ABSTRACT

Socially active users are finding it a challenge to keep up with multiple social networks and communication channels in today's world. Users log in to each and every individual network to compose new posts, check what’s happening in their streams, follow up on conversations and follower requests, and basically perform any other little bit of social media management that they have on their plate. There is a need for an effective and integrated means of handling multiple social media accounts whether its users juggling their personal brand in multiple places or handling a mix of company profiles and personal profiles. Our Android app helps users manage and schedule posts to multiple social media profiles on Twitter, Facebook, LinkedIn and Instagram. This app allows users to publish and share photos to all social networks at once, tweet using their twitter account, post on Facebook timeline and LinkedIn. Users can also manage multiple social network accounts by adding or deleting them.
# TABLE OF CONTENTS

1. BACKGROUND AND RATIONALE
   1.1 Introduction
     1.1.1 Android
   1.2 Existing Applications
     1.2.1 Hootsuite
     1.2.2 Buffer
   1.3 Proposed Solution

2. NARRATIVE
   2.1 Problem Statement
   2.2 Motivation
   2.3 Product Description
   2.4 Product Scope
   2.5 System Requirements

3. APPLICATION DESIGN
   3.1 Application Design and Architecture
     3.1.1 OAuth
     3.1.2 REST Services
     3.1.3 JSON
   3.2 Design Flow
   3.3 Use Case Diagram
   3.4 Class Diagram
   3.5 User Interface
LIST OF FIGURES

Figure 3.1: The System Design.................................................................8
Figure 3.2: End User Design Flow............................................................10
Figure 3.3: Use Case Diagram.................................................................11
Figure 3.4: Class Diagram.................................................................12
Figure 3.5: Login Screen.................................................................13
Figure 3.6: Pop-Up with the Email ID’s.........................................................13
Figure 3.7: Adding Social Networks........................................................14
Figure 3.8: Navigation Drawer.............................................................15
Figure 3.9: View Timeline and Post Comment.............................................17
Figure 3.10: Post Message.................................................................18
Figure 3.11: View Profile.................................................................19
Figure 3.12: View Friends/Contacts......................................................20
Figure 3.13: View Posted Messages......................................................21
Figure 4.1: Login Module.................................................................23
Figure 4.2: Navigation Drawer Activity..................................................24
Figure 4.3: Check if the User is Logged into the Social Networks...............26
Figure 4.4: Display the List of All Social Networking Sites..........................27
Figure 4.5: Facebook Login.................................................................27
Figure 4.6: LinkedIn Login.................................................................28
Figure 4.7: Twitter Login.................................................................29
Figure 4.8: Display the List of Successfully Authenticated Providers..........29
Figure 4.9: Timeline of Twitter..........................................................31
Figure 4.10: To Retrieve Messages Associated with the User’s Facebook............32
Figure 4.11: Display Two Actionbar Menu Items........................................34
Figure 4.12: Post Messages to the Selected Social Networks..........................35
Figure 4.13: Publish Permission for Facebook.............................................35
Figure 4.14: Twitter Profile..........................................................................37
Figure 4.15: Instagram Profile.......................................................................37
Figure 4.16: View Friends/ Contacts............................................................39
Figure 4.17: Logout......................................................................................39
Figure 5.1: Add Content to Post.................................................................41
Figure 5.2: Error Message on Exceeding the Character Limit on Twitter.........42
Figure 5.3: Posting a Comment.................................................................43
1. BACKGROUND AND RATIONALE

1.1 Introduction

Humans have always invented different ways of interacting with friends and family across long distances, which has always been their concern for centuries. As social animals, humans have always relied on communication to strengthen their relationships. When face-to-face discussions are impossible or inconvenient, humans have dreamed up plenty of creative solutions. The earliest methods of communication between humans were through written correspondence delivered by hand from one person to another. The earliest form of postal service dates back to 550 B.C., and this primitive delivery system would evolve and get more sophisticated in future centuries. Telegraph, telephone and radio were some innovations which enabled humans interact with each other. With the advent of internet, the entire landscape of communication patterns changed, and with that, the methods of communication radically underwent transformation.

The earliest form of internet based social network was Bulletin Board System or BBS in short [1]. These online meeting places allowed users to communicate with a central system where they could download files or games and post messages to other users. Classmates.com and SixDegrees.com pushed the concept a bit further by enabling users to create profiles, locate long-lost grade school chums, menacing school bullies and maybe even that prom date they just couldn’t forget – all under the theory that no person is separated by more than six degrees from another. Websites like Friendster, LinkedIn, Myspace and Facebook pushed the envelope further conceptually and technically, and in the process underwent diversification. LinkedIn specialized as a network focused on
business people, YouTube transformed the way users shared and viewed videos and Twitter and Yammer radicalized the way users conveyed their thoughts.

While social networks aged to mature and started to lose its sheen, innovation and a few standout features helped Facebook and Twitter to achieve success where other platforms faltered [2]. For Facebook, opening up the API made it possible for third-party developers to create applications that worked within Facebook itself. The other key to success was Facebook’s ubiquitous ‘Like’ button. As more users joined the social network bandwagon, even businesses began to adopt all the social media platforms to connect to their audience. This helped them reach out to customers and promote their brand in ways which was not possible earlier. Adoption of social networks also provided businesses with tremendous insights, which they could subsequently mine to gather more information about consumers and their behavioral patterns. In addition to this, there was/is a drastic rise of smartphone adoption which changed the way content is published and consumed. Also, the usage patterns of consumers changed resulting in multiple opportunities for developers to mine into data and obtain insights. Since the time Facebook opened up its API, other social networks have followed suite enabling developers to create applications which could leverage federated authentication mechanism. This has resulted in an opportunity for developers to aggregate platforms/applications/feeds, and users to be omnipresent, who can ‘like’ or “tweet” just about everything even when they’re not on Facebook or Twitter’s official applications.

1.1.1 Android

Android is one of the top three mobile operating systems which has contributed significantly to the usability of mobile devices. As of today, Android commands over
80% of the smartphone market – the other major force being Apple’s iOS. The latest version of Android in use is Lollipop, with Android M coming up in the near future. When it initially started off, Android’s openness resulted in it being adopted by multiple OEM’s thereby causing major issues with regards to device fragmentation. Different OEM’s had different standards which resulted in inconsistent behavior of applications on various devices. Since Android IceCream Sandwich, Google has streamlined its approach towards OEM’s thereby ensuring a certain degree of consistency with regards to OEM’s – resulting in much needed sanity to developers. With the latest Lollipop, Android has come up with a set of guidelines under the concept of Material Design to provide users a similar experience when browsing for content across various devices and OS versions. Android is a major force to reckon with in today’s world, and it is imperative that any application which is developed actually runs on Android.

1.2 Existing Applications

1.2.1 Hootsuite

Hootsuite for Twitter and Social allows users to manage and schedule posts to multiple social media profiles on Twitter, Facebook, LinkedIn, Foursquare and more, all from one app. Hootsuite also allows users to publish and share photos to all their social networks at once and schedule a post to Facebook or a tweet to Twitter to be sent at a later time or day. Hootsuite is free for up to five social network accounts.

1.2.2 Buffer

Buffer allows users to share on Twitter, Facebook, Google+, LinkedIn and App.net. It also allows users to schedule their updates to post at optimal times, whether they're around or not. Buffer enables users to follow-up on their posts with its analytics to
see which posts performed best. Buffer also works from news apps such as Flipboard, Zite, Taptu, TweetDeck, Evernote, Pocket, Instapaper, Pulse, Feedly, UberSocial, Plume, Seesmic, Google Currents and almost all other reader apps. This app also gives Twitter analytics, Facebook analytics and LinkedIn stats.

1.3 Proposed Solution

The purpose of this application is to manage and access the social networking sites from a single app rather than accessing them individually. When the user logs in the home screen shows a list of different social networking sites from which the desired networks can be chosen. The application provides the facility of accessing the timelines of the chosen networks and allows to post a message to all the networks at a time or the user can also post the messages/photos to a certain network from the chosen networks. The user is also allowed to view his post to different networks in a single page and to which network it has been posted. The app reduces the redundancy of posting things multiple times and also helps the users in managing their accounts more efficiently.
2. NARRATIVE

2.1 Problem Statement

In the past few years, there has been an explosion of social media usage with a diverse audience irrespective of gender, demographics and age. Concurrently there has also been a fragmentation of user base across multiple social platforms based on choice, preferences or early adoption. As a result, users are finding it rather difficult to keep in touch and communicate with their intended audience as there are multiple mediums of communication that they are forced to adopt. This has resulted in often disgruntled users who get frustrated quite often because of lack of a uniform way in which they can communicate. Business users also find it quite frustrating to communicate with their customers across social networks through a multitude of apps, often resulting in dissatisfied customers, and possibly, reduced sales or customer satisfaction.

2.2 Motivation

To alleviate the problems that regular and business users face while interacting with social networks, an Android application can be created which will enable them to access multiple social networks at a time, reduce redundant tasks and provide a seamless user experience.

2.3 Product Description

Our Android app will help users manage and schedule posts to multiple social media profiles on Twitter, Facebook, LinkedIn and Foursquare. This app allows users to publish and share photos to all social networks at once, tweet using their twitter account,
post on facebook timeline and linkedin. Users can also manage multiple social network accounts by adding or deleting them.

Our Android app will do the following steps to enable access to multiple social networks:

- User opens the app and chooses the provider to request the authentication.
- If the native app of the selected social network is installed, it is opened – else the user is redirected to the selected provider's login site where they enter their credentials.
- Upon successful login, provider asks for user’s permission to share their basic data with our app.
- Once the user accepts it, on successful authentication the user is redirected to our android application.
- Once the above steps are executed successfully, more information can be retrieved from the corresponding social network, like user profile, contacts list, etc.

2.4 Product Scope
This Product is developed for Android smart phone users where the users can avail the service of SociALL only when they are connected to internet because the interaction with the social networking sites is done through internet. To access the app the user must authenticate with the app. This application is compatible on different versions of Android, such as starting from the minimum sdk version of Android 3.0 (Honeycomb) to recent update Android 5.0 (Lollipop).


2.5 System Requirements

To develop this android application, the requirements are mentioned below.

- Operating System: Windows
- Development Environment: Android Studio
- Software: Android Software Development Kit, Java SE 7
- Android Device: Android phone or tablet
3. APPLICATION DESIGN

3.1 Application Design and Architecture

![System Design Diagram]

**Figure 3.1: The System Design**

The architecture diagram, as shown in Figure 3.1, describes the design of the proposed application. The application can be installed on multiple android mobile devices. Users of the application should be authenticated before they can access information via SociALL. This authentication is done using Google API over the internet. When the user tries to sign-in, a list of all the registered email address in play store is presented to him, and he can select one email address to login to SociALL. For this purpose the user should have internet connectivity on the device. After login into the app the user can gain access to the social networking sites using the respective SDK’s or via RESTful services. Once the networks are accessed, all interactions with the social networks will happen via RESTful services. Users are allowed to retrieve/post to the timelines of multiple social networks at a time.
3.1.1 OAuth
The OAuth authorization framework enables a third-party application to obtain limited access to an HTTP service [3]. OAuth is a simple way to publish and interact with protected data. It's also a safer and more secure way for organizations to give access to users. OAuth allows users to access their data while protecting their account credentials.

3.1.2 REST Services
RESTful (Representational State Transfer) services are built to work best on the Web. In the REST architectural style, data and functionality are considered resources and are accessed using Uniform Resource Identifiers (URIs), typically links on the Web [4]. The resources are acted upon by using a set of simple, well-defined operations. The REST architectural style constrains architecture to client/server architecture and is designed to use a stateless communication protocol, typically HTTP. In the REST architecture style, clients and servers exchange representations of resources by using a standardized interface and protocol.

3.1.3 JSON
JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write [5]. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

JSON is built on two structures:
A collection of name/value pairs and
An ordered list of values.

3.2 Design Flow

Figure 3.2: End User Design Flow

Figure 3.2 shows the basic design flow of how the different functionalities interact. First the user needs to login into the application. If the user is using the application for the first time then the user needs to enable all the social networking sites and pass all the validations for the networking sites. If the validations are not passed then the user is not allowed to view any information. If the validations are passed the user is directed to the main activity. If the user is an existing user then the user is directly directed to the main activity. In the main activity the user has the option to add more social networks or view the messages or post messages to multiple networks or view all the posted messages. Users can logout from any of these activities using the logout menu item in navigation drawer.
3.3 Use Case Diagram

As shown in Figure 3.3, the use case diagram shows the interactions between the user and different modules of the application.

![Use Case Diagram](image)

**Figure 3.3: Use Case Diagram**

The user logs into the application and then can add different social networking sites. The user can view the messages from different social networks or post messages to different networks at a time, view profile and view the contacts.
3.4 Class Diagram

Figure 3.4 depicts a class diagram for viewing messages. Once the user clicks on View Messages Fragment, the messages are retrieved from the corresponding logged in social networks and displayed. ViewMessagesFragment is a class which extends Fragment and has an instance of a list view named “lstMessages”. GetInstagramMessages is one method which we use to retrieve messages from Instagram social network. Similarly, there are methods which retrieve messages from other social networks. To display the resulting messages in ListView, we use a custom adapter called MessageAdapter. Data is loaded into the custom adapter and bound to the listview. MessageEntity is a class which constitutes the variables and properties of a message.

![Class Diagram Image]

Figure 3.4: Class Diagram
3.5 User Interface

3.5.1 Login

Figure 3.5: Login Screen  Figure 3.6: Pop-Up with the Email ID’s

Logging into SociALL requires the user to use one of the registered gmail accounts in play store as shown in Figures 3.5 and 3.6. Google Play Services SDK is used to perform authentication. An instance of Google API client is created and the “connect” method of Google API client is used to connect. Profile information is fetched once the user has successfully logged in.
3.5.2 Add Social Network

Users can add social networks in the Add Social Network screen. Currently, SociALL supports Facebook, Twitter, LinkedIn and Instagram providers as shown in Figure 3.7. Material design compliant floating action button is used. Clicking on the floating action button prompts a material design compliant popup window where users can select social provider and authenticate. Helper class is created for each provider to check whether the user is logged into the corresponding social network or not. Networks in which the user is already logged in, are displayed in the list on fragment load.

Figure 3.7: Adding Social Networks
3.5.3 Navigation Drawer

As per material design guidelines, the UI for SociALL is created by importing an excellent library com.mikepenz:materialdrawer:3.0.8. The menu is displayed as a navigation drawer as shown in Figure 3.8. Primary and secondary menu items are created. The corresponding fragments get loaded on clicking each menu item.
3.5.4 View Timeline/Messages

(a)
In this fragment, the users can view messages from all the social networks in which he/she has logged in. Currently, apart from LinkedIn, all other messages are retrieved. A tab based layout is created using ViewPager control for usability. Each tab has the messages displayed for the particular social network as shown in Figure 3.9. Users can post comments on facebook posts as shown in Figure 3.9. This screen also allows users to index all posts from logged in providers. Indexing is done using a library called Lucene. Once messages are indexed, they can be searched from the Action Bar search. Full text search is done using Lucene and results are returned within milliseconds.
3.5.5 Post Message

User can post messages to multiple providers in which he has logged in, at once. Instagram does not expose an API to post, and hence only the supported providers are Facebook, Twitter and LinkedIn as shown in Figure 3.10. The check boxes corresponding to the non-logged in providers are disabled. If the Twitter provider is selected, a text listener is used to count the number of characters in the message – it changes to red when count goes beyond 140. A post button is present in the Action bar menu. Clicking it will post a message to all selected providers.
3.5.6 View profile

Figure 3.11: View Profile

Users can retrieve profile information from logged in providers. Profile image and name are the common properties as shown in Figure 3.11. There are some providers which return additional details which are displayed – for example, Facebook returns date of birth and email address (given proper permissions).
3.5.7 View Friends/Contacts

![Contacts Screen](image)

**Figure 3.12: View Friends/Contacts**

This fragment displays contacts only from Facebook as shown in Figure 3.12. This can be extended to display contacts/followers from other providers too. Facebook taggable friends are retrieved in this screen as getting friends need special permissions for SociALL application from Facebook. Clicking on friend opens the Facebook app. If the mobile does not have the app installed or is unable to open the app, it opens a browser window.
3.5.8 View Posted Messages

![View Posted Messages](image)

Figure 3.13: View Posted Messages

As indicated in Section 3.5.5, messages can be posted to multiple providers from FragmentPostMessage Fragment. In order to make it convenient for users, all the posted messages are stored in a Lucene index in an external storage. This fragment retrieves posted messages and shows it to the user sorted by date as shown in Figure 3.13. The user can view all posted messages across providers.
4. IMPLEMENTATION OF APPLICATION MODULES

4.1 Login

Logging into SociALL app is currently achieved using Google API authentication for Android. Google has federated its authentication mechanism which enables users to use Google authentication in SociALL. Dependencies are added in build.gradle file to import google packages within the application. A configuration file is generated by going to http://developers.google.com, which is included within the application’s app directory [6]. To generate the configuration file, the user needs to provide the app name, app namespace and SHA1 key. Once these are provided, the user can download a JSON file “google-services.json” and include it in “app” folder. The authentication code includes creating an instance of GoogleApiClient to request access to the basic user profile of the user. On clicking of signing button, a list of all the accounts added on the device will be prompted, and the user can choose the account that he/she wants to login to SociALL.

```java
mGoogleApiClient = new GoogleApiClient.Builder(this)
    .addConnectionCallbacks(this)
    .addOnConnectionFailedListener(this)
    .addApi(Plus.API)
    .addScope(new Scope(Scopes.PROFILE))
    .build();

//Retrieve Profile Information
protected void UpdateProfileInformation() {
    try {
        if (Plus.PeopleApi.getCurrentPerson(mGoogleApiClient) != null) {
            Person currentPerson = Plus.PeopleApi.getCurrentPerson(mGoogleApiClient);
            MyUtils.displayName = currentPerson.getDisplayName();
            MyUtils.displayPictureUrl = currentPerson.getImage().getUrl();
        }
    } catch (GoogleApiException e) {
        // Handle exception
    }
}
```
MyUtils.email = Plus.AccountApi.getAccountName(mGoogleApiClient);

try {
    MyUtils.bimage = MyUtils.drawableFromUrl(MyUtils.displayPictureUrl);
} catch (IOException e) {
    e.printStackTrace();
}

} catch (Exception e) {
    e.printStackTrace();
}
}

Figure 4.1: Login Module

As shown in Figure 4.1, the first part of the code creates an instance of Google client API which is subsequently used to login user as well as to retrieve profile information. Apart from connection callbacks and connectionfailed listener, the scope and API can also be passed while instantiating the class. The second part of the code is when successful authentication is performed. This code checks whether the CurrentPerson object is null or not, and gets the properties of the logged in user, if it is not null.

4.2 Navigation Drawer

Once the user is logged in, the user is presented with a Google standardized Navigation Drawer with material design. This UI is created by importing an excellent library com.mikepenz:materialdrawer:3.0.8. Then the primary and secondary menu items can be created using the code as shown in Figure 4.2.

Important Code Snippet

```java
//Create the drawer
result = new DrawerBuilder()
    .withActivity(this)
    .withToolbar(toolbar)
    .withAccountHeader(headerResult) //set the AccountHeader
    .addDrawerItems(
```
Figure 4.2: Navigation Drawer Activity

As shown in Figure 4.2, the code creates an instance of Material design navigation
drawer. An instance of DrawerBuilder class is created and can set multiple properties of
the navigation drawer by passing in relevant input parameters. Account header is the
place where the logged in profile information is displayed. PrimaryDrawerItem and
SecondaryDrawerItem are instances which can be created – the differentiator being the
look and feel. In general, settings and other “generic” menu items can be categorized under SecondaryDrawerItem.

On clicking each menu item, the corresponding fragment will be loaded and the appropriate layout displayed. Dependency is added in build.gradle file. Theme is created in the “res/styles.xml” file and applied in the AndroidManifest.xml file. This activity is the parent activity which displays all child fragments.

4.3 Add Social Networks
This screen enables user to do two things:

- It lists out all the social networks in which the user has previously logged in.
- It also helps the user to login to non-logged in social networks.

For each of the social networks, a custom implementation of authentication using the sdk provided by the corresponding provider is executed. Twitter provides fabric (http://docs.fabric.io/android/twitter/introduction.html) [7], Facebook provides its own sdk (https://developers.facebook.com/docs/android) [8] with most features enabled using Graph API, LinkedIn allows us to download their sdk at https://developer.linkedin.com/docs/android-sdk [9] and Instagram has just recently (relatively) forayed into exposing their API for developers (https://instagram.com/developer/?hl=en) [10]. The dependencies need to be included in build.gradle file for Facebook and Twitter. Linkedin sdk code is added within the application whereas Instagram uses OAuth only – a browser based authentication model. Four helper classes which has methods to determine if a user has logged in to the corresponding social network or not – InstagramHelper, LinkedInHelper, FacebookHelper and TwitterHelper have been created.
If a user is already logged in (we have information stored in SharedPreferences), the list is populated with the corresponding social network. There is a add button at the bottom right of the fragment, which users can click to add social network. The button which helps users to add social networks and the popup window is created in line with the material design principles using a couple of libraries – kcom.melnykov:floatingactionbutton:1.3.0 and com.afollestad:material-dialogs:0.7.6.0. All lists within the application are bound to custom adapters, and the same applies here too.

```java
FacebookHelper facebookHelper = new FacebookHelper();
boolean facebookLoggedInStatus = facebookHelper.facebookCheckLogin();
if (facebookLoggedInStatus) {
    SocialNetworkEntity se = new SocialNetworkEntity("facebook", "facebook");
    arrSocialNetworkList.add(se);
    arrSocialNetworkListToBeShown.remove(se);
}
LinkedInHelper linkedInHelper = new LinkedInHelper();
if (linkedInHelper.LinkedInCheckLogin(getActivity().getApplicationContext())){
    SocialNetworkEntity se = new SocialNetworkEntity("linkedin", "linkedin");
    arrSocialNetworkList.add(se);
    arrSocialNetworkListToBeShown.remove(se);
}
```

**Figure 4.3: Check if the User is Logged into the Social Networks (Facebook, LinkedIn)**

As shown in Figure 4.3, the code creates instances of Facebook and LinkedIn helper classes and checks whether the user is logged into the respective networks or not – via SociALL. *arrSocialNetworkList* keeps track of the social networks in which the user is already logged in, whereas *arrSocialNetworkListToBeShown* has the list of providers
which needs to be displayed in the dialog box when the user wants to login to another provider.

```java
//Material dialog which creates a popup with list of providers to login
MaterialDialog.Builder(getActivity())
    .title("Choose a Social Network")
    .theme(Theme.LIGHT)
    .adapter(new SocialNetworkGetListAdapter(getActivity(),
        R.layout.lst_socialnetworks_item_withoutbutton,
        arrSocialNetworkListToBeShown),
```

**Figure 4.4: Display the List of All Social Networking Sites**

As shown in Figure 4.4, the code creates an instance of Material dialog. This dialog has the user interface which complies with the design guidelines defined by Google. We can set the title, theme and the adapter to the dialog.

```java
//Login to Facebook
LoginManager.getInstance().logInWithReadPermissions(getActivity(),
    Arrays.asList("public_profile", "user_friends", "user_posts", "user_photos", "email", "user_birthday"));
```

**Figure 4.5: Facebook Login**

As shown in Figure 4.5, the code is used to login to Facebook with read permissions. Multiple scope parameters can be passed as an arraylist to the above method. Each scope is equivalent to a role, which gives user permission to access pre-defined information associated with the role.

```java
//Logging into LinkedIn
LSessionManager.getInstance(getActivity().getApplicationContext()).init(getActivity(), LinkedInHelper.buildScope(), new AuthListener() {
    @Override
    public void onAuthSuccess() {
        arrSocialNetworkList.add(new SocialNetworkEntity("linkedin", "linkedin"));
        arrSocialNetworkListToBeShown.remove(new SocialNetworkEntity("linkedin", "linkedin"));
    }
```

adapter.notifyDataSetChanged();
Toast.makeText(getActivity().getApplicationContext(), "LinkedIn logged in", Toast.LENGTH_SHORT).show();
}

@Override
public void onAuthError(LIAuthError error) {
    // Handle authentication errors
    Toast.makeText(getActivity().getApplicationContext(), "LinkedIn error" + error.toString(), Toast.LENGTH_SHORT).show();
}
}, true);

Figure 4.6: LinkedIn Login

As shown in Figure 4.6, the code initializes a session with LinkedIn as an asynchronous task. onAuthSuccess() and onAuthError() are two callback methods which gets executed based on the success or failure of the login usecase. Once the user is successfully logged in, the list which displays the list of logged in users is refreshed with LinkedIn. Subsequent attempts to login to providers will remove LinkedIn from the dialog.

//Logging into Twitter
TwitterAuthClient client = new TwitterAuthClient();
client.authorize(getActivity(), new Callback<TwitterSession>() {
    @Override
    public void success(Result<TwitterSession> result) {
        Toast.makeText(getActivity().getApplicationContext(), "Login worked", Toast.LENGTH_LONG).show();
        System.out.println("UserId is " + result.data.getUserId());
        System.out.println("UserName is " + result.data.getUsername());
        System.out.println("Token is " + result.data.getAuthToken().token);
        System.out.println("Secret is " + result.data.getAuthToken().secret);
        arrSocialNetworkList.add(new SocialNetworkEntity("twitter", "twitter"));
        arrSocialNetworkListToBeShown.remove(new SocialNetworkEntity("twitter", "twitter"));
        adapter.notifyDataSetChanged();
    }
});
@Override
    public void failure(TwitterException e) {
        Toast.makeText(getActivity().getApplicationContext(), "Login failed", Toast.LENGTH_LONG).show();
    }
})

Figure 4.7: Twitter Login

As shown in Figure 4.7, the code initializes a session with Twitter as an asynchronous task. Success() and failure() are two callback methods which gets executed based on the success or failure of the login usecase. Once the user is successfully logged in, the list which displays the list of logged in users is refreshed with LinkedIn. Subsequent attempts to login to providers will remove LinkedIn from the dialog. The result of a successful authentication will return a TwitterSession object which contains profile information of the logged in user.

//Custom adapter which the listview binds to
adapter = new SocialNetworkGetListAdapter(getActivity(), R.layout.lst_socialnetworks_item_withoutbutton, arrSocialNetworkList);
lstLoggedInSocialNetworks.setAdapter(adapter);

Figure 4.8: Display the List of Successfully Authenticated Providers

As shown in Figure 4.8, the code creates an instance of custom adapter SocialNetworkGetListAdapter and binds it to the listview to display the list of successfully authenticated providers.

4.4 View Messages

View Message fragment retrieves messages from Facebook, Twitter and Instagram, if users are logged into it. If the user has not logged into any of the social networks a message is displayed notifying the same. A common entity is created - MessageEntity
which contains a consolidated view of all the provider’s messages. Only common message properties are included, though this entity can be extended to include specific properties also. A custom adapter is created which the listview will bind to. Each list item within the listview is a layout, which is lst_message_item.xml.

Since Twitter and Facebook provide callbacks, the methods are called synchronously. Instagram needs an http call, and since it does not provide any callback’s, the method of retrieving messages is encapsulated within a method, and called using AsyncTask for asynchronous callback implementation. ViewPagerAdapter implementation is done to display separate tabs for each provider.

This fragment also has a menu item in the action bar which can index all messages from multiple providers to which the user has logged in. Lucene is used to index all the messages across multiple providers. Lucene is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search. There is a helper class called LuceneHelper which has the methods for indexing and retrieving search results. The UpdateAndDeleteMessages takes a list of MessageEntity objects and performs indexing on all the messages and stores the information in Environment.getExternalStorageDirectory() + "/SociALL/index" folder. Once the index is created, users can perform full text search on the index to retrieve matching messages very quickly using the search text box in the Action Bar.
private void GetTwitterTimeline() {

    StatusesService statusesService = Twitter.getApiClient().get_statuses_service();
    statusesService.userTimeline(Twitter.getSessionManager().getActiveSession().getUserId(), null, 50, null, null, null, null, null, null, new Callback<List<Tweet>>() {

        @Override
        public void success(Result<List<Tweet>> listResult) {
            try {
                for (Tweet tweet : listResult.data) {
                    MessageEntity messageEntity = new MessageEntity();
                    messageEntity.setProvider("twitter");
                    messageEntity.setMessage(tweet.text);
                    messageEntity.setCreatedTime(getTwitterDate(tweet.createdAt));
                    arrMessageList.add(messageEntity);
                }

                adapter = new MessageAdapter(getActivity(),
                    R.layout.lst_message_item, arrMessageList);

                lstMessages.setAdapter(adapter);
            } catch (Exception ex) {
            }
        }

        @Override
        public void failure(TwitterException e) {
        }
    });
}

Figure 4.9: Timeline of Twitter

As shown in Figure 4.9, the code creates an instance of StatusService class which is used to get the timeline of a user. One of the input parameters is the userid, which is a long datatype, which is provided above. It has many other parameters, but only the count of returned results is sent in the above code. Success() and failure() are two callback methods which are called based on the success or failure of api call. Success() method
provides a List of Tweet objects which is aggregated into an array of MessageEntity classes to be displayed in the listview.

```java
private void GetFacebookMessages() {
    FacebookHelper facebookHelper = new FacebookHelper();
    if (facebookHelper.facebookCheckLogin()) {
        new GraphRequest(
            AccessToken.getCurrentAccessToken(),
            "/me/feed",
            null,
            HttpMethod.GET,
            new GraphRequest.Callback() {
                public void onCompleted(GraphResponse response) {
                    /* handle the result */
                    System.out.println(response);
                    try {
                        JSONObject resp = new JSONObject(response.getRawResponse());
                        JSONArray data = resp.getJSONArray("data");
                        System.out.println("Feeds count : " + data.length());
                        for (int i = 0; i < data.length(); ++i) {
                            messageEntity = new MessageEntity();
                            JSONObject obj = data.getJSONObject(i)
                            .executeAsync();
                        }
                    } catch (Exception e) {
                        System.out.println("Error: " + e.getMessage());
                    }
                }
            });
```

*Figure 4.10: To Retrieve Messages Associated with the User’s Facebook*

As shown in Figure 4.10, the code uses a GraphRequest API, which is a part of facebook android sdk to retrieve a list of messages associated with the user. This is an asynchronous call which invokes the OnCompleted() callback method when successful. Built in JSON parser is used to parse the result and populate into MessageEntity objects.
4.5 Post Message
The Post Message fragment is used to post messages to social networks. Currently, users can post to Facebook, Twitter and Linkedin, if they are logged in. Instagram has not yet provided an API to upload images – so it is not possible to achieve the same. This fragment has send button, which posts message to the selected social networks. The user is provided with the social networks and can check the check boxes of all the social networks or just the required ones to post the message An EditText box is used to input the message- and the send button will send the message to the selected providers. Facebook provides API which supports two methods of logging in – logInWithPublishPermissions and logInWithReadPermissions. To post a message, we need to switch the login mechanism, and as a result, for the first time, the user will be prompted to allow the app to provide additional permissions to Facebook to publish. Once it is accomplished, the rest of the tasks is seamless.

```java
//Displaying menu items
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.item_delete:
            if (arrMessageToBePosted.size() < 1){
                Toast.makeText(getActivity(), "Please select atleast one social network to post", Toast.LENGTH_LONG).show();
            } else {
                if (editText.getText().toString().trim().length() < 1){
                    Toast.makeText(getActivity(), "Please add content to post", Toast.LENGTH_LONG).show();
                } else {
                    PostMessage();
                }
            }
            break;
    }
    return true;
}
```
Figure 4.11: Display Two Actionbar Menu Items

As shown in Figure 4.11, the code overrides the onOptionsItemSelected method to display two actionbar menu items. Clicking on one menu item will help the user to add a list of logged in provider networks to which he/she can post the message. Clicking on the other menu item will post the message to the selected social networks.

```java
//Post message method
private void PostMessage() {
    String messageToBePosted = editText.getText().toString();
    for(String s : arrMessageToBePosted){
        if (s.equals("facebook")){
            AccessToken accessToken = AccessToken.getCurrentAccessToken();
            if (!accessToken.getPermissions().contains("publish_actions")) {
                facebookPost();
            }
        }
        Bundle params = new Bundle();
        params.putString("message", messageToBePosted);
        /* make the API call */
        new GraphRequest(
            AccessToken.getCurrentAccessToken(),
            "/me/feed",
            params,
            HttpMethod.POST,
            new GraphRequest.Callback() {
                public void onCompleted(GraphResponse response) {
```

```java
```
/* handle the result */

Toast.makeText(getActivity(), "Message posted to facebook successfully", Toast.LENGTH_SHORT).show();
}
}
).executeAsync();

if (s.equals("twitter")){

TwitterApiClient twitterApiClient = TwitterCore.getInstance().getApiClient(Twitter.getSessionManager().getActiveSession());

StatusesService statusesService = twitterApiClient.getStatusesService();

statusesService.update(messageToBePosted, null, null, null, null, null, null, null, new Callback<Tweet>() {

Figure 4.12: Post Messages to the selected Social Networks

As shown in Figure 4.12, the code posts a message to the selected social networks to which the user is logged in, and which are selected. For Facebook, a GraphRequest API async call is made to post information whereas Twitter uses the StatusService class. Based on the response status, the corresponding callback methods are invoked indicating the success or failure of the action.

//Code to check publish permissions and get, if not present
private void facebookPost() {
    AccessToken accessToken = AccessToken.getCurrentAccessToken();

    if (accessToken.getPermissions().contains("publish_actions")) {
        LoginManager.getInstance().logInWithReadPermissions(this, Arrays.asList("public_profile", "user_friends", "user_posts", "user_photos", "email", "user_birthday");
    } else {
        LoginManager.getInstance().logInWithPublishPermissions(this, Arrays.asList("publish_actions"));
    }
}

Figure 4.13: Publish Permission for Facebook
There are two ways in which users can login via OAuth to Facebook – one using read permissions and the other using publish permissions. Scope parameters indicate the scope of the request and the type of return values fetched. A user can login to Facebook using either one of the mentioned methods as shown in Figure 4.13. So, initially, the user logs in with read permission, and eventually, while posting to the social networks, especially for Facebook, we check whether the accesstoken currently has publish permissions – if not, we login using publish permissions before posting information. This use case is specific to Facebook only.

### 4.6 View Profile

This screen enables user to view the profile of users in all the four social networks. The number of profile properties provided by each provider is quite limited, and hence this screen has just basic information – profile picture, Username and email, if allowed (LinkedIn needs app to be whitelisted in order to access email address). Facebook provides date of birth additionally. ProfileEntity is the common entity and ProfileAdapter is the custom adapter to which the listview binds. Since Instagram profile API is synchronous, it is called using an AsyncTask - to make sure the UI does not freeze.

```java
//Get twitter profile
private void GetTwitterProfile() {
TwitterApiClient(Twitter.getSessionManager().getActiveSession()).getAccountService()
   .verifyCredentials(true, false, new Callback<User>() {
      @Override
      public void failure(TwitterException e) {
      }
      @Override
      public void success(Result<User> userResult) {
```
User user = userResult.data;

MyUtils.twitterDisplayPictureURL = user.profileImageUrl;
ProfileEntity profileEntity = new ProfileEntity();
profileEntity.setEmailAddress(user.email);
...

lstProfile.setAdapter(adapter);
if (progressDialog != null)
    progressDialog.dismiss();
}
})
}

Figure 4.14: Twitter Profile

verifyCredentials() is the method which is used to retrieve the profile information from Twitter as shown in Figure 4.14. This is an async call. Once the result is successful, the properties are stored in a common entity called ProfileEntity.

//Get Instagram profile using AsyncTask
private class GetInstagramProfile extends AsyncTask<String, Void, String> {

    private static final String API_URL = "https://api.instagram.com/v1";

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

    protected String doInBackground(String... urls) {
        try {
            InstagramSession mSession;
            mSession = new InstagramSession(getActivity().getApplicationContext());
            String urlString = API_URL + "/users/" + mSession.getId() + "/?access_token=" + mSession.getAccessToken();
            URL url = new URL(urlString);
            InputStream inputStream = url.openConnection().getInputStream();  
            String response = streamToString(inputStream);

            return response;
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    protected void onPostExecute(String result) {
        super.onPostExecute(result);
        lstProfile.setAdapter(adapter);
        if (progressDialog != null)
            progressDialog.dismiss();
    }
}

Figure 4.15: Instagram Profile
Since GetInstagramProfile profile involved making an http call and parsing JSON data – post successful API call, the method is called in a AsyncTask so that the UI is not frozen while profile information is retrieved. The doInBackground method does the main processing logic in the background, and once it is done, the onPostExecute method runs on the UI thread, thereby enabling screen modifications/refresh.

4.7 View Friends/Contacts
Currently, only Facebook friends are retrieved. Even Facebook friends are retrieved using a workaround. Facebook does not allow an API to access friends unless exclusive permission is asked for the same. Facebook provides a list of friends who are “taggable” and this API is called to display friends in this screen. This screen can be extended to include friends from other providers - Twitter followers and Instagram followers. If this screen is extended to other providers, the generic ContactEntity can include additional fields specific to other providers. On clicking the contact, Facebook app is opened, if available and instantiable, else a browser window will open allowing the user to navigate to Facebook.

```java
//Code to retrieve Facebook taggable friends
private void GetFacebookContacts() {

    progressDialog = new MaterialDialog.Builder(getActivity())
        .title("Please wait....")
        .theme(Theme.LIGHT)
        .content("Fetching taggable contacts from Facebook...")
        .progress(true, 0)
        .show();

    if (MyUtils.facebookID != null) {
        new GraphRequest(
            AccessToken.getCurrentAccessToken(),
            "facebookID/taggable_friends",
            null,
            HttpMethod.GET,
            new GraphRequest.Callback() {
                public void onCompleted(GraphResponse response) {
                    try{
```
/* handle the result */
System.out.println("Taggable friends" +
response.getRawResponse());
JSONObject resp = new JSONObject(response.getRawResponse());
JSONArray data = resp.getJSONArray("data");
for(int i = 0; i < data.length(); ++i) {
    ContactEntity contactEntity = new ContactEntity();
    JSONObject obj = data.getJSONObject(i);

    ...
}
} catch (Exception ex){
    ex.printStackTrace();
}
).executeAsync();

Figure 4.16: View Friends/ Contacts
This code again uses the GraphRequest API of Facebook sdk to fetch taggable contacts.
Since getting Facebook contacts requires submitting a request to Facebook (the process
takes five business days to complete, and involves lot of documentation, including
uploading apk file), SociALL displays the list of taggable friends. This is an async call
which invokes onCompleted() method on successful completion.

4.8 Logout
Clicking on Logout will log out the user and prompt him/her to sign in again when the
application launches.

if (mGoogleApiClient.isConnected()) {
    Plus.AccountApi.clearDefaultAccount(mGoogleApiClient);
    mGoogleApiClient.disconnect();
    ...
}

Figure 4.17: Logout
This code snippet just checks whether the user is logged in, and if he/she is logged in, it
uses the clearDefaultAccount() method of Google client class to logout.
5. TESTING AND RESULTS
This section describes end user application testing details. The main functionalities where user input are necessary:

1. Post Message
2. Post Comment

5.1 Post Message
There are three scenarios in which testing is performed in this usecase. When the user navigates to FragmentPostMessage, and clicks the “Send” button in the Action Bar, there are three validations that are performed:

1. If none of the logged in social networks are selected, the user is prompted to select at least one social network to post information (Figure 5.1 (a)).

2. If a social network is selected, but the message is empty, the user is prompted to enter a message (Figure 5.1(b)).

3. If the user wants to post to Twitter, and sends a message which is greater than 140 characters, Twitter returns a 403-Forbidden error, which is displayed to the user (Figure 5.2).
Figure 5.1: Add Content to Post
5.2 Post Comment

Users are allowed to comment on a post for Facebook provider. If the user does not enter any comment, he/she is prompted to enter a comment to be attached to a post as shown in Figure 5.4.

Figure 5.2: Error Message on Exceeding the Character Limit on Twitter
Figure 5.3: Posting a Comment
6. CONCLUSION AND FUTURE WORK

SociALL was envisioned to help reduce user redundancy in consuming information from multiple social networks in a seamless and standard manner. The same was achieved to a significant extent. SociALL helps ease of access to multiple social networks from one single application. SociALL also allows users to post content to multiple providers from one app. There is a provision to keep track of messages posted to multiple social networks. SociALL also bundles with a blazing fast full text search engine which can return matching results within milliseconds, irrespective of the number of messages.

Some future enhancements which can be done on SociALL include:

- Extending SociALL to support more providers.
- Enabling SociALL to display many other features for each provider (replies, mentions in Twitter, timeline in LinkedIn, etc.) if the API’s are available.
- Implementing fine grained search features -
  - sorting by date
  - sorting by provider
  - supporting operators in search keyword (keyword “and/or” keyword)
- Integration with phone contacts.
- Modification of data across providers.
REFERENCES


