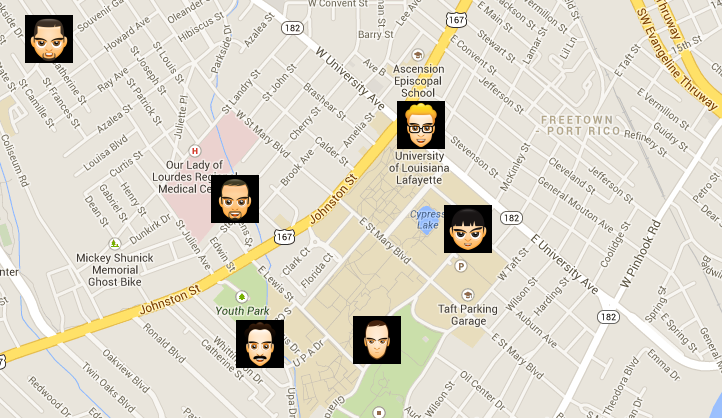
******

PROJECT REPORT

**Pingoin Android Application**

Submitted to Dr. Hongyi Wu, Professor at University of Louisiana, Lafayette

**Prepared by Group ‘D’ – CSCE 576, Fall 2013**

Members: Shailendra Gaikwad, Romas J Hada, Tung T Le, Madhukar Reddy

**Acknowledgements**

We would like to thank Dr. Hongyi Wu, our professor for this course, who gave us this opportunity to create this interesting project, and provided us his time and the tools to complete our project properly.

I also would like to thanks Mr. Elman Bashar, our Teaching Assistant (TA) of the course, because of the excellent documentation and tools introduced by him for creating an Android project.

Next we would like to thank University of Louisiana, Lafayette for promoting such innovative ideas for creation as projects by the students.

Finally, I would like to thank all the members of my Group ‘D’ for their untiring efforts, and co-operation in successfully completing this project.

**Abstract**

This report will discuss the development of *Pingoin,* an android application (app) designed to provide information about the places of interest (POI) a user wants to visit. It could be user entered data or public tweets from internet. It will also provide the ability to store personal updates about the location as private tags as well as post public tweets. When the user is visiting those places, Pingo’in will provide notifications/popups for places that have been tagged (in the vicinity of - predefined range for Pingo’in). The places of interest could be supermarkets, bars, restaurants, theaters, etc. with tags providing information such as discount prices, big sales, live shows, etc. It could also be public data from Twitter users.

The system consists of a client-server architecture, and a two-way GET, PUSH of twitter data over HTTPS. It also involves tagging locations on google maps. The tracking is done with the use of *map-tags,* a term describing precise location (predefined radius), text description, and images associated with a single tag on the route of *user’s* journey/travel.

The system requires internet access for GET and UPDATE the tag information, however it accounts for limited internet connectivity by providing ability to save information of targeted area, prior to actual travel/use, on the android device itself.

The *app* is a fully operational software tested on both the Android virtual device as well as physical Android devices viz. Samsung Galaxy Nexus, and Samsung Galaxy Note II.

**TABLE OF CONTENTS**

1. Introduction 5

1.1 Aims & Goals 5

1.2 New Skills and Management 5

2. Background Information 6

2.1 The Pingoin Project 6

2.2 Target Users of Pingoin 6

2.3 Projects similar to Pingoin 6

3. Overview of Features 7

3.1 Requirements definition 7

3.2 Requirement Analysis 8

4. System Architecture and Design 9

4.1 Java Classes 9

4.2 Data Transfer 9

4.3 User Interface Design and Layout 9

4.4 Local Data Storage 11

4.5 Class Diagram 11

4.6 Sequence Diagram 11

5. Testing 13

5.1 Tools 13

5.2 Testing the main features 13

6. Future Development 15

6.1 Measurement of Success for targeted purpose 15

6.2 Challenges faced 15

6.3 Evaluation 15

6.4 Future Enhancements to the app 16

7. Achievements 17

7.1 Technological 17

8. Project Management 18

8.1 Roles and Responsibilities’ of the team members 18

9. Appendices 19

9.1 Glossary 19

9.2 Procedure for app Installation 19

10. Bibliography 21

# Introduction

Pingoin

* Android App
* Location based content of interest
* Updates to/ from twitter
* Pop-ups of information for places of interest
* Tagging of location on google maps
* Update/ upload information of places of interest – need internet connectivity
* Information also stored on local storage of Android device

## Aims & Goals

Many popular web based and Android applications exist which provide location based information of interest to the user, but none allow the user to customize the information and the locations of interest available as an Android *app.* Our *app* aims to provide this information in a hassle free manner to a novice user with little or no experience of computer systems.

Since an Android device is being used to run the *app*, our goal was to utilize the device’s hardware capabilities like GPS receiver, camera, and wifi/3G/4G data connection along with ready Application Programmable Interfaces (APIs) of popular web based tools like google maps and twitter to provide functionality needed upon the project *app*’s operation.

## New Skills and Management

Since this was a class group project, the aim was also to develop technological and organizational skills involved in building such a software *app*. We had to learn use of tools like ‘Eclipse’ with Android tools, use of APIs with authentication requirements of google maps, and twitter, android programming with some java skills, and project development and management techniques. Due to the amount of new skills that had to be learnt before we could proceed with development/ coding mostly of Android, Java Servlets and HTTPs, we had to allocate the work in sub-groups to utilize our time efficiently. We also had to prepare a Gantt chart to keep track of our progress effectively. The project had very slow initial progress as the functionalities to be implemented were debated upon and modified to its present day availability. Most of the information of what to implement and how to implement was obtained through very good documentation provided by our TA.

# Background Information

## The Pingoin Project

## Target Users of Pingoin

## Projects similar to Pingoin

1. Web based:
2. Android Apps:

# Overview of Features

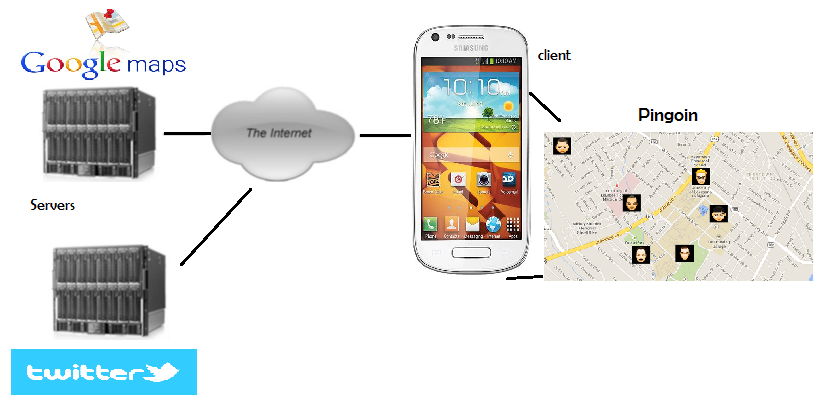
## Requirements definition

\*The priority labels M, S, C, W stand for “must have”, “should have”, “could have”, “would like to have” the requirement respectively.

|  |  |  |
| --- | --- | --- |
| **ID** | **Description** | **Priority\*** |
|  | **App as Client** |  |
| C1 | Provide means for inputting data into the system from a basic Android phone with Internet connectivity. | S |
| C2 | Provide access to twitter location based public data on the Android mobile phone | C |
| C3 | Use GPS tracking technology that collects data about the Android phone’s location, and observations time and description | M |
| C4 | Provide means for using google maps via API from the Android phone for tagging location on it | M |
| C5 | Provide technology accessible to a computer illiterate user | M |
| C6 | Use twitter API to login and fetch location based data onto the Android device | S |
| C7 | The system shall run as autonomously as possible. | S |
| C8 | The data will be recorded automatically in Android database on device. | S |
| C9 | The data can also be recorded manually, and include a location, description, and an image | C |
| C10 | The data can also be saved as Twitter public location based tweets on the twitter server | C |
| C11 | The system shall save its state if there is no Internet connectivity available | M |
| C12 | Multiple users of the app on the same device will have separate login credentials | M |
| C13 | Different users of the app on the same Android device will have their private data stored and accessible separately | M |
| C14 | Different users of the app on the same Android device can share their public data stored on the device | C |
|  | **Server** |  |
| S1 | The Twitter server shall support Java Server Pages. | M |
| S2 | The Twitter server shall return product-related location based data at the device’s app request. | M |
| S3 | User authentication credential data server shall support Java Server Pages | C |
| S4 | User authentication data server shall use a MySQL database to store app user login credentials as well as data created by user | C |
| S5 | Synchronize periodically with the app on user’s Android device. | C |
| S6 | It should be accessible 24x7 | C |

## Requirement Analysis

1. Challenges in software design was to understand..
2. How Android platform makes a difference
3. API use and GPS functionality
4. Limitations: Device has to be switched on, and the app has to be running for the google maps tags to be visible and updated. Moreover the twitter user has to be logged in for the query to fetch for location based tweets.

Diagram of the client-server architecture

Because of the limitations, and challenges faced following assumptions were made in software and database design:

1. Thw
2. Fgf

Thrtt

1. Connecting to Google Maps server operation
2. Connecting to Twitter server operation.
3. Connecting to user login server (optional)
4. Autonomous operation

# System Architecture and Design

## Java Classes

The following java classes were used in the project:

1. Connection Detector:
2. Alert Dialog

Concept Map diagram

## Data Transfer

HTTPS was the main means of transferring data between the Android client and the servers; classes from the API libraries for google play services, and twitter4j were used to prepare, send, and process GET/ POST requests. The transferring of messages was thus achieved by using abstraction layer by the use of REST APIs.

The transfer mechanism works as follows: the location data is fetched using google API and inserted into \_\_object, it is then converted to an OutputStream and passed to object \_\_\_\_\_\_\_\_\_\_\_\_\_. This is used in a GET request with twitter API as well as for POST request. This OutputStream data is also used to create \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_tags\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_mark position on google maps. SSL based server authentication (using API classes) was applied in this project for user login security as well as data transfer security.

## User Interface Design and Layout

We originally built two different applications: one system asking the user upon start for login information, and upon successful authentication to display current user location on map as well as previously entered data for user POIs. It also had the feature of tagging new POIs, updating, and removing them. The second app handled the pingoin app being able to authenticate for the current twitter user to use the app to fetch location based tweets as well as post them.

The final integration of the two apps was completed to bring into single application the features of both individual apps. The following steps will explain in detail the app’s functionality:

1. User login: In order to achieve automatic GPS tracking and display current position on google maps the user must be logged into the app. The login window opens up with the launch of the app as shown below:

The Android’s service *ObservationService* class listens to location changes, and creates an observation when the location is changed by an amount specified by user. If an internet connection is available, it is uploaded to the google server immediately and saved locally on the device.

1. Displaying tags and user entered information: Upon successful user authentication, and detecting internet connection the google maps display recorded tags and information as shown below:

The map also listens for location changes and focuses on the current location should it change by a specified amount. To reduce power consumption this is only done when the map *Activity* is shown.

1. Updating tags: An attempt to update a tag is done upon its creation; when no Internet connectivity is available the information is stored locally on the Android device otherwise the public information field is *POST* as public location based tweets onto twitter. When the *app* is closed and re-opened, it searches for the *tags* and the data saved with them locally on the Android device. The screenshot displays this task option:
2. Removing/ Deleting tags: An attempt to remove a tag will delete the tag and the associated data stored with tag on the local Android device. The public tweet data associated with the tag will not be deleted from the Twitter server.
3. Get Tweet: Upon clicking on this option the twitter query will use the REST API 1.1 oauth class for authentication of *app* with twitter user account, and fetch top 10 location based tweets which are displayed besides the tag as shown below:
4. logout: This button will logout the user from the *app,* and close all the open windows of the *app* or browser.

## Local Data Storage

Android’s built-in SQLite engine was used to store user, tag, location, and POI data locally on the Android device, in order for it to be accessed via the *app* on the device and uploaded to the Twitter server. A *Content Provider*, the Android mechanism for data storage and retrieval, was used as an extra abstraction layer to simplify and automate SQL requests. At the start simple text files were used for the same purpose however difficulty in relating and fetching information from them led us to optimize the storage activity with a simpler manipulation strategy. The SQLite database is as shown below:

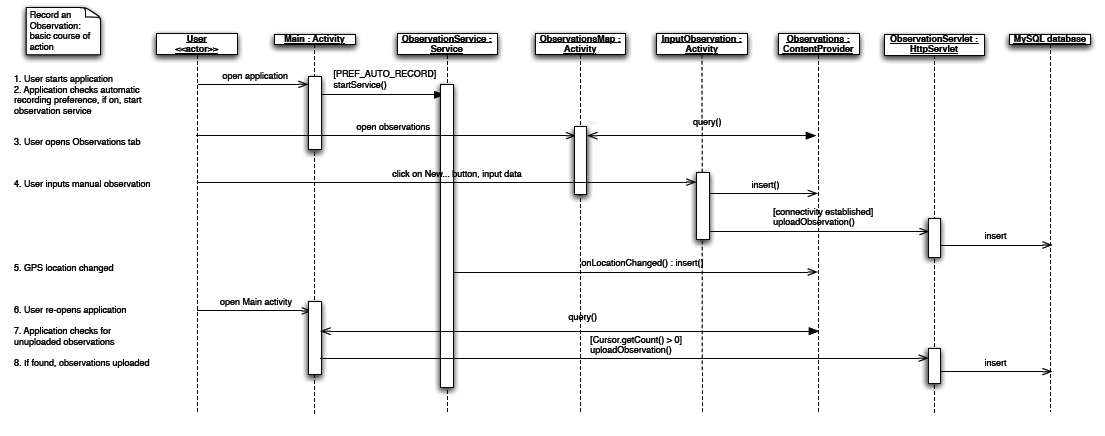
## Class Diagram

Because of Android’s Activity centered design, it’s uninformative to represent the application’s structure with class diagrams. It seems more appropriate to represent the flow of Activities as a sequence. The following diagram represents the \_\_\_\_\_\_\_\_\_\_class and its subclasses:

The parent \_\_\_\_\_class represents an automatic observation created by the r…….. The \_\_\_\_\_\_\_\_subclass adds the descriptive fields that can be added in a manual entry, and a \_\_\_\_\_\_\_\_ additionally includes a byte array image that is used in HTTPS transfer along with local file path from which it is retrieved.

## Sequence Diagram

Actual diagram



# ChECKing the network available and Current location procedure

## Function



## Description:

This procedure describes the checking function in the network whether GPS or the networks like wifi/3G/4G is available for making the connection. After enabled the setMyLocationEnabled in mapFragment from google-play-service.lib, then it will check the system services that whether the current service is GPS or the network for making a connection. Note that requestLocationUpdates consists of two key parameters which are MINIMUM\_TIME\_BETWEEN\_UPDATE and MINIMUM\_DISTANCECHANGE\_FOR\_UPDATE.

MINIMUM\_TIME\_BETWEEN\_UPDATE is the minimum time between two latest update times. It is set at 5 seconds in our project for each update time.

MINIMUM\_DISTANCECHANGE\_FOR\_UPDATE is the minimum distance change in each time the location change, compared to the latest time.

After determining the service, it will request the current location from the getLastKnownLocation. The return value is the Latitute and Longitute of the current location of the user.

# SIGN In Procedure

## Function



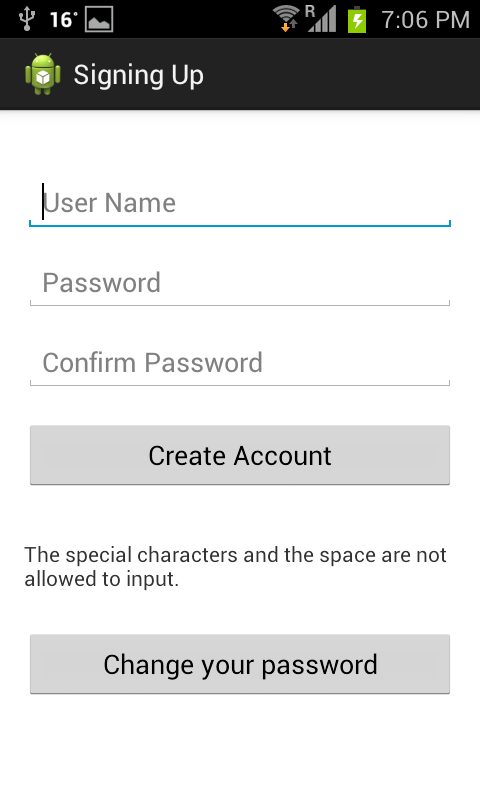
## Description

This activity will support the user an interface for signing into the Pingoin application. After creating an account, the user can be able to login through this activity. After clicking on sign in button, the procedure will check the emptied fields, user matched, password matched, then after passed all checking processes, it goes to the home activity with the current location on the google map.

As you can see in the figure, the login activity is shown with the inputs including username, password, and signin button for logging.

# Sign Up procedure

## Function



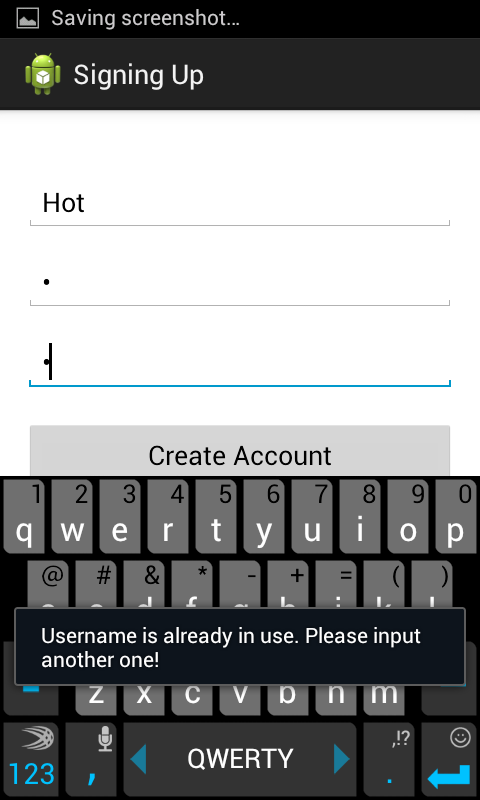
## Description

This activity will support the user who creates an account for using the pingoin application. When the create account button is clicked on, the procedure will check emptied fields, the coincident username, and the passwords unmatched, then if all passed, the password will be processed through the hashing algorithm to encrypt-on-way (SHA1), then this hashing password will be stored into the database on Android app, corresponding to the username. Finally, it will go back to the login activity for signing in.

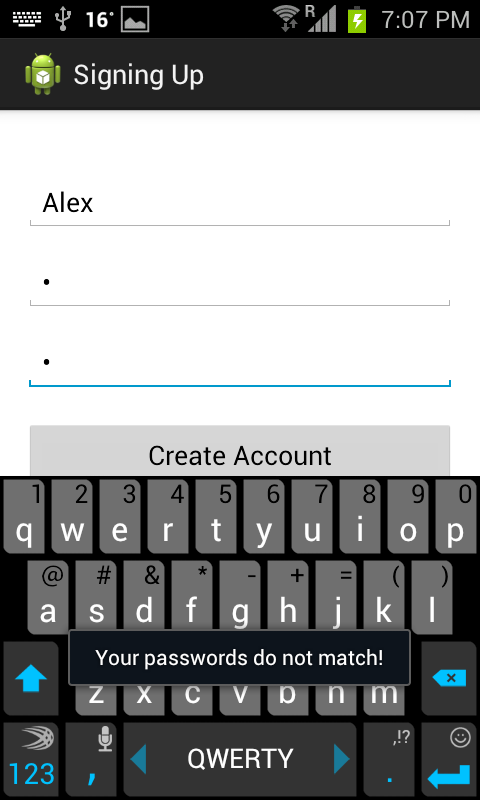
In addition, hashing algorithm is used to hash the data input, in this case, we use for the users’ password since it is needed to be secured. For example, 15 mod4 = 3, where the 15 is the input and mod4 is a hashing function, 3 is the result of hash function. If an eavesdropper can copy the result but he does not know the input, because the input of 7 also has the result of 3. Therefore, it is hard to the eavesdropper for getting the original data. We currently use SHA1 hash function with 160-bit output size.

As can be seen in the figure, signing up activity consists of the inputs such as username, password, confirm password, and buttons including create account and change your password.

If the user is coincident, it will inform a message for input again.



If the password is not matched, it will inform a message involved to notify that the password typed and the confirm password typed are not matched.



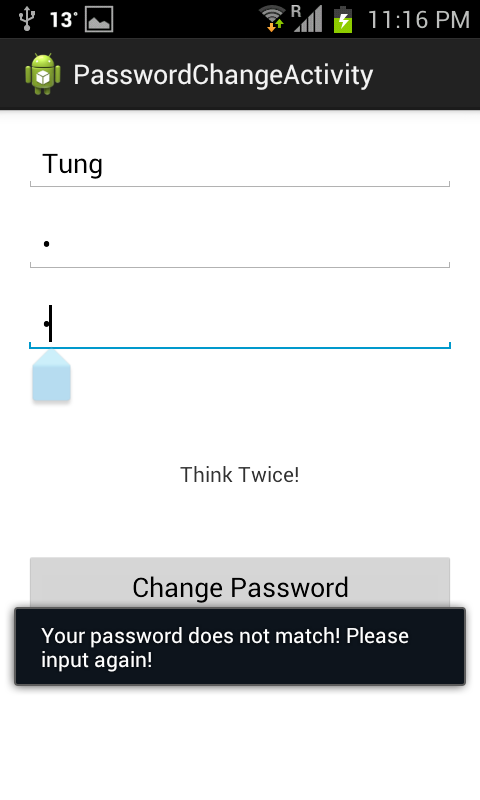
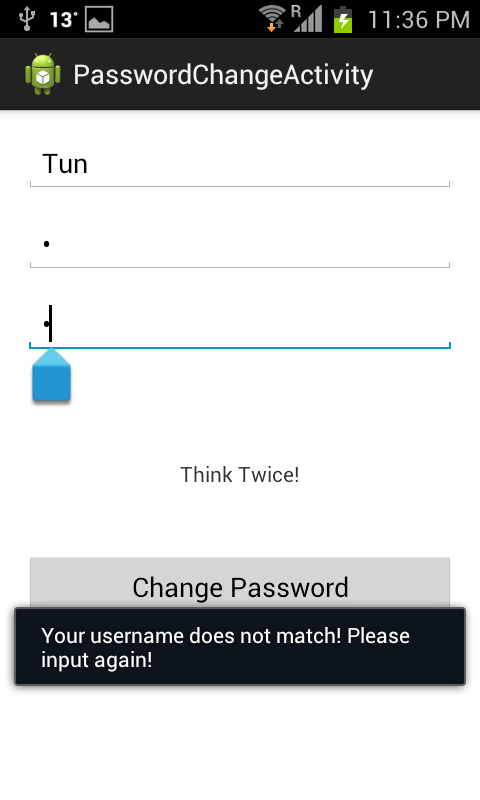
# password change procedure

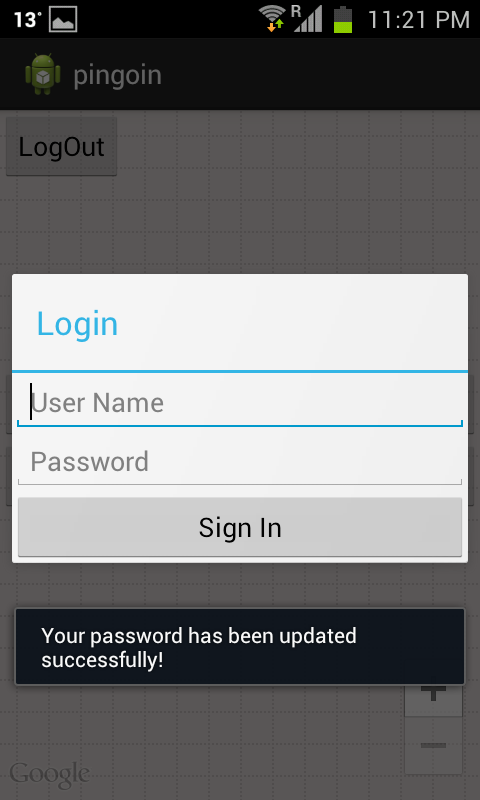
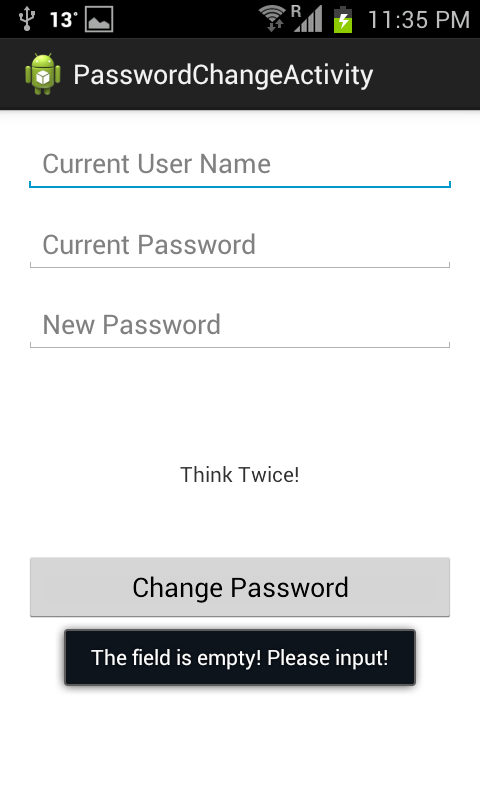
## Function



## Description

This activity will support the user who wants to change the current password to a new one. This procedure includes checking the emptied fields, username matched, password matched, and if all can go through the checking process, the new password will be hashed through the hashing algorithm. Then, the hashing password and the username will be updated in the same entry in the database.

As can be seen in the right two figures that if the user who want to change his current password to the new password, he needs to type the username and password correctly, if not, it will notify that the username or password is not matched to the database.

Moreover, leaving blank fields when clicking on “Change Password” button will cause the notification about the emptied field. After successfully changing the new password, the user is directed to the login activity for signing in as two figures as below.

# Calculation of distance between points of interest

## Description

Assume that we want to get the distance between two points of interest P1(Lat1,Long1) and P2(Lat2,Long2). We have to change the unit from degree to radian since we are calculating the distance in the sphere (great-circle distance).

delta\_Lat = Lat2-Lat1

delta\_Lng = Long2-Long1

Earth radius = 6371 km

A = sin(delta\_Lat/2)\* sin(delta\_Lat/2) + cos(Lat1)\*cos(Lat2)\*sin(delta\_Lng/2)\*sin(deltaLng/2);

// A is the square of half the chord length between the points.

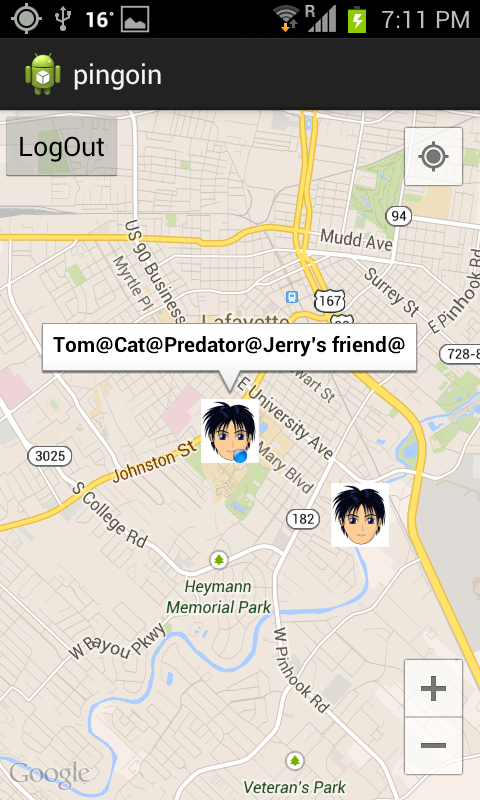
C = 2\*atan2(sqrt(A),sqrt(1-A)) ; C is the angular distance in radians.

Distance = Earth radius \* C ; if you want to get in kilometers;

Distance = Earth radius \* C \* 1000 ; if you want to get in meters;

Note that we use the haversine formula in [1], [2], to calculate the great circle distance between two POIs which is the shortest distance over the earth’s surface.

## Snapshots

As we can see on the screen, the blue dot is the current location, and this user has two tagged friends showed on the map. However, only one of his friend can be allowed to show the information because his friend’s location is within the predefined radius of the current location (we set at 1000 meters). This demo show that our calculate distance function works perfectly.

**References**

[1] <http://www.movable-type.co.uk/scripts/latlong.html>

[2] <http://en.wikipedia.org/wiki/Great-circle_distance>

# Testing

## Tools

Since we were developing for the Android platform, we used the software development kit (SDK), from google for authentication with API key, plug-in for Eclipse IDE, and a number of tools to aid the development. In the light of absence of physical Android device during the initial stages of development, the tools most used in this project was the Android Virtual Device (AVD) which is an emulator as shown below:

Additionally, the Eclipse adt plug-in comes with a Dalvik Debug Monitor Service (DDMS) perspective, which served as the primary testing tool because it includes LogCat, a logging utility which shows information on currently running threads, and emulates certain Android features used extensively in the project, such as pushing geographical locations onto the device, twitter authentication and update of SQLite database entries. For the purpose of the project location *tags* for Girard Park and surrounding areas were created and updated/ deleted successfully. The system handled them effectively without errors or performance lag.

The testing strategy used throughout the course of the project was efficient and informative. The majority of the testing, and data verification was done through LogCat, the logging utility. The utility divides all log messages by tags: *Verbose, Debug, Info, Warning,* and *Error.* There also exists an additional filtering mechanism that can divide messages by class names in which they were generated. In Android, logging is used to print stack trace, notify of system changes, and print user defined information.

LogCat screenshot

In addition to this testing on physical Android devices was also done at later stages to ensure compatibility and functionality requirements were satisfied. However, logging is more versatile since it’s possible to check loop control, Activity and Service status, user preferences etc. Moreover test values can be checked by logging using *systemout*. The Android system prints extensive stack traces with each terminal error along with user logging messages which help to track and eliminate bugs or errors that occur in application.

## Testing the main features

1. Check upon user login for accuracy of user tags and data being displayed: The results were always accurate and as desired.
2. Login into *app* without Internet connectivity: Though the GPS receiver is turned off the geographical location cannot be obtained as it depends on google API to fetch it only via Internet. Thus the *ConnectionDetector* class is used with onProviderDisabled and onStatusChanged methods that account for such behavior.
3. Verify username and password are in the permissible format of Alpha-numeric data only.
4. Using ConnectionDetector class along with function isConnectingToInternet() to detect drop in Internet connectivity while GET / POST tweet. The data is cached locally on the Android device till Internet connectivity is restored and then appropriate queries are executed. Connect/ Disconnect from Internet was tested from settings menu of Android device.
5. Parsing errors were tested by inserting special character in the tag data and username. If the updated data does not fit the parser’s structure nothing is displayed, and the error is logged. Tag data is not updated and the POST method for twitter is not executed.

# Future Development

## Measurement of Success for targeted purpose

The project has met all of its main requirements, and even some of the optional ones; as such, GPS tracking, mapping functions, REST API authentication, and even POST user tweets which was not the original prerequisite. It could be noted, however that the implemented features have not achieved sufficient depth to be used in a real environment. The limitations set upon the software due to lack of expertise in Android programming and time taken to identify skill sets of team members. Since the purpose of the project was to understand the Android programming technology, and development process while the *app’s* goal was to provide POI data with available Internet connectivity; we consider as both goals being achieved as the Android platform was practiced at an advanced level, and the previously unknown API use along with Android SQLite database technology was studied and applied with reasonable efficiency; with few enhancements which could be completed at a later stage.

## Challenges faced

Android programming

Eclipse crashing

## Evaluation

As it stands the small database of user tag data has not caused any performance overheads as also the number of InputStreams to the Google and Twitter servers have been easily handled with high throughput on a wireless infrastructure (wifi) network. The API used share class specification between client and server sides in an efficient way without causing any crash of system. The Android as a client platform has so far provided flexibility, openness, and number of features at programmer’s disposal. The level of support by google and the ease of use of Android SDK enabled us to quickly learn, build and implement the desired *app.*

In terms of adaptability of *Pingoin’s* code, it should be noted that the development followed standard Android practices in design and configuration. Therefore anyone familiar with the platform can easily extend this project. The application used XML based layout, and Values Resources, which are used to store string and array constants in XML files that are separate from the main Java code. An Android application is based on Activities, Services, and other such parameters different from the Object Oriented Programming (OOP) style of coding. However all efforts have been made to keep the program function’s concise, versatile, and re-usable. The code has been properly formatted, documented to improve readability, and commented for limitations/ future improvement possibilities.

The software largely relied on the functions built into the SDK and the Java libraries with the third party code used as libraries from google play services and twitter4j for the server APIs. The few custom made functions are restricted to three Common classes and documented.

A shortcoming of the application is that the minimum distance from the current location on the google map for popping up the POIs does not function as intended due to incomplete distance calculations implemented in the final code. The *app* was built for Android SDK minimum version “9” and maximum “19”. Hence the backward compatibility with targeted Android versions is limited in range. If all the three projects making up this app do not have the same SDK versions then the google play services com.android.gms import throws error and the project crashes. Since Android is still a developing technology the eclipse environment too has limited error correcting and fixing capabilities for the Android projects However, Google has its SDK documentation available for easy access and use thus any new programmer can fix our obsolete code patterns in the future.

## Future Enhancements to the app

This was the first Android project undertaken by Group ‘D’ team members. None of us had attempted an Android programming project before hence only the main targeted features were implemented. However, at the time of development we realized what other features would enhance the features and functionality of the *app.* Some of them could be as follows:

1. Login into *app* using the twitter login (via API) as an option instead of just local login. This would need the device to be connect to the Internet and hence could not be the only login option.
2. Share the POI data as public posts on Facebook for given location.
3. Display user account image on the map besides the user tag to identify his/ her data from the other user’s data being displayed.
4. Expand the location tracking functionality of the *app* by incorporating code to use hardware GPS chips on device. This would enable ability to open maps without Internet connectivity. However google APIs need Internet connectivity for regular operation. Hence this may not be possible to implement.
5. Allow more than one tag data entry for same location co-ordinates per user so as to enable storing multiple event data for a location.
6. Implement GET query with search filter for keywords like “Prices”, “Events”, etc. which could also be user customizable.

# Achievements

## Technological

The following technologies were learned to some depth during the course of development of this project.

1. Java
   1. Data parsing techniques and regular expressions
   2. Threads
   3. Object serialization
   4. Software engineering
2. Server-side and transfer
   1. Google API use
   2. Twitter API (REST) use
   3. HTTPS data transfer
   4. SQLite Android database
3. Android
   1. Basics
   2. XML-based layout
   3. Services
   4. Content providers and SQLite databases
   5. Location based services, geocoding
   6. Mapping and drawing
   7. GET/ POST queries
4. Eclipse IDE
   1. Environment setup
   2. Use of “Android Tools”, and “Project Fix” tools
5. Ant build tool
   1. Operation and configuration

# Project Management

## Roles and Responsibilities’ of the team members

Include Gantt Chart

# Appendices

## Glossary

**Activity** – represents a single, focused window in an Android program, which interacts with the user, much like a Java Frame that takes up entire screen.

**Android** – open-source mobile platform developed by Google. Information, in detail, available at [www.android.com](http://www.android.com)

**Content Provider** – the Android mechanism for storing and retrieving data. It can be used in conjunction with a database, XML file; a plain text file, etc. Android Content Providers include Contacts, Media, Settings, etc.

**DDMS** – Dalvik Debug Monitor Service. It is a debugging tool that ships with the Android Eclipse plug-in.

**GPS** – Global positioning system

**Forward Geocoding** – a technique that searches a specified address and returns its geographical location.

**HTTPS** – hypertext transfer protocol secure

**InputStream/ OutputStream** – Java class representing a stream of bytes, in this context to be transferred over a network

**SQLite** – an open source relational database management system.

**POST** – HTTP request method where the data can be included in the body of the request which produces side effects, such as updating a database.

**REST** – representational state transfer interface that transmits domain-specific data over HTTP/ HTTPS without any additional messaging layer.

**Serialization** – process of converting a Java object into a sequence of bits, to be stored on a storage medium or transferred via a network. Used to transfer objects between client and server side of the system.

**Service** – an Android component that runs in background and without the need for user input. A Service can be either free or bound to an Activity, in which case its lifecycle depends on the lifecycle of the Activity that started it.

**Toast** – Android technique for displaying short-lived messages.

## Procedure for app Installation

1. On Emulator (Android virtual Device (AVD)):
   1. Download and unpack Android SDK, <http://developer.android.com>
   2. Start emulator from <SDK> /tools directory, wait until it loads completely.
   3. Select the project from the Package Explorer, and under menu “Run” select “Run Configurations”. Select the default “Activity” that needs to start at launch of project.
   4. Click on “Run” button to open the AVD window. Select the emulator AVD listed in the window, and click on “OK”.
   5. The application should launch on the emulator. If it does not open automatically then start the *app* on the emulator.
2. On physical Android device:
   1. Connect the physical Android device with a USB cable, and “USB debugging” option enabled; to the computer on which the Eclipse is running.
   2. Select the project from the Package Explorer, and under menu “Run” select “Run Configurations”. Select the default “Activity” that needs to start at launch of project.
   3. Click on “Run” button to open the AVD window. Select the listed physical Android device, and click on “OK”.
   4. The application should launch on the emulator. If it does not open automatically then start the *app* on the Android device.

# Bibliography