## Database and Info Systems

Final Report

Group XYZH

Xiao Hu

Yi Xia

Wenhe Zhang

Hiroyuki Kubota

## CIS 550

Fall 2015

Professor Susan Davidson

1. Introduction
   1. Motivation

The motivation of this project is, for a given database on movies, designing a website that can take user’s key words to provide comprehensive movie information corresponding to user’s inputs. Also as a movie recommendation tool, the designed website would suggest similar movies based on users’ search. Similarities of movies will be calculated based on data provided by the database. To access this website, users need to connect their Facebook accounts (where it is assumed that all users own a Facebook account) with the website. Users can get knowledge on the movies they searched from our website. Additionally, users can make comments on the movie in website pages. Furthermore, they might find similar movies in this website.

* 1. Features

The following features are implemented in the application:

* Search a movie title (if the application finds multiple results, the user can specify release date to find the unique movie)
* Show similar movies to the searched title
* Show detailed information of a movie
* Show an actor or a director information
* Login via Facebook
* Bing image search
* Leave comments

1. Database
   1. Data Cleaning

Majority of our database is based on the TMDB dataset that is provided to us. Original data was in JSON format and was transferred into csv files for simplicity of handling. Furthermore, only those fields that were relevant in our schema design were imported into our MySQL database on AWS. (Shown in Appendix) Along with importing data into our database, we have discovered some issues both in the data itself and in our schema design as well. Most of our design flaws in our schema were misinterpreted data types that we could easily fix along the way. However, the most challenging issue in data cleaning was to deal with those non-English characters that existed in the original TMDB dataset. In order to provide a comprehensive user experience, we finally decided to modify our database encoding format to suit these characters instead of simply throwing them out or replacing with some other random characters. We also removed duplicated data from the dataset and filled in default values for blank fields to ensure smoothness of further iterations.

* 1. Data Structure

Since we have fixed some misinterpreted data type, the final schema is slightly different from those in the previous milestones. Additionally, we replaced our user table with comment table with NoSQL solution. We determined NoSQL is more suitable for the comment table due to the unpredictable variation of comment type and length as well as, in some extremely cases, some less popular movie may never have any comments at all. More detailed relational schema and ER-diagram can be found in the Appendix.

* 1. List of Queries

We used 12 major SQL queries and 1 NoSQL (MongoDB) query in the application. All of those queries are shown in Appendix.

* 1. Optimization

To improve query performances, we used indexes on each table. Since we mostly join tables using the primary key of each table, such as Mid and Pid, we basically created B+ tree indexes on the primary keys. We also evaluated the performance improvement by measuring time to complete each query for 100 times. We used the following command for the query performance evaluation on a Linux machine:

# time ((for i in $(seq 1 100); do echo “{Query}"; done) | mysql --user {username} --host {hostname} –p{password} {dbname} > /dev/null)

Figure 1 shows the results of the performance measurement and the query numbers shown in the diagram refer to the number of the SQL queries in 2.3. Indexing on primary keys did improve performances on most of the cases. As we can see the diagram, the query number 4, 6, 9, 10, 11, 12 show significant improvements of the query execution time. On the other hand, some queries do not show any improvement, for example, we cannot see any improvement in the query #1. This happened because the query #1 needed to search all the tuples in the movie table, since the query used syntax “LIKE” with “%+{key words}+%”.



Figure Query Performance Improvement

1. Score Calculation

An attribute, called score, is created to define the similarity of two movies. In fact, score is used as ‘the score of movie 1 relative to movie 2’, where movie 1 is the movie users searched and movie 2 is the compared movie. The more similar the two movies are, the higher the score should be.

Five important attributes from TMDB are selected to define the movie score. Popularity, release date, county, genre, and keyword are considered as important factors. Since our team members agree on that people compare movies mainly based on these 5 attributes.

Because these five attributes are not all in the same data type, it is necessary to do some transformation and normalization. Data transformation is to make all attributes into numerical data, so that a final numerical movie score can be obtained. Data normalization is to map all numerical data from 0 to 1.

1. Popularity

Popularity is already a numerical data type. Since similar movies should have closer popularity, the sub-score is the 1 over the absolute value of the difference of the two movies’ popularity. In case of that the two movies have same popularity, the denominator of this sub-score is adjusted to he absolute value of the difference of the two movies’ popularity plus 1. Though popularity is considered as an important feature, it is not crucial when comparing two movies. Therefore, the weight of popularity is assigned to be 0.02.

1. Release date

Release date is in data format. It can be directly converted to integer (for example, by EXCEL). As well as popularity, sub-score of this part needs to take the reciprocal of adjusted difference of two release dates. Release date is neither that crucial when comparing two movies, but should be more important than popularity. Hence, the weight of release date is assigned to be 0.1.

1. Country, genre and keyword

These 3 attributes are arrays of strings. To define this part of sub-scores, the intersection (of country, genre or keyword) of the movie users searched and the movie to be compared is needed. The sub-score of country, genre or keyword then equals the cardinality of the intersection over the cardinality of the set of the movie users searched.

Since these 3 attributes are usually widely used when comparing movies, so country, genre and keyword (the word somehow accurately described the movie) are given weights 0.24, 0.32, 0.32, respectively.

The sum of these 5 sub-scores (the product of weight and the numerical sub-score) defines the final score. The top 10 movies, as movies recommended to the users, correspond to the movie the users searched will be shown to the users.

Additionally, since SQL is slower to do such a huge data conversions and calculations, the final scores are computed using R, a statistical software. Furthermore, because of this huge calculations, we could not calculate the score dynamically, so the score data are stored statistically in “Compare” table illustrated in Figure 2.

1. Software Architecture

The software architecture of our applications is shown in Figure 3. As is described in the figure, we use Client-Server architecture for our application, and each element uses the following technologies for the implementation:

* Client
  + Bootstrap

Bootstrap is a front-end development tool for web applications and our application is developed based on Bootstrap. Bootstrap provides web designers with various templates, so the designers can easily implement their web applications. In addition, since Bootstrap automatically adjusts layouts of the web pages based on a screen resolution, the developers do not need to care about the screen resolution during designing. To implement the autocomplete feature, we use “typeahead” extension for the movie search.

* + Arbor.js

Arbor.js is one the data visualization tools for Java Script users. The users only need to call two functions, addNode() and addEdge(), to visualize their data. For our application, we use addNode function to show movie titles and addEdge function to connect two similar movies. We also can specify color and shape of the nodes to enhance visibility of the data.

* Server
  + AWS instance

To host our web application, we created two AWS instances as is shown in Figure 3. For the host server, we select Ubuntu 14.04 Linux, since we can use various tools to make the development more efficiently. We also created a database instance using MySQL to store the movie data.

* + Node.js/Express

Since we learned how to use Node.js and Express in this course, we selected these tools to host our application. In addition, Node.js provides us various extensions to implement various features. We added “Facebook API” and “Bing Search API” to add more features to our application. In other words, “Facebook API” is used to login to our web page and “Bing Search API” is used for searching image data of a movie to obtain more information about the movie.

* + MongoDB

MongoDB is used to store comment data of movies. Since we could not identify primary key for the comment data if use SQL, we decided to use NoSQL to store the data. MongoDB is placed on the same host as the Node.js server. However, we use remote address to access the database, we do not need to change the address of the database even though we place the application on another server.

1. Software Design
   1. Use Case

We identified four main use cases for our application (see Figure 4):

* Login

Users need to login our website with Facebook account.

* Search

Users can search a movie title to find similar movies to the typed one. If the multiple results are found, the users need to specify a release date of the movie to identify a unique movie.

* Show results

The application shows the result of the search to users. The results consist of the following four elements:

* + Searched Movie

The movie typed by the user. The movie is colored in red.

* + 10 movies similar to the searched movie

The movies are colored in green.

* + 2 movies for each similar to the 10 movies

These movies are colored in blue.

* + Edges

Edges describe similarity between two movies.

* Check detailed information of a movie

If the users click a movie title in the result page, the application shows detailed information of the movie. The detailed information includes title, poster, studio, director, rating, revenue, budget, homepage, trailer, casts, and comments. For the director and the actors, the user can find more information about these people from “Detail” link. In addition, the users can also see similar movies to this movie by pressing “Explore Movie Map” button. Furthermore, if the user wants to see more information about the movie, the use can trigger Bing image search for the additional information.

* 1. Modules

Figure 5 shows modules included in the application. For the client side, basically each module represents one page. On the other hand, the server side consists of six main modules, which are mainly designed for obtaining required data from the database.

* 1. Page Transition

Our application consists of the following nine pages:

* Home

This is the first page for the users to access to our application.

* Login

Login page uses the Facebook login page for authentication process.

* Search

The search page consists of a text box and a button to type and to search a movie.

* Select Movie

If the application found multiple results during the search process, the application shows this page to the users. The users need to specify a release date of the movie to identify a unique movie title.

* Search Result

As is mentioned in 5.1, the application shows the search result, which consists of the four key elements, to the users. The users can select a movie from this page to check more detailed information of the movie. Figure 7 shows the interface definition between the client and the server.

* Detailed Information

When the users clicked a movie title in the search result, the application shows detailed information of the selected movie. The information shown in this page are described in 5.1. In addition, Figure 8 shows the interface definition between the client and the server.

* Director’s Information

When the users selected the link to show the director’s information, the application will lead the users to this page. The page includes, the name of the director, the date of birth, the profile, and the list of the movies directed by the person. Figure 9 shows the interface definition between the client and the server.

* Actor’s Information

Similar to the director’s information page, the application will show this page when the users clicked the link for showing the actor’s information. The page includes, the name of the actor, the date of birth, the profile, the list of the movies in which the person acted, and the character name. Figure 10 shows the interface definition between the client and the server.

* Bing Image Search Result

When the users pressed the button on the detailed information page to obtain more information about the movie, the application shows search results of Bing Image Search. The application shows top 48 images related to the movie with source links. Thus, the users can move to external web pages by clicking the images. The information sent from the server to the client is the raw data from Bing Image Search API, which are described in JSON format.

The page transition of the application is shown in Figure 6 using state transition diagram.

* 1. Operation Sequence

Our application needs six main operations to use. The sequence diagrams of these operations are shown in Appendix (see from Figure 11 to Figure 16).

1. Responsibility

* Xiao Hu
  + Database Maintenance
  + Data Cleaning
* Yi Xia
  + Data Structure Design
  + Algorithm and Search Design
* Wenhe Zhang
  + Movie Scores Calculation
  + Query Optimization and Indexing
* Hiroyuki Kubota
  + Overall design of the application
  + Implementation of application

1. Future Extensions
2. Edit and Delete functions for comments

Users can only leave comments for movies in the current version, so we should add edit and delete functions for users who want to modify their comments after the submission.

1. Movie Recommendation Based on Users’ Search History

Our application could store the user’s search history into the database, and recommend more similar movies based on our similarity function and user’s inputs.

1. Friend Recommendation (Based on Facebook)

Based on our similarity score and user’s search history, our application could also find different people that have similar tastes in movies. We can also detect the words and images people posted in Facebook, use some machine learning algorithm to find their tastes in movies, and recommend friends to people that love similar movies.

**Appendix**

**Schema Design**

MySql:

Movie (MId: int(11), title: varchar(255), popularity: double, rdate: date)

CountryLanguage (Mid: int(11), Country: varchar(255))

Has (Mid: int(11), Keyword: varchar(255))

Contain (Mid: int(11), Genre: varchar(255))

Details (Mid: int(11), rating: double, trailer: varchar(255), revenue: double, HP: varchar(255), budget: double, poster: varchar(255))

Compare (Mid: int(11), CMid: int(11), score: double)

Studio (Sid: int(11), name: varchar(255), HQ: varchar(255), logo: varchar(511))

Created\_by (Mid: int(11), Sid: int(11))

Child\_of (PSid: int(11), CSid: int(11))

Person (Pid: int(11), name: varchar(255), Dob: date, Profile: varchar(255))

Act (Pid: int(11), Mid: int(11), CharacterName: varchar(511), OrderNum: int(4))

Work\_on (Pid: int(11), Mid: int(11), department: varchar(255), job: varchar(255))

NoSql:

Comments {‘Mid’: (Movie ID) , ‘comment’: (User Comments))

**ER-Diagram**

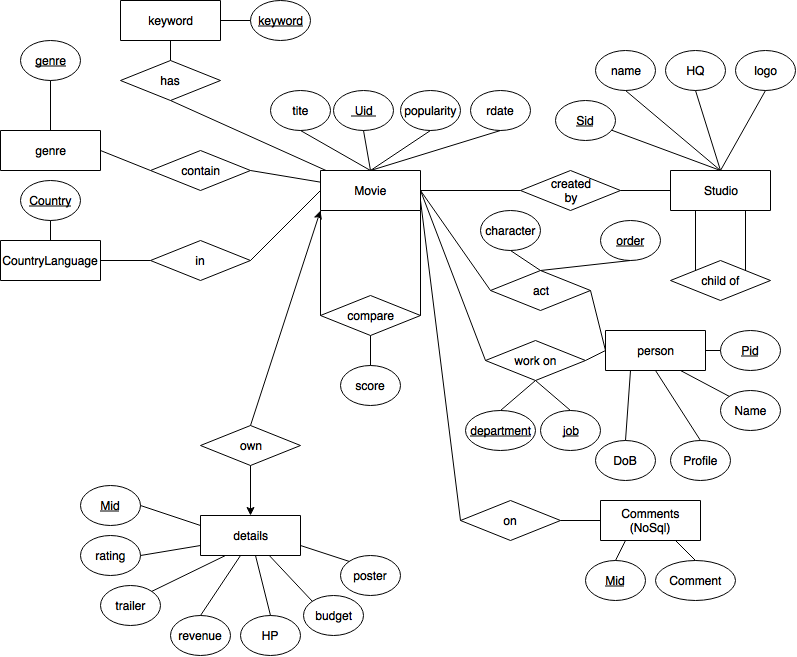
****

Figure ER- Diagram

**Queries used in the application**

1. Autocompletion for searching
2. Autocomplete for searching a movie title

SELECT title from Movie where title like "%'+{Typed Words}+'%";

1. Construct the search result
2. Search typed movie title

SELECT mid, date\_format(rdate, '+"'%m/%d/%Y'"+') as reldate from Movie where title="'+{Typed Title}+'";

1. Search a movie from the movieId (When the above query returned multiple titles)

SELECT mid from Movie where mid="'+{Mid}+'";

1. Explore similar movie titles from the typed movie title

SELECT CMid from Compare where Mid="'+{Mid}+'";

1. Explore similar movie titles from the mids obtained from the above query (limit to 2 tuples)

SELECT CMid FROM Compare WHERE Mid="'+ {Mid} +'" LIMIT 2;

1. Search movie title from mid

SELECT mid, title from Movie where Mid="'+ {Mid} +'";

1. Construct the detailed information about a movie
2. Search title, rating, trailer, revenue, HP, budget, poster from mid

SELECT M.title, D.rating, D.trailer, D.revenue, D.HP, D.budget, D.poster

FROM Movie M INNER JOIN Details D ON M.Mid = D.Mid

WHERE M.Mid="'+{Mid}+'";

1. Search Studio name

SELECT S.name as studio

FROM Created\_by CB INNER JOIN Studio S ON CB.Sid = S.Sid

WHERE CB.Mid="'+{Mid}+'";

1. Search director’s name, DoB, profile (image of the director)

SELECT DR.Name as dname, date\_format(DR.DoB, '+"'%m/%d/%Y'"+') as ddob, DR.Profile as dprofile, DR.Pid as Did

FROM Directors DR INNER JOIN Person P ON DR.Pid = P.Pid

WHERE DR.Mid="'+{Mid}+'";

1. Search Actors’ name, DoB, profile (limit to 14)

SELECT P.Name as actor, date\_format(P.DoB, '+"'%m/%d/%Y'"+') as bdate, P.Profile, A.Pid as Aid

FROM Act A INNER JOIN Person P ON A.Pid = P.Pid

WHERE A.Mid="'+{Mid}+'" ORDER BY A.OrderNum ASC LIMIT 14;

1. Obtain person’s information
2. Search director’s information

SELECT P.Name, date\_format(P.DoB, '+"'%m/%d/%Y'"+') as bdate, P.Profile, M.Mid, M.title

FROM Person P INNER JOIN Directors D ON D.Pid = P.Pid

INNER JOIN Movie M ON M.Mid=D.Mid

WHERE P.Pid="'+{Pid}+'" ORDER BY M.popularity DESC;

1. Search actore’s information

SELECT P.Name, date\_format(P.DoB, '+"'%m/%d/%Y'"+') as bdate, P.Profile, M.Mid, M.title, A.CharacterName

FROM Person P INNER JOIN Act A ON A.Pid = P.Pid

INNER JOIN Movie M ON M.Mid=A.Mid

WHERE P.Pid="'+{Pid}+'" ORDER BY M.popularity DESC;

1. Write and Read Comment Data (NoSQL queries)
2. Write comment data

db.collection('comments').insert({Mid:{Mid},comment:{Comments}})

1. Read comment data

db.collection('comments').find({Mid:{Mid}})

**Architecture**



Figure 3 Software Architecture

**Design Documents**

* Use Case

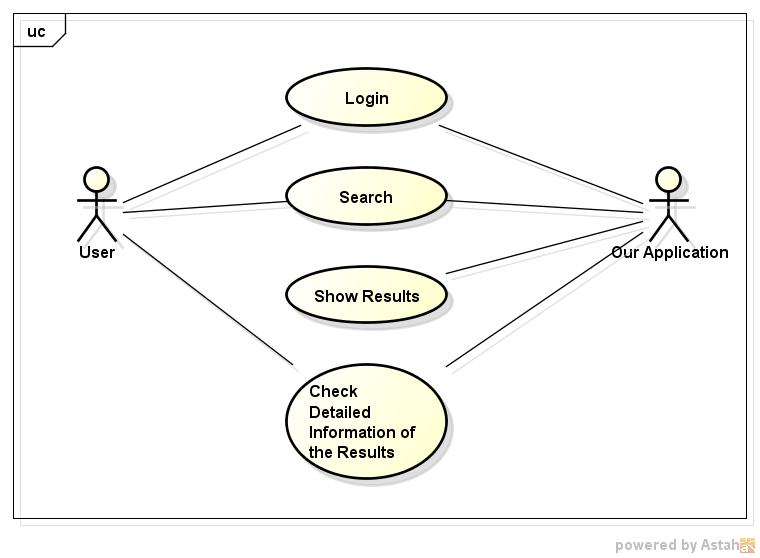


Figure 4 Use Case of our Application

* Modules

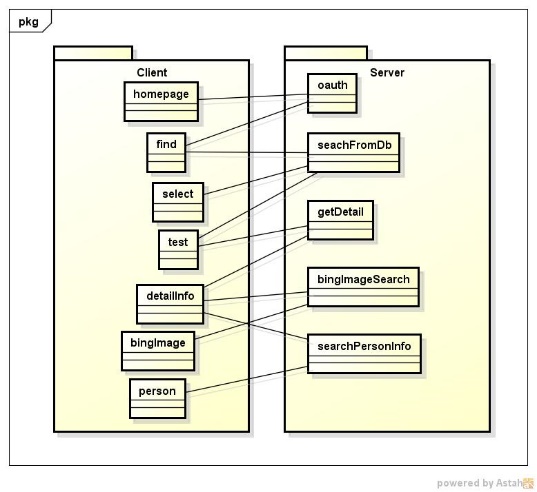


Figure 5 Main Modules

* Page Transition Diagram (State Machine Diagram)

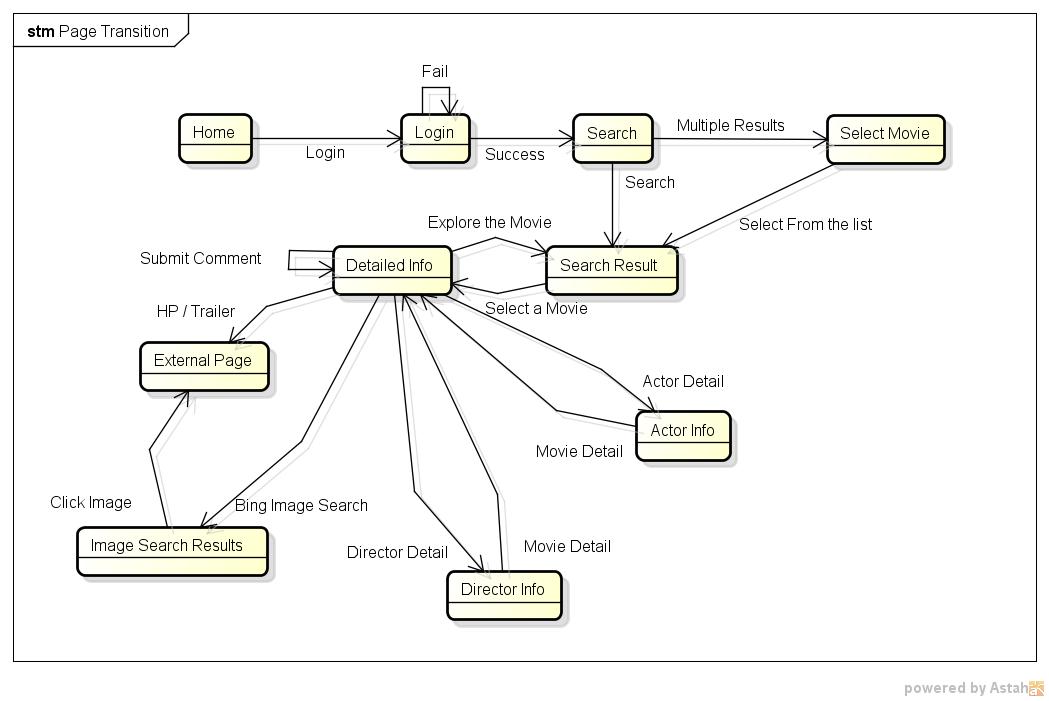


Figure 6 Page Transition

* Interface Control Document



Figure 7 Interface Definition of Search Result



Figure 8 Interface Definition of Obtaining Detailed Information



Figure 9 Interface Definition of Obtaining Director's Information



Figure 10 Interface Definition of Obtaining Actor's Information

* Sequence Diagrams

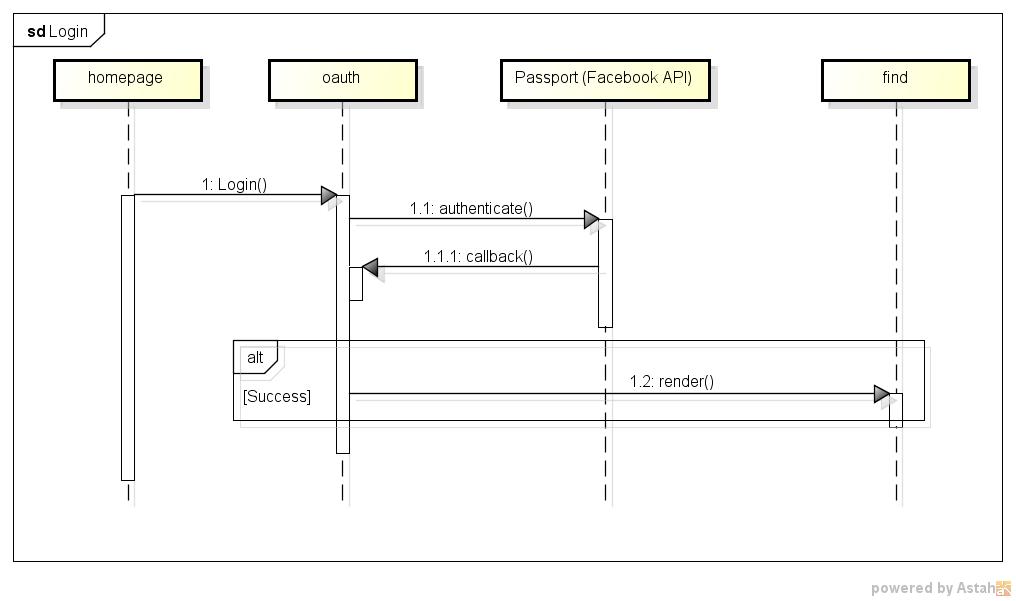


Figure 11 Login Sequence

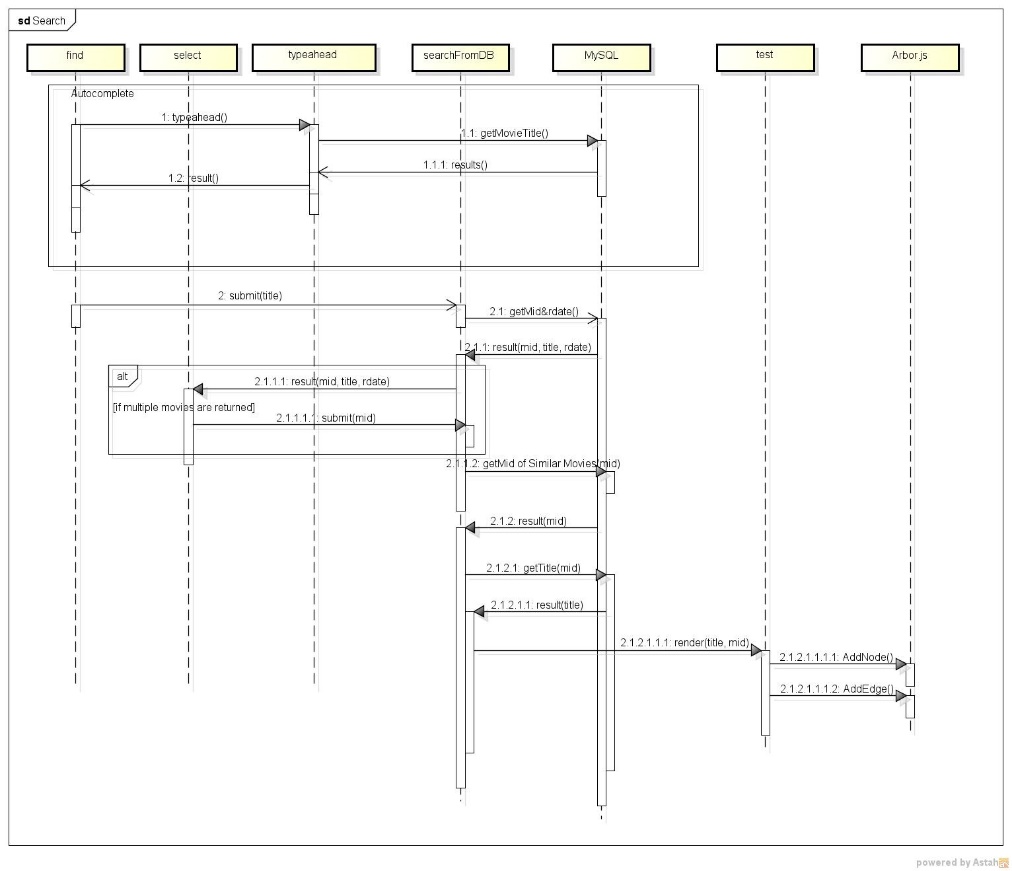


Figure 12 Search Sequence

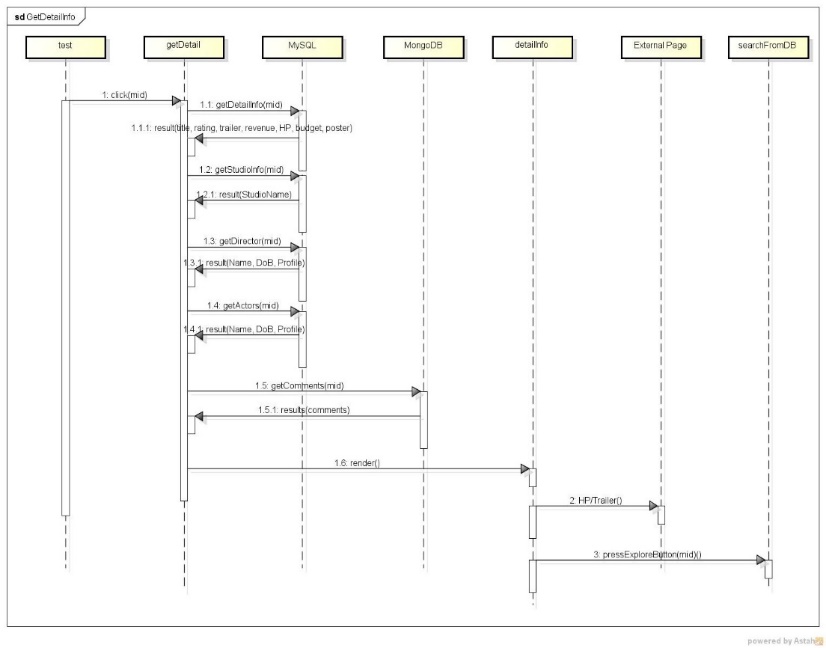


Figure 13 Obtain Detailed Information of a Movie

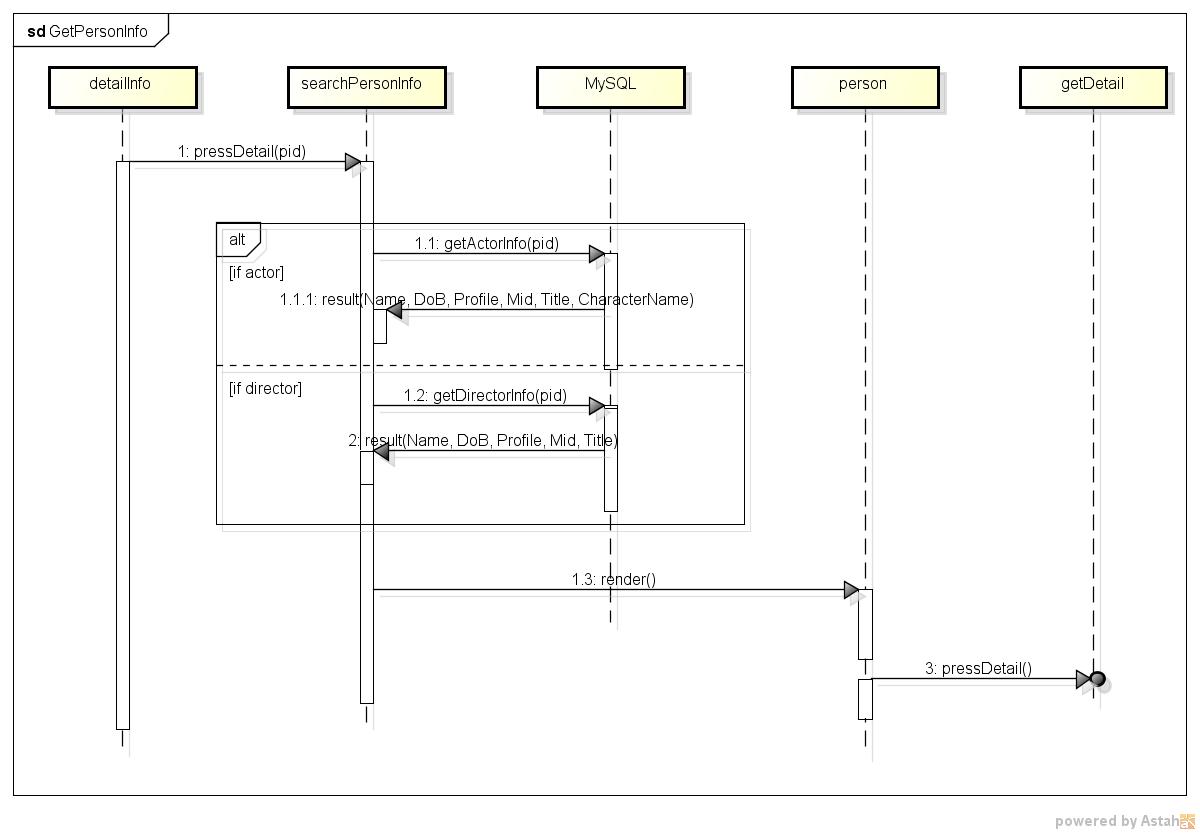


Figure 14 Obtain Person's Information

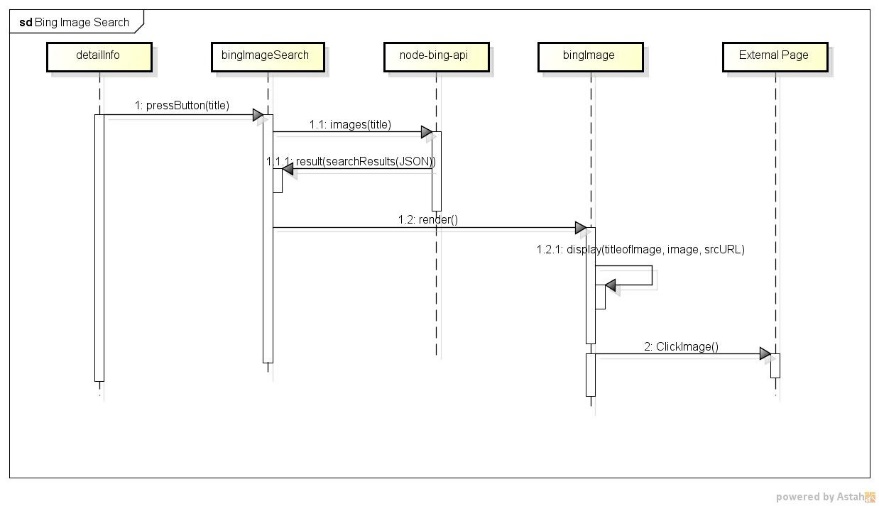


Figure 15 Bing Image Search Sequence

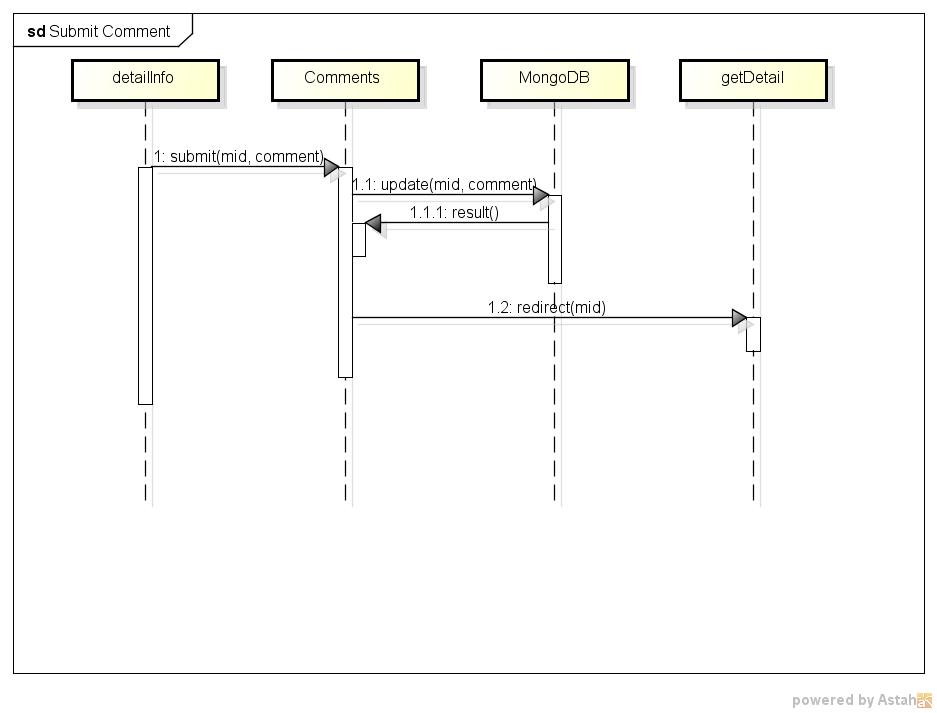


Figure 16 Submit a Comment Sequence

**Note about the application**

[Address on AWS]

<http://ec2-52-20-74-142.compute-1.amazonaws.com:9000>

\* If the page cannot be found, please contact us.

[Modules]

Please refer to 5.2.

[Requirement]

1. node.js

2. Permission for accessing to localhost:3000

\* We only tested the application Ubuntu 14.04 with node.js

[How to use]

1. Extract the application to your local host.

2. Change directory to root directory of our application.

3. Execute the application with the following command:

# node www/bin

4. Access to the following address:

http://localhost:3000

[Note]

1. Since we use Facebook API for login, only authorized people can login to the page when you deployed the application to a local environment (localhost). If you want to login to the application with your local host, please contact us.

2. All the DB are placed on AWS, so you don't need to change the address for the DB in the codes.