Android Tweeter

Class Project

Team SQB

Sridhar

Quynh Quach

Brett Tiller 006638242

Table of Contents

[Overview 3](#_Toc324109977)

[Log into Twitter 3](#_Toc324109978)

[Design 3](#_Toc324109979)

[User Interface 4](#_Toc324109980)

[Development 5](#_Toc324109981)

[Tweets 7](#_Toc324109982)

[Design 7](#_Toc324109983)

[User Interface 7](#_Toc324109984)

[Development 7](#_Toc324109985)

[Timeline 9](#_Toc324109986)

[Design 9](#_Toc324109987)

[User Interface 9](#_Toc324109988)

[Development 9](#_Toc324109989)

[Trends 10](#_Toc324109990)

[Design 10](#_Toc324109991)

[User Interface 10](#_Toc324109992)

[Development 12](#_Toc324109993)

[Local Tweets 14](#_Toc324109994)

[Design 14](#_Toc324109995)

[User Interface 14](#_Toc324109996)

[Development 14](#_Toc324109997)

[Conclusion 15](#_Toc324109998)

# Overview

One of the more popular social network platforms is Twitter. Users of this platform can do things such as send Tweets, follow other tweeters, track trends and those who create them as well as find local tweeters and more. Via the Android platform we developed a mobile phone application that interfaces with the Twitter API.

The features included in this project were the following:

* Log into Twitter
* Send Tweets
* Track trends as well as the tweeters creating them. This feature also allows the user to select and follow a tweeter.
* View the tweets of tweeters being followed by the user.
* Find local tweeters and their tweets, and display via Google maps the location of the tweeter when geo-location or the city is provided.

Each of these features and how it was implemented is discussed in the report below.

There were two techniques utilized to add the features mentioned above. The first technique was to make direct queries to Twitter utilizing the Twitter search API. The received JSON formatted data was parsed, converted into objects and displayed. The second technique utilized the Twitter4J Library which is the unofficial java library for the Twitter API. The provided jar file provided various classes from which the Twitter data could be queried which was then returned in object format and then displayed. All queries utilizing this library were required to be run in a separate thread. Therefore the Android AsyncTask class was utilized to run the query and then to display the data on the main thread as required by Android. The details of each of these techniques are provided in the report.

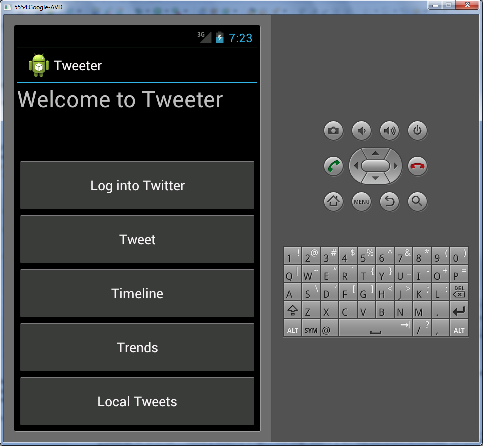
# Log into Twitter

## Design

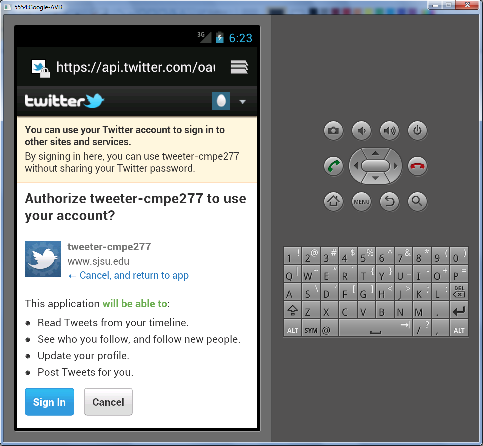
The decision to create a separate login button was relatively simple. There are a few features in Twitter that may not be utilized unless the user is logged into his account. Putting the ‘Log into Twitter’ button on the main screen makes it clear that the user should login. However, not all features require this step, so only in cases where it is needed is the user reminded via a dialog box. On the technology side, this implementation utilizes the Log4J library. This decision was made because this library simplifies the development of the application because the queries are encapsulated in API calls. Twitter4J parses the JSON data which is then returned as objects.

## User Interface

The main screen is shown in Figure 1 below. In the Twitter login page shown in Figure 2, the user logs into Twitter. Once the ‘Sign In’ button is pressed and the authentication validated, the user is then returned via the CALL\_BACK\_URL to the main login page.



**Figure 1: The Main Screen**



**Figure 2: The Twitter Login Web Page**

## Development

In order to run queries via Log4J the application must first be registered with Twitter. This registration provides an authorization consumer key and an authorization consumer secret key which must be included within the twitter object when a query is run. In addition, there are queries that can be run only when the user has logged into Twitter. In order to log into Twitter the Activity that is run must be established as a singleTask or a singleInstance. In addition, the activity must have internet access privileges. These privileges and qualifications are set up in the AndroidManifest.xml file as shown in Figure 3 below.

There is a fair amount of coding required to access the Twitter site and to assure that the data is returned properly as well as that the user interface returns to its original state. We’ll cover the key code here of how to call the twitter site via Log4J. As shown in Figure 4, the authorization request token is requested via the callback URL which is a string data type and specifies what URL should be called when the twitter page is returning back to the originating screen. In our application the CALLBACK\_URL is defined as “login-to-twitter-sqb-01-android:///” . This background step begins the login process; however the user interface portion of the login must occur on the main thread as required by Android which explains why this step occurs in the onPostExecute method call. The basic function of this threading is managed via the AsyncTask class which is provided by the Android API. Once the authorization request key has been received, this key can be used to run queries that require that the user is logged into Twitter.

<uses-permission android:name="android.permission.INTERNET"/>

<activity>

android:name=*".TwitterLogin"*

android:launchMode=*"singleTask"*

android:text=*"@string/activity\_title\_login"*>

<intent-filter>

<action android:name=*"android.intent.action.VIEW"* />

<category android:name=*"android.intent.category.DEFAULT"* />

<category android:name=*"android.intent.category.BROWSABLE"* />

<data android:scheme=*"login-to-twitter-sqb-01-android"* />

</intent-filter>

</activity>

**Figure 3: Snippet of the Android Manifest File**

// Background thread for doing the sign on the twitter

**public** **class** RunTwitterSignOn **extends** AsyncTask<Twitter,String,RequestToken>{

**protected** RequestToken doInBackground(Twitter ... twitter){

RequestToken theReqToken = **null**;

**try**{

theReqToken = twitter[0].getOAuthRequestToken(CALLBACK\_URL);

} **catch** (TwitterException e) {

e.printStackTrace(System.*out*);

}

**return** theReqToken;

}

**protected** **void** onPostExecute(RequestToken token) {

**super**.onPostExecute(token);

**try**{

reqToken=token;

WebView twitterSite = **new** WebView(TwitterLogin.**this**);

twitterSite.loadUrl(token.getAuthenticationURL());

setContentView(twitterSite);

}

**catch**(Exception e){

e.printStackTrace(System.*out*);

}

}

}

**Figure 4: Accessing the Twitter Login Web Page**

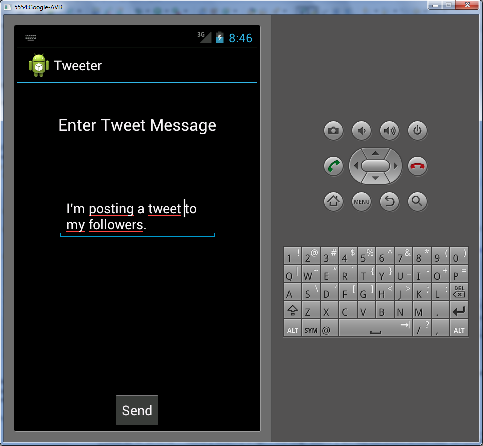
# Tweets

## Design

The sending tweets functionality utilizes the Log4J library and requires that the user has logged into Twitter4J. As a result this login check is done before the Activity is launched. This check is done via the storage class which utilizes the shared preferences functionality to store the AccessToken object after the user has logged into Twitter. If this AccessToken is present, it is assumed that the user has logged into Twitter; otherwise it must be assumed that he has not. If the user has not logged into Twitter, the user is given a message that he must do so. The Log4J library API handles sending the tweet which makes this implementation relatively simple.

## User Interface

The user interface for sending tweets is fairly simple as only a Button, TextView and TextEdit widgets are required as the user interface. As shown in Figure 5, the user merely types in his tweet message and presses the ‘Send’ button.



**Figure 5: Tweet User Interface**

## Development

Once the user log into Twitter has been confirmed, a user interface window is presented in which the user can enter his tweet text, and then press the Send button to post the tweet. This tweet message is posted via the background thread managed in the AsyncTask class and handled in the doInBackground method provided by Android as shown in Figure 6. The method call twitter.updateStatus(<tweet message>) is the call to send the tweet. Once this method has completed, whether the method has succeeded or failed is passed onto the onPostExecute method that then displays a message to the user via Toast.makeText method call of the result.

//Async background/foreground for sending a Tweet

public class RunSendTweet extends AsyncTask<Object,String,Boolean>{

protected Boolean doInBackground(Object ... params){

Boolean bool = new Boolean(true);

try{

AccessToken acToken = (AccessToken)params[0];

String tweetMsg = (String)params[1];

Twitter twitter = new TweeterFactory().getTwitter();

twitter.setOAuthAccessToken(acToken);

twitter.updateStatus(tweetMsg);

} catch (TwitterException e) {

bool = new Boolean(false);

e.printStackTrace(System.out);

}

return bool;

}

protected void onPostExecute(Boolean bool) {

super.onPostExecute(bool);

if(bool.booleanValue()){

Toast.makeText(SendTweet.this,"Your tweet has been sent", Toast.LENGTH\_LONG).show();

}

else{

Toast.makeText(SendTweet.this,"Could not send tweet. Please try again.", Toast.LENGTH\_LONG).show();

}

finish();

}

}

**Figure 6: Code to Send the Tweet**

# Timeline

## Design

Discuss the design of this feature and its function

## User Interface

Discuss how the user interface for this feature works and include a screen shot.

## Development

Discuss the implementation of the feature describing the code functionality and perhaps include a code snippet.

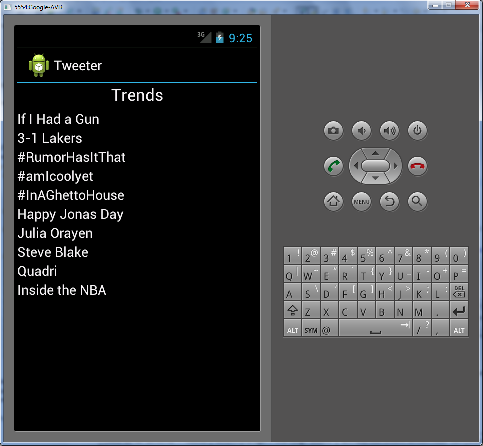
# Trends

## Design

The trends tracking feature utilizes the Log4J library. This feature allows the user to see the latest trending data on a global basis and view the top twenty tweeters who are tweeting on this trend. In addition, following a tweeter is supported here as well. The user can select a tweeter and press the ‘Follow’ button which will then make him a follower of the selected tweeter.

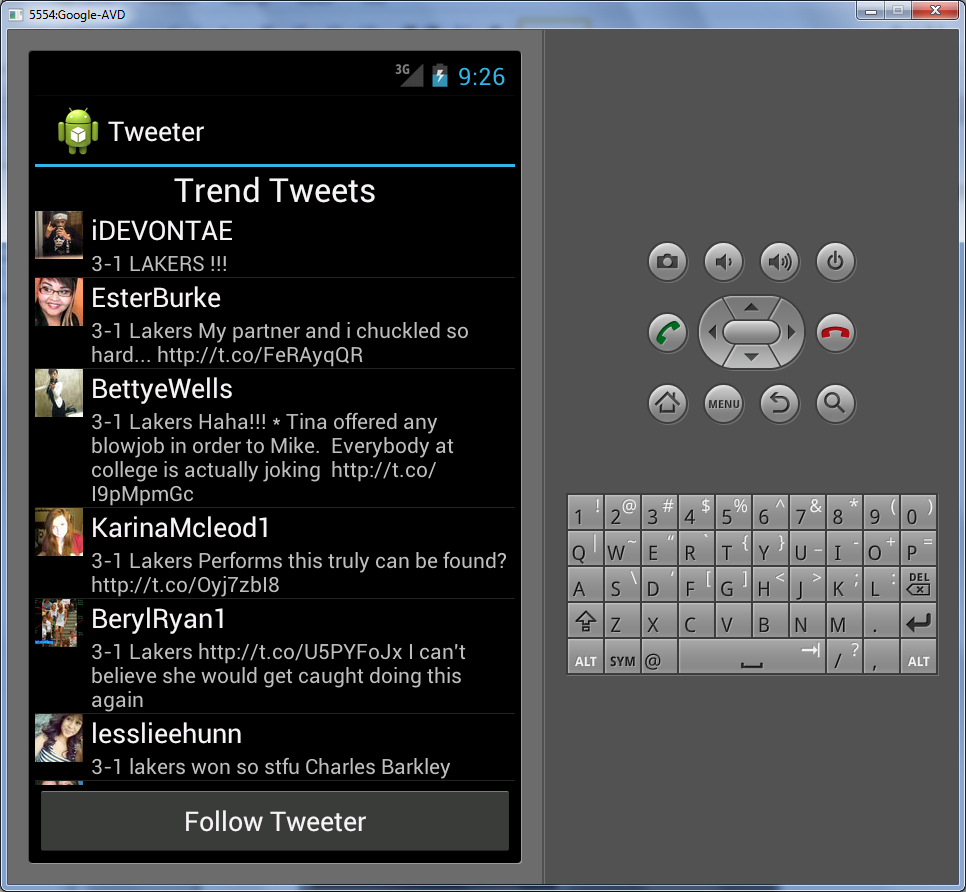
## User Interface

From the main menu, this feature has two levels. In the first level, after the user has clicked on the ‘Trends’ button in the main menu, the trends are provided as shown in Figure 7.



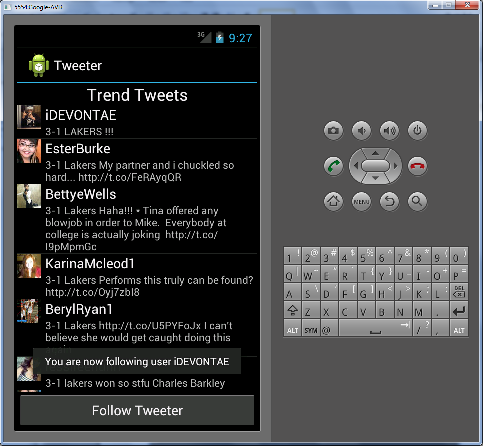
**Figure 7: The Trends**

At this point the user can then click on a trend to view the top twenty tweeters that are tweeting on it. In our user interface we provide an image of the tweeter, the name of the tweeter and the corresponding tweet as shown in Figure 8.



**Figure 8: The Trend Tweeters**

If the user decides the he wants to follow one of the tweeters he simply needs to click on a tweeter and then click on the button ‘Follow Tweeter’. The user is then following that selected tweeter as shown in Figure 9 below, specifically via the message ‘You are now following user iDEVONTAE’.



**Figure 9: Following a Tweeter**

## Development

The implementation of the described functionality required two activities, and therefore class, files TwitterTrends and TrendTweets. The TwitterTrends activity is launched when the ‘Trends’ button is pressed. Using the Log4J library and the AsyncTask class a query is sent requesting the global trends. This functionality is handled in the doBackgroundTask shown in Figure 10. When the trends are returned, they are displayed in a list via the main thread as shown in the onPostExecute method in Figure 10.

//Async background/foreground for querying for trends.

**public** **class** RunQuery **extends** AsyncTask<Twitter,String,Trends>{

**protected** Trends doInBackground(Twitter ... twitter){

Trends resultList=**null**;

**try**{

resultList = twitter[0].getLocationTrends(GLOBAL);

} **catch** (TwitterException e) {

e.printStackTrace(System.*out*);

}

**return** resultList;

}

**protected** **void** onPostExecute(Trends result ) {

**super**.onPostExecute(result);

**if**(result != **null**){

adapter.setTrends(result.getTrends());

}

**else**{

Toast.*makeText*(TwitterTrends.**this**,"No Trends were found!", Toast.*LENGTH\_SHORT*);

}

}

}

**Figure 10: Code Snippet to Get Trends**

When a trend is selected, the selection automatically launches the second activity which is TrendTweets that is passed the selected trend, shown Figure 11 in the setupView() method.

**private** **void** setupView(){

getTrends();

trendsList = (ListView) findViewById(android.R.id.*list*);

trendsList.setAdapter(adapter);

trendsList.setOnItemClickListener(**new** OnItemClickListener() {

**public** **void** onItemClick(AdapterView<?> parent, View view,

**int** position, **long** id) {

Trend trend = (Trend)trendsList.getAdapter().getItem(position);

Intent intent = **new** Intent(TwitterTrends.**this**, TrendTweets.**class**);

intent.putExtra("TREND\_NAME\_FOR\_QUERY", trend.getQuery());

startActivity(intent);

}

});

}

**Figure 11: Launch Activity to Get Trend Tweeters**

After the list of tweeters has been displayed, the user may choose to select and follow a tweeter. This functionality resides within the same activity and is processed with in RunCreateFriendship class called in the followUser method below. Since this functionality requires that the user has logged into Twitter, there is an added check to assure that the AccessToken is present.

**private** **void** followUser(){

**try**{

Storage store = Storage.*getInstance*(getBaseContext());

AccessToken acToken = store.getAccessToken();

**if**(acToken != **null**){

(**new** RunCreateFriendship()).execute(acToken);

}

**else**{

Toast.*makeText*(**this**,"No access token. Attempt to follow tweeter " + selectedTweet.getFromUser() + " failed. Please try again.", Toast.*LENGTH\_LONG*).show();

}

}

**catch**(Exception ex){

Toast.*makeText*(**this**,"Attempt to follow tweeter " + selectedTweet.getFromUser() + " failed. Please try again.", Toast.*LENGTH\_LONG*).show();

}

}

**Figure 12: Code to Follow a Tweeter**

# Local Tweets

## Design

Discuss the design of this feature and its function

## User Interface

Discuss how the user interface for this feature works and include a screen shot.

## Development

Discuss the implementation of the feature describing the code functionality and perhaps include a code snippet.

# Conclusion