

**FINAL TECHNICAL REPORT**

**MOVIE BANK**

**Course:** Advanced Systems Programming – 03-60-656

**Professor:** Dr. Ziad Kobti

**Group ID:** 6

**Group Members:**

Hang Zhou - 104357381 Jimei Peng - 104494147

Juzhou Zhang - 104557945 Yilin Liao - 104380332

Yuze Zou - 104526851

**Date:** March 28, 2017

**CONTENTS**

[**Abstract 3**](#_Toc478470186)

[**Introduction 4**](#_Toc478470187)

[**Requirement Analysis 5**](#_Toc478470188)

[**Design 6**](#_Toc478470189)

[**Implementation 7**](#_Toc478470190)

[**Testing 9**](#_Toc478470191)

[**Prototype 9**](#_Toc478470192)

[**Conclusion 14**](#_Toc478470193)

[**Acknowledgment 15**](#_Toc478470194)

[**Reference 15**](#_Toc478470195)

[**Appendix 15**](#_Toc478470196)

# Abstract

Movie is the one of good art performance methods. It can record different lifestyles in different countries, diverse periods’ cultures, various concepts of value even the modes of thinking. Since movies can spread to all around the world, audiences will gain many inspirations, knowledge, and broad horizons. Movie has become a tendency in people’s live. Because of this tendency and opportunity in the market, our term was planning to develop a tool which could search movies’ information for users, and could compatible with different platform, such as mobile phone, laptop. In general, the goals of our team were to develop a server which can control all the connection in the background, establish a database which server can exchange data and store data inside, and create two different client’s platforms (Shell and Android) which can send users’ requirement to server and get the return data result from server. Our team was using Unix as operation system to develop server and Shell client, and using Eclipse as Android developing tool. During almost two months, our term has finish this project which is call the Movie Bank. All functions of this project exactly meet our specification.

# Introduction

The project which our team developed is called the Movie Bank. There are three sides which this project includes, namely client- side, server- side and database. The goal of the project is to implement client-sides which can input the searching information from users and get output results from server, server-side which can handle the keywords from user, search related information from movie DB and return the correct information to client-sides, and database which can store the related movies’ information. The constrains is that the project will use local movie data resources which were downloaded from the Internet so that we cannot apply big data concept in our project. It means that this project would probably cannot cover all movies all around world. However, we will still collect the movie information as much as possible.

There are two types risks of the project: business risks and technical risks. In terms of business, our group do not make enough market investigation that do people want to use the project that we make. Furthermore, we do not sure that our project can copy with all needs of users. Therefore, our group should have some other feedback researches which not only can improve the functions of our project, but also it can help us to accurately know the business marketing trends. Technical risks are the second type of risks that we need to deal with. Firstly, our group use local movie data resources as our database, which do not implement big data concept. It is obviously that the data is limited. Secondly, our group members have different educational background so that each member has different capacity. At the same time, some techniques exceed the knowledge that we have. It could slow our project progress. Thus, for increasing our progress, we would combine learn and implementation together.

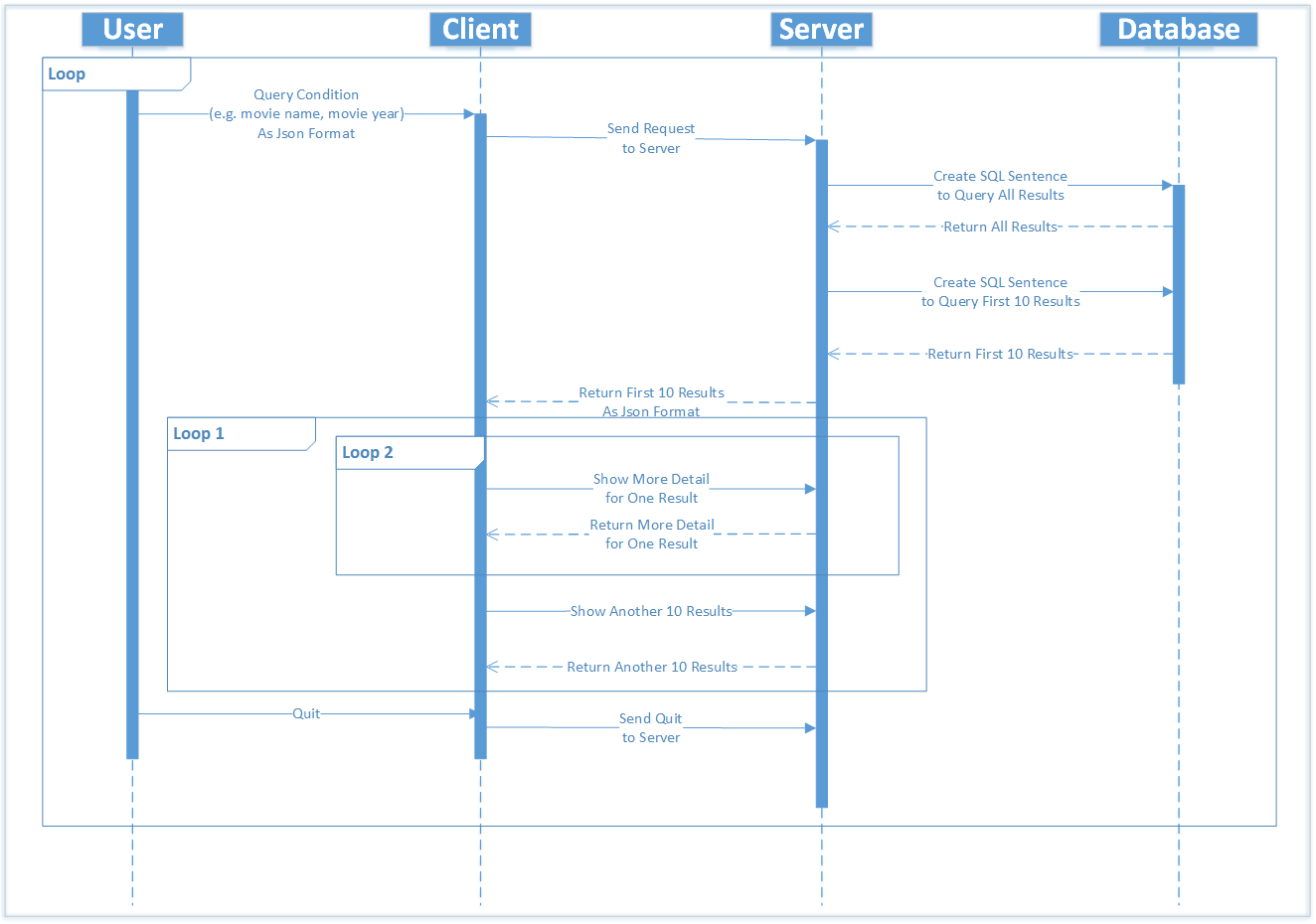
This paragraph will explain how our group can overcome the work risks, and keep tracking our timeline. Our group has five members so that each member will be assign different workload. This means that some parts of work may easy and some may hard. In the end, each member spends different time to finish works. Hence, our strategy is members who finished their works will help someone else who have not finished. There are three phases crossing the whole project developing: inception, implementation and debugging and deployment. Inception was finished in one week. Implementation and debugging would cost three weeks and the last one week for deploying the entire project.

The entire project is based on the UNIX as system operation. Using the C language as server and the Shell client developed language, we apply the Android as another client developed language. Meanwhile, to make sure every phase accurate and effective, Redmine is chosen to record the project management, and Github is for project version control.

# Requirement Analysis

Users input query condition, such as movie name, through interface. The client side collect the query and send request to server which create SQL sentence to query all results. The DB returns the numbers of all results to server. Then the server requests and gets the first 10 results from DB. After that the first 10 result is sent to client. The client requests to show more details for one result and get the details, which can be done repeatedly as long as the user want. At the same time, if client may ask to show another 10 results, the server returns 10 more results to client, which can be repeated as well. Finally, there are two options. One is to create a new query which will begin from the beginning. Another is that the server quits servicing this user when the user sends a quit request.

The sequence diagram below shows all requirements.



# Design

* Design overview

The whole project is divided into three parts, the server, the shell client and the Android client. In terms of languages, data format and the way to communicate, C language is chosen to establish the server, and C utility on shell is used to connect with shell. Database is established by MySQL, and the Android client is established by Java. Moreover, the server and the shell client through socket, and JSON format message is used in communication between server and clients. The platform for the server and the shell client is Unix, and Android platform is for the Android client. As for the testing, we create more connections of client from different platform to test whether the server can copy with those clients and do not crash.

* Server

Every time one client connects to server, the server duplicates a child process to deal with the client. A child process receives messages which are inputted key words from a client, and uses those messages to search the proper movies while limiting the ten movies. As a result, the child process sends these ten movies to the user. Moreover, the child process deals with requests, such as, sending more details for one record, show more records and quit.

* Client

The client side offers 8 different kinds of field for users in both of Shell and Android to input their key words: movie title, genres (adventure, animation, comedy, family, fantasy, musical, romance, action, sci-fi, mystery, drama), actor/actress 1, actor/actress 2, actor/actress 3, director, from year, country.

The first output is ten movies, then the Shell one provides an option that to get more information of one specific movie or show the next ten movies; similarly, the users can scroll to load more or click the link to see more about one movie on the Android client. There are differences between the Shell client and the Android client while providing the movie details. The Shell side lists all the information, such as duration, plot keywords, IMDb movie links of chosen movie from movie DB, while the Android one covers movie names, genres, years and IMDB movie links which links to all the information for the movie.

# Implementation

Our project consists of four parts: MySQL database, back-end server, CLI client side and Android client.

We choose MySQL as our database, which is a popular open-source relational database management system. All the movie information comes from Kaggle, which was founded as a platform for predictive modelling and analytics competitions. In addition, Kaggle has a huge community of data scientists who participant in machine learning competitions. The original dataset was in CSV format, which can be easily viewed by any text editors or Microsoft Excel. This dataset contains about 15,000? rows of IMDb movies. What we did then, is creating a database schema and table structure, and importing all the dataset in CSV format to the database. MySQL database could be deployed in any operating systems and any machines. It does not have to be in the same machine with back-end server as long as server side has the correct MySQL connection string.

At server side, an industry strength make file was provided. We declared many variables so that we can modify any of variables easily. These variables including the compiler name, the compile-time flags, the header files directories, and library paths, etc. The reason why we declared these information as variables in make file is that we used third part libraries, as such JSON C library which deals with JSON data format, and MySQL C driver which was used to connect with MySQL database in C programs. In different platforms, these libraries may be installed in different directories, and our server code depends on those libraries. Therefore, it is a good practice to make those variables configurable, so that our program can be easily migrate to any other platforms just by configuring the make file.

In our server side, we wrote two source files, dealing with MySQL connection and queries, and sockets communications, respectively.

The source code called “mysqlapi.c” mainly deal with connection and queries with MySQL database. Thanks to the modular programming conception, the detail code can be hide in functions to archive maintainability and reusability. We wrote the “setup\_mysql” function, which establishes connection with MySQL. In addition, “close\_mysql” and “finish\_with\_error” were written, closing the database connection and dealing with connection error, respectively. Furthermore, “query\_mysql” function is issuing SQL query statements and get result set from database. This function accepts the movie querying condition as a parameter in JSON data format. We extract all the conditions from JSON format and make SQL query statement. We also query the total rows which satisfies the query and then make pagination, which means our program can not only return 10 rows each time due to the huge bytes for each of them, but also our program can navigate to specific page number as long as the page number is smaller than the total page. This functionality is implemented by using MySQL pagination SQL. After successfully having the result set from database, we convert those result set to JSON data format and return this JSON object so that socket file can send this message to client side.

The server side socket communication source file named “movieServer.c” is mainly deal with socket. It is a classical socket server with fork model. It first establishes server side socket, and then binds the IP address and port, and listens the socket file descriptor, and accepts connections from client sides in a loop, and then fork function is called. The children processes deal with the connections with client sides, and the parent process comes back and accepts new connections. In this model, the program can deal with multiple client connections simultaneously.

Children processes deal with connections in “doprocessing” function. It reads the request message from client side in JSON data format, and send to “messageHandler” function and then get result message in JSON format. After that, it sends result message to the client side. This process is in the loop until the server receive the quit message from the client side or the client side closes the connection without any message.

In the “messageHandler” function, it converts the received message from a string to a JSON object, and send to the function “query\_mysql” in “mysqlapi.c” source file to get the result from database.

It is worth pointing out a better way for server and client to transfer message by using “writen” and “readline” functions instead of default system call write and read. The data has been transferred between server and client could exceed 8 kb. And we sometimes make server and client connections in different part of campus. Due to the large size of the messages and the instability of networks, the system calls write and read cannot send and receive the whole messages. In order to solve these problems, the industry strength “writen” and “readline” have been used in our project. The “writen” function accept socket file descriptor, the message pointer, and the size of message. This function will continuously write message to the socket until all the bytes have been sent. Once finish, this function returns the size of bytes has been sent. The “readline” function is similar to “writen” function, this function continuously read message from socket until it read the end of file or the new line character “\n”. This function returns the bytes read from socket. Both functions promise to send and receive all the message regardless of the size of messages and the condition of networks.

Our Linux client side is straight forward. It requires server’s IP address and port to run the program. Once the client side started, it will establish the connection with server by using socket. And then, the main menu will be shown, users can input some some condition and search the movies. Once users finish inputting conditions, the client side program will convert the users’ input into JSON data format and send to server side using socket. After getting server’s respond, the program will parse the message in JSON format, and extract information and show the results in the screen. User can input the index of the movie to see the details about the selected movie. In addition, users could see the results in the next page if the total results in server side is greater than 10 items. In this case, the client side will make another request with the increased page number parameter to the server, and get the continue results.

Our Android client does the similar thing as Bash shell client. It first establishes the connection with server by means of sockets. On the first page, it reads the users input and form a JSON message format. And then, it sends the request message to the server, and gets the results message in JSON format. The message transmission is dealing in an asynchronous task which runs on a background thread. So the data transmission process will not influence the UI thread, thus providing a great user experience. After the background asynchronous task finish, it will notify the UI thread and a new UI screen will be shown with the results from server side.

A fantastic scroll down refresh component is used in Android client. The server only sends 10 items each time. When users scroll down the screen to the end of last item, it will make a new socket request to the server. Once client gets the new results, they are appended to the end of the current list.

# Testing

We used some black box and white box test in our projects.

We place the server and the clients in different locations of the campus in order to test connectivity. We found the system call “write” and “read” are not very stable, so we make “writen” and “readline” in both server and clients. Furthermore, the clients send many request with different search conditions, and clients got the expected results.

# Prototype

* Server:

When a client connects to the server, server forks a child process whose PID is 50915.

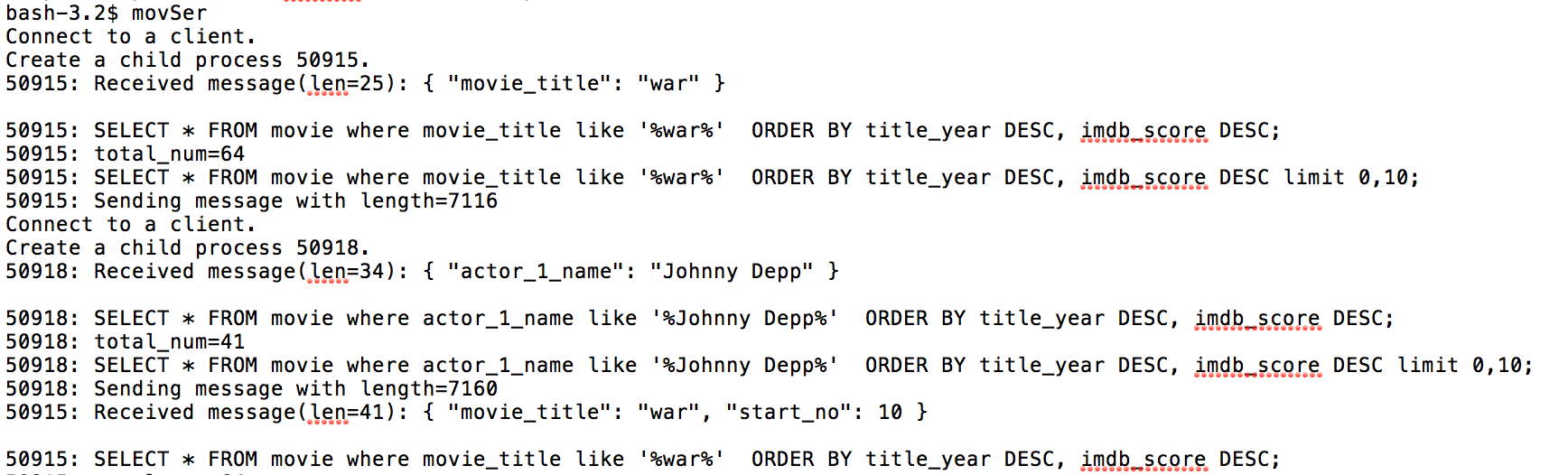
And then this child process will receive JSON format message which the user requires obtaining movies whose title contain a keyword “war”.

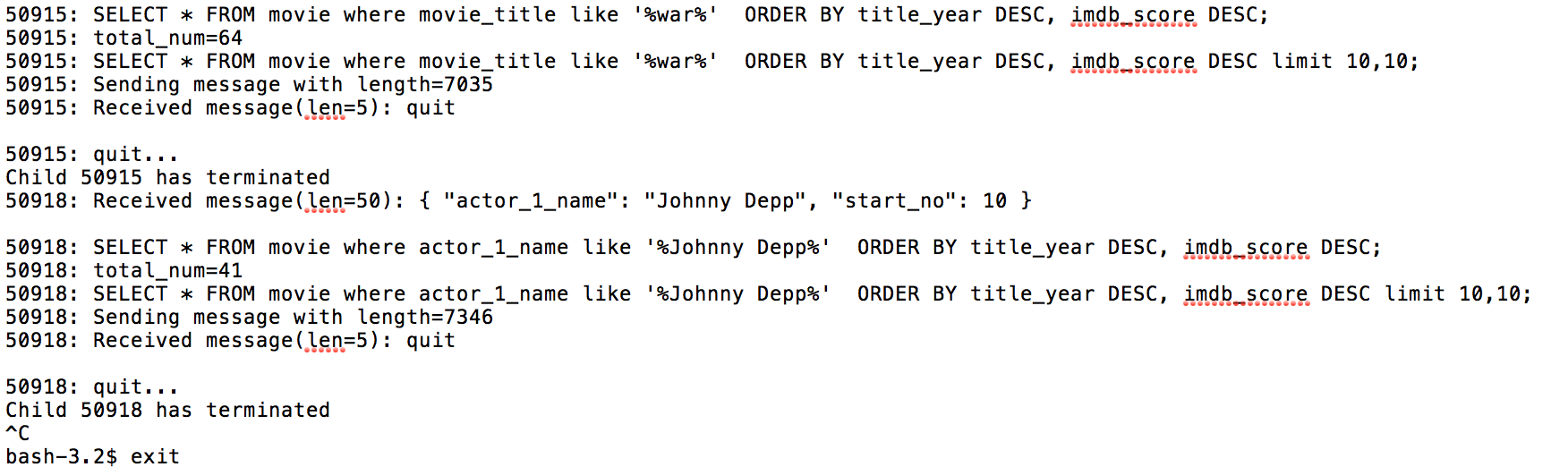
Display SQL sentence and the total number of the result. (The result is in descent order based on year and IMDB score. By default, first 10 movies will be selected if there is no “start\_no” in the message.)

The server sends the result back to the client.

A new client connects to the server, and execute similar procedures 2) to 4).

The server receives “quit” message from a client and terminates the child process.





* Shell client:

Shell client provides 8 different conditions for users to choose movies they want. These conditions include “movie title”, “genres”, “actor/actress”, “director”, “from year” and “country”. In our example, we would like to choose the movies whose titles contain “war” and then input “y” to start searching

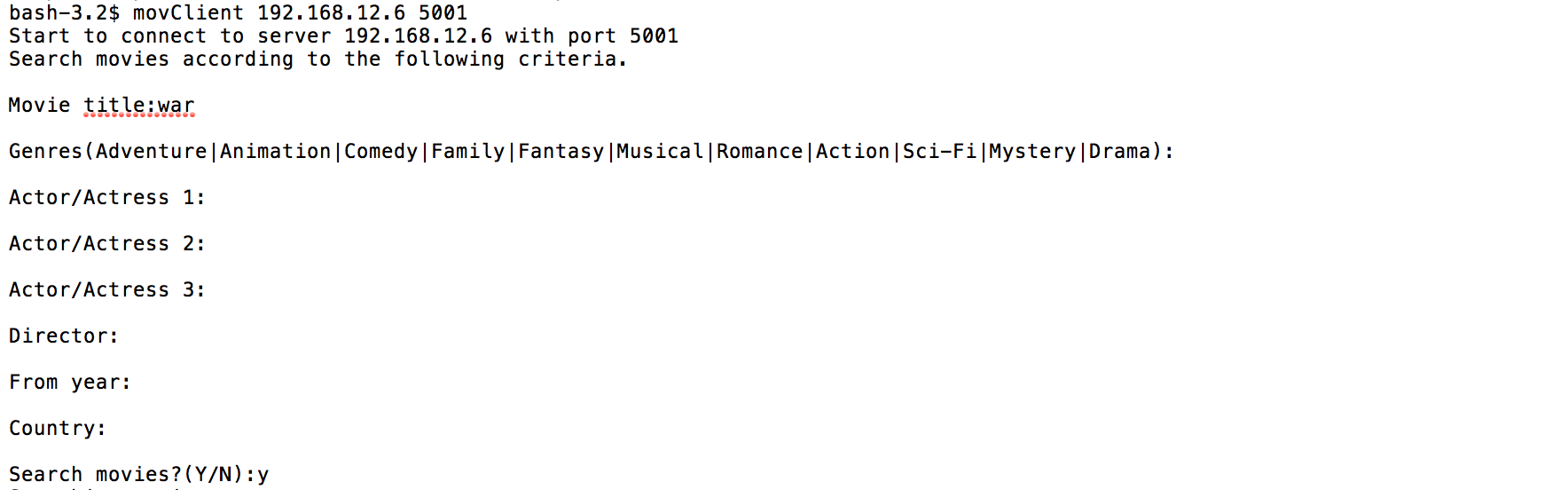
After inputting conditions, contents will be formatted as JSON and sent to the server.

The client gets 10 movie records from the server and displays them on the console.

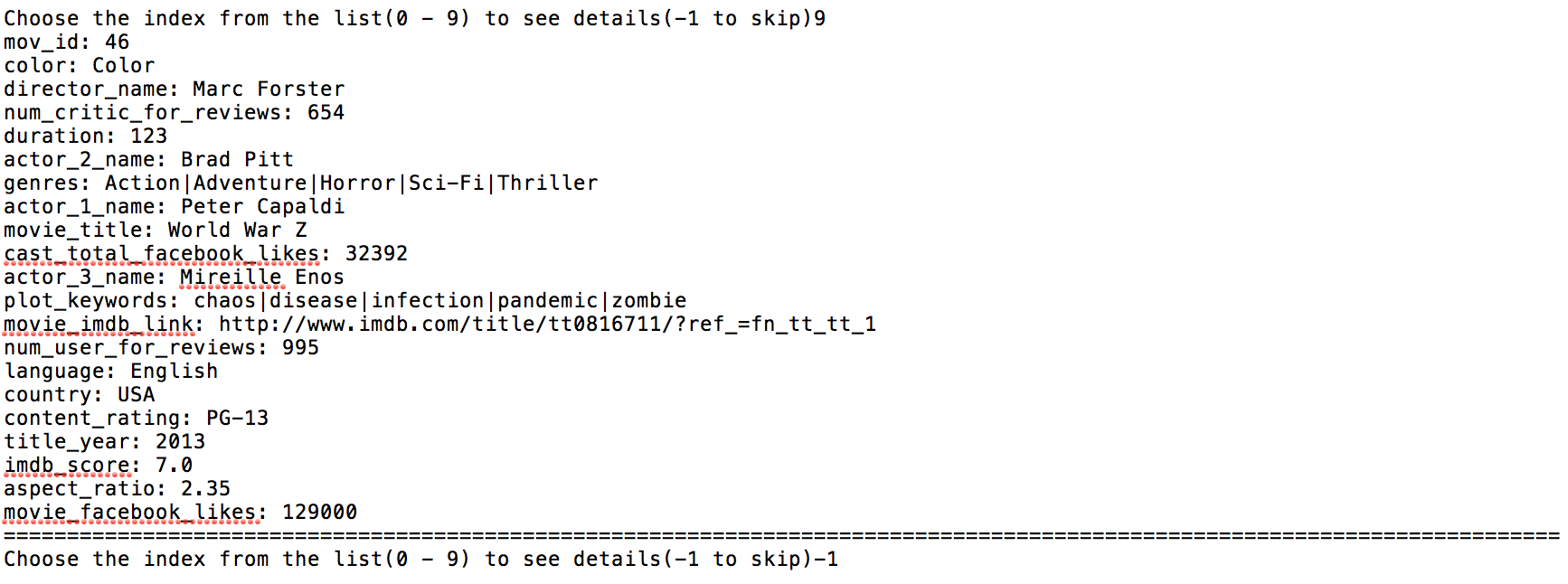
The client asks the user to input an index to see more details of the related movie or -1 to skip. User input ‘9’ to see more movie details.

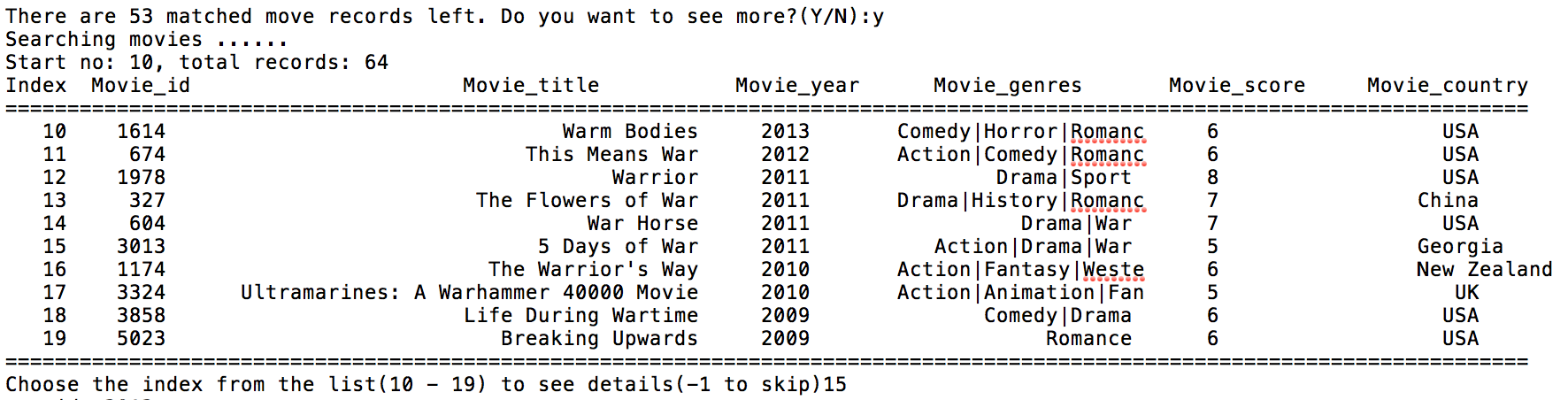
The client asks the user to input “y” to see more movie records or “n” to stop displaying records.

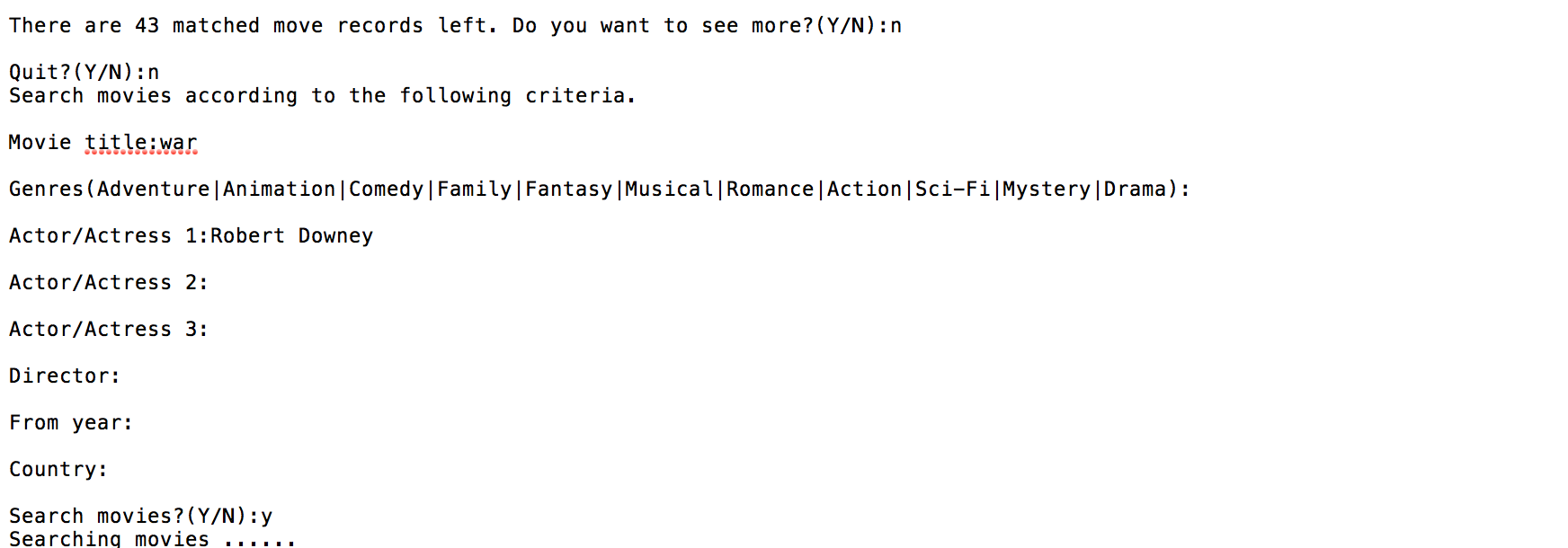
The client asks the user to input “y” to exit or “n” to start a new searching.

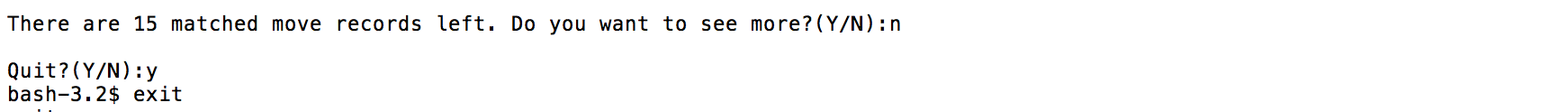










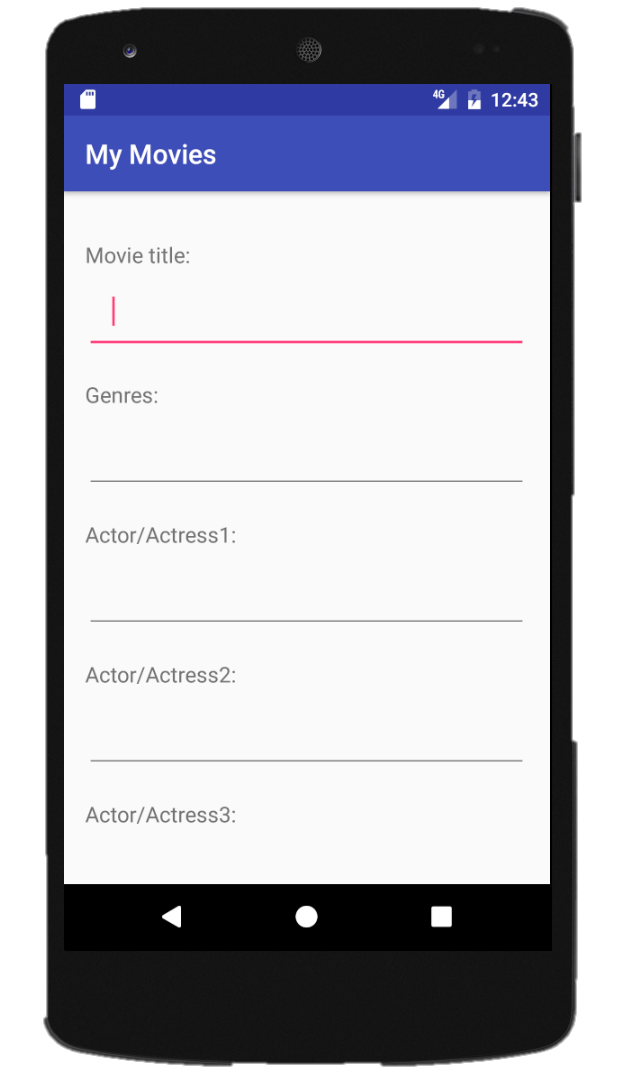
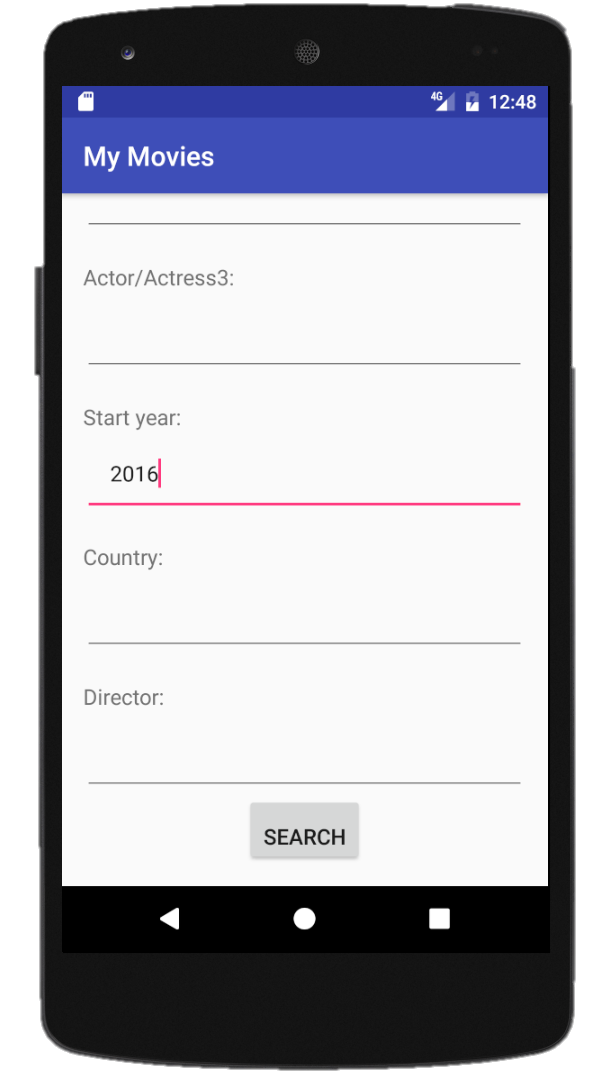


* Android Client:

Filter Page:

Similar with shell client, filter page also contains 8 input fields that allows users to type specific conditions. In following example, we typed “2016” as the “start year”.

Click the search button to send request to the server.

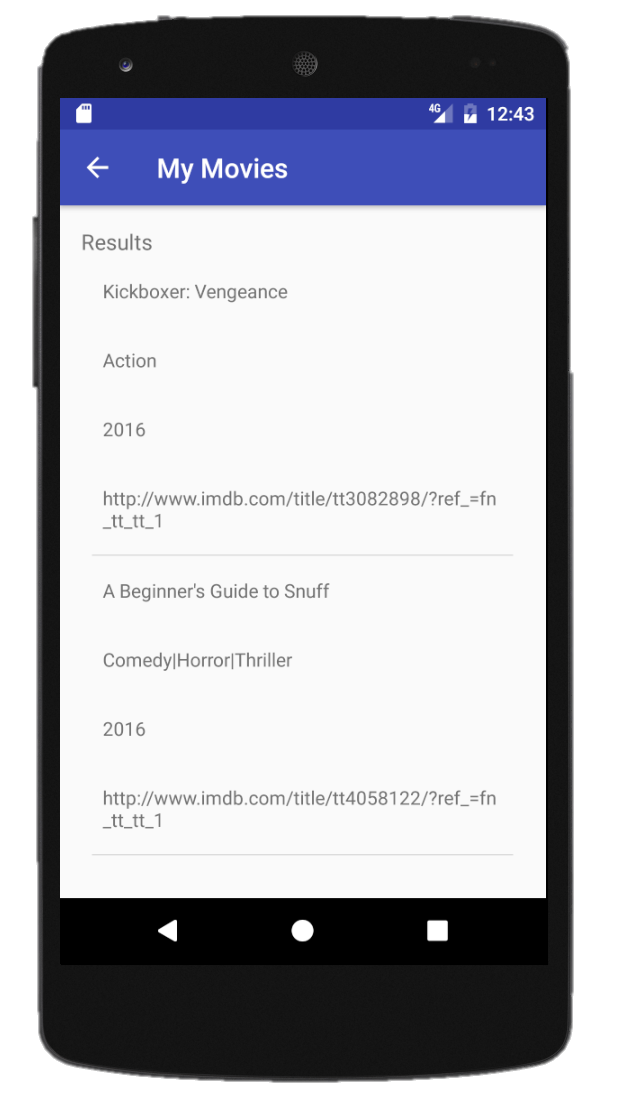
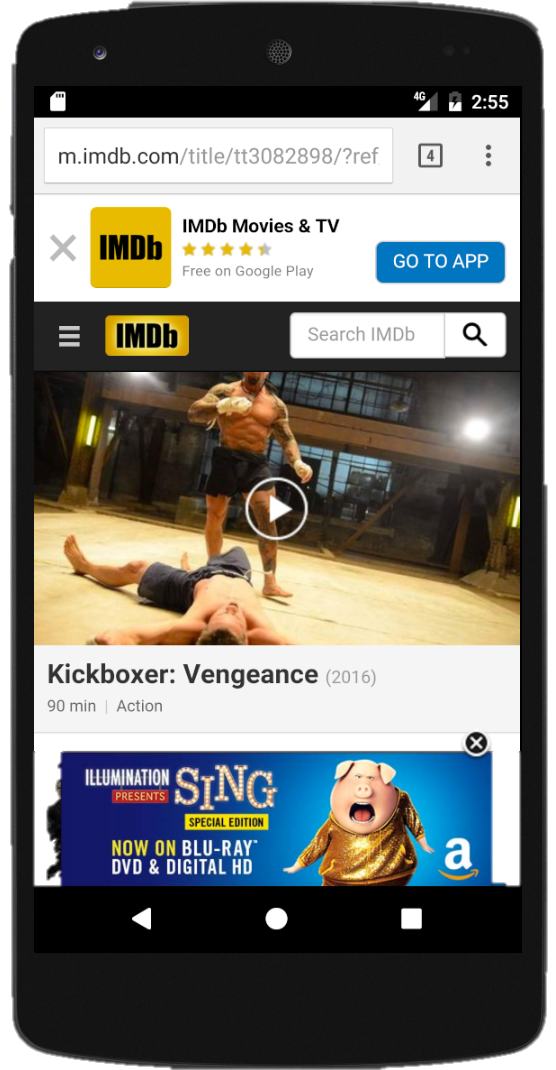


* Result Page:

The processing result shows on result page, which contains movie title, genres, start year and IMDB link for each record.

The user is able to scroll down to browse more movie records

The user is able to click the IMDB link to open IMDb web page to browse movie details.



# Conclusion

In our project, we implemented a server and two versions of clients for movie searching.

Our project satisfies all the functional requirements and whole the procedure runs as expected. Specifically, both shell client and Android client are able to provide user-friendly interfaces and interact with users effectively. It means that users are able to experience our application on PC or mobile devices. The server communicates with two clients smoothly without any exceptions and it processes the requests from clients correctly. Due to the server only returns 10 movie records for each request, it ensures that clients can display the results as soon as possible rather than take too much time on processing results. In order to decrease bugs, we made necessary tests as well. The platform that we develop the server and the shell client is the terminal, we are familiar with it and it works as expected. However, we knew nothing about Android so it spent a few days for us getting used to Android Studio. Every group member did an excellent job during the past few weeks. Even though we met some technical issues when developing the server and the two clients, we managed to solve the problems and completed individual tasks on time.

For future work, what we need to improve is to beautify the user interface on Android client because there are only texts on the interface of Android client. We may post some photos there. In addition, we can also add more functions to our application, for example, building up searching history records.

# Acknowledgment

We are pleased to acknowledge that this work partly supported by IMDb where is the data from.

And also, we gratefully acknowledge that there are very helpful discussions with Dr. Ziad Kobti and teaching assistants.

# Reference

[1] Dev.mysql.com. (2017). C API Function Overview. [online] Available at: <https://dev.mysql.com/doc/refman/5.7/en/c-api-function-overview.html> [Accessed 28 Mar. 2017].

[2] Google.ca. (2017). Google Images. [online] Available at: <https://www.google.ca/imghp?hl=en&ei=MzDPWLTWIIrbjwSZg42oCg&ved=0EKouCAIoAQ> [Accessed 28 Mar. 2017].

[3] Kaggle.com. (2017). Datasets | Kaggle. [online] Available at: <https://www.kaggle.com/datasets> [Accessed 28 Mar. 2017].

# Appendix

1. Archived source code including test code and dataset
2. Movie bank user guide
3. Redmine for project management: <https://redmine.cs.uwindsor.ca/projects/group_6>
4. GIT for version control: <https://github.com/jmpeng/movie_bank>
5. GIT for Android Client: <https://github.com/cindylyl/MyApplication>