Team **The MACK**

Chinyere Isaac-Heslop

Taylor Wacker

Andrew Suter-Morris

CONNECT

I. Abstract

The original CONNECT (Creating Open Networks aNd Expanding Connections with Technology) project was designed to promote conference attendee interaction. It allowed these attendees to "CONNECT" with one another, or rather, share their profile information. This information consisted of a user's name, organization, conference role, interest area(s), phone number, and email. The aforementioned information would traditionally be found on a business card. Attendees could send a connection request by entering an id and a comment. The original project consisted of a website, an Android application and support for text messaging, commonly known as SMS (short message service). Using these tools attendees were able to make connections. The website allowed users to search for other attendees, to make connections, confirm connections and to view all of the connections made during that conference. The Android application also allowed the users to search for other attendees and to initiate connections via id(s) and an optional comment. It also had support for Bump which allows users to essentially fist "bump" their phones to transmit the same information. Our goal was to create a platform for CONNECT on the iPhone as well as provide a proof of concept for long term social networking implementation with Facebook and/or LinkedIn interaction with CONNECT.

The focus of our project was creating an iPhone implementation of the existing CONNECT project. It needed to utilize many of the same features as the Android platform version, some being "bump" functionality and FAYT (Find As You Type) .The iPhone program was written in Objective-C using the xCode IDE (Integrated Development Environment). We also analyzed pre-existing social networking sites and conceptualized a way to integrate them with this short term networking application so that CONNECT could have more longevity in the professional process for connecting

individuals. This was done by inserting a Facebook friend request (“add friend”) button to the corresponding contact on the CONNECT web site's contacts section. The desired social network integration proof of concept and the iPhone platform were achieved.

It is important to understand some terms and concepts whilst reading this document. Please refer to the Glossary for definitions of acronyms, explanations of unfamiliar terms and concepts. There are seemingly simple words like “connection” and “bump” that are used in a particular and meaningful way. The Glossary is located at the rear of the paper.

II. Introduction

Our client was Dr. Cynthia Rader. She acquired both her undergraduate and graduate degrees in computer science from Wright State University in Ohio. Rader, as her students often call her, received a PhD from CU Boulder before becoming a professor at CSM.

Rader and a team of CSM students have been working on the CONNECT project before us. Initially an endeavor focused on interacting with women and minorities, CONNECT was used to allow conference attendees to “connect” to one another. Paralleling the trading of *old school* business cards, CONNECT contacts shared their profiles and could get each other’s specified information via website or mobile interaction.

III. Requirements and Specifications

A. System Architecture

The broad CONNECT process starts with the user. The user enters CONNECT with an Android phone, an iPhone, a website, or any phone with text capabilities. The android and iPhone apps go through the CONNECT API (Application Programming Interface) by sending HTTP (Hypertext Transfer Protocol) requests and receiving JSON (Javascript Object Notation) responses. The API interacts with the Toilers server, which interacts with the CousinIt server. The CousinIt server houses the MySQL database that has all the information we need to give back to the end user. The SMS way interacts with the CousinIt server in much the same way but through a separate backend. There

is also a website that can access all of the information as well. Fig. 1 below is a visual representation of the aforementioned interactions.

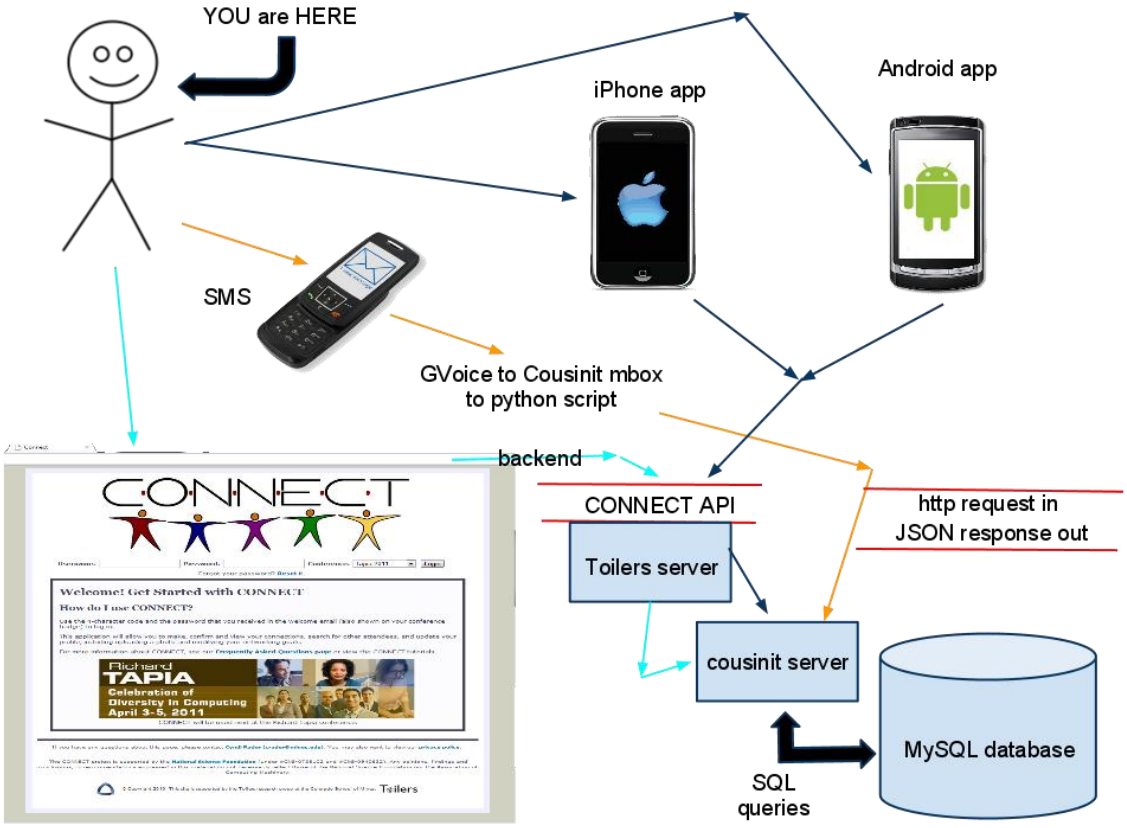

Figure 1: System Architecture of Application

The system architecture diagram is a high level representation of the interactivity of each major device or point of access to CONNECT. Our project for CONNECT is focused simply on the iPhone point of access and the Facebook or LinkedIn gateway (not pictured).

B. Functional Requirements

The system needs to have the following functionality for the iPhone app:

- log in with provided CONNECT ID and password
- store log in information in iPhone settings
- enter a connection to specified IDs
- enter a comment for IDs
- confirm connection

- “bump” with other users to perform connections
- FAYT to allow for easy searching of users to add

The system needs to have the following functionality for Facebook/LinkedIn

- proof of concept for sending friend request (“add friend”) button

C. Nonfunctional Requirements

- Data is stored in a MySQL database - schema is already defined. We do not have access to this database
- Application is constrained to just being written in Objective-C
- Only permitted to be written through xCode
- Must be a member of Apple’s yearly developer program subscription to host on “App Store”
- Must make use of pre-existing CONNECT API and social networking APIs
- Have to use HTTP for outgoing and JSON for requests.
- Must be developed in Mac OS X
- We only have six weeks for this project.

IV. Design

UML Diagram

Figure 2 shows all of the classes involved in the project. Without going into too much technical detail, there are a few types of classes. The controller classes implement all of the graphical interface and its calls. The delegate class handles all of the controllers. The remaining classes are helper classes that make all of the API calls.

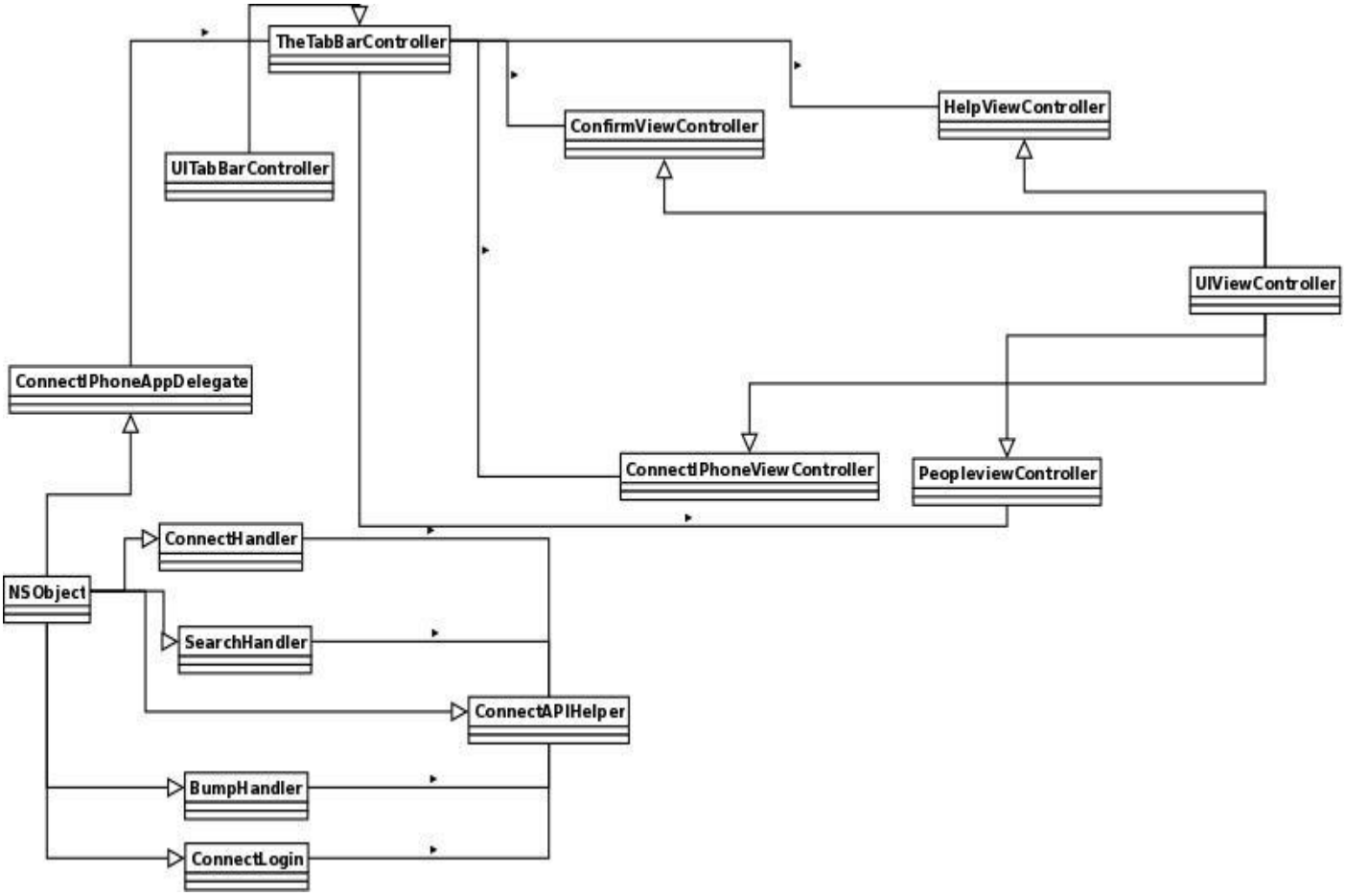

Figure 2: UML of Diagram

Database Schema

The following is the schema of the database for the CONNECT project that is relevant to our particular subset. Included are the three of the tables utilized; the connection_ids table, the connection_members table and the persons table. We did not develop the database, but these are the tables we had to use.

connection_ids

id – primary key for the connection_ids table

time - the time that the connection was made

comment - the comment left for the connection

connection_members

id - primary key for the connection_members table
connection_id - this is an attribute that matches a row in the connection_ids table. Also known as a foreign key.
person_id - this is an attribute that matches a row in the persons table
status - this is the status of the connection
client - this is the client of the connection

persons

id - primary key for the persons table
first_name - first name of the person
last_name - last name of the person
organization - organization that person works with
email - email of the person
phone - phone number of the person
webpage - webpage of the person
image - profile photo of the person
address - where the person is located
zip - see address above
city - see address above
state - see address above
country - see address above
hashpass - a SHA-1 hash of the password the person made
reset_code - code for reset
reset_expire - when person expires, uses date/time.
admin - flag for if the user is an administrator
role_id - reference to roles tables, person can pick role and get the whole row
regid - registration id for user. Used by 3rd party system
regstatus - registration status of user. Active means user can log into CONNECT

Components and Major Milestones

There are several components to this project: The GUI(Graphical User Interface) -includes the keyboard and rotation support- see VII. Section for figures, “bump”, Find As You Type, connection and Facebook/LinkedIn.

The development of each component relies on existing libraries, a part of the iPhone Framework. The Bump and CONNECT APIs were also utilized to ensure full functionality. The Stig-JSON framework was used to handle the request and response system. All of these libraries and APIs have support for the iPhone.

The GUI component consists of what the application actually looks like. There were no additions in terms of API calls or any other advanced or CONNECT related code. The GUI allows the user to use the keyboard, as well as rotate the device into multiple orientations. A user should be able to use the app in a landscape or in portrait position.

The connection component is the part of the project that will deal with approximately 90% of our Connect API calls. It handles the connection through the API to the database. It sends HTTP requests and returns with JSON responses for parsing. The http requests were made via an iPhone library – NSURL – and the JSON responses were parsed and returned using the Stig-JSON framework. The calls we used are:

- make_connection – creates a connection between a user and one or more other users
- check_password – checks and authenticates the user
- confirm_connection – ensures that one user can confirm a connection for an added level of security
- view_connections – view all pending and confirmed connections
- get_name_list – get all of the names for searching

All of the aforementioned calls were used for the project. There were no additional calls needed. All of them use HTTP requests and JSON responses to send and pull the necessary data. See Figure 3 below for an example.

Request
http://toilers.mines.edu/connect/webapi/get_name_list?id=il01&hashpass=[redacted sha1]&q=cyn

id=il01	ID of the user
hashpass=[redacted sha1]	password of the user
q=cyn	parameter for the search

Response


```
{
    "count":3,
    "path":"http:\/\/toilers.mines.edu\/connect\/pics\/Tapia2011\/",
    "contacts":[
        {"id":"CL02","name":"Cynthia Lanius","pic":""},
        {"id":"CP02","name":"Cynthia Prudence","pic":"CP02.jpg"},
        {"id":"CR02","name":"Cyndi Rader","pic":"CR02.jpg"}
    ]
}
```


Figure 3: A HTTP request and the resulting JSON response

Following the connection component is the “bump” component. The API for Bump only needed to transmit an ID to the other device. It did not need to interact with the CONNECT API.

The FAYT was also implemented. The CONNECT API’s get_name_list call supports this method of searching. This component connects to the database and queries the database. The call was made for a Javascript FAYT style. Per the Toilers Wiki, no additional calls were needed.

The last major development milestone is the Facebook/LinkedIn component. The goal of this was to extend the lifespan of a connection and to enhance this social networking experience. This required adding a friend request button on the CONNECT website. This did not require any additional API calls. Time did not permit us to make an actual Facebook/LinkedIn application. It would, however, only need the make_connection, check_password, and get_name_list API calls that have already been implemented.

Use Cases

1) New User to CONNECT wishes to establish connection

Goal: A conference attendee, Chin, has just received her email with her CONNECT ID and her password.

Actors: Chin, Attendee #2, Attendee #3

Preconditions: Automatic email with log in details sent by CONNECT script

Trigger: Chin meets another attendee that wants to “connect”:

Events:

1. Chin meets another attendee with some like-mindedness, she wants to stay in touch with attendee #2
2. Chin pulls out her trusty iPhone, she however, forgot to enter her credentials from the email she received
3. She enters them, never having to do so (at this conference anyways) again.
4. She asks for attendee #2’s id, and is given it
5. She adds that into the box, and adds a comment, “met at Informational Rugby Coaching Conference - fellow coach, time to steal some coaching strategies”
6. Attendee #2 confirms this connection
7. Chin never actually meets Attendee #3 but decides that based on her profile information that she wants to try and meet up
8. She uses the find as you type search method on the iPhone to bring up Attendee #3’s information again so she can make a connection
9. She adds a comment about similar interests
10. Attendee #3 confirms this connection

Alternate Flow:

1. ID doesn’t exist, or ID given was false.
    1a. Program gives ID not found error, try again
2. Either attendee declines confirmation
    2a. Chin receives declined message, and goes off to pout

Post Conditions: Both Chin and the attendee can now check each other’s phone numbers, and websites to stay in touch

2) Pre-existing user on CONNECT

Goal: A conference attendee, Taylor has been using the CONNECT app on iPhone and just now notices Bump. With the number of connections he makes, he wants to start using Bump because his fingers are getting tired.

Actors: Taylor, Attendee #2

Preconditions: Taylor has already registered with CONNECT and has the iPhone app set up.

Trigger: Taylor's fingers are tired and he wants to use a different method to connect

Events:

1. Taylor's fingers are about to fall off, but he remembers seeing the Bump button on the application.
2. Taylor meets yet another attendee he wants to connect with
3. He uses the last of his strength in his fingers to press the button to initiate Bump
4. The other attendee who has just arrived feels like being nice and hits the Bump button as well
5. Both users then fist "bump" and they just made a connection
6. Connection was confirmed via Bump

Alternate Flow:

1. Bump did not work correctly and missed the connection

1a. Taylor's finger falls off, but he uses another one to initiate Bump again

2a. Once Bump works properly, connection is made

Post Condition: Taylor and attendee # 2 can now view each other's information. They can grab phone numbers and other vital contact information to stay in touch.

3) Pre-existing CONNECT User wants to add multiple connections

Goal: A conference attendee, Andrew, has just met a group of individuals that he wants to befriend/connect with because he likes to network

Actors: Andrew, Attendee group

Preconditions: Everybody has CONNECT setup, on any number of mobile devices

Trigger: Andrew walks into a group of Mobile Development Junkies

Events:

1. Andrew meets another group of attendees with some like-mindedness, he wants to annoy them all
2. Andrew pulls out his old iPhone 4, and and grabs all of their IDs
3. He adds a comment saying “best friends who also like mobile development”
4. All of the receivers receive the confirmation message and approve (or deny if they’re mean)

Alternate Flow:

1. ID doesn’t exist, or ID given was false.
    1a. Program gives ID not found error, try again

Post Conditions: Both Andrew and the group of attendees can now check each other’s phone numbers, and websites to stay in touch

4) User is familiar with Facebook and wants to make a connection this way

Goal: A conference attendee, Andrew, has just met a group of individuals that he wants to befriend/connect with because he likes to network, but he doesn’t actually have a phone for some odd reason

Actors: Andrew, Attendee group

Preconditions: Everybody has CONNECT setup

Trigger: Andrew walks into a group of Mobile Development Junkies

Events:

1. Andrew meets another group of attendees with some like-mindedness, he wants to annoy them all
2. Andrew writes down their IDs or information with pen and paper (What’s that?)
3. He goes to his hotel room after the conference and logs in to the Facebook Connect application. He adds the IDs.
4. All of the receivers that have not signed up for the Facebook Connect app receive an application request.
5. Users confirm and now they may share information via Facebook.

6. This leads to everybody friend requesting each other (note: this may have to come prior to application request)

Alternate Flow:

1. User declines Facebook connect request

1a. No information is shared

2. User declines friend request

2a. Again, no information is shared

5) Use case for LinkedIn is similar to Facebook;

V. Implementation and Results

To make an iPhone application developers are limited to using a Mac OS machine with xCode IDE and the language of Objective-C. An existing iPhone application was created prior to our involvement in CONNECT. We reviewed the functionality of this app, and ultimately decided to use it only for reference. The primary reasoning behind this was that it was using the deprecated ThreeTwenty library. We scrapped the ThreeTwenty library in favor of the built in Cocoa and Objective-C libraries developed by Apple. Team The MACK also chose the JSON framework due to JSON being a return format for our HTTP request. One of the difficulties we had with this project was getting the HTTP requests to work properly. The issue that we were running into was that the request wasn't being formed properly. We came to a solution by simply switching the formatting of the requests around until it worked. The other difficulty we had was getting the iPhone tables to refresh properly. Unfortunately, Objective-C tables do not work well with other components, such as constant refreshing of the data. This issue was resolved by refactoring the code, and implementing a few additional methods to allow for proper table reloading.

The major requirements for our project to be considered a success were to have an iPhone application that could

- make a connection request
- view current request from others
- confirm/ignore their request

- include “bump” functionality
- implement FAYT
- demonstrate how to integrate Facebook into the website with an add friend request button

We successfully completed each of these requirements.

The make, view, and confirm connections processes were tested by having a user online to verify what happened while someone was using the iPhone simulator. We had the simulator send a request and verify that it was picked up on the website under the confirm connection page. To view connections we used the demo database to submit a batch of requests and on the simulator we checked to make sure it was capable of viewing them all. Then we would respond to the request by confirming or denying them. Online, if the request was confirmed, they would now be visible in the confirmed connections page. If the request was denied, they would not appear. Bump was not strictly tested. Unfortunately, Apple requires a developer subscription to test on a physical device. This subscription was not obtained. The project does compile and work with no issue from the Bump API. FAYT was tested by typing the same search constraints into both the simulator and the webpage to verify that both produced the same result. For Facebook we created a demo HTML page with the coding to allow us to friend request each other. See Figure 4 below for sample code.

```
<form target="_blank" action="http://www.facebook.com/
addfriend.php">
<input name="id" type="hidden" value="1296301596" />
<input type="image" value="Connect with me on Facebook!"
src="addmefb.png"
style="width:111px;height:22px;margin:0px;padding:0px;" />
</form>
```

Figure 4: Facebook “add friend” HTML insert

VI. Scope and Project Progression

The original scope of work was to develop an iPhone application with “bump” and FAYT functionality. It was also highly recommended to develop some proofs of concept for Facebook or LinkedIn.

The scope was modified shortly after to having the confirming, viewing and making a connection while having secondary, but less important goals like “bump” and

FAYT. The Facebook and LinkedIn requirements stayed the same. All of these goals were met. The iOS application makes connections, confirms, views, does FAYT and supports “bump”. Team The MACK also produced a report on how to implement Facebook on the current CONNECT website.

VII. Future Directions and Conclusions

This project taught our team many things. We learned Objective-C, “bump”-ing, FAYT, generating HTTP requests, parsing JSON responses and of course how to troubleshoot all of this. While we did finish the primary and secondary goals of field session, there are still many things that can be added.

We hope that this project will be expanded and improved. The ability to allow the end user to switch conferences is probably the biggest one. The “confirm” and people screens should both pull profile photos to help spur the memory of who is being “confirmed”. Beyond that, if a user confirms somebody, they should be able to see all of their information right on the phone. Offline support would also be a huge desire too. If a user doesn’t have cell signal or a wireless connection, they should still be able to see the people they have confirmed.

VIII. Figures

Figure 5: Log in, create and make connection screens

Figure 6: Confirm and view connection screens

Figure 7: Find-as-you-type screen

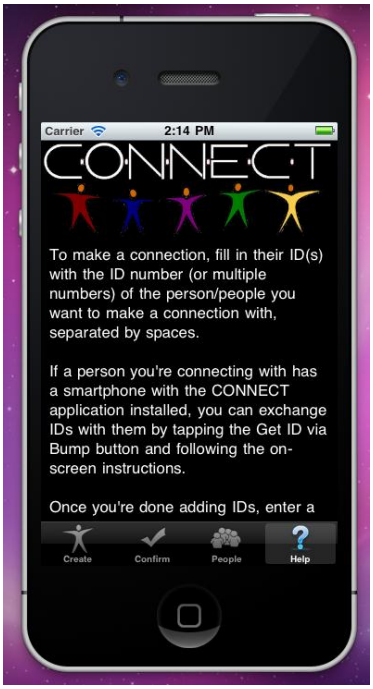

Figure 8: Help Screen

IX. Glossary

API - Application Programming Interface - set of specifications that allow software programs to “communicate” with one another.

App - A shortened version of application, typically used for mobile development

Bump - An API developed for iOS and Android devices to allow “bump”-ing

“bump” - Transfers information between mobile devices when two users perform a fist “bump” using the Bump API

CONNECT - Creating Open Networks aNd Expanding Connections with Technology. The name of the project.

connection - A friend request of sorts. It consists of a request from one person to another that needs to be confirmed so personal information may be shared.

CSM - Colorado School of Mines
Find-as-you-go - a technology used in the CONNECT project where every time the query changes (even by a letter), it updates the search and limits or expands the name list

FAYT - Find As You Type

GUI – graphical user interface, what the user sees.

HTTP - Hypertext Transfer Protocol - Networking protocol that serves as the foundation of data communication on the World Wide Web.

IDE - Integrated Development Environment - Comprehensive application to aid software developers with tools such as a compiler/interpreter, debugger, a source code editor and a set of build automation tools.

iOS - Apple's mobile operating system used on iPhones, iPod Touches and iPads.

JSON - JavaScript Object Notation - standard for human data interchange. It serves as an alternative to the extensible markup language (XML).

OS X - Apple's latest Operating System. Currently on the 10.6, the "Snow Leopard" version.

SMS - short message service - text communication service most commonly used on mobile phones.

UML - Unified Modeling Language - Modeling language used by object oriented programmers to visualize the layout of a program.

xCode - IDE for Apple products including iOS, and Mac OS X.

X. References

Bump API http://bu.mp/api

Connect Website FAQ's http://toilers.mines.edu/connect/help

Facebook API http://developers.facebook.com/docs/guides/web/#login

LinkedIn API http://developer.linkedin.com/docs/DOC-1207

Toilers Wiki Connect http://toilers.mines.edu/twiki/bin/view/Toilers/ConnectProjectAPI

Toilers Wiki iOS http://toilers.mines.edu/twiki/bin/view/Toilers/ConnectIosClient

Toilers Wiki Android http://toilers.mines.edu/twiki/bin/view/Toilers/ConnectAndroidClient