Interim Report

Michael James

"Twitter-style" App for Group Communication

Supervisors
Professor A.D. Preece
Dr I.J. Grimstead

Moderator
Professor N.J. Avis

Abstract

"Twitter-style" App for Group Communication is a project investigating and eventually implementing an anonymous communication application. The app allows users to communicate anonymously; and could be used in a range of contexts where anonymous communication may motivate users and give an insight to observer's unspoken feelings. The observers could potentially make changes to the organisations methods, improving service to the users of the system. This document discusses the progress on the first few phases of this project, identifying current solutions (similar systems) and potential systems that could play as an important carrier for transporting messages. The discussion then progresses on to testing different methods and first stages of design for the first prototype.

Contents

Introduction

This report discusses the progress on the first few phases of the project "Twitter-style" App for Group Communication. The projects fundamental aim is to create anonymous communication for groups of individuals that share something in common. Users of the system will send messages to a central hub, a hub in which repeats the content of the message to all users in the system. When a user receives a message they can reply if they wish. The identity of the sender is stripped away from the message before it is sent to the users of the system; hiding their identity. The original message is stored into a log, if misuse has been reported, the administrator can refer to the log to identify the true sender that inflicted the misuse. Entry to the system will be through SMS to offer cheaper access to the system rather than using data rates. To encourage users in participating in the system, a public display will be used; displaying activity on the system. Again the users identity remains hidden. The display will be located in an area where the individuals using the system pass regularly.

Once the basics of the system are sound and implemented, observers will be able to view the conversations taken place within the system. Observers will view the conversations, without a users true identity being revealed. The observer will use this knowledge from the conversations to determine decisions; this concept could potentially work in various environments, all with a similar goal in mind to improve their service/product.

It is a closed system and requires approval by an administrator or representative to access the system. The environment where the system is used will determine the credentials a user will need to present to the administrator or representative to gain access to the system. For example a student at an educational institute will most likely present their student card, a card with a passport sized photo and their unique student number for that institute. Alternatively the system could be used within a hospital environment; patients will be identified by their medical bracelet or alike, depending on the identification method used in the hospital.

This document focuses on the details of the essential requirements defined in the initial plan and how I plan on solving the problems these requirements address. The solutions will then be taken onboard to help in the implementation of the first prototype. The essential requirements will build a solid foundation for the system. Additional requirements at the moment consist of how the activity on the public display will be appropriate and represented; the document shows a few ways in which this could be accomplished through analytics API’s. Most of the additional requirements will come later in the project when the first prototype is complete; Improvements and ideas will be highlighted creating additional requirements by myself and by users that will help test the first prototype; this will be documented in the final report.

Background

The following section investigates systems that are similar to the “Twitter-Style” app for group communication concept; investigating how they work, whether there is anything to be learned from current systems that could be used to help design the system. Furthermore the research looks into social platforms that could be used as the transportation channel of messages.

Similar systems

Some research was spent on looking at similar systems that use public displays. The closest systems available were text2screen systems. Text2screen systems are typically designed for receiving SMS messages from users in bars, clubs, events and public spaces. Users send an SMS message to a specific number, where the message is then displayed on a public display; their identity is hidden from those viewing the public display.

iWall

A sophisticated and expensive package that retrieves SMS messages through a GSM modem connected to the host (iWall. 2007-2011). A system aimed towards bars, clubs and events. Messages are then processed on the iWall software where it determines whether the text can be displayed through the moderation tools provided. The moderation is interested in whether a number is allowed or whether the content of the message is flagged as profanity. If approved the message will display on a flash application, where the application will be set to full screen on an external display(s).

The software also allows users to optionally set an alias in the required format "{prefix} alias"; their alias will be displayed along with their messages on screen. Users of the system are then able to send direct messages to one another through the @alias command followed by a message. These messages are private and are not displayed on screen; the contents of the messages are logged for misconduct. There are no analysis tools available for this system.

Text2Screen

A package that has similar functions as iWall, although the GUI and display output are not as neat and professional as iWall (Text2Screen. 2010-2011). The system allows administrators to create a list of banned words through its moderation tools provided; to determine whether a text can be displayed on the public display(s). Again this system also provides private messaging between users, like iWall; through a mechanism that's popular on Twitter @username. Users are assigned an alias; their alias is placed on the public screen for all to see. Users can then use the alias to send a direct message; the private messages are not displayed on the public display. Again the system logs messages for misconduct that may be reported/discovered. This system also does not apply any analytics to the content of messages to return further knowledge on its users. The system is again aimed to draw in customers in bars, clubs and events.

Similar Systems Summary

Research collected five different text2screen systems; the systems mentioned previously are the only systems found that provide private messaging in an anonymous fashion. The remaining systems provided only functionality to display texts to a public display through moderation tools; for more information on the remaining systems see (Appendix A). None of the systems provided analytics on the conversations or comments made on the system. The closest form of analysis was used to create a voting system, where users would vote A, B or C to a certain question, the results would be displayed on the public screen; this function appeared on four of the five systems discovered.

Researching these kinds of systems helped identify different use cases in the projects system; identifying interactions different users will have. i.e. a system administrator would have a different screen; a private screen that will display incoming messages, those messages will then be moderated by the system administrator or by automated moderation tools, removing any unwanted language

or misuse. The participant users of the system would submit a message and view an abstract view of their message on a public display.

The system being designed is more closed than the text2screen systems; users must be authenticated before using the system. I.e. a user may have to present "in person" an identity card to the system administrator, where they will be approved access or alternatively the user registers interest in using the system via a web browser, their credentials are taken into consideration and used to determine whether a user will receive access to the system.

Potential Platforms

This section will identify public application programming interfaces (API) that could contribute to the development of the system. Using pre-existing methods available in an API will be reliable as an external group operates the reflection service on the public methods available. Furthermore the approach will increase functionality and save time in implementation. This section will investigate public platforms available that could be used to create the communication channel for the system.

The chosen platform will require full SMS integration to allow any user with a basic cellular device to access the system. As a recommendation, users are required to create an account solely for the purpose of using the system; this will ensure their messages are not leaked to other users outside of the system.

Three popular social networking platforms were chosen: Twitter, Facebook and Google+. All the platforms have public APIs; their functionality and terms of service (TOS) are compared against one another in Figure 1.

Features	Google +		Facebook		Twitter
Real Name Policy?	✓	Very strict policy, must use a name that people will recognise you by. (Google. 2011).	✓	Very strict policy, must use a name that people will recognise you by. (Facebook. 2011)	✗	No strict policy, most users are identified by pseudo name. (Twitter. 2011)
Aliases?	✗	Real name identifier.	✗	Real name identifier. Searchable by username or email. Not fully alias friendly, cannot just set alias.	✓	Twitter typically allows users to be identified by their username. I.e. @CSTweetBot.
Private Accounts	✓	Minimum visibility with highest security settings: display picture, real name.	✓	Minimum visibility with highest security settings: display picture, real name. Can also set how the account can be found on search.	✓	Account is searchable, but activity is kept hidden to those who are not accepted.
Activity visible to followers/friends	✓	Circles to set who sees what.	✓	Privacy settings can adjust who views what.	✓	Public/Private.

Can user have second account?	✗	Against terms but hard for Google to enforce. But If the system takes-off then this maybe a problem. Not a recommendation, user would have to use their normal account to use the system.	✗	Against terms but hard for Facebook to enforce. If the system takes-off then this maybe a problem. Not a recommendation, user would have to use their normal account to use the system.	Not Mentioned ✓	Users can have multiple accounts, no strict rules. Twitter Apps allows for multiple accounts to be logged in at once. See image.
Adequate SMS Integration	✗	Notifications and send to feed. Only available in India and USA.	✗	Very basic functionality.	✓	Close to full functionality. Users are unable to set privacy, accept followers.
Sign up to social network via SMS?	✗	Not available. Users need to sign up to Google + via google.com.	✗	Users need to sign up to Facebook via facebook.com.	✓	Users can sign up to Twitter, via SMS.
Extensive API	✗	Only get methods, no set methods. No write access to Google+.	✓	Practically full functionality.	✓	Unable to accept followers via API on private account. Need to visit twitter.com to accept.
Features = possible anonymity?	✗	With the basic API, the system would be unable to use Google+. Requires set methods (Write access).	✗	Real name policy. Users must have separate account. Friends would see submission.	✓	Tweets can be picked up from users and used to tweet on bots feed.

Figure 1: A table displaying a comparison between Google +, Facebook & Twitter

From the table analysis (Figure 1); it is clear to see that Twitter is the most suitable social platform for the system, it has a more flexible terms of service allowing users to have multiple accounts, user accounts are not identified by real-name, instead they are identified by alias "Pseudo anonymous". Users if they so wish can be identified by real name, but for the purpose of this system, users will use alias. Twitter also has an extensive public API and has a highly functional SMS system.

Figure 2 shows a further investigation into Twitter and its functions. The table also shows a comparison between two platforms Twitter and a totally new platform, which will require all contacts, messages, distribution and many other features to be implemented by myself on an Android application. A possible solution but will be very time consuming to implement in comparison to the Twitter platform. Figure 2 determines whether the functions Twitter provides could be implemented on an Android app, if so would it require a lot of time to implement?

	Twitter		Android Application	
Features	✓ ✗	Comments	✓ ✗	Comments
Notifications received via SMS?	✓	User can set up SMS notifications via Phone. Function exists, no need to implement.	✓	Can be implemented with API, but would require more effort in implementing a function that can be accomplished with the Twitter API.
Available Analytics	✓	There is a vast amount of open source services for raw text example: Wordle java release by IBM. Plus twitter specific analytics, TweeSpeed, Twitter Sentiment.	✓	Producing analytics on an android application would require android specific packages (limited). Therefore to make use of other packages such as Wordle; the data collected will be required to send raw data to a workstation to be processed. Requires additional coding to send data from phone to work station in comparison to Twitter method.

Possible pseudo anonymity?	✓	Auto generated id's or Twitter id's could become pseudo names.	✓	Could use a similar system found in current SMS systems iWall or t2smobile Software. Requires further coding for a feature that's available on twitter.
Can you send a follow request via API?	✓	Friendships/create, minimal amount of coding.	✓	Yes, again would require further coding, additional following functionality.
Can you accept follow requests via API?	✗	Currently not available. Accounts will be created on the 'fly' or in bulk before a user requests access. Could use a script to automate the creation (Selenium. 2011).	✓	Users number will be authenticated before account is created for user. Users account will be hard coded in retrieving messages from Bot.
Can accounts be created via API?	✗	Currently not available. Accounts will be created on the 'fly' or in bulk before a user requests access. Could use a script to automate the creation of accounts. Selenium (Selenium. 2011).	N/A	
Can you pick up followers messages?	✓	Statuses/home_timeline, Statuses/user_ timeline	✓	All messages will be sent directly to the phone, where they will be received and processed.
Can you tweet via API?	✓	Statuses/update Statuses/update_with_media	N/A	
API call restriction?	✓	150 automated API calls per hour (flexible depending on kind of usage). Educational license available.	N/A	Closed system, completely separate from third party API's for communication, no API Restrictions.
Programming Language Experience?	✓	JAVA API available, Twitter4J	✓	Android is written in JAVA, ran in Dalvik virtual machine, optimised for mobile.

Figure 2: A table displaying a comparison between using the Twitter API and Android platform.

The use of twitter would allow for a quick and effective communication model for the system; twitter provides the basic requirements needed for the system. As mentioned the user accounts will be created manually on the 'fly' when a user requests access to the system. Alternatively the system administrator will make accounts in bulk through a web-browser on twitter.com; accounts will be ready waiting for new users to approach the system administrator for access. The method has arrived; due to a minor inflexibility on the public Twitter API. On the other hand this method may not prove to be a bad after all; as it will ensure all users are authorised before accessing the system. Users will meet face-to-face with the system administrator or representative where they will receive their account; based on their credentials. The concept of registering a twitter account is relatively similar to a user registering their mobile phone number into the system for the Android direction. The system administrator will create a map between true identity and the anonymous Twitter account.

When registering Twitter accounts, Twitter requires an email address an address that is not currently assigned to an account. Therefore when new accounts are created; new email addresses will need to be created. This brings in the complexity of real-name policies again; Gmail, Hotmail, Yahoo and many other services require real names. Therefore as a work-around, new email addresses will come from an email-hosting server, a server that is most likely internal and has functionality to create an unlimited amount of email accounts. The accounts will not receive any mail, as notifications from Twitter via email will be turned off when each account is created. The email account will be created for the sole purpose of being able to create an account. Users may not have email addresses; this method will allow users

without email addresses to have access to the system. In the context of a hospital this is most likely to be a common occurrence.

Twitter provides a very functional SMS system. The system allows users to register and login; in the context of this system the users will only be required to login to twitter via SMS. Users will not be required to register to Twitter via SMS as the accounts will need to be created before the user attempts to access the system via SMS. Users are able to login to Twitter and start following an account and start receiving messages that account has sent (Twitter. 2011). In the example (Figure 3) shows a user following CSTweetBot via SMS.

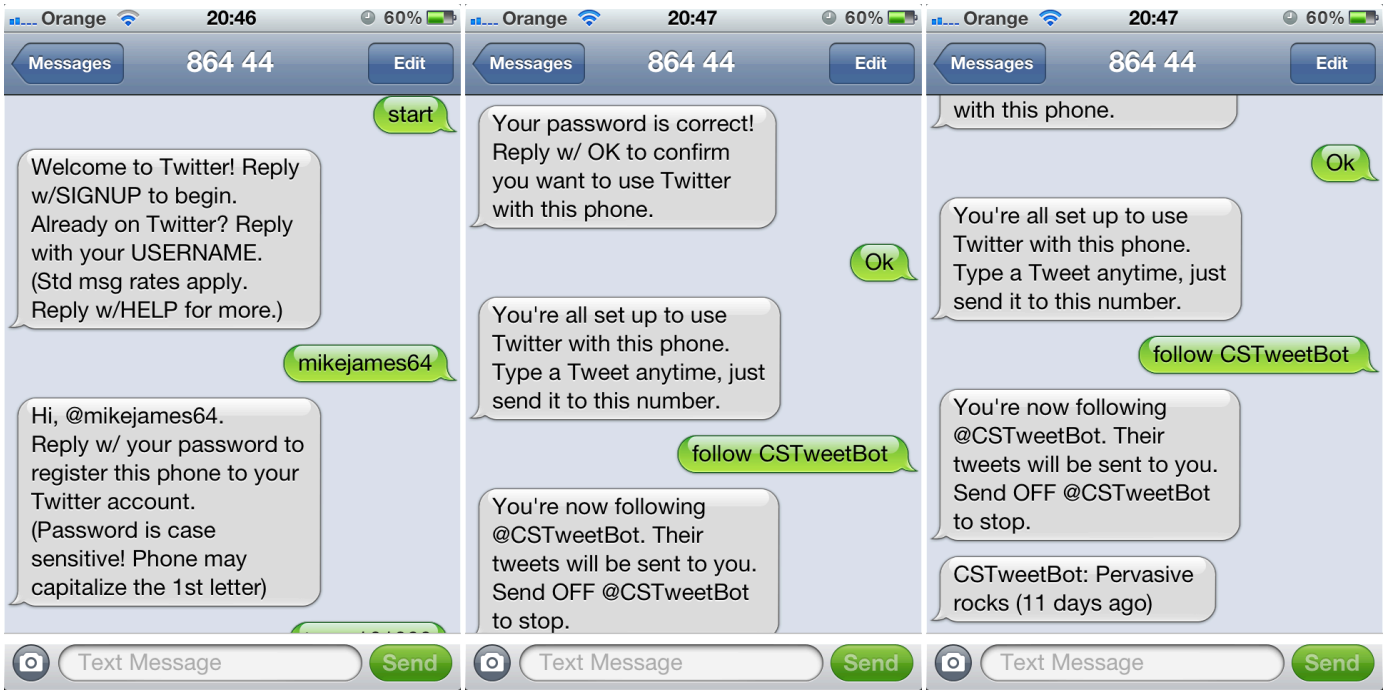


Figure 3: A demonstration on how a user can login to twitter and start following a user.

The demonstration shown in (Figure 3); shows that the system could work using Twitter to provide the SMS transfer of messages; satisfying the SMS requirements for the system.

It is important to understand that the SMS system in twitter retrieves a smaller set of tweets, this is to reduce the amount of SMS messages being sent. Figure 4 helps visualise this. In the context of this system the user will only need to follow a single account.

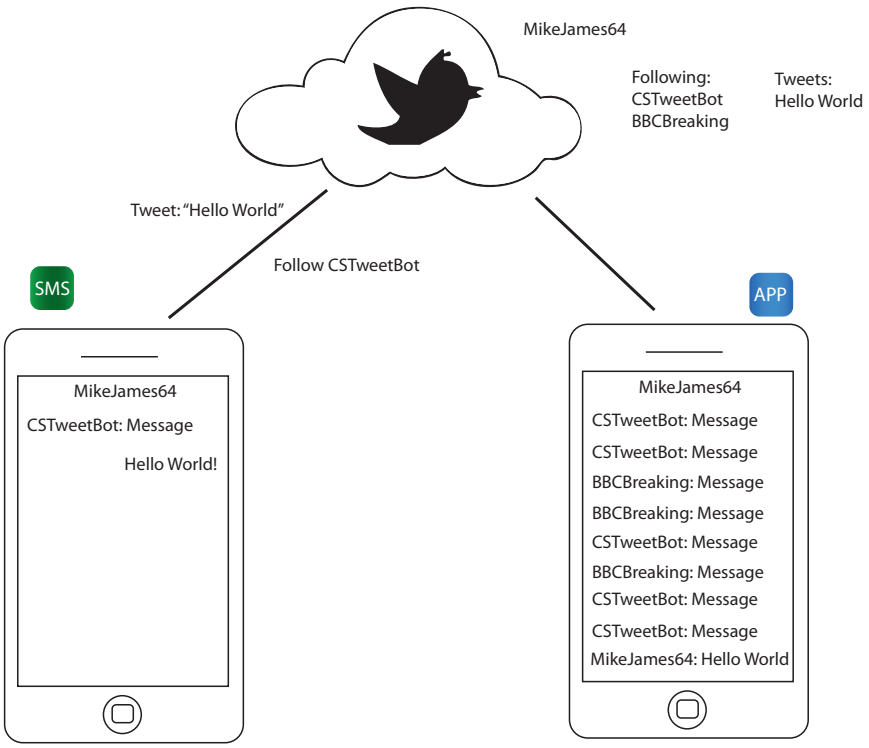


Figure 4: Shows how a SMS user receives fewer notifications than the twitter APP.

A work-around was considered that would solve the issue with the Twitter API where requests cannot be accepted on the twitter API for a private account. The work-around consisted of an automated script that logged into the TweetBot's Twitter account and viewed a list of follow requests, those requests are then compared to a list of new users, if there is a match the script will approve access to the system. The script would be written through JAVA using a free API called selenium (Selenium. 2011). The API talks directly to Mozzila Firefox and allows the script to interact with the browser like a normal user. Later research looked into the Twitter terms of service to identify whether the use of selenium is against the TOS. The research showed there are no direct rules, but the following rule could stop the use of selenium to create accounts. It would most likely stop the Bot from accepting friendship requests through selenium.

> You may not do any of the following while accessing or using the Services: access or search or attempt to access or search the Services by any means (automated or otherwise) other than through our currently available, published interfaces that are provided by Twitter (and only pursuant to those terms and conditions) (Twitter. 2011)

A message will be sent to Twitter in the following semester to determine whether scripting via selenium would be against the TOS.

Figure 5 is used to help determine how the messages and friendship requests are sent across Twitter. The diagram helps visualise how the Bot will strip a users identity from a tweet and distribute the tweet for users to see.

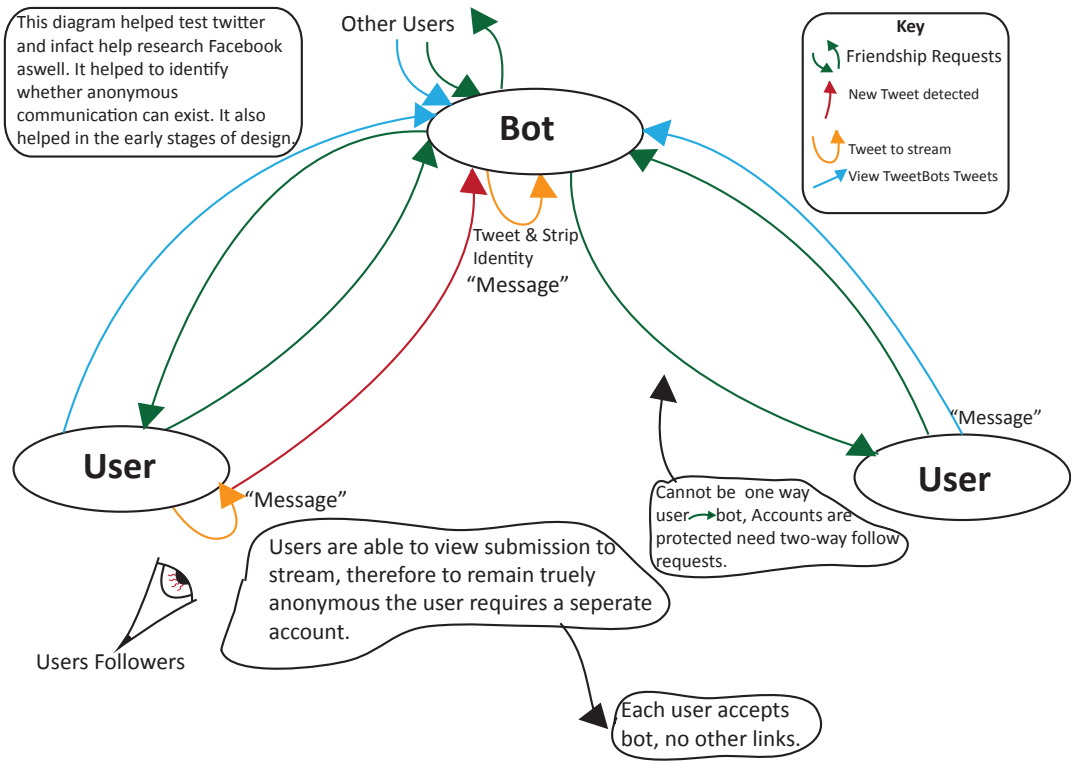


Figure 5: A diagram showing the general connections and messages within the concept of the system.

Implementing upon the twitter platform would allow users alternative access to the system; if they do not wish to access the system via SMS they can access it via a web-browser or a Twitter app on their mobile device. Creating more flexibility for the systems users. Furthermore the use of Twitter would allow time to be invested in implementing analytics tools for a public visual display, creating a 'hook' motivating new users to join in and take part in using the system. The analytics will also be used to analyse the qualitative information allowing professionals to view data and determine decisions upon the data provided in many different formats.

A further investigation of Twitter terms of service (TOS) was conducted. The focus was on whether running automated programs on the twitter API are allowed. The TOS suggests they are allowed, one point shows the following.

> D. Do not store non-public user profile data or content. (Twitter.2011).

This is one rule that could affect the system, although by declaring the content of the system is public to the system and is within the applications TOS; this therefore should be within Twitters TOS.

Analytics Tools

Research was taken out in parallel with the research on Twitter to gather various analytics tools that could analyse the collected tweets. The research showed many open source packages that are specifically designed for twitter and general analytics tools that operate on raw data.

Wordle

In the fortnight meetings of the project the concept of using a Wordle was mentioned regularly and how it could be used on the public display. A feature like this could entice new users to take part in the system and create a 'hook' for current

users to continue using the system. Wordle.net is a web-based application that allows users to copy and paste text into a text area, where the Wordle algorithm will generate a sophisticated word cloud (Feinberg.J, 2011). Words are arranged in various directions and colours; they are randomly placed. The words are placed close together, creating a collage effect, creating a wonderful visual display. The higher the frequency of a word the larger it appears. Commonly users use Wordle's to display a summary of their tweets where links to generated Wordle's are shared over social networks or on their or website. It turns out Wordle.net does not have an API, further research discovered the same algorithm used on Wordle.net has been released in an IBM package. The designer of Wordle is an IBM employee. The package is a JAR file, which is executed in the command line with a set of parameters to set the input file (raw text), configuration (colours and arrangement etc), dimensions and an output file to contain the Wordle; the output image is a PNG file.

Twitter Sentiment

A public API that uses the JSON query format to interact with the sentiment interface (Twitter Sentiment. 2011). The API analyses tweets given and processes the tweets, determines different colours for different sentiments. Green means positive, white means neutral and red means negative. This could prove to be a useful tool in the third party analysis of the data, accessed by a professional (observer) and determine decisions based on the results presented, to increase service, operation or syllabus etc.

Google Chart Tools

A large set of charts that can display the data in various ways; pie charts, bar charts, scatter charts etc, a large amount of the charts allow users to interact with the charts, to identify further details. The charts are Java-script based and can be accessed through the public API available from Google. Not restricted to twitter and could provide charts to professionals again to view statistics on certain topics.

Approach

This section focuses on the first stages of design to help identify actor's interactions in the system. These designs will most definitely be updated further along in the project to add further functionality in the system. After design, the discussion will move onto current solutions to the essential requirements of the project. The essential requirements will be addressed in a practical manner, through small independent test programs, which will be taken into consideration when formally designing the first prototype in the next phase of the project.

The methodical approach the project has been taking is an iterative, agile approach. The iterative part refers to how the project identifies new deliverables to be accomplished every two weeks; so far this method has proven to be realistic, efficient and flexible with my other activities. The project keeps a clear record of what has been accomplished through informal documents and then adapted to be presented at each milestone in formal documents. The project has almost taken a full lap, working through the phases of the project. The next phase will be implementation of the first prototype with concurrent testing whilst being developed. Small internal tests will be conducted on the first prototype. If approved by the CS/IS Department, regarding ethical issues in testing the system with students. A live test will be conducted, by making the system temporarily live with students, users feedback and activity on the system will be taken into consideration, to identify new tasks for the project. This is where the project will loop back to the start of the set of phases and begin the second iteration. Research may need to be carried out on the new features to be implemented.

First Stages of Design

The background has highlighted issues and determined decisions. The project will use the Twitter platform as a communication channel, through its publically available API. The user accounts will have https enabled and protected tweets enabled, ensuring all data is kept internally within the system.

Figure 6 shows a use case, a general use case showing general interactions a user will make upon the basic version of the system. The use case is written at a high level of granularity.

General Use Case
Abstract View

Environment: General
Scenario: General interactions with the system

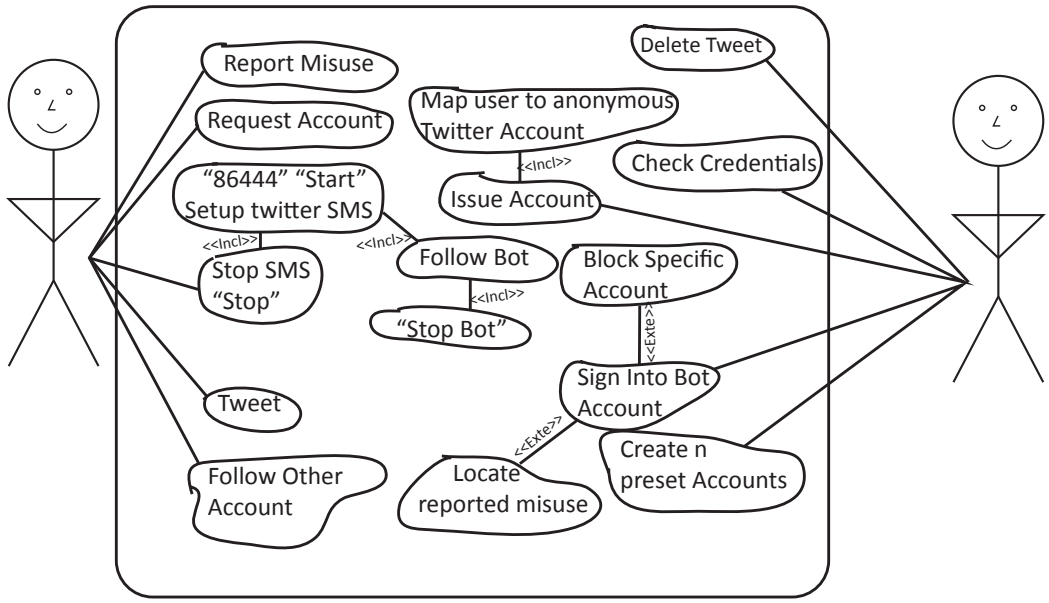


Figure 6: Use Case, General user interactions

Figure 6 helps capture many interactions with the system. The user will start its first interaction with the system by requesting an account. As mentioned previously in research, the user could either request the account in person or via a web browser. In the light of this; I feel that the system will require a strict registration process; therefore a face-to-face registration process is required. This interaction can be found in further detail in (Figures 7 & 8). As shown previously twitter provides a very functional SMS system (Figure 4) the user is able to start/stop Twitter SMS and is able to create a follow request to the private account CSTweetBot; the account will then start to receive Tweet updates from CSTweetBot.

The user is able to tweet, the tweets the user makes are visible to the Bot and the sender only. The content of the tweet is copied and placed onto the Bots wall for all followers to see. The final interaction is allowed in the system as the user can follow a User and read their tweets but that is all. If the user were to @JimBob1 an account they are following, the tweet wouldn't appear on that users wall, as JimBob1 also needs to be following the user to receive the mention. If the user receives a follow request from JimBob1 they will need to deny it In order to stay within the TOS of the system. To enforce this rule the bot will need to have a thread, which scans users profiles periodically to determine whether they have made any followers. The user must have only one friendship and that is to the Bots account. If the user is found having additional followers, they will receive a warning via a direct message (DM) using the Twitter API. If the user is caught again, that friendship is blocked on the bots account using the Twitter API. The account is no longer part of the system, the account is not linked to the institution in anyway as the usernames created will be a

sequence of characters that do not mean anything, deeming useless to the user once their outside of the system. This strict policy is in place to ensure users do not misuse the anonymity the system brings to users. This method will protect the institution from any complaints the user may have caused. The email address assigned to the account will also be deleted; this will be done manually on the email-hosting server.

Figure 7 shows how a user account is issued; in the context of a school within university. Interactions will be face-to-face as mentioned previously.

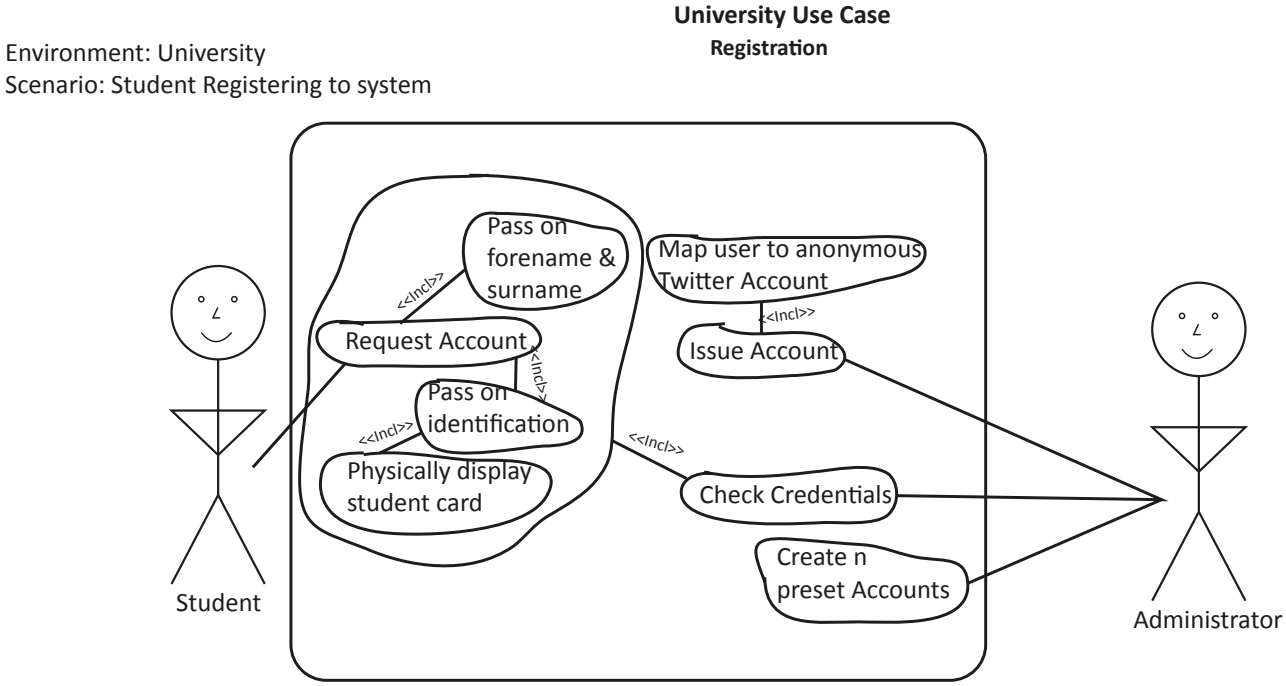


Figure 7: Use Case, Student Registration

Figure 8 shows a use case showing how a nurse within a hospital could approach a patient within a ward and suggest whether they would like to participate in the group communication system. The Nurse would then be able to map a preset account to the patient through a set of credentials; these credentials will most likely be their forename, family name, patient number and contact number.

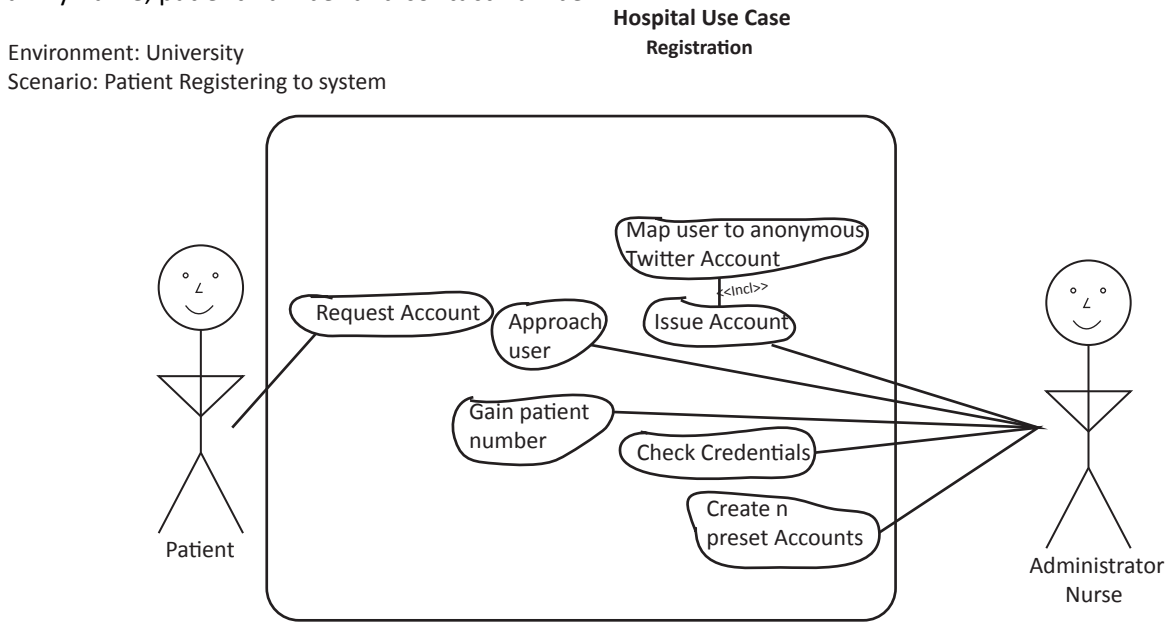


Figure 8: Use case, Patient Registration

Figure 9 shows how the system administrator will be able to create accounts for the system. These accounts will be created prior to a user requesting access to the system. As mentioned previously (Research) the administrator will have to create email accounts for each twitter account. These accounts will be added to an email server that does not have a names policy. Ideally this email server will be an internal email server allowing random address names. The email address will be created for the sole purpose of creating a Twitter account and will allow flexibility to users described in the previous use case Figure 8 where users in this environment may not have an email address.

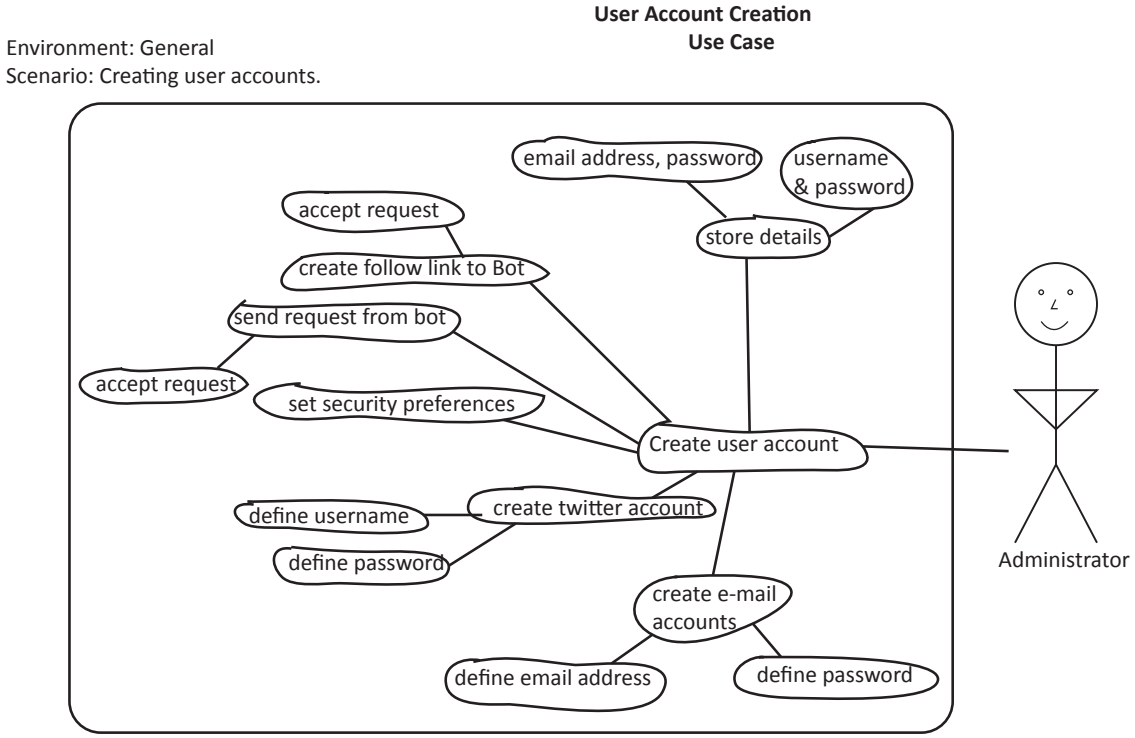


Figure 9: Use Case, displaying account creation.

Testing for First Prototype

To clarify, this section will explore elements of the system individually, elements that will need to be implemented into the overall system to meet the essential requirements. This section is very much a research section looking at how the essential requirements can be met from practical point of view. By creating small programs will help structure the system and will help gain knowledge on how the first prototype will be designed; design of the first prototype will be found in the first few sections of the final report. The small programs will show how each requirement has been tackled; if the test is successful it will later be adapted and improved to later be joined with other test programs; creating the first final prototype which will be created over the Christmas recess. This phase was conducted to ensure security and the success of the first part of the project. The phase helped to further ensure using the Twitter platform was the correct decision. These tests were conducted as soon as Twitter appeared to be an appropriate, pre-existing platform for the system. If the tests proved unsuccessful then there was still time to revert back to further research in Twitters API or alternatively redirect to the Android approach where the project would take a different direction.

Create Anonymous Communication

This program is written in Java using the Twitter4J API, the program is a successful attempt in creating a basic anonymous communication application. The app talks directly to a Twitter account CSTweetBot; the account plays the part as the bot. Two users follow the Bot: MikeJames64 and MikeJames75. The test shows MikeJames75 sending the first tweet. The tweet was sent before the application was turned on, which is the reason why the tweet didn't bounce back until four minutes later. After which MikeJames64 sends a tweet via SMS whilst the Bot is running. Prior to this MikeJames64 can see other users submissions; they are received as incoming SMS from Twitter, MikeJames64 is following the Bot via SMS therefore is able to receive new messages as they come in. Once MikeJames64's message has been sent MikeJames64 will then shortly receive a copy of the message sent back to them informing the message has been picked up by the Bot successfully and is now visible to all users on the system. The test shows (Figure 10,11) users can see other people's tweets without knowing who sent them allowing the users to conduct anonymous communication. For code listings see (Appendix B).

Figure 10: MikeJames75 Account on Web Browser

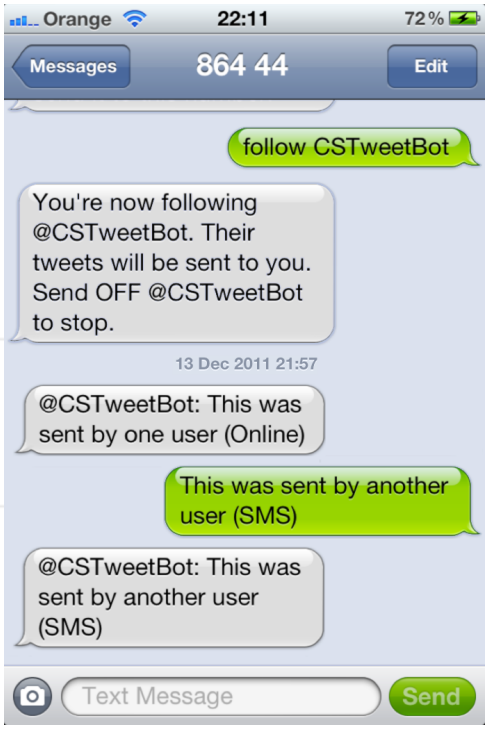


Figure 11: MikeJames64 Account on SMS

Figure 12 shows output from the Java application; displaying MikeJames75's tweet coming into the system first, where it is then used to create a tweet on the Bots feed, the same goes for MikeJames64's tweet.

```
Detetected: MikeJames75:This was sent by one user (Online)
Sent: CSTweetBot:This was sent by one user (Online)
Detetected: MikeJames64:This was sent by another user (SMS)
Sent: CSTweetBot:This was sent by another user (SMS)
```

Figure 12: Application output in Netbeans

The application has no GUI, it creates the connection to Twitter where it then starts to receive new tweets from followers through the "home_timeline" method, currently the method has paging set to 20 tweets; collecting 20 tweets every 30 seconds. The home timeline includes the Bots submissions as well, therefore to stop the bot from submitting its own tweets; a condition is set; checking the screen name set to the tweet is a follower and not the Bot.

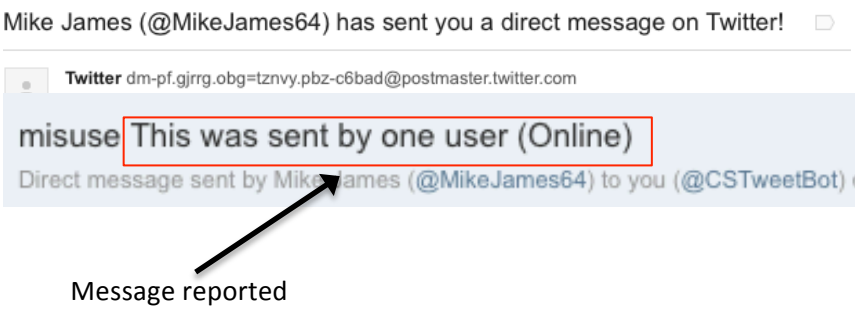
Reporting Misuse

This test shows how users could report misuse in the system; the system administrator will be able to identify the reported message using Twitters timeline; to track back and identify the sender. Once the administrator has identified the sender they may need to identify the true identify of the user, this will be done through a webpage that uses a MySQL database to map a Twitter ID (Screen Name) to true identity, scripts to communicate with MYSQL are written in PHP. To view snapshots of code see (Appendix C). Once the true identity of the sender has been found; the administrator can then take further action if required.

The scenario is as follows; MikeJames64 sends a direct message to CSTweetBot as seen in Figure 12. The message notifies the administrator via email (Figure 13) that a direct message has been sent. The administrator reads the content of the message and notices it's a misuse report.

Figure 12: MikeJames64 reporting misuse via SMS

Figure 13: Administrators Email, containing misuse report



The administrator can then proceed to identify the user; using a user map, a web page. A page in which holds the details for all the users in the system. The administrator can search for Twitter accounts according to their TwitterID. When users are added to the system they are added to the user map through the 'Add User' form (Figure 14). Figure 15 shows the administrator searching for MikeJames75 and shows the search result. Alternatively the administrator can view all users by selecting View Map (Figure 17). The administrator has the functionality to delete accounts from the user map. To permanently delete a user from the system; the administrator would also need to log into the Bots account on Twitter, ending the friendship to the user that's misused the system.

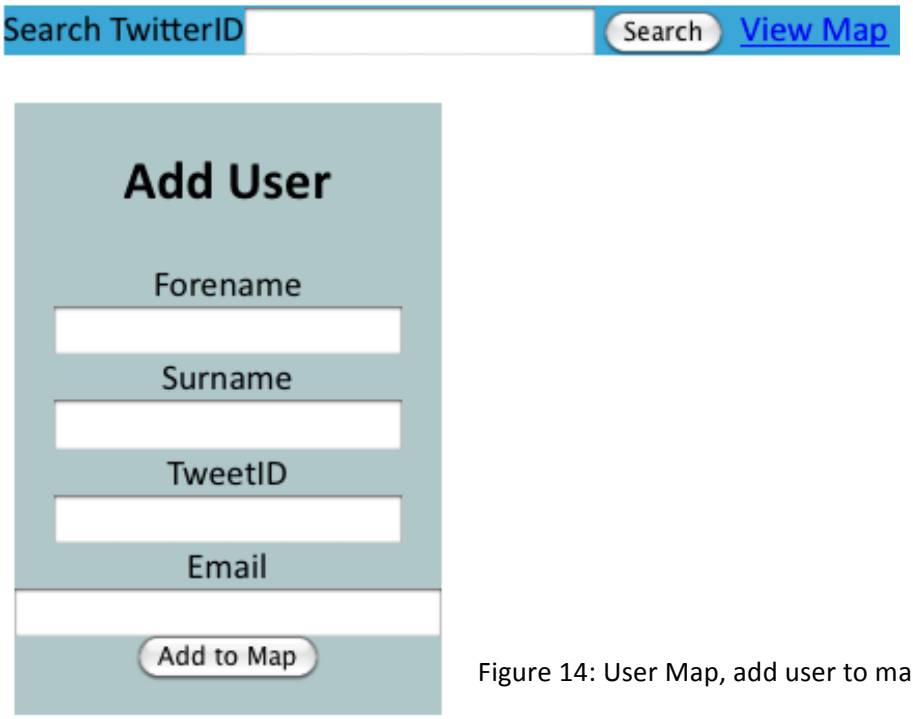


Figure 14: User Map, add user to map form

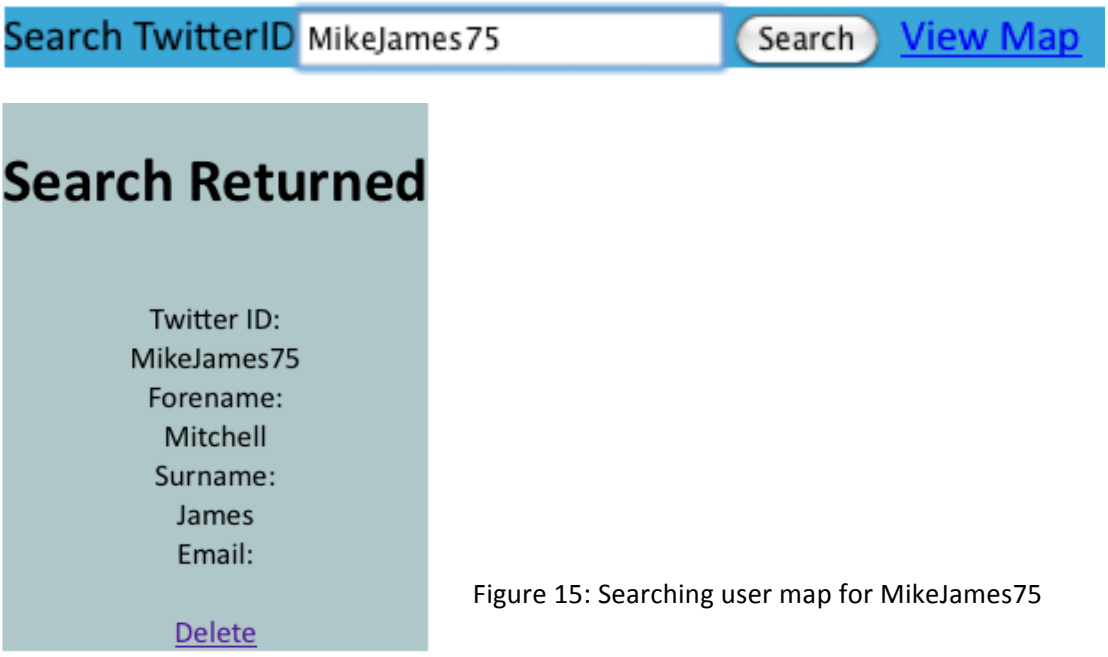


Figure 15: Searching user map for MikeJames75

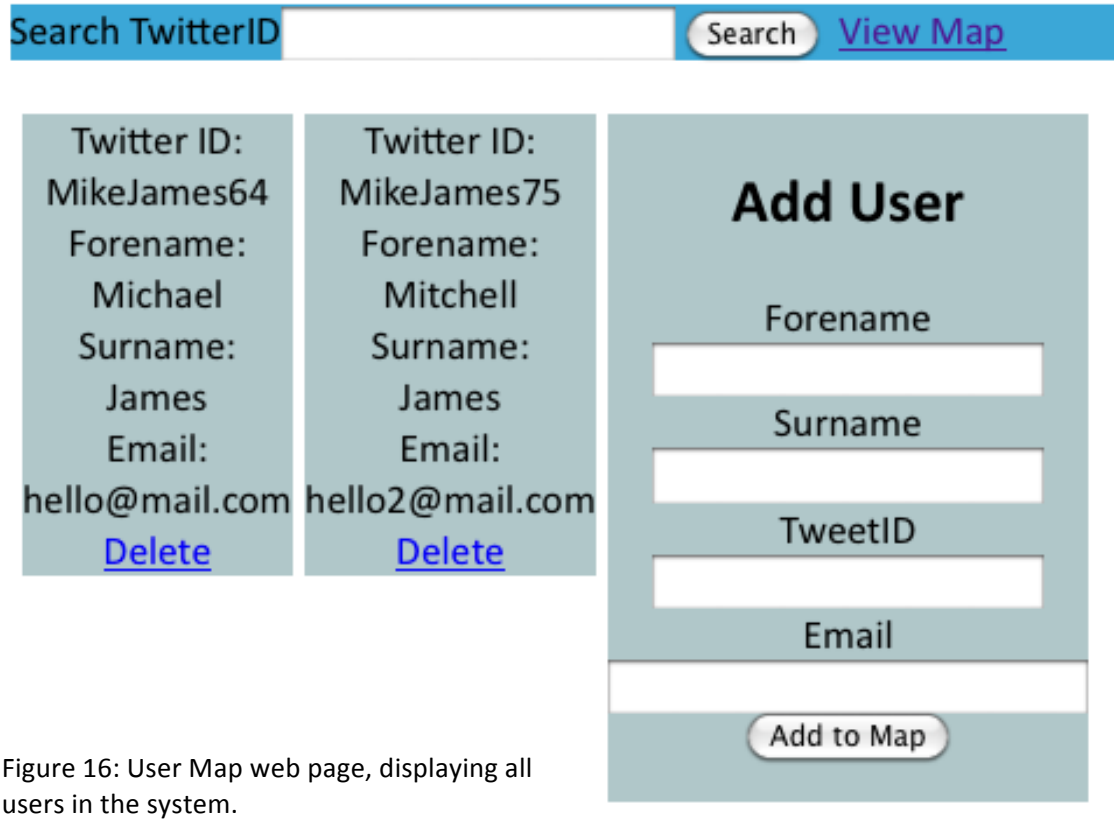


Figure 16: User Map web page, displaying all users in the system.

Public Display

This test (concept) shows a way in which content can be displayed on an external display. The display could be a large screen in public area with a web page displaying content; in this case the web page is being displayed on a mobile phone, pointing to the web server on a local network.

The test shows a Java application creating new content on a web page every minute. The web page is local to the machine and the apache server points directly to a folder, which the Java application has access to. The Java application updates the index.html directly. Figure 17 shows the Java application editing a web page every minute; for each minute that goes by a counter is incremented and the file is updated. For code listings see (Appendix D).

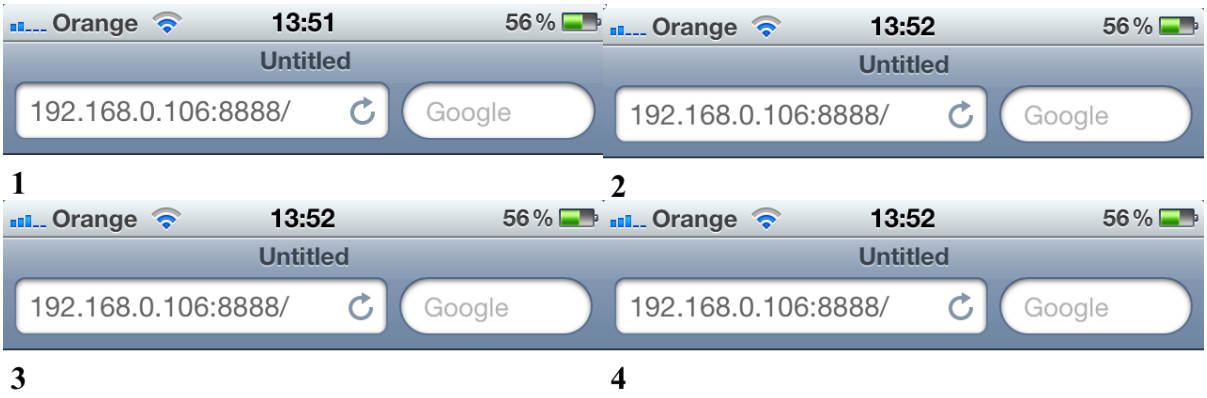


Figure 17: Showing the external display program on a web browser

Conclusion

The project so far has conducted research to identify which direction the project should take. The project will use the existing public platform Twitter provides through their API. The API can be manipulated to create the test programs previously discussed to meet the essential requirements. The test programs will need to be improved significantly to implement the first prototype; visible improvements already consist of improving the security on the user map site. The map is still a concept and provides no security in who is authorised to search, add and delete users from the map an update that will be required before going live.

Once the first prototype has met a certain level of quality it will be used in a live test, at the beginning part of the second semester within the CS&IS Department; as long as the department's ethics officer approves the test. The system will potentially be live for a matter of a few days in which data will be collected containing all conversations. The data collected will be used test analytics tools; helping to determine which forms of analytics are suitable for the data being provided, does the data prove to be useless in certain analytics tools researched? Or does it show something unexpected? This will be discussed in the final report. At the end of the trial run the users will be given an online questionnaire or alike allowing the users to give feedback on how the system can be improved for the final version of the system. This hopefully will bring some new interesting, unpredicted features to the table that could play an important part in the final version of the system. Some predictable feedback may ask for pseudo anonymity to help with conversations readability and additional methods of access such as email. Some users may like the analytics used on the display and may encourage more analytics.

During the time of the live test, the user map tool will come into real-world use. The trial may help highlight ways in which new users can be added in a faster and efficient fashion. Moving away from a more manual system to a more automated system, which still has a strict authorisation process found in the manual system.

Decisions will need to be made before the first prototype is created. We know that data is going to be captured and later analysed but a question remains. How will the data be stored?

- In a raw format in a text file?
- Structured and easily accessible and portable through XML?
- Alternatively the data collected could be stored in a database, Oracle or MYSQL?

Again these questions will be discussed in the final report where another phase of research and decisions will be made to determine the most suitable data storage mechanism. The data needs to be stored as it will reduce the amount of API calls to Twitter and allow data analysis.

Before the system goes live, moderation tools will need to be added to monitor for profanity, this will ensure users of the system are not offended and will most definitely need to be implemented in order to be approved for a live test in the CS/IS department.

Further questions need to be answered, what analytics or techniques will there be in the first prototype to create the 'hook' for current users and to attract attention to bring in new users? Will it be a Wordle? Or will a Wordle provide nonsense to the screen and prove to be useless? A dummy run before the live test needs to be conducted to determine how useful a Wordle is; this will be tested through a collection of conversations or feedback. If it is useful how will it be displayed? Will writing to a local public html folder (discussed previously) prove to be flexible or would it be better to upload the images by FTP/SFTP.

Will the Java application be extended to have a GUI or will it remain as a process that collects data to be accessed through another medium? If it does remain as a process; the application would become more distributed and will use an array of applications to access the data for different actors of the system.

Once decisions have been made; the project will go into a detailed design phase utilising UML and pseudocode to represent the systems functions, communication and processes. The use cases designed so far will be updated according to the new features that will be added into the system.

References

Android. 2011. What is Android? Available at: http://developer.android.com/guide/basics/what-is-android.html [Accessed 1 November 2011]

Facebook. 2011. Terms. Available at: http://www.facebook.com/legal/terms [Accessed 31 October 2011].

Google. 2011. Your Name and Google+ Profiles. Available at: http://support.google.com/plus/bin/answer.py?hl=en&answer=1228271 [Accessed 31 October 2011].

iWall. 2007-2011. Home. Available at: http://www.iwall.com.ar/index.en.html [Accessed 24 October 2011].

Feinberg.J, 2011. Home. Available at: http://www.wordle.net/ [Accessed 20 November 2011].

Selenium. 2011. Index. Available at: http://seleniumhq.org/ [Accessed 2 November 2011].

Text2Screen. 2010-2011. Overview. Available at: http://www.txt2screen.co.uk/text-messaging-software-overview.htm [Accessed 24 October 2011].

Twitter. 2011. Getting Started with Twitter for SMS. Available at: http://support.twitter.com/forums/23786/entries/14589 [Accessed 30 October 2011].

Twitter. 2011. Terms of Service. Available at: http://twitter.com/tos [Accessed 31 October 2011].

Twitter Sentiment. 2011. General Information. Available at: https://sites.google.com/site/twittersentimenthelp/ [Accessed 20 November 2011].

Appendix A

FireText – Text2Screen system, which has moderation tools in place to moderate users and profanity. System has no analytics and private messaging http://www.firetext.com/

TextDemon – Text2Screen system, which has moderation tools in place to moderate users and profanity. http://www.textdemon.com/

Text Messaging Live - Text2Screen system, which has moderation tools in place to moderate users and profanity. http://www.textlive.com/

```
              Appendix B
package TweetBot;

import twitter4j.Status;
import twitter4j.Twitter;
import twitter4j.TwitterException;
import twitter4j.User;
import java.util.*;

public final class Main extends Thread {
  private TwitterAuthenticate twitterAuth;
  private Twitter twitter;
  private User user;
  private Timeline timeline;
  private List<Status>homeStatuses,userStatuses;
  private Thread t;
  private static final String CONSUMERKEY = "9Wy2GeJ8x6CYq8dtuKvSnw";
  private static final String CONSUMERSECRET =
"PXc7jD8tyCKiKMEWvBSZ4SiXoylRArZDM6exQHpoYs";
  private static final String ACCESSTOKEN =
"407325608-lhw5hBuVKKdjD4Q73PV4E7kC3jQESAlMKaFyTWIp";
  private static final String ACCESSTOKENSECRET =
"yraW8ZEoJVhlkrroWxiDKRLMf2HCBUu1XUZECzj7mU";
  public static void main(String args[]) {
      new Main().start();
  }
  public void run(){

          twitterAuth = new TwitterAuthenticate(CONSUMERKEY, CONSUMERSECRET,
ACCESSTOKEN, ACCESSTOKENSECRET);
          twitter = twitterAuth.getInstance();
          User user = null;
          try {
               user = twitter.verifyCredentials();
          }catch(TwitterException te){
              te.printStackTrace();
              System.out.println(te.getExceptionCode() + "this is the error
code\n");
              System.out.println("Failed to get timeline: " +
te.getMessage());
              System.exit(-1);
          }
          timeline = new Timeline(twitter);
          UpdateStatus updateStatus = new UpdateStatus();
            try {
```

```
          while (true) {
                      List<Status> homeTimeline = timeline.getHomeTimeline();
                      //iterate through results and check whether tweet has
been tweeted by a user, if so then tweet onto Bots wall.
                      for (Status statusI : homeTimeline) {
                          if
(!statusI.getUser().getScreenName().matches(user.getScreenName())){
                              updateStatus.update(statusI,twitter);
                          }
                      }
                      Thread.sleep(30 * 1000);
          }
      } catch (InterruptedException e) {
          e.printStackTrace();
      }
      }

  }
}
package TweetBot;

import java.util.List;
import twitter4j.Status;
import twitter4j.Twitter;
import twitter4j.TwitterException;
public class Timeline {

    private Twitter twitter;

    public Timeline (Twitter twitter){
        this.twitter = twitter;
    }

    public List<Status> getHomeTimeline(){
        List<Status> temp = null;
        try {
            temp = twitter.getHomeTimeline();
        } catch (TwitterException te) {
            te.printStackTrace();
            System.out.println(te.getExceptionCode() + "this is the error
code\n");
            System.out.println("Failed to get timeline: " + te.getMessage());
            System.exit(-1);
        }
        return temp;
```

```
    }
}

package TweetBot;

import twitter4j.Twitter;
import twitter4j.Status;
import twitter4j.TwitterException;

public class UpdateStatus {
    public UpdateStatus(){

    }
    public void update(Status status, Twitter twitter){
        try {
            System.out.println("Detected: " +
status.getUser().getScreenName()+": " + status.getText());
            twitter.updateStatus(status.getText());
            //No error returned in update therefore updated fine.
            System.out.println("Sent: CSTweetBot:"+ status.getText());
        } catch (TwitterException ex) {
            System.out.println("error Updating Status");
        }
    }
}
```

Appendix C

```
<!--index.php-->
<html>
<head>
    <title>Map Users - Twitter Style App</title>
    <style>
    .default{
        text-align:center;
        float:left;
        margin-left:5px;
        margin-top:5px;
        background-color:#AFC7C9;
        font-family:calibri;
    }
    .search{
        text-align:left;
        width:100%;
        background-color:#2DA8D3;
        font-family:calibri;
    }
    </style>
</head>
<body>
<div class='search'>
    <form name="search" method="post" action="index.php">
        Search TwitterID<input type='text' name='search'
id='search'width='25'/>
        <input type='submit' name='submit' value="Search">  <a
href="index.php?viewmap=true">View Map</a>
    </form>

<?
//Search
include_once('connect.php');
if($_POST['search']!=null)
{
    $id = $_POST['search'];
    $result = mysql_query("select * from UserMap where TwitterID='$id'");
    $n = mysql_num_rows($result);
    if($n==1){
        print "<div class='default'><H1>Search Returned</H1><br/>Twitter
ID:<br/> ".mysql_result($result,0,"TwitterID")
        ." <br/>Forename:<br/> ".mysql_result($result,0,"Forename")."
<br/>Surname: <br/>
        ".mysql_result($result,0,"Surname")."<br/>Email:<br/> ".$email."
```

```
        <br/><a href='delete.php?email=$email'>Delete</a></div>";
    }
}

?>
</div>

<?
if($_REQUEST['viewmap']==true)
{
    $result = mysql_query("select * from UserMap");
    $n = mysql_num_rows($result);
    for($i = 0; $i<$n; $i++){
        $email = mysql_result($result,$i,"Email");
        $twitterid = mysql_result($result,$i,"TwitterID");

        print "<div class='default'>Twitter ID:<br/> ".$twitterid
        ." <br/>Forename:<br/> ".mysql_result($result,$i,"Forename")."
<br/>Surname: <br/>
        ".mysql_result($result,$i,"Surname")."<br/>Email:<br/> ".$email."
        <br/><a href='delete.php?email=$email'>Delete</a></div>";
    }
}
?>
<div class="default">
    <form name="add-user" method="post" action="add.php">
        <H2>Add User</h2>
        Forename<br/><input type='text' name='forename'
id='forename'width='25'/><br/>
        Surname<br/><input type='text' name='surname'
id='surname'width='25'/><br/>
        TweetID<br/><input type='text' name='tweetid'
id='tweetid'width='25'/><br/>
        Email<br/><input type='text' name='email' id='email' size='25'/><br/>
        <input type='submit' name='submit' value="Add to Map">
    </form>
</div>
</body>
</html>

<!--add.php-->
<?php
    include_once("connect.php");
    $forename = $_POST['forename'];
    $surname = $_POST['surname'];
```

```
    $twitterid = $_POST['tweetid'];
    $email = $_POST['email'];
    $q = "INSERT INTO UserMap (email,forename,surname,twitterid)
VALUES('$email','$forename','$surname','$twitterid')";
    mysql_query($q);
    header("location:index.php");
?>
<!--delete.php-->
<?php
include_once("connect.php");
$email = $_REQUEST['email'];
//$tweeterid = $_REQUEST['tweeterid'];
$q = "DELETE FROM UserMap WHERE email='$email'";
mysql_query($q);
header("Location:index.php");

?>
```

Appendix D

```
package externaldisplaydemo;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;


public class Main extends Thread{

    FileWriter fstream;
    BufferedWriter out;
    String fileName = "web/index.html";//MAMP server points to
.../ExternalDisplay/web
    boolean flag = true;
    public static void main(String[] args) {
        new Main().start();
    }
    public void run()
    {
        int i=0;
        while(flag){
            i++;
            try {
                fstream = new FileWriter(this.fileName);
            } catch (IOException ex) {
                Logger.getLogger(Main.class.getName()).log(Level.SEVERE,
null, ex);
            }
            out = new BufferedWriter(fstream);

            System.out.println("Changed page!");
            try {
                out.write("<h1>" + i + "</h1>");
            } catch (IOException ex) {
                Logger.getLogger(Main.class.getName()).log(Level.SEVERE,
null, ex);
            }
            try {
                out.close();
            } catch (IOException ex) {
                Logger.getLogger(Main.class.getName()).log(Level.SEVERE,
```

```
null, ex);
            }
            try {
                Thread.sleep(60 * 1000);
            } catch (InterruptedException ex) {
                Logger.getLogger(Main.class.getName()).log(Level.SEVERE,
null, ex);
            }
        }
    }
}
```