National College of Ireland

The college for a learning society

National College of Ireland

Master of Science in Web Technologies

September 2011

Thesis

Profiling and Indexing Application for use in Social Networks

Author: Sean Cahill
Student Id: x10206884

I hereby certify that this material, which I now submit for assessment of the programme of study leading to the award of Master of Science in Web Technologies is entirely my own work and has not been taken from the work of others save and to the extent that such work has been citied and acknowledged within the text of my work.

Signed:.....

Date:.....16 Sept 2011

Student Number:.....X10206884

The author of this thesis would like to thank Aidan Mooney and Michael Bradford for the help, advice and assistance throughout the duration of this project. He would also like to acknowledge the assistance of the NCI Library staff in their assistance when doing the research of academic papers and journals. Finally he would like to thank the members of the academic staff of the Master of Web Technologies Program for their assistance throughout the academic year on which a large part of the content of this thesis is based.

Abstract

The current, most popular social networks are great applications for connecting people who know each other. But sometimes the people who you would most want to meet are those that you don't know about. Our project focuses on how it would be possible to allow people to find others within a social network, based on their profile, their business needs and/or their interests and activities. The project developed an algorithm to index a social network member's profile. The project further created application programming interfaces (apis) to allow for efficient retrieval of matching profiles to allow profiled members of a social network find others within their network based on a number of criteria.

The resulting application can be considered as a Software as a Service (SaaS) offering for social networks, providing all the necessary functionality for a social network to index their members and includes mechanisms for efficient retrieval and matching of registered profiled members.

An event based social network application was created as a proof of concept of the work done in creating the profiler application. This application is a web based application intended to run on a smartphone platform. This application will allow signed in users of an event to make connections with fellow attendees at the event, based on their profiles, areas they wish to collaborate on and/or their interests and hobbies. The application allows users to change the context of their profile matching request on demand making it flexible in its use.

The report concludes with an examination of the work carried out so far and presents a set of areas where the project can be taken forward and developed in the future.

Contents

Index

Appendicies

Introduction

This project was motivated by the fact that it seems difficult to find people within social networks that are outside of one's immediate circle of friends and family. It is easy to find people that you know but difficult to find people that you don't know. In this project we wanted to examine if there was another way for members of a social network to find each other. Not based on who you know but on the skills you have, your interests and activities and areas you would like to explore outside of your current skill set. If we could build such a system then people within a network could find people that they don't yet know but have similar profiles to their own or with a profile that they are looking for with potential for collaboration on project they are planning to embarking upon.

Our initial focus was to research the current major social networks and determine if what we were setting out to achieve was already achievable with these social networks. We wanted to mine the data held within a user's social network profile and determine if we could extract this information in a form that was usable in the sense of using the data to index a user in such a way that could be findable by someone else with a similar profile. A large number of social networks are allowing developer's permission access to registered members profile information. This opens up a wide range of opportunities for developers to mine this information in interesting ways to create new application based on this information. We wanted to explore this in our project. Specifically with respect to building new social network applications that do not have to build their own profiling functionality but have the ability to leverage off the existing profile data stored within the major social networks and concentrate on building useful and compelling functionality of their own.

To enable us to build the infrastructure necessary to achieve our goals led us to the field of Information Retrieval and Storage. This is a large topic of research within the Information Technology community with applications in areas such as document storage and retrieval, search engines and library systems. One of the major goals within the project was to create an algorithm that was efficient enough to take the contents of a social network members profile and index it in such a way that would enable fast retrieval of the members profile based on query searches. We also wanted to create a system that was flexible enough to be able to change context based on the requirements of the network. So that a user can change from finding profile matches to finding people with similar interests to their own. Building in this functionality would make the application flexible enough to cross differing types of social network. Business related social networks where members are looking to collaborate with each other, purely social networks where people with similar interest would like to meet up and engage in dialog. The contextual change of the matching process we considered to be an important element in the design of the application as this allows flexibility in its use and makes it applicable to

As part of the project we wanted to build a small scale social network as a proof of concept of the functionality of the profiling application. This will be our showcase for demonstrating its usage and potential attractiveness to existing and potential social networks. We decided to develop an event based social network that could be used by attendees at conference events. We felt this was an ideal area for highlighting the functionality of the profiler application. Events can be specific to a business area or cross business areas as well as having a social element to them.

We believe this an ideal fit for our application allowing event attendees to connect with other users based on different criteria.

Ultimately our goal was to create a Software as a Service (SaaS) application that could be used by existing social networks that felt that what our application was offering was compelling enough to use. The application would provide all the necessary functionality to allow them to index their existing membership and add the functionality to return profile matches all through a well defined interface complying with current and state of the art standards and protocols.

Definitions

Throughout this report we use the following terms.

Bio: a social network member's biography. This is used as the basis of a user's profile.

Likes: a social network member's interests and their activities.

Needs: a social network member's description of what they would like to collaborate with others on. This is generally used to describe areas of interest outside of a member's profile.

API: Application Programming Interface. A well defined protocol that allows registered users of an application access to the application's functionality.

API Key A key provided by the owner of an application that allows client access to the applications underlying functionality

Software as a Service (SaaS): can be defined as a software distribution model in which applications are hosted by a vendor or service provider and made available to customers over a network, typically the Internet.

Profiler API: the application built for this project that indexes a users profile and allows for matching of users based on their profiles

geomingle: the event based social network application developed as part of this project

synonym: different words with almost identical or similar meanings.

anonym: an alternative name for the same thing (similar to pseudonym)

Research Methods

Our initial research focused on those areas that would help us with building up social network profile information and ways sourcing it. To do this we examined the major social networks such as Facebook, LinkedIn and Twitter. We wanted to examine these networks for how they maintain their profile information and the breadth of the information they store.

In order to undertake this process we used a number of members of these social networks, who were willing to allow us access to their profiles on these networks. This allowed us to examine the sort of information they store within these networks. Using this information we can determine if there are differences between the profile data that is stored within these networks.

We also wanted to research existing credible research that has already been done on this topic. Consequently we used the internet to trace any relevant data we could be used to aid in our research.

We wanted to determine if it was possible to extract profile information from the major social networks so that it could be processed in some way to potentially index the information contained within it. This led us to research relatively new protocols such as OAuth which is actively being taken up by social networks to allow application developers gain access to the underlying data contained within the social networks.

Once we have harvested our profile information we wanted to find ways of analysing the data and structuring it in an efficient manner. This led us to research topics including Information Retrieval, Data mining, Natural Language Processing, Pattern Matching Algorithms and Search Algorithms amongst others. These are largely topics of an academic nature. To research these areas we concentrated on sourcing academic books, journals and papers relevant to these areas of research.

Research

This section of the report highlights the research we did in the areas of interest. We initially look at techniques for information storage and retrieval. These techniques are used in areas such as search engines, data mining and natural language processing. We examine these areas to determine a way to create a suitable algorithm to aid us in designing an application that will meet the goals and objectives of our application.

The second major area of research involves analysing the major social networks to determine how they store their members profile information, if we can harvest this information and how to marshal it into a form that will comply with the requirements of the application designed as a result of the above research.

Informational Retrieval

Over the centuries there has always been an attempt to store and to be able to retrieve all forms of information. This became particularly prevalent with the advent of the modern printing press. As a result a large number of books were being produced that needed to indexed in some way, in order to have an efficient retrieval system. With the invention of the computer came the realisation that these machines could be used to store large amounts of information and programmatically retrieving this information.

One of the original visionaries of where information storage and retrieval was going in the late 1940's, was Vannevar Bush. He wrote an influential article called *As We May Think* [21] that has, to some extent, predicted the advent of the internet, online encyclopaedia and hypertext linking of documents. Bush envisioned the ability to retrieve several articles or pictures on one screen, with the possibility of writing comments that could be stored and recalled together. He believed people would create links between related articles, thus mapping the thought process and path of each user and saving it for others to experience. Bush urged scientists to turn their hand to the massive task of creating more efficient accessibility to the increasing store of knowledge. He was particularly concerned that important work by scientists was not being found by those who could expand and build on the work of other. In the article he cites the incidence of the loss of Mendel's law of genetics for generations because his publication did not reach the few who were capable of grasping and extending it.

One of the most influential works that elaborated on the idea of text searching by computer was described by H.P.Luhn in 1958 [17], in which he extended the concept of using words as the indexing unit for documents and measuring word overlap as a criterion for retrieval. In a paper he wrote [17] he states "It is here proposed that the frequency of word occurrence in an article furnishes a useful measurement of word significance. It is further proposed that the relative position within a sentence of words having given values of significance furnish a useful measurement for determining the significance of sentences. The significance factor of a sentence will therefore be based on a combination of these two measurements."

Information Retrieval is primarily concerned with the storage of documents and the efficient retrieval of these documents based on some kind of query. The criteria on which these systems can be measure is based on the quality of the returned matching documents. The most effective

systems also rank the documents by some kind of relevance and order them by the most relevant, descending. Informational Retrieval systems also rank documents by their estimation of the usefulness of a document for a user query.

The implementation of an Information Retrieval system generally follows a standard procedure. The document is broken down in to component parts. This may be by each single word within a document or a more sophisticated approach may extract phrases from the document the system may deem relevant. Other information may be attached to the words or phrases, for example to indicate relevance. A word that appears in the title of a document may be given more weight than that which appears at the end of the document. Once the document has been broken down into its component parts it may undergo further processing. This may include the removal of stop words, the stemming of the remaining words and ultimately the indexing of the remaining words into a form that enables fast retrieval of document based on a user query. These processes are discussed in the following sections.

Stop Words

To improve the quality of a document (or profile) containing words to be parsed and indexed extremely common words which would appear to be of little value in helping indexing a document and generally are excluded from the text entirely. These words are called *stop words*. They generally include words such as *I*, *am*, *be*, *and*, *the* etc. The general strategy for determining a stop list is to sort the terms by *collection frequency* (the total number of times each term appears in the document collection), and then to take the most frequent terms, often hand-filtered for their semantic content relative to the domain of the documents being indexed, as a *stop list*, the members of which are then discarded during indexing. Other strategies use a dictionary of fixed terms that are used to remove matching words from a document.

Using a stop list significantly reduces the number of keywords that a system has to index. Some statistics on this are taken from Manning[4] in Chapter 5(see Table 5.1) where he indicates that up to 47% of words are removed from a document with a stop list of 150 stop words.

Most of the time not indexing stop words does little harm, keyword searches with terms like *the* and *by* don't seem very useful. However, this is not true for phrase searches. The phrase query *President of the United States*, which contains two stop words, is more precise than *President* AND *United States*. A search for Vannevar Bush's article *As we may think* will be difficult if the first three words are stopped out, and the system searches simply for documents containing the word *think*. Some special query types are disproportionately affected. Some song titles and well known pieces of verse consist entirely of words that are commonly on stop lists (To be or not to be, Let It Be).

Stemming

Stemming is the process of conflating various forms of the same word and reducing it to its root form. Stemming can broadly be described in two forms, weak stemming and strong stemming. Weak stemming generally refers to the removing of plurals e.g. bicycles becomes bicycle. Strong stemming removes suffixes to leave behind the root of a word, for example *book illustrator* and *book illustration,* using strong stemming we would drop both the *–or* and the *–ion* leaving

behind *book illustrat* . It can be seen that this techniques is useful in our case as instead of having only one common word in the two phrases we now have both words that we can match on.

There are several types of stemming algorithms which differ in respect to performance and accuracy and how certain stemming obstacles are overcome.

Brute-force algorithms

Brute force stemmers employ a lookup table which contains the relationship between root forms and inflected forms, where an inflected form is the alteration of the form of a word by the addition of an affix. For example the inflected form of the root from *develop* would include inflected forms *development*, and *developer*. To stem a word, the table is queried to find a matching inflection. If a matching inflection is found, the associated root form is returned.

Brute force approaches are criticized for their general lack of elegance in that no algorithm is applied that would more quickly converge on a solution. In other words, there are more operations performed during the search than should be necessary. Brute force searches consume immense amounts of storage to host the list of relations (relative to the task). The algorithm is only accurate to the extent that the inflected form already exists in the table. Given the number of words in a given language, like English, it is unrealistic to expect that all word forms can be captured and manually recorded by human action alone. Manual training of the algorithm is overly time-intensive and the ratio between the effort and the increase in accuracy is marginal at best.

Brute force algorithms do overcome some of the challenges faced by the other approaches. Not all inflected word forms in a given language "follow the rules" appropriately. While "running" might be easy to stem to "run" in a suffix stripping approach, the alternate inflection, "ran", is not. Suffix stripping algorithms are somewhat powerless to overcome this problem, short of increasing the number and complexity of the rules, but brute force algorithms only require storing a single extra relation between "run" and "ran". While that is true, this assumes someone bothered to store the relation in the first place, and one of the major problems of improving brute force algorithms is the coverage of the language.

Brute force algorithms are initially very difficult to design given the immense number of relations that must be initially stored to produce an acceptable level of accuracy (the number can span well into the millions). However, brute force algorithms are easy to improve in that decreasing the stemming error is only a matter of adding more relations to the table.

Suffix-stripping algorithms

Suffix stripping algorithms do not rely on a lookup table that consists of inflected forms and root form relations. They generally employ a smaller list of rules which provide the basis of the algorithm, which when employed for a given an input word form will find its root form. Some examples of the rules include:

if the word ends in 'ed', remove the 'ed'
if the word ends in 'ing', remove the 'ing'

if the word ends in 'ly', remove the 'ly'

Suffix stripping approaches enjoy the benefit of being much simpler to maintain than brute force algorithms, assuming the maintainer is sufficiently knowledgeable in the challenges of linguistics and encoding suffix stripping rules. Suffix stripping algorithms are sometimes regarded as crude given their poor performance when dealing with exceptional relations (like 'ran' and 'run'). Suffix stemming does not do general substring matching or pattern matching. So, for instance, it should make *walk*, *walking*, *walked*, and *walks* all match in searching, but it will not make *walking* a match for *king*. This feature is important for our purposes as we don't want to do any form of pattern matching when retrieving profiles based on keyword searches. Although pattern matching may prove useful in finding common word patterns between to matched profiles so that we can visually see what terms are common in their profiles.

Suffix stemming is not without its critics. David Hull [15] outlined some of the concerns with this approach. In his article he highlights that fact that words with different meanings can be conflated to the same stem and words with similar meaning are not conflated at all. He also highlights the fact that stemmed words often end up as words that have no meaning. In our case this should have no impact as we only use the stemmed words as keywords to index. The actual words are hidden to the user and are not required to have any meaning other that as a key to an index of profiles containing that stemmed word. He states also that these words no longer have the ability to do a dictionary look up, which is true in the case of words that are stemmed to words that no longer have any meaning. In addition he highlights that fact that these words would not be able to be translated into other languages. Again these criticisms are not applicable in our situation.

The effectiveness of stemming is explored in Frakes (1992) [2] in his book he presents a summary of Information Retrieval performance reporting that the combined results make it unclear whether any stemming is useful. In contrast Krovetz, R. (1993) [16] reports a 15-35% increase in performance when stemming is used on some collections. The interesting outcome of this analysis is that these results were achieved on small collections. Larger documents showed less performance improvements. This is significant in our case as the data we are dealing with is not large in comparison to large documents like academic papers. We generally anticipate small collections of data when dealing with profiles stored in social networks.

One of the most often referenced of the stemming algorithms is the Porter Stemmer [18] [15]. This was developed by Martin Porter in 1979. His algorithm is widely used and probably one of the best understood. The algorithm is very simple in concept, with *ca.* 60 suffixes, two recoding rules and a single type of context-sensitive rule to determine whether a suffix should be removed. Rather than rules based on the number of characters remaining after removal of a suffix, Porter uses a minimal length based on the number of consonant-vowel-consonant strings (the *measure, m*) remaining after removal of a suffix. A more detailed outline of the Porter Stemmer Algorithm is given in APPENDIX VI

Indexing

Most Information Retrieval systems are based on the inverted list data structure. This enables fast access to a list of documents that contain a term along with other information (for example, the weight of the term in each document, the relative positions of the term in each document, etc.). A typical inverted list takes the common form.

$$P(i) \longrightarrow \langle d(0),\ldots.\rangle,\langle d(b),\ldots.\rangle,\ldots..\langle d(n),\ldots.\rangle$$

Where P is a phrase-i is contained in documents d(0),d(b),.....,d(n)

Inverted lists exploit the fact that given a query most Information Retrieval systems are only interested in returning a small number of documents that contain some query term. A query that contains only one word phrase will only be required to find one indexed term, which will provide a link to all documents containing that term. Once these documents have been returned some form of ranking can be employed to determine the frequency ranking of these documents and return the documents in order of relevance.

Social Network Research

The research on social networks is focused on how users within social networks can make contact with each other and the mechanisms for enabling this. We also focused on whether it is possible to achieve our aims and objectives of making contact with other members of the network outside of friends and friends of friends, within the existing functionality of these popular web sites. We also look at ways of sourcing profile information from the most popular social network sites. These include Twitter, Facebook and Linkedin. At the time of writing this document Google have announced the introduction of its own social network called Google+ . Initial indications are that Google+ will be a successful competitor to the likes of Facebook and Linkedin, but currently registration is restricted by invitation only. So for our purposes we will not present a comprehensive analysis of Google+ here.

One of the objectives of our project is to create profile information for our users. It doesn't seem productive to create a completely new interface to allow users re-input profile information to our system or to any proposed new social network. If this were the case it is likely that potential users would be put off from registration and be unlikely to engage with the application. Consequently we propose to source profile information from one of the major social networking sites as the basis for building our user profile.

The following sections outlines the profile map of each of the three major social network sites and a determination is made as to which is the best source of information for our purposes and/or a combination of them all.

Before embarking on this analysis we present an explanation of an emerging standard OAuth. OAuth stands for "open authorization" and is quickly becoming an industry standard particularly among the social networking sites. This scheme is "three legged" in that it involves an exchange of information (often called a "dance") among a client application that needs access to a protected resource, a resource owner such as a social network, and an end user who needs to authorize the client application to access the protected resource (without giving it a username/password combination).

In a nutshell, OAuth provides a way for you to authorize an application to access data you have stored away in another application without having to share your username and password. This is useful for social networks that don't want to create their own profile functionality but source profile information from existing networks such as Facebook. This is the procedure we intend to follow when creating our own social network *geomingle*.

For a more detailed outline of the process for accessing Facebook profile information using the OAuth protocol see Appendix II.

Facebook Profile Information

The main details that are stored in a Facebook profile are the following. The components are listed as:

Formatted	-	users input is validated
Auto-suggest	-	user is prompted with suggested keywords based on input
Free format	-	user can type in anything, not validated.

Main Profile Data

Name	-	Formatted
Country,County,City	-	Formatted
Gender	-	Formatted
Date of Birth	-	Formatted
About me:	-	Free format

Education & Work

Employer	-	Auto-suggest
College	-	Auto-suggest
High School	-	Auto-suggest

Interests & Hobbies

Music	-	Auto-suggest
Books	-	Auto-suggest
Movies	-	Auto-suggest
Television	-	Auto-suggest
Games	-	Auto-suggest
Favourite Sports	-	Auto-suggest
Favourite teams	-	Auto-suggest

With the aid of the OAuth process outlined in Appendix II we have the ability to extract this data and marshal it into a form that can be indexed by the Profiler API application.

Facebook has a number of ways for users to connect with other registered users. The main way is using a "search and discovery" on people the user already knows. The user can type in the name of a potential contact and the application will return a list of registered users with that name. Once a contact has been found the users asks the contact if they can be their friend. If the contact accepts the request then they are then connected.

The other ways users can find people to connect with include

- Use a users contact email list to find registered users with these email addresses
- Search friends of friends or mutual friends

- Find people within a particular employer
- Find people in a particular hometown or own hometown
- Find people from particular high school
- Find people from a particular collage
- Find people from a particular Graduate school

The application will suggest possible friends to a registered user. These suggestions are generally mutual friends. It can also be as crude as suggesting people with the same surname as the registered user.

Individual members of Facebook have no way of finding other registered users with profiles similar to their own or to connect with people with similar interests.

The area where Facebook comes close to our applications goal of matching people with similar profiles is via their ad creation facility. This facility allows advertisers to target registered members based on the following criteria.

Location	-	country, city
Demographics	-	age range, marital status, gender
Interests		
Education level		
Work place		

This facility doesn't allow for targeting registered users to the level of the detail contained within their profile.

LinkedIn Profile Information

Linked is a social network predominately targeted at the business professional. It has a number of ways of making connections. They allow a search and discover based on the registered user's email addresses contained within a users email contact list. The system will then attempt to find users within the network based on these email addresses. The site recommends that users should only invite people to join their network that they know well and who know you.

The social network allows for the creation of and joining of groups where discussions can be entered into. This facility relies on the owner of the group accepting a new user who wishes to join the particular group. This is one way that users with a similar profile can connect to each other.

The information that LinkedIn allows users to store is much more detailed than Facebook. We detail it below. Irrelevant details are left out.

Current Position		
	Company Name:	unformatted
	Title:	unformatted
	Location:	auto-suggest
	Time period:	formatted
	Description	unformatted
Past Position	as above	
Education		auto-suggest
Summary		

	Experience and goals :	unformatted
	Specialities	unformatted
Skills		auto-suggest
Interests		unformatted
Groups & Associations		unformatted
Projects		unformatted

A detailed breakdown of the demographics of LinkedIn registered users for Ireland can be found in Appendix I.

Twitter Profile Information

Twitter keeps a minimal amount of information about a user in their profile. This includes

Bio: information about the user (limited to 160 chars)
Name:
Web Site:
Location:

Potentially the most interesting information on twitter is the detail of who a user is following and who is following them. This data, if mined, could provide a way of matching two users based on the similarities in who is following them and who they are following. This information can be retrieved by an api call once the necessary access keys have been obtained. There are however certain limits on the amount of data twitter will allow an api call to make. Currently these limits restrict the user to 350 api calls per hour and an api call is restricted to 100 users per api call. So in theory it would be possible to return 35,000 followers per hour.

An example of how to make calls to twitter via its api is given below using an open source java library once we've obtained the necessary access tokens.

```
import winterwell.jtwitter.*;
public class Main {
        public static void main(String[] args) {
        String consumer_key = "2cgkPKxNbaylC2eQ1OKcgw";
        String consumer_secret = "ILkglJUGxkh2Bl4CgXC73T4XMQzDES2ktvlezuTri0";
        String token = "252655570-aB5XuLfdtSoMPvhTlyzv5HwsDSlgHw1ofFYATNim";
        String secret = "AoGv8PKrSMOFrjGGh1f2dazDAeuQvXfk5aOW3vlw";

        OAuthSignpostClient oauthClient = new OAuthSignpostClient(consumer_key, consumer_secret,
        token,secret );
        // Make a Twitter object
        Twitter twitter = new Twitter("eventsindublin", oauthClient);
        // we can now interrogate the twitter object to get useful information
        String handle = "eventsindublin";
        Twitter twitter = new Twitter(handle, oauthClient);
        Twitter.User user = twitter.getUser(handle);
        List<Twitter.User> followers = twitter.getFollowers(handle);
        List<Number> friends = twitter.getFriendIDs(handle);
        System.out.println("Twitter@: " + user.screenName + " location: " + user.location);
        System.out.println("Is Followed By: ");
        for (Twitter.User u : followers)
        System.out.println("\t" + u.screenName);
        System.out.println("Is Following: ");
```

```
for (Number id : friends)
System.out.println("\t" + twitter.getUser((Long)id).screenName);
```

This produces the following output

```
Twitter@: eventsindublin location: Dublin

bio     A handle for publicising events in Dublin.
        Primarily focused on the Business community

Is Followed By:
    begin_again, brianstorme, businesstartup, o_r_m

Is Following:
    broadsheet_ie, businesstartup, css3pie, mediaflash, politico_ie, thejournal_ie,
    boards_ie, politicise, dublinwebsummit, businessdublin
```

One can see how this information could be useful in matching users with similar interests. If two users were both following a large number of the same twitter handles, you might conclude that those two users had very similar interests. The limits imposed by the twitter api should not be an issue for most users but some exception handling would need to be incorporated for power users, such as limiting the number of followers and following returned and some timeout functionality if an api call is taking too long to return.

If we run the above code snippet for two users we can then do a simple set intersection to determine common followers and friends.

We ran the above for another twitter handle "*newmediamatters*" and got the following results:

```
Twitter@: newmediamatters location: Dublin


bio online marketing company
Is Followed By:
Is Following:
    css3pie, html5andcss3, designingdublin, boards_ie, innovationdub dublinwebsummit,
    mashable
```

Running a set intersection gives us the following matches

```
Set s1; //user1 friends and followers
Set s2; //user2 friends and followers

s1.retainall(s2);

for (Object x : s1) {
        System.out.println(x.toString());
}
```

Produces the following

```
{css3pie, boards_ie, dublinwebsummit }
```

This data is potentially useful information that could be included in any profiling indexing applications as a means of matching people within a social network who have matching twitter handles that they are following and are followed by.

Research Conclusions

Based on the research we have done we propose to distinguish between different types of text that is present, generally, in the profiles of members of social networks.

Free text	-	unformatted
Well formed text		formatted.

This distinction is used to determine how the text will be processed and ultimately indexed.

An example of free text is a user's *bio* within Facebook. This text has no validation placed upon it and can contain any words within the body of the text. We propose that this type of data will go through a clean-up process, removal of stop words and a stemming process.

For text that we consider well formed we will not strip stop words as this may have the effect of changing the meaning of the phrase, for example the title of a song may contain many stop words that if removed would strip the phrase of its meaning and diminish it of its relevance. Text that falls into this category includes interest and activities within Facebook. Facebook applies an auto-suggest facility to prompt users as to the correct phrase e.g. the title of a book or the title of a song. We propose to delimit these phrases (e,g with a comma). These phrases will then be directly indexed by the indexing process by splitting on this delimiter. Other such data could be the data we retrieved from twitter (see previous section). We would take the twitter handles from the process described in the previous section and simply join them together by a delimiter

A lot of social networks allow certain data to be hash tagged in some way to give the associated data some semantic meaning. Again this type of data could be considered to be well formed. But it may improve the quality of the data if it was stemmed. If we look at the Digg social network they allow articles to be hash tagged but they may be tagged with phrases such as *web developer* and *web development*, these phrases could be considered as similar in meaning. If we did not stem these phrases then they would be separately index and not associated in any way.

Based on our research we decided to use the techniques of removing stop words and apply stemming techniques to process free text. We have allowed for a client to make additions to the list of generic stop words provided by the application. This will allow client to provide their own stop word for words that they deem irrelevant to the keyword indexing process.

We decided that the Porter stemmer algorithm[18] best met our requirements and have decided to implement this as one of the major components of our algorithm. This stemming algorithm is repeatedly referenced in the literature as one of the leading implementation of a suffix stemming algorithm and has been widely translated into a number of programming languages including python and java.

The ability to access a users profile from Facebook, Linkedin and Twitter is a powerful tool and saves the headache of having to build separate profiling functionality within a new social network. The data from Facebook and Linkedin is rich in the content that can be extracted with the OAuth protocol described in Appendix II. Linkedin is a more business focused network than Facebook and potentially gives more compelling content for use in business networks. The data within a LinkedIn profile is potentially richer in content than data within Facebook for the purposes of indexing a user's profile. Although this depends to what extent the user has updated their profile with compelling information. The data from Twitter is interesting and requires more

investigation, particularly the data on a user's followers and followed. It is unlikely that the profile information contained within Twitter is as compelling as that contained within Facebook and LinkedIn.

With these issues in mind, we decided to use Facebook as our profile repository for our *geomingle* social network application. This decision is based largely on the fact that it is almost 4 times more popular than Linkedin (see appendix I). As our application is an event based application it was deemed that attendees would have a much higher probability of being a registered user of Facebook than Linkedin. As a consequence we can potentially get more attendees to sign on to the application that if we were using Linkedin as the repository of the profile data. Another factor in the decision was the diversity of the data that makes up a users profile in LinkedIn. There is a large amount of data that potentially make up a users profile and it would take considerably more effort to marshal the data into a form usable by our event based social network

Application Architecture and Design

This section of the report details the technology used to create both the Profiler API application and the *geomingle* social networking application. It discusses the technology used and the design and architecture of both applications.

Technology

The following technologies were used in the creating of the various applications.

Profiler API Application

Languages
- Java, Groovy, Javascript
- JQuery
- HTML, CSS

Framework
- Grails

IDE
- Netbeans 6.9

Database
- MongoDB

Deployment
- Amazon Elastic Beanstalk
- MongoHQ

Geomingle Social Networking Application

Languages
- Python, Javascript
- JQuery
- HTML5, CSS3

Framework
- Google App Engine

IDE
- JetBrains PyCharm 1.5.2

Database
- Google App Engine Datastore

Deployment
- Google App Engine

Technology decisions

Frameworks

When deciding what framework to develop our Profiler API application we wanted to find a framework that allowed for integration of Java code, as this was the platform for the development of our algorithm. We looked at the popular frameworks such as Spring MVC and Struts.

Spring MVC requires a lot of configuration to get your applications environment setup. This is time consuming and tedious. In the end we went with a relatively new framework, Grails. Grails is a modern web development framework that mixes familiar Java technologies like Spring and Hibernate with contemporary practices like convention over configuration. Grails framework sits on top of the Spring MVC framework but relieves the user from the problems of configuring an applications environment. To create an application we simply run the command

- grails create-app <application-name>

This command will create an application with all the necessary folders and files created up front, such as folders for domain model classes, controllers and views. It configures all the necessary configuration file to enable the underlying Spring MVC to work seamlessly. Grails is very similar in concept to the Ruby on Rails framework environment and follows similar conventions based on the MVC architectural design pattern.

Written in Groovy, Grails gives seamless integration with legacy Java code while adding the flexibility and dynamism of a scripting language. As different as Grails development is from other typical Java Web frameworks, you still end up with a Java EE-compliant WAR file. This allows for easy deployment to production. In our case we deployed the WAR file to Amazon Elastic Beanstalk platform. This is a simple process requiring no configuration. We simply need to upload the compiled WAR file to the Amazon environment and it is ready to run.

For the *geomingle* social network application we decided to use the Google App Engine framework. GAE comes with a built in framework called webapp. This is a lightweight framework for developing GAE applications. For our purposes this proved sufficient. The GAE allows users to integrate more complex frameworks like django. But as our application was relatively straight forward we decided to stick with webapp.

Databases

For storing our indexed data generated by our Profiler API application we decided to use one of the new noSql databases, MongoDB, MongoDB has built-in support for horizontal scalability. MongoDB allows users to build and grow their applications more rapidly. With auto-sharding, you can easily distribute data across many nodes. Replica sets enable high availability, with automatic failover and recovery of database nodes within or across data centers. As we anticipate that our database will need to embrace issues such as scalability and availability the features that MongoDB provide are attractive over traditional SQL databases. Making this decision in the initial phase will remove the need to transition off an SQL database if it proves incapable of scaling as the application grows in size. Data in MongoDB is stored in JSON-like documents with dynamic schemas, providing flexibility during the development process.

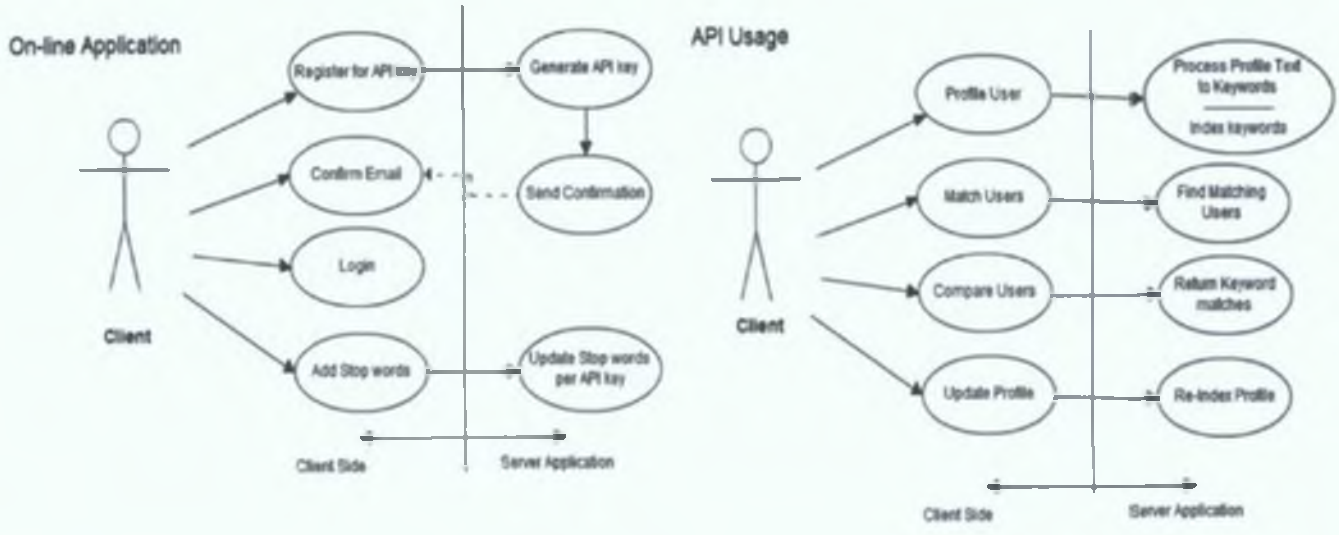
Users who develop on the Google app engine platform have no choice but to use their proprietary database called the *datastore*. This is a NoSQL database that has many differences to more traditional Relational databases.

Profiler API Application

The design and architecture of the Profiler application is given in the following sections

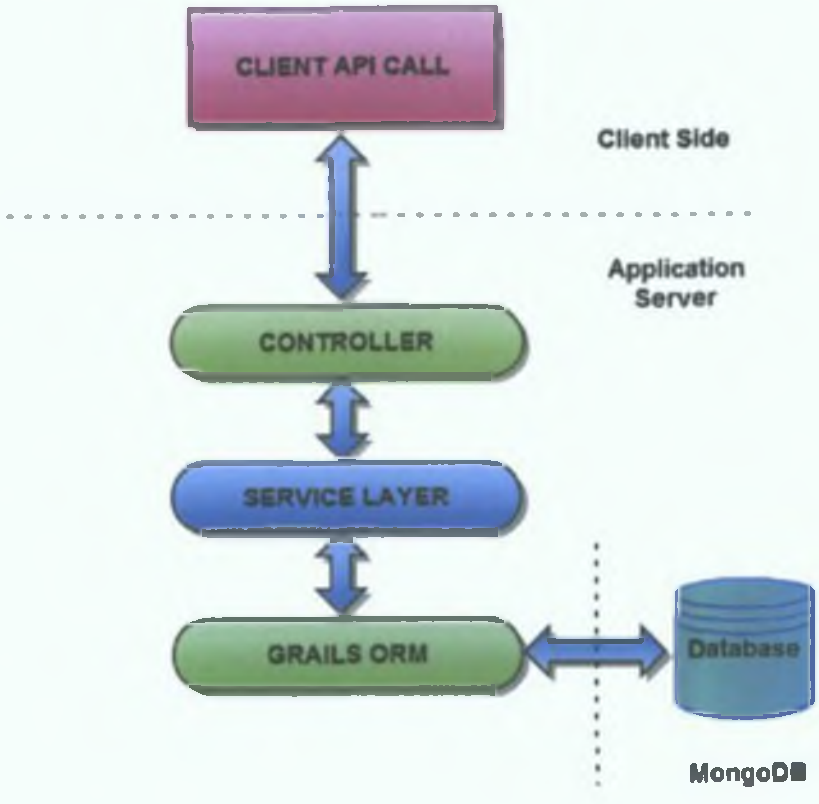
Use Cases

Based on our initial analysis and design we came up with the following use cases.



Architectural Design Patterns

Both applications were built using the MVC (Model View Controller) architectural pattern. The architecture of the Profiler API application can be show in the diagram below



User Object Model

Based on our research we propose an object model that will be used within the Profiler API application, to cater for the profile matching we are attempting to implement.

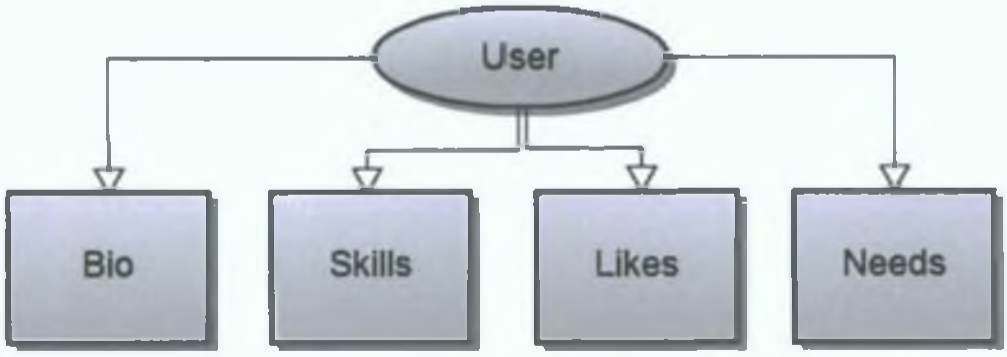
The User model will have four components

Bio: A summary of the users profile. This will be made up of free, unformatted text

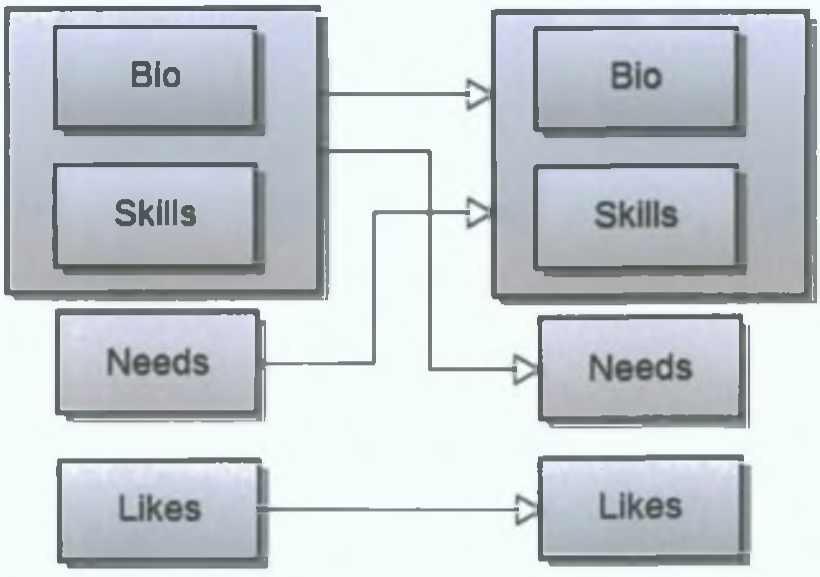
Skills: A formatted, delimited list of a users' skill set

Likes: A formatted, delimited list of a users interest and hobbies

Needs: A summary of what a user would like to collaborate on.

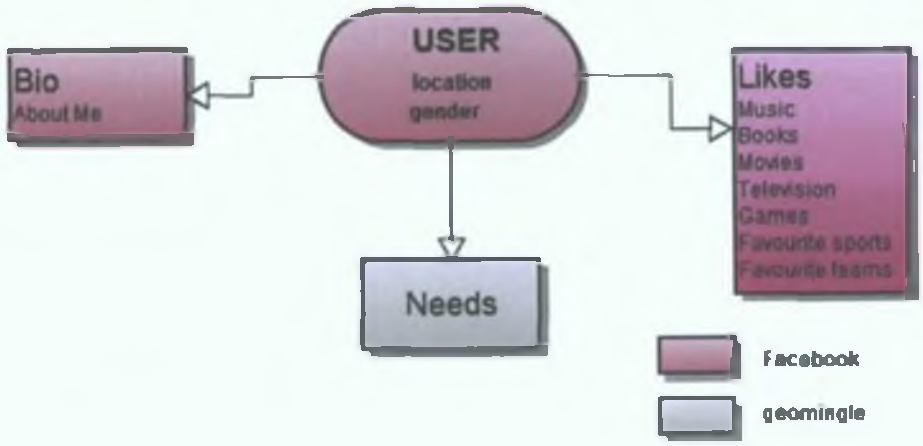


The Profiler API will attempt to find contacts based on the following scenarios



As an example we can show how a Facebook profile can be marshalled into a form compliant with the Profiler API application, we use our *geomingle* social network as an example.

The Facebook profile has no ability to enter a formatted skills profile. The About me section can be used to enter all relevant skills, but in a free formatted manner. There is no section in Facebook to enter details of what a user would like to collaborate on. The *geomingle* application will provide the ability to enter the "needs" details. The marshalled user object will resemble the following object model.



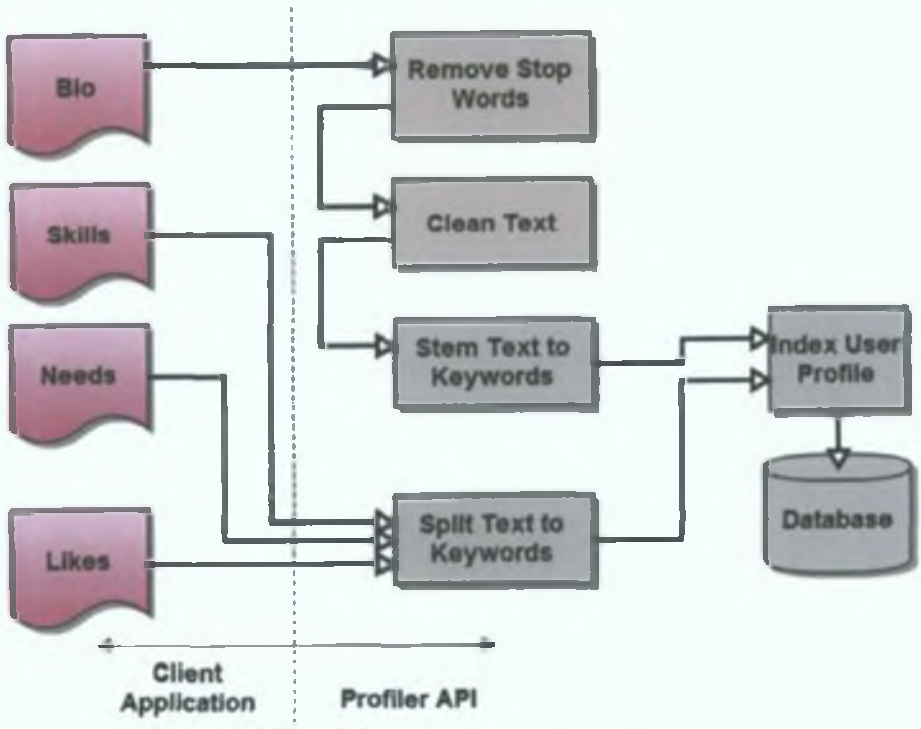
Once a user profile has been marshalled into a format compatible with the Profiler API application the user profile can be indexed using the Profiler API's. These api's will send the profile data through a number of processes before indexing takes place. These processes will depend on the data type. In the application we distinguish between Free Text, text that has no validation or pre-processing on it, and formatted text that has some validation and pre-processing applied to it.

Free Text will undergo processes to

- Remove stop words
- Clean text by stripping non alpha characters (with exceptions)
- Stem remaining words
- Proceed to indexing

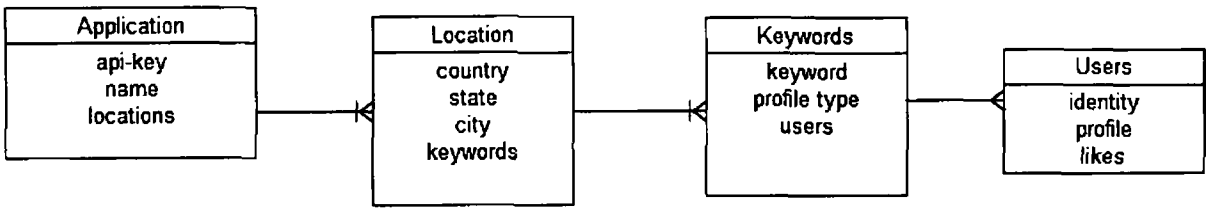
Formatted text

- Will split text into keywords by expected delimiter
- Proceed to indexing



Indexing

The Profiler API application will index a processed profile based on the following model



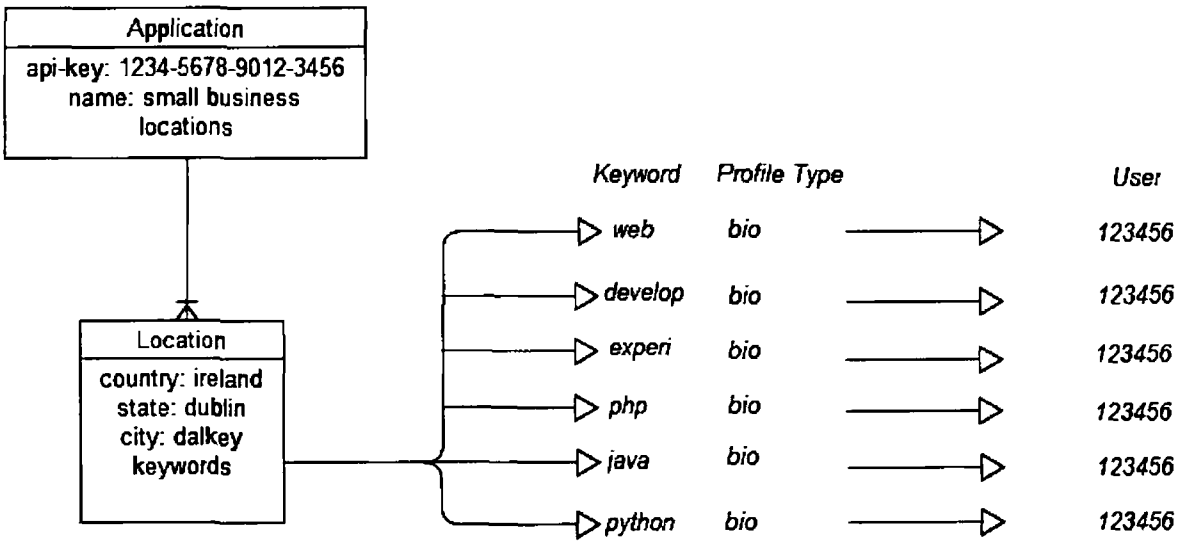
To see how a profile would be indexed we use the following example.

Api-Key: 1234-5678-9012-3456
Client name: "small business"
User Identity: 123456
User location: Ireland,Dublin,Dalkey
User Bio: i am a web developer with experience in php, python, java.

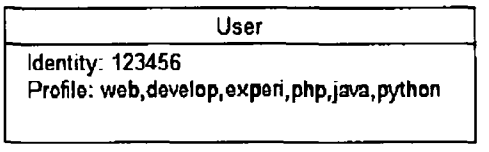
Running this bio through the text processing functionality results in the following keyword terms

web
develop
experi
php
python
java

After running through the indexing process our index will resemble the following



As part of the profiling process we store these keywords with the User object as a coma separated string. This will assist us in retrieval of matches. The resulting user object will look like.



Once the users have been profile we can retrieve matches in the following way

Find Profile matches

Required parameters api-key, country, state, city, user identity

```
find application record by api-key
find user by user identity
find application location by country, state and city
split user profile to keywords by delimiter
for keyword in user profile keywords
        find keyword in locations keywords by keyword profile type 'bio'
        store list of users associated with keyword
                exclude self
create frequency map of list of users and number of times they appear in list
return sorted frequency map
```

Find needs to profile matches

The difference between this procedure and the procedure above is that we use the user needs rather than the user profile (or bio). If the user needs are not stored as part of their profile free text can be passed representing their needs

Required parameters api-key, country, state, city, user identity, needs text(optional)

```
find application record by api-key
find user by user identity
find application location by country, state and city
if user needs
        split user needs to userkeywords by delimiter
else
        process needs free text using processor to userkeywords
for keyword in userkeywords
        find keyword in locations keywords by keyword profile type 'bio'
        store list of users associated with keyword
                exclude self
create frequency map of list of users and number of times they appear in list
return sorted frequency map
```

Find Likes matches

Required parameters api-key, country, state, city, user identity

```
find application record by api-key
find user by user identity
find application location by country, state and city
split user likes to keywords by delimiter
for keyword in user likes keywords
        find keyword in locations keywords by keyword profile type 'likes'
        store list of users associated with keyword
                exclude self
```

```
create frequency map of list of users and number of times they appear in list
return sorted frequency map
```

Compare Users

This process is different to the processes above. Here we are trying to match two users and return common keywords in their profile. This process assumes that the keywords have been pre-processed and stored in the user object. Te exception is if we send in needs as free text. This free text will be processed to stemmed keywords before making a comparison.

The comparison matching uses a pattern matching algorithm to find matching words

Required Parameters api-key, user identity, compare user identity, profile type, needs free text(optional)

```
find application record by api-key
find user by user identity
find compare user by compare user identity
if profile type = 'bio'
        compare  user profile to compare user profile
else if  profile  type = 'needs'
        if needs free text
                process needs free text to stemmed keywords
                compare stemmed keywords to compare user profile
        else
                compare user needs to compare user profile
else if profile type = 'likes'
        compare user likes to compare user likes
return list of matching keywords
```

For a more comprehensive explanation of how these processes can be called via the apis developed in the Profiler API application see Appendix IV. This section outlines how an api call can be made to the server application and the parameters that need to be passed to enable it to function correctly. Any user wanting to use the apis must register for an api-key which is used for authentication on the server application.

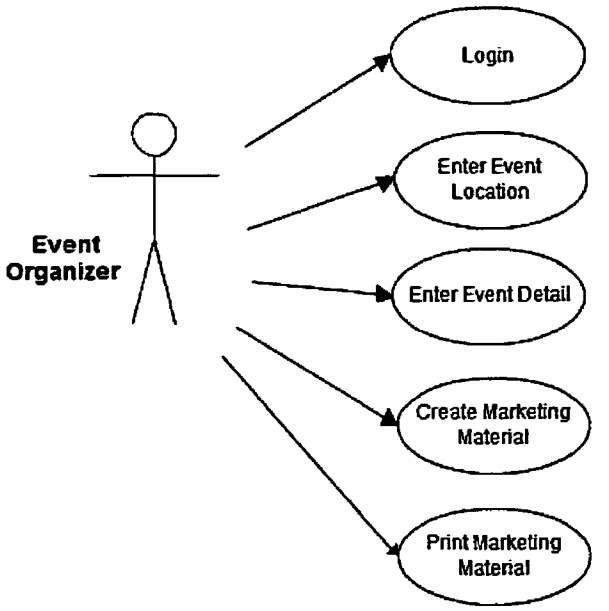
Geomingle Application

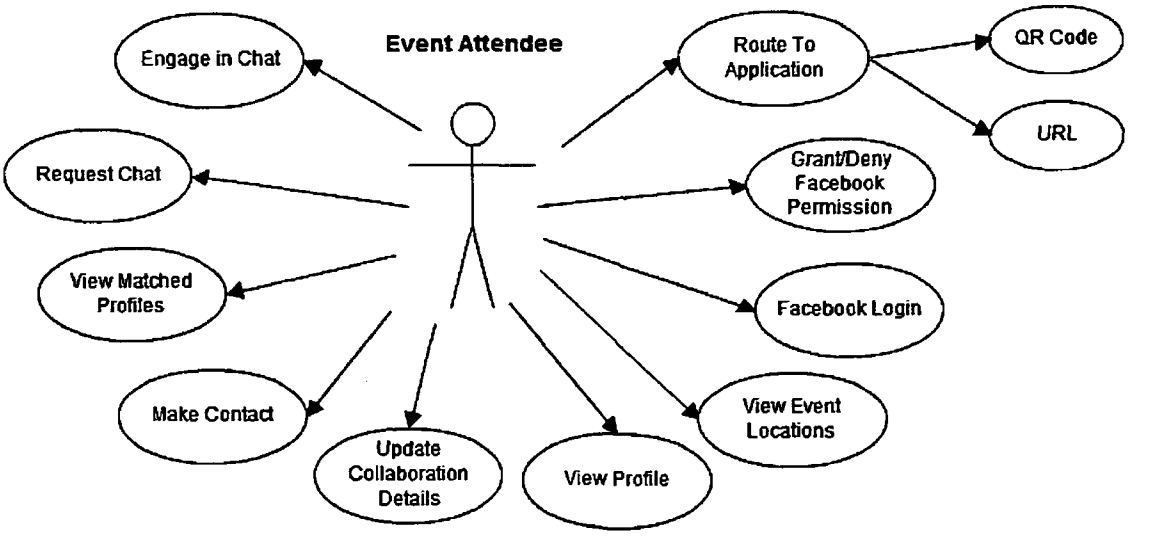
The mingle application is a social networking application associated with events. The application allows users to checkin to an event and it allows them to collaborate with other attendees at the event. The application uses Facebook as the authenticator to the application. The application uses the users profile data from Facebook to index a user within the application which in turn enables them to match with other users attending the event based on their profile similarity

The application communicates with the Profiler API application via well defined apis.

Use Case

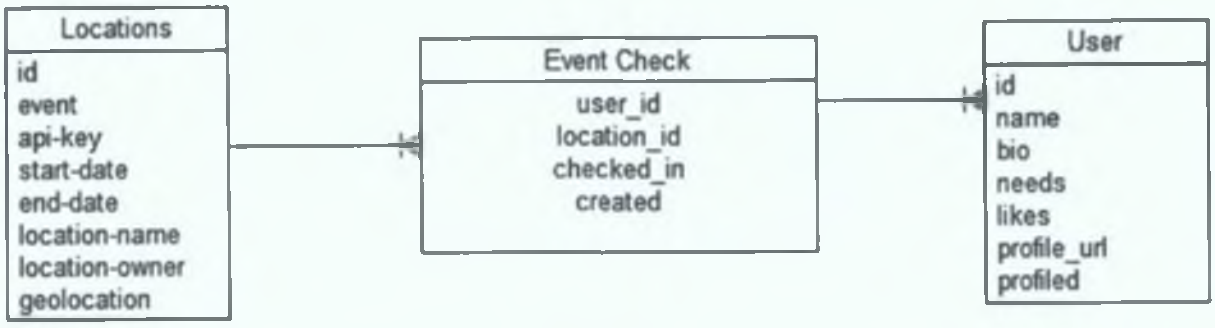
Actors : Event Organiser
Event Attendee





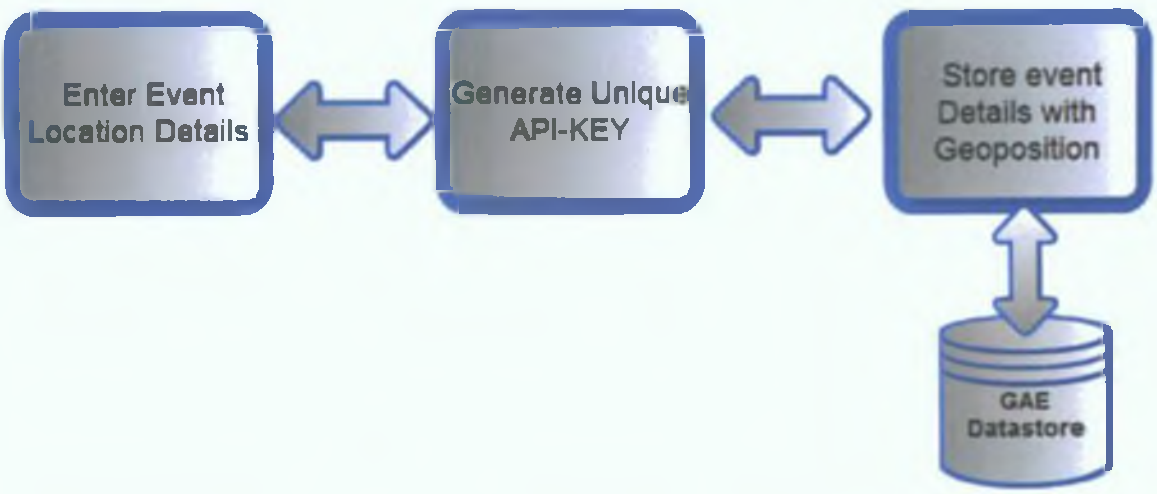
Object Model

Based on the requirements analysis and the social network profile analysis we determined the following object model.

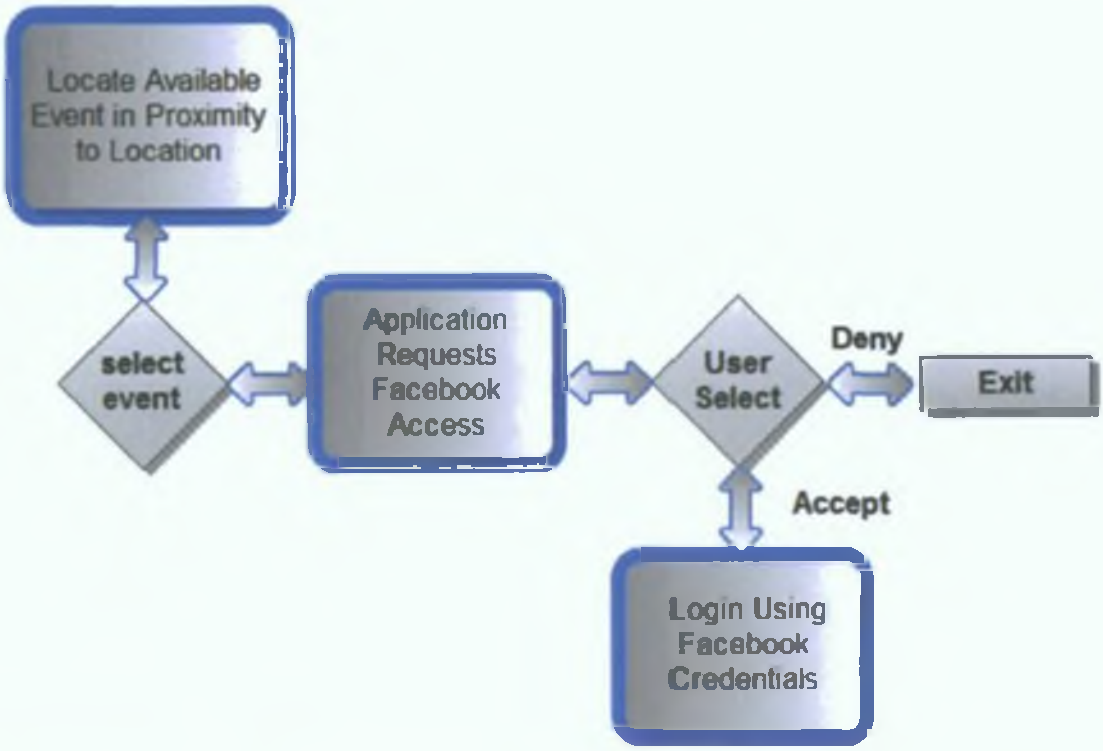


Process Flow

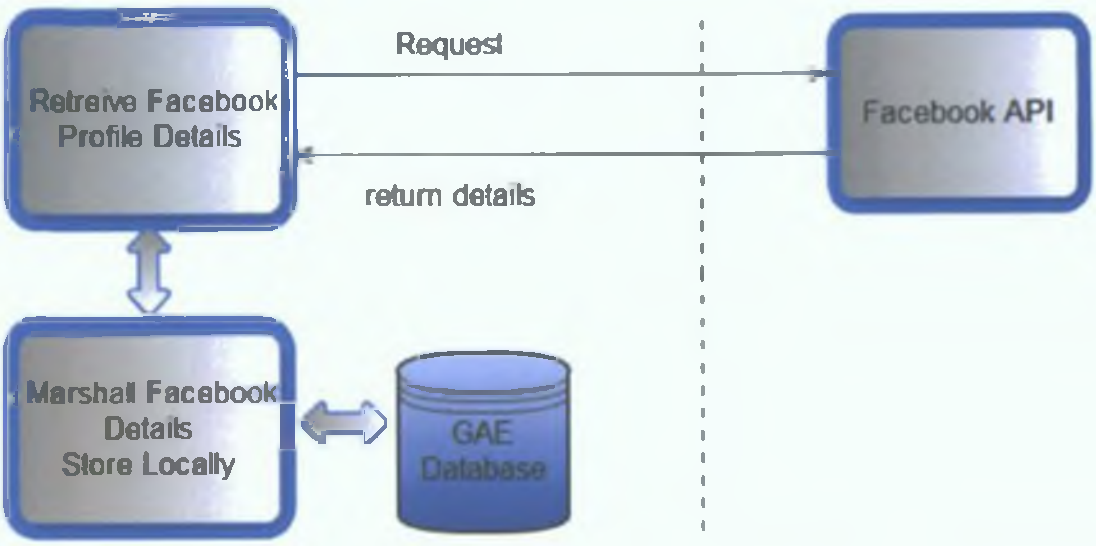
The first stage in the event based social network is to enter a new event. This requires the event organizer to enter the event details and correctly determine the geo location of the event using a google map. Once this is entered the event is stored with the geolocation and all event details. Behind the scenes a unique api-key is generated to comply with the requirements of the Profiler API application. This key is hidden from any users but is used in all api calls to the Profiler API application.



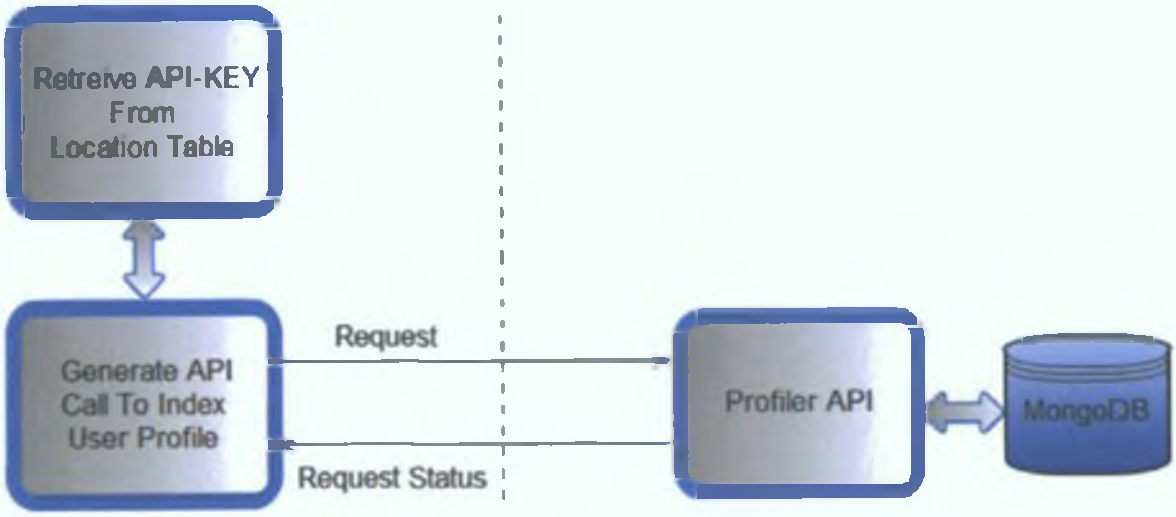
Once an event has been created and stored it is available to event attendees who are within a fixed radius of the events geoposition. The event attendees once they have accessed the application on their smartphone will go through the following procedures to gain access to the system.



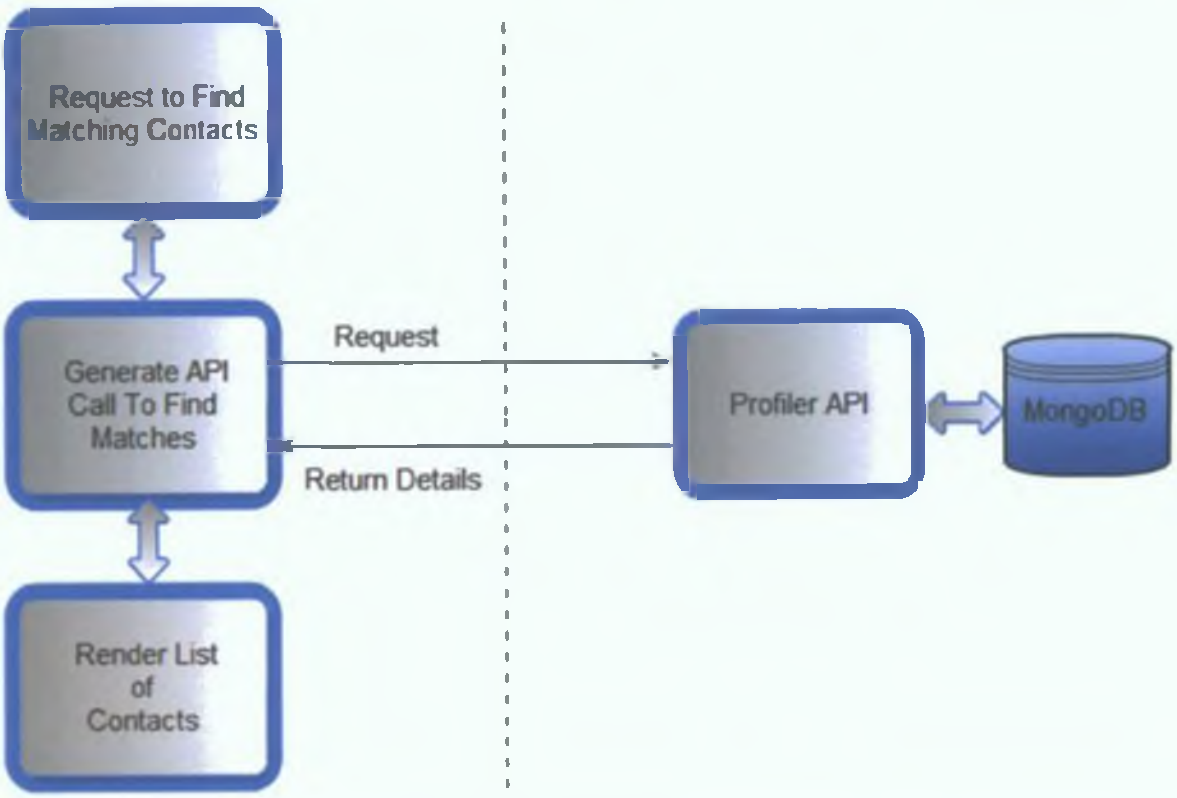
Once a user has successfully login to the system the application will attempt to retrieve to users Facebook profile details and store these details to the local database.



On successful storage of details to local database the application will then send these details to the Profiler API application for indexing for storage within its database.



The User is now in a position to find profile matches with the application. These matches are returned via api calls to the Profiler API application



From the returned list of contacts the user can view the profile of the matched contact and initiate a dialog if they wish.

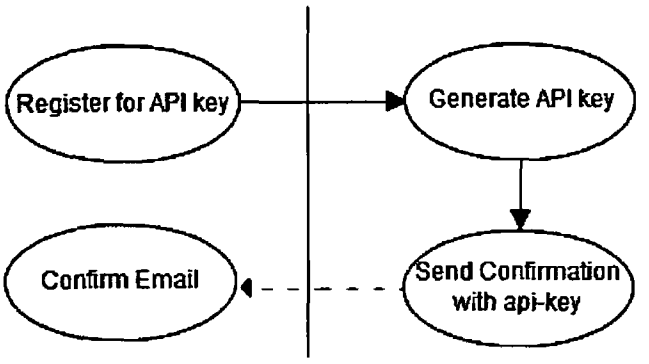
Appendix VI has a comprehensive outline of the operation of the application from a user perspective including screen shots of the application interface.

Security

Security is an important part of any application. This is particularly true of our application as we are potentially handling sensitive information. We have designed the applications in such a way to minimise the possibility of data falling into the wrong hand or being compromised in any way.

Profiler API

In order to use the Profiler API application intended users need to register for an API key. This key is required in any api calls made to the Profiler API application. This api-key is authenticated on the server side to ensure only registered users can access the system.



This api-key will also be used to determine the level of usage the client user may employ. Depending on their membership level they may be restricted to how many profiles they may index.

Upon registration users are required to give a user name and password, which, once user has confirmed their registration will enable them to access the online system. The password is stored on our database using the 'SHA' hashing algorithm.

The application intentionally does not hold any sensitive user information. The system only holds keywords which are associated with users via an identifier. This identifier is provided by the client application. If they wish this can be encrypted in any way they wish, as long as they are consistent in their encrypting they should have no problem indexing and retrieving data from the system.

GeoMingle Social Network

The geomngle social network application has two components. The first is the event registration portion, the second is the web based application developed for use on a smartphone platform.

The event registration portion uses Google App Engine authorization. This requires the user to have a gmail account to gain access to the system.

The web based portion of the application uses profile information from Facebook as the repository for a user's profile. This process is outlined in the appendix section of the report (see appendix II). The procedure uses Facebook for authorisation and authentication. The user must opt-in to allow our system access their Facebook profile data.

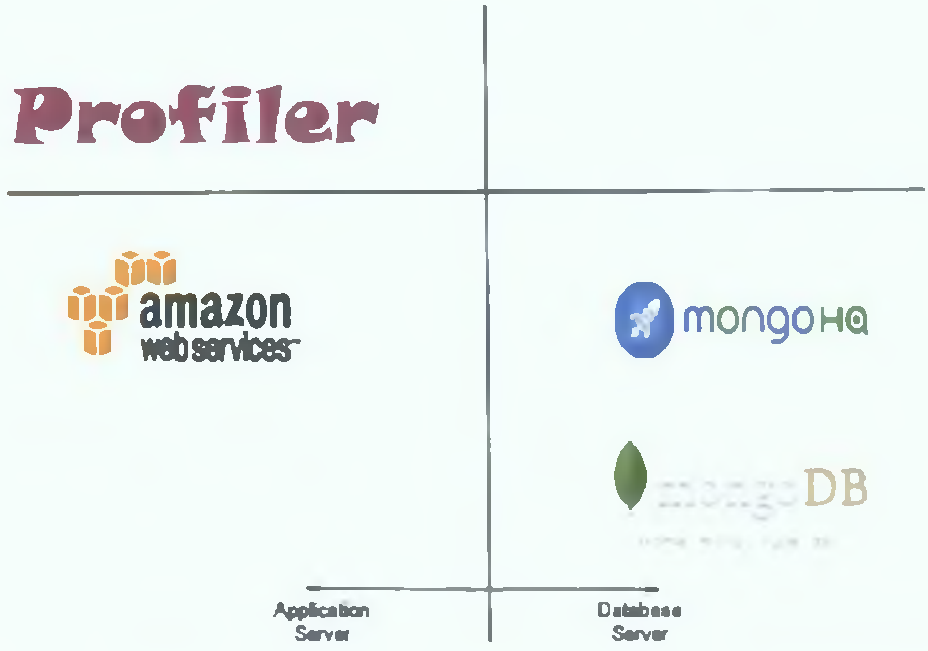
We do store a user's profile data on our local database. It will only reside on our database for the duration of the event they are attending. We have setup a cron task that is scheduled to run every evening and will delete all user information. We have no intention of using the user's profile

information for any purpose other than within our own application. This information will not be stored for later use or for generating an email list

Deployment

The Profiler API application resides on an Amazon Elasticbeanstalk application server. It resides on the url http://profiler.elasticbeanstalk.com

The database is a MongoDB database hosted by platform as a service provider MongoHQ



The geomingle social networking application resides on the Google App engine platform hosted et http://geomingle.appspot.com

Testing and Evaluation

This section of the report outlines the testing done on both applications.

Unit Testing

Throughout the project we did extensive unit testing. For the Profiler API application unit testing is easily facilitated by the Grails framework. All domain classes, views and controllers have unit test stubs generated by the framework automatically. The net beans IDE allows easy running of the unit tests and generates html reports of all tests run. An example a report follows.

An example of the code run to generate these tests follows.

```
class ParserTests extends GrailsUnitTestCase {
    void testParseAtSign() {
         // strip @ from text
        String s = "This is a test to see if @ is stripped";
        String expResult = "this is a test to see if is stripped";
        SStack result = Parser.StripSearch(s);
        assertEquals(expResult, result.toString());
    }
     void testParsePlus() {
        // want to loose + sign not only if c++
        String s = "This is a test to see if + is stripped";
        String expResult = "this is a test to see if is stripped";
        SStack result = Parser.StripSearch(s);
        assertEquals(expResult, result.toString());
     }
     void testParseCPlus() {
        // want to keep + sign only if c++
        String s = "This is a test to see if c++ is stripped";
        String expResult = "this is a test to see if c++ is stripped";
        SStack result = Parser.StripSearch(s);
        assertEquals(expResult, result.toString());
     }
```

```
void testParseApostrope() {
    // strip ' from text
    String s = "This is a test to see I'm is stripped";
    String expResult = "this is a test to see im is stripped";
    SStack result = Parser.StripSearch(s);
    assertEquals(expResult, result.toString());
}
void testParseCommandApostrope() {
    // strip ' and , from text
    String s = "This is a test to see I'm, is stripped";
    String expResult = "this is a test to see im is stripped";
    SStack result = Parser.StripSearch(s);
    assertEquals(expResult, result.toString());
}

void testCSharp() {
     // test c# remains from text
    String s = "This is a test to see if c# is not stripped";
    String expResult = "this is a test to see if c# is not stripped";
    SStack result = Parser.StripSearch(s);
    println result.toString();
    assertEquals(expResult, result.toString());
}
void testDotNet() {
     // test .net remains from text
    String s = "This is a test to see if .net is not stripped";
    String expResult = "this is a test to see if .net is not stripped";
    SStack result = Parser.StripSearch(s);
    println result.toString();
    assertEquals(expResult, result.toString());
}
void testDeveloperStemmed() {
     //test developer group of words correctly stemmed
    String stopped = "developer";
    SStack stemmed = Stemmer.StemText(stopped);
    assertEquals("develop",stemmed.toString());
    stopped = "development";
    stemmed = Stemmer.StemText(stopped);
    assertEquals("develop",stemmed.toString());
}
void testGenrateStemmed() {
     //test gererate group of words correctly stemmed
    String stopped = "generate";
    SStack stemmed = Stemmer.StemText(stopped);
    assertEquals("generat",stemmed.toString())
    stopped = "generation";
    stemmed = Stemmer.StemText(stopped);
    assertEquals("generat",stemmed.toString());
    stopped = "generated";
    stemmed = Stemmer.StemText(stopped);
    assertEquals("generat",stemmed.toString());
}
  void testAnimateStemmed() {
    //test animate group of words correctly stemmed
    String stopped = "animation";
    SStack stemmed = Stemmer.StemText(stopped);
    assertEquals("animat",stemmed.toString())
    stopped = "animator";
    stemmed = Stemmer.StemText(stopped);
    assertEquals("animat",stemmed.toString());
    stopped = "animate";
    stemmed = Stemmer.StemText(stopped);
    assertEquals("animat",stemmed.toString());
```

User Testing

At the time of writing this document the geomingle application was not considered ready for extensive user testing and evaluation. But we can outline the plans we have for conducting this testing. We plan on doing both an Heuristic Evaluation and extensive user testing in a controller environment.

Heuristic Usability Evaluation

We plan on asking an experienced web developer to conduct a heuristic evaluation of the system for us. This evaluation will be based on the principles set down in Jakob Nielsen's Ten Usability Heuristics [22] as a guide line.

The guidelines can be summarized as follows.

1. Visibility of system status (Feedback)

   The system should always keep users informed about what is going on, through appropriate feedback within reasonable time. A feedback message is displayed when an action is performed

2. Match between system and the real world

   The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

3. User control and freedom (NAVIGATION)

   Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Supports undo and redo and a clear way to navigate. Clearly marks where the person is and where they can go by showing the selection in each menu

4. Consistency and standards (CONSISTENCY)

   Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.

5. Error prevention (PREVENTION)

   Even better than good error messages is a careful design, which prevents a problem from occurring in the first place. Minimise user input. Use defaults where possible.Disable the update button after it is clicked, so the person cannot update the post twice by accident. Auto focus on input prevents a common source of frustration, typing only to realize nothing is displayed because the field did not have focus

6. Recognition rather than recall (MEMORY)

   Minimize the user's memory load. Make objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

7. Flexibility and efficiency of use (EFFICIENCY)

   Accelerators — unseen by the novice user — may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

8. Aesthetic and minimalist design (DESIGN)

   Dialogues should not contain information, which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility. Visual layout should respect the principles of contrast, repetition, alignment, and proximity.

9. Help users recognize, diagnose, and recover from errors (RECOVERY)

   Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution. Immediate feedback with specific instructions

10. Help and documentation (Help)

    Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

We plan on instigating this analysis as soon as the system has been effectively unit and system tested.

User Testing

We plan on conducting user testing on the application within a controlled environment so that users can be observed in their use of the system and to get their feedback on their impressions of the system.

We plan on using reports to get feedback from the users on completion of the user testing. An example of the proposed feedback reports is given in Appendix II.

The user testing will consist of a set of 10 users. Each will be given a sheet outlining their activities. They will be asked to.

Ensure that they can access their Facebook profile

They will be assigned keywords they will be asked to enter into their Facebook profile

Groups of users will be separated into separate rooms.

They will be given an indication of the other members of the controlled group that they should be able to connect to and via what criteria

They will be asked to attempt to contact a contact via the "request a chat" feature.

If request successful they will be asked to engage in a dialog with the contact.

Future Work

This section of the report highlights areas where the project could be extended and improved upon. It also highlights work that needs to be done to complete the applications.

Extending the Algorithm

In this section we discuss some approaches to improving the algorithm and increasing it likely hood of retrieving more efficient matches.

Bi-gramming

Some Information Retrieval systems also use multi-word phrases as index terms. A phrase match is considered more informative than individual words. One of the drawbacks of our approach so far is that we use single words in our indexing and matching process (for free text such as the users bio). This approach has its limitations as we are not taking into account the context of the word within a paragraph of text. For example we could have two profiles as follows

Profile 1 - "I am a surgical consultant"
Profile 2 - "I am a management consultant"

In our approach these two profiles would match on the single word *consultant,* but clearly these two profiles have little in common. To improve on our approach we can consider the use of bi-grams, where a bi-gram is a pair of consequent words. In our example above, after removing stop words, we are left with the following bi-gram word phrases

Profile 1 - "surgical consultant"
Profile 2 "management consultant"

This data is more powerful then the use of single words in that we can match users on more relevant terms. We can also give these bi-grams more weight than single keyword phrases to improve the matching capabilities of our algorithm.

The downside of implementing the use of bi-grams is that it increases the amount of storage required to store our inverted linked list. As we increase the number of keyword phrases we will increase the number of keywords to index upon.

An approach to implementing this would be to iterate through the text in a profile and instead of keying on single words we could include consecutive words as keyword phrases. We could also improve on this approach by ignoring words that have a comma or full stop after the word. For example the profile

"I am a web developer programming in php, java, python"

Would produce the following keywords

web
develop
program
php
java
python

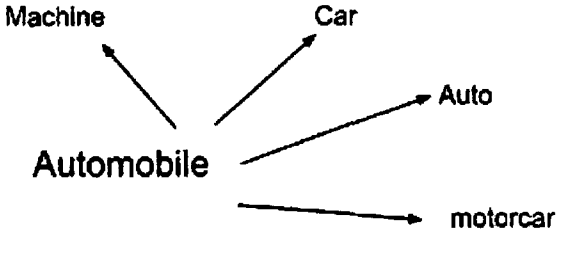
In an enhanced algorithm we would produce the following keyword phrases in addition to the single word keywords

web develop
develop program
program php

Clearly this introduces some "noise" i.e. irrelevant terms, but does include the phrase *web develop* which could be considered useful in creating a relevant keyword phrase.

Thesauri

At present the application relies on the exact matching of indexed keywords. The application resolves certain words to their root stem so that words like animator and animate will resolve to the same stem i.e. *animat* , Therefore profiles with these words in them will match.
What if we have words that have the same meaning, but are unrelated linguistically. For example the words Automobile and Car clearly relate to the same physical thing. At present our system is not sophisticated enough to relate these two words. One way to overcome this limitation is to use thesauri which are dictionary based systems for resolving words and their synonyms and antonyms. For example the word Automobile can viewed visually



Using this approach we could substitute words that have synonyms and antonyms for the word within a text and include these in the indexing process. This would add a degree of complexity to the system and potentially impact on the responsiveness of the application.

Connections

At present the system retrieves connections in real time. This may prove impractical for large scale social networks as the response time may prove impractical. To solve this issue we could provide the ability to store connections in a separate database table. This could be created via overnight batch processing so that connections are already formed when queries are made. We could also ensure that these connections are ranked by relevance for fast retrieval.

Heuristic Refactoring

The algorithm we have created is essentially an heuristic, where we converge on an optimal solution over time. We see the algorithm undergoing constant evaluation and improvement over an extended time period. To put into context the challenge of creating a world class algorithm for profile matching the following quotes from an Irish Times article (Friday, July 15, 2011) by the lead engineer in Google's search projects, Ben Gomes. He puts the challenge into context.

"More than 20,000 potential changes are evaluated by Google annually. Of these, about 6,000 go into live experiments on the site, with engineers closely watching the results. About 50 to 200 experiments might be ongoing at any one time, Gomes says. During the course of the year, more than 500 changes are made of some sort to the search algorithm."

Ranking

At present we have a crude approach to ranking contacts for relevance. We return the frequency of the number of users who match on a keyword and sort by descending order. Improvements in this area include investigation of Vector Space Models and Probabilistic Models. We will do this in conjunction with the bi-gram analysis where term are given higher relevance and ranked accordingly.

Refactoring

One area we would like to address in future is the refactoring of the algorithm we used in the stemming algorithm. We used an open source version of the Porter stemming algorithm. This has not been written in a way that makes it easy to amend and update. We would like to redesign it to use object oriented techniques and design patterns. This would make it easier to effect changes and improvements in future iteration of the algorithm.

Profile Sourcing

Our event based social network sources profile information from Facebook. It should be possible to allow users to choose which social network they wish to source their profile information from. We propose to add the functionality to allow users to choose between Facebook and Linkedin at the sign-on stage. We will also look at using Twitter details to enhance a users indexed profile.

Commercialisation

If it is decided to attempt to commercialise this application we propose to follow the model used by most internet Software as a Service providers. We intend to provide the application free of charge for limited access to the application and charge for unlimited access on a monthly basis with enhanced reporting capabilities. We would need to enhance the application to achieve this. We would need to provide enhanced reporting capabilities to allow client access their indexed user profile. A lot more work would need to be done to enable users to monitor their details via an online reporting system.

Benchmarking

One of the issues we encountered when testing the application was that our testing environment was not adequately setup to provide accurate benchmarking figures. Running the application locally on a desktop computer does not adequately reflect production conditions. On deploying the application to Amazons platform we saw a factorial improvement in performance. What was taking minutes to run on our testing environment took seconds when running on the Amazon platform. We realised this too late in the development process to adequately provide performance figures. We need to set up a parallel testing environment on another Amazon EC2 instance to replicate the anticipated performance of the production environment. This has cost

implications which we need to quantify in order to proceed with setting up this testing platform. As a result a lot of our benchmarking figures are not accurate and are not reported on here.

Database Optimization

We decided at the outset to use a noSql database. This decision was based largely on the ability of these databases to handle large amounts of data and to scale with the requirements of the application as it grows. We also were keen to take advantage of functionality such as automatic sharding. Sharding is the partitioning of data among multiple machines in an order-preserving manner. This feature allows systems to scale transparently as the demands on the database grow. This was deemed an important feature for an application that potential could hold large amounts of data.

We don't have a lot of experience of optimizing the database we choose, MongoDB. Over the next few iterations of the project we plan on attempting to optimise the database to improve performance. This will include the addition of suitable indexes and implementing performance monitoring tools.

Database Backup and Recovery

If the application we have created is to have the potential to be commercialised we need to ensure that the database is adequately backed up to ensure seamless recovery for any potential outages.

Conclusions

In our project we produced a relatively unsophisticated algorithm for indexing of user profiles within a social network. A lot of the research we did in the initial stages of the research phase of the project highlighted issues which we felt were not relevant at this stage of the projects evolution. For instance, early pioneers in the field of information retrieval such as Luhn [17] highlight issues such as, emphasising the relevance of the positioning of words within sentences and giving these words more significance. This research is largely related to the indexing of documents. As we are only indexing relatively small portions of text we did not consider adding this level of complexity to our algorithm at this stage.

Our research shows that none of the major social networks have the functionality that we have developed in this project. It is possible we have identified a gap in the market which could enhance a social networks functionality and increase its desirability to potential members.

It is possible that providing the kind of functionality we propose may impinge on the major social networks business model. They make the majority of their money via ads targeted at specific users. Providing the kind of functionality we propose may break this model, as it would potentially allow, for example, recruiters to find members with specific skills in specific locations. To overcome this, the functionality could be provided as a premium level service where users who want this kind of functionality would pay for the ability to search the network for specific users. Users could opt-in and opt-out of the ability to be found by others within the network. For example, people who are looking for jobs could allow their profiles to be found, those who are not actively looking for jobs could opt-out of being searched for. Privacy could be an issue in this scenario. To overcome this it may be possible to restrict the information that is displayed to searchers e.g. the name of the individual and their employer would not be displayed, but the searcher could contact the user in some way and rely on the individual to decide if they want to enhance the level of communication between the two.

References

[1] Bear Bibeault, 2008. *jQuery in Action*. 1 Edition. Manning Publications.

[2] William B. Frakes, 1992. *Information Retrieval: Data Structures and Algorithms*. 1 Edition. Prentice Hall.

[3] Bruce Lawson, 2010. *Introducing HTML5 (Voices That Matter)*. 1 Edition. New Riders Press.

[4] Christopher D. Manning, 2008. *Introduction to Information Retrieval*. 1 Edition. Cambridge University Press.

[5] Tom Negrino, 2008. *JavaScript and Ajax for the Web: Visual QuickStart Guide (7th Edition)*. Peachpit Press.

[6] Jothy Rosenberg, 2010. *The Cloud at Your Service*. Pap/Psc Edition. Manning Publications.

[7] Matthew A. Russell, 2011. *Mining the Social Web: Analyzing Data from Facebook, Twitter, LinkedIn, and Other Social Media Sites*. 1 Edition. O'Reilly Media

[8] Dan Sanderson, 2009. *Programming Google App Engine: Build and Run Scalable Web Apps on Google's Infrastructure (Animal Guide)*. 1 Edition. O'Reilly Media.

[9] Toby Segaran, 2007. *Programming Collective Intelligence: Building Smart Web 2.0 Applications*. 1st Ed. Edition. O'Reilly Media.

[10] Jonathan Stark, 2010. *Building iPhone Apps with HTML, CSS, and JavaScript: Making App Store Apps Without Objective-C or Cocoa*. 1 Edition. O'Reilly Media.

[11] Ricardo Baeza-Yates, 1999. *Modern Information Retrieval*. 1st Edition. Addison Wesley.

[12] Authentication - Facebook developers. 2011. *Authentication - Facebook developers*. [ONLINE] Available at: http://developers.facebook.com/docs/authentication/. [Accessed 13 July 2011].

[13] 10gen & MongoDB. 2011. *10gen & MongoDB*. [ONLINE] Available at: http://www.mongodb.com. [Accessed 1 August 2011].

[14] MongoHQ . 2011. *MongoHQ - The cloud-based hosted database solution for MongoDB.*. [ONLINE] Available at: http://www.mongohq.com. [Accessed 2 August 2011].

[15] David Hull. Stemming algorithms - a case study for detailed evaluation. *Journal of the American Society for Information Science*, 47(1):70–84, 1996

[16] Krovetz, R. (1993). Viewing morphology as an inference process. In Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 191-202.

[17] LUHN, H.P., 'The automatic creation of literature abstracts', *IBM Journal of Research and Development*, 2, 159-165 (1958).

[18] M.F. Porter, 1980, An algorithm for suffix stripping, *Program*, **14**(3) pp 130–137.

[19] C.J. van Rijsbergen, S.E. Robertson and M.F. Porter, 1980. *New models in probabilistic information retrieval*. London: British Library. (British Library Research and Development Report, no. 5587).

[20] Amit Singhal Google, Inc. 2011. *Modern Information Retrieval: A Brief Overview* [ONLINE] Available at: http://pages.cs.wisc.edu/~anhai/courses/784-sp08-anhai/ir_overview.pdf. [Accessed 1 September 2011].

[21] Bush, Vannevar. "As We May Think." *The Atlantic Monthly*. July 1945.

[22] 10 Heuristics for User Interface Design. 2011. *10 Heuristics for User Interface Design*. [ONLINE] Available at: http://www.useit.com/papers/heuristic/heuristic_list.html. [Accessed 02 September 2011].

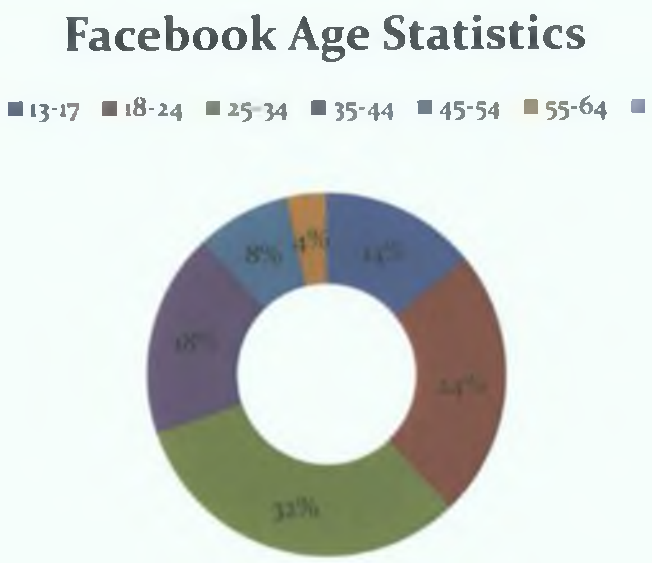
APPENDIX I

Facebook Statistics

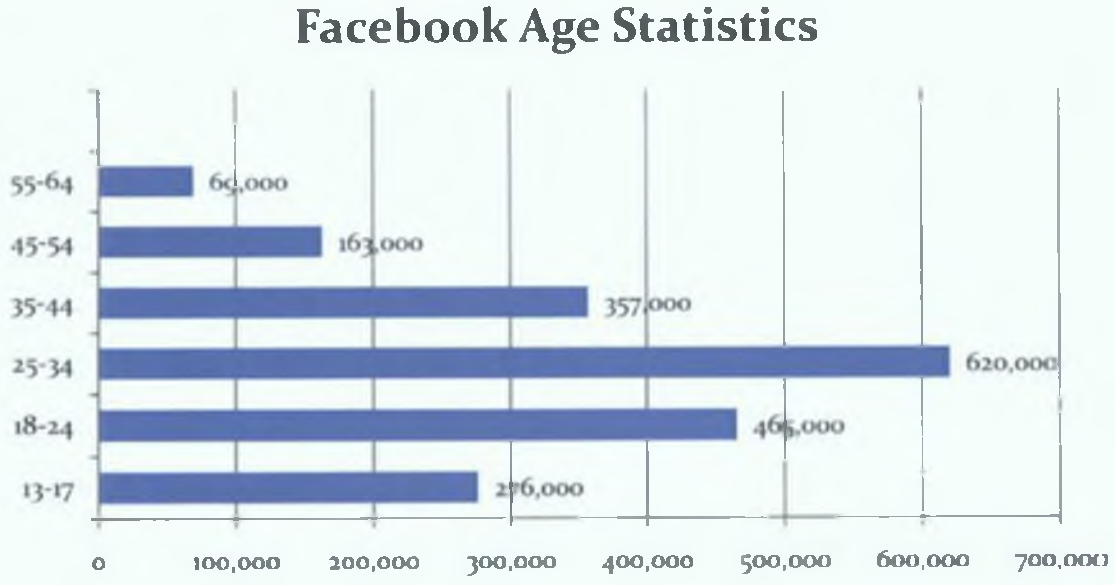
Facebook is the world's most popular social network. At the time of writing this report(31 Aug 2011) there are

750million	globally registered users	
2,025,280	users on Facebook Ireland	
927,700	have stated their gender as male	45.8%
1,058,580	have stated their gender as female	52.26%
39,000	have no gender stated	1.92%

The following chart shows Facebook usage in Ireland, by age group.



The next chart shows the number of users by age grouping.



These figures can be verified by using the Facebook ad targeting tool (as of 31 Aug. 2011)

N.B. select "exact age match" to get accurate figures otherwise data will be distorted.

The breakdown of registered users by city shows that Dublin is by far the biggest user of Facebook, which isn't surprising given the population distribution of Ireland

Dublin	-	752,040
Cork	-	145,920
Limerick	-	79,040
Galway	-	71,420

Some global statistics show:

- 50% of active users log on to Facebook in any given day
- The average user has 130 friends
- Entrepreneurs and developers from more than 190 countries build with Facebook Platform
- People on Facebook install 20 million applications every day
- Every month, more than 250 million people engage with Facebook on external websites
- Since social plug-ins launched in April 2010, an average of 10,000 new websites integrate with Facebook every day
- More than 2.5 million websites have integrated with Facebook, including over 80 of comScore's U.S. Top 100 websites and over half of comScore's Global Top 100 websites.
- There are more than 250 million active users currently accessing Facebook through their mobile devices.
- People that use Facebook on their mobile devices are twice as active on Facebook than non-mobile users.

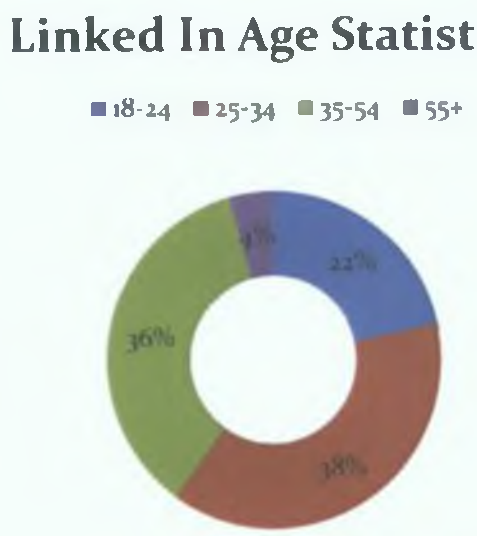
Source: http://www.facebook.com/press/info.php?statistics

Linked In Statistics

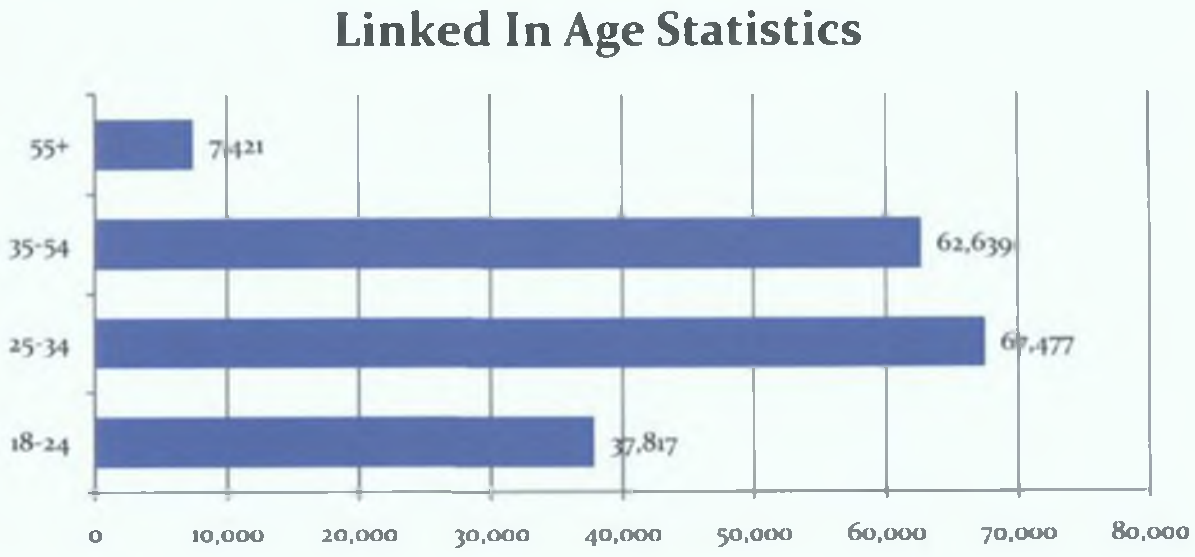
At the time of writing this report there are

536,108	users subscribed to Linked In Ireland	
202,541	have stated their gender as female	37.8%
286,270	have stated their gender as male	53.4%
47,297	have not stated their gender	8.8%

Source: Linked In ad campaign tool (31 Aug 2011)



The next chart shows the number of users by age grouping.



The figures can be verified via the Linked In ad campaign tool

The Linked In Ad campaign tool doesn't allow for breakdown by city within a country such as Ireland, so we can't get the same level of statistics about usage as in Facebook.

APPENDIX Ia

Smartphone Usage

Smartphone penetration in Ireland is currently at 37%, with strong adoption among males (45%) compared with females (29%), according to new research from Amárach. Smartphones are most popular among 25 to 34-year-olds, where the penetration level is 53%. On average, a smartphone owner downloads 4.8 apps per month and, as the table shows, social networking apps are the most used.

Amárach is predicting that the acceleration in smartphone use will stimulate demand for mobile commerce in Ireland - and forecasts that €800 million worth of transactions will be conducted through mobile devices in 2012. Already mobile commerce has taken off internationally, and often for some unlikely product categories

http://www.iia.ie/resources/resource/1/state-of-the-net/ State of the Net issue 21 (01.06.2011)

APPENDIX II

OAuth Explained

In this project we used OAuth to allow users gain access to their Facebook profile information via our social network application *geomingle*. The reason for doing this is that OAuth provides the ability to:

- Avoid a user setting up a new username/password pair
- Allows our applications to pull profile information from the service provider and use it within our application.

Before understanding the OAuth workflow we will define the following terms.

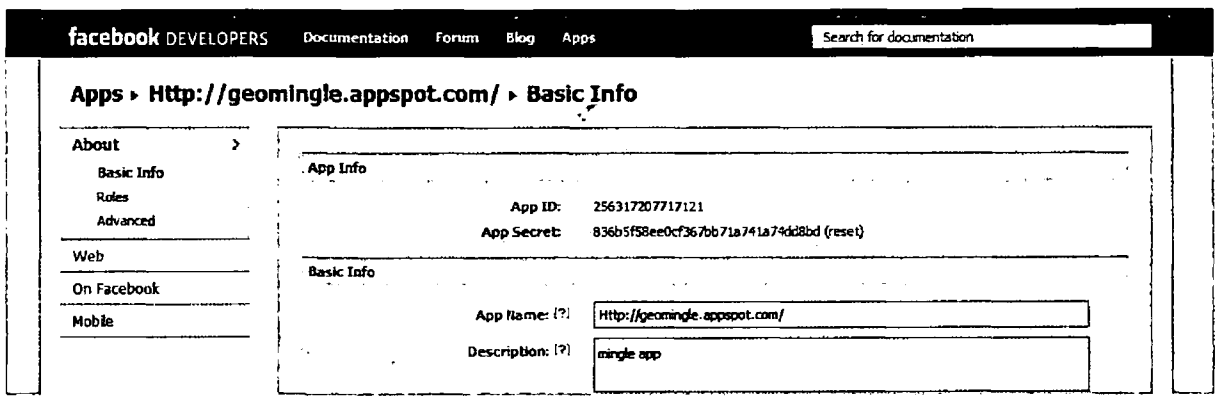
Service Provider: A web application, that allows access via OAuth. In our discussion we are using *Facebook* as the service provider

User: An individual who has an account with the Service Provider.

Consumer: A website or application that uses OAuth to access the Service Provider on behalf of the User. This is our *geomingle* application.

Registration

The consumer (*geomingle*) needs to register their application with the service provider (*facebook*) providing OAuth access. This process results in the consumer being provided with a number of keys, an application id and a secret key. See figure below for an example for our *geomingle* application



Authorization and Authentication

User authentication and application authorization are handled at the same time by redirecting the user to the Facebook OAuth dialog When invoking this dialog, we pass in our **app id** that is

generated in the above process and the URL that the user will be redirected back to once application authorization is completed (the **redirect Uri** parameter).

```
https://www.facebook.com/dialog/oauth?
        client_id=YOUR_APP_ID&redirect_uri=YOUR_URL
```

in our case the **redirect_uri** will look like http://geomingle.appspot.com/auth/login

Within our application this will route to a controller method based on our routing handler.

The next step is that the user is asked to enter their credentials.

If this is the first time the user has accessed the application they will be prompted to grant the third party application (*geomingle*) authorization.

The OAuth Dialog will redirect the user's browser to the URL we passed in the **redirect_uri** parameter with an **authorization code** provided by the service provider.

```
http://geomingle.appspot.com/auth/login?code=A_CODE_GENERATED_BY_SERVE
R
```

Within our application we then can access the users profile using this authorization code.

In order to authenticate our app, we must pass the authorization code and our **app secret** to the Graph API **token endpoint** at https://graph.facebook.com/oauth/access_token.

```
https://graph.facebook.com/oauth/access_token?
        client_id=YOUR_APP_ID&redirect_uri=YOUR_URL&
        client_secret=YOUR_APP_SECRET&code=THE_CODE_FROM_ABOVE
```

If our app is successfully authenticated and the authorization code from the user is valid, the authorization server will return an **access token**:

With a valid **access token** we can invoke the Graph API by appending the **access_token** parameter to Graph API requests:

```
https://graph.facebook.com/me?access_token=ACCESS_TOKEN
```

For example in our application we return the following data from the user's profile

{'bio': 'I am a web developer programming in python, php, ruby rails, c# . I develop applications using the Google App Engine. Also used frameworks such as .net, rails, django.',

'first_name': 'Sean',

'last_name': 'Cahill',

'verified': True,

'name': 'Sean Cahill',

'locale': 'en_US',

'gender': 'male',

'email': 'sean@nmm.ie',

'link': 'http://www.facebook.com/profile.php?id=1749478181',

'timezone': 1,

'updated_time': '2011-07-20T09:41:14+0000',

'id': '1749478181'}

Additional calls to the Facebook api are required to obtain more profile information.

For example to return a users interests and activities we make calls such as (code is based on python code)

```
books = json.load(urllib2.urlopen("https://graph.facebook.com/me/books?" +
                        urllib.urlencode(dict(access_token=access_token))))
tv = json.load(urllib2.urlopen("https://graph.facebook.com/me/television?" +
```

```
                        urllib.urlencode(dict(access_token=access_token)))
movies = json.load(urllib2.urlopen("https://graph.facebook.com/me/movies?" +
                        urllib.urlencode(dict(access_token=access_token))))
music = json.load(urllib2.urlopen("https://graph.facebook.com/me/music?" +
                        urllib.urlencode(dict(access_token=access_token)))
```

APPENDIX III

Thank you for completing this phase of the testing process. Please fill in the attached questionnaire to record your experiences of using the system.

Post Test Questionnaire

How useful is this application ?

Please choose the appropriate response for each item

	strongly disagree				*strongly agree*
This web site meets my needs	☐	☐	☐	☐	☐
It does everything I would expect It to do	☐	☐	☐	☐	☐
The web site allows me to do the Things I want to do	☐	☐	☐	☐	☐

How easy was this web site to use ?

Please choose the appropriate response for each item

	strongly disagree				*strongly agree*
This web site meets my needs	☐	☐	☐	☐	☐
It does everything I would expect It to do	☐	☐	☐	☐	☐
The web site allows me to do the Things I want to do	☐	☐	☐	☐	☐

Post Test Questionnaire

Feedback

What did you like most about this web site ?

Please write your answer

What did you dislike most about this web site ?

Please write your answer

How could this web site be improved ?

Please write your answer

Where you able to find the event on the home page of the application
Yes / No

Where you able to find the login using your Facebook id and password
Yes / No

If Yes does the profile presented accurately reflect your Facebook profile
Yes / No

Where you able to find the users you were assigned as possible matches in your work sheet

Yes / No

If Yes above here you able to initiate a dialog with one of the users assigned as possible matches in your work sheet

Yes / No

APPENDIX IV

API Usage

The following is a list of the available apis accessible by registered users with valid api-keys using the Profiler API application

Index a User

This api will take a user and index it based on the text passed in the api call. The text will be reduced to keywords which will then be indexed and store on our database. This indexed profile can then be queried to find matches with other users.

URL: http://<hostname>/api/profile

Method: POST

Parameters

country:the country location of user (optional)
state: the state location of user (optional)
town: the town location of user (optional)
name: the name of the client application (required)
api: the client api key (required)
user: unique identifier identifying user (required)
profile: text based description of users profile (required)
type: profile type
Options: bio
needs
likes

Example:

The following is an example of usage implemented in python

```
from urllib import urlencode
import httplib2

httplib2.debuglevel = 1

data = { "country":"Ireland","state":"Dublin","town":"dalkey","name": "smallbusiness",
"api": "1234-5678-9012-3456","user":"123450","profile": "i am a web developer with
experience in php,python,java.","type": "bio"}

h = httplib2.Http('.cache')

resp, content =
h.request('http://localhost:8080/socnet/api/create','POST',urlencode(data),
      headers={'Content-Type': 'application/x-www-form-urlencoded'})

print(content.decode('utf-8'))
```

Return a Users Matches

This api call takes a user, identified by its unique identifier, and will proceed to find the most relevant matches based on the users profile from the pool of other users within the networks pool of users. The api call with return the most relevant matches ordered by the closest matches, descending.

URL: http://<hostname>/api/match

Method: POST

Parameters

country:	the country location of user	(optional)
state:	the state location of user	(optional)
town:	the town location of user	(optional)
name:	the name of the client application	(required)
api:	the client api key	(required)
user:	unique identifier identifying user	(required)
type:	profile type	(required)
	Options: bio	
	Needs	
	Skills	
	Likes	

Example code

```
from urllib import urlencode
import httplib2
httplib2.debuglevel = 1

data = { "country":"Ireland","state":"Dublin","Town":"dalkey","name": "demo", "api": "92a68c27-
d1b3-4cc2-ac86-e477bcdcf239","user":"123456","type":"bio"}
h = httplib2.Http('.cache')

resp, content = h.request('http://localhost:8081/socnet/api/match','POST',urlencode(data),
      headers={'Content-Type': 'application/x-www-form-urlencoded'})
print(content.decode('utf-8'))
```

The above request results in the following output. The api call returns a list of user contacts, identified by their unique identifier, along with the frequency of that user. The list is sorted by most popular matches, decreasing. The output is in json format.

```
connect: (localhost, 8081)
send: 'POST /socnet/api/match HTTP/1.1\r\nHost: localhost:8081\r\nContent-
Length: 63\r\ncontent-type: application/x-www-form-urlencoded\r\naccept-
encoding: gzip, deflate\r\nuser-agent: Python-httplib2/0.7.0
(gzip)\r\n\r\napi=1234-5678-9012-1111&type=bio&name=smallbusiness&user=123456'
reply: 'HTTP/1.1 200 OK\r\n'
header: Server: Apache-Coyote/1.1
header: Content-Type: application/json;charset=UTF-8
header: Transfer-Encoding: chunked
header: Date: Fri, 26 Aug 2011 16:51:43 GMT

{"123457":4,"123458":3}

Process finished with exit code 0
```

Delete a User

This api will remove a user from the index. The process finds all instances of the user and will re index based on the profile information.

URL: http://<hostname>/api/delete

Method: POST

Parameters

country:	the country location of user	(optional)
state:	the state location of user	(optional)
town:	the town location of user	(optional)
name:	the name of the client application	(required)
api:	the client api key	(required)
user:	unique identifier identifying user	(required)
type:	profile type	(required)
	Options: bio	
	needs	
	likes	

Example code

```
from urllib import urlencode
import httplib2
httplib2.debuglevel = 1

data = { "name": "smallbusiness", "api": "1234-5678-9012-1111","user":"123456",'type": "bio"}

h = httplib2.Http('.cache')

resp, content = h.request('http://localhost:8081/socnet/api/delete,'POST',urlencode(data),
      headers={'Content-Type': 'application/x-www-form-urlencoded'})
print(content.decode('utf-8'))
```

Produces the following output.

```
connect: (localhost, 8081)
send: 'POST /socnet/api/delete HTTP/1.1\r\nHost: localhost:8081\r\nContent-
Length: 63\r\ncontent-type: application/x-www-form-urlencoded\r\naccept-
encoding: gzip, deflate\r\nuser-agent: Python-httplib2/0.7.0
(gzip)\r\n\r\napi=1234-5678-9012-1111&type=bio&name=smallbusiness&user=123456'
reply: 'HTTP/1.1 200 OK\r\n'
header: Server: Apache-Coyote/1.1
header: Content-Type: text/html;charset=utf-8
header: Transfer-Encoding: chunked
header: Date: Fri, 26 Aug 2011 17:36:20 GMT

successfully deleted user 123456

Process finished with exit code 0
```

Compare Two Users

This api will return keywords that two users have in common based on the "type" passed. If the user's needs have not been indexed it is possible to send needs as a free text parameter

URL: http://<hostname>/api/compare

Method: POST

Parameters

name:	the name of the client application	(required)
api:	the client api key	(required)
user:	unique identifier identifying user	(required)
compareUser:	unique identifier identifying user to compare with	(required)
needs:	free text identifying user needs	(optional)
type:	profile type	(required)
Options:	bio	
	needs	
	likes	

If type of 'bio' user is compared to compareUser bio to bio
If type of 'needs' users needs are compared to compareUser bio
If type of 'likes' user is compared to compareUser likes to likes

```
from urllib import urlencode
import httplib2
httplib2.debuglevel = 1

data = { "name": "demo", "api": "92a68c27-d1b3-4cc2-ac86-
e477bcdcf239","user":"123456","compareUser":"123457","type":"bio"}

h = httplib2.Http('.cache')

resp, content = h.request('http://localhost:8080/socnet/api/compare','POST',urlencode(data),
      headers={'Content-Type': 'application/x-www-form-urlencoded'})
print(content.decode('utf-8'))
```

```
connect: (localhost, 8080)
send: 'POST /socnet/api/compare HTTP/1.1\r\nHost: localhost:8080\r\nContent-
Length: 73\r\ncontent-type: application/x-www-form-urlencoded\r\naccept-
encoding: gzip, deflate\r\nuser-agent: Python-httplib2/0.7.0
(gzip)\r\n\r\ncompareUser=123457&api=1234-5678-9012-
1111&name=smallbusiness&user=123456'
reply: 'HTTP/1.1 200 OK\r\n'
header: Server: Apache-Coyote/1.1
header: Content-Type: application/json;charset=UTF-8
header: Transfer-Encoding: chunked
header: Date: Sat, 27 Aug 2011 14:47:27 GMT

["experi","php","python","java"]

Process finished with exit code 0
```

Find Users to Collaborate with

This api will attempt to find indexed users with keywords contained in a user's needs. The api will match based on these needs to a compare user's profile ('bio').
If users needs have been indexed pass the user's identifier else pass the needs as a free text parameter.

URL: http://<hostname>/api/collaborate

Method: POST

Parameters

country:	the country location of user	(optional)
state:	the state location of user	(optional)
town:	the town location of user	(optional)
name:	the name of the client application	(required)
api:	the client api key	(required)
user:	unique identifier identifying user	(optional)
needs:	free text string	(optional)

n.b. api requires one of user or needs

```
from urllib import urlencode
import httplib2
httplib2.debuglevel = 1

data = { "country":"Ireland","state":"Dublin","town":"dalkey","name": "demo", "api": "92a68c27-
d1b3-4cc2-ac86-e477bcdcf239","needs":"i am looking for a web developer","type":"bio"}
h = httplib2.Http('.cache')

resp, content = h.request('http://localhost:8081/socnet/api/collaborate','POST',urlencode(data),
    headers={'Content-Type': 'application/x-www-form-urlencoded'})
print(content.decode('utf-8'))
```

produces the following output. The output contains the identifier of the matched users and a frequency count of the matching terms. Output is in json format

```
connect: (localhost, 8081)
send: 'POST /socnet/api/collaborate HTTP/1.1\r\nHost:
localhost:8081\r\nContent-Length: 139\r\ncontent-type: application/x-www-form-
urlencoded\r\naccept-encoding: gzip, deflate\r\nuser-agent: Python-
httplib2/0.7.0
(gzip)\r\n\r\nTown=dalkey&needs=i+am+looking+for+a+web+developer&name=demo&coun
try=Ireland&state=Dublin&api=92a68c27-d1b3-4cc2-ac86-e477bcdcf239&type=bio'
reply: 'HTTP/1.1 200 OK\r\n'
header: Server: Apache-Coyote/1.1
header: Content-Type: application/json;charset=UTF-8
header: Transfer-Encoding: chunked
header: Date: Fri, 02 Sep 2011 19:30:46 GMT

{"123456":2,"123458":2}

Process finished with exit code 0
```

Text to Keywords

The following api call will reduce any unformatted text to a set of key words. The text goes through text cleaning, removal of stop words and stemming processes.

URL: http://<hostname>/api/stemToKeywords

Method: POST

Parameters

country:	the country location of user	(optional)
state:	the state location of user	(optional)
town:	the town location of user	(optional)
name:	the name of the client application	(optional)
api:	the client api key	(optional)
text:	the text to be stemmed	(required)

```
from urllib import urlencode
import httplib2
httplib2.debuglevel = 1

data = { "country":"Ireland","state":"Dublin","Town":"dalkey","name": "demo", "api":
"92a68c27-d1b3-4cc2-ac86-e477bcdcf239","text": "I am a web developer"}

h = httplib2.Http('.cache')

resp, content =
h.request('http://localhost:8081/socnet/api/stemToKeywords','POST',urlencode(data),
    headers={'Content-Type': 'application/x-www-form-urlencoded'})
print(content.decode('utf-8'))
```

Produces the following output. Output is in json format

```
connect: (localhost, 8081)
send: 'POST /socnet/api/stemToKeywords HTTP/1.1\r\nHost:
localhost:8081\r\nContent-Length: 117\r\ncontent-type: application/x-www-form-
urlencoded\r\naccept-encoding: gzip, deflate\r\nuser-agent: Python-
httplib2/0.7.0
(gzip)\r\n\r\nTown=dalkey&name=demo&country=Ireland&state=Dublin&api=92a68c27-
d1b3-4cc2-ac86-e477bcdcf239&text=I+am+a+web+developer'
reply: 'HTTP/1.1 200 OK\r\n'
header: Server: Apache-Coyote/1.1
header: Content-Type: application/json;charset=UTF-8
header: Transfer-Encoding: chunked
header: Date: Tue, 06 Sep 2011 12:15:13 GMT

["web","develop"]

Process finished with exit code 0
```

Javascript Invocation

All the above apis can be called via javascript ajax calls. To enable these javascript apis users are required to have JQuery installed. To deal with cross domain issues the apis include a callback function. This callback function needs to be included in the parameters passed to the api.

```
var data = { "country":"any","state":"any","town":"any","name": "mingle", "api": <api-
key>,"user":<user identity>,"profile": <user profile text>,"type": "bio"}
 var headers = {"Content-type": "application/x-www-form-urlencoded"}
var url = http://<hostname>/api/profile

    $.ajax({  type: "POST", url: url , data: data,dataType: "jsonp",
              jsonp : "callback",
              jsonpCallback: "jsonpcallback",
              success: function (response) {
                 var mess = response;
                 profiled = 'yes';
              },
              error: function (XMLHttpRequest, textStatus, errorThrown) {
               // handle error
               alert('problem with profiling service');
              },
              complete: function() {

                    }
          });
```

```
function jsonpcallback(overlays) {
/// handle response
   var response = overlays;
}
```

APPENDIX V

An outline of the Porter Stemmer Algorithm is given below. This has been reproduced from the official Porter Stemmer web site []

To present the suffix stripping algorithm in its entirety we will need a few difinitions.

A consonant in a word is a letter other than A, E, I, O or U, and other than Y preceded by a consonant. (The fact that the term 'consonant' is defined to some extent in terms of itself does not make it ambiguous.) So in TOY the consonants are T and Y, and in SYZYGY they are S, Z and G. If a letter is not a consonant it is a \vowel\.

A consonant will be denoted by c, a vowel by v. A list ccc... of length greater than 0 will be denoted by C, and a list vvv... of length greater than 0 will be denoted by V. Any word, or part of a word, therefore has one of the four forms:

```
CVCV ... C
CVCV ... V
VCVC ... C
VCVC ... V
```

These may all be represented by the single form

```
[C]VCVC ... [V]
```

where the square brackets denote arbitrary presence of their contents. Using (VC){m} to denote VC repeated m times, this may again be written as

```
[C](VC){m}[V].
```

m will be called the \measure\ of any word or word part when represented in this form. The case m = 0 covers the null word. Here are some examples:

```
m=0    TR,  EE,  TREE,  Y,  BY.
m=1    TROUBLE,  OATS,  TREES,  IVY.
m=2    TROUBLES,  PRIVATE,  OATEN,  ORRERY.
```

The rules for removing a suffix will be given in the form

```
(condition) S1 -> S2
```

This means that if a word ends with the suffix S1, and the stem before S1 satisfies the given condition, S1 is replaced by S2. The condition is usually given in terms of m, e.g.

```
(m > 1) EMENT ->
```

Here S1 is `EMENT' and S2 is null. This would map REPLACEMENT to REPLAC, since REPLAC is a word part for which m = 2.

The `condition' part may also contain the following:

*S - the stem ends with S (and similarly for the other letters).

v - the stem contains a vowel.

*d - the stem ends with a double consonant (e.g. -TT, -SS).

*o - the stem ends cvc, where the second c is not W, X or Y (e.g.

-WIL, -HOP).

And the condition part may also contain expressions with \and\, \or\ and \not\, so that

(m>1 and (*S or *T))

tests for a stem with m>1 ending in S or T, while

(*d and not (*L or *S or *Z))

tests for a stem ending with a double consonant other than L, S or Z. Elaborate conditions like this are required only rarely.

In a set of rules written beneath each other, only one is obeyed, and this will be the one with the longest matching S1 for the given word. For example, with

```
SSES -> SS
IES  -> I
SS   -> SS
S    ->
```

(here the conditions are all null) CARESSES maps to CARESS since SSES is the longest match for S1. Equally CARESS maps to CARESS (S1=`SS') and CARES to CARE (S1=`S').

In the rules below, examples of their application, successful or otherwise, are given on the right in lower case. The algorithm now follows:

Step 1a

```
SSES -> SS                      caresses  ->  caress
IES  -> I                       ponies    ->  poni
                                ties      ->  ti
SS   -> SS                      caress    ->  caress
S    ->                         cats      ->  cat
```

Step 1b

```
(m>0) EED -> EE                 feed      ->  feed
                                agreed    ->  agree
(*v*) ED  ->                    plastered ->  plaster
                                bled      ->  bled
(*v*) ING ->                    motoring  ->  motor
                                sing      ->  sing
```

If the second or third of the rules in Step 1b is successful, the following is done:

```
AT -> ATE                       conflat(ed)  ->  conflate
BL -> BLE                       troubl(ed)   ->  trouble
IZ -> IZE                       siz(ed)      ->  size
(*d and not (*L or *S or *Z))
   -> single letter
                                hopp(ing)    ->  hop
                                tann(ed)     ->  tan
                                fall(ing)    ->  fall
                                hiss(ing)    ->  hiss
                                fizz(ed)     ->  fizz
(m=1 and *o) -> E               fail(ing)    ->  fail
                                fil(ing)     ->  file
```

The rule to map to a single letter causes the removal of one of the double letter pair. The -E is put back on -AT, -BL and -IZ, so that the suffixes -ATE, -BLE and -IZE can be recognised later. This E may be removed in step 4.

Step 1c

(*v*) Y -> I	happy	->	happi
	sky	->	sky

Step 1 deals with plurals and past participles. The subsequent steps are much more straightforward.

Step 2

(m>0) ATIONAL	->	ATE	relational	->	relate
(m>0) TIONAL	->	TION	conditional	->	condition
			rational	->	rational
(m>0) ENCI	->	ENCE	valenci	->	valence
(m>0) ANCI	->	ANCE	hesitanci	->	hesitance
(m>0) IZER	->	IZE	digitizer	->	digitize
(m>0) ABLI	->	ABLE	conformabli	->	conformable
(m>0) ALLI	->	AL	radicalli	->	radical
(m>0) ENTLI	->	ENT	differentli	->	different
(m>0) ELI	->	E	vileli	- >	vile
(m>0) OUSLI	->	OUS	analogousli	->	analogous
(m>0) IZATION	->	IZE	vietnamization	->	vietnamize
(m>0) ATION	->	ATE	predication	->	predicate
(m>0) ATOR	->	ATE	operator	->	operate
(m>0) ALISM	->	AL	feudalism	->	feudal
(m>0) IVENESS	->	IVE	decisiveness	->	decisive
(m>0) FULNESS	->	FUL	hopefulness	->	hopeful
(m>0) OUSNESS	->	OUS	callousness	->	callous
(m>0) ALITI	->	AL	formaliti	->	formal
(m>0) IVITI	->	IVE	sensitiviti	->	sensitive
(m>0) BILITI	->	BLE	sensibiliti	->	sensible

The test for the string S1 can be made fast by doing a program switch on the penultimate letter of the word being tested. This gives a fairly even breakdown of the possible values of the string S1. It will be seen in fact that the S1-strings in step 2 are presented here in the alphabetical order of their penultimate letter. Similar techniques may be applied in the other steps.

Step 3

(m>0) ICATE	->	IC	triplicate	->	triplic
(m>0) ATIVE	->		formative	->	form
(m>0) ALIZE	->	AL	formalize	->	formal
(m>0) ICITI	->	IC	electriciti	->	electric
(m>0) ICAL	->	IC	electrical	->	electric
(m>0) FUL	->		hopeful	->	hope
(m>0) NESS	->		goodness	->	good

Step 4

(m>1) AL	->		revival	->	reviv
(m>1) ANCE	->		allowance	->	allow
(m>1) ENCE	->		inference	->	infer
(m>1) ER	->		airliner	->	airlin
(m>1) IC	->		gyroscopic	->	gyroscop
(m>1) ABLE	->		adjustable	->	adjust
(m>1) IBLE	->		defensible	->	defens

```
(m>1) ANT    ->                  irritant        ->  irrit
(m>1) EMENT  ->                  replacement     ->  replac
(m>1) MENT   ->                  adjustment      ->  adjust
(m>1) ENT    ->                  dependent       ->  depend
(m>1 and (*S or *T)) ION ->      adoption        ->  adopt
(m>1) OU     ->                  homologou       ->  homolog
(m>1) ISM    ->                  communism       ->  commun
(m>1) ATE    ->                  activate        ->  activ
(m>1) ITI    ->                  angulariti      ->  angular
(m>1) OUS    ->                  homologous      ->  homolog
(m>1) IVE    ->                  effective       ->  effect
(m>1) IZE    ->                  bowdlerize      ->  bowdler
```

The suffixes are now removed. All that remains is a little tidying up.

Step 5a

```
(m>1) E      ->                  probate         ->  probat
                                 rate            ->  rate
(m=1 and not *o) E ->            cease           ->  ceas
```

Step 5b

```
(m > 1 and *d and *L) -> single letter
                                 controll        ->  control
                                 roll            ->  roll
```

The algorithm is careful not to remove a suffix when the stem is too short, the length of the stem being given by its measure, m. There is no linguistic basis for this approach. It was merely observed that m could be used quite effectively to help decide whether or not it was wise to take off a suffix. For example, in the following two lists:

```
list A          list B
------          ------
RELATE          DERIVATE
PROBATE         ACTIVATE
CONFLATE        DEMONSTRATE
PIRATE          NECESSITATE
PRELATE         RENOVATE
```

-ATE is removed from the list B words, but not from the list A words. This means that the pairs DERIVATE/DERIVE, ACTIVATE/ACTIVE, DEMONSTRATE/DEMONSTRABLE, NECESSITATE/NECESSITOUS, will conflate together. The fact that no attempt is made to identify prefixes can make the results look rather inconsistent. Thus PRELATE does not lose the -ATE, but ARCHPRELATE becomes ARCHPREL. In practice this does not matter too much, because the presence of the prefix decreases the probability of an erroneous conflation.

Complex suffixes are removed bit by bit in the different steps. Thus GENERALIZATIONS is stripped to GENERALIZATION (Step 1), then to GENERALIZE (Step 2), then to GENERAL (Step 3), and then to GENER (Step 4). OSCILLATORS is stripped to OSCILLATOR (Step 1), then to OSCILLATE (Step 2), then to OSCILL (Step 4), and then to OSCIL (Step 5).

APPENDIX VI

Geomingle Social Network

New Event Details

The first step in using the mingle application is to create an event that is upcoming. The organizer enters their event details. This portion of the application is run on a desktop browser and can be accessed via url http://geomingle.appspot.com

The event organizer must first enter the location of the event. It is important that the location of the event is located on the Google map accurately to reflect the actual geo position of the event. If this is entered incorrectly then attendees will not be able to log onto the social network once they have arrived at the location of the event.

Once located the event organizer must enter the remaining event details.

The geo location of the event location is stored along with the other event details

The user can then generate marketing material associated with the event

The application will then generate QR codes to link to the url of the event and to the geomingle social networking application.

Social Networking Web Application

The social networking application has been developed as a web based mobile application. It is optimized for use on a browser with a smart phone. It has been developed using HTML5 and css3 standards. Consequently the application may not run optimally on a desktop browser, not supporting these latest technologies.

By scanning the QR code generated in the event publishing application or by going to the url

http://geomingle.appspot.com/geo

User will be directed to the following screen in their browser.

The application uses Facebook to authenticate the users to the application. The user must enter their facebook credentials in order to gain access to the system.

user gives permission to allow miingle application access facebook data

user enters facebook credentials

miingle application displays users profile

The application takes the users profile from facebook and sends the data to the Profiler API application for indexing. The application divides the profile data into the following

Profile:	based on the Facebook bio
Likes:	generated from facebook likes and interests
Collaborate on:	not sourced from Facebook. User can enter details of what they wish to collaborate on, based on the event they are attending

user can select how they want to contact other event attendees

view list of contacts

view contact detail. request a dialog

Users can attempt to find fellow attendees to collaborate on based on

Profile	to	Profile
Collaborate on	to	Profile
Likes	to	Likes

The application accesses the Profiler API aplication to find matches based on its matching algorithm. The user is then presented with a list of matchng contacts. From this list the user can view the contact details and request a dialog if they desire.