Project Report on

JustTalk: An instant messaging and VoIP calling based android application

Project Report submitted in partial fulfilment of the requirement for the degree of

Bachelor of Technology.

in

Computer Science & Engineering

Under the Supervision of

Ms. Nishtha Ahuja

By

Aman Khurana (Roll no-111284)

To

Jaypee University of Information and Technology Waknaghat, Solan – 173234, Himachal Pradesh

Certificate

This is to certify that project report entitled "*JustTalk: an android application*", submitted by Aman Khurana in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science & Engineering to Jaypee University of Information Technology, Waknaghat, Solan has been carried out under my supervision.

This work has not been submitted partially or fully to any other University or Institute for the award of this or any other degree or diploma.

Date:

Supervisor's Name: Ms Nishtha Ahuja

Designation: Assistant Professor

Acknowledgement

I take this opportunity to express our sincere thanks and deep gratitude to all those people who extended their wholehearted co-operation and helped me in completing this project successfully. First of all, I would like to thank **Dr S.P Ghrera (Head of Department, CSE)** for creating opportunities .Special thanks to **Ms Nishtha Ahuja,** Project Mentor for all the help and guidance extended to me by her in every stage during our project development. Her inspiring suggestions and timely guidance enabled us to perceive the various aspects of the project in a new light. I am highly indebted and grateful for her strict supervision, constant encouragement, inspiration and guidance, which ensure the worthiness of our work.

Date:	**Name of student:**
30/04/2015	Aman Khurana

TABLE OF CONTENTS

Abstract

This application provides a platform for people to communicate and connect on a voluntary condition of anonymity. This application aims to conceive a community of volunteers, who interact with the users of the application via VOIP calls and instant messaging to empathize, sympathize or just connect on a human level. This application will act as a catalyst for people to share and talk about the happy and sad moments of their life with an anonymous volunteer who can be anyone. This would be useful as communication can help a great deal in every aspect of life. Talking to anonymous people can be useful in many situations and circumstances. The best part being of the application is that the volunteers since they are anonymous will be non-judgemental.

The user needs to authenticate using a username & password. When the user is authenticated it gains to the list of volunteers for either instant messaging or making an internet call .The list is populated using a parse backend. Every user has an option of volunteering simply by turning on a toggle which once done implies that the user is now a member of the volunteer community. Then it can set its availability status as a volunteer .After call completion the user is required to rate the volunteer. This would help us to maintain a consistent user experience as it would provide us with the overall rating of any particular volunteer.

List of Figures and Tables:

Acronyms

Eclipse IDE- Eclipse Integrated Development Environment

Android SDK- Android Software Development Kit

AVD – Android Virtual Device

JDK- Java Development Kit

PCs – Personal Computers

RAM- Random Access Memory

AOSP- Android Open Source Project

VOIP-Voice Over I P

GCM-Google Cloud Messaging

SMS- Short Messaging Service

ACID- Atomicity, Consistency, Integrity, Durability

CHAPTER 1

INTRODUCTION

With the boom of android devices in the market, any application built on android has the potential to reach the masses. Thus the inception of this project was done by consistent observation of our immediate and not so immediate environment. The observation revealed a not so obvious fact that many a times people do not talk about things in their lives. The happy things they keep with themselves, scared that they might not lose it and the sad things also they keep with the perception that nobody cares and sharing of their sad concerns may seem too daunting a task because of the fear of judgement by their own loved ones. The sad reality is that people want to share these things. These things result in lowering of the happiness quotient of our society and increase of the sadness index. The reason: people do not talk. If people start talking and sharing these things the society can change for good. Here is where our application JustTalk comes into picture.

As the name indicates you need to Just Talk about your sadness to get it out of your system or Just Talk to share your happiness to increase its magnitude. It helps to get rid of many evils like suicide, depression, stress, etc from our society and thus the Solution: Just Talk.

1.1 PURPOSE

The inception of this project was done by consistent observation of our immediate and not so immediate environment. The observation revealed a not so obvious fact that many a times people do not talk about things in their lives. The happy things they keep with themselves, scared that they might not lose it and the sad things also they keep with the perception that nobody cares and sharing of their sad concerns may seem too daunting a task because of the fear of judgement by their own loved ones. The sad reality is that people want to share these things. These things result in lowering of the happiness quotient of our society and increase of the sadness index. The reason: people do not talk. If people start talking and sharing these things the society can change for good. Here is where our application JustTalk comes into picture.

As the name indicates you need to Just Talk about your sadness to get it out of your system or Just Talk to share your happiness to increase its magnitude. It helps to get rid of many evils like suicide, depression, stress, etc from our society and thus the Solution: Just Talk

1.2 OVERVIEW

This application will act as a catalyst for people to share and talk about the happy and sad moments of their life with an anonymous volunteer who can be anyone. This would be useful as communication can help a great deal in every aspect of life. Talking to anonymous people can be useful in many situations and circumstances. The best part being of the application is that the volunteers since they are anonymous will be non-judgemental. The application will also support messaging with a volunteer.

When the application is launched for the first time the user will have to login using his phone number as the unique identification id and a verification code will be sent to his phone using which he will be required to verify the authenticity of his phone number. This will be a one-time procedure. When the user has been authenticated, he will have options to talk to or chat with a volunteer from list of available volunteers or to register himself as a volunteer. If the user chooses to talk then also there are two options available, one is a regular call and other of an internet call. The internet call uses VoIP. The chat functionality is provided using Google Cloud Messaging APIs. After the call is completed the user is asked to rate the quality of the call. This would help us to maintain a consistent user experience as it would provide us with the overall rating of any particular volunteer. If a user is registered as a volunteer, he would have the option of setting his availability status on or off.

1.3 SCOPE

- Universality of android devices makes it available to a wide section of the society.
- The simplicity and all time availability of this app makes it more worthy as the user carries his phone everywhere and he can easily use the various application functionalities.
- As this application caters to a very subtle and important aspect of the entire society, This application has the potential to transform , in a small or a big way, the very life of its users.

CHAPTER 2

SOFTWARE ENVIRONMENT

Android is an operating system based on the Linux kernel and designed primarily for touchscreen mobile devices such as smartphones and tablet computers. Initially developed by Android, Inc., which Google backed financially and later bought in 2005. Android's user interface is based on direct manipulation using touch inputs that loosely correspond to real-world actions, like swiping, tapping, pinching, and reverse pinching to manipulate on-screen objects.

Since Android devices are usually battery-powered, Android is designed to manage memory (RAM) to keep power consumption at a minimum, in contrast to desktop operating systems which generally assume they are connected to unlimited mains electricity. Android has an active community of developers and enthusiasts who use the Android Open Source Project (AOSP) source code to develop and distribute their own modified versions of the operating system. These community-developed releases often bring new features and updates to devices faster than through the official manufacturer/carrier channels, albeit without as extensive testing or quality assurance provide continued support for older devices that no longer receive official updates; or bring Android to devices that were officially released running other operating systems, such as the HP Touch Pad .

2.1 SECURITY AND PRIVACY

Android applications run in a sandbox, an isolated area of the system that does not have access to the rest of the system's resources, unless access permissions are explicitly granted by the user when the application is installed. Before installing an application, Play Store displays all required permissions: a game may need to enable vibration or save data to an SD card, for example, but should not need to read SMS messages or access the phonebook.

SOFTWARE STACK

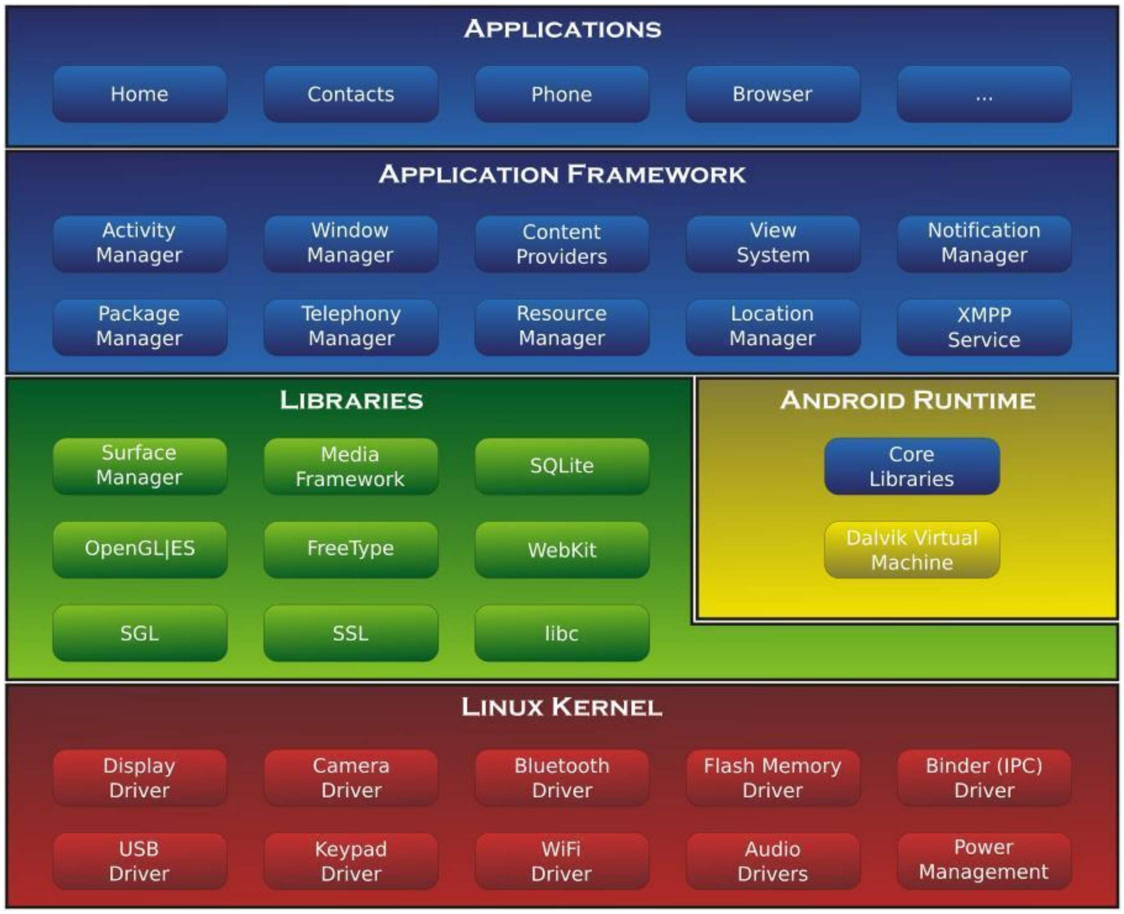


Figure 2.1 Software Stack

2.2 SOFTWARE SPECIFICATION

- **Android 2.2 (minimum)**

Android is an operating system based on the Linux kernel and designed primarily for touchscreen mobile devices such as smartphones and tablet computers. Android's user interface is based on direct manipulation using touch inputs that loosely correspond to real-world actions, like swiping, tapping, pinching, and reverse pinching to manipulate on-screen objects.

- **Android emulator 2.2 or higher**

The Android SDK includes a mobile device emulator — a virtual mobile device that runs on your computer. The emulator lets you develop and test Android applications without using a physical device.

- **Jdk 1.5 or higher**

The Java Development Kit (JDK) is an implementation of either one of the Java SE, Java EE or Java ME platforms released by Oracle Corporation in the form of a binary product aimed at Java developers on Solaris, Linux, Mac OS X or Windows. Since the introduction of the Java platform, it has been by far the most widely used Software Development Kit (SDK)

- **Eclipse IDE**

Eclipse is an integrated development environment (IDE). It contains a base workspace and an extensible plug-in system for customizing the environment. Written mostly in Java, Eclipse can be used to develop applications.

2.3 HARDWARE SPECIFICATION

- **Android smart phone**

Android phones are smart phones where we can deploy applications on each of these phones. Android phones are smart phones where we can deploy applications on each of these phones. Example of android phones are HTC Dream, Nexus One, Samsung Galaxy.

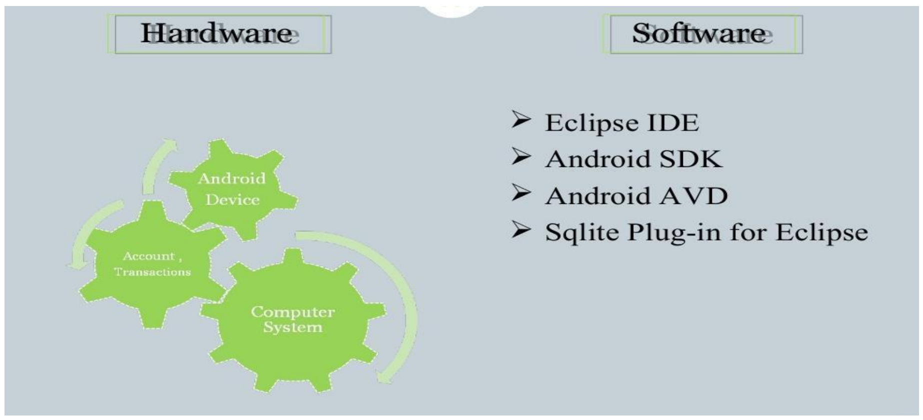


Figure 2.2 Hardware And Software Specification

2.4 SYSTEM DEVELOPMENT REQUIREMENTS

- **Android SDK**

Android software development is the process by which new applications are created for the Android operating system. Applications are usually developed in the Java programming language using the Android Software Development Kit, but other development tools are available.

- **Eclipse IDE**

  Eclipse is an integrated development environment (IDE). It contains a base workspace and an extensible plug-in system for customizing the environment. Written mostly in Java, Eclipse can be used to develop applications.

- **Android AVD**

  The AVD Manager provides a graphical user interface in which you can create and manage Android Virtual Devices (AVDs), which are required by the Android Emulator.

- **Android emulator 2.2 or higher**

  The Android SDK includes a mobile device emulator — a virtual mobile device that runs on your computer. The emulator lets you develop and test Android applications without using a physical device

- **Genymotion Android Emulator**

  Genymotion is an Android emulator for building and testing great Android apps. It's fast, simple and powerful. AOSP-based for perfect Android compliance, 20 pre-configured devices, CPU and OpenGL acceleration, etc. Genymotion is the next generation of the AndroVM open source project.

- **Third Party API's**

  - **Google cloud messaging API for android**
    Google Cloud Messaging (GCM) for Android is a service that allows you to send data from your server to your users' Android-powered device, and also to receive messages from devices on the same connection. The GCM service handles all aspects of queueing of messages and delivery to the target Android application running on the target device, and it is completely free.

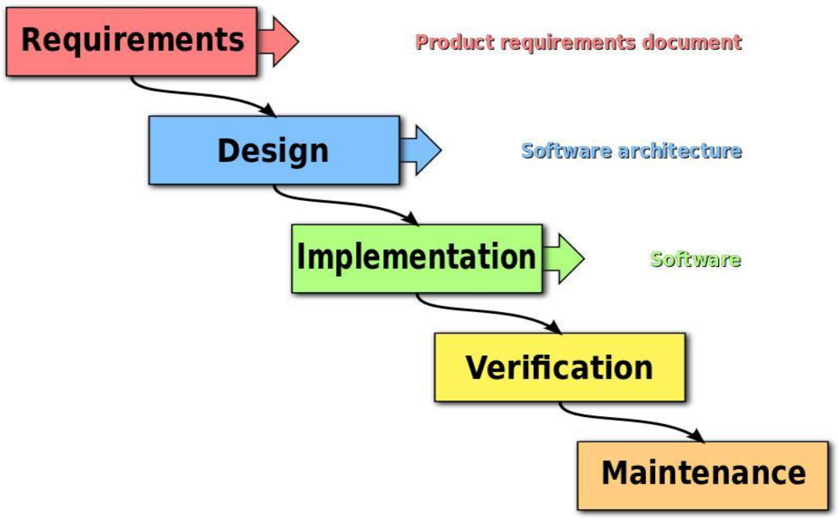
  - **Sinch**
  - **Parse**
- **Parse Database**

CHAPTER 3

METHODOLOGY

3.1 WATERFALL METHODOLOGY

The waterfall model is a sequential design process which is used in software development processes in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Conception, Initiation, Analysis, Design, Construction, Testing, Production/Implementation, and Maintenance. The waterfall model was first defined by Winston W. Royce in 1970 and has been widely used for software projects ever since. Our own implementation of the requirements/design phase is to produce a Functional Specification (detailing what the application will do) and a User Interface Specification (detailing how it will do it). When using this methodology it is vital that all requirements are captured during the Requirements/design phase as it can be very expensive to re-visit requirements once implementation (coding) has begun.



.

3.2 PROTOTYPE METHODOLOGY

The basic idea here is that instead of freezing the requirements before a design or coding can proceed, a throwaway prototype is built to understand the requirements. This prototype is developed based on the currently known requirements. By using this prototype, the client can get an “actual feel” of the system, since the interactions with prototype can enable the client to better understand the requirements of the desired system.

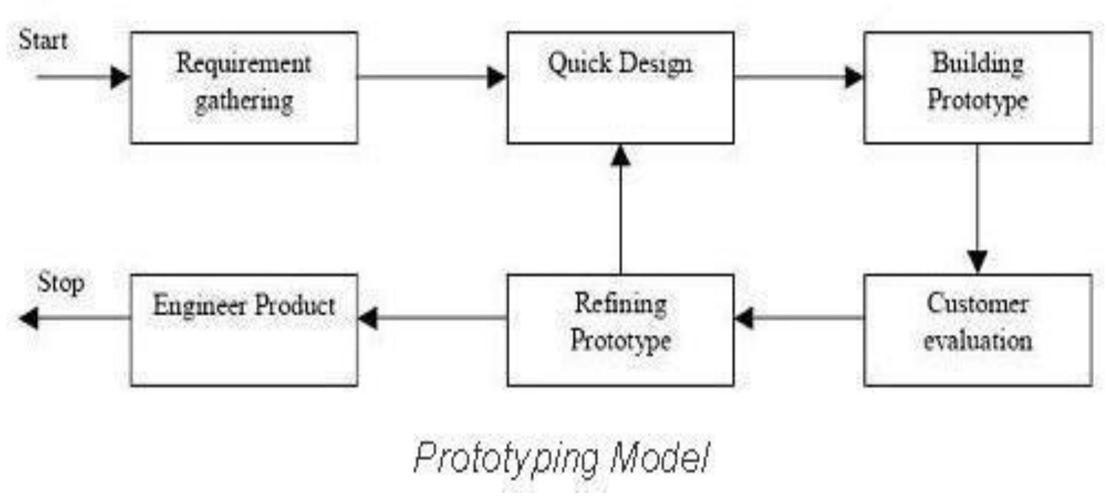


Figure 3.2 Prototype Model

CHAPTER 4

Project Design

4.1 Use Case Diagram

A use case diagram at its simplest is a representation of a user's interaction with the system and depicting the specifications of a use case. A use case diagram can portray the different types of users of a system and the various ways that they interact with the system. This type of diagram is typically used in conjunction with the textual use case and will often be accompanied by other types of diagrams as well.

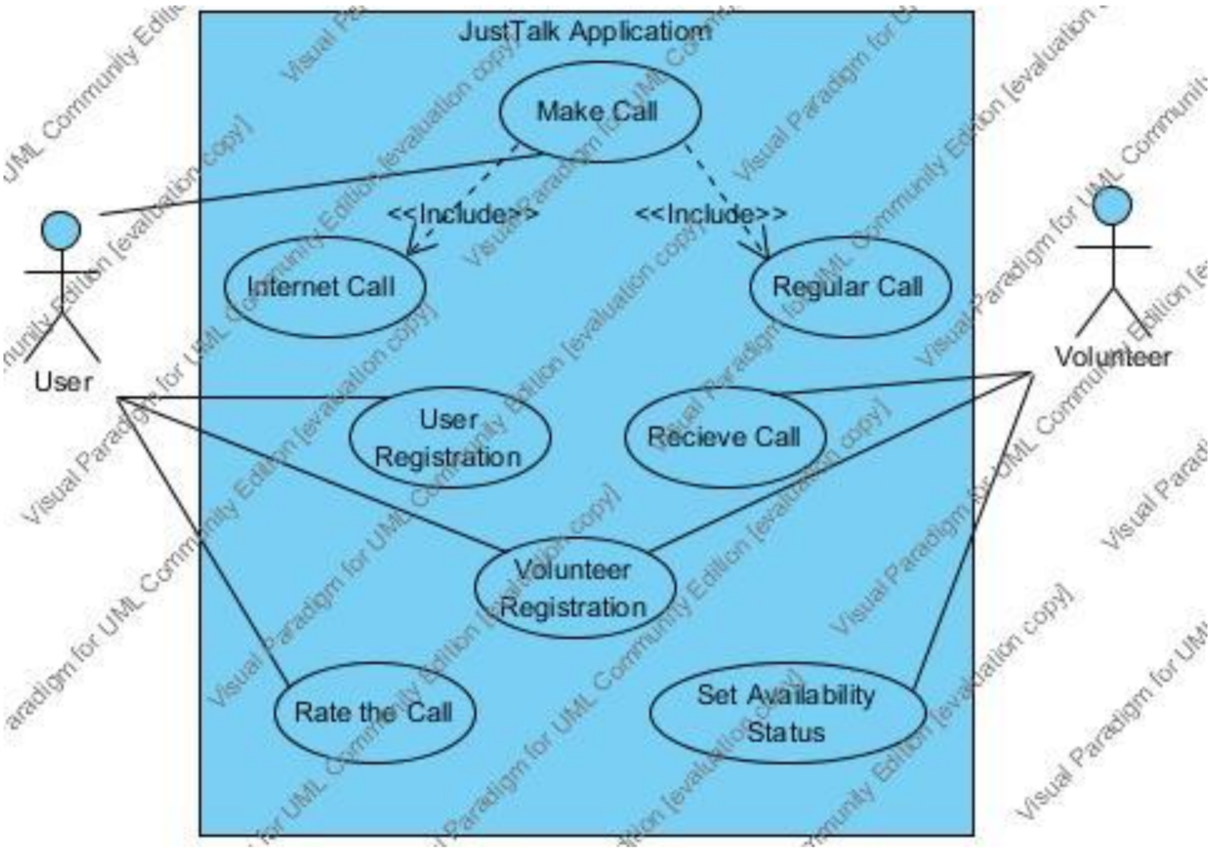


Figure 4.1 Use Case Diagram

4.2 Activity Diagram

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modelling Language, activity diagrams are intended to model both computational and organisational processes (i.e. workflows. Activity diagrams show the overall flow of control.

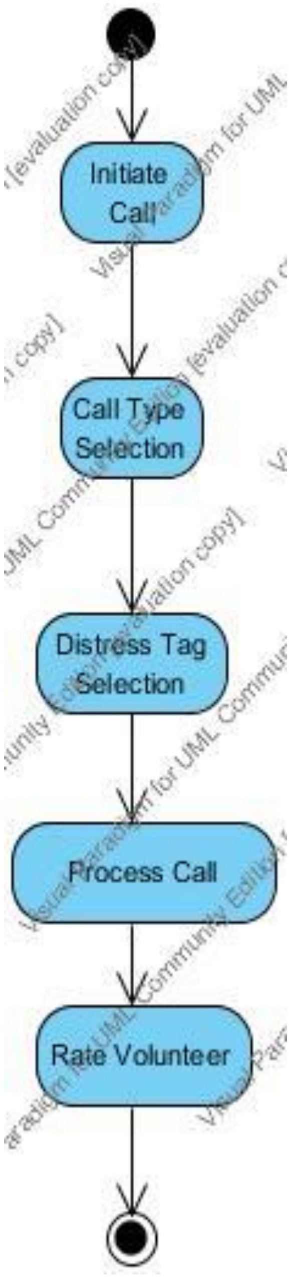


Figure 4.2 User Call Activity

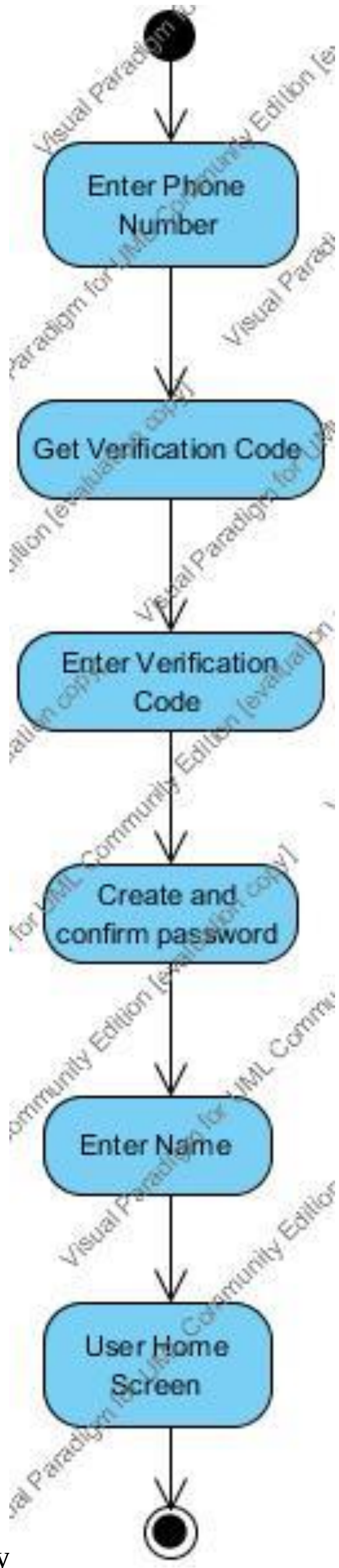


Fig 4.3 user login activity

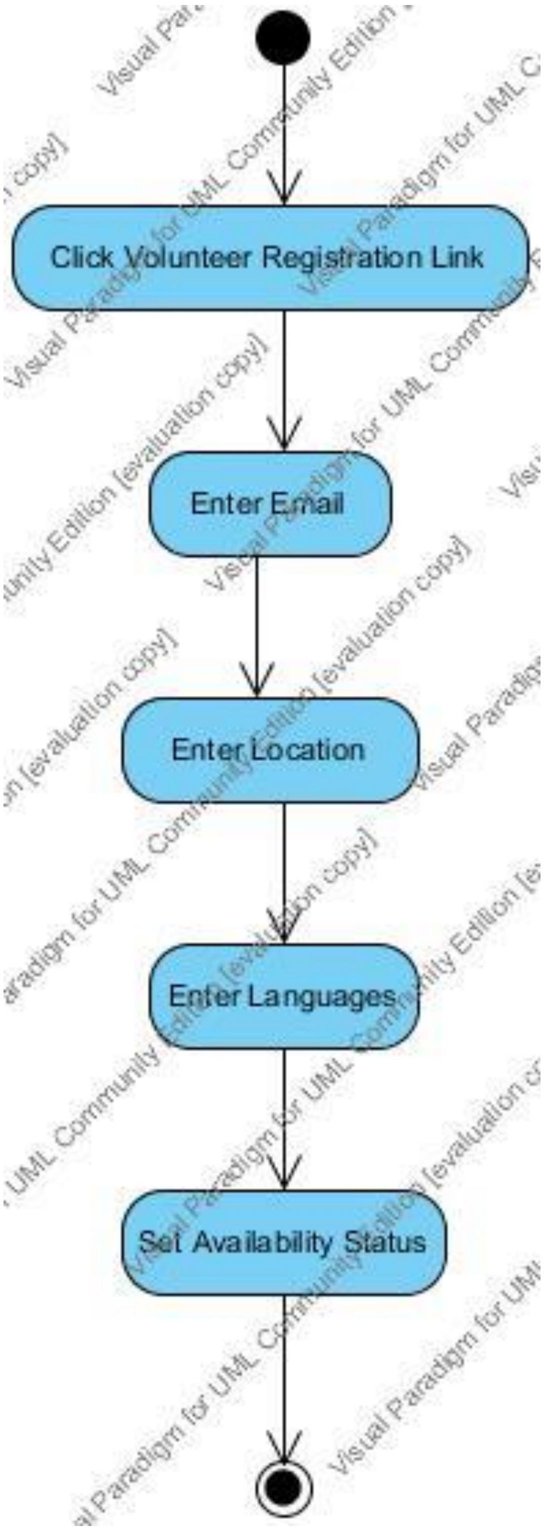


Figure 4.4 Volunteer Registration Activity

4.3 CLASS DIAGRAM

A class diagram in the Unified Modelling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations, and the relationships among objects.

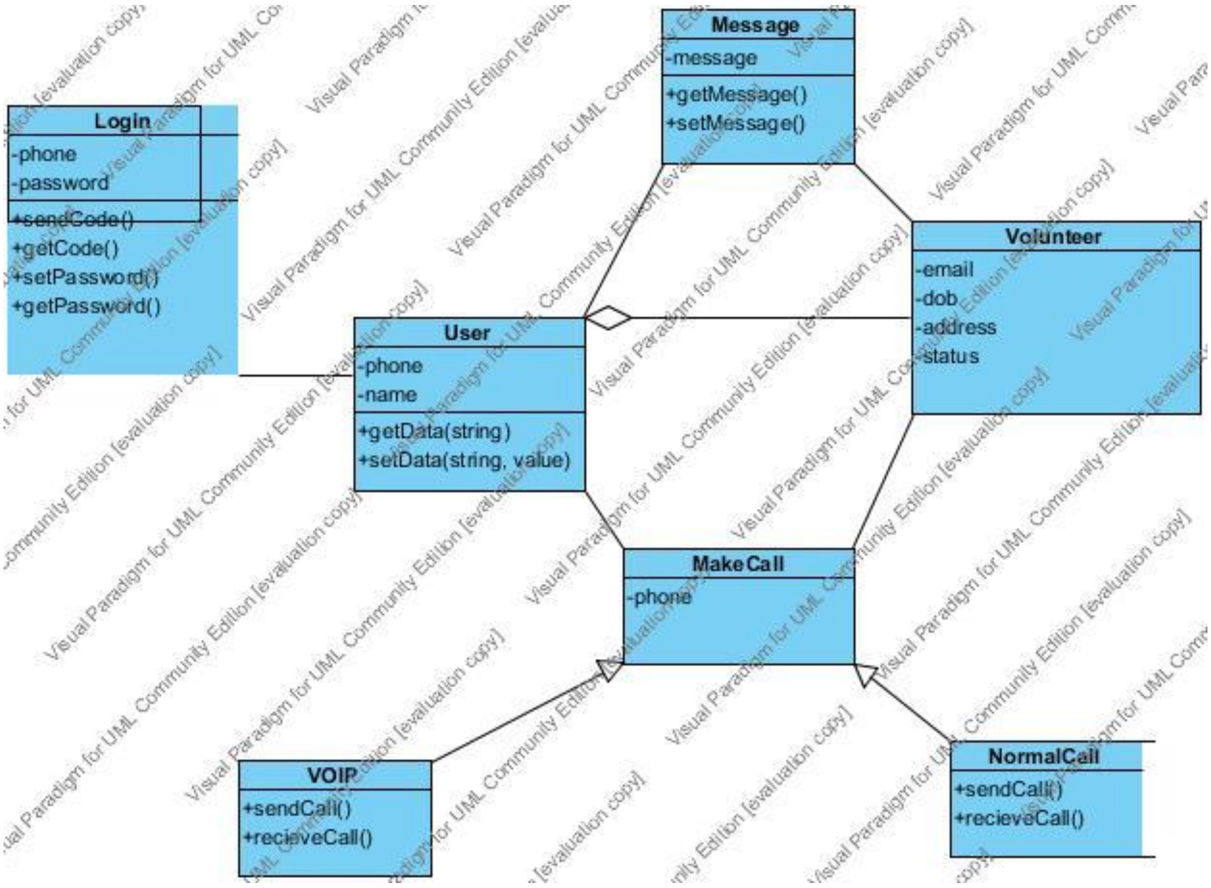


Figure 4.5 Class Diagram

CHAPTER 5

CODE DETAILS

5.1 Coding Standards

The standard used in the development of the system is JAVA Programming standards. it includes naming conversions of variables, constants and objects, standardized formats or labelling and commenting code, spacing, formatting and indenting.

5.2 Use Short Methods

To the extent that it is feasible, methods should be kept small and focused. It is, however, recognized that long methods are sometimes appropriate, so no hard limit is placed on method length. If a method exceeds 40 lines or so, think about whether it can be broken up without harming the structure of the program.

5.3 Definition of Fields in Standard Places

Fields should be defined either at the top of the file, or immediately before the methods that use them.

5.4 Limit Variable Scope

The scope of local variables should be kept to a minimum . By doing so, you increase the readability and maintainability of your code and reduce the likelihood of error. Each variable should be declared in the innermost block that encloses all uses of the variable**.**

5.5 Order Import Statements

The ordering of import statements is:

1. Android imports
2. java and javax

To exactly match the IDE settings, the imports should be:

- Alphabetical within each grouping, with capital letters before lower case letters (e.g. Z before a).
- There should be a blank line between each major grouping.

5.6 NAMING CONVENTIONS

Classes' names and interface names will start with capital letter. The function names will start with small letters and the first letter of each word in the function name will be in capital letter.

Follow Field Naming Conventions

- Non-public, non-static field names start with m.
- Static field names start with s.
- Other fields start with a lower case letter.
- Public static final fields (constants) are

ALL_CAPS_WITH_UNDERSCORES

5.7 LABELS AND COMMENTS

Sufficient labels and comments are included in the description of it for the benefits if the developer and other programmers who might examine it later.

CHAPTER 6

Technologies Used

6.1 Google Cloud Messaging API

Google Cloud Messaging (GCM) for Android is a free service that helps developers send data from servers to their Android applications on Android devices, and upstream messages from the user's device back to the cloud. This could be a lightweight message telling the Android application that there is new data to be fetched from the server (for instance, a "new email" notification informing the application that it is out of sync with the back end), or it could be a message containing up to 4kb of payload data (so apps like instant messaging can consume the message directly). The GCM service handles all aspects of queueing of messages and delivery to the target Android application running on the target device

Here are the primary characteristics of Google Cloud Messaging (GCM):

It allows 3rd-party application servers to send messages to their Android applications.

Using the GCM Cloud Connection Server, you can receive upstream messages from the user's device.

An Android application on an Android device doesn't need to be running to receive messages. The system will wake up the Android application via Intent broadcast when the message arrives, as long as the application is set up with the proper broadcast receiver and permissions.

It does not provide any built-in user interface or other handling for message data. GCM simply passes raw message data received straight to the Android application, which has full control of how to handle it. For example, the application might post a notification, display a custom user interface, or silently sync data.

It requires devices running Android 2.2 or higher that also have the Google Play Store application installed, or an emulator running Android 2.2 with Google APIs. However, you are not limited to deploying your Android applications through Google Play Store.

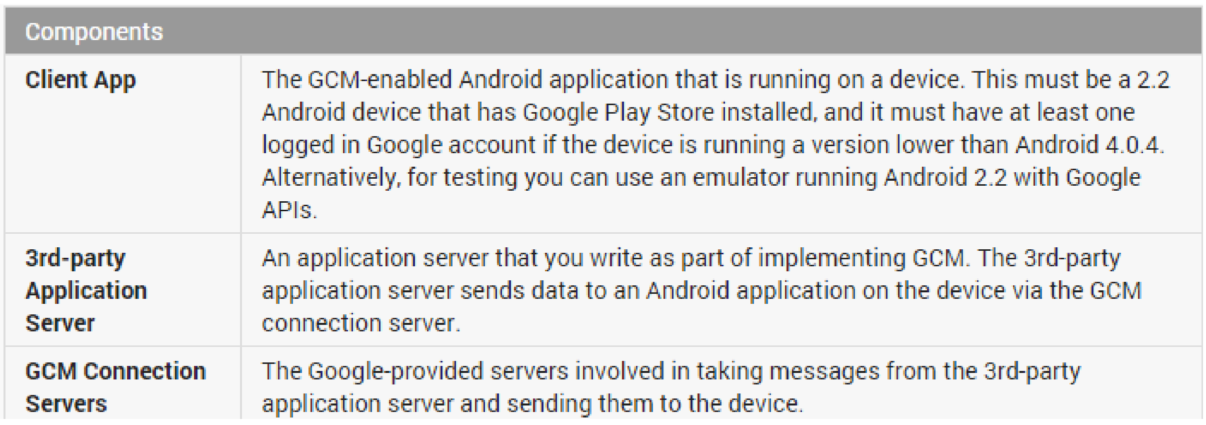
It uses an existing connection for Google services. For pre-3.0 devices, this requires users to set up their Google account on their mobile devices. A Google account is not a requirement on devices running Android 4.0.4 or higher.

6.1.1 Key Concepts

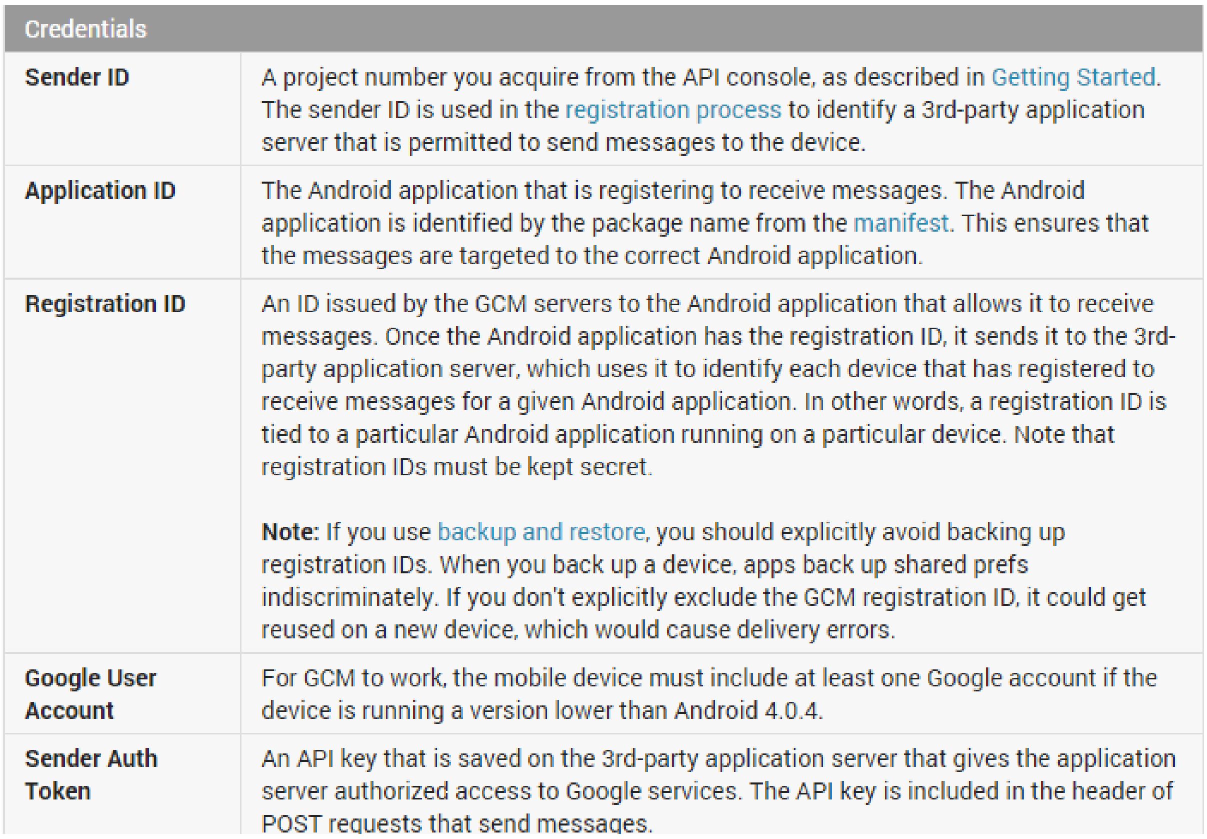
This table summarizes the key terms and concepts involved in GCM. It is divided into these categories:

•Components - The entities that play a primary role in GCM.

•Credentials - The IDs and tokens that are used in different stages of GCM to ensure that all parties have been authenticated, and that the message is going to the correct place.



Components	
Client App	The GCM-enabled Android application that is running on a device. This must be a 2.2 Android device that has Google Play Store installed, and it must have at least one logged in Google account if the device is running a version lower than Android 4.0.4. Alternatively, for testing you can use an emulator running Android 2.2 with Google APIs.
3rd-party Application Server	An application server that you write as part of implementing GCM. The 3rd-party application server sends data to an Android application on the device via the GCM connection server.
GCM Connection Servers	The Google-provided servers involved in taking messages from the 3rd-party application server and sending them to the device.



Credentials	
Sender ID	A project number you acquire from the API console, as described in Getting Started. The sender ID is used in the registration process to identify a 3rd-party application server that is permitted to send messages to the device.
Application ID	The Android application that is registering to receive messages. The Android application is identified by the package name from the manifest. This ensures that the messages are targeted to the correct Android application.
Registration ID	An ID issued by the GCM servers to the Android application that allows it to receive messages. Once the Android application has the registration ID, it sends it to the 3rd-party application server, which uses it to identify each device that has registered to receive messages for a given Android application. In other words, a registration ID is tied to a particular Android application running on a particular device. Note that registration IDs must be kept secret. **Note:** If you use backup and restore, you should explicitly avoid backing up registration IDs. When you back up a device, apps back up shared prefs indiscriminately. If you don't explicitly exclude the GCM registration ID, it could get reused on a new device, which would cause delivery errors.
Google User Account	For GCM to work, the mobile device must include at least one Google account if the device is running a version lower than Android 4.0.4.
Sender Auth Token	An API key that is saved on the 3rd-party application server that gives the application server authorized access to Google services. The API key is included in the header of POST requests that send messages.

Table 1: GCM components and credentials

6.1.2 Architecture Overview

A GCM implementation includes a Google-provided connection server, a 3rd-party app server that interacts with the connection server, and a GCM-enabled client app running on an Android device:

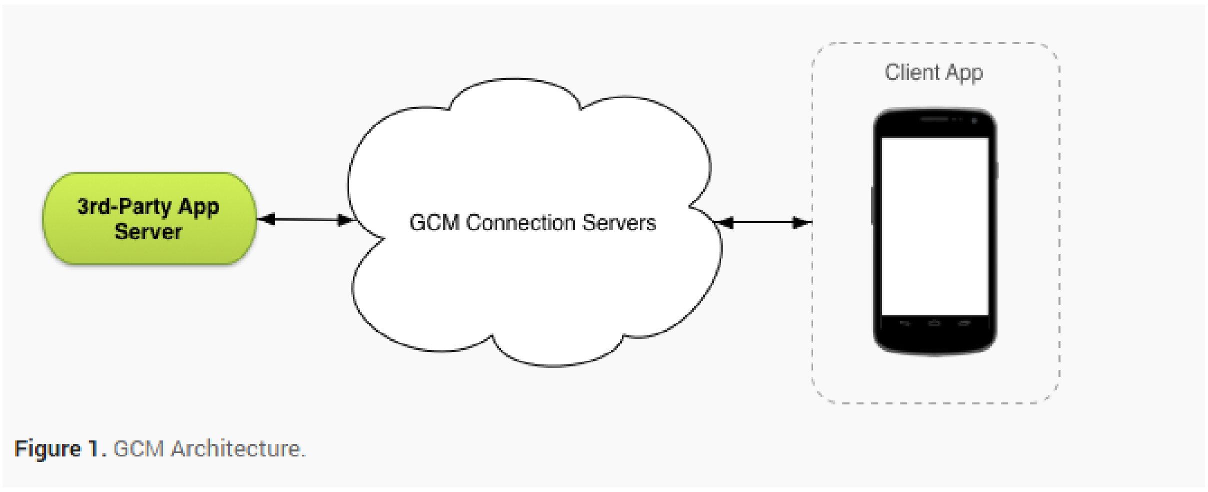


Figure 6.1 GCM Architecture

This is how these components interact:

Google-provided GCM Connection Servers take messages from a 3rd-party application server and send these messages to a GCM-enabled Android application (the "client app") running on a device. Currently Google provides connection servers for HTTP and XMPP.

The 3rd-Party Application Server is a component that you implement to work with your chosen GCM connection server(s). App servers send messages to a GCM connection server; the connection server enqueues and stores the message, and then sends it to the device when the device is online. For more information, see Implementing GCM Server.

The Client App is a GCM-enabled Android application running on a device. To receive GCM messages, this app must register with GCM and get a registration ID. If you are using the XMPP (CCS) connection server, the client app can send "upstream" messages back to the connection server. For more information on how to implement the client app, see Implementing GCM Client.

6.1.3 Lifecycle flow

1. Enable GCM. An Android application running on a mobile device registers to receive messages.

2. Send a message. A 3rd-party application server sends messages to the device.

3. Receive a message. An Android application receives a message from a GCM server.

Enable GCM

The first time the Android application needs to use the messaging service, it calls the GoogleCloudMessaging method register(), as discussed in Implementing GCM Client. The register() method returns a registration ID. The Android application should store this ID for later use (for instance, to check in onCreate() if it is already registered).

Send a message

Here is the sequence of events that occurs when the application server sends a message:

The application server sends a message to GCM servers.

Google enqueues and stores the message in case the device is offline.

When the device is online, Google sends the message to the device.

On the device, the system broadcasts the message to the specified Android application via Intent broadcast with proper permissions, so that only the targeted Android application gets the message. This wakes the Android application up. The Android application does not need to be running beforehand to receive the message.

The Android application processes the message. If the Android application is doing non-trivial processing, you may want to grab a PowerManager. WakeLock and do any processing in a service.

An Android application can unregister GCM if it no longer wants to receive messages.

Receive a message

This is the sequence of events that occurs when an Android application installed on a mobile device receives a message:

The system receives the incoming message and extracts the raw key/value pairs from the message payload, if any.

The system passes the key/value pairs to the targeted Android application in a com.google.android.c2dm.intent.RECEIVE Intent as a set of extras.

The Android application extracts the raw data from the com.google.android.c2dm.intent.RECEIVE Intent by key and processes the data.

6.1.4 Implementing GCM Client

A Google Cloud Messaging (GCM) client is a GCM-enabled app that runs on an Android device. To write your client code, we recommend that you use the GCM APIs. The client helper library that was offered in previous versions of GCM still works, but it has been superseded by the more efficient GCM APIs.

A full GCM implementation requires both a client implementation and a server implementation. For more information about implementing the server side, see Implementing GCM Server.

The following sections walk you through the steps involved in writing a GCM client-side application. Your client app can be arbitrarily complex, but at bare minimum, a GCM client app must include code to register (and thereby get a registration ID), and a broadcast receiver to receive messages sent by GCM.

Step 1: Set Up Google Play Services

To write your client application, use the GCM APIs. To use this API, you must set up your project to use the Google Play services SDK, as described in Setup Google Play Services SDK.

Caution: When you add the Play Services library to your project, be sure to add it with resources, as described in Setup Google Play Services SDK. The key point is that you must reference the library—simply adding a .jar file to your Eclipse project will not work. You must follow the directions for referencing a library, or your app won't be able to access the library's resources, and it won't run properly. If you're using Android Studio, this is the string to add to the dependency section of your application's build.gradle file:

```
dependencies {
  compile "com.google.android.gms:play-services:3.1.+"
}
```

Step 2: Edit Your Application's Manifest

Add the following to your application's manifest:

The com.google.android.c2dm.permission.RECEIVE permission so the Android application can register and receive messages.

The android.permission.INTERNET permission so the Android application can send the registration ID to the 3rd party server.

The android.permission.GET_ACCOUNTS permission as GCM requires a Google account (necessary only if if the device is running a version lower than Android 4.0.4)

The android.permission.WAKE_LOCK permission so the application can keep the processor from sleeping when a message is received. Optional—use only if the app wants to keep the device from sleeping.

An applicationPackage + ".permission.C2D_MESSAGE" permission to prevent other Android applications from registering and receiving the Android application's messages. The permission name must exactly match this pattern—otherwise the Android application will not receive the messages.

A receiver for com.google.android.c2dm.intent.RECEIVE, with the category set as applicationPackage. The receiver should require the com.google.android.c2dm.SEND permission, so that only the GCM Framework can send a message to it. If your app uses an IntentService (not required, but a common pattern), this receiver should be an instance of WakefulBroadcastReceiver. A WakefulBroadcastReceiver takes care of creating and managing a partial wake lock for your app.

A Service (typically an IntentService) to which the WakefulBroadcastReceiver passes off the work of handling the GCM message, while ensuring that the device does not go back to sleep in the process. Including an IntentService is optional—you could choose to process your messages in a regular BroadcastReceiver instead, but realistically, most apps will use a IntentService.

If the GCM feature is critical to the Android application's function, be sure to set android:minSdkVersion="8" or higher in the manifest. This ensures that the Android application cannot be installed in an environment in which it could not run properly.

Here are excerpts from a sample manifest that supports GCM:

```
<manifest package="com.example.gcm" ...>

    <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="17"/>
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.GET_ACCOUNTS" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission
android:name="com.google.android.c2dm.permission.RECEIVE" />

    <permission android:name="com.example.gcm.permission.C2D_MESSAGE"
        android:protectionLevel="signature" />
    <uses-permission android:name="com.example.gcm.permission.C2D_MESSAGE"
/>

    <application ...>
        <receiver
            android:name=".GcmBroadcastReceiver"
            android:permission="com.google.android.c2dm.permission.SEND" >
            <intent-filter>
                <action
android:name="com.google.android.c2dm.intent.RECEIVE" />
                <category android:name="com.example.gcm" />
            </intent-filter>
        </receiver>
        <service android:name=".GcmIntentService" />
    </application>

</manifest>
```

6.2 Sinch

The Sinch SDK is a product that makes adding voice calling and instant messaging to android mobile applications very convenient. It handles all the complexity of signalling and audio management while providing the freedom to create different features.

First Time Setup:

The Steps are as follows:

Register an application

1. Register a Sinch Developer account at http://www.sinch.com/signup.
2. Setup a new Application using the Dashboard where you can then obtain an Application Key and an Application Secret.

Add the Sinch client to Android Studio

1. Copy the entire libs folder to your project's root directory.
2. Right-click the jar-files and choose 'Add As Library'.
3. Create a new folder under src/main and name it jniLibs.
4. Move the armeabi and armeabi-v7a folders into the newly created jniLibs folder.

Permissions

A minimum set of permissions are needed for the app to use the Sinch SDK. These are specified in the `AndroidManifest.xml` file. If the calling functionality will be used, all five permissions listed here are needed. However, if the calling functionality isn't used, the last three (RECORD_AUDIO, MODIFY_AUDIO_SETTINGS and READ_PHONE_STATE) can be omitted.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
```

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

Note: By default, the Sinch SDK hangs up any Sinch call if the regular phone app has an active call. This functionality requires the permission READ_PHONE_STATE. However, if this default functionality isn't wanted, turn it off by calling `sinchClient.getCallClient().setRespectNativeCalls(false);` and the permission READ_PHONE_STATE is not needed.

Verify Manifest in runtime during development

To verify that the manifest has the necessary permissions the `sinchClient.checkManifest()` method can be used. This method should be called before starting the client and will throw an exception if the manifest isn't setup correctly. `sinchClient.checkManifest()` should only be called during development. When the application is ready for release the method call can safely be removed.

Note: This method takes into consideration which features the app supports (for example, calling, instant messaging, respecting native calls, and so on).
Call `sinchClient.checkManifest()` after the setup but before the start of the SinchClient.

Sinch Client:

The SinchClient is the Sinch SDK entry point. It is used to configure the user's and device's capabilities, as well as to provide access to feature classes such as the CallClient, MessageClient and AudioController.

Create a Sinch Client:

```
// Instantiate a SinchClient using the SinchClientBuilder.
android.content.Context context = this.getApplicationContext();
SinchClient sinchClient = Sinch.getSinchClientBuilder().context(context)
                                          .applicationKey("<application key>")
                                          .applicationSecret("<application secret>")
                                          .environmentHost("sandbox.sinch.com")
                                          .userId("<user id>")
                                          .build();
```

The Application Key and Application Secret are obtained from the Sinch Developer Dashboard. The User ID should uniquely identify the user on the particular device.

Note: All listener call-backs emitted from the Sinch SDK are invoked on the same thread that the call to `SinchClientBuilder.build` is made on. If the invoking thread is *not* the main-thread, it needs to have an associated `Looper`.

Specify Capabilities:

The following example shows how to setup the client with both voice calling and instant messaging enabled.

```
// Specify the client capabilities.
// At least one of the messaging or calling capabilities should be enabled.
sinchClient.setSupportMessaging(true);
sinchClient.setSupportCalling(true);
sinchClient.setSupportManagedPush(true);
// or
sinchClient.setSupportActiveConnectionInBackground(true);
sinchClient.startListeningOnActiveConnection()
```

Calling `startListeningOnActiveConnection` allows your application to receive incoming calls and messages without using push notifications.

Note: If the application is meant to only make outgoing calls but not receive incoming calls, don't call `startListeningOnActiveConnection` or `setSupportManagedPush`. Outgoing calls can be made after calling the start method.

Start the Sinch Client:

Before starting the client, add a client listener.

```
sinchClient.addSinchClientListener(new SinchClientListener() {

    public void onClientStarted(SinchClient client) { }

    public void onClientStopped(SinchClient client) { }

    public void onClientFailed(SinchClient client, SinchError error) { }

    public void onRegistrationCredentialsRequired(SinchClient client, ClientRegistration registrationCallback) { }

    public void onLogMessage(int level, String area, String message) { }
});

sinchClient.start();
```

Terminate the Sinch Client:

When the app is done using the SinchClient, it should be stopped. If the client is currently listening for incoming events, it needs to stop listening as well. After `terminate` is called, any object retrieved directly from the client object (that is, `CallClient`, `MessageClient`, and `AudioController`) is considered invalid.

Terminating the client:

```
sinchClient.stopListeningOnActiveConnection();
sinchClient.terminate();
```

Calling:

The Sinch SDK supports two types of calls: app-to-app calls and app-to-phone calls. The CallClient is the entry point for the calling functionality of the Sinch SDK.

Calls are placed through the `CallClient` and events are received using the `CallClientListener`. The call client is owned by the SinchClient and accessed using `sinchClient.getCallClient()`. Calling is not enabled by default.

Enable calling with the following method before starting the `SinchClient`:

```
sinchClient.setSupportCalling(true);
```

Set Up an App to App Call:

Use the CallClient to start the call (the `callUser` method). Pass the user identifier of the callee (the user receiving the call) to the call method, so that Sinch services can connect the call to the callee.

```
CallClient callClient = sinchClient.getCallClient();
Call call = callClient.callUser("<remote user id>");
call.addCallListener(...);
```

A call object is returned, containing details about the participants in the call, call details such as start time, call state, possible errors, and so on.

Assuming the callee's device is available, the method onCallProgressing is called on the CallListener. It notifies the application that the outgoing call is progressing. If a progress tone should be played, this is where it should be started. When the other party answers, the onCallEstablished method is called. Now, the users can start talking. If a progress tone was previously played, it should be stopped now.

Handle Incoming Calls:

To answer calls, the application must be notified when the user receives an incoming call. Add a CallClientListener to the CallClient to act on the incoming calls. The CallClientListener is notified using onIncomingCall() as calls come in to the device.

```
CallClient callClient = sinchClient.getCallClient();
callClient.addCallClientListener(...);
```

When the incoming call method is executed, the call can either be connected automatically without any user action, or it can wait for the user to press the answer or the hangup button. If the call is set up to wait for a user response, we recommended that a ringtone is played to notify the user that there is an incoming call.

```
@Override
public void onIncomingCall(CallClient callClient, Call call) {
    // Start playing ringing tone
    ...

    // Add call listener
    call.addCallListener(...);
}
```

To get events related to the call, add a call listener. The call object contains details about participants, start time, potential error codes, and error messages.

Answer Incoming Call:

To answer the call, use the answer method on the call to accept it. If a ringtone was previously played, it should be stopped now. User presses the answer button:

```
// User answers the call
call.answer();

// Stop playing ringing tone
...
```

Now, the clients on both ends establish the connection. When the call is established and the voice streams are running in both directions, the onCallEstablished listener method is called.

Decline Incoming Call:

If the call should not be answered, use the hangup method on the call to decline. The caller is notified that the incoming call was denied. If a ringtone was previously played, it should be stopped now. User presses the hangup button.

```
// User does not want to answer
call.hangup();

// Stop playing ringing tone
...
```

Disconnecting a call:

When the user wants to disconnect an ongoing call, use the hangup method. Either user taking part in a call can disconnect it. Hanging up a call:

```
call.hangup();
```

When either party disconnects a call, the application is notified using the call listener method onCallEnded. This allows the user interface to be updated, an alert tone to be played, or similar actions to occur. A call can be disconnected before it has been completely established. Hanging up a connecting call:

```
// Starting a call
Call call = callClient.callUser("<remote user id>");

// User changed his/her mind, let’s hangup
call.hangup();
```

Volume Control:

To make sure that the volume of the call can be modified by the hardware volume controls, setVolumeControlStream(AudioManager.STREAM_VOICE_CALL) must be called on the Activity where the call is handled. Make sure that volumeControlStream is reset to a suitable value when the call has ended.

For example, after creating a call (using CallClient.callUser) or when answering a call (using Call.answer()) you should call setVolumeControlStream(AudioManager.STREAM_VOICE_CALL);. When the call ends, set the volume control stream back to it's previous value. For example in your implementation of CallListener:

```
@Override
  public void onCallEnded(Call call) {
      setVolumeControlStream(AudioManager.USE_DEFAULT_STREAM_TYPE);
  }
```

Instant Messaging:

The MessageClient is the entry point to Instant Messaging functionality in the Sinch SDK. Messages are sent through the MessageClient and events are received using the MessageClientListener. The message client is owned by the SinchClient and accessed using SinchClient.getMessageClient(). Instant messaging is not enabled by default. To enable instant messaging, SinchClient.setSupportMessaging(true) must be set.

```
sinchClient.setSupportMessaging(true);
sinchClient.start();

...

MessageClient messageClient = sinchClient.getMessageClient();
messageClient.addMessageListener(...);
```

Send a Message:

Sending a message with the Sinch SDK is easy. Get hold of a MessageClient as described earlier and pass it a WritableMessage.

```
// Create a WritableMessage
WritableMessage message = new WritableMessage(
        "someRecipientUserId",
        "Hello someRecipientUserId! How are you?");

// Send it
messageClient.send(message);
```

Message Delivery Success

When a message to a recipient is successfully sent, there is an event on the MessageClientListener, onMessageSent.

```
@Override

public void onMessageSent(MessageClient client, Message message) {

    // Persist message

    // Update UI

}
```

Updating the UI from the onMessageSent callback is especially convenient when a user is logged in into more than one device simultaneously. The onMessageSent callback is fired on each device. This aids in keeping the UI consistent across devices. When the system has confirmed the messages were delivered the listener is notified using the onMessageDeliveredmethod. Inspecting the MessageDeliveryInfo parameter passed to the callback reveals more details on the specific event.

```
@Override
public void onMessageDelivered(MessageClient client, MessageDeliveryInfo
deliveryInfo) {
  Log.d(TAG, "The message with id "+deliveryInfo.getMessageId()
    +" was delivered to the recipient with id"+ deliveryInfo.getRecipientId());
}
```

Message Delivery Failure:

Delivering a message to a recipient can fail for various reasons: there might not be a network available, the recipient does not have instant messaging support and so on. When a message failed to reach its destination the listener is notified using the onMessageFailed callback. The reason for failing to deliver a message is propagated back as an MessageFailureInfo instance.

```
@Override
public void onMessageFailed(MessageClient client, Message message,
MessageFailureInfo failureInfo) {
        Log.d(TAG, "Failed to send to user: "+info.getRecipientId()
                    +" because: "+failureInfo.getSinchError().getMessage());
}
```

Messages are persisted internally in the SDK. In case the message was not sent successfully it will be retried automatically at a later point in time. The message will be retried for 12 hours and then fail permanently firing the failure callback.

Messages are stored in the backend for 30 days before being removed. If the recipient has not started the app and downloaded the message history within this time frame, the message will be lost and no notification will be received.

A message should be retried only in case of network unavailability (use messageFailureInfo.getSinchError().getErrorType().equals(ErrorType.NETWORK)). In this case create a new WritableMessage (using new WritableMessage(message)) and send that instance because the previous message is considered stale.

Receive a Message:

Incoming messages (Message) are delivered using the method onIncomingMessage on the MessageClientListener.

```
@Override
public void onIncomingMessage(MessageClient client, Message message) {
    // Persist message
    // Update UI
}
```

6.3 Parse

Introduction:

The Parse platform provides a complete backend solution for your mobile application. The Data services of this cloud based platform are used for the purpose of database of users and the messages.

On Parse, you create an App for each of your mobile applications. Each App has its own application id and client key that you apply to your SDK install. Your account on Parse can accommodate multiple Apps. This is useful even if you have one application, since you can deploy different versions for test and production.

The ParseObject

Storing data on Parse is built around the ParseObject. Each ParseObject contains key-value pairs of JSON-compatible data. This data is schemaless, which means that you don't need to specify ahead of time what keys exist on each ParseObject. You simply set whatever key-value pairs you want, and our backend will store it. For example, let's say you're tracking high scores for a game. A single ParseObject could contain:

```
score: 1337, playerName: "Sean Plott", cheatMode: false
```

Keys must be alphanumeric strings. Values can be strings, numbers, booleans, or even arrays and objects - anything that can be JSON-encoded. Each ParseObject has a class name that you can use to distinguish different sorts of data. For example, we could call the high score object a GameScore. We recommend that you NameYourClassesLikeThis and nameYourKeysLikeThis, just to keep your code looking pretty.

Saving Objects:

Let's say you want to save the GameScore described above to the Parse Cloud. The interface is similar to a Map, plus the saveInBackground method:

```
ParseObject gameScore = new ParseObject("GameScore");
gameScore.put("score", 1337);
gameScore.put("playerName", "Sean Plott");
gameScore.put("cheatMode", false);
gameScore.saveInBackground();
```

After this code runs, you will probably be wondering if anything really happened. To make sure the data was saved, you can look at the Data Browser in your app on Parse. You should see something like this:

```
objectId: "xWMyZ4YEGZ", score: 1337, playerName: "Sean Plott", cheatMode: false,

createdAt:"2011-06-10T18:33:42Z", updatedAt:"2011-06-10T18:33:42Z"
```

There are two things to note here. You didn't have to configure or set up a new Class called GameScore before running this code. Your Parse app lazily creates this Class for you when it first encounters it. There are also a few fields you don't need to specify that are provided as a convenience. objectId is a unique identifier for each saved object. createdAt and updatedAt represent the time that each object was created and last modified in the cloud. Each of these fields is filled in by Parse, so they don't exist on a ParseObject until a save operation has completed.

Retrieving Objects:

Saving data to the cloud is fun, but it's even more fun to get that data out again. If you have the objectId, you can retrieve the whole ParseObject using a ParseQuery:

```
ParseQuery<ParseObject> query = ParseQuery.getQuery("GameScore");

query.getInBackground("xWMyZ4YEGZ", new GetCallback<ParseObject>() {

  public void done(ParseObject object, ParseException e) {

    if (e == null) {

      // object will be your game score
    } else {
      // something went wrong
    }
  }
});
```

To get the values out of the ParseObject, there's a getX method for each data type:

```
int score = gameScore.getInt("score");
String playerName = gameScore.getString("playerName");
boolean cheatMode = gameScore.getBoolean("cheatMode");
```

If you don't know what type of data you're getting out, you can call get(key), but then you probably have to cast it right away anyways. In most situations you should use the typed accessors like getString. The three special values have their own accessors:

```
String objectId = gameScore.getObjectId();
Date updatedAt = gameScore.getUpdatedAt();
Date createdAt = gameScore.getCreatedAt();
```

If you need to refresh an object you already have with the latest data that is in the cloud, you can call the fetchInBackground method like so:

```
myObject.fetchInBackground(new GetCallback<ParseObject>() {
  public void done(ParseObject object, ParseException e) {
    if (e == null) {
      // Success!
    } else {
      // Failure!
    }
  }
});
```

The code in the GetCallback will be run on the main thread.

6.4 VoIP

Voice over IP (VoIP) is a methodology and group of technologies for the delivery of voice communications and multimedia sessions over Internet Protocol (IP) networks, such as the Internet. Other terms commonly associated with VoIP are IP telephony, Internet telephony, broadband telephony, and broadband phone service.

The term Internet telephony specifically refers to the provisioning of communications services (voice, fax, SMS, voice-messaging) over the public Internet, rather than via the public switched telephone network (PSTN). The steps and principles involved in originating VoIP telephone calls are similar to traditional digital telephony and involve signaling, channel setup, digitization of the analog voice signals, and encoding. Instead of being transmitted over a circuit-switched network, however, the digital information is packetized, and transmission occurs as IP packets over a packet-switched network. Such transmission entails careful considerations about resource management different from time-division multiplexing (TDM) networks.

Early providers of voice-over-IP services offered business models and technical solutions that mirrored the architecture of the legacy telephone network. Second-generation providers, such as Skype, have built closed networks for private user bases, offering the benefit of free calls and convenience while potentially charging for access to other communication networks, such as the PSTN. This has limited the freedom of users to mix-and-match third-party hardware and software. Third-generation providers, such as Google Talk, have adopted[1] the concept of federated VoIP—which is a departure from the architecture of the legacy networks. These solutions typically allow dynamic interconnection between users on any two domains on the Internet when a user wishes to place a call.

VoIP systems employ session control and signaling protocols to control the signaling, set-up, and tear-down of calls. They transport audio streams over IP networks using special media delivery protocols that encode voice, audio, video with audio codecs, and video codecs as Digital audio by streaming media. Various codecs exist that optimize the media stream based on application requirements and network bandwidth; some implementations rely on narrowband and compressed speech, while others support high fidelity stereo codecs. Some popular codecs include μ-law and a-law versions of G.711, G.722, which is a high-fidelity codec marketed as HD Voice by Polycom, a popular open source voice codec known as iLBC, a codec that only uses 8 kbit/s each way called G.729, and many others.

VoIP is available on many smartphones, personal computers, and on Internet access devices. Calls and SMS text messages may be sent over 3G or Wi-Fi

Protocols

Voice over IP has been implemented in various ways using both proprietary protocols and protocols based on open standards. Examples of the VoIP protocols are:

H.323
Media Gateway Control Protocol (MGCP)
Session Initiation Protocol (SIP)
H.248 (also known as Media Gateway Control (Megaco))

Real-time Transport Protocol (RTP)
Real-time Transport Control Protocol (RTCP)
Secure Real-time Transport Protocol (SRTP)
Session Description Protocol (SDP)
Inter-Asterisk eXchange (IAX)
Jingle XMPP VoIP extensions
Skype protocol
Teamspeak

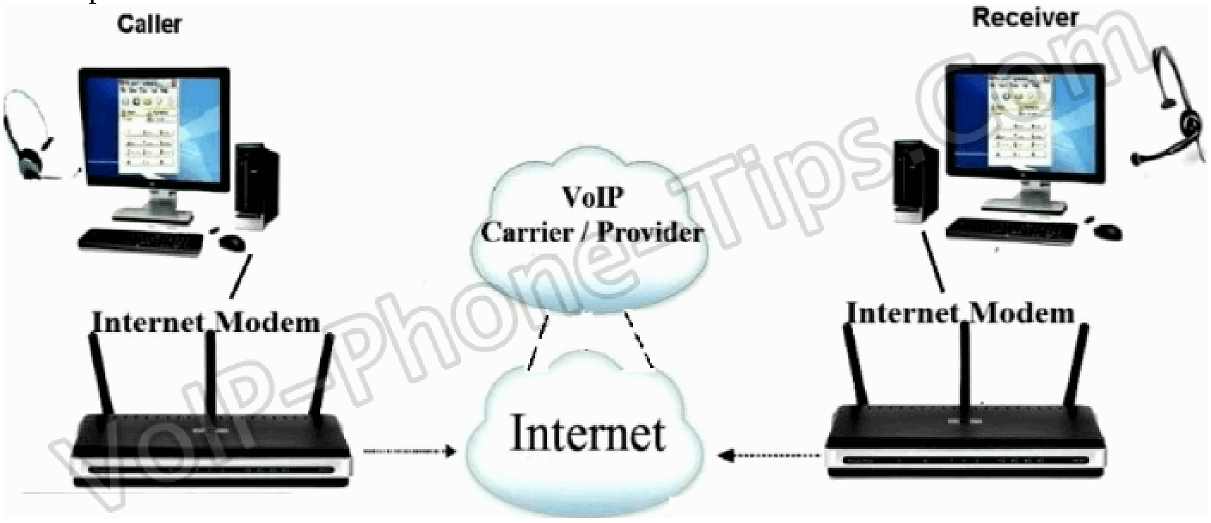


Figure 6.3 VoIP using Pc

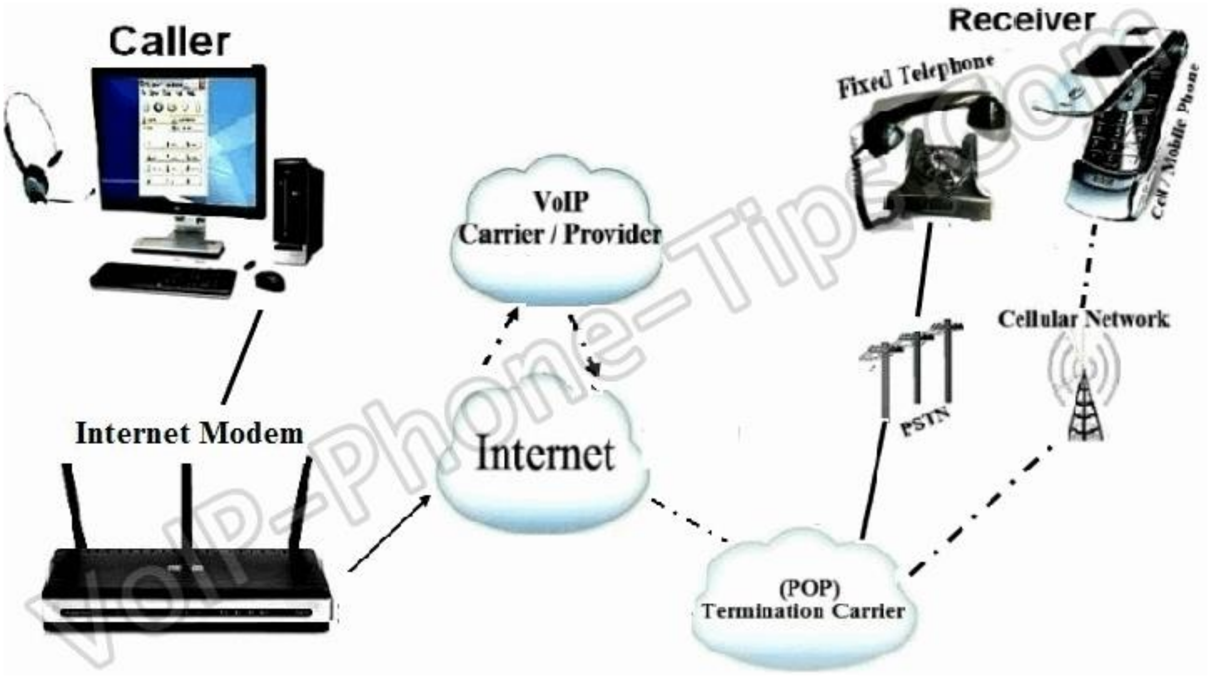
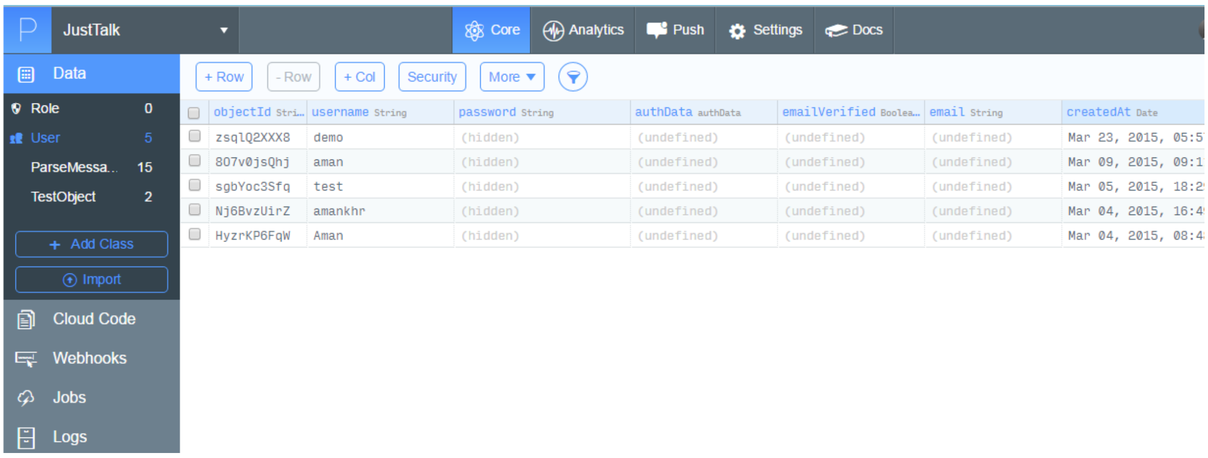


Figure6.4. VoiP using phone

CONCLUSION AND FUTURE WORK

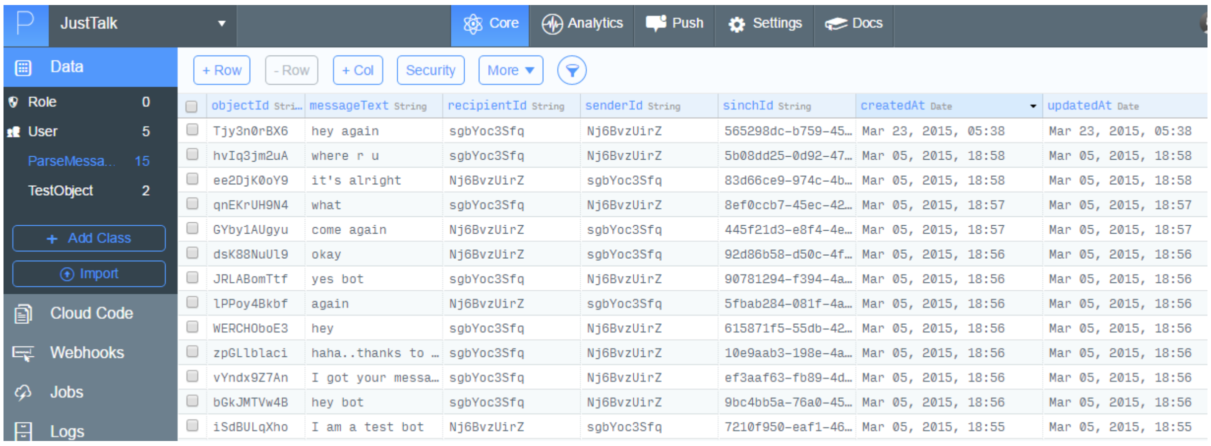
The requirement analysis and design of the project has been done successfully. The project has been successfully implemented. Subsequent updates will include volunteer blocking and video calling features. The project is now live and running. The maintenance and support for the users of the application is in process.

SNAPSHOTS OF THE ANALYTICS



objectId Stri...	username String	password String	authData authData	emailVerified Boolea...	email String	createdAt Date
zsqlQ2XXX8	demo	(hidden)	(undefined)	(undefined)	(undefined)	Mar 23, 2015, 05:5
807v0jsQhj	aman	(hidden)	(undefined)	(undefined)	(undefined)	Mar 09, 2015, 09:1
sgbYoc3Sfq	test	(hidden)	(undefined)	(undefined)	(undefined)	Mar 05, 2015, 18:2
Nj6BvzUirZ	amankhr	(hidden)	(undefined)	(undefined)	(undefined)	Mar 04, 2015, 16:4
HyzrKP6FqW	Aman	(hidden)	(undefined)	(undefined)	(undefined)	Mar 04, 2015, 08:4

Sample Users.



objectId Stri...	messageText String	recipientId String	senderId String	sinchId String	createdAt Date	updatedAt Date
Tjy3n0rBX6	hey again	sgbYoc3Sfq	Nj6BvzUirZ	565298dc-b759-45...	Mar 23, 2015, 05:38	Mar 23, 2015, 05:38
hvIq3jm2uA	where r u	sgbYoc3Sfq	Nj6BvzUirZ	5b08dd25-0d92-47...	Mar 05, 2015, 18:58	Mar 05, 2015, 18:58
ee2DjK0oY9	it's alright	Nj6BvzUirZ	sgbYoc3Sfq	83d66ce9-974c-4b...	Mar 05, 2015, 18:58	Mar 05, 2015, 18:58
qnEKrUH9N4	what	sgbYoc3Sfq	Nj6BvzUirZ	8ef0ccb7-45ec-42...	Mar 05, 2015, 18:57	Mar 05, 2015, 18:57
GYby1AUgyu	come again	Nj6BvzUirZ	sgbYoc3Sfq	445f21d3-e8f4-4e...	Mar 05, 2015, 18:57	Mar 05, 2015, 18:57
dsK88NuUl9	okay	Nj6BvzUirZ	sgbYoc3Sfq	92d86b58-d50c-4f...	Mar 05, 2015, 18:56	Mar 05, 2015, 18:56
JRLABomTtf	yes bot	sgbYoc3Sfq	Nj6BvzUirZ	90781294-f394-4a...	Mar 05, 2015, 18:56	Mar 05, 2015, 18:56
lPPoy4Bkbf	again	Nj6BvzUirZ	sgbYoc3Sfq	5fbab284-081f-4a...	Mar 05, 2015, 18:56	Mar 05, 2015, 18:56
WERCHOboE3	hey	sgbYoc3Sfq	Nj6BvzUirZ	615871f5-55db-42...	Mar 05, 2015, 18:56	Mar 05, 2015, 18:56
zpGLlblaci	haha..thanks to ...	sgbYoc3Sfq	Nj6BvzUirZ	10e9aab3-198e-4a...	Mar 05, 2015, 18:56	Mar 05, 2015, 18:56
vYndx9Z7An	I got your messa...	sgbYoc3Sfq	Nj6BvzUirZ	ef3aaf63-fb89-4d...	Mar 05, 2015, 18:56	Mar 05, 2015, 18:56
bGkJMTVw4B	hey bot	sgbYoc3Sfq	Nj6BvzUirZ	9bc4bb5a-76a0-45...	Mar 05, 2015, 18:56	Mar 05, 2015, 18:56
iSdBULqXho	I am a test bot	Nj6BvzUirZ	sgbYoc3Sfq	7210f950-eaf1-46...	Mar 05, 2015, 18:55	Mar 05, 2015, 18:55

Users Messages

REFERENCES

[1]D. Griffiths, *Head first android development*. [Place of publication not identified]: O'Reilly Media, 2015.

[2]M. Murphy, *The busy coder's guide to advanced Android development*. [U.S.]: CommonsWare, LLC, 2009.

[3]O. Ayokunle, 'Implementing Security on a Voice over Internet Protocol (VoIP) Network: A Practical Approach', *IOSRJCE*, vol. 7, no. 4, pp. 24-30, 2012.

[4]'VoIP for emerging devices', *Communications Engineer*, vol. 5, no. 5, pp. 26-31, 2007.

[5]I. Poole, 'What exactly is VoIP?', *Communications Engineer*, vol. 3, no. 2, pp. 44-45, 2005.

[6]R. Ebbinghaus, 'VoIP lessons', *Communications Engineer,* vol. 1, no. 5, pp. 28-31, 2003.

[7]L. Bhebhe and R. Parkkali, 'VoIP Performance over HSPA with Different VoIP Clients', *Wireless Pers Commun*, vol. 58, no. 3, pp. 613-626, 2010.

[8] Developer.android.com, 'Android Developers', 2015. [Online]. Available: http://developer.android.com. [Accessed: 01- Oct- 2014].

[9]J. Hamel, 'Android Messaging App Tutorial with Parse | Android App Tutorial | Sinch', *Sinch*, 2015. [Online]. Available: https://www.sinch.com/tutorials/android-messaging-tutorial-using-sinch-and-parse/. [Accessed: 01- Mar- 2015].

[10]C. Jensen, 'Using Sinch with Parse | Sinch', *Sinch*, 2015. [Online]. Available: https://www.sinch.com/tutorials/using-sinch-parse-sinch/. [Accessed: 08- Mar- 2015].

[11]J. Hamel, 'Send Push Notifications In a Android Messaging App with GCM', *Sinch*, 2015. [Online]. Available: https://www.sinch.com/tutorials/send-push-notifications-android-messaging-app-using-gcm/. [Accessed: 14- Mar- 2015].

[12]J. Hamel, 'App to Phone Calling with Android | Sinch', *Sinch*, 2015. [Online]. Available: https://www.sinch.com/tutorials/app-to-phone-calling-android/. [Accessed: 08- May-2015].

[13]J. Hamel, 'Build a Simple Android Calling App | Sinch', *Sinch*, 2015. [Online]. Available: https://www.sinch.com/tutorials/android-app-to-app-calling-tutorial/. [Accessed: 08- Apr- 2015].

[14] YouTube, '#1 Android Material Design Tutorial 1 with Android Studio [HD 1080p]', 2015. [Online]. Available: https://www.youtube.com/watch?v=h57QpXp2TRg&list=PLonJJ3BVjZW6CtAMbJz1XD8ELUs1KXaTD. [Accessed: 03- Apr- 2015].

[15] YouTube, '#2 Android Material Design Colors: Android Tutorial For Beginners [HD 1080p]', 2015. [Online]. Available: https://www.youtube.com/watch?v=hrlGVU8z7zc&index=2&list=PLonJJ3BVjZW6CtAMbJz1XD8ELUs1KXaTD. [Accessed: 04- Apr- 2015].

[16] YouTube, '#3 Android ToolBar Example: Android Tutorial For Beginners [HD 1080p]', 2015. [Online]. Available: https://www.youtube.com/watch?v=pMO8EVkhJO8&list=PLonJJ3BVjZW6CtAMbJz1XD8ELUs1KXaTD&index=3. [Accessed: 04- Apr- 2015].

[17] YouTube, '#4 Android Customize Toolbar: Android Material Design Tutorial [HD 1080p]', 2015. [Online]. Available: https://www.youtube.com/watch?v=4XfDDfa3rv8&list=PLonJJ3BVjZW6CtAMbJz1XD8ELUs1KXaTD&index=4. [Accessed: 05- Apr- 2015].

[18] YouTube, '#5 Adding Actions to ToolBar/ActionBar/AppBar : Android Tutorials [HD 1080p]', 2015. [Online]. Available: https://www.youtube.com/watch?v=EAZv1fP-5TM&list=PLonJJ3BVjZW6CtAMbJz1XD8ELUs1KXaTD&index=5. [Accessed: 06- Apr- 2015].

[19] YouTube, '#6 Android Material Design Navigation Drawer Part 1 [HD 1080p]', 2015. [Online]. Available: https://www.youtube.com/watch?v=zWpEh9k8i7Q&index=6&list=PLonJJ3BVjZW6CtAMbJz1XD8ELUs1KXaTD. [Accessed: 06- Apr- 2015].

[20] YouTube, '#7 Android Navigation Drawer Material Design Part 2 [HD 1080p]', 2015. [Online]. Available: https://www.youtube.com/watch?v=tR2K_Sav7q8&index=7&list=PLonJJ3BVjZW6Ct

AMbJz1XD8ELUs1KXaTD. [Accessed: 07- Apr- 2015].

[21] YouTube, '#8 Material Design Navigation Drawer Part 3: Android Tutorials [HD 1080p]', 2015. [Online]. Available: https://www.youtube.com/watch?v=AsbtXnJ1zww&list=PLonJJ3BVjZW6CtAMbJz1XD8ELUs1KXaTD&index=8. [Accessed: 07- Apr- 2015].

[22] YouTube, '#9 Android Navigation Drawer Material Design Part 4 [HD 1080p]', 2015. [Online]. Available: https://www.youtube.com/watch?v=TBKpsGar5mc&index=9&list=PLonJJ3BVjZW6CtAMbJz1XD8ELUs1KXaTD. [Accessed: 08- Apr- 2015].

[23] YouTube, '#10 Material Design Navigation Drawer Types: Android Tutorials [HD 1080p]', 2015. [Online]. Available: https://www.youtube.com/watch?v=pJPRPvZoNt4&index=10&list=PLonJJ3BVjZW6CtAMbJz1XD8ELUs1KXaTD. [Accessed: 08- Apr- 2015].

[24] Parse.com, 'Parse', 2015. [Online]. Available: https://www.parse.com/docs/android/guide. [Accessed: 08- Mar- 2015].