

Keyword Search in Social Networks

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment of the

Requirements for the

Degree Master of Computer Science

by

Vijeth Patil

Spring 2012

© 2012

Vijeth Patil

ALL RIGHTS RESERVED

SAN JOSÉ STATE UNIVERSITY

The Undersigned Writing Project Committee Approves the Writing Project Titled

Keyword Search in Social Networks

by

Vijeth Patil

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Chris Pollett, Department of Computer Science

05/24/2012

Dr. Mark Stamp, Department of Computer Science

05/24/2012

Dr. Soon Tee Teoh, Department of Computer Science

05/24/2012

ACKNOWLEDGEMENTS

I would like to thank Dr. Chris Pollett for his advising, encouragement, guidance and support throughout this project. Thanks to my committee members Dr. Mark Stamp and Prof. Soon Tee Teoh for their suggestions, valuable time and support. I also would like to express gratitude to my family and friends for their support and especially those who helped in usability testing.

ABSTRACT

KEYWORD SEARCH IN SOCIAL NETWORKS

People often tend to ask their friends whenever they want some information related to topics like events, restaurants, or movies as majority of the search engines do not yield the desired results which people are seeking [1]. At present, majority of the current Open Source search engines like those based on Nutch also do not yield desired or expected results. Popular search engine, Google recently incorporated the feature of providing information from your social circle but only limited to Google Plus in your search results. On the other hand, micro blogging site Twitter has emerged as a vital source of information with more than 140 million active users [2] and nearly 250 million new tweets every day [2]. People also like to see more results from the blogs or news websites they follow and generally subscribe to their Really Simple Syndicate(RSS) [3] feed service to get the data and have to use RSS reader to find them. A web search engine which can provide results from user's social network content along with the indexed web results would be a great deal of help for people interested in results from their social circle.

This project's goal is to include results from your Social Networks (Twitter, RSS feeds) in Yioop! search results by using feeds database created from your Twitter account and RSS feeds you follow.

Table of Contents

Table of Contents.....	6
1 Introduction.....	9
2 Preliminary Work.....	10
3 Design and Technical Challenges	12
3.1 Introduction to Yioop!.....	12
3.2 Yioop! Architecture.....	13
3.2.1 System Architecture and Directory Structure	14
3.2.2 How the search component works in Yioop!.....	16
3.2.3 Crawling in Yioop!	16
3.2.4 Data base Architecture	17
3.3 Social Feed Search Engine	19
3.3.1 Modified Yioop! Architecture:.....	20
3.3.2 Modified Yioop! directory structure	22
3.3.3 Modified Search Architecture	25
3.3.4 Ranking Function:.....	25
3.3.5 Data base Architecture (Changes and Relationship between tables)	26
4 Implementation	29
4.1 Yioop! User Interface and Modifications.....	36
5 Testing and Results	41
5.1 Unit Tests	41
5.2 Usability Test	43
6 Conclusion	48
7 References.....	49

List of figures

Figure 1: Twitter Authorization Dialog.....	11
Figure 2: Twitter application posting the tweets on user's behalf	12
Figure 3: Directory structure of Yioop!	14
Figure 4: Yioop! Search Architecture	16
Figure 5: Database Structure of Yioop!.....	18
Figure 6: Sequence diagram of Initial design for Search Query.....	21
Figure 7: Sequence diagram of final design of Search Query	22
Figure 8: Modified Yioop! directory structure	23
Figure 9: Modified Yioop! Search Structure with Social search feature	25
Figure 10: Modified Database Architecture.....	27
Figure 11: Feed Server start() function calling the Crawl Daemon's init function.....	29
Figure 12: Loop function of FeedServer	30
Figure 13: Twitter API request	31
Figure 14: JSON Data returned from the Twitter API	32
Figure 15: Extracting the data from RSS feed in feed crawler	33
Figure 16: AddRSSFeed() function to check for existing RSS feed	34
Figure 17: Reciprocal Rank Fusion score algorithm in the FeedModel class	34
Figure 18: Manage Feeds function in AdminController.....	35
Figure 19: Yioop! user interface for search landing page	36
Figure 20: Search results view	37
Figure 21: Modified search results view containing results from social network	38
Figure 22: Admin Panel user interface (manage account).....	39
Figure 23: Manage Feeds View	40
Figure 24: Results of the Unit Tests for FeedCrawler using unit test framework of Yioop!	41

Figure 25: Results of unit tests for FeedModel using unit test frame work of Yioop!.....	42
Figure 26: First iteration of user interface created for the search results	43
Figure 27: Second iteration of the user interface for Search results.....	44
Figure 28: Final user interface for the Social search results	45
Figure 29: Graph showing the relative usability of the three user interfaces in comparison	46
Figure 30: Relevancy of results comparision between Yioop! social search and Wajam.....	47

List of Tables

Table 1: Usability test results for three different user interfaces	46
---	----

1 Introduction

The idea of this project is to access data (feeds, posts) from social networks and create a search engine over the collected data. The users of the system need to provide their credentials for the given social networks and then my search project will access the posts of the user and their friends from these social networks [4]. On a social network, a user has access to the documents authored by him and to the documents authored by his friends. Social networking sites like Facebook allow you to search for a user but not the posts, while other sites like Twitter allow you to search based on keywords but the results are not based on people you are following. My search project provides the ability to search by keywords from the posts of a user's circle of social network. Social search provides richer sources for relevance information than traditional web pages. Some examples of this are recency, followers count, retweet, and verified account, friends count. My project uses these additional signals to enhance the search ranking function in ways that traditional search engines and social networks are not able to do. As an example of this, if a user searches for a product and gets to know that 20 of his/her friends also talk good about the same product, he/she will immediately feel more comfortable and will be more likely to use that product. Similar process occurs on Twitter, when a person sees his or her trusted followers already engaging with a business, it acts as an unspoken endorsement. Another example is that when a user searches for a keyword the user gets from web resource blogs posted by their friends or from someone of user's interest which are available as Really Simple Syndicate (RSS) feeds are more relevant as compared to the search results returned by traditional way of search which do not give preference to your interests.

2 Preliminary Work

Social search has been an interesting topic and is supported by the growth of the content shared in the social media. Much of the prior work has been on the document centric ranking [14] of the social data ignoring the importance of the social graph of the user. Network centric search work is mostly concentrated on importance to searching people within a social network [15] and not the content.

The initial phase of the project involved understanding of various Social networks and implementing the data access to harness the fast growing content of your online social circle. Various experiments for accessing data from social network websites were worked on and indexing of the collected data was done in a search engine. A Twitter application was developed to access user's data from the social networking site Twitter. This application is able to successfully access other's tweet and posts' tweet.

Twitter Application

In this experiment, I created a non-trivial Twitter application to randomly access the latest tweets from the twitter users and post arbitrary tweet among them into user's profile after every few seconds. The goal of this experiment was to access Twitter user's data.



Figure 1: Twitter Authorization Dialog

Building Twitter Application: Twitter gives access to its enormous corpus of data via Twitter Application Programming Interface (API) [5]. A Twitter application will interact with Twitter API to access the data. In-order to access a users' private data, a Twitter application must be authorized by the user. Twitter supports OAuth authentication [6] where users are not required to share their passwords with third party application in order to authorize them on behalf to access data.



Figure 2: Twitter application posting the tweets on user's behalf

Twitter applications can access the data using different APIs such as REST API, Streaming API and Search API. In this experiment we made use of REST API to access the data seen on user's home timeline.

Experiments to access Facebook [7] and Del.icio.us [8] data using their APIs were also implemented successfully to understand the Social data.

3 Design and Technical Challenges

3.1 Introduction to Yioop!

Yioop! [9] is open source search engine written by Dr. Chris Pollett in PHP. It is licensed under GPLv3 and can be configured to provide search results for a set of pre-configured domains or for the whole web. Yioop! search engine provides a much better control to users

for choosing the exact set of sites indexed. The version of Yioop! documented during this report is v0.86. Yioop!'s parent site is SeekQuarry.com and you can find more information at this website.

3.2 Yioop! Architecture

Yioop!'s design is based on a web-based Model-View-Controller framework. It uses PHP multi-curl library for simultaneous page downloads and can distribute the tasks to several machines. BM25F is the ranking function used in Yioop! to rank the pages of the result set for search queries. Yioop! system needs a web server running and has name server to coordinate for crawls.

Queue server is a process responsible to perform scheduling and indexing jobs. Fetcher is another process which is responsible for downloading pages from the web. Yioop! can be configured to have multiple queue servers with fetchers running simultaneously using the simplified distributed model. Yioop! also has the ability to be configured multiple installations as "mirrors" having exact copy of data and hence can increase the query response throughput.

Installation of Yioop! requires you to have a web-server with PHP version 5.3 or higher installed and Curl libraries to download web pages. Yioop! provides a traditional web search interface to input the query and for the results.

3.2.1 System Architecture and Directory Structure

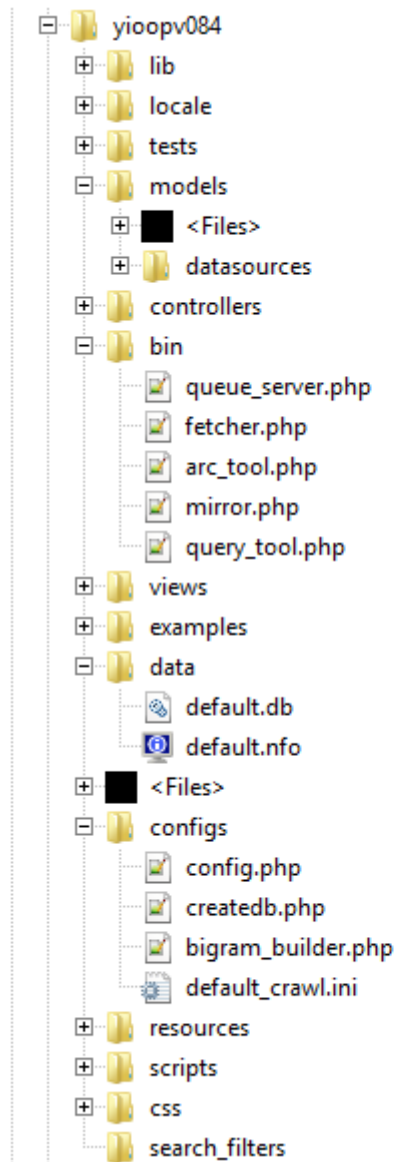


Figure 3: Directory structure of Yioop!

The directory structure of a Yioop! installation is as shown in Figure 3 above:

bin : bin contains fetcher and queue_server along with other command line scripts needed in conjunction with Yioop!

configs: It contains configuration files such as config.php and createdb.php script. *config* file holds a set of parameters that controls the various activities of Yioop!. *createdb* script is used to create a initial instance of database with just the root user.

controllers: This directory contains all the controller classes used in Yioop!. Controllers classes recieves most of the requests through index.php

lib: Lib contains all the common classes utilized for indexing, parsing, creating crawl daemons, stemming, archiving and iterating over various kind of web archive

locale: contains the data associated with various languages and region. Yioop! currently supports around 17 languages.

css : contains the stylesheets used to style the different web pages. Search.css is used for styling the search interface.

models: contains the various subclasses of Model class responsible for the accessing of database or the filesystem. There also exists the datasources directory containing the datasource_manager class providing the abstraction for various database systems and file systems. At this time, Yioop! currently supports three different database systems(sqlite, mysql and sqlite3)

views: this folder holds various View subclasses, elements, helpers and layouts.

tests: This folder contains the Unit tests for various components of Yioop!

work: work directory is created during configuration of Yioop!. It contains all the data including the crawl and index data. Crawl log and Scheduler log files are written in log directory.

3.2.2 How the search component works in Yioop!

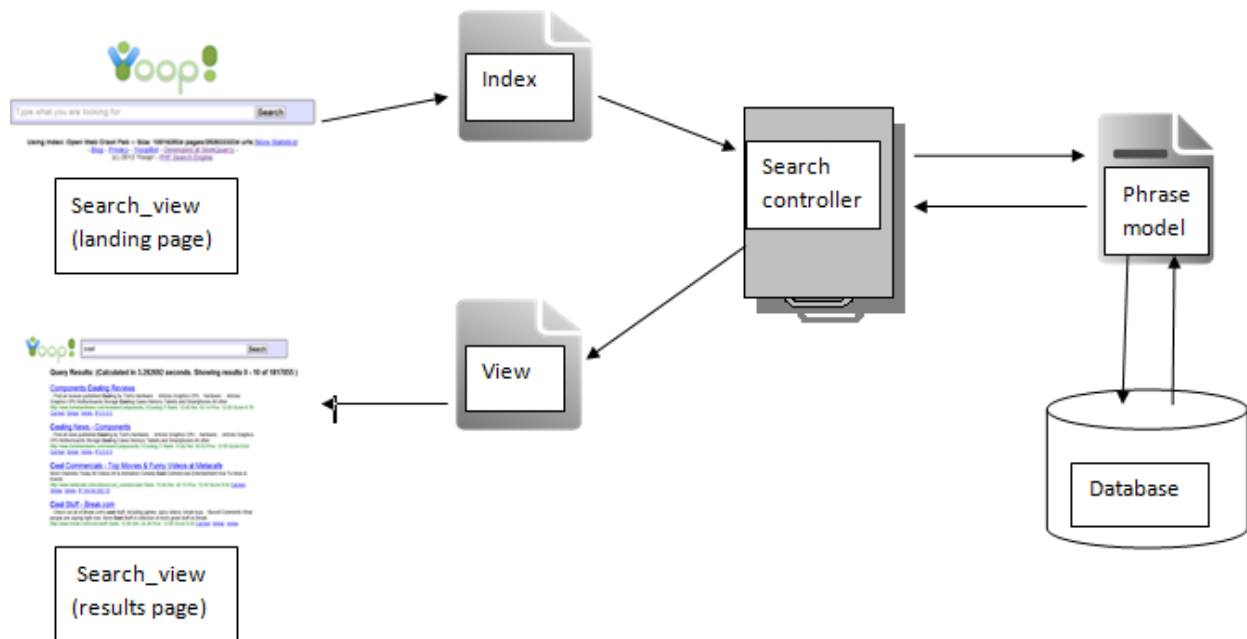


Figure 4: Yioop! Search Architecture

Yioop! search begins with the query input from the user. The search query passes to the index.php file where it is checked for the kind of requests whether they are part of and then the phrase query requests are passed on to Search controller. Search controller is responsible for handling requests and processing the search queries. This makes call to the phrase model , which in turn retrieves the data from the index database for the given search request. The results are scored and ranked accordingly. The results are returned back to the search controller which invokes the view to render the query results.

3.2.3 Crawling in Yioop!

Crawling in Yioop! requires having queue server and fetcher running. You can start these two processes using graphical user interface in the Yioop! admin panel. These can also be started using commands to run them as daemon process .

To start a queue_server type:

php queue_server.php terminal

To start a fetcher type:

php fetcher.php terminal

Crawling can be managed from Manage Crawl activity in the admin panel. It allows you to start, stop and resume the crawls when needed, select the crawl results as index for the search results. The Manage Crawl activity allows you to choose specific websites that you would like to crawl. The crawled data are stored under the work directory. Crawl contents are found in the folder `WORK DIRECTORY/cache/IndexDataUNIX_TIMESTAMP` one can see its contents in the folder `WORK DIRECTORY/cache/IndexDataUNIX_TIMESTAMP`

3.2.4 Data base Architecture

Yioop! maintains a separate database mainly to store user's information such as user account, user roles, activities they are allowed to perform in the admin panel. This database also has tables that store information about the version, crawl mixes, machine and locale.

Database Structure Browse Data Execute SQL		
Name	Object	
+	VERSION	table
+	USER	table
+	USER_ROLE	table
+	CURRENT_WEB_INDEX	table
+	CRAWL_MIXES	table
+	MIX_GROUPS	table
+	MIX_COMPONENTS	table
+	MACHINE	table
+	EMPLOYEE	table
+	sqlite_sequence	table
+	USER_SESSION	table
+	TRANSLATION	table
+	LOCALE	table
+	TRANSLATION_LOCALE	table
+	ROLE	table
+	ROLE_ACTIVITY	table
+	ACTIVITY	table
...	sqlite_autoindex_USER_1	index
...	sqlite_autoindex_TRANSLATION_1	index
...	sqlite_autoindex_CRAWL_MIXES_1	index
...	sqlite_autoindex_CRAWL_MIXES_2	index
...	sqlite_autoindex_MACHINE_1	index
...	sqlite_autoindex_MACHINE_2	index

Figure 5: Database Structure of Yioop!

VERSION: This table is used to store the version of the database. It will modify during upgrade of Yioop! using the database upgrade script.

USER: User table contains the user's account information such as a unique user id, user name and password.

USER_ROLE: Each user's has a role associated with him. This table maintains the mapping between user ID and role ID which are from user table and role table respectively.

USER_SESSION: Session information is stored in this table for each user.

CRAWL_MIXES: This table maintains the information about the crawl mixes such as crawl mix timestamp and crawl mix name.

CURRENT_WEB_INDEX: It contains the time stamp of the current web index used for search.

MIX_COMPONENTS and **MIX_GROUPS** are additional tables used during crawl mixing responsible for maintaining grouping.

MACHINE: It contains the URL, information about the number of fetchers and queue servers of all the machines used for crawling.

TRANSLATION: This table maintains a unique id for all the identifier strings that need to be translated during localization.

LOCALE: Locale contains the list of all the languages supported by the Yioop!

TRANSLATION_LOCALE: Maintains the translations of all the identifier strings from translation table to all the languages available in the locale table. Yioop! provides a graphical user interface to translate the identifier strings to a new language(locale) easily.

ROLE: Role table contains a Role ID and Role Name for each of the roles created.

ACTIVITY: Activity tables contains the list of activities available in the admin panel

ROLE_ACTIVITY: Role activity defines the mapping between the roles and the permitted activities for each role.

3.3 Social Feed Search Engine

The objective of this project is to enhance the user's search experience by providing them results from their social network along with the web results. Users should be able to access their

social network content using this search in more useful way and should also be able to access the results from RSS feeds they wish to follow. Users can manage their social network settings through the Manage Feeds activity in the Admin page. The feed crawler makes use of this information to fetch the user's social content into the database and the feed model is responsible for the database operations and the ranking of the search results. The social search engine makes use of model-view-controller framework provided by Yioop!.

3.3.1 Modified Yioop! Architecture:

3.3.1.1 Initial thoughts on Query Search Design

The addition of social search to Yioop!'s existing search architecture was a challenging task. The main goal of the design here was to not break the existing web search architecture and hence the initial thought of the design was to have the social search feature to be completely independent of the web search with social search having its own controller independent of the existing SearchController class which handles all the incoming queries. This design had a separate FeedSearchController class which handled the feed query requests and interacted with the feed model and then passed the results back to the index to combine the results and call the SearchView class to display the results.

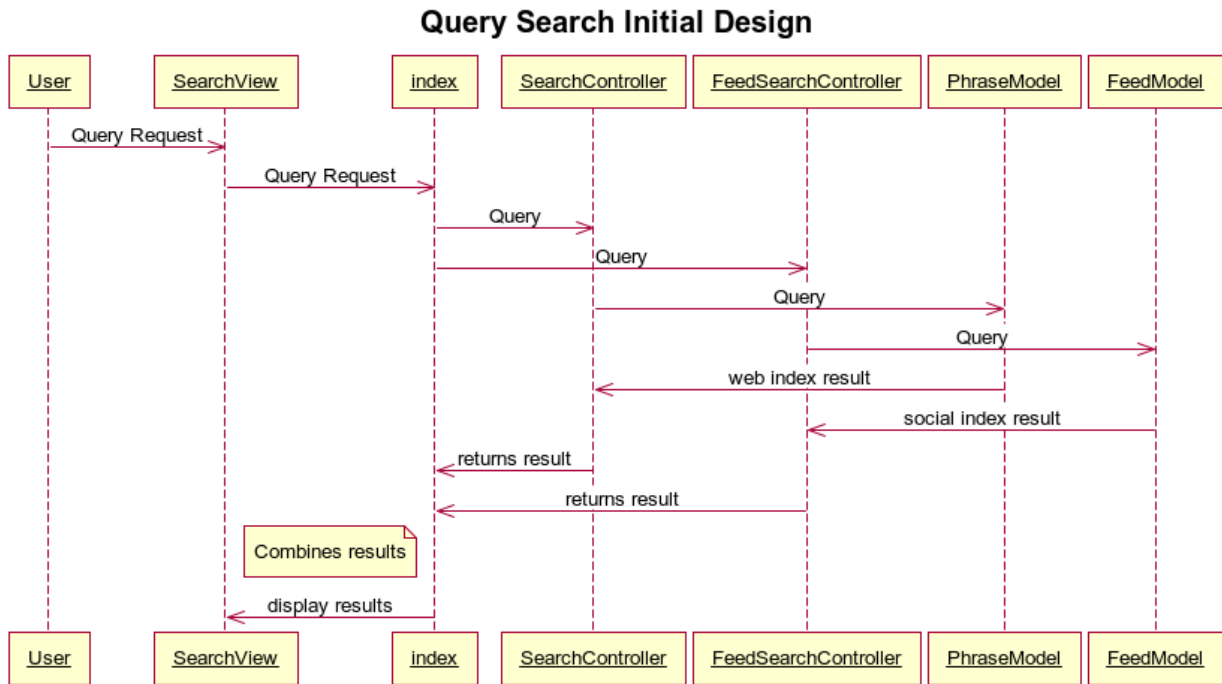


Figure 6: Sequence diagram of Initial design for Search Query

The above design was not suitable since the view is called through the index to return the results as the existing architecture needed to be changed as the SearchView was called by SearchView directly, whereas in this design each of the controllers will return the results to index to combined and then index calls the SearchView return to display the results.

3.3.1.2 Query Search Design

After thinking several possibilities over the design, the final search design had only one controller to handle both the web index search and social feed search as this design is much aligned with the existing Yioop! structure and the SearchController handles the call to the PhraseModel and the FeedModel to get the results from both the models and combines the result to display the results by calling the View.

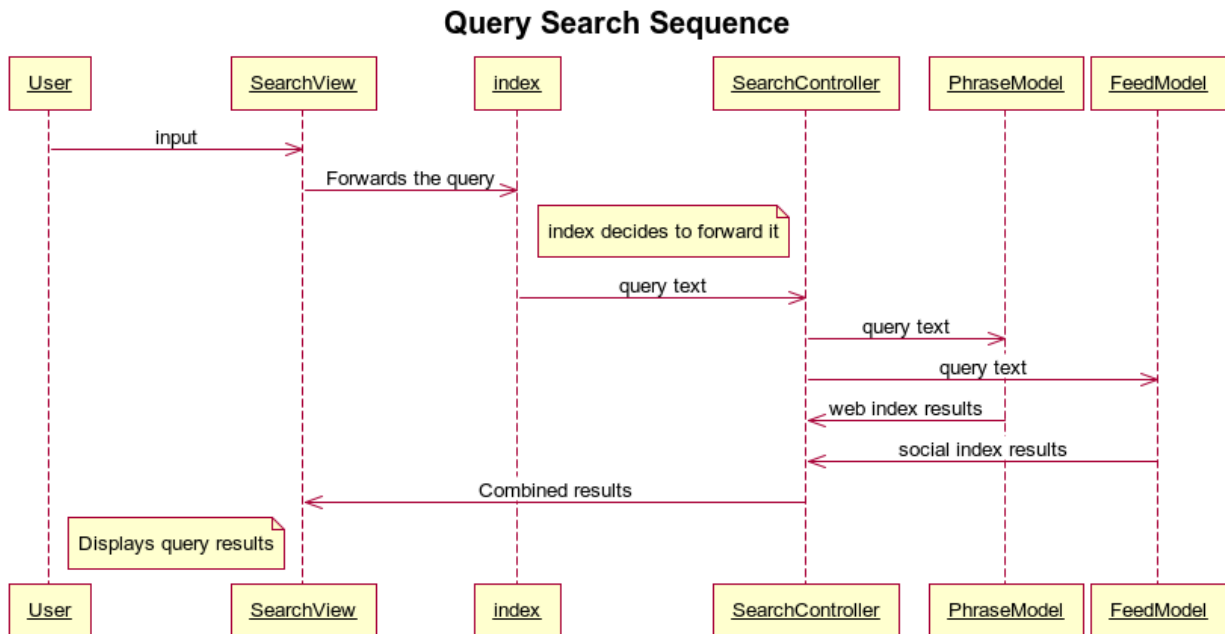


Figure 7: Sequence diagram of final design of Search Query

This design provided much better integration with the current Yioop! architecture. This design requires a change in the format of the results array passed on the View to accommodate both web and social results.

3.3.2 Modified Yioop! directory structure

The following directories and files are added to the Yioop! to support the Feed search feature:

feeds: This directory contains various classes and files responsible for the crawling of feeds from Twitter and RSS. FeedCrawler and RSSFeedCrawler are responsible for fetching the data from Twitter and RSS links. Twitter API library files are used for helping to create Twitter API requests.

Feed Crawler: Feed Crawler is responsible for fetching, converting and storing the feeds from Twitter to the database [10]. It gets the user's Twitter account tokens and uses the tmhOAuth Twitter API library [11] to make requests to Twitter. Twitter returns the user's home timeline data, which essentially contains the tweets posted from the people followed by the user on the network and also his own tweets.

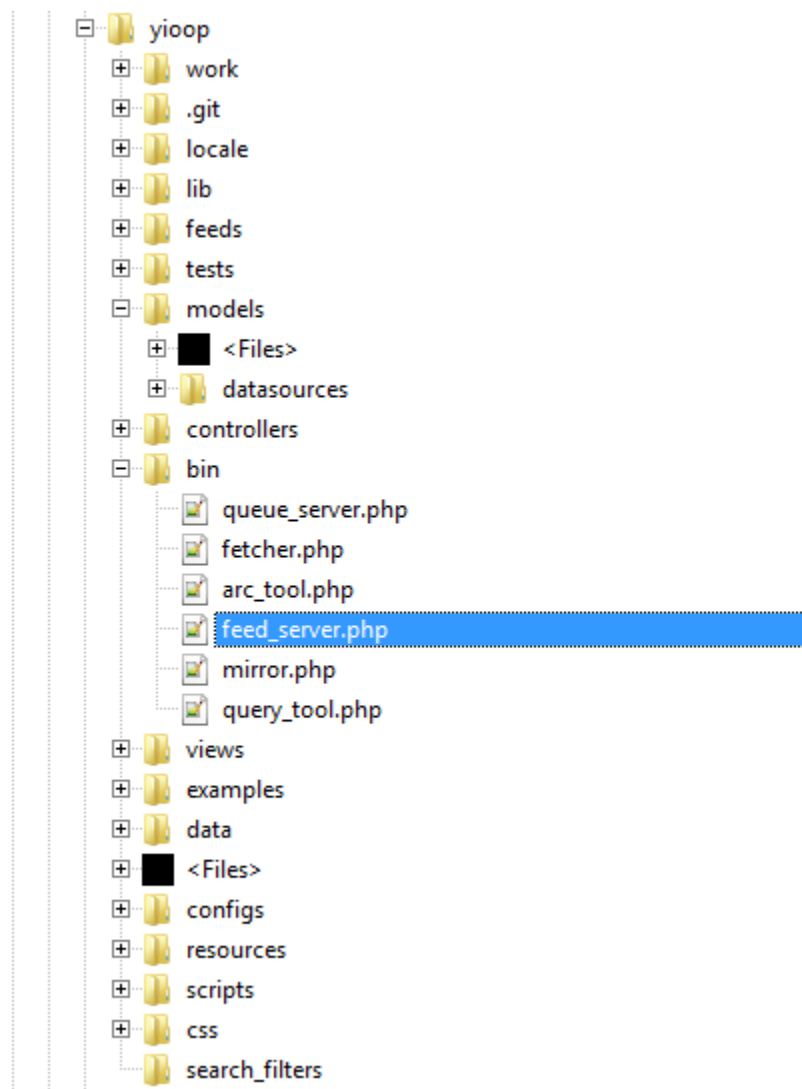


Figure 8: Modified Yioop! directory structure

RSS Feed Crawler: RSS feed crawler takes the rss feed links pertaining to the user from the database table and retrieves the RSS feeds from the web, parses it and inserts it to the database.

Feed Server: Feed server resides under the bin directory running as a daemon process. It is responsible for maintaining the crawling activity. It calls the Twitter feed crawler and rss feed crawler at regular intervals to fetch the newly posted data from Twitter and the subscribed rss feeds. Feed server also writes logs to a log file named after it in the log directory. The crawling activity includes the fetching of users tokens and secrets from the database and also the RSS feed links subscribed the user and creating a request for Twitter API and parsing data returned in the JavaScript Object Notation (JSON) data format returned by it. It is much simpler to fetch a RSS feed data as it only involves obtaining the standardized Extensible Markup Language (XML) format of web feed data and parsing the necessary tags from the data.

In order to start or stop the feed server, the script should be run from the command prompt using the following commands.

To start a feed server type:

php feed_server.php start

To stop the feed server type:

php queue_server.php stop

3.3.3 Modified Search Architecture

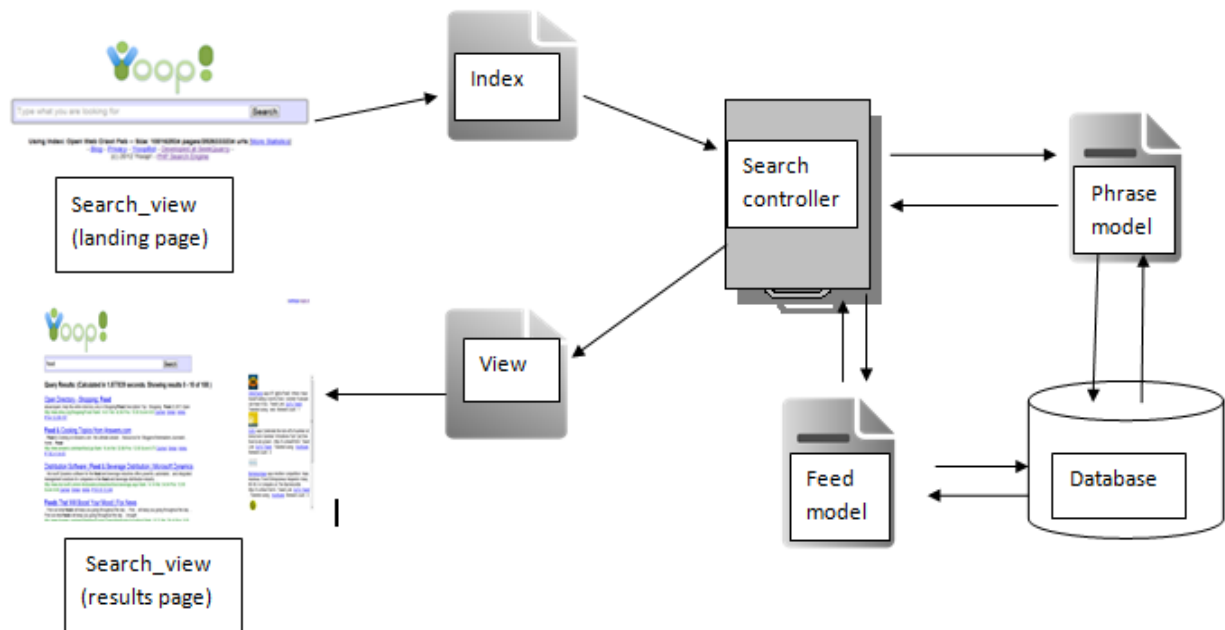


Figure 9: Modified Yioop! Search Structure with Social search feature

The user query from the search view is passed to the SearchController through index and then the SearchController determines the type of query and now calls the FeedModel along with the PhraseModel to retrieve the search results. Feed Model is responsible for the database operations for the various database operations and for the ranking of the feed search results.

3.3.4 Ranking Function:

The ranking function in the keyword search uses various social factors for ranking the results. The ranking function uses the information such as recency, followers count, verified account flag from the feed table to rank the feeds and the result is returned back to the search controller. The search controller then calls the view to display the results of the keyword search.

Reciprocal Rank Fusion:

In this method of ranking [12], each feed is assigned a rank for each of the factors separately and then their collective score determines the Reciprocal Rank Fusion score for that feed.

$$RRFscore(d \in D) = \sum_{r \in R} \frac{1}{k + r(d)}, \quad (\text{eq. 1})$$

where d is document belonging to the set of documents D .

r is the ranking within the set R .

k is a constant determined to be **60** at the pilot phase and never altered. The constant k mitigates the impact of highly ranked documents from only few factors.

3.3.5 Data base Architecture (Changes and Relationship between tables)

Yioop! creates and maintains a separate database to store each user's information, such as account, roles, activities user is allowed to perform.

Database Structure Browse Data Execute SQL		
Name	Object	
+ CRAWL_MIXES	table	
+ MIX_GROUPS	table	
+ MIX_COMPONENTS	table	
+ sqlite_sequence	table	
+ MACHINE	table	
+ USER_SESSION	table	
- USER_FEED	table	
USER_ID	field	
FEED_ID	field	
FEED_TYPE	field	
- USER_KEYS	table	
USER_ID	field	
FEED_TYPE	field	
CONSUMER_KEY	field	
CONSUMER_SECRET	field	
USER_KEY	field	
USER_SECRET	field	
USER_SCREENNAME	field	
CURRENT_SINCEID	field	
- FEED	table	
FEED_ID	field	
FEEDER	field	
FEEDTEXT	field	
REFEEDCOUNT	field	
FEEDTIME	field	
FEEDSOURCE	field	
FEEDERPIC	field	
FOLLOWERS_COUNT	field	
FRIENDS_COUNT	field	
VERIFIED	field	
- RSSFEED	table	
FEEDURL_ID	field	
FEEDURL	field	
- USER_RSSFEED	table	
USER_ID	field	
FEEDURL_ID	field	
+ TRANSLATION	table	

Figure 10: Modified Database Architecture

The following tables are added to the Yioop!'s database structure:

USER_KEYS: Information such as user's *screen_name*, *user_key*, *user_secret* and *since_id* fields are stored in this table. This information is used by feed crawler to create a request to API in order to retrieve the data from Twitter.

FEED: This table stores all the feeds retrieved from Twitter and RSS feed links of users and contains information such as their creation time, source of each feed which are useful for ranking them.

RSSFEED: This feed maintains the list of all RSS feeds added by all the users. It assigns a unique Feed id for each feed URL.

USER_RSSFEED: This table contains the mapping information between the users and the RSS feeds that belong to them.

USER_FEED: It's a table that stores the relationship between users and the feeds related to them by using user id and feed id.

4 Implementation

FeedServer:

```
/**
 * This is the function that should be called to get the feedserver
 * to start. Calls init to handle the command line arguments then enters
 * the feed_server's main loop
 */
function start()
{
    global $argv;

    declare(ticks=200);
    if(isset($argv[1]) && $argv[1] == "start") {
        $argv[2] = "none";
        $argv[3] = self::INDEXER;
        CrawlDaemon::init($argv, "feed_server");
    } else {

        .
        .
        .
        $this->loop();
    }
}
```

Figure 11: Feed Server start() function calling the Crawl Daemon's init function

Feed server looks similar to queue server and manages both RSS and Twitter feed crawlers. Feed server should be started with the command line using `php feed_server.php start` command and this will call the `start()` function that creates a daemon process using the `CrawlDaemon::init()` function. The `start()` function also creates the logging and scheduling for the process and then calls the `loop()` function.

```

/**
 * Main runtime loop of the feed_server.
 * Loops until a stop message received, check for start, stop,
 */
function loop()
{
    crawlLog("In feed loop!! $server_name", "feed_server");
    .
    .
    .
    while ($info[self::STATUS] != self::STOP_STATE) {
        .
        .
        .
        $time_diff = time() - $start_loop_time;
        crawlLog("time diff is $time_diff , sleeping for ".(FEED_CRAWL_SLEEP_TIME-$time_diff)." seconds");
        if( $time_diff < FEED_CRAWL_SLEEP_TIME) {
            crawlLog("Sleeping...");
            sleep(FEED_CRAWL_SLEEP_TIME - $time_diff);
        }
    }
    crawlLog("$server_name shutting down!!");
}

```

Figure 12: Loop function of FeedServer

loop() is the main runtime doing the crawl activities until a stop message is received. The loop() function creates an instance of FeedCrawler and RSSFeedCrawler to begin the new data fetch from the social networks and rss links. The feed_server goes to sleep until the next time interval for fetch has arrived.

FeedCrawler:

FeedCrawler is instantiated from FeedServer. It establishes a connection to database and gets the user tokens and secrets from the USER_KEYS table. FeedCrawler uses Twitter API library TmhOAuth [5] to establish OAuth connection with the Twitter server. To establish an OAuth connection, a consumer key, consumer secret, user token and user secret are needed.

Twitter applications are called consumers. Each Twitter application has a unique consumer key and consumer secret. A unique user token and user secret is generated when the application is authorized by the user. After establishing an OAuth connection with Twitter's Feed Crawler accesses the REST API of Twitter querying for the Tweets on user's Home Timeline. Home Timeline consists of the Tweets from the people whom the user is following and the tweets posted by user himself. The data returned from Twitter is in the JSON format and is parsed to extract and store relevant information fields such as tweet id, tweet text, followers count, friends count, retweet, verified account and screen name into the database. The Twitter API imposes a rate limit on the number of requests sent to it per hour and we need to honor it in order to be not banned by them. We use the since_id while making a request to the Twitter API which will fetch you the tweets posted after this since_id. Since_id is stored in the database for each user.

```
$tmhOAuth = new tmhOAuth(array(
    'consumer_key'    => $consumer_key,
    'consumer_secret' => $consumer_secret,
    'user_token'      => $user_token,
    'user_secret'     => $user_secret,
));

$time_start = microtime(true);
$code = $tmhOAuth->request('GET', $tmhOAuth->url('1/statuses/home_timeline'), array(
    'include_entities' => '1',
    'include_rts'      => '1',
    'screen_name'      => '$user_screenname',
    'count'             => 200,
    'since_id'          => $since_id
));

if ($code == 200) {
    $timeline = json_decode($tmhOAuth->response['response'], true);
    $this->crawlTweets($timeline, $user_id);
}
```

Figure 13: Twitter API request

Twitter API request: http://api.Twitter.com/1/statuses/home_timeline.json

```

{
  "in_reply_to_user_id_str": null,
  "in_reply_to_status_id": null,
  "in_reply_to_user_id": null,
  "id_str": "201009390562713600",
  "contributors": null,
  "truncated": false,
  "in_reply_to_screen_name": null,
  "created_at": "Fri May 11 18:02:27 +0000 2012",
  "retweeted": false,
  "geo": null,
  "user": {
    "id": 145125358,
    "id_str": "145125358",
    "default_profile": false,
    "profile_use_background_image": true,
    "location": "Mumbai, India",
    "profile_text_color": "0C3E53",
    "statuses_count": 13074,
    "following": true,
    "utc_offset": 19800,
    "profile_sidebar_border_color": "F2E195",
    "listed_count": 21196,
    "name": "Amitabh Bachchan",
    "protected": false,
    "profile_background_tile": true,
    "is_translator": false,
    "profile_sidebar_fill_color": "FFF7CC",
    "contributors_enabled": false,
    "profile_image_url_https": "https://si0.twimg.com/profile_images/1758722272/N5pcyOv5x4yBZO3KI7",
    "geo_enabled": false,
    "friends_count": 299,
    "description": "Actor ... well at least some are STILL saying so !!",
    "profile_background_image_url_https": "https://si0.twimg.com/profile_background_images/14422135",
    "default_profile_image": false,
    "notifications": false,
    "profile_image_url": "http://a0.twimg.com/profile_images/1758722272/N5pcyOv5x4yBZO3KI7M9NA_",
    "show_all_inline_media": true,
    "favourites_count": 3,
    "followers_count": 2654079,
    "follow_request_sent": false,
    "profile_background_color": "BADFCD",
    "url": "http://bigb.bigadda.com",
    "screen_name": "SrBachchan",
    "verified": true,
    "lang": "en",
    "created_at": "Tue May 18 05:16:47 +0000 2010",
    "profile_background_image_url": "http://a0.twimg.com/profile_background_images/144221357/t-1.g",
    "profile_link_color": "FF0000",
    "time_zone": "Mumbai"
  },
  "retweet_count": 13,
  "source": "web",
  "in_reply_to_status_id_str": null,
  "place": null,
  "coordinates": null,
  "favorited": false,
  "id": 201009390562713600,
  "text": "T 740 - Another end of day .. still on song and dance with Bol Bachchan .. meeting media, inaugurating Ma
busy .."
},

```

Figure 14: JSON Data returned from the Twitter API

RSSFeedCrawler:

```
$feedpage = FetchUrl::getPage($url);
```

```
$doc->loadXML($feedpage);  
$arrFeeds = array();  
$title = $doc->getElementsByTagName('title')->item(0)->nodeValue;  
$description = $doc->getElementsByTagName('description')->item(0)->nodeValue;  
$link = $doc->getElementsByTagName('link')->item(0)->nodeValue;  
$pubdate = $doc->getElementsByTagName('pubDate')->item(0)->nodeValue;
```

Figure 15: Extracting the data from RSS feed in feed crawler

It has the same objective as the FeedCrawler but only fetches and stores the content from RSS feeds subscribed by the user. The RSSFeedCrawler fetches all the RSS links a user has added from the USER_RSSFEED and RSSFEED tables. The content from these links are downloaded using the Multi Curl feature provided by Yioop! framework. RSS feed content returned is standardized XML file format which allows the information contained to be easily readable by parsing through different tags using the DOM structure.

FeedModel:

FeedModel is responsible for the database operations related to managing feeds as well as retrieving and ranking of the query results. Feed model extends the Model class. All the database queries are handled through datasource_manager which abstracts the underlying database system.

```

if(isset($feedurl_id)){
    $checkdupstmt = "SELECT * FROM USER_RSSFEED WHERE USER_ID = '$user' AND FEEDURL_ID = '$feedurl_id' LIMIT 1";
    $result = $this->db->execute($checkdupstmt);
    $row = $this->db->fetchArray($result);
    $user_id = $row['USER_ID'];
    if(isset($user_id)){
        return;
    }
    else{
        $addusererrssfeedstmt = "INSERT INTO USER_RSSFEED VALUES ('$user','$feedurl_id')";
        $this->db->execute($addusererrssfeedstmt);
        return;
    }
}

```

Figure 16: AddRSSFeed() function to check for existing RSS feed

The Ranking Function for the search results is implemented using the various fields stored in the database tables.

Reciprocal Rank Fusion:

```

$page[self::SCORE] = $this->getReciprocalRankFusionScore($rankinfo,$j);

```

```

/**
 * To get the ranking of each of the feeds according to reciprocal rank fusion ranking method
 */
function getReciprocalRankFusionScore($rankinfo,$j)
{
    $k=60;
    $score['refeedcount'] = 1/($k + $rankinfo['REFEEDCOUNT'][$j]);
    $score['feedlength'] = 1/($k + $rankinfo['FEEDLENGTH'][$j]);
    $score['feedtime'] = 1/($k + $rankinfo['FEEDTIME'][$j]);
    $score['followerscount'] = 1/($k + $rankinfo['FOLLOWERSCOUNT'][$j]);
    $score['friendscount'] = 1/($k + $rankinfo['FRIENDSCOUNT'][$j]);
    $score['verified'] = 1/($k + $rankinfo['VERIFIED'][$j]);

    $rrfscore = $score['refeedcount'] + $score['feedlength'] + $score['feedtime'] + $score['followerscount'] + $score[
    'friendscount'] + $score['verified'] ;
    return $rrfscore;
}

```

Figure 17: Reciprocal Rank Fusion score algorithm in the FeedModel class

AdminController:

AdminController extends the controller and handles the admin activities such as Managing account, users, roles, and crawl and also now has a feeds. The available activities are stored in ACTIVITIES table. Most of the requests to the admin controller are passed along with a string representing the activity determines which controller to be used. Manage feed gives you the ability to add or delete Twitter account information or a RSS feed.

```
function manageFeeds()
{
    $possible_arguments =
        array("addfeed", "deletefeed", "addRSSfeed");
    $data["ELEMENT"] = "managefeedsElement";

    if(isset($_REQUEST['arg']) &&
        in_array($_REQUEST['arg'], $possible_arguments)) {

        switch($_REQUEST['arg'])
        {
            case "addfeed":
                if($this->feedModel->getFeedId($usertoken) > 0){
                    $data['SCRIPT'] .= "doMessage('<h1 class=\"red\">'.
                        t1('admin_controller_feedname_exists').
                        "</h1>')";
                    return $data;
                }
                $this->feedModel->addFeed($usertoken,$usersecret,$userscreenname);
                $data['SCRIPT'] .= "doMessage('<h1 class=\"red\" >'.
                    t1('admin_controller_rolename_added').
                    "</h1>')";
                break;
        }
    }
}
```

Figure 18: Manage Feeds function in AdminController

Database Structure: Changes are made to the **createdb** script file which creates the initial database for the Yioop!

configs : The file contains the most common configuration settings and few additions were made to accommodate new names.

4.1 Yioop! User Interface and Modifications

Yioop!'s main search page contains a search bar at the center with a placeholder to help user understand the input method. Above the search bar is the Yioop! logo. Below the search is the information related to the Index in use for the current search and links to the various web pages related to Yioop! search engine.



Figure 19: Yioop! user interface for search landing page

At the top right are the two links 'Settings' and 'Sign In'. One can set the number of results shown per page, language and the index used for the search results by going to the 'Settings' page. User can sign in with their username and password to enter into Account Management



Query Results: (Calculated in 3.282692 seconds. Showing results 0 - 10 of 1817855)

[Components Cooling Reviews](#)

.. Find all reviews published **Cooling** by Tom's Hardware. .. Articles Graphics CPU... Hardware. .. Articles Graphics CPU Motherboards Storage **Cooling** Cases Memory Tablets and Smartphones All other
<http://www.tomshardware.com/reviews/Components,1/Cooling,7/> Rank: 15.60 Rel: 53.14 Prox: 12.00 Score 9.79
[Cached](#) [Similar](#) [Inlinks](#) [IP:0.0.0.0](#)

[Cooling News - Components](#)

.. Find all news published **Cooling** by Tom's Hardware. .. Articles Graphics CPU... Hardware. .. Articles Graphics CPU Motherboards Storage **Cooling** Cases Memory Tablets and Smartphones All other
<http://www.tomshardware.com/news/Components,1/Cooling,7/> Rank: 15.60 Rel: 50.63 Prox: 12.00 Score 9.64
[Cached](#) [Similar](#) [Inlinks](#) [IP:0.0.0.0](#)

[Cool Commercials - Top Movies & Funny Videos at Metacafe](#)

More Channels Today All Videos Art & Animation Comedy **Cool** Commercials Entertainment How To News & Events
http://www.metacafe.com/videos/cool_commercials/ Rank: 15.60 Rel: 45.75 Prox: 12.00 Score 9.60 [Cached](#)
[Similar](#) [Inlinks](#) [IP:184.84.222.10](#)

[Cool Stuff - Break.com](#)

.. Check out all of Break.com's **cool** stuff, including games, spicy videos, break buzz... Recent Comments What people are saying right now. More **Cool** Stuff A collection of more great stuff on Break.
<http://www.break.com/cool-stuff/> Rank: 12.80 Rel: 44.46 Prox: 12.00 Score 9.20 [Cached](#) [Similar](#) [Inlinks](#)

Figure 20: Search results view

One can type their search query and press enter key or click on Search button to get the search results for their query. The search results are listed on by their ranking with a short description of each link. Links are colored blue and underlined to give the intuition to user that they are clickable. User can click on the links below to view the cached page, similar pages or inlinks in the pages. It also displays the calculated amount of time for the query results and displays the results using pagination.

Modification to the Search results view

Yioop! user interface has been modified to allow it to display the results from your social circle along with the results from the web index. When a query is typed, the results are generated from both the web index search engine and feed search engine for the same input. The results from

web index are displayed as usual under the search bar and the results from the feed engine are display in a side box with the each result in it having a small picture of the people who posted it, a link to their profile, text that contains the match for the query search, link to the tweet posted.

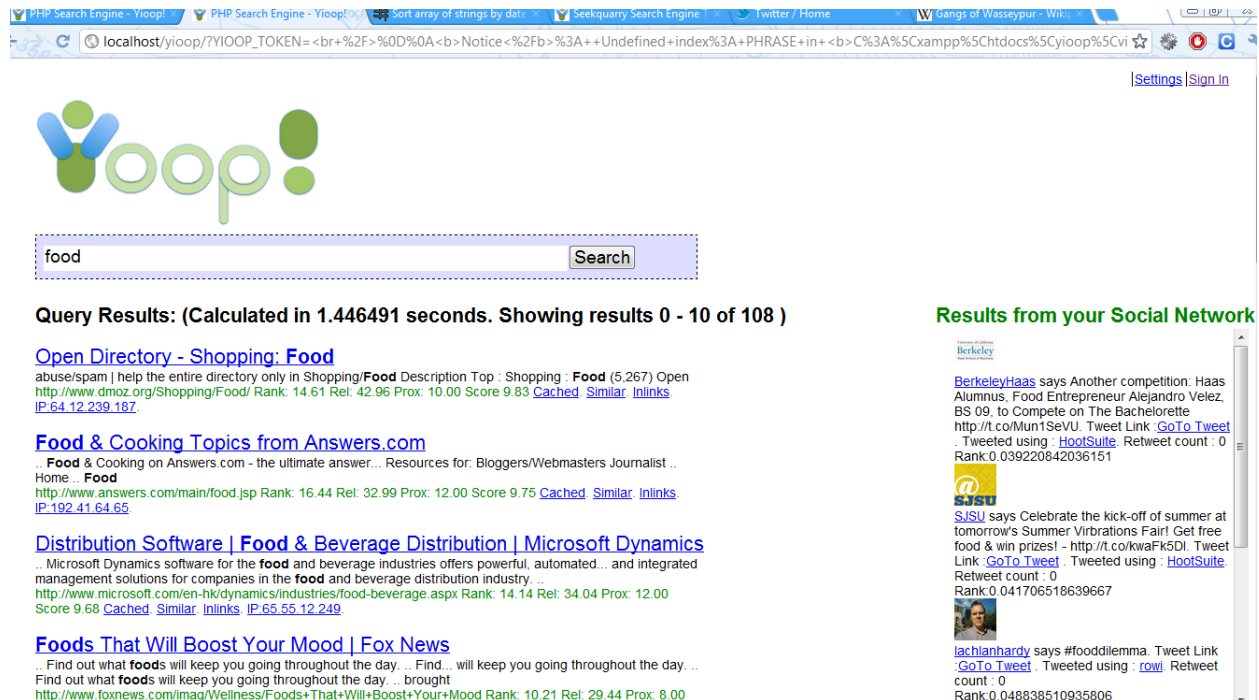


Figure 21: Modified search results view containing results from social network

It also shows other information such as the application used by the source to post the feed, ranking and the amount of times the post has been retweeted. Only the top ten results are shown from your social network limiting to the most relevant results. This user interface will provide a uncluttered experience for the users who are interested in either of the search results.

Admin Panel:

Yioop! provides you a personalized search experience allowing you to perform your own crawls, users and roles. To perform all these activities, a user needs to sign in using username and password which will take him to the following screen.

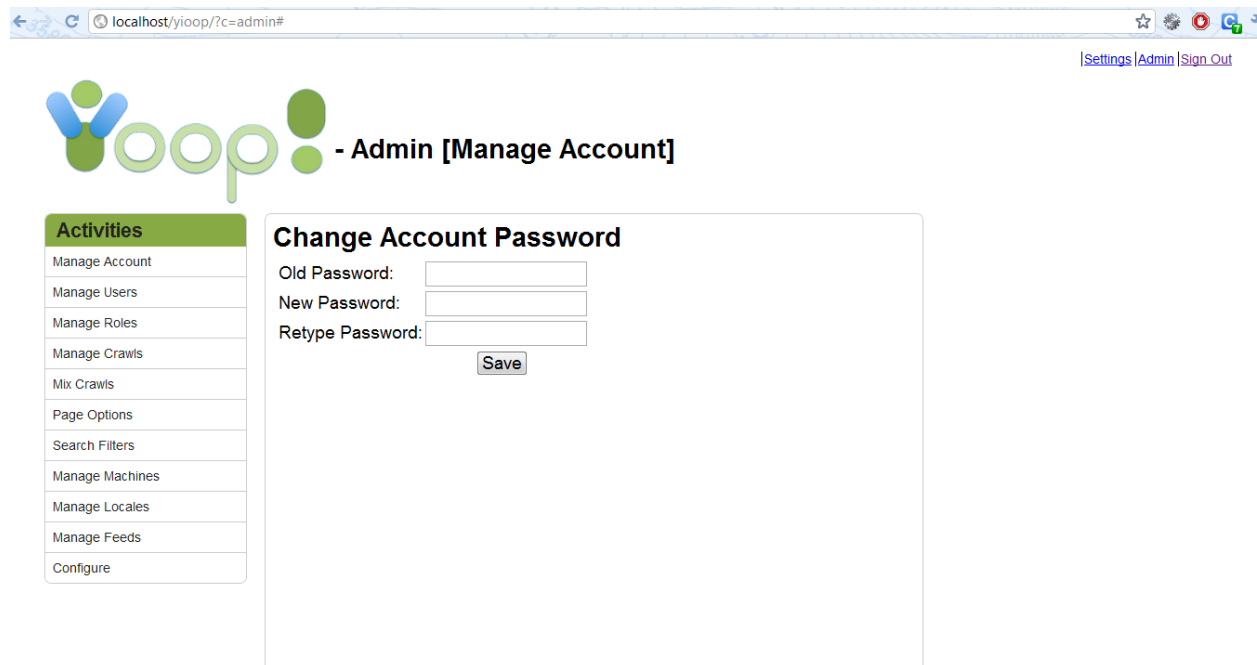
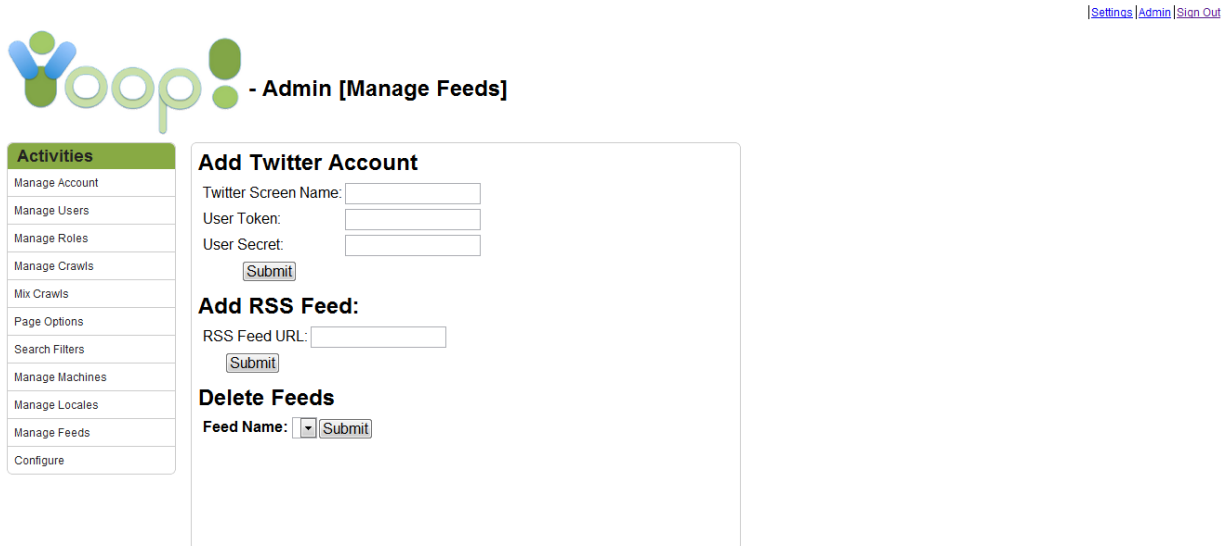


Figure 22: Admin Panel user interface (manage account)

The Admin panel by default takes you to the Manage Account activity which can be used to change your account password. All the activities are listed on left side navigation menu. Manage Users and Manage Roles provide you the users and role management. Manage crawls lets you create a new crawl and manage it. Configure activity deals with the various configurations related to setting up work directory, robot information and name server set up. Clicking each of the activities loads the user interface for that particular Activity.

Modifications to the Admin panel:

To support the management of the feeds a new activity is added in the Activities list. The manage feeds activity lets you to manage your social content.



The screenshot shows the 'Manage Feeds' admin panel. At the top right, there are links for [Settings](#), [Admin](#), and [Sign Out](#). The main header features the 'voop!' logo and the title '- Admin [Manage Feeds]'. On the left is a sidebar menu titled 'Activities' with the following items: Manage Account, Manage Users, Manage Roles, Manage Crawls, Mix Crawls, Page Options, Search Filters, Manage Machines, Manage Locales, Manage Feeds (highlighted), and Configure. The main content area contains three sections: 'Add Twitter Account' with input fields for 'Twitter Screen Name', 'User Token', and 'User Secret', followed by a 'Submit' button; 'Add RSS Feed:' with an 'RSS Feed URL' input field and a 'Submit' button; and 'Delete Feeds' with a 'Feed Name' dropdown menu and a 'Submit' button.

Figure 23: Manage Feeds View

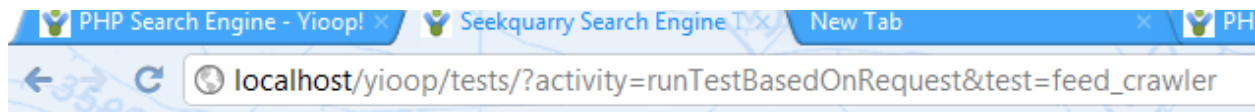
User can add their Twitter account by providing the input for these forms. Twitter screen name, user token and user secret for the twitter application. Add RSS feed needs just the RSS feed URL and clicking on submit would add it to his account allowing him to see the results matching feeds from this RSS link. Users can easily delete the feed account just by selecting the Feed name which contains both the RSS feeds unified resource links (URL) and the Twitter screen name.

5 Testing and Results

The testing of the feed search is done by using the unit test framework provided by the Yioop! search engine.

5.1 Unit Tests

The unit test classes were written and placed under tests directory. Feed Crawler and Feed Model are the two important classes for which unit tests were written.



SeekQuarry Tests

[See test case list.](#)

FeedCrawlerTest

testFeedCrawlerUserIDTestCase	1/1 Tests Passed
testFeedCrawlerUserKeyTestCase	1/1 Tests Passed
testFeedCrawlerUserSecretTestCase	1/1 Tests Passed

Figure 24: Results of the Unit Tests for FeedCrawler using unit test framework of Yioop!

The FeedCrawler is tested for a function to check whether the function retrieves back the correct user ID, user key and the user secret for the current user logged in. This is important in order to retrieve the results feeds from the right user's account.



Figure 25: Results of unit tests for FeedModel using unit test frame work of Yioop!

The second unit test class was written to test the FeedModel class which is responsible for the database access and the ranking of the result set. This test function checks whether the database is queried for the right key word and compares the result set for the matching user data.

5.2 Usability Test

Usability walkthrough testing [13] is a technique used in user oriented interaction designing to evaluate the effectiveness of a particular user interface on the users. The design is evaluated by testing it on users. The main goal of our usability tests for the search interface is to measure the ease-of-use or usability of the social search results.

In the course of these tests, five users who have a fair knowledge of using computer and browser were chosen and were asked to perform search and arrive to the specific result among the result set. All the users were aged between 18 – 28 years. Users were asked to talk out loud while they were doing the test and interviewed after the test.

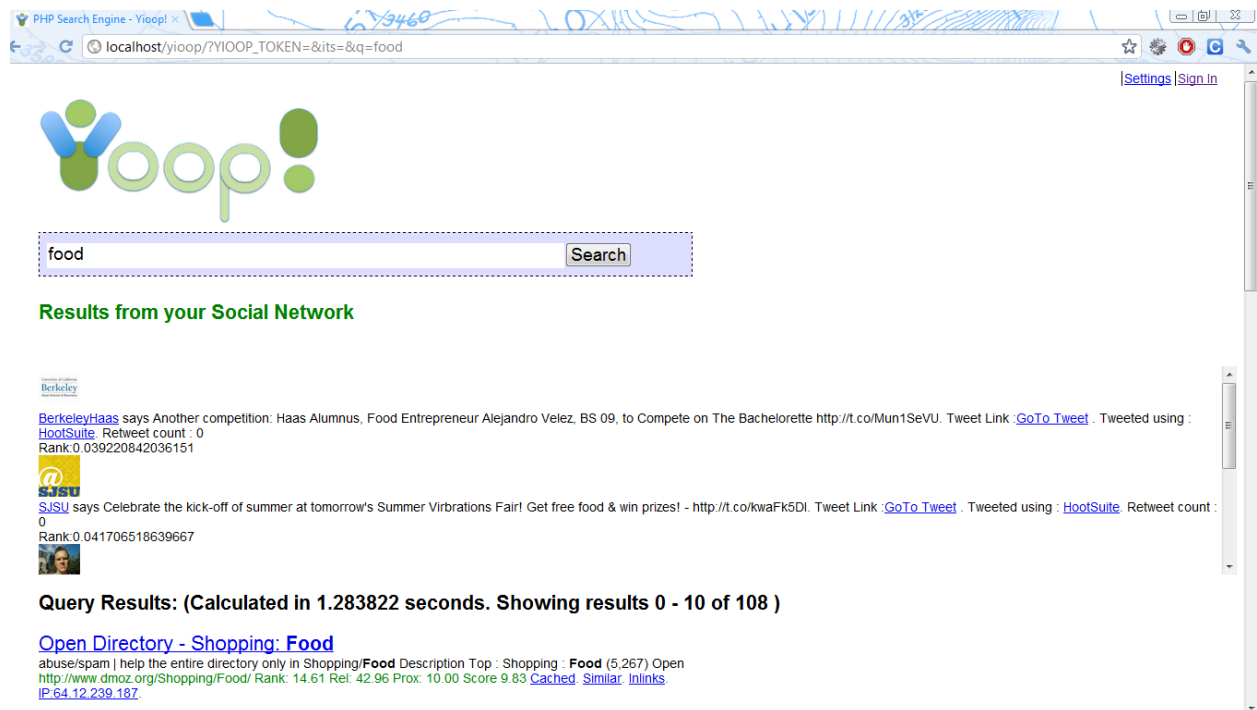


Figure 26: First iteration of user interface created for the search results

The initial user interface created for showing the search results showed the search results from the user's social circle before the web search results with a feed result box showing 3 results and a scroll bar added to browse the next results from the social search. The overall results of this test indicated that although it makes the social results visible to the user, it hindered them from being able to see the web search results. As a result of this next iteration of the user interface is created to minimize the cluttered experience for the users to see the web results.

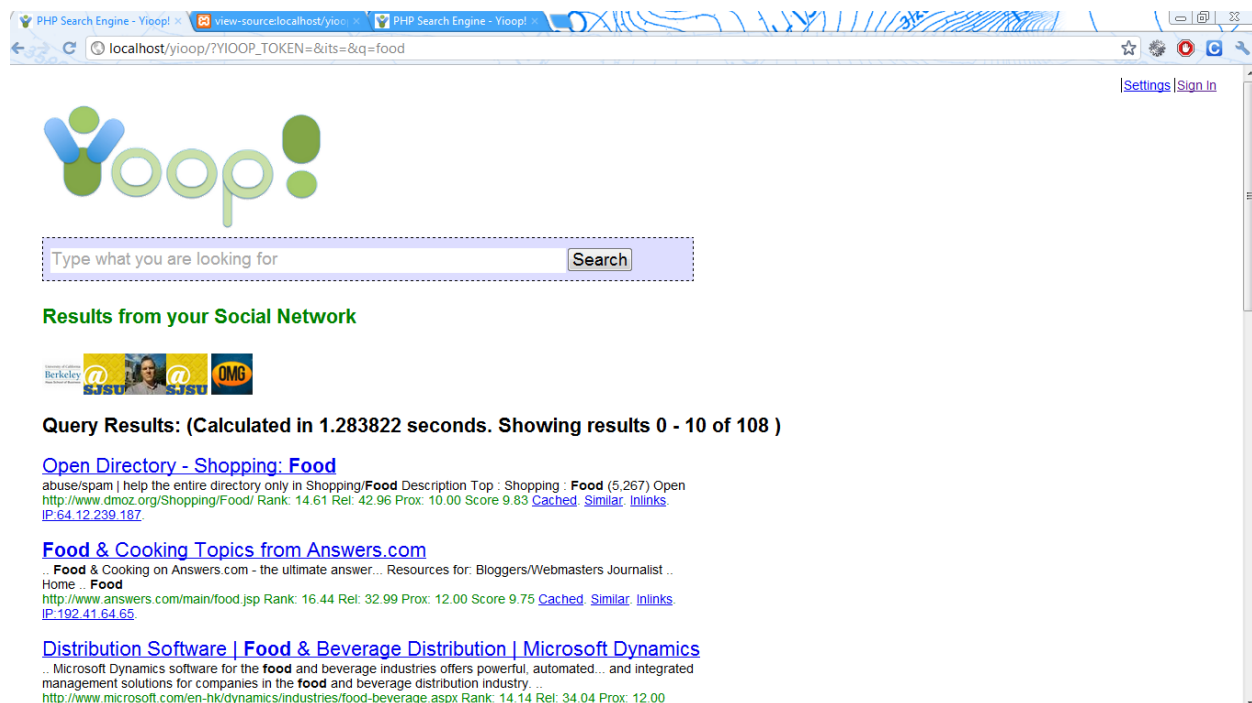


Figure 27: Second iteration of the user interface for Search results

In this iteration, the user interface was changed to display only the icons for the social search results. Each icon is an image of the source of the particular search result's profile picture on the network. This was well received by the user's but it took them a lot more time to arrive at the correct result showing the lack of information and ease of use.

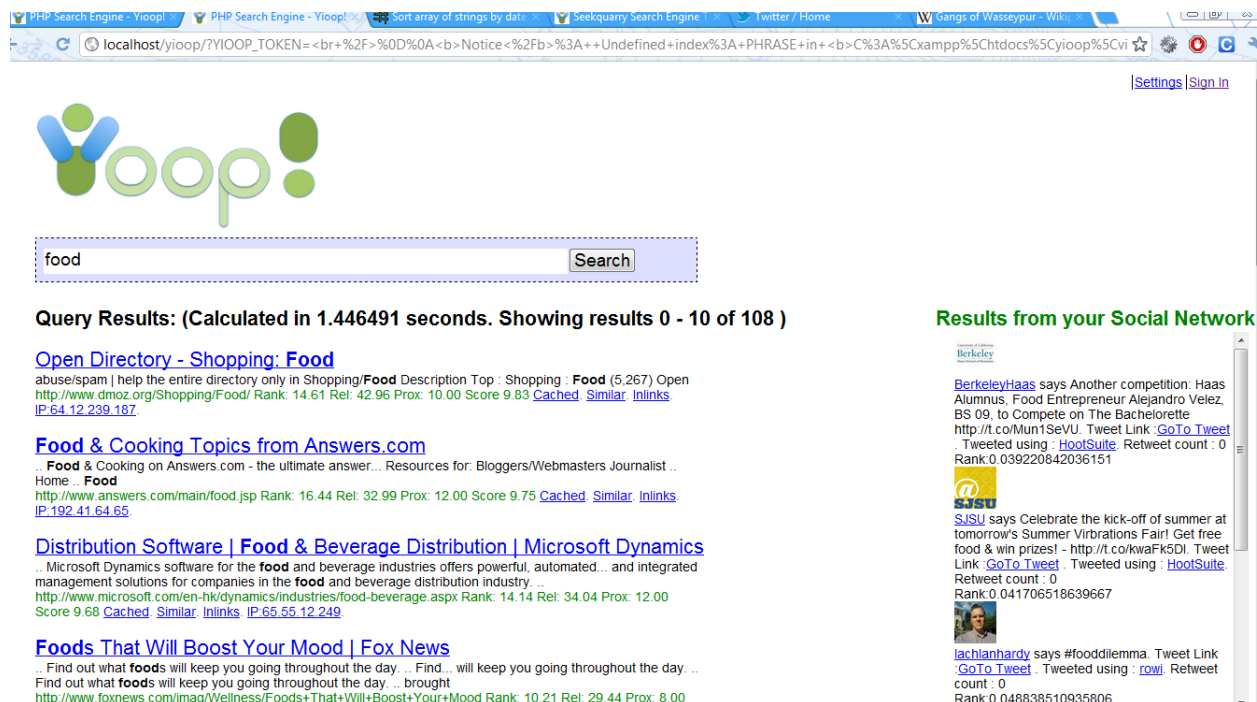


Figure 28: Final user interface for the Social search results

The final prototype of the user interface was had a side bar that had the results from the social network of the user. Each result consists of the profile picture of the source and the feed text along with other information and a link to the tweet. This interface clearly separated the web results from the search results. This design was chosen since this clearly separated the web and social results along with the ease of use.

The following graph results are from the usability test conducted for the three user interface prototypes.

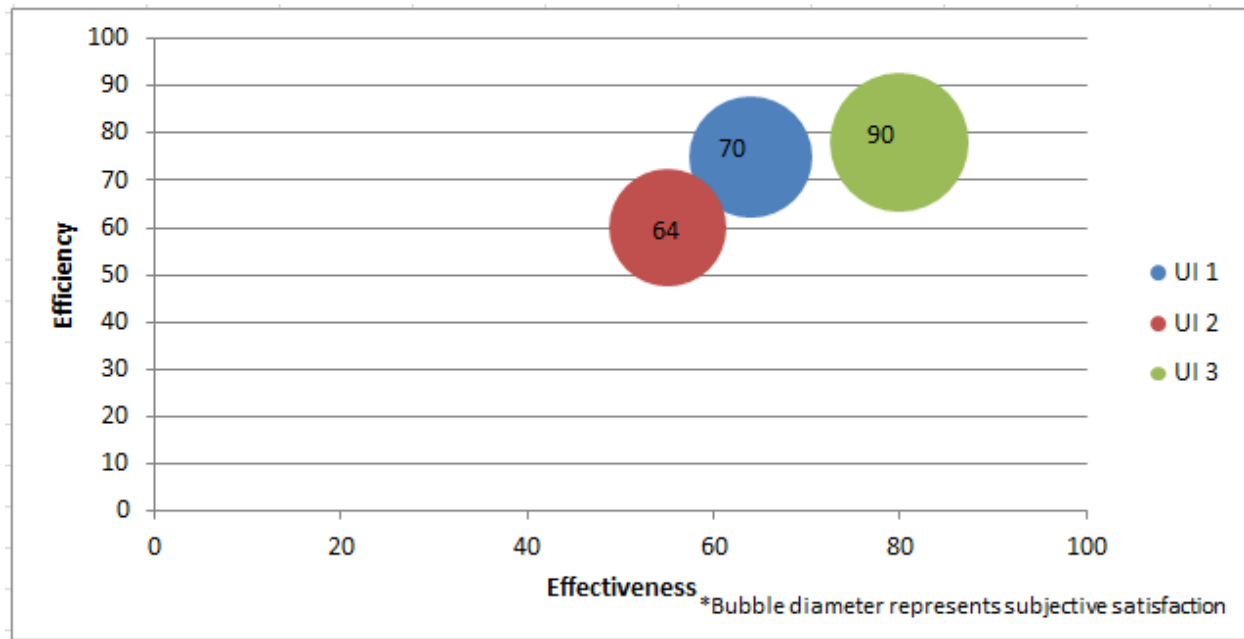


Figure 29: Graph showing the relative usability of the three user interfaces in comparison

Usability walkthrough is one way of evaluating usability. In this method users are given a task and upon completion user rates the effectiveness, efficiency and subjective satisfaction. Efficiency is the actual task time and Effectiveness is the measure of task completion relative to the number of errors done. Satisfaction is subjective and is measured from user's emotion towards the user interface.

Task	Effectiveness	Efficiency	Satisfaction	Overall
UI Prototype 1	64%	75%	70%	69%
UI Prototype 2	55%	60%	64%	60%
UI Prototype 3	80%	78%	90%	83%

Table 1: Usability test results for three different user interfaces

Search query results

The social search results are more personalized to user as it is from the content from his connections in the social networks. Our social search results uses metadata related to each post to rank the feeds. Search results are subjective to the user and dependent on the factors like the number of people he follows, their knowledge relevant to the topic and the amount of feeds they post. We compare here our social search results with another feed search engine Wajam launched in December 2011.

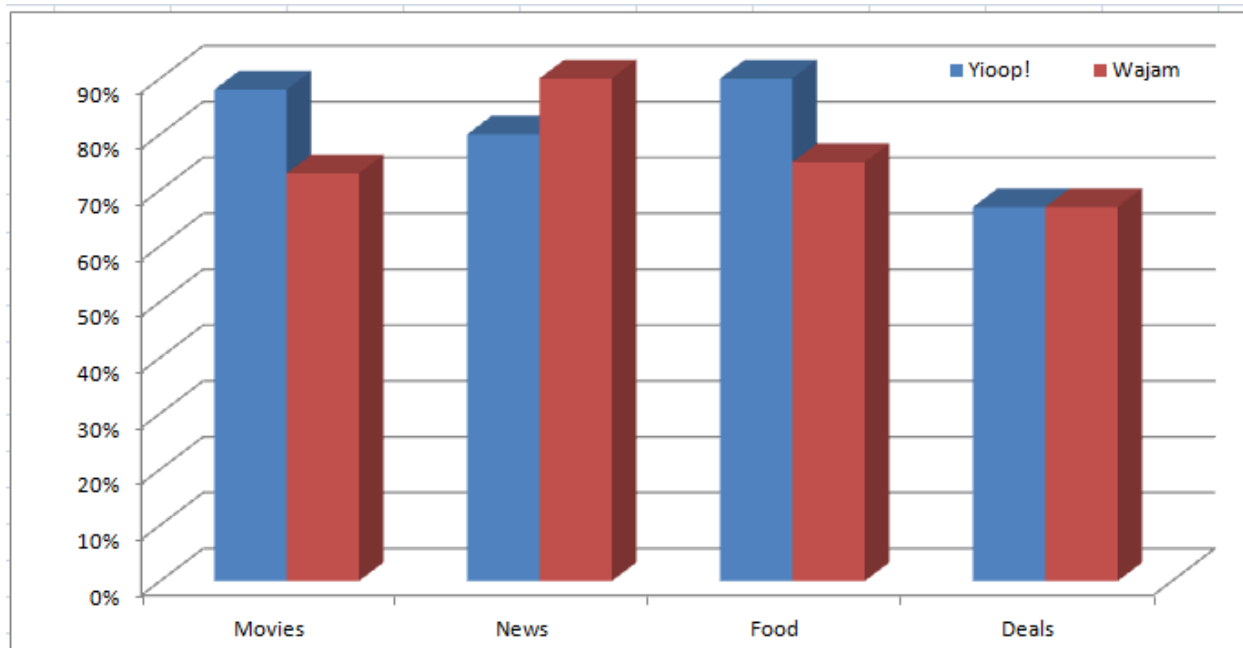


Figure 30: Relevancy of results comparison between Yioop! social search and Wajam

6 Conclusion

With the information exchange over the social networks growing largely over the years, it can no longer be ignored to use this information in a search over internet. There is a need for social data from user's network to be a part of the user's query results. The aim of my project to provide the social content relevant to user from user's social network and selected feeds to be a part of their search results integrated with Yioop! search engine.

In this project, a social feed search engine is implemented by applying Reverse Reciprocal Rank Fusion[6] ranking on the social feed data. The data is collected from the user's social network account from Twitter and also from the RSS feeds subscribed by the user through Yioop!'s Manage Feeds activity. The same search query box that lets you type in your web search allows you to see results from both the web search and your personal social circle. This will help users get valuable information when they search on movie reviews, restaurants, and social events and so on.

Yioop! is a great platform for implementing this social feed search engine. Yioop!'s provided the necessary platform with web based model –view-controller framework for the integration of social feed search engine.

7 References

1. [2009] Information Seeking Can Be Social. Ed H. Chi. Computer, vol. 42, no. 3, pp. 42-46, March 2009, doi:10.1109/MC.2009.8
2. Twitter Team (2012-03-21). "Twitter Turns Six". Retrieved May 19, 2012 from Twitter Blog (blog of Twitter) web site: <http://blog.twitter.com/2012/03/twitter-turns-six.html>.
3. RSS 2.0 specification. Retrieved on May 19, 2012, from web site <http://www.rssboard.org/rss-specification>
4. [2010] Search in Social Networks with Access Control. Truls A. Björklund, Michaela Götz, Johannes Gehrke. ACM. 2010.
5. Twitter Documentation. Retrieved on May 7, 2012, from web site: <http://www.dev.Twitter.com/docs>
6. The OAuth 2.0 Authorization Protocol. Retrieved on Dec 5, 2011, from web site: <http://tools.ietf.org/html/draft-ietf-oauth-v2-22>
7. Facebook Documentation. Retrieved on May 19, 2012, from web site: <http://developers.facebook.com/docs/>
8. Delicious developers resources. Retrieved on May 19, 2012, from web site: <http://delicious.com/developers>
9. Yioop! Documentation. (n.d.) Retrieved May 7, 2012 from SeekQuarry web page: <http://seekquarry.com/?c=main&p=documentation>

10. [2011] Workload-Aware Indexing for Keyword Search in Social Networks. Truls A. Bjørklund, Michaela Götz, Johannes Gehrke, Nils Grimsø. *In CIKM, 2011*
11. tmhOAuth by Matt Harris. Retrieved on May 19, 2012, from web site: <https://github.com/thematharris/tmhOAuth#readme>
12. Reciprocal rank fusion outperforms Condorcet and individual rank learning methods. In SIGIR '09: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval (2009), pp. 758-759, doi:10.1145/1571941.1572114 by Gordon V. Cormack, Charles L. A. Clarke, Stefan Buettcher.
13. [1994] the cognitive walkthrough method: A practitioner's guide. Wharton, C., Rieman, J., Lewis, C., and Polson, P. In Nielsen, J., and Mack, R. (Eds.), Usability inspection methods. New York, NY: John Wiley & Sons, Inc
14. [2002] Identity and Search in Social Networks. Watts, D. J., Dodds, P., & Newman, M. J. Science, 296(5571), 1302.
15. . [2006] Exploiting Social Networks for Internet Search. Alan Mislove, Krishna P. Gummadi, and Peter Druschel. In Proceedings of the 5th Workshop on Hot Topics in Networks (HotNets'06).