tinder for Android

DIY CookBook

(with complete Project & Source code)

Proven Logic

Android apps development - Cookbook

Hands on Android Tutorial to develop a real-world Android app

Proven Logic

This book is for sale at http://leanpub.com/androidtutorial

This version was published on 2014-06-04

Tweet This Book!

Please help Proven Logic by spreading the word about this book on Twitter!

The suggested hashtag for this book is #androidtutorial.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#androidtutorial

Contents

Chat/Messaging

In this section we will learn how to do the chat feature in android. We will learn how to send and receive messages from an already liked profile, how to set up push notification services with chat and how to poll messages from remote database for showing new chat messages. Also here we will learn something new : Its called the **Volley library** for android for fast networking. At the end of this section you will be able to set up your own chat/messaging app for android. Moreover you can combine the concepts like Volley and GCM to make new interesting apps :)

Files to open and refer

Please keep these files open, and follow the code inside to get a better understanding of this section.

- com.appdupe.androidpushnotifications/ChatActivity.java
- com.appdupe.androidpushnotifications/ChatMessageData.java
- com.appdupe.androidpushnotifications/ChatMessageList.java
- com.appdupe.androidpushnotifications/GcmBroadcastReceiver.java
- com.appdupe.androidpushnotifications/GcmIntentService.java
- com.appdupe.androidpushnotifications/MessageObjForDB.java
- com.appdupe.androidpushnotifications/SendMessageResponse.java

If you are directly opening files from the folder, follow these steps:

- Un-zip ***Tinder-for-android.zip***
- Un-zip ***Android-Code.zip***
- Un-zip ***Flamer_Final.zip***
- Open the Flamer folder
- and goto this path ***Flamer/src/com/appdupe/androidpushnotifications/ChatActivity.java***
- and goto this path ***Flamer/src/com/appdupe/androidpushnotifications/ChatMessageData.java***
- and goto this path ***Flamer/src/com/appdupe/androidpushnotifications/ChatMessageList.java***
- and goto this path ***Flamer/src/com/appdupe/androidpushnotifications/GcmBroadcastReceiver.java***
- and goto this path ***Flamer/src/com/appdupe/androidpushnotifications/GcmIntentService.java***
- and goto this path ***Flamer/src/com/appdupe/androidpushnotifications/MessageObjForDB.java***
- and goto this path ***Flamer/src/com/appdupe/androidpushnotifications/SendMessageResponse.java***

Google Cloud Messaging

Important : To get a better understanding of this section, its good to have some knowledge on how to use Google Cloud Messaging services in you app. If you would like to get some knowledge on this, please read through this official documentation : http://developer.android.com/google/gcm/gs.html and try to get hold of GCM.

What is GCM ? In a nutshell, Google Cloud Messaging for Android (GCM) is a free service that helps developers send data from servers to their Android applications on Android devices, and upstream messages from the user's device back to the cloud. The GCM service handles all aspects of queuing of messages and delivery to the target Android application running on the target device.

In our app we are using GCM for push notifications in two cases. One is when the user initiates a chat, we deliver the message using GCM and second is when a match is found between two users, we notify them using push notifications.

Volley project

First lets import the Volley project into the workspace and add it as a library to our project in the same way that you added Facebook SDK and Actionbar Sherlock. **Volley** is a library that makes networking for Android apps easier and most importantly - faster. It manages the processing and caching of network requests and it saves developers valuable time from writing the same network call/cache code again and again.

A few advantages of using Volley

- Volley automatically schedules all network requests ie. Volley will be taking care of all the network requests your app executes for fetching response or image from the web.
- Volley provides transparent disk and memory caching.
- It provides a powerful cancellation request API. It means that you can cancel a single request or you can set blocks or scopes of requests to cancel.
- Volley also provides powerful customization abilities.
- Volley provides useful Debugging and tracing tools.

There are 2 main classes in Volley

1. Request queue
2. Request

Request queue: It is the interest you use for dispatching requests to the network, you can make a request queue on demand if you want, but typically, you'll instead create it early on, at startup time, and keep it around and use it as a Singleton.

Request: It contains all the necessary details for making a web API call. For example: which method to Use (GET or POST), request data to pass, response listener, error listener etc.

We will use Request queue first, as we will be dispatching constant request to the chat server. Lets create a reference,

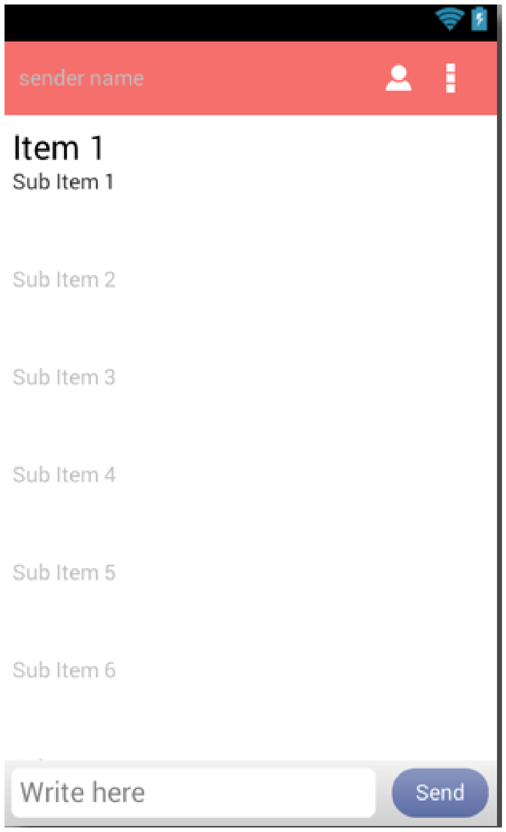
```
RequestQueue mRequestQueue;
```

Also lets get a reference to the Broadcast receiver, as it will be used to broadcast services in this app.

```
private BroadcastReceiver receiver;
```

On the **onCreate()** method, first we will instantiate a new request queue called ***mRequestQueue***

Also we will construct a new **ImageLoader** class. It will take two parameters, one for the request queue, which will handle network requests and other is for the L1 cache. Please note that we are using the android's **LruCache** class for this purpose. We have defined a basic constructor class called **BitmapLruCache.java** inside the **com.android.slidingmenu** package. Now let's set the XML view to our chat class. Create a xml file called **chat_screen.xml** which will hold the chat activity in itself. The chat screen looks like this,



As you see in the above image, we have two main layouts. One is for the title bar and other is for the chat area. The title bar is a relative layout which contains one TextView for name of the friend, an ImageView for the rounded profile image of the friend and two buttons. One button for user info (that looks like a human icon) and other is a pull over menu. The chat area contains one list view, EditText and one submit button.

We will call a method called **initComponent();** which will attach a reference to each of the views inside the xml file and add a click listener where ever required. The code for this is pretty straight forward. It looks like this,

```
private void initComponent() {
                popumenubutton = (Button) findViewById(R.id.popumenubutton);
                popumenubutton.setVisibility(View.GONE);
                popumenubutton.setOnClickListener(new OnClickListener() {

@Override
public void onClick(View v) {
// Creating the instance of PopupMenu
PopupMenu popup = new PopupMenu(ChatActivity.this,popumenubutton);
// Inflating the Popup using xml file
popup.getMenuInflater().inflate(R.menu.popup_menu,popup.getMenu());

// registering popup with OnMenuItemClickListener
popup.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener() {
        public boolean onMenuItemClick(MenuItem item) {
Toast.makeText(ChatActivity.this,"You Clicked : " + item.getTitle(),
                                                                Toast.LENGTH_SHORT).show();
                                                return true;
                                        }
                                });

                                popup.show();// showing popup menu
                        }
                });// closing the setOnClickListener method

blockUserLayout = (RelativeLayout) findViewById(R.id.blockUserLayout);
headerView = ((LayoutInflater) getSystemService(Context.LAYOUT_INFLATER_SERVICE))
                                .inflate(R.layout.header, null, false);
chatEditText = (EditText) findViewById(R.id.chat_editText);
sendChatMessageButton = (Button) findViewById(R.id.send_chat_message_button);

loadmoreButton = (RelativeLayout) headerView
                                .findViewById(R.id.loadmore_button);
loadmoreButton.setVisibility(View.GONE);
senderimage = (ImageView) findViewById(R.id.senderimage);
senderName = (TextView) findViewById(R.id.senderName);
MoreOptionLayout = (TextView) findViewById(R.id.MoreOptionLayout);

userProfilelayout = (RelativeLayout) findViewById(R.id.userProfilelayout);
flagreportlayout = (RelativeLayout) findViewById(R.id.flagreportlayout);
blockuserlayout = (RelativeLayout) findViewById(R.id.blockuserlayout);
userinfoimageview = (Button) findViewById(R.id.userinfoimageview);

Typeface topbartextviewFont = Typeface.createFromAsset(getAssets(),
                              "fonts/HelveticaLTStd-Light.otf");
                senderName.setTypeface(topbartextviewFont);
                senderName.setTextColor(Color.rgb(255, 255, 255));
                senderName.setTextSize(20);

                MoreOptionLayout.setTypeface(topbartextviewFont);
                MoreOptionLayout.setTextColor(Color.rgb(255, 255, 255));
                MoreOptionLayout.setTextSize(20);

        }
```

Now let's get a reference to the local app directory and image directory which we will use after downloading the image.

```
Bundle bucket = getIntent().getExtras();
                Ultilities mUltilities = new Ultilities();
                File appDirectory;
                File _picDir;
                File myimgFile;
```

Next let's set the profile picture by calling the **setProfilePick();** method. This method will look in the database for the user's profile image URL in the local storage folder. Also let's get the intent extras that were sent by the **MainActivity** when the user clicked on a liked profile to chat. We have the facebookId of the selected profile for chat.

```
if (bucket != null) {
String checkForPush = bucket.getString(Constant.CHECK_FOR_PUSH_OR_NOT);
        if (checkForPush != null && checkForPush.equals("1")) {
                strFriendFbId = bucket.getString(Constant.FRIENDFACEBOOKID);
                        } else {
        Bundle bucket1 = getIntent().getBundleExtra("PUSH_MESSAGE_BUNDLE");
        strFriendFbId = bucket1.getString("sfid");
                        }

                } else { }
```

As you see from the above code, we will get the Friend's FB id from the extras of the intent. Now using this facebook id we can get the sender's name and profile picture URL for local folder from the database. Remember when we created the right menu, we fetched all the liked profiles from the remote server and saved it to the SQLite database before adding it to the listview adapter.

Note : SQLite is a lightweight database used mainly with mobile operating systems for fast and light data transfer. You can learn more about SQLite here : http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html

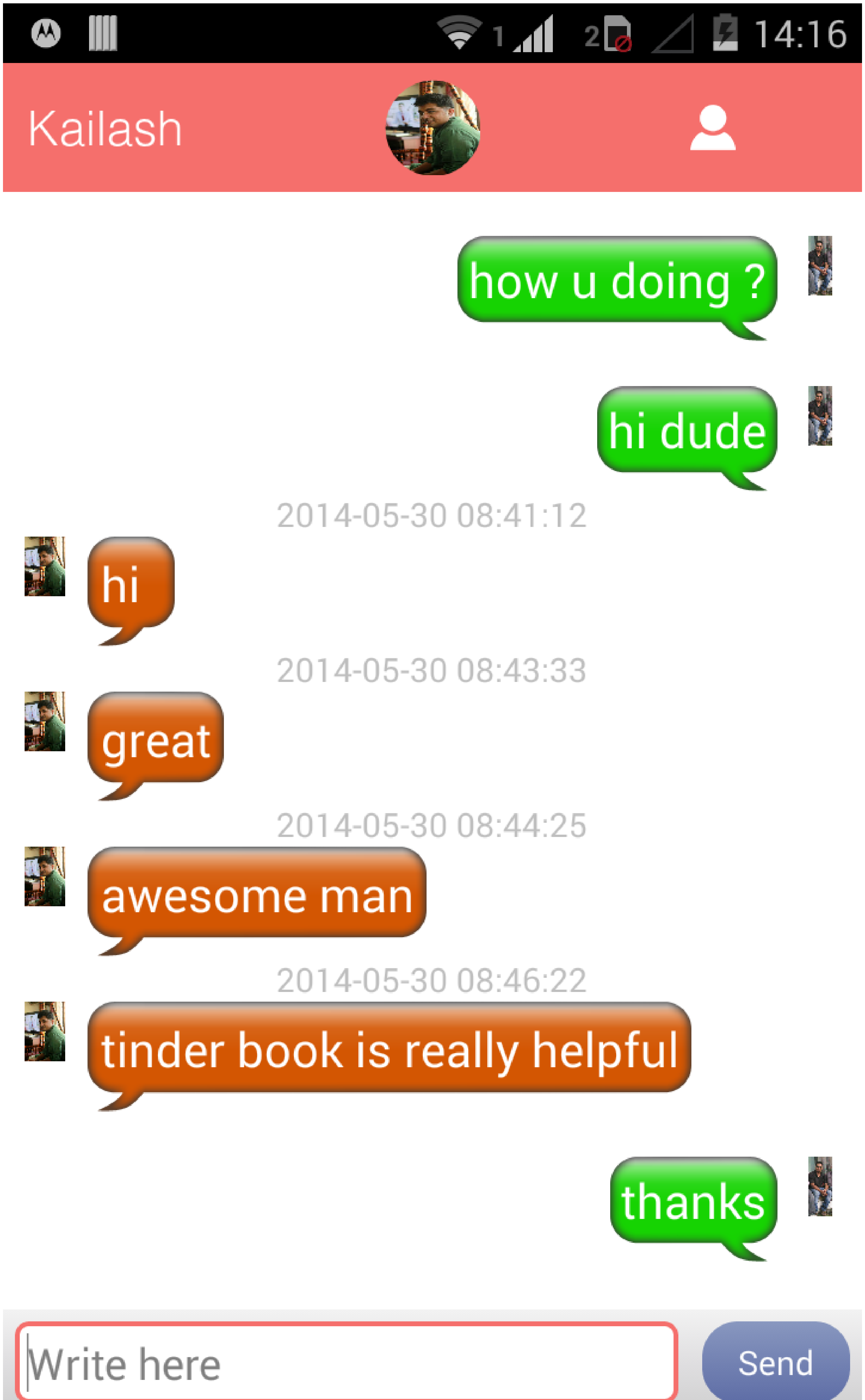
We will use aQuery to load the sender image to the circular **imageview** on the top and then just set the sender name into the **senderName** TextView. The code for this is shown below,

```
LikeMatcheddataForListview matcheddataForListview = mDatabaseHandler
                                .getSenderDetail(strFriendFbId);
                Log.i(TAG, "oncreate imagePath  matcheddataForListview"
                                + matcheddataForListview);

                String imagePath = matcheddataForListview.getFilePath();
                senderimageUrl = matcheddataForListview.getImageUrl();
                String senderNamevalues = matcheddataForListview.getUserName();

                ScalingUtilities mScalingUtilities = new ScalingUtilities();

                aQuery.id(senderimage).image(senderimageUrl);
                senderName.setText(senderNamevalues);
```

Also we will set up the chat area. The chat area will hold messages from each user with timestamp and their profile image. Lets design each chat message to look like a SMS balloon.

Chat

Below is the sample image of the chat area,



how u doing ?

hi dude

2014-05-30 08:41:12

hi

2014-05-30 08:43:33

great

2014-05-30 08:44:25

awesome man

2014-05-30 08:46:22

tinder book is really helpful

thanks

To achieve this we will add an adapter to the **listview**. We name this adapter as **Awesome adapter** and it will take an array list of chat data as its elements. This array list will be filled with chat messages either on the Broadcast receiver's, on receive method when a new chat message comes or from the local SQLite database, in case we want to pre-populate the chat messages. Let's discuss this section in detail now,

```
public class AwesomeAdapter extends BaseAdapter {
                private Context mContext;
                private ArrayList<ChatMessageList> mMessages;

                public AwesomeAdapter(Context context,
                ArrayList<ChatMessageList> messages) {
                super();
                this.mContext = context;
                this.mMessages = messages;
                }

                @Override
                public int getCount() {
                return mMessages.size();
                }

                @Override
                public Object getItem(int position) {
                return mMessages.get(position);
                }

                @Override
                public View getView(int position, View convertView, ViewGroup parent) {
                ChatMessageList message = (ChatMessageList) this.getItem(position);

                ViewHolder holder;
                if (convertView == null) {
                holder = new ViewHolder();
                convertView = LayoutInflater.from(mContext).inflate(
                              R.layout.sms_row, parent, false);
                              holder.message = (TextView) convertView
                                .findViewById(R.id.message_text);
                                holder.userImageview = (NetworkImageView) convertView
                                .findViewById(R.id.userImageview);
                                holder.chatedate = (TextView) convertView
                                .findViewById(R.id.chatedate);
                                holder.chateboxlayout = (RelativeLayout) convertView
                                .findViewById(R.id.chateboxlayout);
                                convertView.setTag(holder);
                        } else
                                holder = (ViewHolder) convertView.getTag();

                        holder.message.setText(message.getStrMessage());
                        holder.chatedate.setText(message.getStrDateTime());

                        String userId = "";
Log.i(TAG,"getView userFacebookid...."
+ userFacebookid+ "..........getStrSenderFacebookId..."
+ message.getStrSenderFacebookId());
if (userFacebookid.equals("" + message.getStrSenderFacebookId())) {

RelativeLayout.LayoutParams paramsImage =
new RelativeLayout.LayoutParams(50, 50);
paramsImage.addRule(RelativeLayout.ALIGN_PARENT_RIGHT);
paramsImage.setMargins(0, 0, 10, 0);
holder.userImageview.setLayoutParams(paramsImage);

RelativeLayout.LayoutParams paramsTextview =
new RelativeLayout.LayoutParams(
RelativeLayout.LayoutParams.WRAP_CONTENT,
RelativeLayout.LayoutParams.WRAP_CONTENT);
paramsTextview.addRule(RelativeLayout.LEFT_OF,
holder.userImageview.getId());
paramsTextview.setMargins(0, 0, 10, 0);
holder.message.setLayoutParams(paramsTextview);

holder.message
.setBackgroundResource(R.drawable.speech_bubble_green);
holder.userImageview.setImageUrl(cureentImageurl, imageLoader);
else {
RelativeLayout.LayoutParams paramsImage =
new RelativeLayout.LayoutParams(50, 50);
paramsImage.addRule(RelativeLayout.ALIGN_PARENT_LEFT);
paramsImage.setMargins(10, 0, 0, 0);
holder.userImageview.setLayoutParams(paramsImage);

RelativeLayout.LayoutParams paramsTextview =
new RelativeLayout.LayoutParams(RelativeLayout.LayoutParams.WRAP_CONTENT,
                RelativeLayout.LayoutParams.WRAP_CONTENT);
                paramsTextview.addRule(RelativeLayout.RIGHT_OF,
                holder.userImageview.getId());
                paramsTextview.setMargins(10, 0, 0, 0);
                holder.message.setLayoutParams(paramsTextview);

        holder.message.setBackgroundResource(R.drawable.speech_bubble_orange);

holder.userImageview.setImageUrl(senderimageUrl, imageLoader);
               }

                        return convertView;
                }

                private class ViewHolder {
                        TextView message;
                        NetworkImageView userImageview;
                        TextView chatedate;
                        RelativeLayout chateboxlayout;
                }

                @Override
                public long getItemId(int position) {
                        // Unimplemented, because we aren't using Sqlite.
                        return 0;
                }
        }
```

In the above code, we are inflating a new layout called **sms_row.xml** which will have a **TextView** for chat message date, an **ImageView** for both the chat sender's profile pictures and a **TextView** for the actual chat message. We will define a View holder, which will hold the view for a single chat message and inflate it to the adapter once a new message is available to show. We will add reference of the views from the **sms_layout.xml** to the **ViewHolder** element and then assign the text from message data to the message and **chatedate TextViews**. Also we will check to see if the message in the chat area is from the current user or from their friend. Based on this we will put a green chat bubble as user's chat message background and an orange chat bubble for sender's chat message. To do this we will define a new relative layout and add it to the parent. If the facebook id of the user is equal to sender id of the message then we will set the background green

```
holder.message.setBackgroundResource(R.drawable.speech_bubble_green);
```

or we will set it to orange

```
holder.message.setBackgroundResource(R.drawable.speech_bubble_orange);
```

The next step is to get the history of messages between you and the sender and arrange them in the ListView. We can do this either by pulling all messages from the remote database in real time or by quering the local SQLite database. To query the remote server in the background, we need to pass the session token, device id and facebook id of the sender as parameters to the server. Following is the code,

```
userFacebookid = mSessionManager.getFacebookId();
                sessiosntoken = mSessionManager.getUserToken();

                deviceId = Ultilities.getDeviceId(ChatActivity.this);
                getListView().addHeaderView(headerView);
                messageAdapter = new AwesomeAdapter(ChatActivity.this, listChatData);
                setListAdapter(messageAdapter);
                if (bFlagForHistoryMessage) {
                        String[] params = { sessiosntoken, deviceId, strFriendFbId, "300" };
                        new BackgroundforMessagehistory().execute(params);
                } else {
                        new BackgroundForGetDataFromDB().execute();
                }
```

As you see from the above code, we have two methods for getting chat messages. **BackgroundforMessagehistory()** will get the data from the remote database whereas **BackgroundForGetDataFromDB()** will get it from the local SQLite database. We will explain these two methods in detail. Lets look at the **BackgroundForGetDataFromDB()** first,

```
private class BackgroundForGetDataFromDB extends
                        AsyncTask<String, Void, ArrayList<ChatMessageList>> {

                @Override
                protected ArrayList<ChatMessageList> doInBackground(String... params) {
                        DatabaseHandler objDBHandler = new DatabaseHandler(
                                        ChatActivity.this);

                        if (pageNum == 0) {
                                listChatData.clear();
                        } else {

                        }

ArrayList<ChatMessageList> listChatDataDb = objDBHandler
```

```
                                        .getChatMessages(strFriendFbId, startLimit, limit);
Log.i(TAG, "BackgroundForGetDataFromDB listChatDataDb size....."
                                        + listChatDataDb.size());

                        return listChatDataDb;
                }

                @Override
                protected void onPreExecute() {
                        super.onPreExecute();
                }

                @Override
                protected void onPostExecute(ArrayList<ChatMessageList> result) {
                        super.onPostExecute(result);
                        Log.i(TAG,"BackgroundForGetDataFromDB result size....."
                                                + result.size());
                        Collections.reverse(result);
                        listChatData.addAll(0, result);
                        messageAdapter.notifyDataSetChanged();
                        if (listChatData != null && listChatData.size() > 0) {
                                getListView().setSelection(listChatData.size());
                        }

                }
        }
```

In the above code, first we are clearing any data present inside the list view. Then we are calling the **getChatMessages()** method of the **DatabaseHandler** class with friend’s facebook id and limits as parameter. This will query the SQlite database and get all the list of chat messages that were saved during chat. Then we will just reverse the order of the messages to show the recent messages.

The second method is to fetch chat history data from the remote database. For this we will call the **yourHostName.com/getChatHistory_url** and pass the parameters that we discussed above. Once we receive the JSON data from the server, we can parse it and add it to our **objChatMessageData**. Finally on the **onPostExecute()** we will add this data to our listview.

```
private class BackgroundforMessagehistory extends
        AsyncTask<String, Void, ChatMessageData> {

        @Override
protected ChatMessageData doInBackground(String... params) {
Utility myUtility = new Utility();
List<NameValuePair> sendMessageReqList = myUtility
.getPullMessageReq(params);
String messageResponse = myUtility.makeHttpRequest(
Constant.getChatHistory_url, Constant.methodeName,
sendMessageReqList);
ChatMessageData objChatMessage = null;
if (messageResponse != null) {
Gson gson = new Gson();
objChatMessage = gson.fromJson(messageResponse,
ChatMessageData.class);
if (objChatMessage != null) {
if (objChatMessage.getErrFlag() == 0) {
List<ChatMessageList> listChat = objChatMessage
                .getListChat();
ChatMessageList objChatMessageData = listChat.get(0);
objChatMessageData.setStrSenderId(objChatMessageData
.getStrSenderFacebookId());
objChatMessageData.setStrReceiverId(userFacebookid);
objChatMessageData.setStrFlagForMessageSuccess("1");
DatabaseHandler objDBHandler = new DatabaseHandler(
ChatActivity.this);
listChatData.add(objChatMessageData);
objDBHandler.insertMessageData(objChatMessageData);

                        } else {
                }
                        } else {

                                }
                        } else {
runOnUiThread(new Runnable() {
        @Override
        public void run() {
        Toast.makeText(ChatActivity.this,
        R.string.request_timeout, Toast.LENGTH_SHORT)
        .show();
```

```
                        }
                                });
                        }

                        return objChatMessage;

                }

                @Override
                protected void onPreExecute() {
                        super.onPreExecute();
                }

                @Override
protected void onPostExecute(ChatMessageData result) {
        super.onPostExecute(result);
        if (result != null) {
                if (result.getErrFlag() == 0) {
        List<ChatMessageList> listChat = result.getListChat();
        listChatData.clear();
        listChat.addAll(listChat);
                } else {

                }
                        } else {

                        }
                }
        }
```

Now that we are getting the history of chat messages from the database, let's see how we can receive and send messages. To receive the message we are using android's **BroadcastReceiver** class. First let's define a broadcast receiver and how it works in our project.

We will integrate the GCM services into our app, which will receive the messages sent by friends as push notifications. We have to add GCM jar files to our app dependency which will give us the required methods and classes to access GCM. To work with GCM we will create two Class files, **GcmBroadcastReceiver.java** and **GcmIntentService.java**. Then we have to define this broadcast receiver in our app's manifest file. Once any message comes through GCM, it will be received by the GcmBroadcastReceiver.java class and then it will pass it to the **GcmIntentService.java** class to process the message. But if the app is not in the background or in closed state, then it will receive it while it's been used by the user. So we will define a receiver in our **ChatActivity.java** which will receive this message when the app is in foreground and will process the message. Let see how this

onReceive function works first, then we will look into the GCM classes.

```
receiver = new BroadcastReceiver() {

@Override
public void onReceive(Context context, Intent intent) {
Bundle bucket = intent.getExtras();

String strMessage = bucket.getString("MESSAGE_FOR_PUSH");
String strMessageType = bucket
.getString("MESSAGE_FOR_PUSH_MESSAGETYPE");
String strMessageId = bucket
                .getString("MESSAGE_FOR_PUSH_MESSAGEID");
                String strSenderName = bucket
                .getString("MESSAGE_FOR_PUSH_SENDERNAME");
                String strDateTime = bucket
                .getString("MESSAGE_FOR_PUSH_DATETIME");
                String strFacebookId = bucket
                .getString("MESSAGE_FOR_PUSH_FACEBOOKID");

                if (strMessageType.equals("2")) {
                String[] params = { sessiosntoken, deviceId, strMessageId };
                new BackgroundForPullMessage().execute(params);
                } else {
                ChatMessageList objMessageData = new ChatMessageList();
                objMessageData.setStrMessage(strMessage);
                objMessageData.setStrDateTime(strDateTime);
                objMessageData.setStrSenderFacebookId(strFacebookId);
                objMessageData.setStrSendername(strSenderName);
                objMessageData.setStrFlagForMessageSuccess("1");
                objMessageData.setStrReceiverId(userFacebookid);
                Log.i(TAG, "onReceive setStrSenderId...." + strFriendFbId);
                objMessageData.setStrSenderId(strFriendFbId);
                if (strFriendFbId.equals(strFacebookId)) {
                listChatData.add(objMessageData);
                } else {
                Toast.makeText(ChatActivity.this,
                "You have message from :" + strSenderName,
                Toast.LENGTH_SHORT).show();
                                        }

                new BackgroundForInsertMessageDB().execute(objMessageData);
                                }
```

```java

                messageAdapter.notifyDataSetChanged();
                getListView().setSelection(listChatData.size());
                        }
                };
```

Do remember to register this receiver in the **onResume()** method and unregister the receiver on the app's onDestroy() or onStop() method. Also define this receiver in the App's manifesto file.

We can receive messages of two types. One is with a message id only –or- with full message text. If it is with only the message id then we will perform a background task named **BackgroundForPullMessage()** to pull the message from the remote database with this message id. But if we have the text inside the message, then we will just extract it and add it to the listview. In this case, as we are getting the text directly and not the message id, it means the message is not present in our remote database. So lets insert this message to our remote database by calling the **BackgroundForInsertMessageDB()** method.

Incase with the **BackgroundForPullMessage()** method we will call the **yourdomain.com/ getChatMessage_url**, which will pull the message from the remote database with the message Id that we have passed to the class. This is a simple HTTP request as we have done before. Have a look at the following code for better understanding,

```java
private class BackgroundForPullMessage extends
        AsyncTask<String, Void, ChatMessageData> {

        @Override
        protected ChatMessageData doInBackground(String... arg0) {

        Utility myUtility = new Utility();
                        // String params[]={};
        List<NameValuePair> sendMessageReqList = myUtility
        .getPullMessageReq(arg0);
        String messageResponse = myUtility.makeHttpRequest(
        Constant.getChatMessage_url, Constant.methodeName,
        sendMessageReqList);
        ChatMessageData objChatMessage = null;
        if (messageResponse != null) {
        Gson gson = new Gson();
        objChatMessage = gson.fromJson(messageResponse,
        ChatMessageData.class);
        if (objChatMessage != null) {
        if (objChatMessage.getErrFlag() == 0) {
        List<ChatMessageList> listChat = objChatMessage
```

```
                                .getListChat();
        ChatMessageList objChatMessageData = listChat.get(0);
        objChatMessageData.setStrFlagForMessageSuccess("1");
        objChatMessageData.setStrSenderId(objChatMessageData
        .getStrSenderFacebookId());
        objChatMessageData.setStrReceiverId(userFacebookid);
        DatabaseHandler objDBHandler = new DatabaseHandler(
        ChatActivity.this);
        listChatData.add(objChatMessageData);
        objDBHandler.insertMessageData(objChatMessageData);

        } else {

                        }
        } else {

        }
        } else {
        runOnUiThread(new Runnable() {

        @Override
        public void run() {
        Toast.makeText(ChatActivity.this,
        R.string.request_timeout, Toast.LENGTH_SHORT)
        .show();
                                        }
                                });
                        }

                return objChatMessage;
        }

        @Override
        protected void onPreExecute() {
        super.onPreExecute();
                }

        @Override
        protected void onPostExecute(ChatMessageData result) {
        super.onPostExecute(result);

        messageAdapter.notifyDataSetChanged();
        getListView().setSelection(listChatData.size());
                }
        }
```

The **BackgroundForInsertMessageDB()** method is similar to the **BackgroundForPullMessage()** ,except that it will do a HTTP Put. This will insert the message data into the database.

Message sending

Now as we have finished the receiving part, let's start the message sending part. For this we will attach a click listener to the send button. **onClick** we will get the text message, friend's device id, session token and device id and pass as parameters to the **BackGroundForSendMessage()** class for send the message to the friend of user. Also we will set the text message into the **listview** adapter to show it to the user in the chat area.

Let's see how this method **BackGroundForSendMessage()** works. Here we will get the text message, sender facebook id, session token and device id, post which we will make a HTTP request to the **yourdomain.com/sendMessage_url** to process the chat message and do the delivery. Once we get a success response from the server, we can insert this to our local SQLite database which we need in case of offline processing. Have a look at the code,

```
private class BackGroundForSendMessage extends
AsyncTask<String, Void, SendMessageResponse> {
@Override
protected SendMessageResponse doInBackground(String... arg0) {

Utility myUtility = new Utility();
List<NameValuePair> sendMessageReqList =
myUtility.getSendMessageReq(arg0);
String messageResponse =
myUtility.makeHttpRequest(Constant.sendMessage_url,
 Constant.methodeName,sendMessageReqList);
SendMessageResponse sendMessagerespObj = null;
if (messageResponse != null) {
 Gson gson = new Gson();
sendMessagerespObj =gson.fromJson
 (messageResponse,SendMessageResponse.class);
objMessageData = new ChatMessageList();
  objMessageData.setStrSenderFacebookId(userFacebookid);
  objMessageData.setStrMessage(arg0[3]);
  objMessageData.setStrDateTime("");
  objMessageData.setStrSendername("");
  objMessageData.setStrSenderId(userFacebookid);
  objMessageData.setStrReceiverId(strFriendFbId);
  DatabaseHandler objDBHandler = new DatabaseHandler(
  ChatActivity.this);
  if (sendMessagerespObj != null) {
  if (sendMessagerespObj.getStatusNumber() == 0) {
ForMessageSuccess("1");
        objDBHandler.insertMessageData(objMessageData);
                                } else {
        objMessageData.setStrFlagForMessageSuccess("0");
        objDBHandler.insertMessageData(objMessageData);
                                }
                        } else {                                objMessageData.set
        objDBHandler.insertMessageData(objMessageData);
                        }
                } else {
        runOnUiThread(new Runnable() {

@Override
public void run() {
Toast.makeText(ChatActivity.this,
  R.string.request_timeout,Toast.LENGTH_SHORT)
                        .show();
                                }
                        });
                }
                return sendMessagerespObj;
        }

        @Override
        protected void onPreExecute() {
                super.onPreExecute();
        }

        @Override
protected void onPostExecute(SendMessageResponse result) {
        super.onPostExecute(result);
        if (result != null) {
        if (result.getStatusNumber() == 0) {

        } else {
Toast.makeText(ChatActivity.this,
 result.getStatusMessage(), Toast.LENGTH_SHORT).show();
                }
                        } else {

                        }
                }
        }
```

In the above code we are making a HTTP request to **sendMessage_url** which will do the processing of sending message to the friend. The php code for this in server side looks like this,

```
protected function sendMessage($args) {

        if ($args['ent_message'] ==
        '' || $args['ent_user_fbid'] == '')
        return $this->_getStatusMessage(1, 49);
        $returned = $this->_validate_token($args['ent_sess_token'],
                      $args['ent_dev_id']);

        if (is_array($returned))
            return $returned;
        $recieverId = $this->_getEntityId($args['ent_user_fbid']);

        if (is_array($recieverId))
            return $recieverId;
        $recEntityArr[] = $recieverId;

        $msg_type = 1;

        if (strlen($args['ent_message']) > 165) {
            $msg_type = 2;
            $args['ent_message'] =
            substr($args['ent_message'], 160) . '..';
        }

        $curr_date = time();
        $curr_gmt_date = gmdate('Y-m-d H:i:s', $curr_date);

        $storeMsgQry = "insert into t_chatmessages
                         (sender,receiver,message,msg_dt)
            values('" . $this->User['entityId'] .
"','" . $recieverId . "','" . $args['ent_message'] .
"','" . $curr_gmt_date . "')";
        mysql_query($storeMsgQry, $this->db->conn);

        $insertedId = mysql_insert_id();

        if ($insertedId > 0) {
$sendPush = $this->_sendPush($this->User['entityId'],
$recEntityArr, $args['ent_message'], '2',
$this->User['firstName'], $this->User['fbId'],
$curr_gmt_date, $insertedId, $msg_type);

            return $this->_getStatusMessage($sendPush['errNum'], 50);
        } else {

            return $this->_getStatusMessage(37, 51);
        }
    }
```

The above code receives the request from our android app and gets the parameters that we sent ie. sender id, receiver id, message and session token etc. It checks if your message is more than 165 characters and just limit it to 160 characters before sending the push notification to the actual device. Also it stores the message in our remote MySQL database. Once this message is received by the device, the **GcmBroadcastReceiver.class** will take care of showing it to the user and saving this message in the local SQLite database. We will discuss in detail about this class and services at the end of this section.

Other than the send button, we will add some click listener's to load more buttons, friend's profile images in the chat area. The load more button will load history messages between two users while the click on friend's image will show the detail profile of the friend. Have a look at the code,

```
if (arg0.getId() == R.id.loadmore_button) {
                        pageNum = 1;
                        startLimit = startLimit + limit;
                        String params[] = { "1" };
                        new BackgroundForGetDataFromDB().execute(params);
                }

                if (arg0.getId() == R.id.senderimage) {
                        SessionManager mSessionManager =
                new SessionManager(
                                        ChatActivity.this);
                mSessionManager.setMatchedUserFacebookId(strFriendFbId);
                mSessionManager.setImageIndexForLikeDislike(0);
```

```
                    Intent mIntent = new Intent(ChatActivity.this,
                            MatChedUserProfile.class);
                            Bundle bundle = new Bundle();
                            bundle.putBoolean(Constant.isFromChatScreen, true);
                            mIntent.putExtras(bundle);
                            startActivity(mIntent);
                    }
```

In the above code, we get to see how to design the chat area and how to show the chat messages in the chat area. Also we see how to send the chat through HTTP request to **sendmessage_url** which ultimately delivers the push messages. Now let's see how to receiving messages is done through GCM. As we discussed before the message that comes through GCM is saved into the local SQLite database, which will be pulled when the user opens the chat history or want to load message history. So let's look into the GCM services in details.

The GCM has two parts. One which receives the message and the other which handles the intent to process and show it. The first one is done by **GcmBroadcastReceiver.class** and the second one by **GcmIntentService.java**. The **GcmBroadcastReceiver** class first checks if this is the first time the app is opened. Then it gets the payload or message data from the GCM. Then it stores the message parameters in the session for temporary storage.

```
SessionManager session=new SessionManager(context);
        boolean bFlagForCurrent=session.isFirstScreen();
        userFacebookid=session.getFacebookId();
        Bundle extras = intent.getExtras();
        String message=extras.getString("payload");
        String action=extras.getString("action");
        strMessageType=extras.getString("mt");

        strMessageID=extras.getString("mid");
        String strSenderName=extras.getString("sname");
        String strDateTime=extras.getString("dt");
        String strFacebookId=extras.getString("sfid");

        session.setLastMessage(strFacebookId,
        message.substring(message.indexOf(":")        +1,message.length()));
```

Then it checks what type of action this GCM message wants us to do. First it checks if it's a message about a new match found and if not then it should be a normal chat message.

If it's a new match found, then it calls **findLikedMatched()** which will store the user's matches in the local database and show it to them in the right menu listview. We have already discussed how this method works in the previous section.

If the GCM message is about an actual chat message then we have to save this chat message to our local SQLite database and show it to the user by pulling it from there in real time. We will call the **BackGroundForSaveChat()** method, which will save the message in background.

The **GcmIntentService** class will take this message from the broadcast receiver and show it to the user. When the user clicks on this message then it redirect's the user to **ChatActivity.class** , which will open the chat page between user and the friend. Here we use the **NotificationManagaer** class which will take care of showing the notifications.

Settings module

Files to open and refer

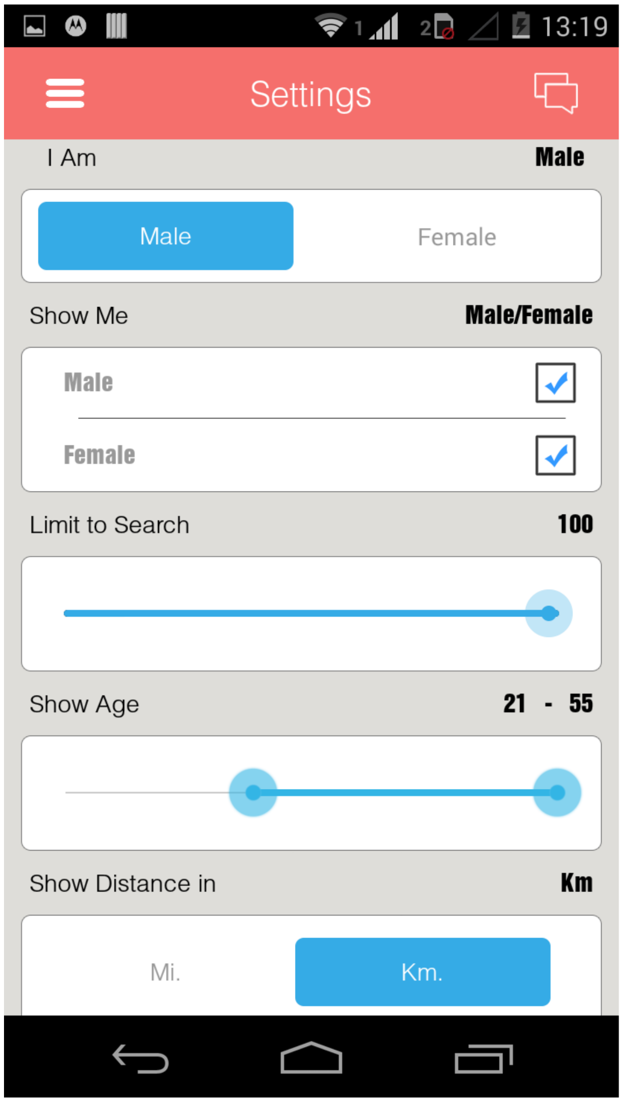
Please keep these files open, and follow the code inside to get a better understanding of this section.

- com.android.slidingmenu/SettingActivity.java

If you are directly opening files from the folder, follow these steps: * Un-zip ***Tinder-for-android.zip*** * Un-zip ***Android-Code.zip*** * Un-zip ***Flamer_Final.zip*** * Open the Flamer folder

- and goto this path ***Flamer/src/com/android/slidingmenu/SettingActivity.java***

The click handler of the settings button on the left menu will start a new activity called **SettingsActivity.java**. This class will show the available settings to the user. Let's discuss this class in details. First let's create a xml file called **settingactivity.xml** in the layout folder. This will create the view for the settings activity.



I Am Male

Show Me Male/Female

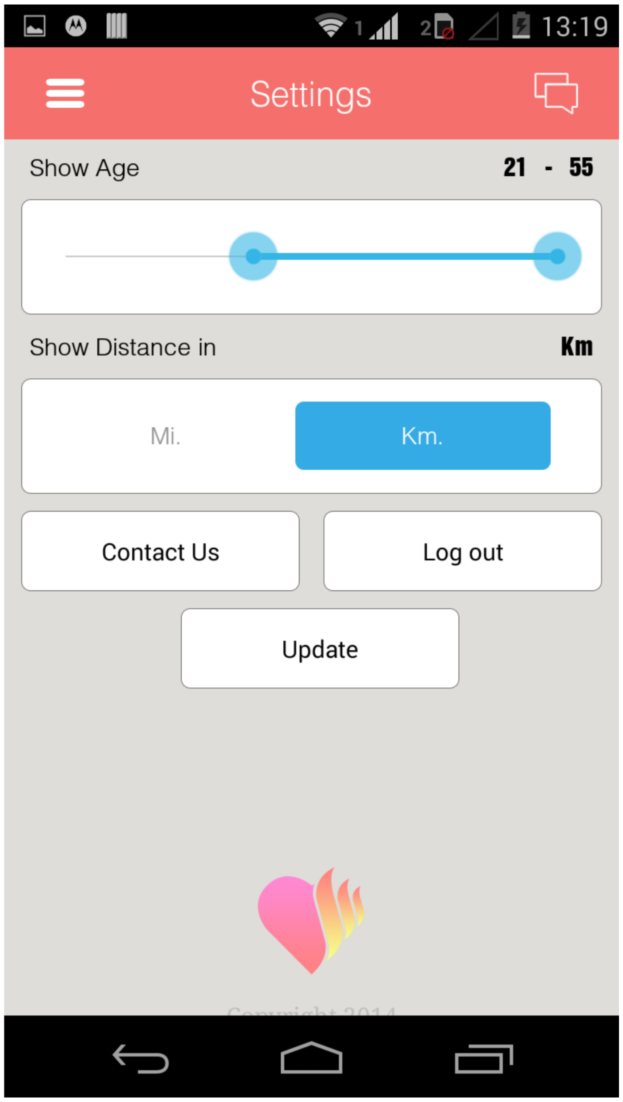
Male ✓

Female ✓

Limit to Search 100

Show Age 21 - 55

Show Distance in Km



Show Age 21 - 55

Show Distance in Km

We have a main scroll view layout which will hold our settings layout. We have gender selection layout, show male or female interest layout, limit of profile search layout, show age range layout, show search distance layout and buttons layout which will hold the contact US, Logout and Update button. All the layouts are simple layouts as we have used before except the search range bars. For this we use a **RangeBar** third party library. You can import it as a library project to our workspace.

Now in the **onCreateView**, we will first add the view and get a reference to the view elements. Then we will check the session state and see if the user has already been logged in. Initialize the rengebar Tick count and height. Now attach a lisetner to the range bar for age which will get the minimum and maximum age specified by user. Also let's get the gender of the current user which we saved previously in the session during facebook login. Based on the gender we will set the Male or female button to on or off. We will attach a seekbar change listener to the distance search and change the text view of distance according to that. We can get the user's preferred distance unit and set the distance unit buttons ON or OFF. We will add a listener to the male/female checkbox which will set the user preferences for the gender of interest.

Now we can listen to the button clicks. If user clicks on the contact us button then we will initiate an email to the admin by redirecting the user to choose an email service. Android will automatically detect the intent and choose an email service from available options. On click of the logout button we will logout the user from the app. For this we first get the current session token and device id before executing **BackGroundTaskForLogout()**. Let's see how this method works,

```
private class BackGroundTaskForLogout extends AsyncTask<String, Void, Void> {

                private String response;
                private boolean responseSuccess;
                private List<NameValuePair> logOutParameter;
                private LogOutData logOutData;
                private Ultilities mUltilities = new Ultilities();

                @Override
                protected Void doInBackground(String... params) {

                try {
                logOutParameter = mUltilities.getLogOutParameter(params);

                response = mUltilities.makeHttpRequest(Constant.logout_url,
                Constant.methodeName, logOutParameter);
                Gson gson = new Gson();
                logOutData = gson.fromJson(response, LogOutData.class);


                if (logOutData.getErrFlag() == 0
                && logOutData.getErrNum() == 41) {
                Session session = Session.getActiveSession();
                if (!session.isClosed()) {

                session.closeAndClearTokenInformation();
                } else {

                                }
                } else if (logOutData.getErrFlag() == 1
                && logOutData.getErrNum() == 31) {
                Session session = Session.getActiveSession();
                if  (!session.isClosed()) {

                session.closeAndClearTokenInformation();
```

```

                                } else {
                                        }
                        }

                } catch (Exception e) {

                }

                return null;
        }

        @Override
        protected void onPostExecute(Void result) {
                super.onPostExecute(result);
                try {
                mDialog.dismiss();
                if (logOutData.getErrFlag() == 0
                && logOutData.getErrNum() == 41) {
                SessionManager mSessionManager =
                new SessionManager(
                        getActivity());
                mSessionManager.logoutUser();
                Intent intent = new Intent(getActivity(),
                                LoginUsingFacebook.class);
                intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                startActivity(intent);
                getActivity().finish();
                } else if (logOutData.getErrFlag() == 1
                && logOutData.getErrNum() == 37) {
                ErrorMessage("Alert", logOutData.getErrMsg());
                } else if (logOutData.getErrFlag() == 1
                && logOutData.getErrNum() == 31) {
                ErrorMessageInvalidSessionTOken("Alert",
                logOutData.getErrMsg());
                        }

                } catch (Exception e) {
                }

        }

```

```
                @Override
                protected void onPreExecute() {
                        super.onPreExecute();

                        try {
                mDialog = mUltilities.GetProcessDialog(getActivity());
                        mDialog.setMessage("Please wait..");
                        mDialog.setCancelable(true);
                        mDialog.show();
                        } catch (Exception e) {
                           }
                }
        }
```

Here we will make a HTTP request to the logout URL of the remote server. Once we get a success response from the server we will clear all the session token's of the user which will log out the user. Once we log out the user, we will redirect the user to the **LoginUsingFacebook** class on the **onPostExecute()** method.

Let's see how we can update user data once user changed their preferences. On the onClick listener of the Update Button we will call **updateUserPreference()** method & we will first get the age preference, sex preference, distance preference and the user's distance preference. Then we call the **BackgroundTaskForUpdatePrefrence()** which will update the user preference to the remote server. This method will make a HTTP request to updatePreference_url which will save the required data in the remote database. Once we get the success response we will set the user preferences in the session for further use.

Invite Activity

We have an invite button for the logged in user to invite friends to chck out the app. This invite button is visible both in the left menu and in the home page, if there is no profile matches available. This invite button will start message sharing intent which will choose one of the sharing option to share the message.

```
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, "This is my text to send.");
sendIntent.setType("text/plain");
startActivity(sendIntent);
```