Cygnus - Microblogging Platform

CS04-608 Mini Project 2010

Presented by

Aneesa N
Khadeeja Febin
Niya Simon C
Primal Pappachan
Vigil A J

under the guidance of

Mrs Rani Koshi
Lecturer

Department of Computer Science And Engineering
Government Engineering College, Thrissur

Acknowledgement

ACKNOWLEDGEMENT

We would like to take this opportunity to express our heartfelt and sincere gratitude to **Prof. Manoj Kumar**, the Head of the Department of Computer Science and Engineering, for permitting us to do the project and for the support he gave us throughout the project.

We hereby, honor **Prof.K S Valsaraj** and **Mr.Ajay James**, lecturer, for their encouragement and guidance in various stages of this project.

We would like to show our greatest appreciation to **Mrs. Rani Koshy**, lecturer, Department of Computer Science and Engineering , for her able guidance and useful suggestions and for the encouragement she gave us throughout the completion of the project work.

We also use this oppurtunity to convey our sincere thanks to all other teaching and non teaching staff of the department of Computer Science and Engineering for their encouragement and support.

Last but not the least we would like to express our sincere gratitude to all our classmates for being a constant source of encouragement throughout the completion of the project.

Above all, we thank God almighty who helped us through all our ways..

Abstract

Cygnus is a microblogging application for Android phones. Microblogging is a networking service that allows the subscriber to broadcast short messages to other subscribers of the service. It allows mobile users of cell phones to stay abreast of activities within a group by receiving frequent published updates, of 140 characters or less. Just like any other Internet tool, microblogging can be utilized for various purposes in fields such as business, education etc.

The front end of Cygnus was developed using Android SDK and back end using Django which is a Web development framework over Python. The communication between front end and back end was established through HTTP requests. The Application Programming Interface (API) for connecting between server and client was programmed in Python.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Once upon a time when life was simpler and slower-paced, people kept journals and wrote letters. Then, out of the blue Internet entered the scene, people still kept journals and wrote emails. When the Internet grew in popularity, people began to keep blogs and wrote emails. It was the turn of social networking next. Facebook, Social networking sites allowed people to share pictures, stories, links, etc. Those social networks grew, and the messages in emails grew shorter. That's when the age of microblogging began.

Sometimes or most of the times blogging applications like WordPress and Blogger are too much for what an user really wants. They really are content management systems. In cases where writing long-form content might not be the goal as in communicating with friends, or sharing links. Sometimes all what an user wants to do is put something on the web with the least amount of effort as possible. Its in those circumstances that microblogging comes to the fore.

Micro-blogging can be defined as a form of blogging that allows users to write brief text updates and publish them to be viewed by anyone. Popular examples of micro-blogging services include Twitter and FriendFeed. Microblog posts usually involve a few hundred characters or less. Instead of posting a message on their regular blog, people who microblog use Web services designed to make microblogging very easy.

Benefits of Micro-blogging include

- Useful in educational institutions for student-teacher interactions or maybe even higher staffs-employees relationships.
- Social learning for classrooms.
- Used as a tool to train communicative and cultural competence.
- A room of experts,friends, strangers.
- Can be used to share thoughts and ideas of presenter, presentations with both fellow conference attendees and the wider world.
- Listening into announcements, discussions or informal conversations about your organisation or the services provided by your organisation.
- To create social awareness and thereby organising support for and against contemporary issues.
- For work related discussions and informal chat

Cygnus is a microblogging application intended for Android [4] Mobile Operating System. The application interface should be accessible from the android emulator that comes along

with android sdk which is capable of running on computer as well as phones which are capable of running android. The backend was developed using Django [1] which is a web framework and the communication between the frontend and backend is established using RESTful services.

The reasons why we choose Android over any other platform were

- Gets to code in Java which is widely known.
- Can develop on any platform (Mac/Windows/Linux).
- Great Android devices coming out this year.
- Easy to do XML way of laying out views.
- SDK can be integrated with Eclipse IDE.
- Huge and vibrant developer community.

The main page of Cygnus have a public timeline, which lists all of the latest posts from users. A user can have a timeline of your own updates, favourites etc. It also allows logged in users to subscribe to (also referred to as following) an user's updates, or at the very least read them. Cygnus, even though developed and tested on Android Virtual device that comes with Android SDK, can be installed on any phone which supports from Android 2.1 onwards.

Chapter 2

Requirement Analysis

2.1 Hardware and Software Requirements

The microblogging application is intended to work in the mobile device emulator, a virtual mobile device that runs on the computer which comes as part of Android Software Development Kit. Eclipse IDE 3.5(Galileo) is used as the devel- opment environment.The Android Development Tools (ADT) plugin for Eclipse adds powerful extensions to the Eclipse integrated development environment and allows for creation and debugging Android applications in an easier and faster route. We preferred to use Linux as the operating system for development even though Android SDK is supported in all major operating system platforms. The table below provides a rough idea of the diskspace requirements to expect, based on the components.

Component type	Approximate size	Comments
SDK Tools	50 MB	Required.
Android platform	150 MB	Required.
SDK Add-on	100 MB	Optional.
Onine documentation	250 MB	Optional.

Table 2.1: Disk Space Requirements

Python is used for developing Application Programming Interface(API) for communication between user interface in Android and the database implemented in SQLite. The API uses SQLite's functionality through simple function calls which is a relational database management system contained in a relatively small C programming library. The hardware require- ments for Python and SQLite are minimal.

2.2 Detailed Functionalities

2.2.1 Login

This window is displayed when the application is loaded. New users need to sign up by providing details like user-name, password and so on. Authentication of an already signed up user is done by verifying the validity of user-name and password combination. Depending on the result of authentication; authorization of permissions is done. This window is most important along with home window for the basic functionality of the application.

2.2.1.1 Use Case 1: Sign in

Primary Actor: User
Precondition: Nil
Main Scenario:

- Start the application. User prompted for user-name and password.
- User gives the user-name and password.
- System does authentication.
- Home window is displayed

Alternate Scenario:
Authorization fails

- Prompt the user that he typed the wrong password.

2.2.1.2 Use Case 2 : Sign up

Primary Actor: User
Precondition: Nil
Main Scenario:

- User initiates the 'Sign up' Functionality
- System asks the user for the necessary details.
- User enters all the mandatory details at least.
- User is signed up and the home window is displayed.

Alternate Scenario:
User with same name exists.

- System asks the user for a different name.
- User enters a different name.
- The name is checked again for uniqueness.

2.2.2 Home

After the authentication is performed in the login window, home screen is displayed. This window contains a time-line showing the various posts, followers and followees specefic to an user. It also provides features to delete posts and change settings.

2.2.2.1 Use Case 1: Post message

Primary Actor: User
Precondition: User Logged in
Main Scenario:

- User types the message in the space provided.
- The 'post' button is clicked after completing message within the constraints.
- System adds the message to the time-line with a time stamp.

Alternate Scenario: Any of the constraints are violated.

- The user is notifed of the violation.
- User makes the necessary amendment
- The message is validated again.

2.2.2.2 Use Case 2: Delete message

Primary Actor: User
Precondition: User Logged in
Main Scenario:

- User presses the delete button under the required post.
- A dialog box for confrmation of deletion is displayed.
- User confrms the deletion of the message.
- System deletes the message from the time-line.

Alternate Scenario: The response to the confirmation window is negative.

- Message is not removed from time-line.

2.2.2.3 Use case 3: More

Primary Actor: User
Precondition: User Logged in
Main Scenario:

- User presses the More button.
- A new window with a short bio on the user and options followers and followees are shown.
- User can select either of followers or followees.
- The list of followers or being followed upon is displayed.
- List has options for managing followers or followees.

2.2.2.4 Use Case 4: Settings

Primary Actor: User
Precondition: User Logged in
Main Scenario:

- The user profile is displayed in a window.
- User can edit various aspects of profile like nickname, password etc.
- After completion of editing user should press 'save changes' button.
- System updates the profile based on the changes made.

Alternate Scenario: A void change is made in the profile.

- System displays an error message and leaves profile intact.

A false change is made in the profile.

- System displays an error message and leaves profile intact.

2.2.2.5 Use Case 5: Reply

Primary Actor: User
Precondition: User Logged in
Main Scenario:

- User presses the Reply button under a post.
- The name of the user to which reply is intended is displayed in the text box for 'posting.'
- User can type the message.
- Upon completing message Post button is pressed.

2.2.3 Use Case 6: Like

Primary Actor: User
Precondition: User Logged in
Main Scenario:

- User presses the Reply button under a post.
- An icon showing that the user liked the post is shown below the post.
- The like button is disabled under the post.

2.2.3.1 Use Case 7: Logout

Primary Actor: User
Precondition: User Logged in
Main Scenario:

- User presses the Log out button.
- The user is redirected to the Login Screen

2.2.4 Search

Allows the user to search for people, posts or channels containing a keyword.

2.2.4.1 Use case 1: Search for people

Primary Actor: User
Precondition: User Logged in
Main Scenario:

- User enters the name of the friend to be searched in the text box.
- System tries to find occurrences of the name among valid users.
- The list of successful matches is displayed.

Alternate Scenario:
Search is unsuccessful.

- Message displaying possible corrections is displayed.
- An empty list is displayed

2.2.4.2 Use case 2: Search for post

Primary Actor: User
Precondition: User Logged in
Main Scenario:

- User enters the name of the post to be searched in the text box.
- System tries to find occurrences of the keyword among posts.
- The list of successful matches is displayed.

Alternate Scenario:
Search is unsuccessful.

- Message displaying possible corrections is displayed.
- An empty list is displayed.

2.3 Scope of the project

The breadth of Cygnus covers the following areas.

1. Use in education CyGNUS would be useful in educational institutions for student-teacher interactions or maybe even higher staffs-employees relationships.It can be used for listening into announcements, discussions or informal conversations about a particular organisation or the services provided by that organization. CyGNUS is a useful tool for evaluating a course formatively. Because of its simple use and the electronic handling of data, the administrative effort remains small.
2. Use in emergencies It can be to exchange minute-to-minute information about local disasters including statistics and directions.

3. Use in campaigning It can be used for publicity in an college election or selection of an committee.

4. Use in business It can serve as a powerful tool for identifying the current trends in the market like the latest fashion trends and dress styles. Users can get beauty tips and latest trends by following relevant people or organizations. Public dedications to friends,relatives,lovers etc are possible.

Chapter 3

Design And Implementation

3.1 System Design

This gives the system view in terms of the software components.

The user interface of the android helps to access the application and thereby our account. Whenever we access our account providing a username and password, it binds with the Python Application Programming Interface and Searches the SQLite database for the given details. The application authorizes the user to its home page only if it is authenticated. SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. The application's API to make the database developed in SQLite interact with the actions in co-operated in the front end of the application. Programs talk to the application API over HTTP, the same protocol that your browser uses to visit and interact with web page. Django is the web framework base on python used for connecting the database with the android application.Json strings are used to connect between these two.The front end is programmed with the help of java.The whole application is coded as different classes in android (which can also be called as activities) and they communicate with the help of the keyword Intent provided in android. An Intent provides a facility for performing late runtime binding between the code in different applications. It can be used with startActivity to launch an Activity. Its most significant use is in the launching of activities, where it can be thought of as the glue between activities. It is basically a passive data structure holding an abstract description of an action to be performed.XML layouts should also be specified for communication between android and java code.Every application must have an AndroidManifest.xml file (with precisely that name) in its root directory. The manifest presents essential information about the application to the Android system, information the system must have before it can run any of the application's code.
Whenever the user tries to login by providing his username and password, server responds with a JSON array which contains Posts objects sorted by time.The Posts objects are message,time,username,location,folower,followee.These objects were added to an ArrayList, which contains objects of type Posts which serves as the items for the ListView.This response was parsed using Google GSON by building the Class Heirarchy. Conversion of JSON Objects to Plain Old Java Objects (POJO) is done using Google Gson library.
The various system features of the application described in Software Requirements Specification is described here using Data Flow Diagrams.

3.1.1 Login

In the login page, the user has 2 options:- sign in or sign up. Users who already have an account can access their account by signing in. Those who are new can create an account by providing profile information. Those details are stored in a database. When a user signs in, he is authorized to the home page only after the authentication process. The username and password provided by the user is verified as part of authentication.

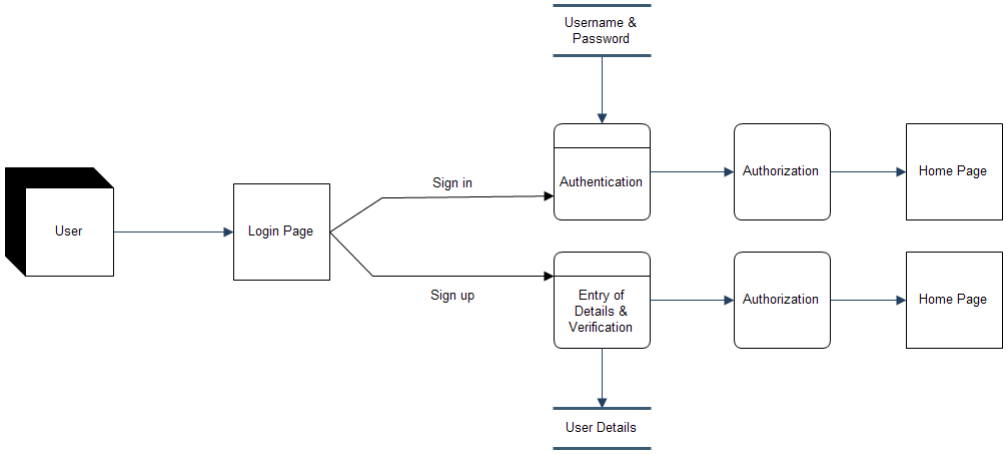

Figure 3.1: Login Page DFD

3.1.2 Home

A user can access his home page by signing in. The home page contains variety of options like posting messages, deleting messages, and replying to messages. When the user wants to post a message, he must write the message and click 'post'. Deleting and replying is done by selecting the messages and clicking the respective buttons. Whenever a message is posted, it is added to the timeline and replies are added as references. Likewise, when a message is deleted, it is deleted from the timeline.

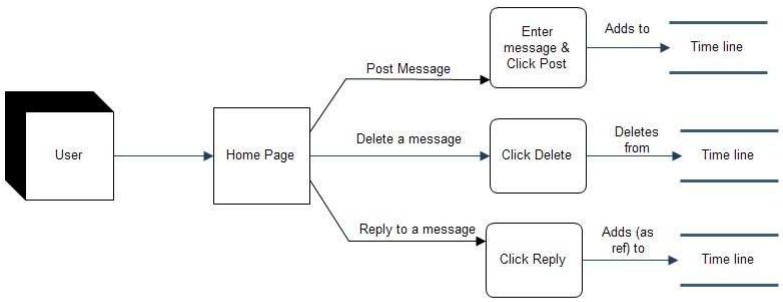

Figure 3.2: Home Page DFD

3.1.3 Search

The user once logged into the home page can search for users, posts, channels using the search page. Here the user has to provide a keyword which is searched in the database for a matching entity. Then the search results are displayed.

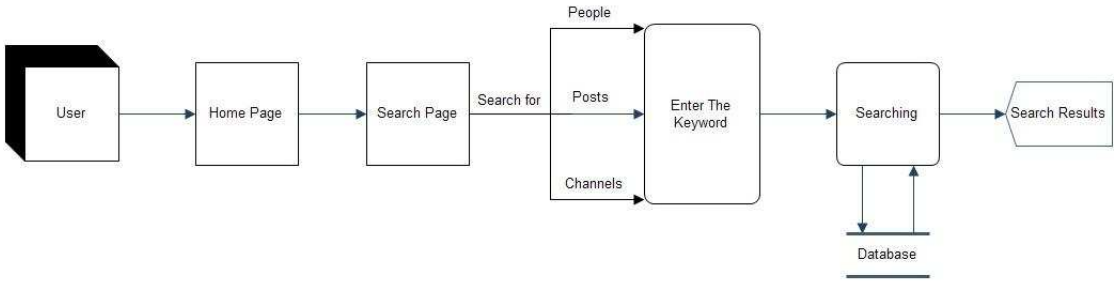

Figure 3.3: Search Page DFD

3.2 Database Design

We use the database provided by django in our application, specifically sqlite.Django is well suited for making database-driven Web sites, as it comes with easy yet powerful ways of performing database queries using Python.Once youve created your data models, Django automatically gives a database-abstraction API that lets you create, retrieve, update and delete objects.To represent database-table data in Python objects, Django uses an intuitive system: A model class represents a database table, and an instance of that class represents a particular record in the database table.To create an object, instantiate it using keyword arguments to the model class, then call save() to save it to the database.The user import the model class from wherever it lives on the Python path.Here is an ER diagram representing the database used in the application.
The tables used in our application are:

Post:- This table contains details of the post like time, text and the user that posted the message.

UserDetail:- This table contains details of the user like username, college, full_name, birth date etc. Django has inbuilt table for maintaining user's password, username and email which is mainly used for authentication purpose. So we use two tables the one we created and the one already provided by Django to maintain user information.

Followers:- This table maintains fields like user and the follower of that particular user which could be later manipulated for updatating , adding or deleting. There is a one-to-many relationship between the user and his followers. Table for the follower of a particular user and Followee is the inverse relationship of the Follower Table.

Like:- This is a table of posts marked as favourite by the user. It is maintained so that the user can later view his liked posts. It could have been implemented in table Post by keeping a new field but then the retrival of the favourite post could be slower.

Reply:- Contains the message for the user posted to by other users. This table could

also have been maintained in Post table by keeping aditional fields mentioning the messages as reply to a particular user

The Following represents the E-R Diagram showing the relationships between different entities used in the table and a view of the attributes for each entity.

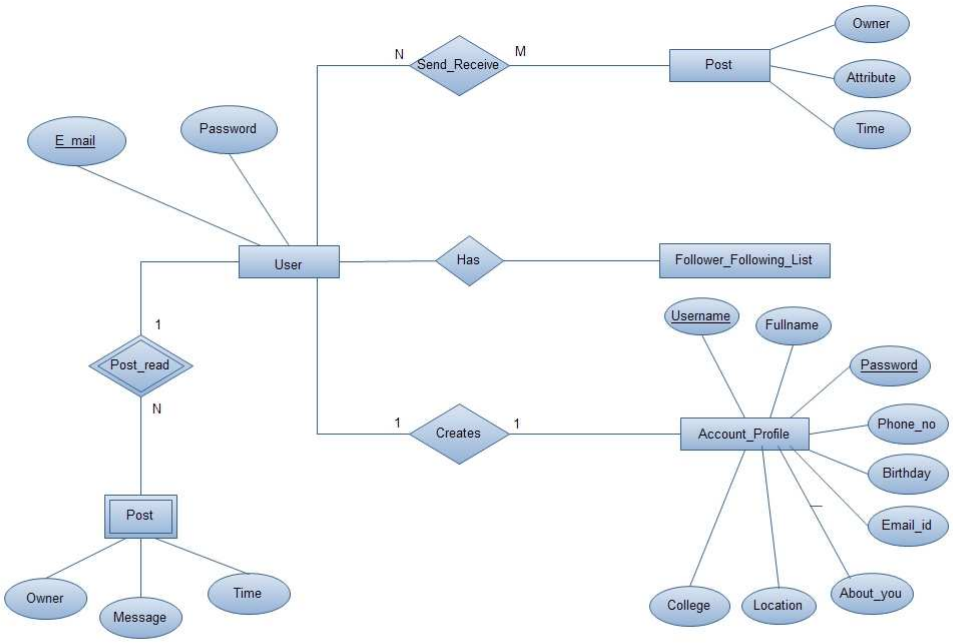

Figure 3.4: ER Diagram for Database

A user is an entity with attributes username and password with username as the key attribute.The user has a relation post_read with entity Post.Post has attributes message,owner of the post and time.Another relation 'creates' creates account profile by providing the attributes fullname, username, phone_no, birthday, location, about_you and college.The user has a follower_following list and can also send and receive posts along with the attributes owner,message and time.

3.3 GUI/User Interface Design

Android SDK is the software development kit that is used to create the interface for the application. It includes sample projects with source code, development tools, an emulator, and required libraries to build Android applications.The microblogging application CyGNUS developed in android , has a startup screen with 2 buttons 'Get started' and 'Learn more'.'Learn more' gives a short tutorial on how to use Cygnus. When the user press 'Get Started', the user get a Login page.

The user is provided with a login screen, having text boxes to enter the username and password. The Login Screen mainly has two text boxes, one for 'username' and other for 'password',based on which the user accesses his account. For novices, a button is provided naming 'Sign Up' which launches a new window/screen that prompts user to enter details like username for the account, email id, password, college and personal description in a text box . The user can read terms and conditions which declares the license for use of the application. If the condition for creating new account is satisfied then a new personalized account is created for the user. The home page for each user has a timeline of posts by his followees. There is a panel having options like Search used for searching users or posts, followers who will be able to see the posts of the current user, followees whose posts appear in the timeline of the current user, then 'favourites' which is a collection of post specified by the user and logout for redirecting the user to the login screen.
A user can post a new message containing less than or equal to 140 characters to his timeline. Each post, not belonging to user has three options reply, delete and like. On clicking on reply button a user can reply to the corresponding post. Like button can be used to mark the posts that appear in the user's timeline which can be later accessed through the option Favourites in the options menu. Delete button is provided for user owned posts which will delete the post of the user. There is a Profile screen for each user that displays the user information like a brief description of the user, number of followers, followees and recent posts and buttons for following the user if he/she isn't in the following list of the logged in user. A Search screen is displayed with a textbox to enter the keyword for search. There are three tabs: user, channel and post to accomplish search of keyword with any of this options with a search button. On clicking on the followers tab the user can view all the users that follow him/her and for followees tab the user can view the entire list of users that he is following .Favourites button will open up a screen that will list all the posts that were marked by the user as 'Like' in his timeline.

A peek into how each feature is provided in android.

3.3.1 Opening a screen

To specify a specific screen, call Intent.setClass or setClassName with the exact activity class to open. Otherwise, set a variety of values and data, and let Android decide which screen is appropriate to open. Android will find one or zero Activities that match the specified requirements; it will never open multiple activities for a single request.

When the user opens a new screen he can decide whether to make it transparent or floating, or full-screen. The choice of new screen affects the event sequence of events in the old screen (if the new screen obscures the old screen, a different series of events is called in the old screen).
Transparent or floating windows are implemented in three standard ways:
Create an app.Dialog class
Create an app.AlertDialog class
Set the Theme Dialog = theme attribute to @android:style/Theme.Dialog in your AndroidManifest.xml file.
For example:

```
<activity class="AddRssItem" android:label="Add an item" android:theme="@android:style
```

3.3.2 Layouts

Layout is the architecture for the user interface in an Activity. It defines the layout structure and holds all the elements that appear to the user. Each layout file must contain exactly one root element, which must be a View or ViewGroup object. Once the root

element is defined, additional layout objects or widgets can be added as child elements to gradually build a View hierarchy that defines your layout.

3.3.2.1 Linear Layout

LinearLayout is a ViewGroup that displays child View elements in a linear direction, either vertically or horizontally.If there is a nesting of multiple LinearLayouts, RelativeLayout can be used instead.There is a root LinearLayoutthat defines its orientation to be verticalall child Views(of which it has two) will be stacked vertically.It is a Layout that arranges its children in a single column or a single row. The direction of the row can be set by calling setOrientation(). Gravity can also be specified, which specifies the alignment of all the child elements by calling setGravity() or specify that specific children grow to fill up any remaining space in the layout by setting the weight member of LinearLayout.LayoutParams. The default orientation is horizontal.

3.3.2.2 Relative Layout

RelativeLayout is a ViewGroup that displays child View elements in relative positions. The position of a View can be specified as relative to sibling elements(such as to the left-of or below a given element) or in positions relative to the RelativeLayout area (such as aligned to the bottom, left of (center).A RelativeLayout is a very powerful utility for designing a user interface because it can eliminate nested ViewGroups.

3.3.3 Displaying a Progress Bar

An activity can display a progress bar to notify the user that something is happening. To display a progress bar in a screen, call

Activity.requestWindowsFeature(Window.FEATURE_PROGRESS).

To set the value of the progress bar, call

Activity.getWindow().setFeatureInt(Window.FEATURE_PROGRESS,level).

Progress bar values are from 0 to 9,999, or set the value to 10,000 to make the progress bar invisible. ProgressDialog class can also be used, which enables a dialog box with an embedded progress bar to send a "I'm working on it" notification to the user.

3.3.4 Creating Menus

Menus are an important part of any application. They provide familiar interfaces that reveal application functions and settings.

3.3.4.1 Options Menu

This is the primary set of menu items for an Activity. It is revealed by pressing the device MENU key. Within the Options Menu are two groups of menu items:

3.3.4.2 Icon Menu

This is the collection of items initially visible at the bottom of the screen at the press of the MENU key. It supports a maximum of six menu items. These are the only menu

items that support icons and the only menu items that do not support checkboxes or radio buttons.

3.3.4.3 Expanded Menu

This is a vertical list of items exposed by the "More" menu item from the Icon Menu. It exists only when the Icon Menu becomes over-loaded and is comprised of the sixth Option Menu item and the rest.
The Options Menu is opened by pressing the device MENU key. When opened, the Icon Menu is displayed, which holds the first six menu items. If more than six items are added to the Options Menu, then those that can't fit in the Icon Menu are revealed in the Expanded Menu, via the "More" menu item. The Expanded Menu is automatically added when there are more than six items.The Options Menu is where we include basic application functions and any necessary navigation items (e.g., to a home screen or application settings).When this menu is opened for the first time, the Android system will call the Activity onCreateOptionsMenu() callback method. Override this method in the Activity and populate the Menu object given to you. The menu can be populated by inflating a menu resource that was defined in XML, or by calling add() for each item you'd like in the menu. This method adds a MenuItem, and returns the newly created object to you. The returned MenuItem can be used to set additional properties like an icon, a keyboard shortcut, an intent, and other settings for the item.When a menu item is selected from the Options Menu, a callback will be received to the onOptionsItemSelected() method of your Activity. This callback passes you the MenuItem that has been selected. The item can be identified by requesting the itemId, with getItemId() which returns the integer that was assigned with the add() method. Once you identify the menu item, you can take the appropriate action.

3.3.4.4 Context Menu

This is a floating list of menu items that may appear when you perform a long-press on a View (such as a list item). The Android context menu is similar, in concept, to the menu revealed with a "right-click" on a PC. When a view is registered to a context menu, performing a "long-press" (press and hold for about two seconds) on the object will reveal a floating menu that provides functions relating to that item. Context menus can be registered to any View object, however, they are most often used for items in a ListView, which helpfully indicates the presence of the context menu by transforming the background color of the ListView item when pressed. (The items in the phone's contact list offer an example of this feature.) .To create a context menu, the Activity's context menu callback methods must be overrided: onCreateContextMenu() and onContextItemSelected(). Inside the onCreateContextMenu() callback method, menu items can be added using one of the add() methods, or by inflating a menu resource that was defined in XML. Then, register a ContextMenu for the View, with registerForContextMenu().

3.3.5 Displaying Alert

Android provides a number of ways to show pop-up notifications to the user as they interact with your application.

app.Dialog

A generic floating dialog box with a layout that you design.

app.AlertDialog

A popup alert dialog with two buttons (typically OK and Cancel) that take callback handler

ProgressDialog

A dialog box used to indicate progress of an operation with a known progress value or an indeterminate length (setProgress(bool)).

3.3.6 Views

This class represents the basic building block for user interface components. A View occupies a rectangular area on the screen and is responsible for drawing and event handling. View is the base class for widgets, which are used to create interactive UI components (buttons, text fields, etc.). The ViewGroup subclass is the base class for layouts, which are invisible containers that hold other Views (or other ViewGroups) and define their layout properties. All of the views in a window are arranged in a single tree. Views can be added either from code or by specifying a tree of views in one or more XML layout files.

3.3.6.1 TextView

Displays text to the user and optionally allows them to edit it.A TextView is a complete text editor, however the basic class is configured to not allow editing.

3.3.6.2 Adapter

Adapters provide a common interface to the data model behind a selection- style widget, such as a listbox. An Adapter object acts as a bridge between an AdapterView and the underlying data for that view. The Adapter provides access to the data items. The Adapter is also responsible for making a View for each item in the data set.An AdapterView is a view whose children are determined by an Adapter.It usually has interface definition for callback methods to be invoked when a button is clicked or selected.

3.3.6.3 ListView

ListView is a that creates a list of scrollable items. The list items are automatically inserted to the list using a ListAdapter.

3.3.6.4 ScrollView

ScrollView is a container that provides scrolling for its contents. You can take a layout that might be too big for some screens, wrap it in a ScrollView, and still use your existing layout logic. It just so happens that the user can see only part of your layout at one time; the rest is available via scrolling.

3.3.7 Handling Events

On Android, there's more than one way to intercept the events from a user's interaction with your application.The View class provides the means to do so.For instance, when a View (such as a Button) is touched, the onTouchEvent() method is called on that object.In order to intercept this, extend the class and override the method. However, extending every View object in order to handle such an event would not be practical. This is why the View class also contains a collection of nested interfaces with callbacks that you can much more easily define. These interfaces, called event listeners,helps in capturing the user interaction with your UI.
An event listener is an interface in the View class that contains a single callback method. These methods will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI.

Included in the event listener interfaces are the following callback methods:

onClick()

From View.OnClickListener.

This is called when the user either touches the item (when in touch mode), or focuses upon the item with the navigation-keys or trackball and presses the suitable "enter" key or presses down on the trackball.

onLongClick()

From View.OnClickListener.

This is called when the user either touches and holds the item (when in touch mode), or focuses upon the item with the navigation-keys or trackball and presses and holds the suitable "enter" key or presses and holds down on the trackball (for one second).

onTouch()

From View.OnTouchlistener.

This is called when the user performs an action qualified as a touch event, including a press, a release, or any movement gesture on the screen (within the bounds of the item).

onCreateContextMenu()

From View.OnCreateContextMenuListener.

This is called when a Context Menu is being built (as the result of a sustained "long click")

To define these methods and handle the events, we implemented the nested interface in the corresponding Activity or defined it as an anonymous class. Then, an instance of the implementation was passed to the respective View.setOnClickListener() method. (E.g., call setOnClickListener() and pass it your implementation of the OnClickListener..).

3.3.8 Buttons

Button represents a push-button widget. Push-buttons can be pressed, or clicked, by the user to perform an action.

3.3.8.1 ImageButton

Displays a button with an image (instead of text) that can be pressed or clicked by the user. By default, an ImageButton looks like a regular Button, with the standard button background that changes color during different button states. The image on the surface of the button is defined either by the android:src attribute in the ¡ImageButton¿ XML element or by the setImageResource(int) method. To indicate the different button states (focused, selected, etc.), you can define a different image for each state. E.g., a blue image by default, an orange one for when focused, and a yellow one for when pressed. An easy way to do this is with an XML drawable "selector."

3.3.8.2 CompoundButton

A button with two states, checked and unchecked. When the button is pressed or clicked, the state changes automatically.

3.3.8.3 CheckBox

A checkbox is a specific type of two-states button that can be either checked or unchecked. It is used in our application near 'Remember Me' in the login page for ease of signin in.

3.3.9 Animation

An Animation object can be attached to a view using setAnimation(Animation) or startAnimation(Animation). The animation can alter the scale, rotation, translation and alpha of a view over time. If the animation is attached to a view that has children, the animation will affect the entire subtree rooted by that node. When an animation is started, the framework will take care of redrawing the appropriate views until the animation completes.

3.3.10 Toast messages

A Toast is a transient message, meaning that it displays and disappears on its own without user interaction. Moreover, it does not take focus away from the currently active Activity . The Toast is mostly for advisory messages, such as indicating a long-running background task is completed, or authentication failed etc.

Following images are sketches of GUI which was included in design document.

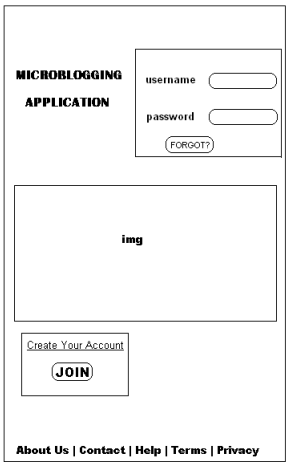

Figure 3.5: Login Sketch

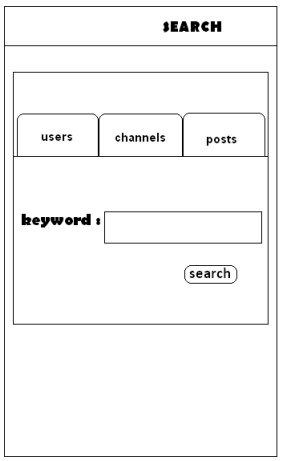

Figure 3.6: Search Sketch

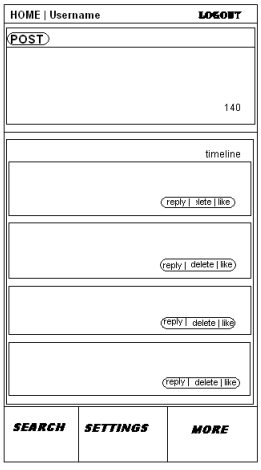

Figure 3.7: Timeline Sketch

Chapter 4

Coding

4.1 The front end

CyGNUS being a network application have a front end as well as a backend. As has been already pointed out the the front end Graphical User interface was coded in Java and XML. XML was used for laying out views and Java for rest of the purposes. [2]

4.1.1 Declaring the Layout

While designing the layouts for the GUI, we had the choice of doing it in Java or XML. The advantage to declaring UI in XML is that it enabled better and separate the presentation of the application from the code that controls its behavior. UI descriptions are external to the application code, which means that modification or adaptation of it was possible without having to modify source code and recompile.

4.1.2 Splash Screen

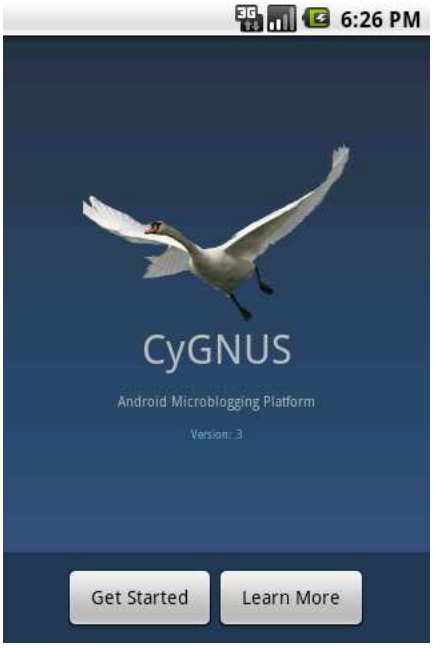

Figure 4.1: Splash Screen

The splash screen was developed using a ViewFlipper. As quoted from official documentation a View Flipper is simple ViewAnimator that will animate between two or more views that have been added to it. Only one child is shown at a time. If requested, it can automatically flip between each child at a regular interval. The XML code for View Flipper was as follows.

```
<ViewFlipper android:id="@+id/splash_more_flipper"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:outAnimation="@anim/push_left_out"
android:inAnimation="@anim/push_left_in">
```

This Viewflipper had as children, 6 TextViews which explained in layman terms how to use the application. The switching between child views was initiated by a button click as shown by this code snippet.

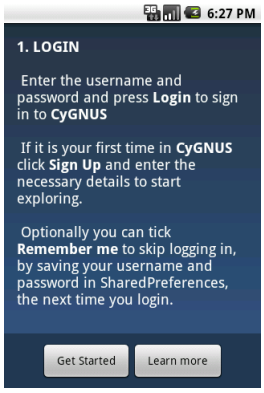

Figure 4.2: Splash Screen 1

Figure 4.3: Splash Screen 2

```
ViewFlipper mFlipper = ((ViewFlipper) this.findViewById(R.id.splash_more_flipper));

Button learn_more = (Button) findViewById(R.id.button_splash_learn_more);
learn_more.setOnClickListener(new OnClickListener() {
public void onClick(View v) {
mFlipper.showNext();
}
});
```

The following code ensured that the user had the option of skipping this page the next time he login if he had checked Remember me in the login screen allowing to store the username and password in the SharedPreferences.

```
@Override
protected void onResume() {
super.onResume();
mSP = getSharedPreferences("CurrentUser", MODE_PRIVATE);
String username = mSP.getString("username", null);
String password = mSP.getString("password", null);
if (username != null && username.length() > 0 && password != null && password.length()
```

```
mSkipPreferences = true;
}
if (mSkipPreferences) {
Log.d(TAG, Boolean.toString(mSkipPreferences));
Intent intent = new Intent(this, Timeline.class);
intent.putExtra("username", username);
intent.putExtra("password", password);
startActivity(intent);
finish();
}
```

4.1.3 Login

Figure 4.4: Login Screen

Figure 4.5: Loading

The Login Screen Layout was done in XML as well. Edittext views were used to allow the user to enter the username and password. The user could optionally check the Remember me checkbox so as to skip logging in the next time he uses CyGNUS. A progressDialog was used to show the progress of the logging in.

```
Progress Dialog progressDialog = new ProgressDialog(Cyg_Login.this);
        progressDialog.setMessage("Plz Wait Until Login Completes");
        progressDialog.setIndeterminate(true);
        progressDialog.setCancelable(true);
```

HttpPost request was used to send the username and password. The username and password were add as BasicNameValuePairs to a ListNameValuePair and set as URLEncodedFormEntity for the HTTP request. HttpClient Object was used to execute the Http Request and get the response from the server. The response from server was checked to ensure the user credentials entered were right. This whole procedure was done in a seperate thread than the UI Thread so as not to lock up the UI thread during the process.

```
HttpClient client = new DefaultHttpClient();
HttpPost httppost = new HttpPost(URL);
login_details.add(new BasicNameValuePair("username", un));
login_details.add(new BasicNameValuePair("password", pwd));
httppost.setEntity(new UrlEncodedFormEntity(login_details));
HttpResponse response = client.execute(httppost);
```

Upon pressing Signup button in login screen a new child activity was launched which contains Views for entering various details using an Intent object.

Figure 4.6: Sing Up

```
Intent intent = new Intent(getApplicationContext(), Sign_Up.class);
startActivityForResult(intent, REQ_CODE_1);
```

Pressing Terms and Conditions button in Sign Up Activity brings up an activity which contains a view showing the license for use of the application.

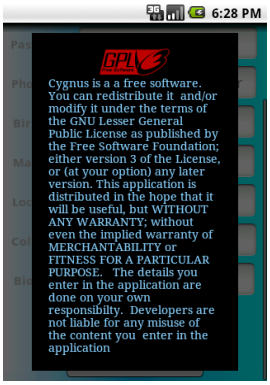

Figure 4.7: Terms and Conditions

When Sign Up activity is completed by calling finish(), the following method is invoked in the parent activity which redirected the user to the timeline activity after having username and password to the intent.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
/*
 * Gets invoked on finish() from Sign_Up.class
 */
super.onActivityResult(requestCode, resultCode, data);
switch(requestCode){
case REQ_CODE_1:
if(resultCode == RESULT_OK){
String u_signup, p_signup;
u_signup = data.getStringExtra("username");
p_signup = data.getStringExtra("password");
intent = new Intent(Cyg_Login.this, Timeline.class);
intent.putExtra("first_time", true); //Setting Boolean first_time to true
intent.putExtra("username", u_signup);
intent.putExtra("password", p_signup);
startActivity(intent);
}
}
```

The timeline for the user is displayed with a message from the bot 'Cygnet' welcoming the user to the application.

Figure 4.8: Welcome Message

4.1.4 Timeline

After authentication is complete, Timeline is launched as a seperate activity. The ListAdapter for Timeline, i.e MyClickableListAdapter, was implemented by extending the BaseAdapter. In the implementation we used holder object to access the list items efficiently. The listView for the timeline contains 2 textviews and a button displaying the message, time and username respectively. Additionally, click listeners are provided, which was be connected to the view items, message TextView and username Button. Their subclasses were implemented listeners was added to the clickable views.

CreateHolder function in the ClickableListAdapter will be called only as long, as the ListView is not filled entirely. That is, where performance gain is obtained : We use the relatively costly findViewById() methods and bind the view's reference to the holder objects. Additionally, we make some views in the list item clickable, by adding a click listener to the view required.

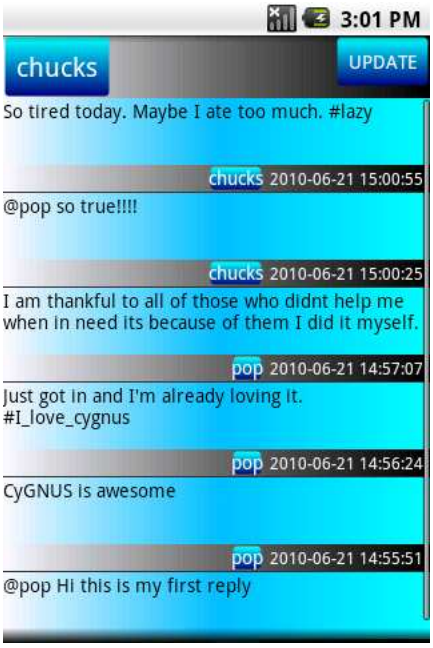

Figure 4.9: Timeline

ViewHolder provides fast access to arbitrary objects and views. This class is extended and adapted to the needs of timeline.

BindHolder function binds the holder and keeps our data up to date. In contrast to createHolder this method is called for all items. So, not a lot of heavy stuff is not done here. We simply transfer our object's data to the list item representatives.

If the user is logging in the application not for the first time, which is ensured by checking a boolean variable which is set to true if the user has just signed in and false otherwise, a request for getting the latest 10 posts for the user is sent to the specific URL. Server responds with a JSON array which contains 'Posts' objects sorted by time.

```
{ "posts":
  [
     { "post":
        {
       "username": "John",
       "message": "I'm back",
       "time": "2010-5-6 7:00:34"
            }},
        {    "post":
            {
                "username": "Smith",
                "message": "I've been waiting",
                "time": "2010-4-6 10:30:26"
            }}]}
```

This response was parsed using Google GSON by building the following Class Heirarchy.

```
public class PostList {
```

```
private List<PostContainer> posts = new ArrayList<PostContainer>();
public List<PostContainer> getPostContainterList() {
return posts;
}
}
class PostContainer{
Posts post;
public Posts getPost(){
return post;
}
}
public class Posts {
String message;
String time;
String username;
}
```

Conversion of JSON Objects to Plain Old Java Objects (POJO) was done using Google Gson library.

```
protected PostList getPostList (String jsonString){
PostList pl = null;
Gson gson = new GSON;
pl = gson.fromJson(jsonString, PostList.class);
return pl;
}
```

These objects were added to an ArrayList, which contains objects of type Posts which serves as underlying data for list view items.

The update button at the top of the view, when clicked adds any new posts to the timeline since the latest post in the timeline.

An Options Menu was implemented in the timeline activity(Refer Screenshot) to show various options such as Post, Search, More Posts, Favourites, Direct Messages and Logout.

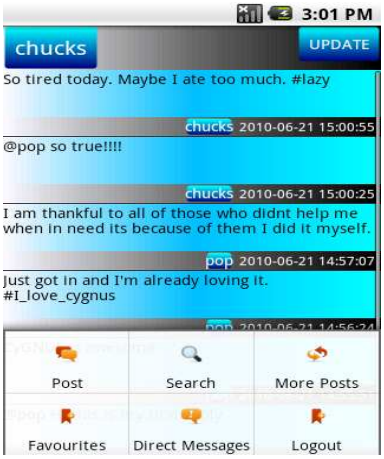

Figure 4.10: Options Menu

Figure 4.11: New Post

4.1.4.1 New Post

Upon clicking New Post in the Options menu a new child activity is launched which allows user to add a new Post to the timeline. The message typed out in the Edittext is sent to the server as BasicNameValuePairs along with username and the current time and at the same time passes the same details to parent Timeline Activity which adds the new post to the top of timeline.

4.1.4.2 Search

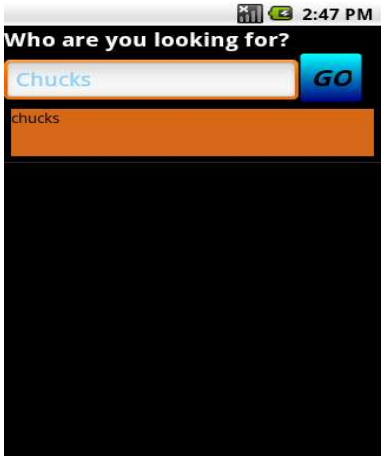

Figure 4.12: Search for People

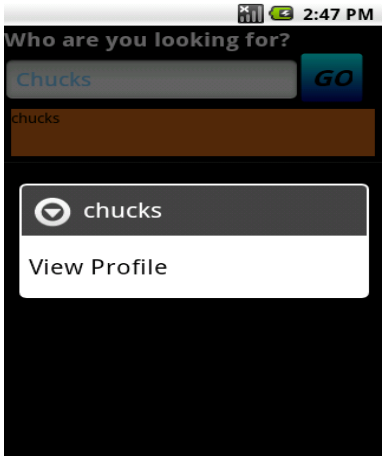

Figure 4.13: Context Menu

User can either search for Posts or People. In the case of Posts, user has enter a relevant keyword in the search box and press GO which retrieves a response from the server which contains the list of posts containing the relevant keyword. In case of searching for people, user has to specify the name of the user in the search box. The ListView for showing the search results was implemented by extending the ArrayAdapter. The following getView function presents the results after parsing the response from the server. In case of searching for people, a context menu was implemented on the search results so as to display an option to view profile of the specific user upon long click.

```
@Override
        public View getView(int position, View convertView, ViewGroup parent) {
                View v = convertView;
                if (v == null) {
                    LayoutInflater vi = (LayoutInflater)getSystemService(Context.LAYOU
                    v = vi.inflate(R.layout.search_list, null);
                }
```

```
                Posts pst = items.get(position);
                if (pst != null) {
                        tv = (TextView) v.findViewById(R.id.search_username);
                        if (tv != null)
                         if(ppl == true)
                              tv.setText(pst.getUsername());
                         else
                           tv.setText(pst.message);
                        /*if(iv != null)
                         iv.setImageBitmap(pst.getIcon()); */
                }
                return v;
        }
```

getView() gets a View that displays the data at the specified position in the data set. You can either create a View manually or inflate it from an XML layout file. Parameters

- position The position of the item within the adapter's data set of the item whose view we want.
- convertView The old view to reuse, if possible. Note: You should check that this view is non-null and of an appropriate type before using. If it is not possible to convert this view to display the correct data, this method can create a new view.
- parent The parent that this view will eventually be attached to
- Returns: A View corresponding to the data at the specified position.

4.1.4.3 More Posts

Retrieves a list of older posts. The time and username of the earliest post was send to the server as BasicNameValuePairs.

```
List<NameValuePair> nvps = new ArrayList<NameValuePair>();
nvps.add(new BasicNameValuePair("more_or_update", "more"));
nvps.add(new BasicNameValuePair("username", u_signup));
nvps.add(new BasicNameValuePair("post_user", mObjectList.get((mObjectList.size()-1)).u
nvps.add(new BasicNameValuePair("post_time", mObjectList.get(((mObjectList.size())-1))
```

4.1.4.4 Favourites

Retrieves list of posts liked by the user as a JSONList which is parsed using GSON and the results are presented in a ListView just like Search.

4.1.4.5 Direct Messages

Displays the replies to the user from other users. The procedure is same as for favourites.

4.1.4.6 Logout

Pressing logout signs out the user along with deleting the username and password, which may have been stored in SharedPreferences if the user had clicked Remember me in Login Screen and goes back to Login Screen so that another user can sign in.

Figure 4.14: List of User Posts

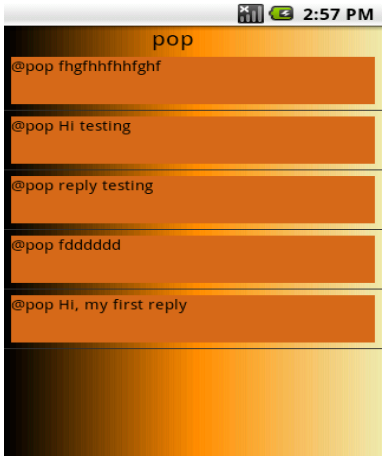

Figure 4.15: Direct Messages to user

```
SharedPreferences uPreferences = getSharedPreferences("CurrentUser", MODE_PRIVATE);
Editor editor = uPreferences.edit(); //Instantiating editor object
editor.putString("username", null);
editor.putString("password", null);
editor.commit();  //Commiting changes
Intent intent = new Intent(Timeline.this, Cyg_Login.class);
startActivity(intent);
finish();
```

4.1.5 Profile

Figure 4.16: Profile

Upon clicking the username under the message, Profile activity is launched which shows the various details of that specific user. The name of the user who is logged in and name of

the user whose profile is to be visited are sent to the server as BasicNameValuePairs. The server responds with JSON object which contains various details about the user. The JSON object is parsed in POJO using GSON. The follow attribute is checked in the response to ensure whether the current user is already following the user in the profile or not. There are 3 conditions that are to be handled.

```
private void check_follow(String follow_check) {

if(follow_check.equalsIgnoreCase("true")){  //Checking the current_user is following
follow.setText("Follow");
}
else if(follow_check.equalsIgnoreCase("false")){
follow.setText("Unfollow");
}
else {
follow.setVisibility(View.INVISIBLE);
follow.setClickable(false);
}
}
```

Rest of the details from the response are populated in the corresponding views in the Parent View.

4.1.5.1 Follow/ Unfollow

When 'follow' button is clicked the two usernames are sent as BasicNameValuePairs to the server along with action to be taken i.e follow or unfollow. The text displayed in that button is inversed. If 'follow' was displayed when the button was clicked, then after clicking the button it was changed to 'Unfollow.'

4.1.5.2 List of Posts/Followers/Followees

Upon clicking either of these 3 buttons, a new corresponding Activity was started and the Profile name and the status(i.e either posts or followers or followees) was sent to the specific url as BasicNameValuePairs. The server responded with the corresponding list will be displayed as in favourites.

4.1.5.3 Sms

This works just like New Post except that the message is sent as SMS to the phone number in the profile rather than adding it to the timeline. The following function sends the message which is typed out in the space displayed to the specific user.

```
protected void sendSMS(String phoneNo, String message) {
/*Function to Send the sms */
        PendingIntent pi = PendingIntent.getActivity(this, 0,
            new Intent(this, Profile.class), 0);
        SmsManager sms = SmsManager.getDefault();
        sms.sendTextMessage(phoneNo, null, message, pi, null);
}
```

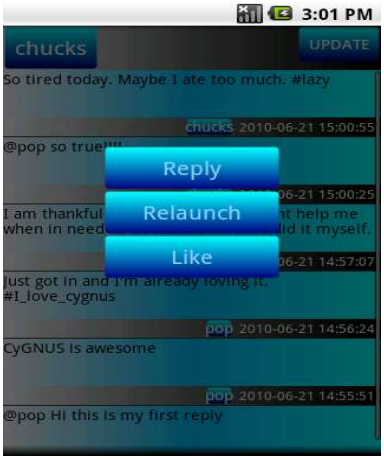

Figure 4.17: Long Click Menu for Reply/Retweet/Like

4.1.6 Reply/Retweet/Like

The options to reply, retweet and like were displayed upon long click on the message. It is a seperate activity with transparent background and 3 buttons.

Reply

User can reply to a post, except that belonging to user, which is in his timeline. The reply from the user is sent to the specific user as direct message. The message is added to timeline of user and sent to the server as BasicNameValuePairs with username in the post, name of the user who is replying to that post and the time at which the reply was sent.

Retweet

User can repost one of the posts in his timeline other than his own. Reposting the post adds the post to the timeline of the user under his name with a prefix added to the message showing the name of orginal author of message. The post is sent to the server along with message, the user who reposted it and time of repost. Like

User can favourite a post in his timeline upon which the message, along with username and time of the post to the user. The post is added to list of favourited posts by the user.

4.2 Backend

Django is a python [3] framework for Web Development. We create Django Applications, in modern Web applications, often involves interacting with a database. Behind the scenes, a database-driven Web site connects to a database server, retrieves some data out of it, and displays that data on the required interface and in our project it is Android.

4.2.1 Starting a Project

A project is a collection of settings for an instance of Django. It includes Database configuration, Django-specific options and application specefic settings. The following command is used to start a project

```
django-admin.py startproject mysite
```

The project directory contains four files:

- manage.py : It is a command-line utility which allow us to interact with Django project. We can also run the development server, validate models and create tables etc.

- urls.py : This file contains the urls created. It is like a table that contains the Urls for the Django powered site.
- settings.py : This file contains the detail about the database we are using, time-zone, installed apps, template loaders, serialization moudules etc. It gives necessary information for the development server to run
- __init__.py : It is a file required by python to treat mysite as a package,usually this file remains empty.

4.2.2 Database Configuration

Database configuration lives in the Django settings file, called settings.py by default. We need to edit the following database setting as per the requirements of the database we are using.

> DATABASE_ENGINE = ” DATABASE_NAME = ” DATABASE_USER = ” DATABASE_PASSWORD = ” DATABASE_HOST = ” DATABASE_PORT = ”

Since we are using SQLite we need to give the name 'sqlite' in DATABASE_ENGINE, then the path of the databse file we are using in DATABASE_NAME. Rest is to be kept blank for SQLite database.CREATE TABLE ”mba5_userdetail” (

> ”id” integer NOT NULL PRIMARY KEY, ”full_name” varchar(30) NOT NULL, ”username” varchar(30) NOT NULL UNIQUE, ”phone_no” integer, ”email” varchar(75) NOT NULL UNIQUE, ”location” varchar(60) NOT NULL, ”college” varchar(30) NOT NULL, ”bio_data” varchar(1024) NOT NULL, ”birthdate” date NOT NULL, ”image” varchar(100) NOT NULL) ;

4.2.3 Starting an App

A django app is a bundle of django code that contains models which is a description of data in the database and views which contains the function that are invoked when the particular web request is triggered. An Application in Django is started using the command:

```
python manage.py startapp mba5
```

It creates a application named mba5 in the project directory. It also contains four files

- __init__.py
- views.py
- test.py
- models.py

After the Application is created it is needed to be mentioned in the settings.py file in the INSTALLED_APPS tuple.

4.2.3.1 Defining models in Application

```
class UserDetail(models.Model):
     full\_name = models.CharField(max\_length=30)
     username = models.CharField(max\_length=30, unique=True)
     phone\_no = models.IntegerField(null=True, blank=True)
     email = models.EmailField(unique=True)
     location = models.CharField(max\_length=60, blank=True)
     college = models.CharField(max\_length=30, blank =True)
     bio_data = models.CharField(max\_length=1024, blank=True)
     birthdate = models.DateField(blank=True)
```

The above is the equivalent Python code for creating table in SQL:

```
CREATE TABLE "mba5\_userdetail" (
      "id" integer NOT NULL PRIMARY KEY,
      "full\_name" varchar(30) NOT NULL,
      "username" varchar(30) NOT NULL UNIQUE,
     "phone\_no" integer,
     "email" varchar(75) NOT NULL UNIQUE,
     "location" varchar(60) NOT NULL,
     "college" varchar(30) NOT NULL,
     "bio\_data" varchar(1024) NOT NULL,
     "birthdate" date NOT NULL,
)
;
```

Here a table UserDetail is created. It contains fields like username, full_name, phone_no, email, location, college, bio_data, birthdate and image. Each of the fields are given specific datatypes like CharField which would contain only characters with the constraints mentioned of maximum length to be of 30 characteres in username field. Similarly unique=True constraint will see that the data is not redundant for this field.

4.2.3.2 Defining Views for an Application

In the file views.py created we can write API to access the database, add ,delete update and retrieve data from the database. It also contains functions for user signing up, user authentication etc.

4.2.3.3 User creation

The following is the function for creating user:

```
def create\_user(request):
    if request.method == 'POST':
     username = request.POST['user\_name']
    password = request.POST['password']
email = request.POST['mail\_id']
if (username and password):
  if not (check\_user(username) and check\_mail(email)):
     user = authenticate(username=username, password=password)
     if user is None:
         user = User.objects.create\_user(username, email, password)
```

```
user.is\_staff = True
user.save()
full_name = request.POST['full\_name']
phone_no = request.POST['phone\_number']
location = request.POST['location']
college = request.POST['college']
bio\_data = request.POST['bio']
birthdate = request.POST['birth\_day']
p1 = UserDetail(full_name = full\_name,
username = username,
phone_no = phone_no,
email = email,
location = location,
college = college,
bio\_data = bio\_data,
birthdate = birthdate)
p1.save()
            return HttpResponse("200")
```

The view functions takes at least one parameter, i.e the called request by convention. This is an object which is an instance of the class django.http.HttpRequest that contains information about the current Web request that has triggered this view. Here the user related data is send through Http Post method, so we need to check that first and then get the username, password and email from the Http Post method. We check with these parameters if the user is previously authenticated or not, if the user is not authenticated then other details of the user like full_name, phone_no etc are obtained from the HttpPost and is added to the table and a response of string 200 returned to the client.

4.2.3.4 User Login

```
user = authenticate(username=username, password=password)
if user is not None:
     if user.is\_active:

return HttpResponse("200")
else:
        return HttpResponse("400")
else:
     return HttpResponse ("403")
```

We authenticate the user, if the username and password is correct and the user's account is not disabled then a response string 200 is sent to the client.

4.2.3.5 User Profiles

User details are fetched from the database, including the number of followers, followee and the number of the posts done by the user. User details are directly obtained from the UserDetail table like username, full_name college etc. The number of followers, followees and the number of posts by the current user is not directly obtained from the table. So a dictionary is created for these values and userdetails since dictionaries in Python are corresponding data structure for Json array which is responded to the client.

```
user\_dict = dict(username = c.username,
full\_name = c.full\_name,
college = c.college,
bio\_data = c.bio\_data,
birthdate = c.birthdate.isoformat(),
phone\_no = c.phone\_no,
user\_post_count = user\_post\_count,
user\_follower_count = user\_follower\_count,
user_followee_count = user\_followee\_count,
is\_following = is\_following)
json\_str = simplejson.dumps(user\_dict)
return HttpResponse(json\_str)
```

Json string is created using the python inbuilt function and is given as response to the Android client

4.2.3.6 User or Post search

```
search\_list = UserDetail.objects.filter(username\__contains = search\_param)
if search\_list == []:
return HttpResponse("403")
sorted\_search\_list = sorted(search_list, key = attrgetter('username'))
serial\_str = serializers.serialize('json',sorted\_search\_list, excludes=('birthdate'
json\_str = serial\_str.replace("[",'{"posts":[').replace("]",']}').replace("fields",
return HttpResponse(json\_str)
```

The search item is obtained using HttpPost method and is used to filter out the necessary list of search items. The list item is converted using Json string and passed to the client.

4.2.3.7 More or Update

We retrieve the post of the user first and convert the QuerySet obtained int a list. Similarly we convert the QuerySet of posts of the users those are followers of the current user into a list. The name of the Follower of the given user is also filtered out. With the names obtained in a list we obtain the posts of those users. Then we make a list comprising of the post of the user and the follower and the list is sorted according to the time of posting. Then the list comprising of the post at the time which is obtained as a parameter via HttpPost is made and also a list with the post of the a particular user also send via HttpPst by the client. Taking Intersection of these list gives the particular post from which the updation or more posts need to be retrived from the database

```
list2 = Post.objects.filter(time = time)
list3 = Post.objects.filter(user = post\_by)
list4 = filter(lambda x:x in list3,list2)
intersection\_list = filter(lambda x:x in post\_list,list4)
if (check\_list(intersection\_list)):
return HttpResponse("601")
n = post\_list.index(intersection\_list[0])

result = post_list[0:n]
result\_final = result[0:10]
```

After getting the the final result string, it is converted to JSON and sent as response to the client.

4.2.3.8 Populating Timeline initially

After making the list of posts consisting of the logged in user and the users whom the logged in user follows, is sorted on the basis of time and reversed to get proper view in Android client. The list is then send as a Json array.

```
#sorting the list with the attribute of time
post\_list = sorted(user\_post\_list, key = attrgetter('time'))
post\_list.reverse()
result\_post\_list = post\_list[0:10]
```

4.2.3.9 Post, Favourite, Direct message, Repost, Delete

The detail for the table entry for Post, Favourite, DirectMessage, Repost and Delete for doing the required manipulation of the database is acquired via HttpPost method. And the required operation is done.

Post: Details of the posts are added in the Post table.

Favourite: User liked posts are saved in table called LIKE. Similar is the case for Direct message and Repost.

Details regarding the post to be deleted is obtained and the entry of corresponding post is deleted. A user can also view his Favourite post and Direct messages and can delete his own posts.

4.2.3.10 Followers and Followees

Table for the follower of the particular user is kept and Followee is the inverse relationship of the Follower Table. The details for the table for follower is obtained and inserted into the table. Functions for viewing follower list and followee list responds in Json string.

4.2.4 Defining URLS

The URLs required by the client to send or receive data is defined in urls.py.

```
(r'$^$profile/\$' , views.profile),
(r'$^$signup/\$' , views.create\_user)
```

It tells which view function to use on accessing the defined url. For example on accessing the url /profile/ the function profile will be executed and similar is the case for /signup/. Basically it is a mapping between the Urls and view function that should be called for those Urls.

Chapter 5

Testing and Implementation

5.1 All the possible testing methods done for the project

5.1.1 Testing in small

For easiness of testing, the application was developed as seperate modules in the client side. The first one was Login module which included functionalities for logging in and signing up for a new user. The second module consisted of Timeline with features to view the 10 latest posts, add new post, reply to a post, search etc. These modules were rigourously tested before integrating them to form the entire application. Testing was carried by inputing various sequence of actions and checking that the response was the expected one.

5.1.1.1 Testing Database Configuration

:

The initial configuration of the database need to be done. Django needed to be informed about the database server intended to be used and how to connect to it. The initial configuration is done in settings.py file in the django project directory. If everything is done fine the bellow commands in shell shouldnot provide any error

```
>>> from django.db import connection
>>> cursor = connection.cursor()
```

If nothing happens the database is configured properly. Django App is a bundle of django code, contains the models and views. A Django model is a description of the data in your database, represented as Python code. Django uses a model to execute SQL code behind the scenes and return convenient Python data structures representing the rows in your database tables. For validating the models we can run the following command: python manage.py validate Committing the SQL to database is provided by Django using: python manage.py syncdb. So database tables for the models in the Django apps installed, will be created. For Basic data access, Django provides a higer-level Python API for working on those models "from python manage.py shell"

```
>>> form mba5.models import UserDetail
>>> p = UserDetail(usermane = 'abc',
email = 'abc@xyz.com',
location = 'xyz' ,)
>>> p.save
```

This causes the UserDeatil object to be insatinated, if any error occurs it wont be able to save it to the database table. We can view the data using the objects insatinated: p.username,

p.email etc as many fields are specified in the models and if the fields are initialized. We can also get every record of the given model example:

```
>>> UserDetail.objects.all()
```

The proper way of data entry in the table and accessing shows the propoer installation of the database. Before actually using the database in Djanog powered sites it should be throughly tested using manage.py shell, by giving various arguments to test its datafields and constraints for each fields in the table. After installing Django admin interface and regstring the models to the admin site, Django admin site can also be used for testing of the database entry. We can check for the consistency of data entered through the shell and the admin site.

5.1.1.2 Testing Creation of User

Testing was mainly done in two different interfaces: a Web interface for the intial testing of the API developed for adding user-deatails in the table. The second interface is the Android Framework over Java which requires a different form of response string than that of the web interface. In web interface, Contact Forms have to be made for Http methods like HttpPost, HttpGet etc. Html Page for entering the user details have been created for entering the specefic values to the database. Datas are entered in the fields as per the constraints specified in the models created.

5.1.1.3 Testing SignUp

All the fields that are mandatory are given to the view function via HttpPost method through html page. The tables pertaining to SignUp is checked for the proper entry of values to the fields. This can be verified using command-line interpreter using manage.py shell, or it can also be verified in admin site if the admin site app is installed and the models are registered in the admin site.

5.1.1.4 Testing follow(to follow, to be followed, to unfollow)

The values of that are required for the table entry of Follower is obtained by maintaining proper tags for HttpPost method. The use of proper tags allows the method to retrieve the values from HttpPost and the function then tries to match the fields of the table, their corresponding data-types and if everything is correct the value is added onto table. To unfollow also the same method is applied, but just the values are deleted from the table of Follower. Various types of parameter checking was done to verify the proper working of API and database consistency.

5.1.1.5 Testing search(user and post)

Many search parameters was send for searching Users and Posts. The search parameters were send from Html pages for initial testing. Different conditions like seach parameter not apperaring in database, null parameters etc were used to test the consistency and see that search never caused the database an alteration.

5.1.1.6 Testing post operations(post, delete, like, reply, repost)

Post operations were tested with lots of different test objects. Fields like "time", "posted by" etc were tested for consistency. Tested with parameters that caused to check the fields are mandatory and other such features. Favourite, Reply and Repost tables were also done

through same procedure with the API manipulating the corresponding table. Parameters for the view function for the deletion of the appropriate post were checked to be enough and correct.

5.1.1.7 Testing profile

The data that are needed to be shown in the profile of the user were determined and the parameter required for the retrival of the data were obtained from the Contact form made to test it. Fields that are required by the client side is known prior to the sending of the required information which is displayed on html page to check if the format of the string required by the client is appropriate.

5.1.1.8 Testing more or update and timeline

Parameter for timeline or more or update is fetched from the Contact Form and is used in the API developed to get the list of the post as per the operations in the functions. The ordering of the list of posts were determined for timeline and more or update operations through various test objects. Null parameter, parameter that causes conflict in retrival of data from database, parameter set that is inconsistent etc were used and the problems raised due to such ussage of parameters were handled properly.

5.1.2 Testing in Large

The two seperate modules were integrated to form the final application. Testing was done mainly using Android Monkey.

> The Monkey is a program that runs on your emulator or device and generates pseudo-random streams of user events such as clicks, touches, or gestures, as well as a number of system-level events. You can use the Monkey to stress-test applications that you are developing, in a random yet repeatable manner.

The Monkey was launched using a command line on development machine. Because the Monkey runs in the emulator/device environment, it must be launched from a shell in that environment. This was done by prefacing adb shell to each command.

Monkey does the following When the Monkey runs,

- It generates events and sends them to the system.
- If application crashes or receives any sort of unhandled exception, the Monkey will stop and report the error.
- If application generates an application not responding error, the Monkey will stop and report the error.

```
$ adb shell monkey -p nor.cross.cyg -v 500
```

The above command will launch CyGNUS application and send 500 pseudo-random events to it. Heres a section of output after executing the above command.

```
:Monkey: seed=0 count=500
:AllowPackage: nor.cross.cyg
:IncludeCategory: android.intent.category.LAUNCHER
:IncludeCategory: android.intent.category.MONKEY
// Event percentages:
```

```
//   0: 15.0%
//   1: 10.0%
//   2: 15.0%
//   3: 25.0%
//   4: 15.0%
//   5: 2.0%
//   6: 2.0%
//   7: 1.0%
//   8: 15.0%
:Switch: #Intent;action=android.intent.action.MAIN;category=android.intent.category.LA
    // Allowing start of Intent { act=android.intent.action.MAIN cat=[android.intent.c
:Sending Pointer ACTION_MOVE x=-4.0 y=2.0
:Sending Pointer ACTION_UP x=0.0 y=0.0
:Sending Pointer ACTION_DOWN x=207.0 y=282.0
:Sending Pointer ACTION_UP x=189.0 y=289.0
:Sending Pointer ACTION_DOWN x=95.0 y=259.0
:Sending Pointer ACTION_UP x=95.0 y=259.0
:Sending Pointer ACTION_DOWN x=295.0 y=223.0
:Sending Pointer ACTION_UP x=290.0 y=213.0
:Sending Pointer ACTION_MOVE x=-5.0 y=3.0
:Sending Pointer ACTION_MOVE x=0.0 y=-5.0
    // Rejecting start of Intent { act=android.intent.action.MAIN cat=[android.intent.
:Sending Pointer ACTION_DOWN x=74.0 y=201.0
:Sending Pointer ACTION_UP x=74.0 y=201.0
:Sending Pointer ACTION_MOVE x=3.0 y=-2.0
:Sending Pointer ACTION_UP x=0.0 y=0.0
    // Allowing start of Intent { cmp=nor.cross.cyg/.Cyg_Login } in package nor.cross.
    // activityResuming(nor.cross.cyg)
:Sending Pointer ACTION_MOVE x=-4.0 y=2.0
    // Sending event #100
```

5.1.3 Testing the use cases

The various use cases were tested rigourously for errors.

5.1.3.1 Post Message

A new post was added by clicking Post in options menu and it was checked whether the new post was added to the timeline of the user.

After relogging in, it was verified whether the specific post was in the list of posts recieved from the server.

Whether the new post was shown in the timeline of a follower of the user upon clicking Update button in the timeline of follower.

5.1.3.2 Search Post/User

A post containing a specific keyword was searched and the search results verified. Usernames with various length of characters was searched for and the list obtained was verified.

5.1.3.3 Delete Post

A post belonging to the user was deleted and it was checked whether the post was removed from the user's timeline as well as that of his followers.

5.1.3.4 Like Post

A post in the timeline of the user but not belonging to him, was liked by the user. It was verified whether the liked post was present in the list of posts obtained upon pressing favourites in Options menu.

5.1.3.5 Logout

Logged out by pressing log out in the options menu and it was checked whether the user was rightly redirected to the Login screen. Also it was verified whether upon next time the application is taken the user is taken to the Splash Screen rather than the Splash Screen direclty.

5.2 Advantages and Limitations

5.2.1 Advantages

1. Being a mobile application Cygnus can be used on the go provided there is internet connection.
2. As identified in scope of the project, microblogging can influence various fields of the society.
3. Small installation size of the file makes sure that Cygnus doesn't overload your phone.

5.2.2 Limitations

1. Replies wouldn't be added to the timeline of the user to whom the reply is addressed to and rather be displayed as direct messages.

5.3 Future Extensions if possible

Due to limitation of time some of the features we had in mind couldn't be implemented in CyGNUS.

1. Add user avatars to the timeline
   Add an option of enabling user avatars in the timeline. Currently the text-only interface can seem a bit daunting to the average user.
2. Show unread (new) posts between sessions
   New posts may have some "new" icon (instead of "star" to the right...) The definition of the "session" (as some grace period) is also useful in order not to loose the list of new tweets if user left the timeline window for the short time (e.g. to answer a call...)
3. Channels
   Allow an user to create a channel to group other users of his choice according to some common criterion. Users have the option of following the channel to read the posts from the users belonging to that channel.

4. Different themes
   Single theme for the application as is the case at present may seem too boring for an user who wishes to have a change in looks once in a while. This feature if enabled will allow users to switch different themes which includes unique text styling, backgrounds, animations etc.

5. Notification about new posts via sms.
   When a new post is added to the timeline, by anyone to whom the user has subscribed, a sms is sent to the user with the content of the message.

Chapter 6

Conclusion

This project was an attempt to implement a microblogging platform for Android phones. Microblogging has long replaced traditional forms of internet communication like email and most of the times required than conventional blogging. Cygnus as a microblogging application can find use in various spheres of life. Cygnus was developed and tested entirely on computer using the Android SDK and the Virtual device that comes along with it. It should run without troubles on a real Android real phone which supports Android 2.1, even though we couldn't test it under real conditions. As of now, Cygnus supports important features of typical microblogging applications/websites. In future we hope to add more gusto to the application by adding the features specified in future extensions.

Bibliography

[1] Jacob Kaplan-Moss Adrian Holovaty, *The definitive guide to django: Web development done right*, Apress, 2008.

[2] Google Dev, *Android api demos*, website, 2008, `http://developer.android.com/resources/samples/ApiDemos/index.html`.

[3] Python Dev, *The python tutorial*, website, 2008, `http://docs.python.org/tutorial/`.

[4] Mark Murphy, *Beginning android*, Apress, 2008.