Summer 2011

Social Network Leverage Search

Payal Gupta
San Jose State University

Social Network Leverage Search

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Payal Gupta

Spring 2011

ABSTRACT

Social networks are at an all time high, nowadays. They make the world a smaller place to live in. People can stay in touch with friends and can make new friends on these social networks which traditionally were not possible without internet service. The possibilities provided by social networks enable vast and immediate contact. People tend to spend lot of time on the social networks like Facebook, LinkedIn and Twitter peeping into their friend's accounts and trying to stay connected with the world.

However, recently people have started closing their accounts on these famous social networks after having been irritated with the large amount of data that floods these networks. Although there are many problems associated with these social networks like: privacy issues, identity fraud, information overload, etc.; the problem that bothers people the most is that of information overload.

This project provides a solution to the information overload problem by filtering all the user's friend's posts on the basis of user's likes without explicitly asking the user to specify their likes. The project analyzes the user's posts to find out their likes, and then returns the filtered posts to them from their friends on Facebook, Twitter and LinkedIn.

Thus, this project attempts to remove noise from the huge amount of data on these social networks.

ACKNOWLEDGEMENT

Firstly, I am grateful to my advisor, Dr. Robert Chun, for his continuous assistance and support. In the initial stage I was novice to this subject but Dr. Robert Chun gave me ample time to understand and research on the subject. He provided valuable guidance and help. Without his help it would not have been possible to finish this project on time.

I greatly appreciate Dr. Teoh Soon Tee and Mr. Snehal Patel's contribution as thesis committee members. Lastly I thank my family and friends for keeping faith in me and providing me with support during the project.

Thank you.

Contents

LIST OF FIGURES

1. INTRODUCTION

The various technological advancements have brought change in the way service providers correspond to their customers' needs. In spite of a larger business focus on service, a big number of companies are still failing to meet up their customers' service requirements and expectations. One way to help improve customer service would be to come up with a complete performance management solution to make sure that their product is providing the preferred level of service. The right technology will help put the right things in the right place at the right time; performing the right tasks to deliver the right results. The Internet has now reached a "tipping point" and developers have begun to do a lot more than just creating their own applications and making them available to the users. Developers have started combining two or more Web services to create useful combinations. This helps the users to make decisions by analyzing and comparing data from various sources. Mashup is a way to enhance the ability or increase the capability on the run without wasting time to train the new employees or spending money in building new infrastructure.

"SOCIAL NETWORK," I am sure that no one is unaware of this term. The popularity of social networking sites has increased by leaps and bounds in recent years. Everyday internet users, businesses and social organizations have chosen social networking websites as an unmatched resource. Social networking sites offer a variety of methods to implement end users' generation of contacts and opportunities. Although many sites are limited in complexity, their popularity is spurred by ease of use and familiar processes. UKOM General Manager, James Smythe,

explains: “Over the last seven years, we see two broad developments: first, huge growth in the use of sites built on social content, where we mostly find contributions from people we trust; and second, websites with a high-street or ‘real-world’ presence translating the strength of their offline brands into online audiences.”

Taking into consideration the massive amount of social networking websites accessible on the internet these days, picking the paramount one that suits our business needs the best, often turns out to be really difficult and sometimes, even unfeasible. Users often find that one particular social networking site does not fulfill all their requirements. So to target audience masses, companies need to integrate more than one social networking site and this is when the proper integration of sites and a more useful product comes into picture. Companies need to provide a place to the users where they can effortlessly carry out both professional and personal communications at the same time.

1.1 PROBLEM ADDRESSED

With the increasing use of Internet and sky touching popularity of social networks, there is not only a need to stay up-to-date with the day to day events going on all across the globe, but also to stay in touch with one’s professional and social life. The pressures of handling the information from all the sides can cause a serious problem called information overload. Thus, information overloading can be defined as getting besieged by one’s information demands.

At this point, we are still at the very beginning of trading data in the social networks. We've been introduced to four kinds of things that were hard to do or simply not available, before. These things are:

Profiles – Summaries of online conceptions and expressions of individuality or group affiliations.

Connections – Channels between ourselves and others, or a connection between groups in our "network."

Content – The various media: text, pictures and video that are shared online.

Activities – A demonstration of what we are doing on our own and conglomerate networks. [1]

In the past, "media-driven" globe, the content pillar was truly the only angle that we perceived. Information processing became strategically developed to deal with content-overload by going through the data posted by "trusted" sources only, and even from those trusted sources, only reading self relevant items. (For example, if we read a newspaper, we only read those articles that interest us and about things that we like)

In the case modern dynamic social networking, with its newly-polished profiles, links and actions, we need to recognize observation techniques. It becomes apparent that one does not implicitly need to respond to everything. [1]

The goal of the project will be to develop a mash up of API's from few very famous social networking sites that will combine various services such as: social, storage, search, etc. The basic

idea of the project is to use the social networking concepts and present the data to the user in more meaningful way.

2. RELATED WORK

"Social networking" has been around since the beginning of time. Basically, it is the act of introducing friends and expanding groups with friends becoming friends, and so on. Today, many people use Twitter and Facebook to advance their on hand and forthcoming businesses. Companies looking to expand their business-associated acquaintances usually shift to networks like LinkedIn.

Top 3 Social Networking Sites (December 2010)

Observing the figure below, it's seen that Facebook is the favorite social network among all the countries in the list. A remarkable growth of Twitter can be seen when compared to MySpace (in Italy, Germany, Canada and Australia). The measured but stable expansion of LinkedIn (in UK, Australia and Canada) can also be witnessed. The fields in the table that are in yellow are the new entries that were made after June. [2]

Countries	SNS #1	SNS #2	SNS #3
Australia	Facebook	Twitter	Linkedin
Canada	Facebook	Twitter	Linkedin
France	Facebook	Skyrock	Twitter
Germany	Facebook	Xing	Twitter
Italy	Facebook	Badoo	Twitter
Russia	V Kontakte	Odnoklassniki	LiveJournal
Spain	Facebook	Tuenti	Badoo
United Kingdom	Facebook	Twitter	Linkedin
United States	Facebook	MySpace	Twitter

Figure 1: World Map of Social Networks [2]

2.1 SOCIAL NETWORKS

2.1.1 FACEBOOK

Facebook, one of the most famous social networking sites, keeps us updated with our friend's information no matter which corner of the world our friends are. It allows us to see all the information that our friends post. This great invention called Facebook continues to expand users all over the world. Since June 2010, Facebook was able to take away the position previously held by its competitors in many countries. Facebook has held the number one position in 115 countries out of a total 132 countries analyzed. For example:

"- From Hi5: Mongolia

- From Nasza-Klasa: Poland

- From Orkut (Google): Paraguay and India

- From Iwiw: Hungary."[2]

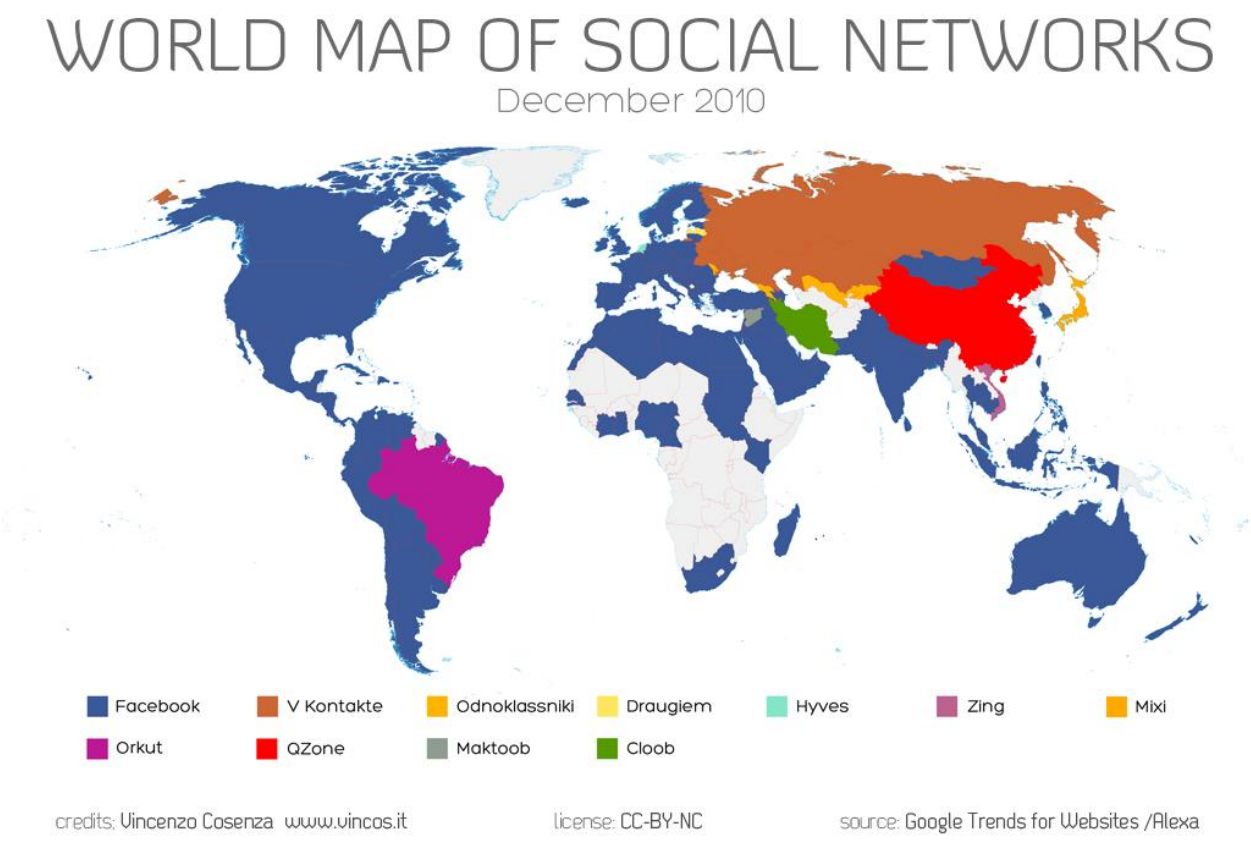

Figure 2: Facebook in blue across the world [2]

The following are the steps to be followed to access the functionalities provided by Facebook:

1. The user is required to create a new account on Facebook.com. The user does not have to pay to own an account as the service provided by them is free. The service requires the user to be at least 13 years of age to have a legitimate email account. Once the user is registered he/she can:
2. The user can connect with friends all over the world by searching for them using their names, identities, etc.

3. The user can join some networks of interest by browsing them based on the user's location, education, workplaces, etc.
4. The user can share and connect to people from other web based accounts by importing them to the Facebook account.
5. The latest figures show that Facebook already has more than 500 million users and is one of the top social networking sites across the globe.

2.1.2 TWITTER

Nowadays, Twitter is acquiring recognition as one of the favorite social networking sites amongst of all the providers of social networking services. Users on Twitter can send and accept messages to any particular network, instead of transmitting email messages individually. Users can also build their own networks and can allow people to follow other's posts and receive tweets from other member users. The user is given full control of his information. The user is given the privilege to leave any network any time he/she wishes and to stop receiving tweets from any person whose meaningless posts irritates him/her.

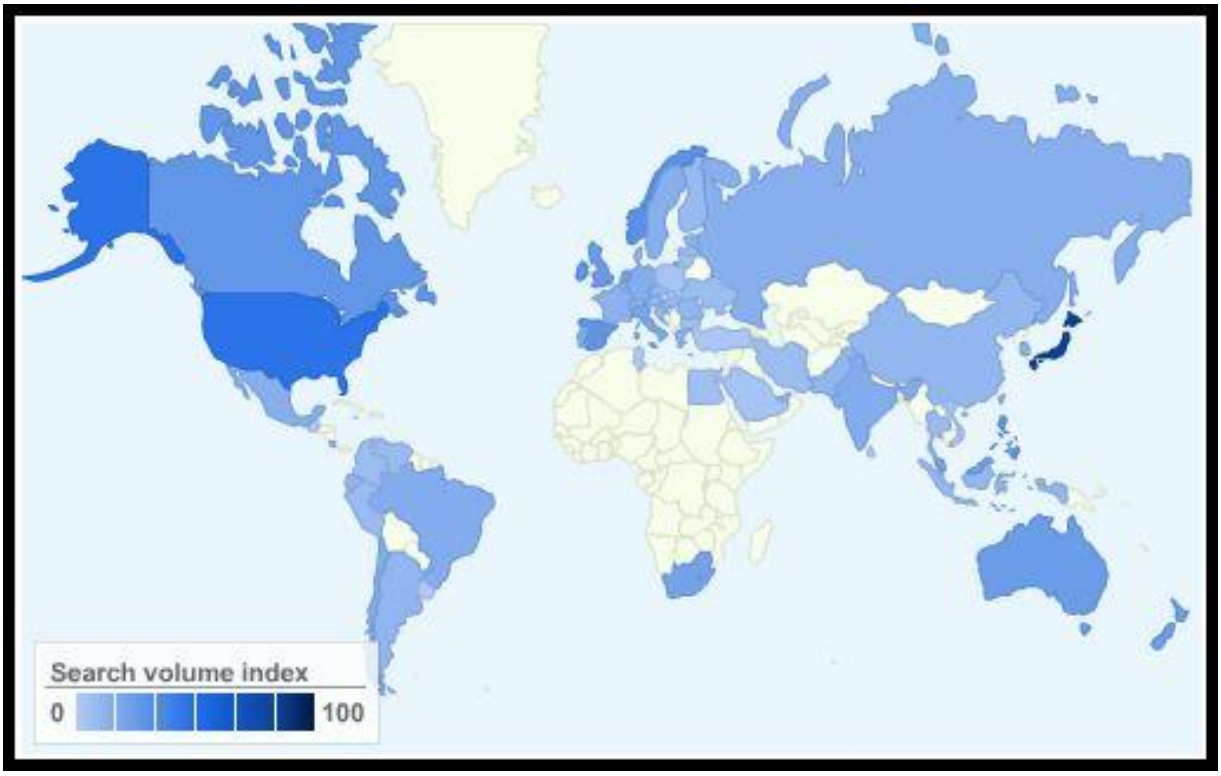

Figure 3: Popularity of Twitter across the Globe [4]

Countries with highest traffic with Twitter:

”

1. Singapore
2. Japan
3. Hong Kong
4. United States
5. Taiwan” [4]

2.1.3 LINKEDIN

LinkedIn is a social network choice for professionals. The users on LinkedIn can find a job based on their field of interest. Its professional networking design is focused to help members find work and connect with prospective business associates. LinkedIn is different from other famous social networking sites as it mainly deals with sharing content and socializing.

To register with LinkedIn, the user is required to create an account with them. The user must fill out a profile page. The user has to fill in personal information, all of which is editable at any point of time. To add to this, users can get suggestions and recommendations from their colleagues and bosses. Results show that there are already about 75 million users registered on LinkedIn.

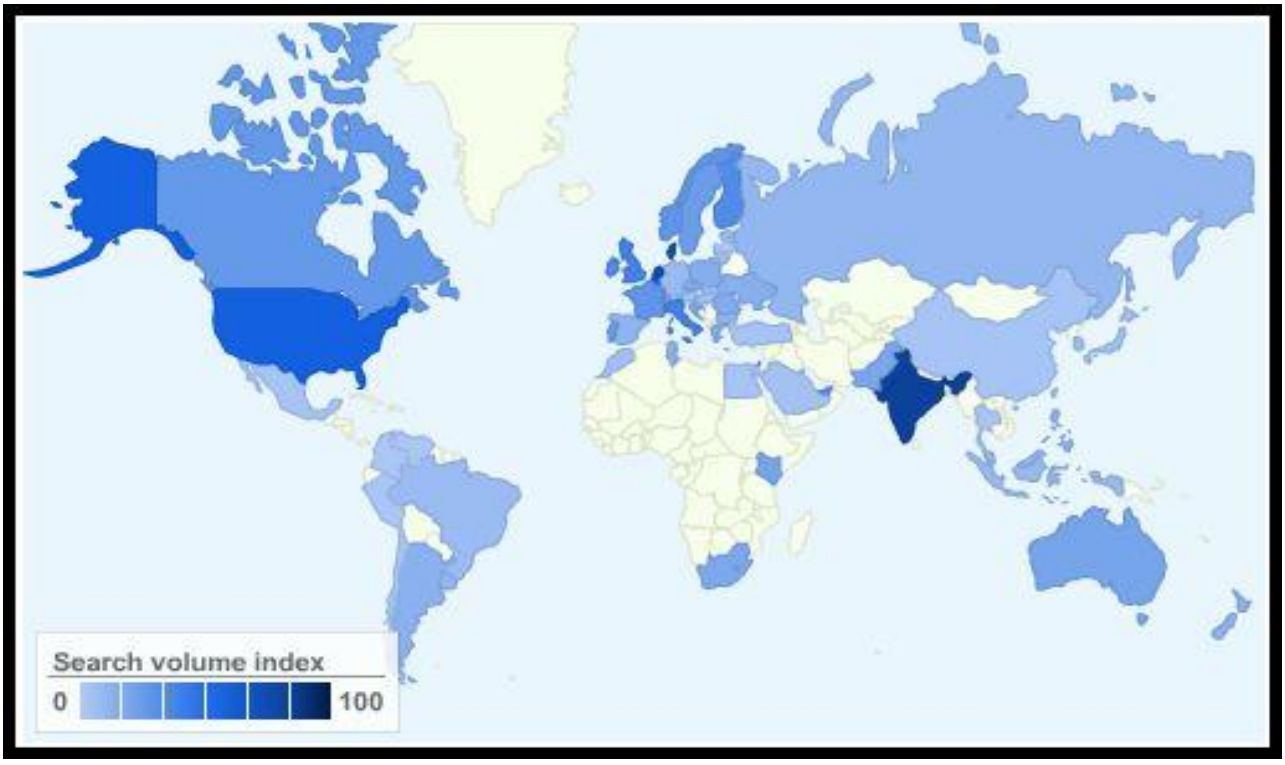

Figure 4: Popularity of LinkedIn across the Globe [4]

“ Countries with highest interest in LinkedIn:

- Denmark
- India
- United State
- Netherlands
- Belgium” [4]

2.2 PREVIOUS WORK

When I started with the project, I was sure that I wanted to do something valuable in the social networking area. My interest in the field and the boom of social networking in today’s world are my primary reasons for this report.

After few attempts, I was able to come up with the feasible and very useful idea of filtering the user’s friends' posts from the three major social network names: Facebook, Twitter and LinkedIn, and present them in a meaningful way to the user.

In this section I would like to describe in brief about my previous attempts to build something unique in this area of social networking.

Idea 1:

Topic: To implement search on the social graph.

The goal of the project was to develop a mashup that combined various services such as: social, storage, search, etc., from various available API's. The basic idea of the project was to implement a search on the derived social graphs stored in a database.

The project consisted of three main parts. The first part consisted of deriving the social graph from the general information of my friends available from various social networks like LinkedIn, Facebook, and Twitter.

In the second part, I tried to build a website that pulled articles on the run and I also developed a database to store the social graph of my friends.

In the third part, I tried to implement the semantic search on the social graph. Let's follow an example to understand this idea. For example, there is one article from the New York Times on my website. The user can see how many people viewed that article. Then, this search can be implemented on the social graph such that we can determine our friends out of the people who read the article.

This kind of service would have helped the user, as he would have been able to see how many of his friends liked a particular article; and seeing that, he might have shown interest in reading that article. This assumption was derived from the general human psychology where users prefer to watch the video that's being uploaded by more than 1 friend rather than the one that's being uploaded by only one friend. It also would have proven beneficial to the service providers, like

the New York Times, to increase its sales and provide better and quick service, thus gaining better user satisfaction and support.

Initially the challenge was to determine **who** read the article on my website. While the project was still in the research phase, Facebook came up with something called "Social Plugins." These plugins can be added to any website to see what my friends on facebook have liked. Adding a social plugin is easy and does not require much work. So my idea of detecting who read the article came to an end, there.

Idea 2:

Derive a social graph of my friends and store it in the database. I used three social networks for this:

- Facebook
- LinkedIn
- Twitter

The system used beencounter API to browse the web history of the users for a particular website. For example, to search the user's web history for www.google.com/analytics. The result returned by beencounter was something like:

Figure 5: beencounter API [5]

This graph shows that 72% of my site visitors view www.google.com/analytics.

On the basis of this data, I had the articles on my website. These articles had Facebook's social plugin. Users were able to "like" these articles and that were displayed on their respective accounts. By using google's API, I found out websites similar to www.google.com/analytics.

The result returned was something like:

http://www.webanalyticsassociation.org

http://en.wikipedia.org/wiki/Web_analytics

I utilized these similar websites lists to compare it with the browsing history of the user once again. If I found out that there were 3 or more matches between 2 users' web history, I used to find out the geographical location of those 2 people using Google maps API. Then I suggested them to be friends with each other on the basis of their location and likes. The idea behind this module was that we were tracking the users' web history for particular tags. On the basis of this list of returned websites, I found out how many people using my system had the similar list of websites returned. I suggested people to be friends with others who shared same interest.

There were some concerns for this application.

1) Peeping into user's web history did not seem a very good idea. There could have been privacy issue with it.

2) Scalability of the developed system was a question mark. Since I was trying to compare the list of websites with the user's browser history, the system had to make lot of comparisons. So the concern was, "what if million people are using our system."

Idea 3:

Filtering the user's friend's posts from Facebook, Linkedin and Twitter according to user's likes without the user explicitly specifying what he likes. This idea and the related work are being discussed in depth in this report.

3. PROBLEMS WITH SOCIAL NETWORKS

3.1 PROBLEMS WITH FACEBOOK

1) The top three social networks taken into consideration in this project have many problems with the type of system they utilize.
2) Users can get overwhelmed with friend requests from unwanted people or strangers.
3) As users share the content on these social networks, everybody including the user's family, can peep into the user's Facebook page.
4) Users often post their pictures on Facebook which can be misused in some ways. Moreover, people can be "tagged" in their pictures– making those pictures viewable by unknown people.
5) The user might accept the friend request from someone with tricky plans.
6) One will be flooded with all the updates posted by their friends no matter if the user is interested in them or not.
7) Any post posted by the user will similarly receive a large number of feeds.
8) Unwanted advertisements, annoying advertisements, and often spam can flood the Facebook page.
9) One can observe needless staging of irrelevant passages in the user's friends' lives.
10) If the privacy settings are not set properly, any Facebook user, even one not in the user's friend list will be able to view the user's content on Facebook.
11) There are some great applications provided by Facebook, but most of them require the user to provide access to the user's personal information.

12) And lastly, one fact that cannot be ignored is that Facebook is so addictive that it becomes a habit for most of its users. Hours are wasted on Facebook while one reads unnecessary information which otherwise could have been invested in doing something productive. [6]

3.2 PROBLEMS WITH TWITTER

These are several strong reasons for me to dislike Twitter. They are as follows:

1) It can really be snobbish. No one is interested in knowing what someone had in snacks today.
2) It can sometimes be annoying. Why would I want to share the details of my entire day with everybody?
3) The content can be redundant. If I already have a blog, why should I tweet?
4) Twitter is also addictive like Facebook. So, the amount of time wasted on Twitter could actually be used by doing something meaningful.
5) There are also some privacy issues. As the user shares all one's content, like where one is at any point of time on Twitter, the user could get followed by obsessive users. [7]

3.3 PROBLEMS WITH LINKEDIN

1) There does not seem to be a verification control on LinkedIn.
2) The inner organization is unidentified at LinkedIn and there is no direct way to reach the higher management.
3) There are not a lot of groups or communities to join on LinkedIn.

4) LinkedIn asks the users to pay to get the premium membership. But there is not a shared reimbursement given to the members of group organizations who are also the premium account holders on LinkedIn.

4. INFORMATION OVERLOAD

The everyday assault of information causes more frustration than its constructive output. One of the many influences that affect me is the abundant reception of email messages and feeds. The biggest concern for me is the large amount of information that I have to grasp and discover to be efficiently up to date with vital topics.

A research done recently infers that the continuous bombardment of too much information; and the fact that most of the companies allow access to these social networks at work; can impact one's capability to formulate apparent and crucial choices. It can also influence one's efficiency and individual well being. One of the studies suggest that people on average take around 25 minutes to get back to work following one email interruption. That does not sound very good for both the employees and the employer of the organization. [8]

4.1 INFORMATION OVERLOAD IN FACEBOOK

These days I hardly log in to Facebook and I have begun to speculate about what has brought that change in me. I mostly make use of Facebook to manage my contacts and events and in rare case upload pictures.

I’ve realized that I am finding it difficult to remove all the noise from the data on Facebook. Honestly, Facebook now happens to be a storm of information for me. There are definitely some very valuable parts on Facebook that satisfy my instant needs. For

example, I can receive a notification of an event that I am interested to attend and I can find my friend's contact details when needed.

However, the live feed from my friends now terrifies me and seems extremely dull.

Facebook, is highly infected with the problem of "information overload" and we do not have enough resources to deal with this problem. These days Facebook allows me to hide a few of my friends who create noise. It does allow me to see all the feeds from the friends that I most interact with, but it seems that the whole Facebook system is badly infected with the noise problem. I am not trying to say that Facebook is worthless because it publishes personal or other information which otherwise is difficult to find. I think the biggest drawback with Facebook is that it floods our brains with lots of worthless information.

Figure 6: Example of information Overload in Facebook

This screen shot above is from my Facebook account. It shows how one of my friends has posted worthless information and is flooding my home page with it every few minutes. This is how Facebook is suffering from this disease.

This project provides a solution to this problem of information overloading.

4.2 INFORMATION OVERLOAD IN TWITTER

We are yet at the premature stage of dealing with the devastating quantity of worthless data. Twitter just adds one more layer to it. Last year I removed all my friends from Facebook to clear my account's dorm. And this year, I was struggling to "unfollow" people every time I logged in to Twitter. However, that's was not the best solution to the problem and so I came up with a way to deal with it.

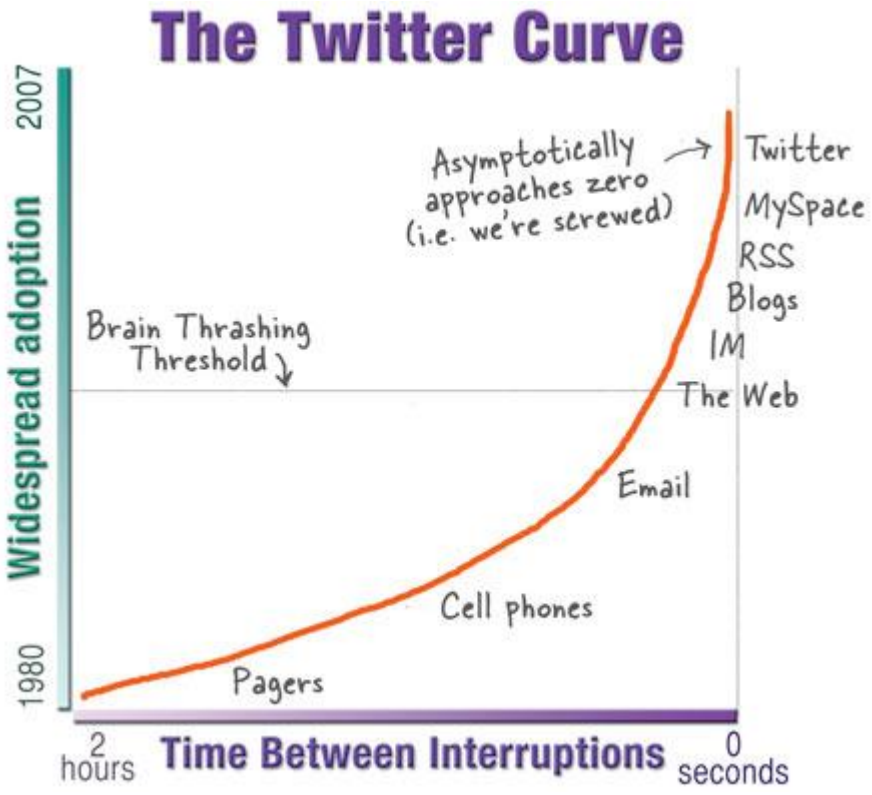

Figure 7: The Asymptotic Twitter Curve [9]

For all the people who know anything about Twitter, it has one and only one intention in life. A global community of friends and strangers answering one simple question: “What are you doing?” This adored question is answered by millions of people millions of time in a day.

Twitter allows users to post small posts to a kind of small blog provided by Twitter. Twitter is highly infected with information overload as well. Here are few examples of the kind of uninteresting posts that people tweet:

"Missed the my hair appointment, again."

"Attempting to figure out why my dog has his legs crossed."

"I just had a hiccup."

"I'm on my way to work."

"Scanning photos of cheerleaders doing high-kicks..."

"Getting very tired, now."

"Thinking about something delicious to eat."

"About to make a phone call."

"I'm watching my cat chase flies!"

"Totally sick of work. Not sure if it is my feelings or the work."

"Trimming my nose hairs. Fetching dinner." [9]

4.3 INFORMATION OVERLOAD IN LINKEDIN

Recently, LinkedIn is trying to match the popularity of its competitors and in that race it now provides many features that have been offered and that were a huge hit on Twitter and Facebook.

An example of this is follow and feeds. To add to it now, LinkedIn has joined hands with Twitter so that the tweets of the people can be directly posted on LinkedIn. This feature along with the Group Updates feature has increased the content generated by users on the LinkedIn network. Thus, Linkedin is not being spared the deadly disease of information overloading. [10]

5. DESIGN

"Rails" is the development framework for ruby used to create web applications. It was designed to make the programming of web based applications easier by assuming what all the developers might need to start building the applications. It allows one to code less, so that the use of other frameworks and languages becomes easier.

The main philosophy followed by Rails is:

- DRY – "Don't Repeat Yourself" – which infers that repetition of the code in a program is not a good idea.
- Convention Over Configuration – which means that rather than parsing all the configuration files; Rails will assume the user's intent to do something and will also assume the way he/she is going to do it
- REST allows one to speed up the application based on the resources available and HTTP verbs.

"Rails" is using MVC (Model-View-Controller) architecture.

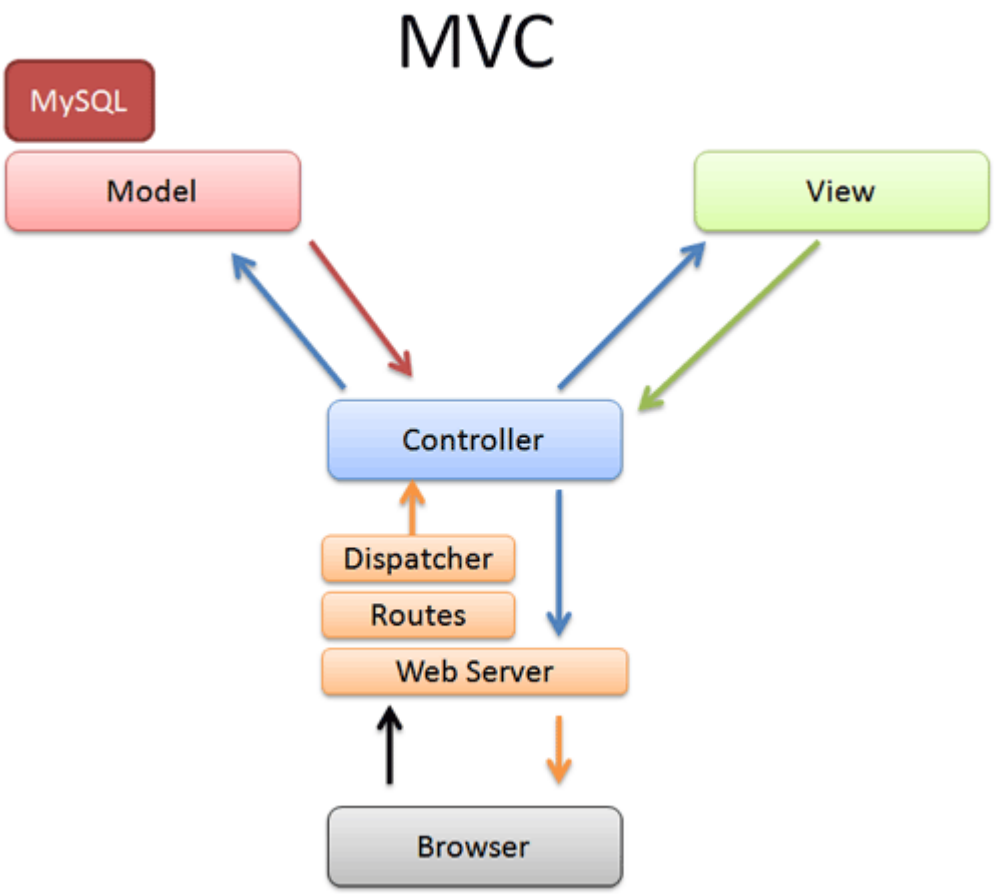

Figure 8: MVC Architecture [11]

In the middle of Rails is the Model, View, Controller architecture, called MVC. Its benefits include

:

- Isolating the UI from the business logic.
- The simplicity of keeping the code free of repetition.
- Making the maintenance easy by identifying different types to which the code belongs.

Models

A model is responsible for representing the data of the application and it also defines the rules to operate that data. For Rails, the rules are matched with the respective database tables. Therefore, one table is mapped to one model. Thus the model consists of an application's business logic.

Views

Views are responsible to present the UI of an application. Views in Rails are typically HTML files having the ruby code embedded within so it can perform the responsibilities related exclusively to the presentation of the data. Views take the data to the web browser that issues the requests from the application.

Controllers

Controllers do the work on gluing views and models. In Rails, they take the requests that come from the web browsers and then ask the models to render the data which the controller then passes to the views for presenting it.

Models

Authorization: The application is using OmniAuth gem for Facebook,Twitter and LinkedIn API authorizations.

This rack based system called OmniAuth eases the external authentication steps without assumptions made about the final results of the preferred authentication system.

Thus, this method allows the user to have new way of control over the application by authenticating it through Facebook,Twitter and LinkedIn.

Controllers (along with the actions they are responsible for):

Application (main controller, inaccessible directly): application controller in Rails basically implements functions which will be available in each controller. It is a class which is extended by each of our controllers. The Application controller in our app has functions to set/reset/identify logged in user, and a function to redirect guest to home page if it tries to access the user-only area.

redirect_if_not_user: Redirects to home page is user is not signed in

current_user: Gets current user in a variable

signed_in?: Is our user signed in? Checks if current_user variable exists

sign_out: Deletes current_user session

current_user=: Assigns new current_user

Authorization:

destroy: Delete current_user authorization for a specific social network; authorizations are essentially stored auth security tokens for each of the network. Our app uses them to fetch specific user posts.

Cron:

run: Calls Rake task to get new/existing user posts and rebuild tags. We won't need this controller when one has the application's source code in one's computer.

All server tasks are implemented using the Rake tool, which comes along with Ruby on Rails. Using Rake we can run the tasks that we need from the command line and there is no need to go to the web page. Rake tasks were used for

- getting the user posts from Facebook, Twitter and LinkedIn for the new users
- updating user posts for the existing users
- building tags from the user posts
- anything needed to search friends posts by tags.

Pages:

home: renders home page template (view) file.

Sessions

create: When our user clicks login with "facebook" (or other) button, Omniauth library redirects user to the facebook login page, then facebook fills in the security tokens and other information and redirects to session/create action if the user agrees to join our app. We update the user profile and store new "Facebook" authorization info. This action stores a session variable with user id, so our app can know this user in future requests.

destroy: User signs out. Deletes user_id variable from the session and redirects him/her to home page

CopusTerm: We have a corpus_terms table in our database in which the structure is: "id, term, count, created_at, updated_at." Here 'term' is the word and 'count' is the number of documents that have this word. Currently I'm using "Brown" corpus

Tag: Tags are the key words derived after analyzing the user's posts. They define the likes of the user.

User: Users are the customers using our system.

UserPost: User posts are the posts that are posted by the user on the three social networking sites taken into consideration in this project (Facebook/Twitter/LinkedIn).

Views: Views are html files with ruby code. They are like templates for our system. We used several layers of views. The first layer is called 'application' and it basically renders common elements like header, footer, navigation, and sign in block. The second layer is called actual action.

Currently we are using only two views here:

pages/home: home page template, renders tags for our users and sample text

users/edit: renders profile page, with a form to edit user name, and links to remove authorizations.

6. IMPLEMENTATION

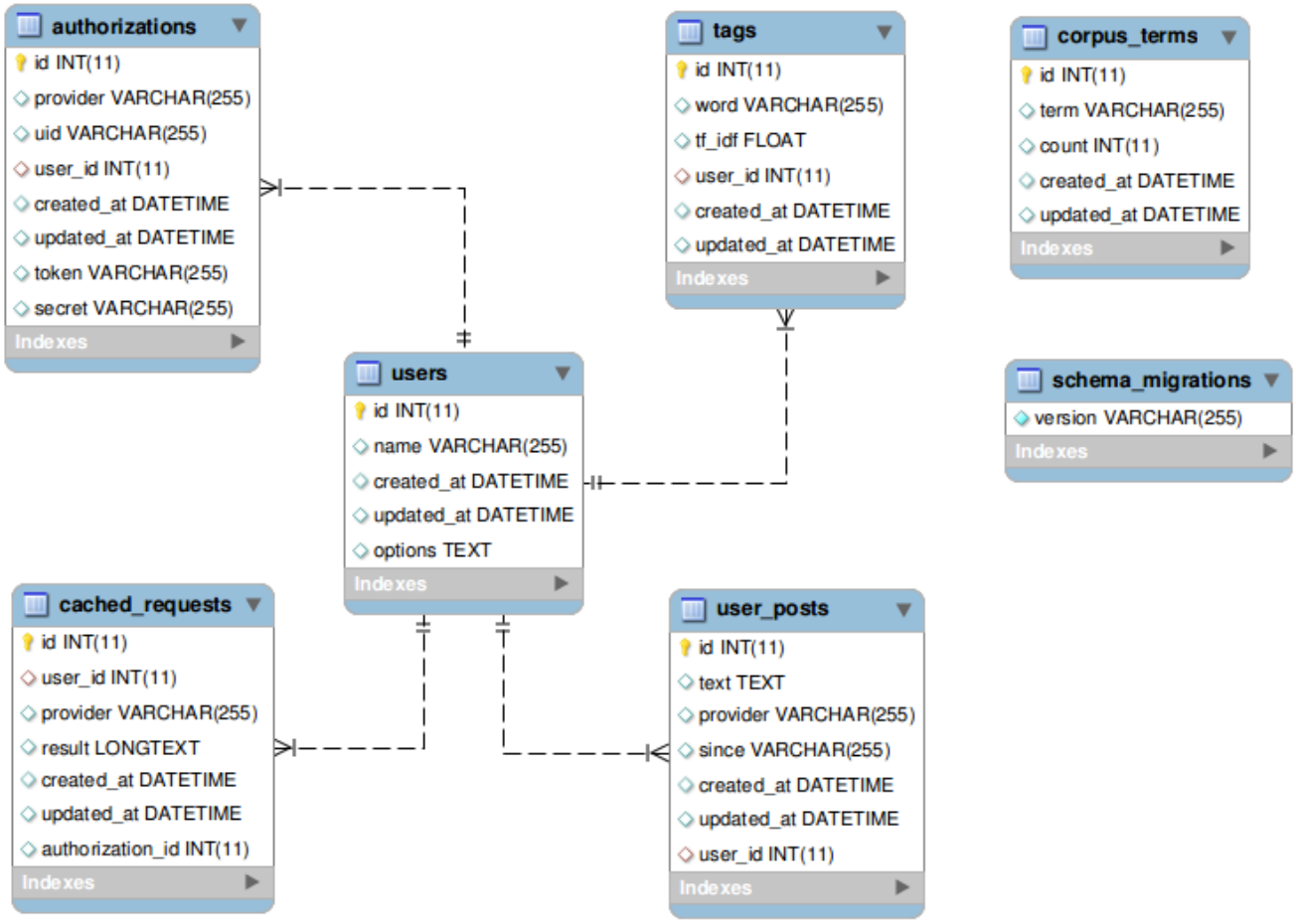

Figure 9: Database structure

User Sign up

Users can connect to the system using one of the supported social networks. The system is using OAuth authorization standard to authorize users with a social network account. We are using 'omniauth' ruby library to implement this functionality.

Omniauth works by redirecting users to a selected social network, where they will sign in using their credentials. By signing in, the user agrees to share his profile data with our system. After signing in, the system redirects the user to our callback url that we provided for the service.

Once we get a request to our callback url, we create a new **Authorization** model instance with 'token' and 'secret' parameters that we got with the callback request. The model is saved to database and the new User model is created if needed.

Next we add 'user_id' variable to session storage and the user is signed in. Any subsequent calls to sign in/connect new service will create a new Authorization model for the same signed in user.

Fetching user posts

Whenever the user signs in, we check if he has any posts in our database. These posts are stored as the User Post model in the database. If there are no posts, we use our Consumer classes to fetch the user's own posts for each connected social network. The new posts fetching action is also triggered by connecting the new social network for the user account.

The results returned by the consumer will be saved to the database, so now we have fetched user posts and can proceed to deriving tags.

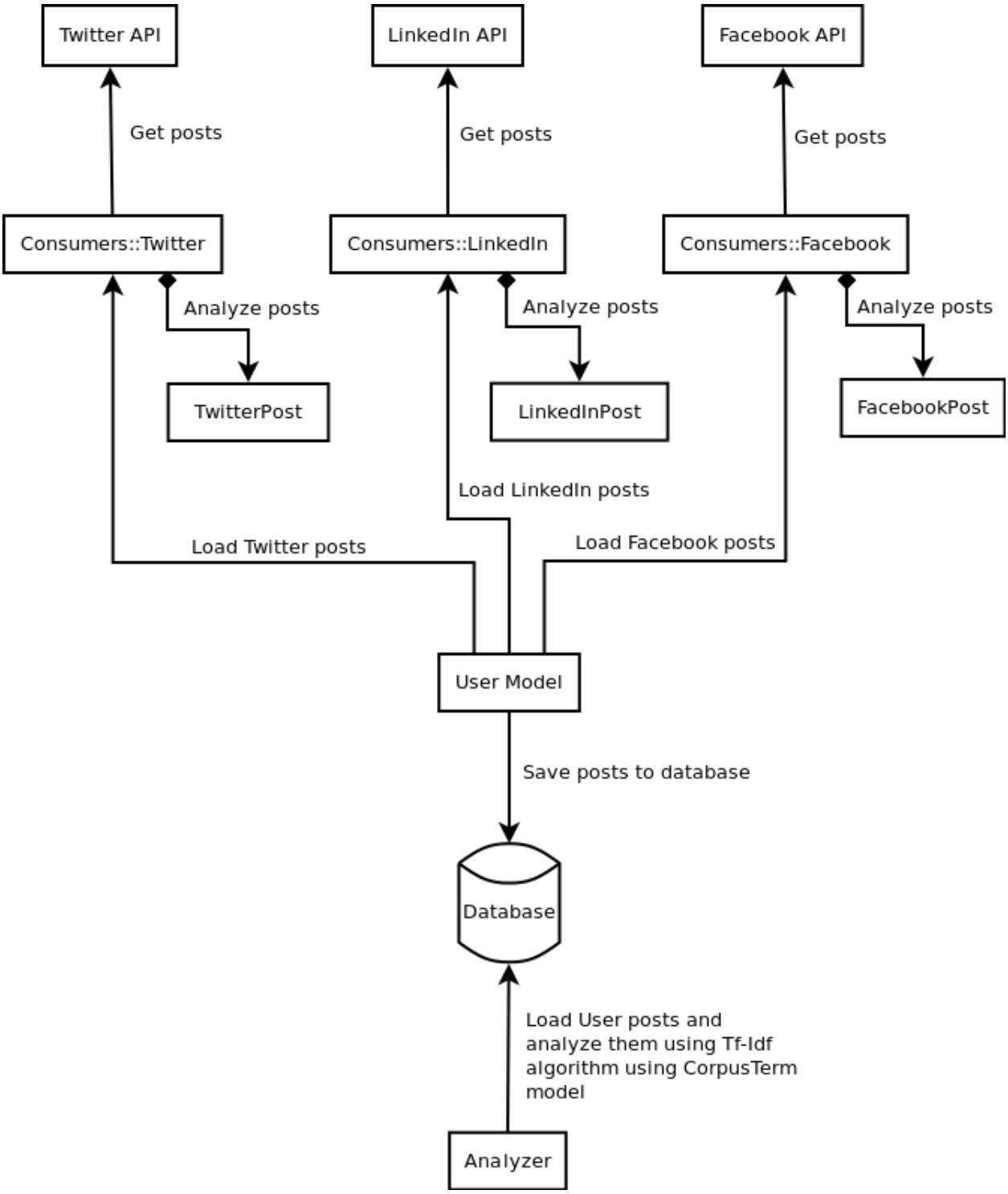

Figure 10: Deriving tags using TF-IDF algorithm

Deriving tags can be triggered by connecting the new social network to the user profile or running CRON worker task.

load_ * _posts is called first, which loads and saves all newer user posts in the db .Then Rebuild_tags method is called from the Consumer:* classes which is executed once the user signs up for a new service, or when we run a CRON/Rake task. Rebuild_tags method only checks for already existing user posts, it doesn't check for new user posts. It collects all user posts to a single document .Then we use the Analyzer class to create a list of possible tags with their Tf-Idf values and then the tags are added to the database if the TF_IDF value is high enough.

Analyzer class

Analyzer class is used to derive tags for a given document. We are using the TF-IDF algorithm to extract tags from the document. We begin by going through each word in the document and counting its Term Frequency (TF). Next, for each word, we check the CorpusTerm model for Inverse Document Frequency (IDF) and calculate the TF-IDF value. Lastly, the derived tags list is sorted according to the TF-IDF value and returned as a result.

Corpus

Corpus is used as the IDF part of the TF-IDF algorithm. Our system has a Rake task which loads all text documents in the 'corpus' directory and counts all the word occurrences in every document. So for example, if the word 'the' appears in 500 corpus documents, the system will create the CorpusTerm module with the word parameter as 'the' and count the parameter as 500. Currently our system is loaded with the Brown corpus which derives ~40,000 **CorpusTerm** models.

Consumer classes

Our system has one consumer class for every social network that we are working with.

Therefore, there are three consumer classes:

Consumer: Twitter

Consumer: Facebook

Consumer: LinkedIn

These classes are responsible for all the interactions with social network API's. Every consumer class will first contact a social network with a user Token and Key to get an authorized security token. We are again using the Omniauth library, here.

All consumer classes consist of methods to get the users' own and friends' posts. Those methods execute the API call with given parameters and instantiate the Post type class for each post in the result set.

When fetching the new user posts, we don't want to download old posts, so we pass the 'since' parameter to the API call, so we can only fetch the newer posts that we have in the database.

Post type classes

Our system has one Post type class for each social network:

TwitterPost

FacebookPost

LinkedInPost

These classes are used to extract the relevant information from the API call results. The TwitterPost class is the simplest one since Twitter only has one type of post. The FacebookPost is more complicated, because there are a lot of different types of posts, but we mainly focused on two of them: User post and User News (Wall) posts.LinkedIn is the most complicated of all, since it has many different post types and most of them have an entirely different structure from any other.

Therefore, Post type classes are used to abstract resulting objects to a common interface, which can reliably be used in various operations.

Fetching User's Friends Posts

The User's friend's posts were fetched once every 2 hours. It was possible to use a Rake task for an automated update, however, the system was configured to check and fetch posts if needed on every request.

The User's posts storage has strict regulations set forth by the social network's license. So we only stored posts for a short time within a model named "CachedRequest". We used this model every time the user requested to see a list of friend's posts.

Fetching the user posts is triggered by PagesController 'reload_cache' filter. This filter executes function in the User model– which checks for expired cached requests and reloads them before rendering pages.

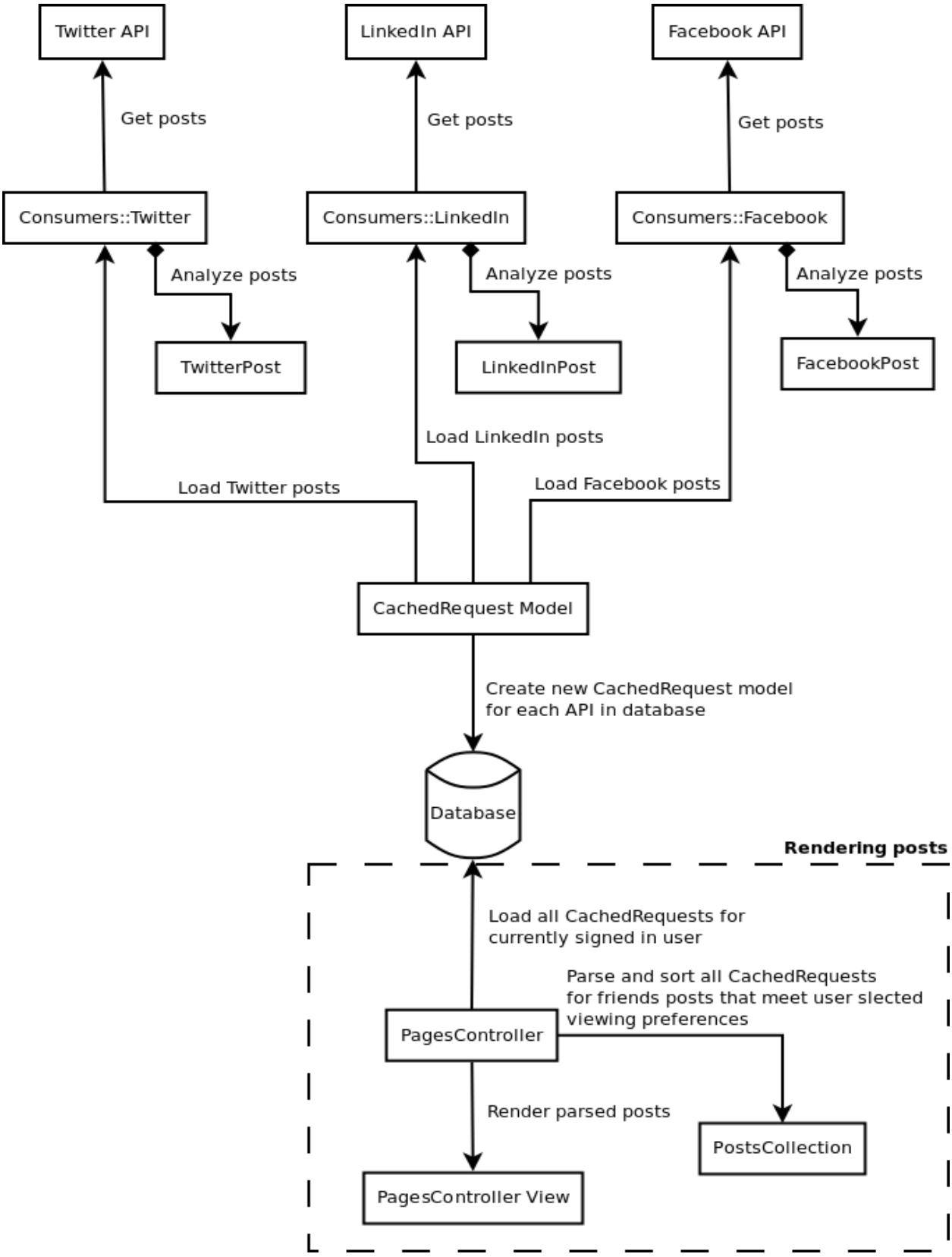

Figure 11: Deriving User's Friend's Posts

Cached Requests

The CachedRequest model is responsible for keeping a list of recent user friend's posts. This model will load the data for each social network connected to by the user. This data is stored in the database text column as serialized array of Post type classes. As a result, we did not need to create Post type class for each post again. Each CachedRequest model is deleted and recreated with new information every 2 hours.

PostsCollection class

PostCollection class is used by PageController to collect and order user friends posts. This class will collect only valid posts and exclude the users own posts from the list.

PostCollection class has the ability to order posts by date or relevance. For relevance, we need to pass a tag as a parameter for which the relevance should be measured.

This class first collects all posts or posts which has a tagged word if we pass 'tag' as parameter. Each post is analyzed for a tagged word when selecting the tagged posts. If a post includes the tagged word it goes into collection.

Once a collection contains all filtered posts, then we can use the class functions to sort them according to user preference.

Here is description of how relevance analyzing works:

Relevance analyzing in posts works by checking number of times given tag appears in post.

Posts having higher tag frequency appear higher in the list.

The Home page needs a different technique from the 'selected tag' pages, since we have no selected tag. We are displaying posts sorted by date or relevance here using same technique as mentioned above, but instead of analyzing all posts at once, we are splitting posts in blocks by tag. Highest showing tag will have highest TF-IDF value. We need to keep our home page as short as possible to minimize information noise, so instead of showing all posts for each tag, we are showing only 3 having highest relevance/newest date.

7. EXPERIMENT

The experiment was conducted to see how efficient the newly built system is.

There are two factors that can determine how efficient my system is:

Experiment 1: To see how much noise is removed from the three accounts for the user.

Experiment 2: How accurate are the results returned by the system.

Experiment 1: To do the first test we counted the number of posts on user's accounts. (facebook+ Twitter+ LinkedIn).For 10 users, the average results came out to be 52% that means out of 100 posts the user's accounts show; the system returned 48% relevant posts.

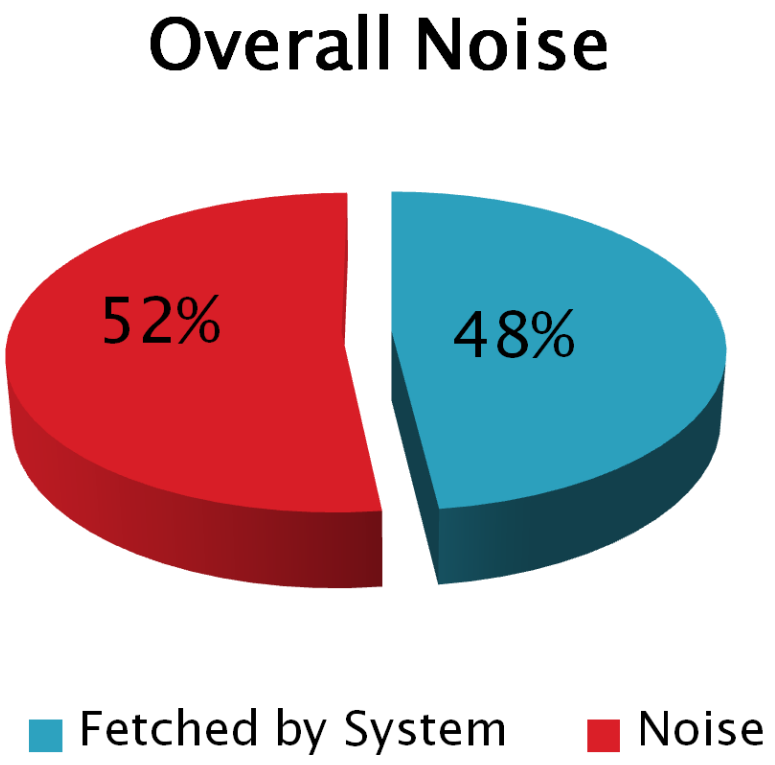

Figure 12: Pie chart suggesting overall noise removal

Experiment 2: The experiment was conducted on three sets of people .Each set consisted of 10 users.

The users were asked to select the post that they might be interested to read and write them down. Then the user was asked to login to my system and then I compared the sheet with the results written down by the user with the results shown by my system. To understand the experiment better let's take 1 test case. The user has signed in with his LinkedIn account on my system. He has 38 posts on his account. He is asked to select posts that might interest him from his LinkedIn account. The user ends up selected 25 posts that he might read from his account. My system returns 23 posts to the user when he logs in to my system. Out of these 23 posts, 19 posts overlap with the list of 25 posts selected by the user manually. So the efficiency here is 76% that is 19 out of 25 posts that user was interested in reading were returned by my system. The following pie chart explains this test case.

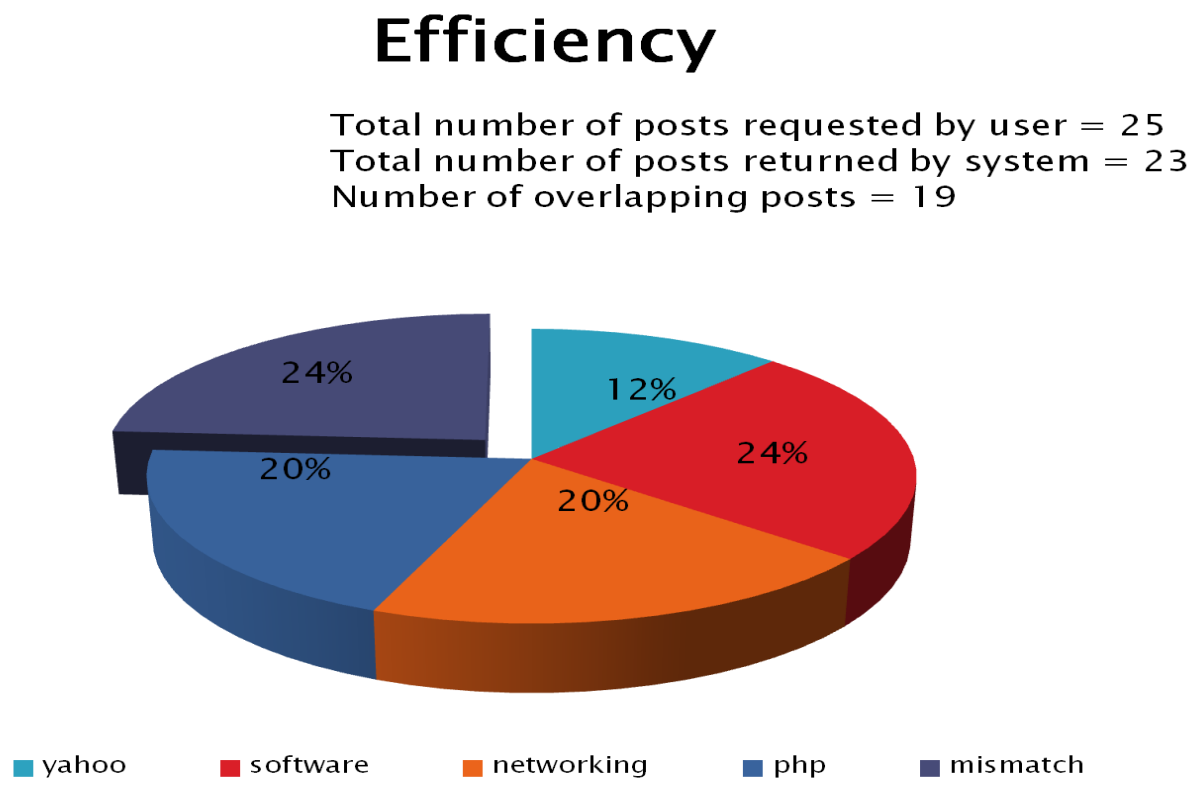

Figure 13: Efficiency for a particular test case

Overall results for 30 users:

For set 1 the Average result was 70% –that is for every 20 posts that the user selected, 14 matched with the results shown by my system.

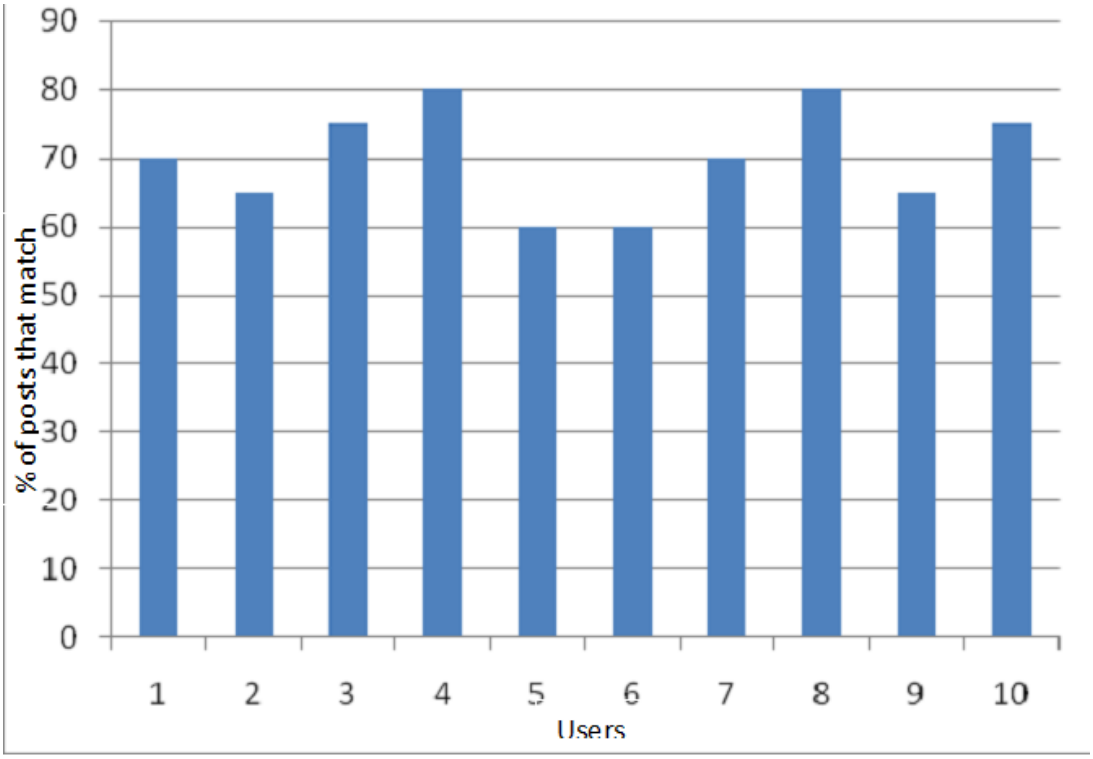

Figure 12: Set 1: Avg=70%

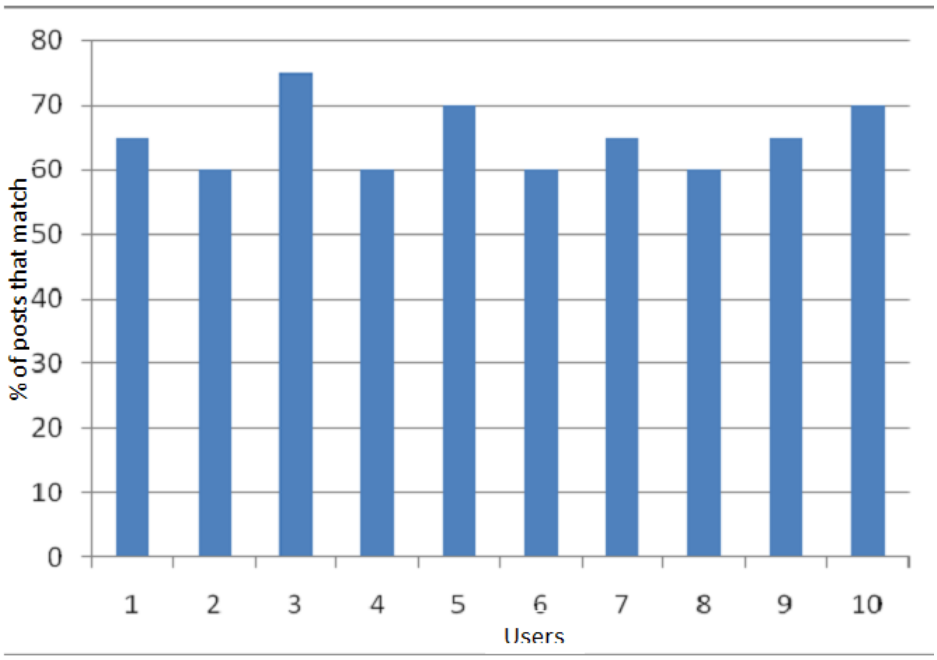

Figure 13: Set 2:Avg=65%

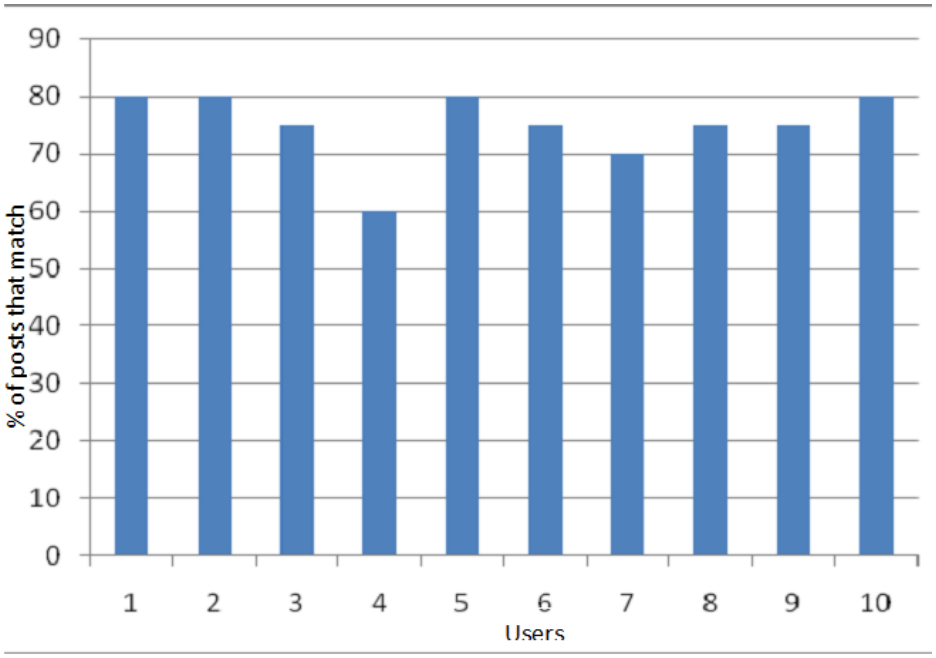

Figure 14:Set 3: Avg=75%

The average of all the three sets gives us 70%.Hence; the experiment suggests that 70% of the user's likes were detected correctly by the system.

8. CONCLUSION

This project “Social Network Leverage search” makes better use of the search done on the social networks. It solves the problem of information loading which infects the three most famous social networks: Facebook, Twitter and LinkedIn. It finds the user’s likes, analyzing the posts that are posted by the user on the three social networks, rather than explicitly asking the user to specify them. Then it removes the noise from Facebook, Twitter and LinkedIn to provide the users with only the posts that are of his interest. To evaluate the usability of the project an experiment was conducted on 30 users and the results suggest that the system provides the users with 70% posts of their likes. Thus, it resulted with the removal of noise and kept the user updated with the information of his interest.

9. FUTURE WORK

Any new idea evolved also has a scope of improvement. My system tracks the likes of the user by analyzing the posts that are posted by the user and the comments on the user's posts. This system matched the likes of users to 70%. The TF-IDF algorithm is used by this system to analyze the posts and form the tags. However, we can improve the algorithm and the results by also tracking the behavior of the user. We can keep track of all the places where the user comments, what posts he reads, where he blogs, etc. This analysis can give accurate resulting tags. Thus, tracking the behavior of the user to find out his/her likes will help improve the system to some extent.

10. APPENDIX A: SOURCE CODE

APP

CONTROLLERS:

Application_controller.rb

```
class ApplicationController < ActionController::Base
  protect_from_forgery

  helper_method :current_user, :signed_in?

  protected
  def redirect_if_not_user
    unless signed_in?
      flash[:error] = "You need to be signed in to access this action."
      redirect_to root_url
    end
  end

  def current_user
    @current_user ||= User.find_by_id(session[:user_id])
  end

  def signed_in?
    !!current_user
  end

  def sign_out
    if signed_in?
      @current_user = nil
      session[:user_id] = nil
    end
  end

  def current_user=(user)
    @current_user = user
    session[:user_id] = user.id
  end
end
```

authorizations_controller.rb

```
class AuthorizationsController < ApplicationController
  before_filter :redirect_if_not_user
```

```
  # Delete user session, sign out
  def destroy
    # do not allow to delete last authorization
    if current_user.authorizations.count > 1
      Authorization.where('id = ? AND user_id = ?', params[:id],
        current_user.id).first.destroy
      flash[:notice] = "Authorization successfuly removed."
    else
      flash[:error] = "Cannot delete last authorization."
    end

    redirect_to edit_user_path(current_user)
  end
end
```

cron_controller.rb

```
class CronController < ApplicationController
  # load all users posts and rebuild tags
  def run
    User.all.each { |user|
      user.load_twitter_posts
      user.load_facebook_posts
      user.load_linked_in_posts
      user.rebuild_tags

      # delete CachedRequests so they will be
      # reloaded on next user request
      CachedRequest.delete_all
    }
    render :text => "finished"
  end
end
```

pages_controller.rb

```
class PagesController < ApplicationController
  before_filter :reload_cache

  # Home page
  def home
    if signed_in? && params[:id]
      # find user tag
      @tag = Tag.where("id = ? AND user_id =? ", params[:id], current_user.id).first.try(:word)
    end
```

```
  if signed_in?
    if @tag
      # list all posts tagged by @tag if user selected tag

      @collection = PostsCollection.new(current_user.cached_requests)
      @title = "Listing posts tagged by \"#{@tag}\""
      @collection.parse(@tag)

      if params[:sort] == 'relevance'
        # sort by relevance
        @collection = @collection.sort_by_word_count(@tag)
        return
      else
        # sort by date
        @collection = @collection.sort_by_date
      end

      # slice posts array to only include set number of posts
      @collection = @collection.slice(0,APP_CONFIG['app']['tags_per_page'])
    else
      # list all posts
      @title = "Listing all posts"
      @posts_by_tag = Array.new

      # for each user tag get 3 posts and sort according to
      # user preference
      current_user.tags.all.each { |tag|
        @collection = PostsCollection.new(current_user.cached_requests)
        @collection.parse(tag.word)
        @sorted = params[:sort] == 'relevance' ? @collection.sort_by_word_count(tag.word):
@collection.sort_by_date
        @posts_by_tag << [[tag.word, @sorted.slice(0,3)]]
      }

    end
  else
    # user is not signed in, so show welcome message
    @title = "Welcome, please sign in"
    @collection = Array.new
  end
end

# reload user CachedRequests if needed on each request
def reload_cache
  if signed_in?
    current_user.reload_expired_cached_requests
  end
```

```
  end
end
```

sessions_controller.rb

```
class SessionsController < ApplicationController
  # sign in/up user
  def create
    auth = request.env['omniauth.auth']

    unless @auth = Authorization.find_from_hash(auth)
      # Create a new user or add an auth to existing user, depending on
      # whether there is already a user signed in.
      @auth = Authorization.create_from_hash(auth, current_user)
    end

    # update access token in database
    @auth.update_access_token(auth)

    # Log the authorizing user in.
    self.current_user = @auth.user

    redirect_to root_url
  end

  # sign out user
  def destroy
    sign_out

    redirect_to root_url
  end
end
```

users_controller.rb

```
class UsersController < ApplicationController
  before_filter :redirect_if_not_user

  # impersonate to user account, only in development!
  # delete once app is deployed to live server!
  def impersonate
    if params[:id]
      session[:user_id] = User.find(params[:id]).id
      redirect_to root_url
    end
  end
```

```
  # edit user page
  def edit
    @user = current_user
  end

  # update user attributes
  def update
    @user = current_user

    if @user.update_attributes(params[:user])
      redirect_to edit_user_path(@user), :notice => "Your profile successfuly updated."
    else
      render :action => 'edit'
    end
  end
end
```

HELPERS

Application_helper.rb

```
module ApplicationHelper
  # signed in user links
  def user_links
    menu = Array.new

    menu << { :title => 'Profile', :class => "", :path => edit_user_path(current_user) } if
signed_in?
    menu << { :title => 'Logout', :class => 'logout', :path => logout_path } if signed_in?

    # set current page
    menu.each do |link|
      link[:class] += " active" if current_page? link[:path]
    end
  end

  # Returns link to specified serfice html <a> tag
  def link_to_service(service)
    case service
      when "facebook"
        link_to image_tag("facebook.png") + " Facebook", "/auth/facebook"
      when "twitter"
        link_to image_tag("twitter.png") + " Twitter", "/auth/twitter"
      when "linked_in"
        link_to image_tag("linkedin.png") + " LinkedIn", "/auth/linked_in"
    end
  end
```

```
  # Returns image to specified service html tag
  def provider_image_tag_small(provider)
    case provider
      when "facebook"
        image_tag("facebook_small.png")
      when "twitter"
        image_tag("twitter_small.png")
      when "linked_in"
        image_tag("linkedin_small.png")
    end
  end

  # Returns image to specified service html tag
  def provider_image_tag(provider)
    case provider
      when "facebook"
        image_tag("facebook.png")
      when "twitter"
        image_tag("twitter.png")
      when "linked_in"
        image_tag("linkedin.png")
    end
  end
end
```

authorizations_helper.rb

```
module AuthorizationsHelper
  def destroy

  end
end
```

cron_helper.rb

```
module CronHelper
end
```

pages_helper.rb

```
module PagesHelper

end
```

sessions_helper.rb

```
module SessionsHelper
end
```

users_helper.rb

```
module UsersHelper
end
```

MODELS

Authorization.rb

```
class Authorization < ActiveRecord::Base
  belongs_to :user

  validates_presence_of :user_id, :uid, :provider
  validates_uniqueness_of :uid, :scope => :provider

  after_create :get_own_posts, :get_friends_posts, :rebuild_user_tags
  before_destroy :delete_cached_requests

  # destroys all cached requests for given user
  def delete_cached_requests
    CachedRequest.where('user_id = ? and provider = ?', self.user.id, self.provider).first.destroy
  end

  # loads user own posts
  def get_own_posts
    case self.provider
      when 'twitter'
        self.user.load_twitter_posts
      when 'linked_in'
        self.user.load_linked_in_posts
      when 'facebook'
        self.user.load_facebook_posts
    end
  end

  # loads friends posts
  def get_friends_posts
    CachedRequest.update_provider_cache(self.user, self.provider)
  end

  # rebuilds/builds user tags
  def rebuild_user_tags
    self.user.rebuild_tags
  end
```

```
  # search for authorization by hash parameters
  def self.find_from_hash(hash)
    find_by_provider_and_uid(hash['provider'], hash['uid'])
  end

  # updates access token from hash parameters
  def update_access_token(hash)
    self.update_attributes({
      :token => hash['credentials']['token'],
      :secret => hash['credentials']['secret']
    })
  end

  # creates instance from hash parameters
  def self.create_from_hash(hash, user = nil)
    user ||= User.create_from_hash!(hash)
    Authorization.create(
      :user => user, :uid => hash['uid'],
      :provider => hash['provider'],
      :token => hash['credentials']['token'],
      :secret => hash['credentials']['secret'])
  end
end
```

cached_request.rb

```
require 'twitter_post'
require 'linked_in_post'

class CachedRequest < ActiveRecord::Base
  belongs_to :user
  belongs_to :authorization

  serialize :result, Array

  validates_uniqueness_of :provider, :scope => :user_id

  # time to keep CachedRequest
  EXPIRE_IN = 2.hours

  # update all CachedRequests for user connected providers
  def self.update_cache(user)
    user.connected_providers.each do |provider|
      self.update_provider_cache(user, provider)
    end
```

```
  end

  # update CachedRequest for given user and provider
  def self.update_provider_cache(user, provider)
    # remove old CachedRequest
    destroy_all("user_id = '#{user.id}' AND provider = '#{provider}'")
    user.reload

    case provider
      when "twitter"
        auth = user.twitter_authorization
        consumer = Consumers::Twitter.new(auth['token'], auth['secret'])
        create(:provider => provider, :user_id => user.id,
          :result => consumer.get_friends_posts,
          :authorization_id => auth.id
        )
      when "linked_in"
        auth = user.linked_in_authorization
        consumer = Consumers::LinkedIn.new(auth['token'], auth['secret'])
        create(:provider => provider, :user_id => user.id,
          :result => consumer.get_friends_posts,
          :authorization_id => auth.id
        )
      when "facebook"
        auth = user.facebook_authorization
        consumer = Consumers::Facebook.new(auth['token'], auth['secret'])
        create(:provider => provider, :user_id => user.id,
          :result => consumer.get_friends_posts,
          :authorization_id => auth.id
        )
    end
  end

  # Is specified user provider CachedRequest expired?
  def self.not_expired?(provider, user_id)
    where("created_at >= ? AND user_id = ? AND provider = ?", EXPIRE_IN.ago, user_id,
      provider).first
  end

  # Delete all expired CachedRequests
  def self.delete_expired
    destroy_all("created_at <= '#{EXPIRE_IN.ago}'")
  end
end
```

corpur_term.rb

```
class CorpusTerm < ActiveRecord::Base
end
```

facebook_post.rb

```
class FacebookPost < Post
  attr_reader :message, :description, :link, :caption
  def initialize (data, provider=nil)
    @provider = provider
    parse(data)
    self
  end

  private
  def parse(data)
    @text = parse_text(data)
    @message = data['message']
    @description = data['description']
    @link = data['link']
    @caption = data['caption']
    @name = data['from']['name']
    @all_text = [@message, @description, @link, @caption, @name].join(' ')

    @date = DateTime.parse(data['created_time'])
    @since = DateTime.parse(data['created_time']).to_time.to_i
    @uid = data['from']['id']
  end

  def parse_text(data)
    text = data['description'].to_s
    text += " " + data['caption'].to_s
    text += " " + data['message'].to_s
    text += " " + data['name'].to_s
  end
end
```

linked_in_post.rb

```
class LinkedInPost < Post
  attr_reader :headline, :comment, :link

  def initialize (data, provider=nil)
    @provider = provider
    parse(data)
    self
  end
```

```
  private
  def parse(data)
    if data['timestamp']
      @date = DateTime.parse(Time.at(data['timestamp'].to_i / 1000).to_time.to_s)
      @since = data['timestamp']
      @type = data['updateType']

      parse_by_type(data)
    end
  end

  def parse_by_type(data)
    case data['updateType']
      when "CONN"
        @text = data['updateContent']['person']['connections']['values'].collect { |c|
          c['headline'] }.join(" ")
        @uid = data['updateContent']['person']['id']
        @name = @name.to_s + data['updateContent']['person']['firstName'] + " " +
data['updateContent']['person']['lastName']
        @all_text = @text

      when "STAT"
        @text = data['updateContent']['person']['currentStatus']
        @headline = data['updateContent']['person']['headline']
        @all_text = [@text, @headline].join(' ')
        @uid = data['updateContent']['person']['id']
        @name = @name.to_s + data['updateContent']['person']['firstName'] + " " +
data['updateContent']['person']['lastName']

      when "SHAR"
        @text = data['updateContent']['person']['currentShare']['content']['title']
        @comment = data['updateContent']['person']['currentShare']['comment']
        @link = data['updateContent']['person']['currentShare']['content']['submittedUrl']
        @headline = data['updateContent']['person']['headline']
        @all_text = [@text, @headline, @link, @comment].join(' ')
        @uid = data['updateContent']['person']['id']
        @name = @name.to_s + data['updateContent']['person']['firstName'] + " " +
data['updateContent']['person']['lastName']
    end
  end
end
```

post.rb

```
# Base Post class. All services post classes will extend this class.
# Implements default attributes, accessors and methods.
```

```
class Post
  attr_reader :text, :date, :name, :provider, :since, :uid, :all_text

  def valid?
    true
  end

  def own?(user_id)
    user_id.to_s == @uid.to_s
  end
end
```

tag.rb

```
class Tag < ActiveRecord::Base
  belongs_to :user

  scope :sorted, order("tf_idf DESC")

  # tags to display in list
  DISPLAY = 30

  # find Tag from hash parameters
  def self.find_from_hash(hash)
    find_by_word_and_user_id(hash[:word], hash[:user_id])
  end

  # Create/update Tag from hash parameters
  def self.add_from_hash(hash)
    if tag = find_from_hash(hash)
      if tag.tf_idf != hash[:tf_idf]
        tag.update_attributes(:tf_idf => hash[:tf_idf])
      end
    else
      create(hash)
    end
  end
end
```

twitter_post.rb

```
class TwitterPost < Post
  def initialize (data, provider=nil)
    @provider = provider
    parse(data)
    self
  end
```

```
  private
  def parse(data)
    @text = data['text']
    @all_text = @text
    @date = DateTime.parse(data['created_at'])
    @name = data['user']['name']
    @since = data['id']
    @uid = data['user']['id']
  end
end
```

user.rb

```
class User < ActiveRecord::Base
  has_many :authorizations
  has_many :user_posts
  has_many :cached_requests
  has_many :tags
  serialize :options

  validates_presence_of :name

  # list of providers
  PROVIDERS = %w{facebook twitter linked_in}

  # minimum Tf-Idf threshhold for tags
  TF_IDF_THRESHOLD = 2.0

  def rebuild_tags
    # collect all user posts to single document
    text = self.user_posts.collect { |p| p.text }.join("\n")

    # analyze document for tags
    tags = Analyzer.analyze(text)

    # add tags to database if TF_IDF value is high enough
    tags.each { |tag|
      if tag[1][:tf_idf] > TF_IDF_THRESHOLD
        logger.info("Adding \"#{tag[0]}\" tag to \"#{self.name}\"")

        hash = {:word => tag[0], :tf_idf => tag[1][:tf_idf], :user_id => self.id}
        Tag.add_from_hash(hash)
      end
    }
  end
```

```
# Reloads all expired cached requests
def reload_cached_requests
  CachedRequest.update_cache(self)
end

# Reloads expired cached requests
def reload_expired_cached_requests
  self.connected_providers.each { |provider|
    unless CachedRequest.not_expired?(provider, self.id)
      CachedRequest.update_provider_cache(self, provider)
    end
  }
end

# Fetch and save to database user Facebook posts
def load_facebook_posts
  if auth = self.facebook_authorization
    consumer = Consumers::Facebook.new(auth['token'], auth['secret'])

    since = self.since_param(auth.provider)

    consumer.get_posts(since).each { |post|
      logger.info("Creating Facebook UserPost \"#{post.text}\" for \"#{self.name}\"")

      UserPost.create(
        :text => post.all_text,
        :since => post.since,
        :user_id => self.id,
        :provider => auth.provider
      )
    }
  end
end

# Fetch and save to database user LinkedIn posts
def load_linked_in_posts
  if auth = self.linked_in_authorization
    consumer = Consumers::LinkedIn.new(auth['token'], auth['secret'])
    since = self.since_param(auth.provider)

    consumer.get_network_updates(since).each { |post|
      logger.info("Creating LinkedIn UserPost \"#{post.all_text}\" for \"#{self.name}\"")

      UserPost.create(
        :text => post.all_text,
        :since => post.since,
        :user_id => self.id,
```

```
      :provider => auth.provider
    )
   }
  end
end

# Fetch and save to database user Twitter posts
def load_twitter_posts
  if auth = self.twitter_authorization
    consumer = Consumers::Twitter.new(auth['token'], auth['secret'])
    since = self.since_param(auth.provider)

    consumer.get_posts(since).each { |post|
      logger.info("Creating Twitter UserPost \"#{post.all_text}\" for \"#{self.name}\"")

      UserPost.create(
        :text => post.all_text,
        :since => post.since,
        :user_id => self.id,
        :provider => auth.provider
      )
    }
  end
end

# Get latest 'since' paramater for user authentication posts
# Since parameter is used to fetch only newer posts, skipping
# all existing in database posts
def since_param(provider)
  self.user_posts.where(:provider => provider).order("since DESC").first.try(:since)
end

# Reload cached requests for all connected providers
def reload_cached_requests
  CachedRequest.delete_expired

  if self.connected_providers.count != self.cached_requests.count
    CachedRequest.update_cache(self)
  end
end

# Create User from hash parameters
def self.create_from_hash!(hash)
  create(:name => hash['user_info']['name'])
end

# Is all providers connected to specified user?
```

```
  def all_providers_connected?
    self.authorizations.count == 3
  end

  # Returns array of connected providers
  def connected_providers
    self.authorizations.collect { |auth| auth.provider }
  end

  # Returns array of providers still not connected
  def pending_providers
    @pending_providers = User::PROVIDERS.dup

    self.authorizations.each { |auth|
      @pending_providers.delete(auth.provider)
    }

    @pending_providers
  end

  # Returns Twitter Authorization model for specified user
  def twitter_authorization
    self.authorizations.where('provider = ?', 'twitter').first
  end

  # Returns Facebook Authorization model for specified user
  def facebook_authorization
    self.authorizations.where('provider = ?', 'facebook').first
  end

  # Returns LinkedIn Authorization model for specified user
  def linked_in_authorization
    self.authorizations.where('provider = ?', 'linked_in').first
  end
end
```

user_post.rb

```
class UserPost < ActiveRecord::Base
  belongs_to :user

  validates_uniqueness_of :since, :scope => [:user_id, :provider]
end
```

VIEWS

Application.html.erb

```
<!DOCTYPE html>
<html>
<head>
  <title>Social Network Search App</title>
  <%= stylesheet_link_tag "web-app-theme/base", "web-app-theme/themes/default/style", "web-app-theme/override", "style" %>
  <%= javascript_include_tag 'jquery.min.js', 'rails.js', :defaults %>
  <%= csrf_meta_tag %>
</head>
<body>
  <div id="container">
    <div id="header">
      <h1><a href="/">Social Network Search App</a></h1>
      <div id="user-navigation">
        <ul class="wat-cf">
          <% user_links.each do |link| %>
            <li><%= link_to link[:title], link[:path], :class => link[:class] %></li>
          <% end %>
        </ul>
      </div>
      <div id="main-navigation">
        <ul class="wat-cf">
          <li class="first active last"><%= link_to "Home page", root_url%></li>
        </ul>
      </div>
    </div>
    <div id="wrapper" class="wat-cf">
      <div class="flash">
        <% flash.each do |type, message| -%>
          <div class="message <%= type %>">
            <p><%= message %></p>
          </div>
        <% end -%>
      </div>
      <div id="main">
        <%= yield %>
        <div id="footer">
          <div class="block">
            <p>Copyright &copy; <%= Time.now.year %> Social Network Search App.</p>
          </div>
        </div>
      </div>
```

```
      <div id="sidebar">
        <%= render :partial => 'partials/connect' %>
        <%= yield :sidebar %>
      </div>
    </div>
  </div>
</body>
</html>
```

Home.html.erb

```
<div class="block">
  <div class="content">
    <h2 class="title"><%= @title %></h2>
    <div class="inner">
      <% if signed_in? %>
        <ul class="list">
          <p>Sort by <%= link_to "date", '?' %> | <%= link_to "relevance", '?sort=relevance' %>
          <% if @tag %>
            <%= render :partial => 'partials/post', :collection => @collection %>
          <% else %>
            <%= render :partial => 'partials/post_list', :collection => @posts_by_tag %>
          <% end %>
        </ul>
      <% end %>
    </div>
  </div>
</div>
<% content_for :sidebar do %>
  <% if signed_in? %>
    <div class="block">
      <h3>Tags</h3>
      <p class="tags">
        <% current_user.tags.sorted.slice(0,Tag::DISPLAY).each do |tag| %>
          <%= link_to tag.word, "/tags/#{tag.id}", :class => 'tag' %>
        <% end %>
      </p>
    </div>
  <% end %>
<% end %>
```

_connect.html.erb

```
<% unless signed_in? %>
  <div class="block notice">
    <h3>Sign in with</h3>
    <ul class="navigation social_links">
```

```
    <% User::PROVIDERS.each do |provider| %>
      <li><%= link_to_service(provider) %></li>
    <% end %>
  </ul>
 </div>
<% else %>
 <% if current_user.all_providers_connected? %>
  <p id="connected_to">
   Your profile is connected to:
   <% current_user.connected_providers.each do |provider| %>
    <%= provider_image_tag_small(provider) %>
   <% end %>
  </p>
 <% else %>
  <div class="block notice">
   <h3>Improve your experience by connecting your profile to more social networks:</h3>
   <ul class="navigation social_links">
    <% current_user.pending_providers.each do |provider| %>
     <li><%= link_to_service(provider) %></li>
    <% end %>
   </ul>
  </div>
 <% end %>
<% end %>
```

_post.html.erb

```
<li class="">
 <div class="left"><%= provider_image_tag(post.provider) %></div>

 <% case post.provider %>
  <% when 'twitter' %>
   <div class="item">
    <p><%= post.text %></p>
    <p><strong>By <%= post.name %></strong> at <%= l post.date.to_time, :format => :long
%></p>
   </div>

  <% when 'facebook' %>
   <div class="item">
    <% if post.caption %>
     <p><b><%= post.caption %></b></p>
    <% end %>

    <% if post.message %>
     <p><%= post.message %></p>
    <% end %>
```

```
      <% if post.description %>
        <p><span class="gray"><%= post.description %></span></p>
      <% end %>

      <% if post.link %>
        <p><a href="<%= post.link %>" class="link"><%= image_tag 'web-app-theme/icons/link.png'%> <%= post.link %></a></p>
      <% end %>

      <p><strong>By <%= post.name %></strong> at <%= l post.date.to_time, :format => :long %></p>
    </div>

  <% when 'linked_in' %>
    <div class="item">
      <% if post.headline %>
        <p><strong><%= post.headline %></strong></p>
      <% end %>

      <% if post.text %>
        <p><%= post.text %></p>
      <% end %>

      <% if post.comment %>
        <p><span class="gray"><%= post.comment %></span></p>
      <% end %>

      <% if post.link %>
        <p><a href="<%= post.link %>" class="link"><%= image_tag 'web-app-theme/icons/link.png'%> <%= post.link %></a></p>
      <% end %>

      <p><strong>By <%= post.name %></strong> at <%= l post.date.to_time, :format => :long %></p>
    </div>
  <% end %>
</li>
```

_post_list.html.erb

```
<% post_list.each do |list| %>
  <% unless list[1].empty? %>
    <h3>Posts tagged by "<%= list[0] %>"</h3>
    <%= render :partial => 'partials/post', :collection => list[1] %>
  <% end %>
<% end %>
```

Edit.html.erb

```
<% post_list.each do |list| %>
  <% unless list[1].empty? %>
    <h3>Posts tagged by "<%= list[0] %>"</h3>
    <%= render :partial => 'partials/post', :collection => list[1] %>
  <% end %>
<% end %>
```

LIB

Analyzer.rb

```
class Analyzer
  # Constructor method.
  def initialize(user)
    @user = user
  end

  # add text to document
  def add_text(text)
    @user.document += "\n#{text}"
    @user.save
  end

  # Returns sorted tags list
  def self.analyze(text)
    words = Hash.new

    # split text into words and calculate
    # each word frequency in the text - TF
    text.split(/[^A-Za-z]/).each { |word|
      word = word.downcase
      if words[word]
        words[word][:tf] += 1
      elsif word.length > 2
        words[word] = {:tf => 1}
      end
    }

    # calculate IDF value for each word
    # in the text
    words.each { |word, value|
      if corpus_term = CorpusTerm.find_by_term(word)
        words[word][:idf] = Math.log10(504 / corpus_term.count)
      else
```

```
      words[word][:idf] = Math.log10(504 / 1)
    end

    # calculate TF-IDF value
    words[word][:tf_idf] = words[word][:tf] * words[word][:idf]
  }

  # sort words by TF-IDF value
  words.sort { |a,b| b[1][:tf_idf] <=> a[1][:tf_idf] }
 end
end
```

consumers.rb

```
require 'json'
require 'consumers/base'
require 'consumers/twitter'
require 'consumers/facebook'
require 'consumers/linked_in'
```

corpus.rb

```
require 'ar-extensions'

class Corpus
 # Constructor method
 def initialize
  @collection = Hash.new
 end

 # add new document for analization
 def add_document(doc)
  words = split_into_words(doc)

  temp = Array.new

  words.each { |word|
   if word and word.length > 2
    temp << normalize_word(word)
   end
  }

  temp.uniq.each{ |word|
   add_to_collection(word)
  }
 end
```

```
  # do mass insert of analyzed terms to DB
  def insert
    fields = [:term, :count]
    data = []
    @collection.each { |term,count|
      data << [term, count]
    }

    puts "Starting to import #{data.length} terms..."

    CorpusTerm.import fields, data

    puts "Import finished!"
  end

  private
  # add word to main collection
  def add_to_collection(word)
    if @collection[word].nil?
      @collection[word] = 1
    else
      @collection[word] += 1
    end
  end

  # split text into words
  def split_into_words(text)
    text.split(/[^a-zA-Z\/]/).collect { |w| w.split('/').try(:first) }
  end

  # normalize word
  def normalize_word(word)
    word.downcase
  end
end
```

posts_collection.rb

```
class PostsCollection
  attr_reader :collection

  # Constructor method
  def initialize(cached_requests)
    @cached_requests = cached_requests
    @collection = Array.new
    self
  end
```

```
# parse all posts
def parse(tag = nil)
  if tag.nil?
    collect_all_posts
  else
    collect_posts_by_tag(tag)
  end
end

# sort posts by word count
def sort_by_word_count(tag)
  @collection.sort { |a,b|
    word_count(b.all_text, tag) <=> word_count(a.all_text, tag)
  }
end

# sort posts by date
def sort_by_date
  @collection.sort { |a,b| b.date <=> a.date }
end

private
# collect all posts
def collect_all_posts
  @cached_requests.each do |request|
    request.result.each { |post|
      # post must not be own and invalid
      if !post.own?(request.authorization.uid) && post.valid?
        @collection << post
      end
    }
  end
end

# collect all posts including 'tag' word
def collect_posts_by_tag(tag)
  @cached_requests.each do |request|
    request.result.each { |post|
      # post must not be own and invalid
      if !post.own?(request.authorization.uid) && post.valid? && word_search(post.all_text, tag)
        @collection << post
      end
    }
  end
end
```

```
  private
  # Returns word frequency in document
  def word_count(document, word)
    document.to_s.scan(/#{word}/i).count
  end

  # Search for word in document
  def word_search(document, word)
    document =~ /#{word}/i
  end
end
```

CONSUMERS

Base.rb

```
# Consumers Base class, all services consumers will extend this
# class
module Consumers
  class Base
    # Constructor method, it takes user auth tokens
    # and initializes connection to API
    def initialize(oauth_token, oauth_token_secret)
      @consumer = OAuth::Consumer.new(
        APP_CONFIG[provider_name]['key'],
        APP_CONFIG[provider_name]['secret'],
        { :site => site_name }
      )

      @token_hash = {
        :oauth_token => oauth_token,
        :oauth_token_secret => oauth_token_secret
      }

      @access_token = OAuth::AccessToken.from_hash(@consumer, @token_hash )
    end

    # provider name
    def provider_name
      nil
    end

    # provider site url
    def site_name
      nil
    end
  end
```

```
end
```

facebook.rb

```
require 'open-uri'

module Consumers
  class Facebook < Base
    def provider_name
      'facebook'
    end

    def site_name
      'http://www.facebook.com'
    end

    # load friends posts
    def get_friends_posts
      token = CGI.escape(@access_token.token)

      # posts per request
      limit = 100
      posts = Array.new

      # request url
      url = "https://graph.facebook.com/me/home?limit=#{limit}&access_token=" + token

      begin
        # initialize request
        result = JSON.parse(
          open(url).read
        )
        posts = result['data']
      rescue
        Rails.logger.fatal("Facebook fetching failed")
      end

      # create FacebookPost object for each post in result
      posts.collect { |post|
        FacebookPost.new(post, provider_name)
      }
    end

    # load own posts
    def get_posts(since = nil)
      token = CGI.escape(@access_token.token)
```

```
    # posts per request
    limit = 100

    # build request url
    url = "https://graph.facebook.com/me/posts?limit=#{limit}&access_token=" + token
    url += "&since=#{since}" if since

    puts "Getting #{url}"

    posts = Array.new

    begin
      # initialize request
      result = JSON.parse(
        open(url).read
      )

      posts = result['data']

      # build request url for next page
      next_url = result['paging'].nil? ? nil : result['paging']['next']

      # fetch next page while there is one
      while next_url
        puts "Getting #{next_url}"
        #parse request result JSON
        result = JSON.parse(open(next_url).read)

        next_url = result['paging'].nil? ? nil : result['paging']['next']

        # append posts from current page to main array
        posts += result['data']
      end
    rescue
      Rails.logger.fatal("Facebook fetching failed")
    end

    # create FacebookPost object for each post in result
    posts.collect { |post|
      FacebookPost.new(post, provider_name)
    }
  end
 end
end
```

linked_in.rb

```
module Consumers
  class LinkedIn < Base
    def provider_name
      'linked_in'
    end

    def site_name
      'http://www.linkedin.com'
    end

    # load user friends posts
    def get_friends_posts
      count = 50
      posts = Array.new

      begin
        # build request url
        url = "http://api.linkedin.com/v1/people/~/network/updates?"
        url += "count=#{count}"

        # initialize request
        response = @access_token.request(:get, url,'x-li-format'=>'json' )

        # parse request result JSON
        posts = JSON.parse(response.body)['values'] || []
      rescue
        Rails.logger.fatal("LinkedIn fetching failed")
      end

      # create LinkedInPost object for each post in result
      posts.collect { |post|
        LinkedInPost.new(post, provider_name)
      }
    end

    # load user own posts
    def get_network_updates(since = nil)
      count = 50
      page = 0
      posts = Array.new

      # build request url
      url = "http://api.linkedin.com/v1/people/~/network/updates?scope=self"
      url += "&count=#{count}"
      url += "&after=#{since}" if since

      # fetch next page while there is one
```

```
      begin
        while true
          get_url = url + "&start=#{page*count}"
          puts "Fetching #{get_url}"

          # initialize request
          response = @access_token.request(:get, get_url,'x-li-format'=>'json' )
          new_posts = JSON.parse(response.body)['values'] || []

          # create LinkedInPost object for each post in result
          new_posts.each { |post|
            # assign current page posts to main array
            posts << LinkedInPost.new(post, provider_name)
          }

          page += 1
          if new_posts.count != count then
            break
          end
        end
      rescue
        Rails.logger.fatal("LinkedIn fetching failed")
      end

      posts
    end
  end
end
```

twitter.rb

```
require 'twitter_post'
module Consumers
  class Twitter < Base
    def provider_name
      'twitter'
    end

    def site_name
      'http://api.twitter.com'
    end

    def get_friends_posts
      # results per page
      count = 200
      posts = Array.new
      page  = 1
```

```
  # building API request url
  url = "http://api.twitter.com/1/statuses/home_timeline.json"
  url += "?count=#{count}"

  begin
    # initialize request
    response = @access_token.request(:get, url)

    # parse JSON response to Hash
    posts = JSON.parse(response.body)
  rescue
    Rails.logger.fatal("Twitter fetching failed")
  end

  # create TwitterPost object for each post in result
  posts.collect {|post|
    TwitterPost.new(post, provider_name)
  }

end

def get_posts(since = nil)
  # results per page
  count = 200

  page  = 1
  posts = Array.new

  # building API request url
  url = "http://api.twitter.com/1/statuses/user_timeline.json"
  url += "?count=#{count}"
  url += "&since_id=#{since}" if since

  begin
    # fetch all pages
    while true
      get_url = url + "&page=#{page}"
      puts "Getting #{url}"

      # do fetch
      response = @access_token.request(:get, get_url)

      # parse JSON response to Hash
      new_posts = JSON.parse(response.body)

      posts += new_posts
```

```
        page += 1

        # stop fetching if we got less tweets than
        # specified in our count parameter
        if new_posts.count != count then
          break
        end
      end
    rescue
      Rails.logger.fatal("Twitter fetching failed")
    end

    # create TwitterPost object for each post in result
    posts.collect {|post|
      TwitterPost.new(post, provider_name)
    }
  end
 end
end
```

TASKS

Application.rake

```
namespace :app do
 # worker will load new user posts, and
 # rebuild tags, probably should be run
 # once every 24 hours
 desc "Run worker"
 task :run_worker => :environment do
  Rake::Task["app:get_all_posts"].invoke
  Rake::Task["app:tags:build"].invoke
 end

 desc "Get all users new posts"
 task :get_all_posts => :environment do
  Rake::Task["app:twitter:load_posts"].invoke
  Rake::Task["app:facebook:load_posts"].invoke
  Rake::Task["app:linked_in:load_posts"].invoke
 end

 desc "Delete all posts in db"
 task :delete_all_posts => :environment do
  UserPost.delete_all
 end

 desc "Delete posts and tags"
```

```
task :delete_all_data => :environment do
  UserPost.delete_all
  CachedRequest.delete_all
  Tag.delete_all
end

desc "Show user tags"
task :show_tags => :environment do
  User.all.each { |user|
    puts "Tags for #{user.name}"
    user.tags.sorted.each { |tag|
      puts "\t#{tag.word} => #{tag.tf_idf}"
    }
  }
end

namespace :tags do
  desc "Build tags from users posts"
  task :build => :environment do
    User.all.each { |user|
      user.rebuild_tags
    }
  end
end

namespace :twitter do
  desc "Fetch all system users Twitter posts"
  task :load_posts => :environment do
    User.all.each { |user|
      user.load_twitter_posts
    }
  end
end

namespace :facebook do
  desc "Fetch all system users Twitter posts"
  task :load_posts => :environment do
    User.all.each { |user|
      user.load_facebook_posts
    }
  end

  task :show_own_posts, :id, :needs  => :environment do |t, args|
    if id = args[:id]
      if user = User.find(id)
        puts "User \"#{user.name}\" Facebook posts:"
        user.user_posts.where("provider = 'facebook'").all.each { |post|
```

```
        puts post.text
       }
     end
   end
 end
end

namespace :linked_in do
 desc "Fetch all system users LinkedIn posts"
 task :load_posts => :environment do
  User.all.each { |user|
   user.load_linked_in_posts
  }
 end

 task :api_response_all, :id, :needs  => :environment do |t, args|
  if id = args[:id]
   if user = User.find(id)
    puts "User \"#{user.name}\" LinkedIn response:"
    if auth = user.linked_in_authorization
     @consumer = OAuth::Consumer.new(
      APP_CONFIG['linked_in']['key'],
      APP_CONFIG['linked_in']['secret'],
      { :site => 'http://www.linkedin.com' }
     )

     @token_hash = {
      :oauth_token => auth['token'],
      :oauth_token_secret => auth['secret']
     }

     @access_token = OAuth::AccessToken.from_hash(@consumer, @token_hash )

     url = "http://api.linkedin.com/v1/people/~/network/updates"

     response = @access_token.request(:get, url,'x-li-format'=>'json' )
     puts JSON.pretty_generate(JSON.parse(response.body))
    end
   end
  end
 end

 task :api_response_self, :id, :needs  => :environment do |t, args|
  if id = args[:id]
   if user = User.find(id)
    puts "User \"#{user.name}\" LinkedIn response:"
    if auth = user.linked_in_authorization
```

```
      @consumer = OAuth::Consumer.new(
        APP_CONFIG['linked_in']['key'],
        APP_CONFIG['linked_in']['secret'],
        { :site => 'http://www.linkedin.com' }
      )

      @token_hash = {
        :oauth_token => auth['token'],
        :oauth_token_secret => auth['secret']
      }

      @access_token = OAuth::AccessToken.from_hash(@consumer, @token_hash )

      url = "http://api.linkedin.com/v1/people/~/network/updates?scope=self"

      response = @access_token.request(:get, url,'x-li-format'=>'json' )
      puts JSON.pretty_generate(JSON.parse(response.body))
    end
  end
 end
end

task :api_response_all_posts, :id, :needs  => :environment do |t, args|
 if id = args[:id]
   if user = User.find(id)
     puts "User \"#{user.name}\" LinkedIn response:"
     if auth = user.linked_in_authorization
       @consumer = OAuth::Consumer.new(
         APP_CONFIG['linked_in']['key'],
         APP_CONFIG['linked_in']['secret'],
         { :site => 'http://www.linkedin.com' }
       )

       @token_hash = {
         :oauth_token => auth['token'],
         :oauth_token_secret => auth['secret']
       }

       @access_token = OAuth::AccessToken.from_hash(@consumer, @token_hash )

       url = "http://api.linkedin.com/v1/people/~/network/updates"

       response = @access_token.request(:get, url,'x-li-format'=>'json' )
       posts = JSON.parse(response.body)['values'].collect { |post|
         LinkedInPost.new(post, 'linked_in')
       }
       puts posts.inspect
```

```
        end
      end
    end
  end

  task :api_response_self_posts, :id, :needs  => :environment do |t, args|
    if id = args[:id]
      if user = User.find(id)
        puts "User \"#{user.name}\" LinkedIn response:"
        if auth = user.linked_in_authorization
          @consumer = OAuth::Consumer.new(
            APP_CONFIG['linked_in']['key'],
            APP_CONFIG['linked_in']['secret'],
            { :site => 'http://www.linkedin.com' }
          )

          @token_hash = {
            :oauth_token => auth['token'],
            :oauth_token_secret => auth['secret']
          }

          @access_token = OAuth::AccessToken.from_hash(@consumer, @token_hash )

          url = "http://api.linkedin.com/v1/people/~/network/updates?scope=self"

          response = @access_token.request(:get, url,'x-li-format'=>'json' )
          posts = JSON.parse(response.body)['values'].collect { |post|
            LinkedInPost.new(post, 'linked_in')
          }
          puts posts.inspect
        end
      end
    end
  end
end

namespace :test do
  desc "Test analyzer"
  task :analyzer => :environment do
    User.all.each { |user|
      puts "User #{user.name} tags"
      text = user.user_posts.collect { |p| p.text }.join("\n")
      analyzer = Analyzer.analyze(text)
    }
  end
end
end
```

corpus.rake

```
namespace :corpus do
  desc "Corpus database status"
  task :status => :environment do
    puts "Corpus currently has #{CorpusTerm.all.count} terms."
  end

  desc "Load corpus from .txt file"
  task :load => :environment do
    dir = Dir.new(Rails.root.join('corpus'))
    corpus = Corpus.new

    dir.each { |filename|
      path = "#{dir.path}/#{filename}"

      if File.file?(path)
        puts "Reading #{filename} ..."

        file = File.open(path,'r')
        text = file.read
        corpus.add_document(text)
      end
    }

    corpus.insert
  end
end
```

DB

MIGRATE

20110407175015_create_authorizations.rb

```
class CreateAuthorizations < ActiveRecord::Migration
  def self.up
    create_table :authorizations do |t|
      t.string :provider
      t.string :uid
      t.integer :user_id

      t.timestamps
    end
  end
```

```
  def self.down
    drop_table :authorizations
  end
end
```

20110407175046_create_users.rb

```
class CreateUsers < ActiveRecord::Migration
  def self.up
    create_table :users do |t|
      t.string :name

      t.timestamps
    end
  end

  def self.down
    drop_table :users
  end
end
```

20110413094708_add_token_and_secret_to_authorizations.rb

```
class AddTokenAndSecretToAuthorizations < ActiveRecord::Migration
  def self.up
    add_column :authorizations, :token, :string
    add_column :authorizations, :secret, :string
  end

  def self.down
    remove_column :authorizations, :token
    remove_column :authorizations, :secret
  end
end
```

20110414150930_create_corpus_terms.rb

```
class CreateCorpusTerms < ActiveRecord::Migration
  def self.up
    create_table :corpus_terms do |t|
      t.string :term
      t.integer :idf

      t.timestamps
    end
  end
```

```
  def self.down
    drop_table :corpus_terms
  end
end
```

20110414174200_rename_idf_to_count.rb

```
class RenameIdfToCount < ActiveRecord::Migration
  def self.up
    rename_column :corpus_terms, :idf, :count
  end

  def self.down
  end
end
```

20110414192340_add_index_to_corpus_term.rb

```
class AddIndexToCorpusTerm < ActiveRecord::Migration
  def self.up
    add_index :corpus_terms, :term
  end

  def self.down
    remove_index :corpus_terms, :term
  end
end
```

20110414194302_add_options_to_user.rb

```
class AddOptionsToUser < ActiveRecord::Migration
  def self.up
    add_column :users, :options, :text
  end

  def self.down
    remove_column :users, :options
  end
end
```

20110415064823_create_user_posts.rb

```
class CreateUserPosts < ActiveRecord::Migration
  def self.up
    create_table :user_posts do |t|
      t.text :text
```

```
      t.string :provider
      t.string :since

      t.timestamps
    end
  end

  def self.down
    drop_table :user_posts
  end
end
```

20110415071047_add_user_id_to_user_posts.rb

```
class AddUserIdToUserPosts < ActiveRecord::Migration
  def self.up
    add_column :user_posts, :user_id, :integer
  end

  def self.down
    remove_column :user_posts, :user_id
  end
end
```

20110415115954_create_tags.rb

```
class CreateTags < ActiveRecord::Migration
  def self.up
    create_table :tags do |t|
      t.string :word
      t.float :tf_idf
      t.integer :user_id

      t.timestamps
    end
  end

  def self.down
    drop_table :tags
  end
end
```

20110418065317_create_cached_requests.rb

```
class CreateCachedRequests < ActiveRecord::Migration
  def self.up
    create_table :cached_requests do |t|
```

```
      t.integer :user_id
      t.string :provider
      t.text :result

      t.timestamps
    end
  end

  def self.down
    drop_table :cached_requests
  end
end
```

20110419132313_add_authorization_id_to_cached_requests.rb

```
class AddAuthorizationIdToCachedRequests < ActiveRecord::Migration
  def self.up
    add_column :cached_requests, :authorization_id, :integer
  end

  def self.down
    remove_column :cached_requests, :authorization_id
  end
end
```

CONFIG

Application.rb

```
require File.expand_path('../boot', __FILE__)

require 'rails/all'

# If you have a Gemfile, require the gems listed there, including any gems
# you've limited to :test, :development, or :production.


Bundler.require(:default, Rails.env) if defined?(Bundler)
#ENV['BUNDLER_HOME']="C:\My Documents\Rails\app"

module App
  class Application < Rails::Application
    # Settings in config/environments/* take precedence over those specified here.
    # Application configuration should go into files in config/initializers
    # -- all .rb files in that directory are automatically loaded.

    # Custom directories with classes and modules you want to be autoloadable.
```

```
    config.autoload_paths += %W(#{config.root}/lib)

    # Only load the plugins named here, in the order given (default is alphabetical).
    # :all can be used as a placeholder for all plugins not explicitly named.
    # config.plugins = [ :exception_notification, :ssl_requirement, :all ]

    # Activate observers that should always be running.
    # config.active_record.observers = :cacher, :garbage_collector, :forum_observer

    # Set Time.zone default to the specified zone and make Active Record auto-convert to this zone.
    # Run "rake -D time" for a list of tasks for finding time zone names. Default is UTC.
    config.time_zone = 'Pacific Time (US & Canada)'

    # The default locale is :en and all translations from config/locales/*.rb,yml are auto loaded.
    # config.i18n.load_path += Dir[Rails.root.join('my', 'locales', '*.{rb,yml}').to_s]
    # config.i18n.default_locale = :de

    # JavaScript files you want as :defaults (application.js is always included).
    config.action_view.javascript_expansions[:defaults] = %w()

    # Configure the default encoding used in templates for Ruby 1.9.
    config.encoding = "utf-8"

    # Configure generators
    config.generators do |g|
      g.test_framework :rspec
      g.fixture_replacement :factory_girl, :dir => "spec/factories"
    end

    # Configure sensitive parameters which will be filtered from the log file.
    config.filter_parameters += [:password]
  end
end
```

boot.rb

```
require 'rubygems'

# Set up gems listed in the Gemfile.
ENV['BUNDLE_GEMFILE'] ||= File.expand_path('../../Gemfile', __FILE__)

require 'bundler/setup' if File.exists?(ENV['BUNDLE_GEMFILE'])
```

database.yml

```
# SQLite version 3.x
```

```
#   gem install sqlite3

development:
  adapter: sqlite3
  database: db/development.sqlite3
  pool: 5
  timeout: 5000
#development:
#  adapter: mysql2
#  database: socialnetworksearch
#  pool: 5
#  timeout: 5000

# Warning: The database defined as "test" will be erased and
# re-generated from your development database when you run "rake".
# Do not set this db to the same as development or production.


test:
  adapter: sqlite3
  database: db/test.sqlite3
  pool: 5
  timeout: 5000

#test:
#  adapter: mysql2
#  database: socialnetworksearch_test
#  pool: 5
#  timeout: 5000




production:
  adapter: sqlite3
  database: db/production.sqlite3
  pool: 5
  timeout: 5000

#production:
#  adapter: mysql2
#  database: socialnetworksearch
#  pool: 5
#  timeout: 5000
```

Config.yml

```
development: &development
  twitter:
    key: aWSfkZhK2qfcOrVWqM0Rrg
    secret: mQaX6QTBVRT0AXb0m2tTvT04j63mDyjrtenztvsy8E8
  facebook:
    key: 2c54454543dbfe1a3d3a6fb63e4f6b6c
    secret: a9f3b6342c152cbd47948a3480a0ff8e
  linked_in:
    key: 4zDVLKF3QjiundO4-e-XC4rhLzhJyWcIdX8OpPHX21YNXawXFWIPTot0EcGElPqb
    secret: _-xxXs9PP_n_iTtI2krrnYLerKN8VUaP-FcXT4Uqqn8uiwgPXNT_vWVc4YPTWg0F
  app:
    tags_per_page: 20


production:
  <<: *development

test:
  <<: *development
```

Environment.rb

```
# Load the rails application
require File.expand_path('../application', __FILE__)

# Initialize the rails application
App::Application.initialize!

# show form error messages inside the generated forms
ActionView::Base.field_error_proc = Proc.new do |html_tag, instance|
  if html_tag =~ /<label/
    %|<div class="fieldWithErrors">#{html_tag} <span class="error">#{[instance.error_message].join(', ')}</span></div>|.html_safe
  else
    html_tag
  end
end
```

routes.rb

```
App::Application.routes.draw do
  resources :authorizations, :only => [:destroy]
  resources :users, :only => [:edit, :update] do
    member do
      get 'impersonate'
    end
```

```
end

match '/tags/:id', :to => 'pages#home'
match '/cron/run', :to => 'cron#run'
match '/logout', :to => 'sessions#destroy'
match '/auth/:provider/callback', :to => 'sessions#create'

root :to => "pages#home"

# The priority is based upon order of creation:
# first created -> highest priority.

# Sample of regular route:
#   match 'products/:id' => 'catalog#view'
# Keep in mind you can assign values other than :controller and :action

# Sample of named route:
#   match 'products/:id/purchase' => 'catalog#purchase', :as => :purchase
# This route can be invoked with purchase_url(:id => product.id)

# Sample resource route (maps HTTP verbs to controller actions automatically):
#   resources :products

# Sample resource route with options:
#   resources :products do
#     member do
#       get 'short'
#       post 'toggle'
#     end
#
#     collection do
#       get 'sold'
#     end
#   end

# Sample resource route with sub-resources:
#   resources :products do
#     resources :comments, :sales
#     resource :seller
#   end

# Sample resource route with more complex sub-resources
#   resources :products do
#     resources :comments
#     resources :sales do
#       get 'recent', :on => :collection
#     end
```

```
  #   end

  # Sample resource route within a namespace:
  #   namespace :admin do
  #     # Directs /admin/products/* to Admin::ProductsController
  #     # (app/controllers/admin/products_controller.rb)
  #     resources :products
  #   end

  # You can have the root of your site routed with "root"
  # just remember to delete public/index.html.
  # root :to => "welcome#index"

  # See how all your routes lay out with "rake routes"

  # This is a legacy wild controller route that's not recommended for RESTful applications.
  # Note: This route will make all actions in every controller accessible via GET requests.
  # match ':controller(/:action(/:id(.:format)))'
End
```

11. REFERENCES

1. "How to deal with information overload brought on by Social media", Retrieved from http://marketingwizdom.com/archives/2048
2. "World cup of social networks", Retrieved from http://www.vincos.it/world-map-of-social-networks/
3. Paul,R., & Roger, G. "Comparing corpora using frequency Profiling", Retrieved from http://www.comp.lancs.ac.uk/~paul/publications/rg_acl2000.pdf
4. Pingdom. (2008, August): http://royal.pingdom.com/2008/08/12/social-network-popularity-around-the-world/
5. T.Peterson, E. (2010). Know Where Your Visitors Have Been: beencounter: http://tech.webanalyticsdemystified.com/2010/02/know-where-your-visitors-have-been-beencounter.html
6. Thadani, R. (2010). Pros and cons of Facebook: http://www.buzzle.com/articles/pros-and-cons-of-facebook.html
7. Trevin, W., "Weighing the Pros and Cons of Twitter", Retrieved from http://trevinwax.com/2009/08/04/weighing-the-pros-and-cons-of-twitter/
8. *Paul, H., "Death by Information Overload", Retrieved from* http://hbr.org/2009/09/death-by-information-overload/ar/1
9. "The Asymptotic Twitter Curve" , Retrieved from http://headrush.typepad.com/creating_passionate_users/2006/12/httpwww37signal.html
10. "What's behind LinkedIn's Mspoke acquisition?", Retrieved from http://hitechenergy.blogspot.com/2010/08/whats-behind-linkedins-mspoke.html

11. MVC Architecture:

http://1.bp.blogspot.com/_R2pbFBgV4uk/TItwslMxZPI/AAAAAAAAA9s/vu80e5mAbEY/s1600/mvc-rails.png

12. Elena, D., & Wolfgang,N. "Integrating RDF Querying Capabilities into a

http://wortschatz.uni-leipzig.de/~fwitschel/papers/ordIDag.pdf

13. Distributed Search Infrastructure", Retrieved from

http://www.l3s.de/web/upload/documents/1/IntegratingRDFQueryingCapabilities.pdf

14. Eoin, W., & Robin, T. "Managing Information Overload: Examining the Role of the Human Filter", Retrieved from http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1718455

15. David, K. F., "Designing for Selective Reading to Address Information Overload", Retrieved from http://faculty.washington.edu/farkas/TC510/FarkasSelectiveReadingDraft.pdf

16. Programmable Web: http://www.programmableweb.com/

17. Social Plugins, Like plugin feature retrieved from http://developers.facebook.com/docs/reference/plugins/like

18. Social Plugins, Activity feed plugin feature retrieved from http://developers.facebook.com/docs/reference/plugins/activity

19. Social Plugins, Recommendations plugin feature retrieved from http://developers.facebook.com/docs/reference/plugins/recommendations

20. Social Plugins, Comments plugin feature retrieved from http://developers.facebook.com/docs/reference/plugins/comments

21. Young.,(2010, April).How to Add Facebook Social Plugins to your Blog. Retrieved December 20, 2010, from http://freenuts.com/how-to-add-facebook-social-plugins-to-your-blog/

22. Damien.,(2010, April). The Howto Guide to Add Facebook Social Plugins to Your WordPress Site. Retrieved December 20, 2010, from http://maketecheasier.com/howto-guide-to-add-facebook-social-plugin-to-your-site/2010/04/28

23. Patty.S., Outside Innovation from http://outsideinnovation.blogs.com/pseybold/2006/03/why_mash_ups_ma.html

24. Dingyi,C., Xue,l., Jing, l., & Xia, C. "Ranking-Constrained Keyword Sequence Extraction from Web Documents" Retrieved from http://crpit.com/confpapers/CRPITV92Chen.pdf

25. Github Social Coding, https://github.com/intridea/omniauth