FIT5192 Internet Applications and Web Services

Assignment 1: Web Services Practical Assignment

Weiwei SUN ID: 25467247

PART I: Design Document

1.1 Overview

This document will be helpful to clarify the detail of my work in Assignment One of FIT5192. We will design and implement a personalized search engine for movies on DVD, which will invoke and interact with several real-world web services, such as Google, Flickr, and Twitter, in this project. In this document, we will describe the key points of the implementation of this project (named MSearch). MSearch can provide local, content, image, video, content, and search results aggregation. What's more, we also implement a local REST web service providing the movie information, and a basic sentiment analysis.

This document will demonstrate the design detail of MSearch, a mashup web application consuming server web services providing by the third parties. The application design follows the design principal of Model-View-Controller (MVC). The local service is a RESTful web service which is based on Jersey, and to support the local web service, the open source product MySQL is used.

Git is selected to facilitate the source control, and Github is chosen as a remote repository. Therefore, you can download all source codes by the following commands:

```
$git clone https://github.com/wwsun/monash-msearch.git
```

1.2 System Architecture

MSearch is a personalized search engine which integrate with serveral real world web services. Clients send their requests to the MSearch server, and then the requests will be handled. Based on different request types, requests will make MSearch Server consumes different web services. For example, Alice want to get some relevant images about her favorite movie *Inception*, he could use the MSearch Image Search service, after type the keywords, the query will be sent to the MSearch server, and then sent to Flickr for

requesting the relevant images, that is to say, the actual work is done on the servers in Flickr.

The system architecture of MSearch is shown in Figure 1. In the center part is MSearch server, which handles the requests form user clients. Five child modules are comprised in MSearch, including Local search for popular movie detail, Image Search integrated with Flickr API, Video Search integrated with YouTube API, Content Search integrated with Google API, and also a simple Sentiment Analysis integrated with Viraheat API. Each function have a counterpart web service provided by different corporations. It is important to note that the Movie Detail is developed by ourselves, consuming the RESTful web service provided by MovieDB Server, and the data is stored in a MySQL database.

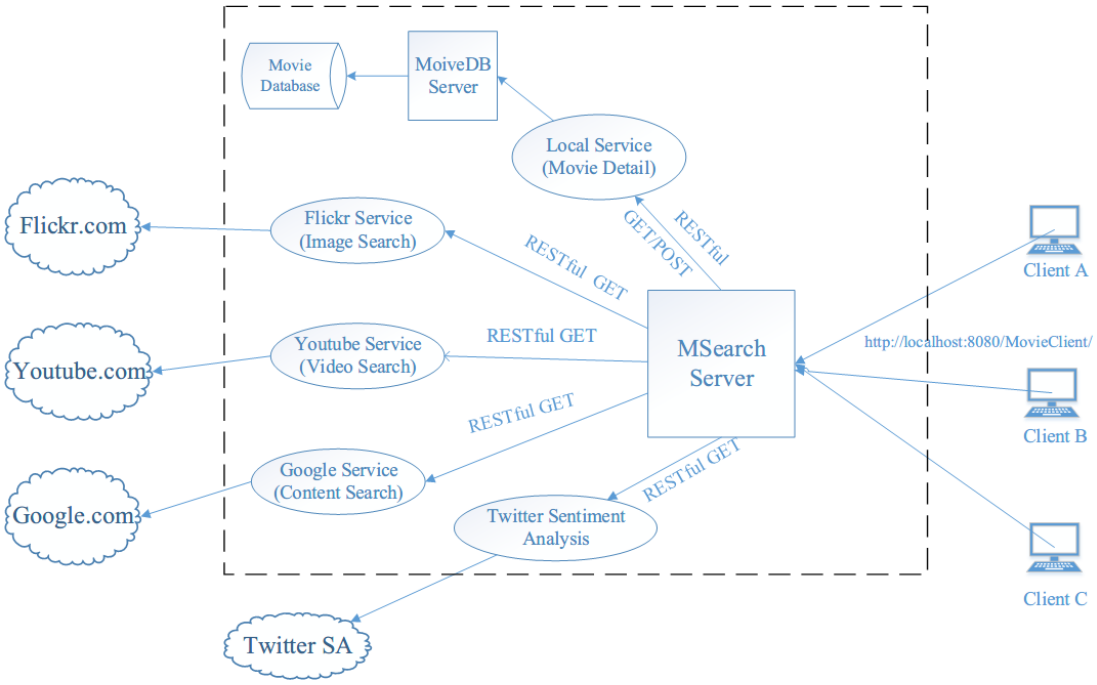

Figure 1: System architecture of MSearch

1.3 Database schema of movie database

In this project, we will implement a local web service based on the local database, which affording the basic movie information retrieving. Therefore, the database should be designed followed a certain database schema. The database schema is shown in Table-1, the key is Imdbnum.

Table 1: Database schema of the database

Name	Type	Allow empty	Memo
Imdbnum	Varchar	No	Key
Title	Varcahr	No	
Type	Varchar	No	
Rating	Double	Yes	
Description	Varchar	Yes	
Director	Varchar	Yes	
Starts	Varchar	Yes	
Coverpath	Varcahr	Yes	

PART II: Design Detail and Deployment Guide

To begin with, the High Distinction level is expected by my job with my hard work for three reasons. Firstly, all tasks are finished in this project, including a local web service, three real world service, adding moving and retrieving movie trailers, a sentiment analysis function. Secondly, the project programming followed the principal of software engineering and the Java code conventions. Thirdly, the project is built in a quite clear structure followed the principal of MVC and REST.

2.1 System deployment

2.1.1 Deployment environment

Operating system: Windows, Linux, or Mac OS (installed with JDK1.6 above)

Server: Glassfish 4

Database: MySQL community server 1.7 and above

2.1.2 Import the database

Database username: fit5192a1

In order to using the system, user should use the following commands to import the sql file into MySQL firstly.

```
mysql -u fit5192a1 -p < id25467247.sql
```

2.1.3 Deploy the War file

In order to access MSearch via a web client, you should deploy the ***moiveDB.war and movieClient.war*** into Glassfish via ***http://localhost:4848/*** first.

2.1.4 Setup the Glassfish connection pool

In order to making the service available, the connection pool should be configured. Adding the following codes into the file named *domain.xml*, which is located at *{Glassfish installment path}/glassfish-4.0/glassfish/domains/domain1/config/domain.xml*.

```
<!--mapping pool-name with jndi-name -->

<jdbc-resource pool-name="movieDBPool" jndi-name="movie_datasource"></jdbc-resource>

<!--setting connection-pool detail -->

<jdbc-connection-pool datasource-
classname="com.mysql.jdbc.jdbc2.optional.MysqlDataSource" name="movieDBPool" wrap-jdbc-
objects="false" connection-validation-method="auto-commit" res-
type="javax.sql.DataSource">

        <property name="URL" value="jdbc:mysql://localhost:3306/id25467247"></property>
        <property name="driverClass" value="com.mysql.jdbc.Driver"></property>
        <property name="portNumber" value="3306"></property>
        <property name="databaseName" value="id25467247"></property>
        <property name="User" value="fit5192a1"></property>
        <property name="serverName" value="localhost"></property>
        <property name="Password" value=""></property>

</jdbc-connection-pool>

<!--setting resource-reference -->

<resource-ref ref="movie_datasource"></resource-ref>
```

2.2 User guide and System operations

2.2.1 User interface and service introduction

After you set up the running environment, you can input the following URLs in the location bar in your web browser. The output in your browser is a typical style of search engine, which is the homepage of MSearch. The homepage is shown in Figure 2.

To run the local web service: http://localhost:8080/MovieDB/

To run the web client: http://localhost:8080/MovieClient/

Tip: You should make sure the Internet connection is available before you open this link in your browser, because of that several online resources are invoked in this project.

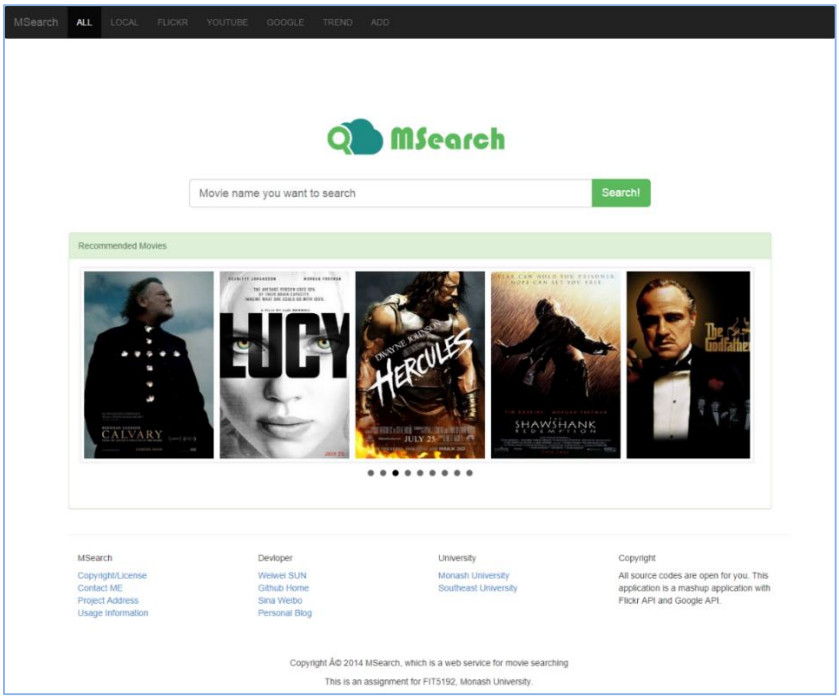

Figure 2: Homepage of MSearch

In MSearch, the default search setting searches aggregated results from all integrated web services. For a better service, you can choose into a specific service page in the navigation bar, such as Local, Flickr, YouTube, Google, and adding new movies that cannot retrieved in our local service information to the databases.

2.2.2 Local search, sentiment analysis, and add new movies

Local search is a service for retrieving movie information in the web services implemented by ourselves. In this part, we provide a simple and pure information card to display the results. In order to give more useful information, we also integrate the sentiment analysis in this part, which can give you the general opinions about the movie are negative or positive.

If you want to find some more information about this movie (e.g. movie 2012), you can click the buttons under the movie cover picture to find the counterpart contents, such as images, videos, and contents. Each button will lead you to the counterpart services provided by MSearch. The Local Search is shown in Figure 3.

The sentiment analysis is implemented by the help of a web application named ***Tweet Sentiment Analysis*** which opens a REST API to get the analysis result. Therefore, we acquire the analysis data between -1 and 1. For the purpose of visualizing the data, we use

the formula $\frac{x+Max}{Max-Min}$ to normalize the data into the interval 0 ~ 1. At last, a JavaScript library ***HighCharts*** is used to visualize the data into a half-pie chart.

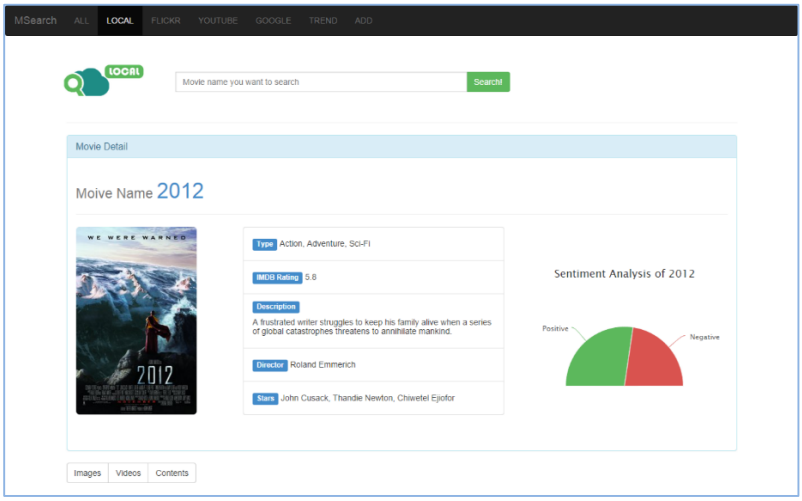

Figure 3: Local search with sentiment analysis

In the other hand, if the movie name you retrieved is not found in our local web service, the server will return you a prompt encouraging you to complete our database. The prompt information is shown in Figure 4. Figure 5 shows that the add movie popup after you click the add movie button. If your form successfully handled, it will redirect to the movie page you submit.

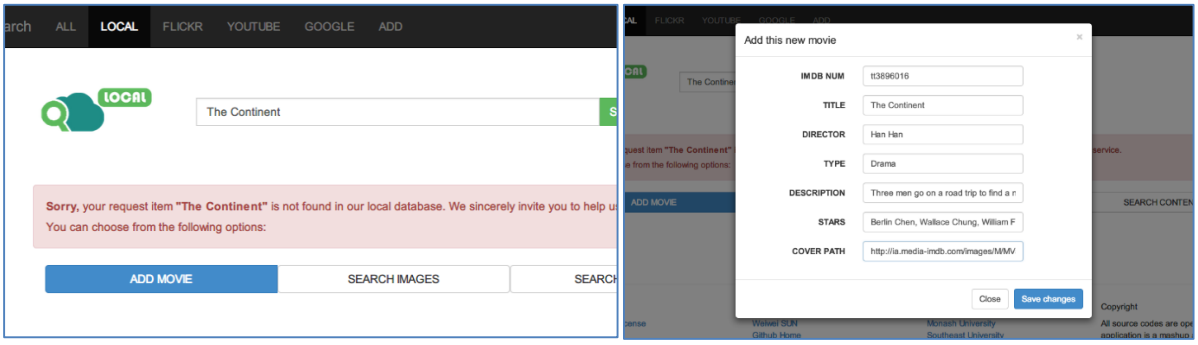

Figure 4, 5: the search item is not existed, add this new movie

2.2.2 Image search

Image search is a service for retrieving movie relevant pictures (e.g. movie stills). In image search service, the Flickr API is implemented with the purpose of providing more

useful and valuable pictures for our users. As default, the search engine will return you 22 pictures. Image search is shown in Figure 6.

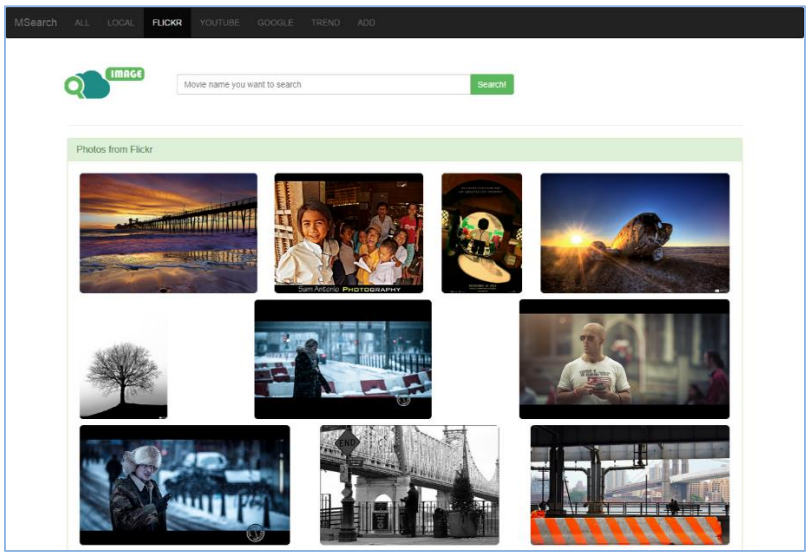

Figure 6: Image Search

2.2.2 Video search

Video search is a service for retrieving trailers of your interested movies. In video search service, the YouTube API is implemented with the purpose of providing more useful and valuable videos for our users. As default, the search engine will return you the most matched video which can played in the current page, and top three matched video snippets linking to the YouTube. Video search is shown in Figure 7.

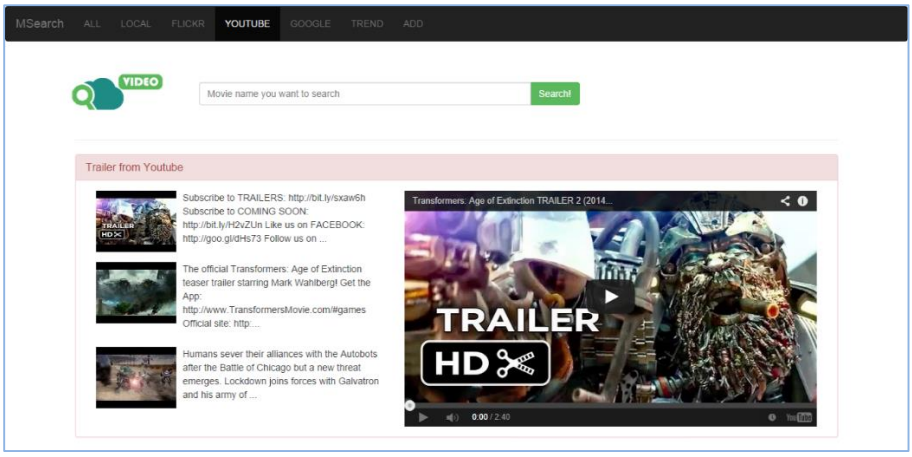

Figure 7: Video Search

2.2.2 Content Search

Content search is a service for retrieving information related with your interested movies. In content search service, the Google Custom Search REST API is implemented with the purpose of providing more useful and valuable contents for our users. As default, the search engine will return you ten most matched items linking to the specific pages. The content search result shows in Figure 8.

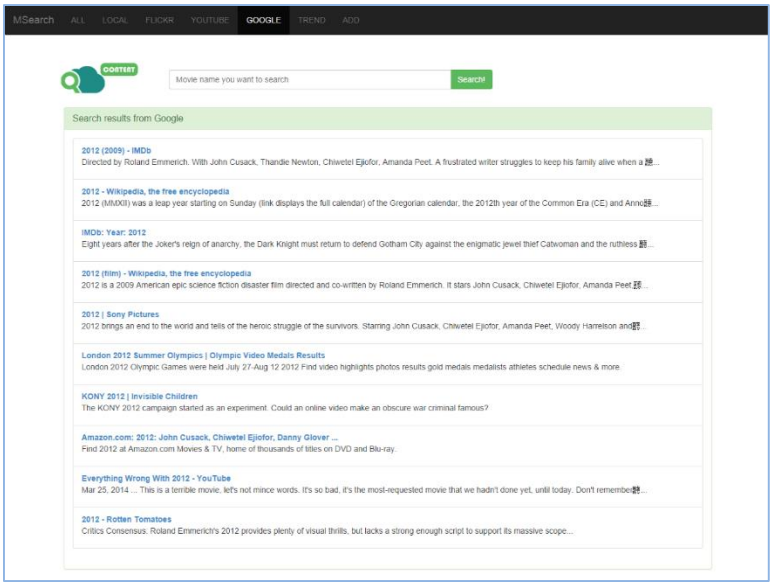

Figure 8: Content search

PART III: Referenced Third Party Projects

3.1 Bootstrap

3.1.1 Bootstrap Introduction

Bootstrap is a sleek, intuitive, and powerful front-end framework for faster and easier web development, created by Mark Otto and Jacob Thornton, and maintained by the core team with the massive support and involvement of the community.

3.1.2 Implementation in this project

In this project, all the user interface is based on the Bootstrap front-end framework, that is to say, the visual appearance of every components you can see is implemented based on Bootstrap. Some shortcuts of Bootstrap implementation are shown in Figure 9-11.

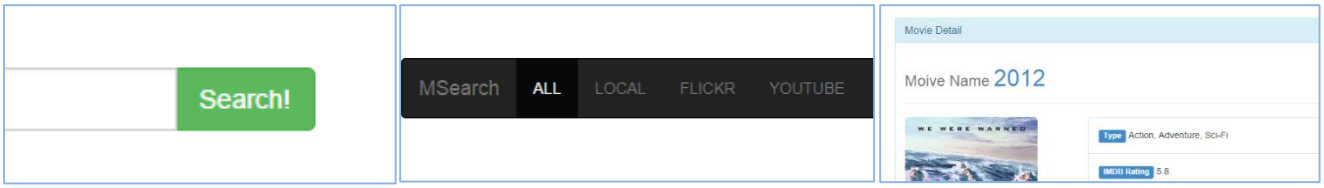

Figure 9, 10 and 11: Bootstrap implementation

3.2 YUI AutoComplete

3.1.1 YUI AutoComplete Introduction

The YUI AutoComplete widget provides a flexible, configurable, and accessible implementation of the AutoComplete design pattern, which offers suggestions or provides some other form of filtering or completion as a user types text in an input field.

3.1.2 Implementation in this project

In this project, all input boxes are integrated with YUI AutoComplete widget, that is to say, if you input some keyword an input box, the box will list some available choices for you. What's more, we provide autocomplete suggestions using the YQL query as the remote data source. The result of YUI AutoComplete implementation are shown in Figure 12.

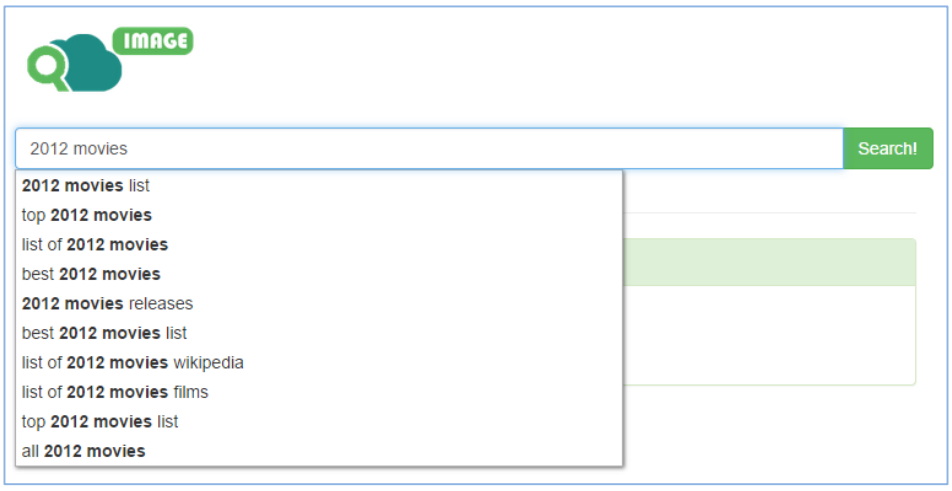

Figure 12: YUI AutoComplete Implementation

3.3 Highcharts

3.3.1 Highcharts Introduction

Highcharts is solely based on native browser technologies and doesn't require client side plugins like Flash or Java. The Highcharts Standalone framework is designed for those who do not already use jQuery, MooTools or Prototype in their web page, and wish to use Highcharts with minimal overhead.

3.3.2 Implementation in this project

In this project, Highcharts is used to implement data visualization. Especially in the sentiment analysis, we use a pie chart to display the analysis results. The implementation of Highchart is shown in Figure 13.

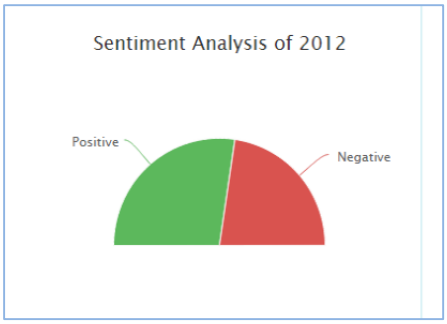

Figure 13: Highcharts implementation