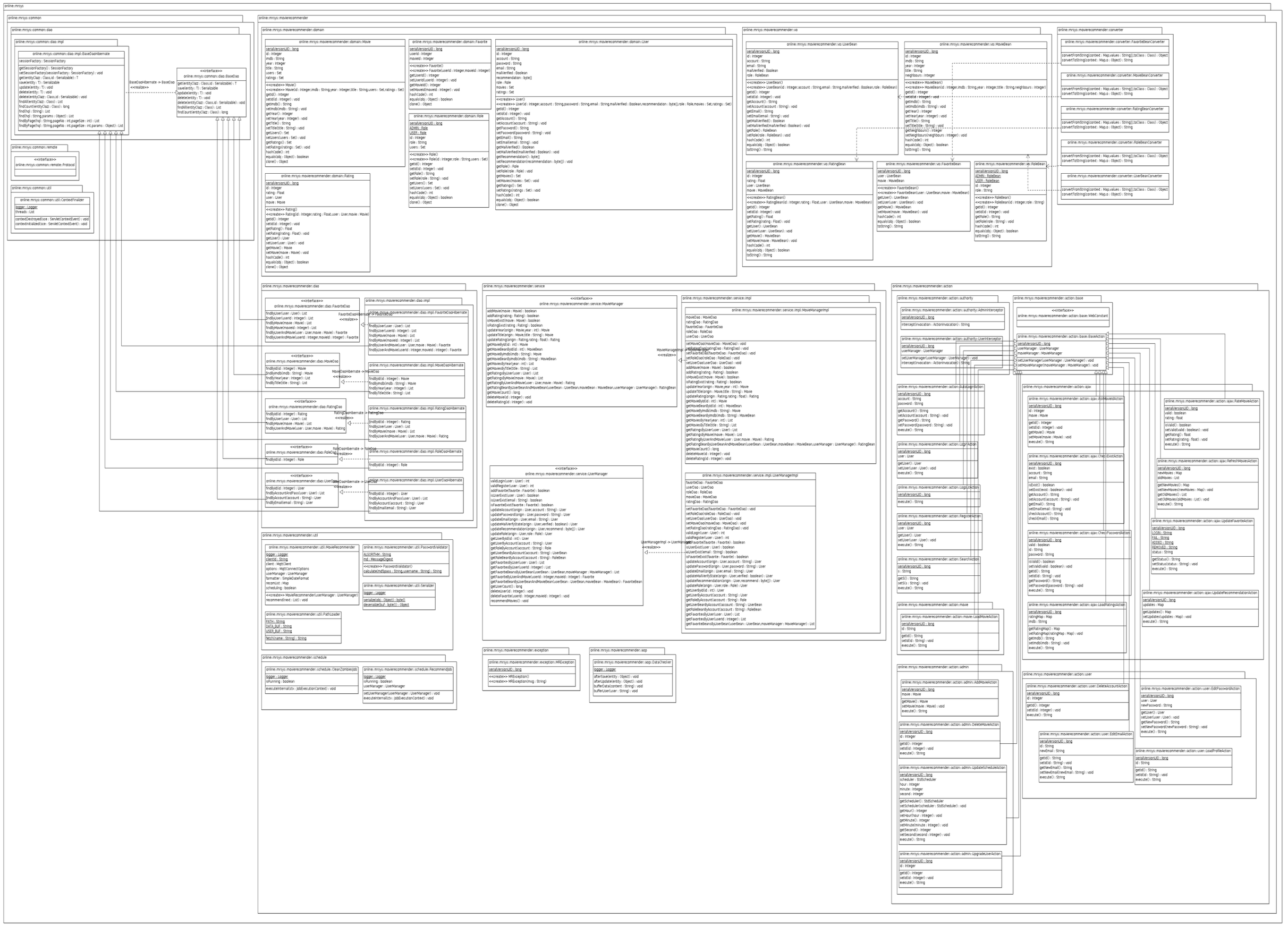
[9] "MQTT," [Online]. Available: http://mqtt.org/.

[10] Chin F Y, Ozsoyoglu G. Statistical database design[J]. ACM Transactions on Database Systems, 1981, 6(1): 113-139.

[11] Teorey T J, Yang D, Fry J P, et al. A logical design methodology for relational databases using the extended entity-relationship model[J]. ACM Computing Surveys, 1986, 18(2): 197-222.

[12] Ramez Elmasri, Shamkant B. Navathe: Fundamentals of Database Systems, 3rd. ed. sect. 17.5, p. 582

5 APPENDIX I: CLASS DIAGRAM



6 APPENDIX II: GANTT CHART

ID	Task Mo	Task Name	Duration	Start	Finish
1		**Requirement Phase**	**12 days**	**Mon 02/06/17**	**Fri 02/17/17**
2		Basic Idea	2 days	Mon 02/06/17	Tue 02/07/17
3		Background Search	1 day	Wed 02/08/17	Wed 02/08/17
4		Realization Testing	2 days	Thu 02/09/17	Fri 02/10/17
5		Features List	2 days	Sat 02/11/17	Sun 02/12/17
6		Functional Requirements	1 day	Mon 02/13/17	Mon 02/13/17
7		Non-Functional Requirements	1 day	Mon 02/13/17	Mon 02/13/17
8		Form-based Specification	2 days	Tue 02/14/17	Wed 02/15/17
9		Requirement Report	5 days	Mon 02/13/17	Fri 02/17/17
10		Requirement Completed	0 days	Sat 02/18/17	Sat 02/18/17
11		**Design Phase**	**26 days**	**Sat 02/18/17**	**Fri 03/17/17**
12		**System Design**	**21 days**	**Sat 02/18/17**	**Sat 03/11/17**
13		System Model Design	2 days	Sat 02/18/17	Sun 02/19/17
14		Use Cases	4 days	Mon 02/20/17	Thu 02/23/17
15		Interface Design	4 days	Fri 02/24/17	Mon 02/27/17
16		Interaction Chart	2 days	Tue 02/28/17	Wed 03/01/17
17		Data Structures Design	2 days	Thu 03/02/17	Fri 03/03/17
18		Algorithms Design	3 days	Sat 03/04/17	Tue 03/07/17
19		Pseudo-code for Key Algorithms	4 days	Wed 03/08/17	Sat 03/11/17
20		Pseudo-code for Key Methods	4 days	Wed 03/08/17	Sat 03/11/17
21		Design Pattern	5 days	Fri 02/24/17	Tue 02/28/17
22		Class Diagrams	10 days	Wed 03/01/17	Sat 03/11/17
23		**Database Design**	**19 days**	**Mon 02/20/17**	**Sat 03/11/17**
24		Data Dictionaries	4 days	Mon 02/20/17	Thu 02/23/17
25		Entity-relationship Diagrams	4 days	Fri 02/24/17	Mon 02/27/17
26		Table Structures Design	11 days	Tue 02/28/17	Sat 03/11/17
27		**UI Design**	**19 days**	**Mon 02/20/17**	**Sat 03/11/17**
28		Web Site Map Diagrams	6 days	Mon 02/20/17	Sat 02/25/17
29		User Interface Design	13 days	Sun 02/26/17	Sat 03/11/17
30		Design Report	5 days	Mon 03/13/17	Fri 03/17/17
31		**Background Knowledge Learning**	**31 days**	**Mon 02/13/17**	**Fri 03/17/17**
32		Recourse Searching	6 days	Mon 02/13/17	Sat 02/18/17
33		Knowledge Learning	25 days	Sun 02/19/17	Fri 03/17/17
34		Design Completed	0 days	Fri 03/17/17	Fri 03/17/17
35		**Realization Phase**	**29 days**	**Sat 03/18/17**	**Thu 04/20/17**
36		**Development Preparation**	**6 days**	**Sat 03/18/17**	**Fri 03/24/17**
37		Development Environment Config	1 day	Sat 03/18/17	Sat 03/18/17
38		Production Environment Configur	1 day	Sat 03/18/17	Sat 03/18/17
39		Server Debugging	1 day	Mon 03/20/17	Mon 03/20/17
40		Database Configuration	1 day	Tue 03/21/17	Tue 03/21/17
41		Web Container Deployment	1 day	Wed 03/22/17	Wed 03/22/17
42		Dependent Libraries Management	1 day	Thu 03/23/17	Thu 03/23/17
43		Git Configuration	1 day	Fri 03/24/17	Fri 03/24/17
44		**Database Development**	**5 days**	**Sat 03/25/17**	**Thu 03/30/17**
45		Table Development	2 days	Sat 03/25/17	Mon 03/27/17
46		Entity Space Optimization	1 day	Tue 03/28/17	Tue 03/28/17
47		Database Debugging	2 days	Wed 03/29/17	Thu 03/30/17
48		**ORM Development**	**3 days**	**Fri 03/31/17**	**Mon 04/03/17**
49		Hibernate Configuration	1 day	Fri 03/31/17	Fri 03/31/17
50		Domain Mapping	1 day	Sat 04/01/17	Sat 04/01/17
51		ORM Debugging	1 day	Mon 04/03/17	Mon 04/03/17
52		**DAO Development**	**3 days**	**Tue 04/04/17**	**Thu 04/06/17**
53		DAO Interfaces List	1 day	Tue 04/04/17	Tue 04/04/17
54		DAO Interfaces Implementation	1 day	Wed 04/05/17	Wed 04/05/17
55		DAO Debugging	1 day	Thu 04/06/17	Thu 04/06/17
56		**Algorithms Development**	**6 days**	**Fri 03/31/17**	**Thu 04/06/17**
57		Algorithm Implementation	2 days	Fri 03/31/17	Sat 04/01/17
58		Algorithm Time Optimization	2 days	Mon 04/03/17	Tue 04/04/17
59		Algorithm Debugging	2 days	Wed 04/05/17	Thu 04/06/17
60		**Service Development**	**5 days**	**Fri 04/07/17**	**Wed 04/12/17**
61		Service Interfaces List	1 day	Fri 04/07/17	Fri 04/07/17
62		Service Interfaces Implementation	2 days	Sat 04/08/17	Mon 04/10/17
63		Service Debugging	2 days	Tue 04/11/17	Wed 04/12/17
64		**User Interface Development**	**16 days**	**Sat 03/25/17**	**Wed 04/12/17**
65		jQuery Configuration	1 day	Sat 03/25/17	Sat 03/25/17
66		Bootstrap Configuration	1 day	Sat 03/25/17	Sat 03/25/17
67		UI Functions List	1 day	Sat 03/25/17	Sat 03/25/17
68		Web Pages List	1 day	Sat 03/25/17	Sat 03/25/17
69		Basic UI Development	6 days	Mon 03/27/17	Sat 04/01/17
70		JavaScript Functions Development	4 days	Mon 04/03/17	Thu 04/06/17
71		JavaScript Debugging	2 days	Fri 04/07/17	Sat 04/08/17
72		Brower Compatibility Fixing	3 days	Mon 04/10/17	Wed 04/12/17
73		**Web Pages Actions Development**	**7 days**	**Thu 04/13/17**	**Thu 04/20/17**
74		Actions List	1 day	Thu 04/13/17	Thu 04/13/17
75		Actions Implementation	4 days	Fri 04/14/17	Tue 04/18/17
76		Actions Debugging	2 days	Wed 04/19/17	Thu 04/20/17
77		Realization Completed	0 days	Thu 04/20/17	Thu 04/20/17
78		**System Testing Phase**	**7 days**	**Fri 04/21/17**	**Thu 04/27/17**
79		Examinations List	1 day	Fri 04/21/17	Fri 04/21/17
80		Structured Testig	4 days	Sat 04/22/17	Tue 04/25/17
81		Issues Fixing	2 days	Wed 04/26/17	Thu 04/27/17
82		System Testing Completed	0 days	Thu 04/27/17	Thu 04/27/17
83		**Documentation Phase**	**4 days**	**Fri 04/28/17**	**Mon 05/01/17**
84		JavaDoc Reorganization	1 day	Fri 04/28/17	Fri 04/28/17
85		User Manuals Writing	4 days	Fri 04/28/17	Mon 05/01/17
86		Design Documentation Reorganizatio	4 days	Fri 04/28/17	Mon 05/01/17
87		Testing Documentation Reorganizati	4 days	Fri 04/28/17	Mon 05/01/17
88		Documentation Completed	0 days	Mon 05/01/17	Mon 05/01/17
89		Project Report	10 days	Tue 05/02/17	Fri 05/12/17
90		Project Completed	0 days	Fri 05/12/17	Fri 05/12/17