# MessagingApp

## An app to send messages over the Internet

**José Luis Cerrada Barrios**

**10/05/2012**

Dundalk Institute of Technology

# Acknowledgements

Thanks to Ciaran Cawley and Dermot Logue for
his patience and for helping me out.

# Abstract

*For my final project, I propose to develop an app to send messages over the internet using android devices, obviously the application has more functionalities, but this is the main function of MessagingApp.*

*In this document I'm going to explain the process I have followed to develop my application (MessagingApp), as well as the researching I have made to be able to build a competitive messaging system. To achieve this objective, I have divided the report into two clearly differentiated parts: The Literature Review and The Proposed Solution.*

*In the Literature Review I'm going to explain the current context of text messaging and what is the profile of users who can use this kind of apps. Also, I'm going to talk about the success of others applications which are used to send messages using the internet connection of mobile devices. I'm going to describe how some of these apps were created, how work and what their strong and weak points are. Furthermore, I will explain why I have chosen Android as development platform, for this task, I have to compare this OS with others mobile OSs which are currently in the market like iOS for iPhone or Windows Phone. To carry out this comparison, I'm going to consider different aspects for each operating system: firstly I'm going to see what are the politics of each company regard uploading an app to their respective market; later I'm going to evaluate the difficulty of developing an application for the different OSs, taking into account my current knowledge, and finally I will decide what is the most suitable OS to develop my app considering the advantages and disadvantages of using an OS or other, and also taking into account the potential market size.*

*In the "Proposed Solution" section I'm going to explain in detail all functionalities of my application showing some screenshots to understand everything more easily. I will describe the technologies I'm going to use to develop MessagingApp on both the client site as in the server site. Furthermore, I will make a comparison between my app and the other apps described in the Literature Review, telling what are the ideas I have taken from them to design MessaginApp and what features has MessagingApp that the other apps don't have.*

*And finally, I will show and example of MessagingApp running.*

# Index

1 Introduction ..... 4
2 Literature Review ..... 5
2.1 Introduction ..... 5
2.2 User Profile ..... 5
2.3 Instant Messaging Apps ..... 6
2.3.1 WhatsApp [12] ..... 6
2.3.2 GroupMe [7] ..... 7
2.3.3 Viber [10] ..... 7
2.3.4 Comparison between Whatsapp, GroupMe and Viber ..... 7
2.4 Mobile Platforms ..... 8
2.4.1 Parts of a Mobile Platform ..... 8
2.4.2 Android ..... 9
2.4.3 iOS ..... 10
2.4.4 Windows Phone ..... 11
2.4.5 Android vs iOS vs Windows Phone ..... 13
3 Proposed Solution ..... 15
3.1 What is MessagingApp? ..... 15
3.2 Design ..... 15
3.2.1 Class Diagram ..... 17
3.2.2 Protocol ..... 18
3.3 Implementation ..... 20
3.3.1 Choosing a Mobile Platform ..... 20
3.3.2 Choosing a Platform for Server Site ..... 20
3.3.3 Development IDE ..... 21
3.3.4 Classes ..... 21
3.4 Testing ..... 29
3.4.1 MessagingApp ..... 29
4 Conclusion ..... 35
5 References ..... 36
6 Appendix ..... 38

# 1 Introduction

Nowadays, the most people need to be permanently connected with the people around them. According to recent investigations, there are 5.3 billion mobile subscribers [6], although it's not means that the 77% of world population has a mobile phone, as there are people who have several phones. One of the main ways of communications is texting, only in 2010 were sent over 6 trillion of text messages [13] and in 2011 were sent 8 trillion of text messages [6]. We don't have to calculate the amount of money which is 8 trillion of messages to know it's so much money. For that reason, I believe that an application, which allows you to send messages for free and easy, would have success, especially knowing that half a billion of people were broadband subscribers in 2010 and it's expected that this amount is doubled in 2015 [6].

I think that in the future, when the most people have a smartphone with an unlimited data plan (this is the current trend, the smartphone market is showing a strong growth, more and more smartphones are sold), this kind of applications will end up with the SMSs.

In this document I will show to the reader the process I have followed to develop MessagingApp, starting with the *Literature Review,* followed by the *Proposed Solution* and finishing by the *Conclusion.*

# 2 Literature Review

## 2.1 Introduction

The aim of this literature review is to show the researching I have realized for the development of my project, starting with the studying of the current context, this part is very important, because by this way you can know what the current trends are and what is the way to follow to develop a success app. After of that, I'm going to explain the profile of users who usually use this kind of application, this is a critical point, because if you don't know your target market, surely your app will be misplaced.

To develop a good app, I think you have to investigate others applications which currently are in the market, see what are their best features and where they can improve and check if the users of those application wants more functionalities, and then basing in this researching you start to develop your app.

This app is thought for mobile phones, so I have to choose the most appropriate platform to develop MessagingApp. For this task, I have researched the most popular platforms currently: Android, iOS and Windows Phone. And I have chosen the platform according to this investigation and my previous experiences working with the programming languages used for development in each platform.

## 2.2 User Profile

This application is thought for all those people who want to be permanently connected with all of their friends, family or co-workers. The age range can be very large. I don't think that this app is only for teenagers, the confusion is due to you can think that this app is a type of instant messaging like windows messenger or something like that, and you are right, but there is a difference because this application is always running either in foreground or in background, by this way we can get a similar service to the SMSs, but much more powerful and with the possibility of adding more functionalities based on internet services which can make easier the day by day of users.

A typical user can be somebody who wants to communicate with his friends sending messages for free, somebody who wants to be able to create groups of people to send messages to these groups, somebody who has a group named "*Friends*" and wants to send a message to this group to meet for coffee.

This app can be very useful for workers of a company. For example, in a software development company his workers can be divided into groups, so that each group develop a part of an application. So that the members of the groups work well together, each group has to have a manager and is here where the app can be very useful, because the leader can send a message to all of members of his group telling them what they have to do, if they have they change something or whatever he needs, in the same way, the members of the group can send messages each other to ask or report something to the others members.

We can conclude telling that this app can be used by a broad variety of people, although I have to recognize that the most users are going to be from 16 to 40 years old, but each one of them can use the app for a different end.

## 2.3 Instant Messaging Apps

Instant messaging is a form of communication in real time between two or more people based on text. The text is sending through devices connected to a network such as the internet.

The instant messaging requires the use of a messaging client which realizes the service and the difference regarding the email is that the communication is in real time. The most applications of this type offers "Notice of presence", telling to the user when a member of his list of contacts is connected or what is his state, if is available to talk or not. In the first applications of instant messaging each letter was sent at the same time as the user wrote it, and the corrections of the errors were saw in real time too. This feature gave to the conversations the sensation of a phone call more than an exchange of text. In the current programs, usually it's sent complete sentences. Furthermore, in some messaging programs you can send a message even if the other user is not connected at this moment.

For mobile phones there are several applications of instant messaging, although the applications I have researched, like mine, are not exactly instant messaging apps, but are pretty similar.

### 2.3.1 WhatsApp [12]

Whatsapp is a multiplatform proprietary software of instant messaging for smartphones. In addition to sending text, WhatsApp allow sending pictures, audio and video, and the location of the user if it is possible. The application uses the data network of the mobile device in which is running, therefore works connected to the Internet unlike to traditional short messages. WhatsApp is available for several operating systems: iOS, Android, BlackBerry OS and Symbian.

The first year of use, the app is free for all operating systems, but after this time the user has to pay a little amount of money to renew the contract another year.

The company which created this app is WhatsApp Inc. and was founded by Jan Koum, who had been previously the director of platform operations team at Yahoo and former head of the engineering team Brian Acton.

**Differences between WhatsApp and SMSs:** SMS is an old messaging system, with lower levels of functionality, a limited number of characters (between 140 and 160) and high cost. The purpose of WhatsApp Messenger is to improve these functionalities and with a much lower cost (either with a data plan or via Wi-Fi).

### 2.3.2 GroupMe [7]

GroupMe is also a multiplatform proprietary software of instant messaging focused on sending messages to groups of people. Furthermore, this app offers the same services as WhatsApp, to send text messages, pictures, audio, video and the location of the user if this is possible. GroupMe uses the internet connection of the mobile phone, either through a data plan or via Wi-Fi, therefore the sending of messages is free.

This app is completely free and is available for the most popular operating systems: iOS, Android, BlackBerry OS and Windows Phone.

This app was founded by Jared Hecht and Steve Martocci in 2010. There is something curious in the creation of this app and it is that the idea to develop GroupMe arose because the wife of Jared needed to send a message to all her friends to arrange a meeting and she didn't find a good way to do it.

GroupMe was acquired by Skype in August of 2011 for around $80 million.

### 2.3.3 Viber [10]

Viber is an application to make phone calls for free using the internet connection of the mobile phones. Furthermore, this app allows users to send free text messages to others users who have installed the application. Viber checks the user's address book and detects if some of his contacts has also installed the application.

Viber is only available for the main mobile platforms in the current market: Android and iOS. His founders say that soon it will be available for BlackBerry too. Viber is a completely free app either for iOS or Android.

Really, this app is pretty different from the others that I have explained, because is not focused on sending messages, his main feature is to make free phone calls, but I have researched this app because the most people with I have talking here, in Dundalk, only know this app to send free messages.

### 2.3.4 Comparison between Whatsapp, GroupMe and Viber

To develop a good app is necessary to research what are the other applications in the market and what are their strong and weakness points, for this reason I make this comparison.

Whatsapp was created thinking in the communication between two people, not between groups of people. Later, due to the success of GroupMe which offered the same functionalities plus the possibility to create groups of people and send messages to groups, Whatsapp was force to implement group chats. But this functionality is not integrated very well, it's pretty difficult to create groups and doesn't let you add new people once the group has been created, furthermore groups have a limit of 10 people.

GroupMe was created after Whatsapp and as I have said before, has the same functionalities plus the possibility to create groups of people. But this app is focused on creating groups and group chats. For this reason, to send messages to an individual person is more complicated than in Whatsapp. His own creator, Jared Hecht, said that

GroupMe was born due to the necessity to send messages to group of people in real time through mobile phones.

Viber is an app thought to calls using the Internet connection (something like Skype), furthermore has another functionality which is sending messages to other people who have also installed the app, but this is a secondary service therefore is not very polished. Whatsapp or GroupMe are better in this aspect.

What I want to say is these apps are very good in their fields, Whatsapp is very good to send messages to single person, GroupMe to send messages to groups of people and Viber to call using Internet, but are not very good in the fields of the others.

My app unifies the easiness to send messages to individual people of Whatsapp and the easiness to create groups and send messages to these groups of GroupMe. Furthermore, my app will offer other functionalities which the rest of applications don't have, and these are:

- Searching for messages from a friend/group or all contacts between two dates.
- In the cases of group chats, the choice to reply only to some members of the chat.
- Sending messages to all your contacts who have a mutual particular friend.

## 2.4 Mobile Platforms

A mobile platform is an operating system which runs a mobile device, as in the case of the computers that we can find Windows, Linux, MAC OS…Although are more simples than a desktop operating system and are focused on wireless connection, multimedia formats for mobiles and the different ways of input information on them.

### 2.4.1 Parts of a Mobile Platform

#### Kernel

The kernel provides access to the different hardware elements of the device. Offers various services to the other parts like drivers for the hardware, process management, file system and access and management of the memory.

#### Middleware

Middleware is the set of modules that make possible the existence of mobile applications. It is completely transparent to the user and provides key services such as messaging and communications engine, multimedia codecs, websites interpreters, device management and security.

### User Interface

The user interface facilitates the interaction between the user and applications. The services which this part offers are the graphics components (buttons, layouts, lists…).

According to the current trend regarding mobile platforms in smartphones, we can highlight the following: Android, iOS and Windows Phone. Once we have researched all of them, we will be able to know what is the best platform to develop MessagingApp.

## 2.4.2 Android

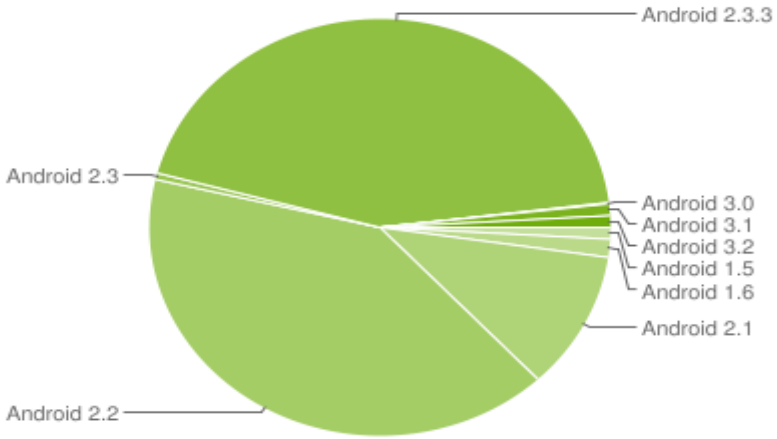
Android is an operating system based on the Linux Kernel and designed originally for mobile devices like smartphones, but later his development was expanded to support other devices such as tablets, mp3 players, netbooks, PCs, TVs and e-book readers.

Android was initially developed by Android Inc. a start-up company which was acquired by Google in 2005. It's the main product of the Open Handset Alliance, a cluster of manufacturers and developers of hardware, software and service operators.

The first version of Android was released on 23 September 2008 (Android 1.0), after Google released new versions: Android 1.1, Android 1.5 (Cupcake), Android 1.6 (Donut), Android 2.0 (Eclair), Android 2.2 (Froyo), Android 2.3 (Gingerbread), Android 3.0 (Honeycomb) and the latest version is Android 4.0 (Ice Cream Sandwich).

We can see an infographic of the History of Android in the next website: http://holaandroid.com/wp-content/uploads/2011/08/The-Andriod-Story.png [14]

A lot of expert in this area think that the worst problem of Android is his fragmentation, because currently there are many mobile phone with different versions of Android, as we can see in the next picture:

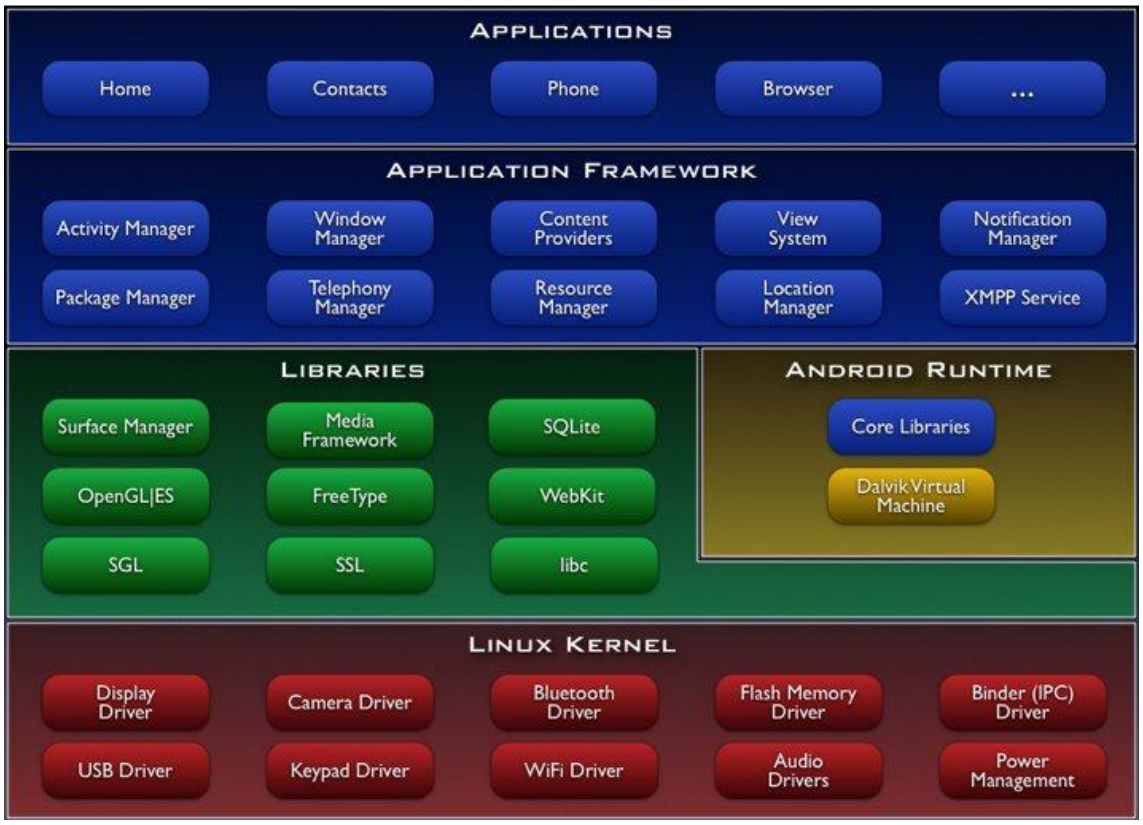



*Android Platform Versions chart*
http://developer.android.com/resources/dashboard/platform-versions.html

### 2.4.2.1 Android Application Development

Applications are usually developed in Java, using the Android Software Development Kit (Android SDK), although there is available other development tools to design applications in C or C++. The Android SDK is available for Windows, Linux and MAC OS.

To develop an Android application, you don't have to learn a completely new language, because the Android SDK is based on Java [2].




*Android System architecture*
*(Wikipedia, 2007)*

### 2.4.2.2 Android Market

Android Market is the online store software developed by Google for Android devices. An app called "Market" is preinstalled on most Android devices and allows users to browse and download applications which are hosted on Android Market.

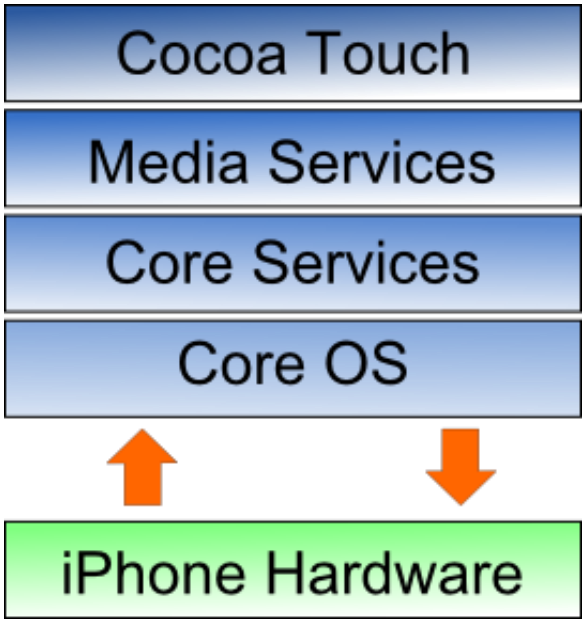
Developers can upload their applications to the Market without any obstacle, although previously they have to realize three steps: register as a merchant, upload and describe their app and publish it. To register as a developer and be able to upload applications, you must pay a registration fee (U.S. $ 25.00). Google rewards developers with 70% of the revenues of their application [1].

## 2.4.3 iOS

iOS is an operating system developed by Apple for the iPhone, although later it has been also used in iPod Touch and iPad. iOS is based on MAC OS X which in turn is

based on Darwin BSD. This operating system has been programed in C, C++ and Objective-C.

iOS has four layers of abstraction: the layer of the operating system kernel, the layer of "Core Services", the layer of "Media" and the layer of "Cocoa Touch" [3].




*iOS Architecture*
*(Techotopia, 2010)*

### 2.4.3.1 Application Development in iOS

Applications for iOS are developed in Objective-C, which is an object orientated programming language based on C language. To be able to develop apps for iOS, developers need the specific SDK for this OS; this SDK was released by apple in February 2008 and only is available for the XCode IDE which in turn only can be used in MAC OSX, for this reason, the development applications for iOS is more complicated than in Android. Furthermore, Apple developed a simulator in which developers can test their apps before upload them to the App Store.

### 2.4.3.2 App Store

App Store is a service for iPhone, iPod Touch and Mac OS X, created by Apple Inc. that allows users to search and download applications. Developers who publish their applications in the App Store receive 70% of sales revenue and don't have to pay any costs of distribution of their application, however, they must pay a fee of $ 99 in the case of *Developer Program* and $200 in the case of *Enterprise Program* to use the iPhone SDK and upload applications to the store.

## 2.4.4 Windows Phone

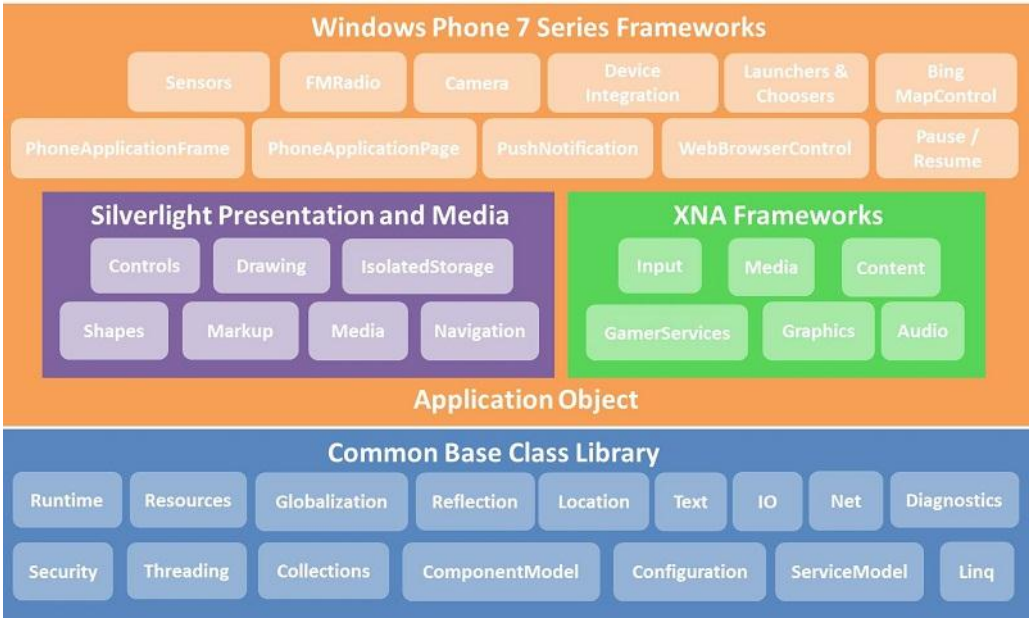
Windows Phone, previously named Windows Mobile, is an operating system for smartphones developed by Microsoft. Windows Phone is based on the kernel of Windows CE and has a set of basic applications which use the API of Microsoft

Window. The first version of this operating system was called PocketPC and after her have been released new versions: Windows Mobile 2003, Windows Mobile 5.0, Windows Mobile 6.0, Windows Mobile 6.1, Windows Phone 6.5 and the latest version which is Windows Phone 7.

### 2.4.4.1 Application Development in Windows Phone

To develop applications for Windows Phone, developers can use two types of implementations [4]:

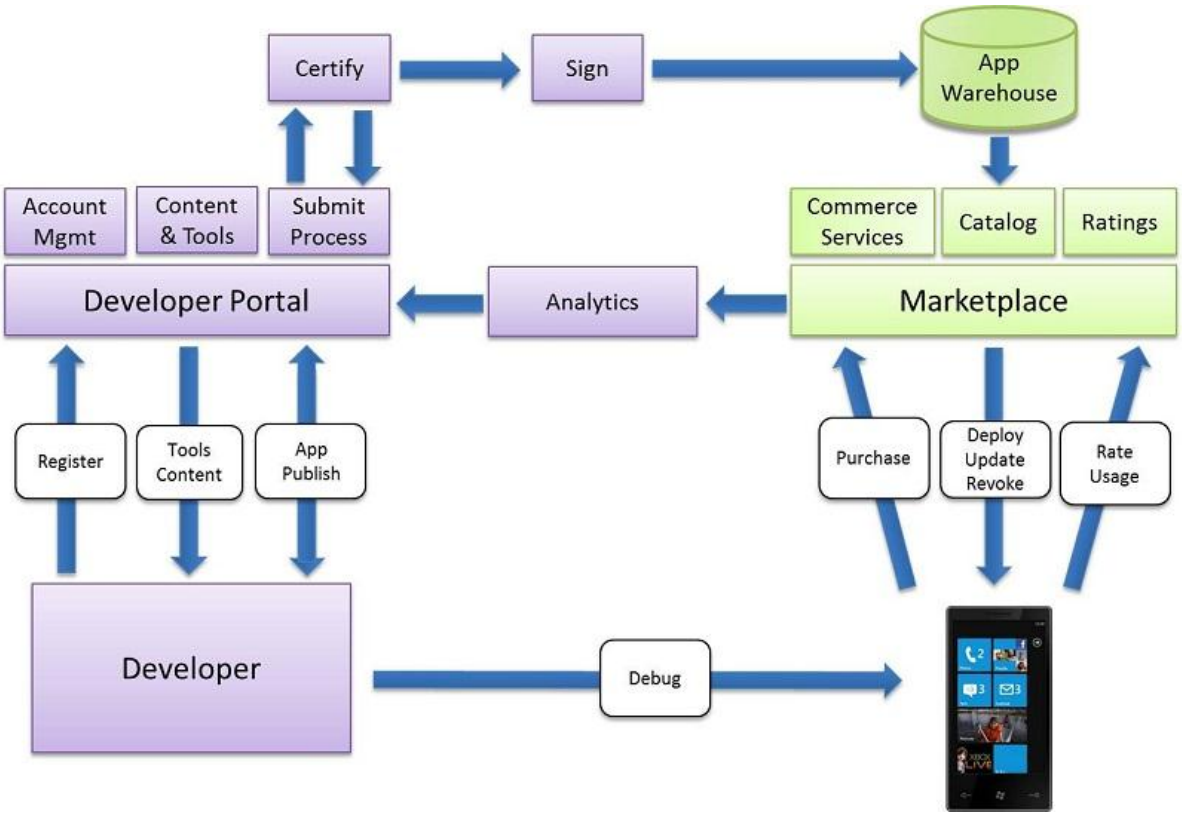
1. Microsoft Silverlight which allows developing apps that contain transitions and visual effects. Silverlight lets the development of applications based on XAML and includes the Microsoft .NET Compact Framework which in turn inherits from .NET Framework architecture.
2. Microsoft XNA Framework which is a native implementation of .NET Compact Framework and includes a broad set of library classes which are specifics for the development of games.




*Windows Phone 7 Series Frameworks*

### 2.4.4.2 Hub Marketplace

The Hub Marketplace is a site where Windows Phone users can buy and download any type of content such as applications, music, films, TV programs… A lot of contents can be tested before being downloaded. To be able to upload an application to the Hub Marketplace, developers have to create an account in this site and pay a fee of $99. Either way, every app must be approved by Microsoft. We can see the process to develop an app and submit it to the market in the next picture. As in the other two mobile platforms, Microsoft rewards the 70% of the revenues of the applications to their developers.




*Windows Phone Life Cycle*

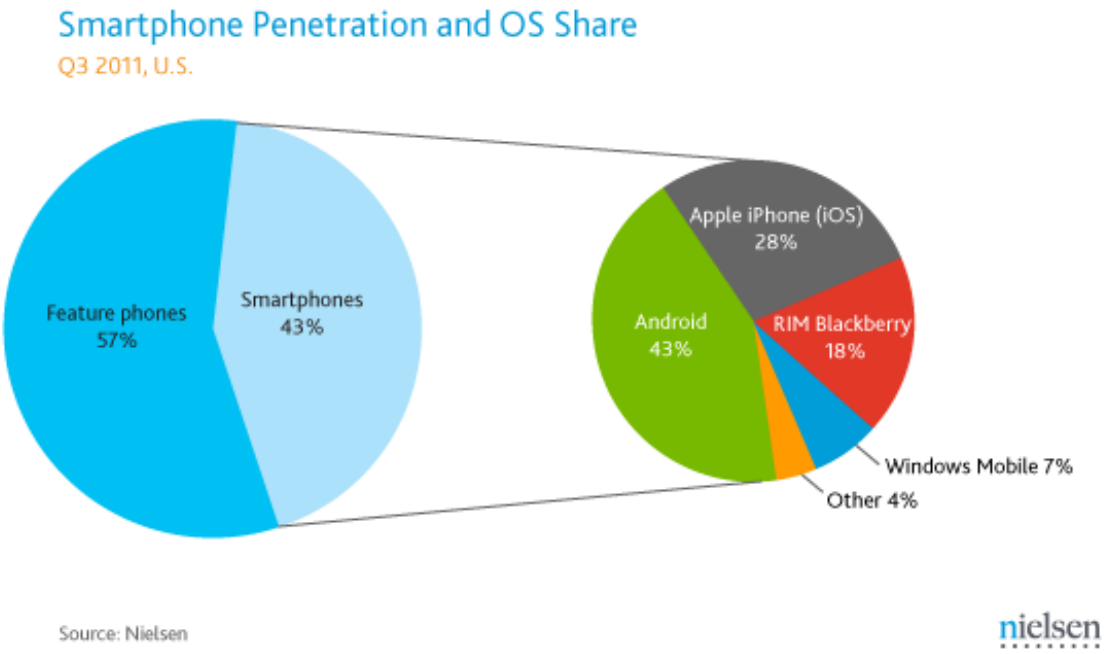
## 2.4.5 Android vs iOS vs Windows Phone

Once we have researched the main mobile platforms which exist in the current market, we have to choose what is the most suitable of all of them to develop an app, obviously the best choice would be to develop the application for all mobile platforms, but there is no enough time to do this, so we have to choose one.

| | Android | iOS | Windows Phone |
|---|---|---|---|
| Experience with the necessary programming language | ✓ | ✗ | ✗ |
| Computer with the necessary operating system for development | ✓ | ✗ | ✓ |
| Access to a mobile phone with the operating system concerned | ✓ | ✗ | ✗ |
| Fee required to be able to develop and upload applications | 25$ | 99$ | 99$ |

| | | | |
|---|---|---|---|
| Number of applications in the market | 332700 [5] | 500,000+ [9] | 40.000+ [8] |

(Figure 5.4.1)




*Smartphone Penetration and OS Share*

# 3 Proposed Solution

## 3.1 What is MessagingApp?

*MessagingApp* is an application to send messages over the internet. In addition to this, the app has other functionalities:

- Users can send a message to multiple people; furthermore they have the possibility to create group chats.
- The application will allow to users creating group of people to keep organized their contacts, like *friends, co-workers, family* ... and by this way, they will be able to send messages to a group.
- Users can also send messages to people who are not connected at that time, and they will receive the messages when they connect.
- All messages are stored in the mobile-phone, by this way the user can see all his conversations and the date of the receiving/sending messages. Of course, the user can delete his conversations when he wants.
- The application has another functionality which is to send a message to all his contacts who have a mutual particular friend.

## 3.2 Design

Every time that a developer wants to design an application, the first thing he has to do is identifying the classes that he is going to need to develop his app. In my case, I have to identify the classes which are going to compose the server and on the other hand, the classes of the client site.

### Server

The server is composed by the following classes: *ServerMA, ClientThread, ProtocolMessage, MessagesDB, UsersDB* and *GroupsDB.*

- **ServerMA:** is the main class of the server. His function is to receive the petitions of connection from clients. It has a hashmap which contains all the connections of the clients who are connected at this time.
- **ClientThread:** this class allows us to manage each client connection. Using this class, the server can send messages to the clients and process the requests from them.
- **ProtocolMessage:** this class allows us building messages to be sent. Also it allows us to split the messages to get the different parts of them. Using this class we can convert a Message Object into a string with JSON format to be sent.
- **MessagesDB:** through this class we create the table “messagesDB” in the server database where the server is going to store the messages that could not be sent to the clients due to they were not connected or any other problem.

- **UsersDB:** through this class we create the table "usersDB" in the server database where the server is going to store the information of each user who has installed MessagingApp.
- **GroupsDB:** through this class we create the table "groupsDB" in the server database where the server is going to store the information on group chats (name of the group chat and members).

## Client

The client is composed by the following classes: *MessagingAppClient, Login, MessagingAppService, ProtocolMessage, ChatsTab, ChatsDB, ContactsTab, ContactsDB, GroupsTab, GroupsDB, CreateGroup, GroupMembers, MembersGroupDB, MessagesChat* and *MessagesDB.*

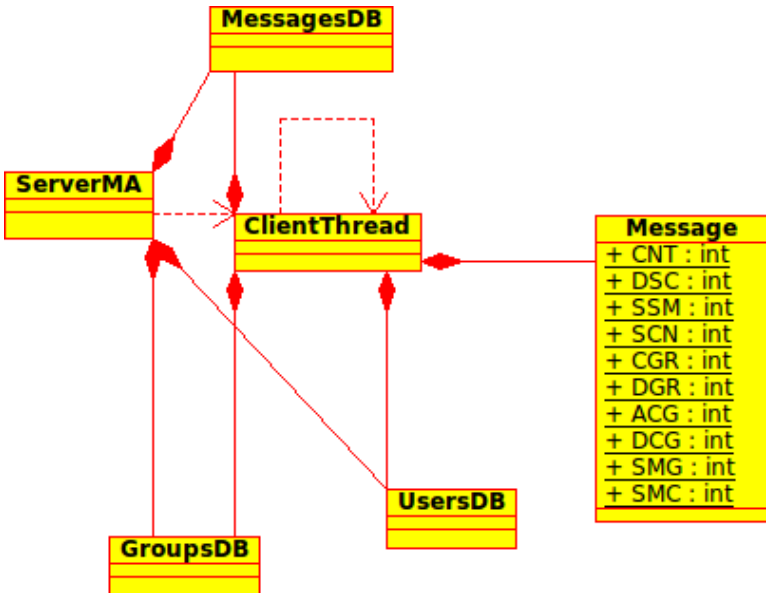
- **MessagingAppClient:** is the main class of the client. This class shows the main interface of the app.
- **Login:** through this class the client site asks the user his phone number.
- **MessagingAppService:** this class is going to handle the connection between the server and the client site. Through this class, the client will receive the messages from the server and will send the messages to the server. Also, this class will notify the user the incoming messages through a notification.
- **ProtocolMessage:** is the same class than in the server site. This class allows us building messages to be sent. Also it allows us to split the messages to get the different parts of them. Using this class we can convert a Message Object into a string with JSON format to be sent.
- **ChatsTab:** this class defines the interface where it will be shown the last conversations.
- **ChatsDB:** through this class the client creates the table "chatsDB" in his database. This table is going to store the information on chats (name of the chat, last message and date of this message).
- **ContactsTab:** this class defines the interface where it will be shown the contacts of the client who have installed the MessagingApp.
- **ContactsDB:** through this class the client creates the table "contactsDB" in his database. This table is going to store the information on contacts (name and phone number);
- **GroupsTab:** this class defines the interface where it will be shown the groups that the client has created.
- **GroupsDB:** through this class the client creates the table "groupsDB" in his database. This table is going to store the information on groups (only the name);
- **CreateGroup:** this class defines the interface to create a group. The user has to type the name of the group and select the members.
- **GroupMembers:** this class defines the interface where it will be shown the members of a specific group.

- **MembersGroupDB:** through this class the client creates the table "membersGroupDB" in his database. This table is going to store the name and the phone number of all members of a specific group, and the id of such group.
- **MessagesChat:** this class defines the interface where it will be shown all the messages of a specific conversation (chat).
- **MessagesDB:** through this class the client creates the table "messagesDB" in his database. This table is going to store all the information on messages of a chat (id of the chat, name of the sender, phone number of the sender, the message and the date).

## 3.2.1 Class Diagram

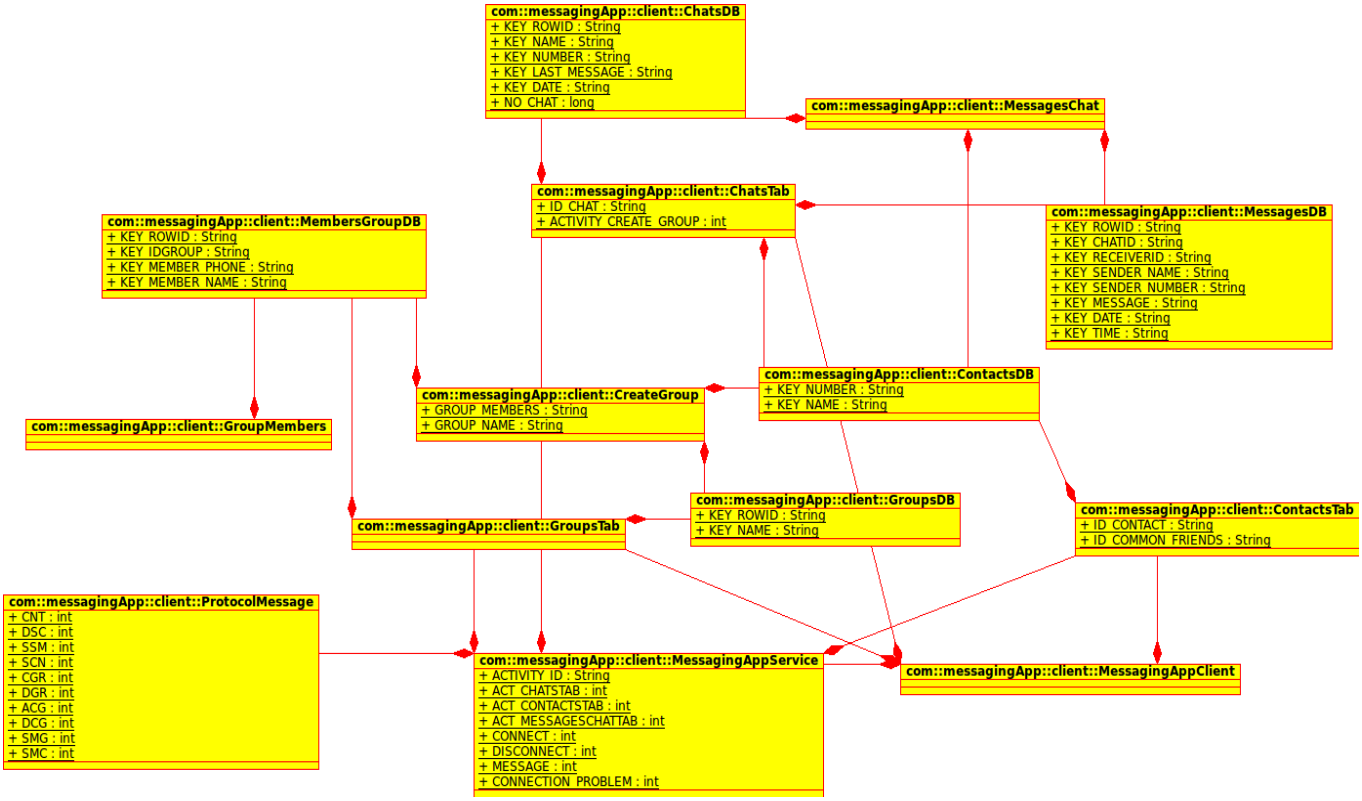
Once we know what classes are going to compose our application, now we have to see the interactions between them. For this task, I have used class diagrams, because are the best way to describe the interaction between the classes of an application.

### Server




(*Figure 4.2.1.1*)

## Client




(*Figure 4.2.1.2*)

## 3.2.2 Protocol

Every client-server communication needs a protocol for that each part can understand the other one. In the case of MessagingApp, we have a protocol with the following types of messages:

- **CNT:** This message is only sent by the client. Using this message, the client indicates the server that he wants to connect. If the user has never connected, then the server stores the information of the user in the table “usersDB” of his database.
- **DSC:** As in the previous type of message, this one is also only used by the client to indicate the server that he wants to disconnect.
- **SSM:** this type of message indicates to the server that he has to send the message to the specified contact or contacts in the “receivers” field of the message. If some receiver is not connected at this time, the server stores the message in the table “messagesDB” of his database, and this receiver will receive the message when he connects.
- **SCN:** this message can be sent by the client and by the server. When the client runs the app, automatically this message is sent to the server with all the contacts he has, then the server checks who of these contacts have installed the app and send back a SCN message indicating those contacts.
- **CGR:** this type of message can be sent by the client and by the server. When a client wants to create a group chat, he sends this message to the server indicating the name of the group chat and the members of that group, then the server stores

the group chat information in the table "groupsDB" of his database and sends the same type of message (CGR) to all members of the group.

- **DGR:** this type of message can be sent by the client and by the server. When the client who created the group chat wants to delete it, he sends this message to the server indicating the name of the group chat, then the server deletes the group chat from his database and sends a DGR message to all members of the group to tell them that the group chat has been deleted.
- **ACG:** this message is sent by the user and by the server. When a member of a group chat wants to add a new contact to the group, he sends a ACG message to the server indicating the name of the group chat and the number of the new member, then the server updates the concerned group chat in the database and notify to all members of the group sending them a ACG message.
- **DCG:** this type of message can be sent by the client and by the server. When a member of a group chat wants to delete a contact of that group, he sends this type of message to the server indicating the name of the group chat and the number of the contact he wants to delete, then the server delete that contact from the database and sends a DCG message to all members of the group to notify them that a member has been deleted.
- **SMG:** this type of message can be sent by the client and by the server. When a client wants to send a message to a group chat, he send a message of this type to the server, then the server resends the same message to all members of the group chat. If some of these clients are not connected at this time, the server stores the message in the table "messagesDB" of his database and when those clients connect, they will receive the message.
- **SMC:** This type of message is only sent by the client. When a client wants to send a message to all his contacts who have a common friend, he sends this type of message indicating all his contacts and the common friend, and when the server receives the message, he checks the common friends and sends a SSM message to all these clients.

The structure of a message is the following:

| Type | Sender | Receivers | Date | Group | Message |
|---|---|---|---|---|---|

Type: The type of message.

Sender: The phone number of the sender.

Receivers: The phone numbers of all receivers.

Date: The date in which the message was sent.

Group: This field is optional. In the case of CGR, DGR, ACG, DCG and SMG messages, it contains the name of the group.

Message: This field is also optional. In the case of SSM and SMC messages, it contains the message that the user wants to send.

# 3.3 Implementation

Once we have clear all the matters related with the design of our application, now it's time to start to develop it. Firstly we have to choose a mobile platform for the client site, then we must decide in which language we are going to develop the server site and what are going to be the technologies that we will use to communicate the client and the server. And finally, I will explain all the methods of each class of MessagingApp, either the client or the server site.

## 3.3.1 Choosing a Mobile Platform

After all this research about the different mobile platforms, we can conclude that the most suitable mobile platform to develop an application is Android. The reasons of this choice can be seen in the table above (Figure 5.4.1) and are the following:

- I have experience with C, C++ and Java and the only platform which uses one of these programming languages is Android, because iOS uses Objective-C which is similar to C and Windows Phone uses .NET.
- I can use my computer to develop applications for Android and Windows Phone, but if I want to develop an app for iOS, then I need a computer with MAC OS X, for this reason I have ruled out iOS.
- Another reason to choose Android is because I have a mobile phone with Android Gingerbread, so I can test my app in a real mobile phone when I want.
- Furthermore, I have found a lot of tutorials about how to develop an Android application and there is a webpage which is very good for beginners in this task: http://developer.android.com/index.html

## 3.3.2 Choosing a Platform for Server Site

As any messaging application, MessagingApp needs a server site to allow the communication between users. The programming language that I'm going to use to develop the server site is Java, because is also the language that Android uses, therefore I think that the compatibility is higher than if I use C or C++ for the server. Furthermore, the development of MessagingApp will be faster and will have better quality, because I only have to research one programming language instead of two.

SQLite will be the chosen database to store the information relative to the users, the groups they have created and the messages they have sent, due to his easiness of use and his power, and because is also used in client site to store the messages.

For the communication between the client and the server, I'm going to use the java object *Socket*, because this class allows us managing the communication in a low level, so we will have a higher control of the communication and hence, we will be able to have more knowledge about what it's going on in any moment.

To codify the messages I will use *JSON* with a Google API called *GSON* which allows us converting any java Object into a *JSON* Object. So, when we send a message, the first thing we do is transform the object *ProtocolMessage* into a JSON object, and

then we parse this *JSON* object into a *String* object, and this *String* is the object that we are going to send to the other site.

## 3.3.3 Development IDE

To develop my application I'm going to use Eclipse with the Android Development Tools plugin (ADT). ADT extends the capabilities of Eclipse to allow creating Android projects quickly, add components based on Android Framework API, debugging applications using the SDK tools…

Before add ADT plugin to Eclipse, you have to install the Android Development Kit (ADK) that after all is which will let us to develop apps for Android phones.
ADT offers to us the possibility of creating the user interface of our application in a visual way, it means, dragging and dropping components to an artificial mobile screen. ADT also offers a simulator to taste our applications before install them in a real mobile phone, although I have to say this simulator could be better, because is very slow.

## 3.3.4 Classes

Once we have clear the platforms that we are going to use to develop the client and the server site, and the IDE in which we will develop our app, now it's time to describe all the methods which compose each class of our app, either the client or the server site.

### Server

**ProtocolMessage**

***ProtocolMessage(int type, int sender, Vector<Integer> receivers, Calendar date, String groupName, String message):*** Through this method, we create a new *ProtocolMessage* object with the indicated parameters.

***ProtocolMessage(String message):*** Through this method, we create a new *ProtocolMessage* object from the String message.

***getType():*** Using this method, we get the type of message.

***getSender():*** Through this method we get the sender of the message.

***getNumReceivers():*** Using this method we get the number of clients who are going to receive the message.

***getReceivers():*** Through this method we get the number of all the receivers of the message.

***getDate():*** Through this method we get the date in which the message was sent.

***getGroupName():*** Through this method we get the name of the group for whom is directed the message.

***getMessage():*** Using this method we get the content of the message.

***messToString():*** Through this method we convert a *ProtocolMessage* object into a *JSON String* object.

***toStringVector():*** Using this method we create a message for each receiver and store all the messages into a vector. Each message only has one receiver.

**ClientThread**

***ClientThread(HashMap<Integer, ClientThread> mapClients, Socket client, MessagesDB messagesDB, UsersDB usersDB, GroupsDB groupsDB):*** Through this method we create a *ClientThread* object with the indicated parameters which is going to manage the connection between a client and the server.

***closeConnection():*** Through this method we end the connection with a client.

***sendMessage(String message):*** Through this method we send a message to a client.

***notifyUsers(Vector<Integer> users):*** By means of this method, we send a message to all the clients who are in the vector *users*. If some of those clients are not connected at this time, then the server stores the message to send them later. This method is used to notify the users that a group has been created, deleted…

***sendSimpleMessage():*** Through this method we send a message to all the users who are included in the field *receivers* of the message.

***sendContacts():*** This method is used by the server to send to a client all those contacts that he has and are registered in the server.

***createGroup():*** This method creates a new group and notifies all the members telling them that the group has been created.

***deleteGroup():*** This method delete an existing group and notifies his members.

***sendMessageToGroup():*** By means of this method, we send a message to all the members of the group.

***sendMessageToCommonFriends():*** Through this method, the server sends a message to all those contacts that the sender has in common with the receiver.

**MessagesDB**

***MessagesDB():*** This method opens the database of the server and creates the table m*essages* if not exists into the database to store the messages from users.

***addMessage(int receiver, String message):*** Adds a new message to the table *messages* with the specified parameters.

***getMessagesReceiver(int receiver):*** Through this method, we obtain all the messages which are directed to the specified receiver and which are stored in the table *messages.*

***closeDB():*** closes the server database.

**UsersDB**

***UsersDB():*** This method opens the database of the server if it's not opened and creates the table *users* if not exists into the database to store the information relative to the users.

***addUpdateUser(int phoneNumber, Vector<Integer> contacts):*** Through this method we add a new user with his contacts to the table *users* or if that client already exists, then the server updates his contacts.

***areRegistered(Vector<Integer> contacts):*** By means of this method, we can know what contacts of the vector *contacts* are registered in the table *users*.

***getContactsUser():*** Through this method, the server gets all the contacts of a specific user.

***deleteAllUsers():*** deletes all the users of the table *users.*

***closeDB():*** closes the server database in which the table *users* is stored.

**GroupsDB**

***GroupsDB():*** This method opens the database of the server if it's not opened and creates the table *groups* if not exists into the database to store the information relative to the groups that the users have created.

***membersToString(Vector<Integer> members):*** Using this method the server transforms a vector of I*ntegers* into a *String* with all the numbers concatenated.

***createGroup(String name, Vector<Integer> members):*** Adds a new group to the table *groups* with the specified parameters.

***deleteGroup(String groupName):*** deletes de specified group from the table *groups.*

***getMembersGroup(String name):*** Through this method the server obtains the members of the group given by the parameter *name*.

***addMemberToGroup(String groupName,Integer member):*** adds the client *member* to the group given by the parameter *groupName.*

***deleteMemberFromGroup(String groupName, Integer member):*** deletes the client *member* from the specified group.

## Client

**ChatsDB**

***open():*** opens the client database and creates the table *chatsDB* if not exists.

***createChat(String name, long phoneNumber):*** adds a new chat to the table *chatsDB* with the specified parameters.

***deleteChat(long id):*** deletes the chat which has the specified *id* from the table *chatsDB.*

***deleteChatByName(String name):*** deletes the chat which has the name specified by *name* from the table *chatsDB.*

***getAllChats():*** Through this method the server obtains all the chats stored in the table *chatsDB.*

***getChat(long id):*** By means of this method the server gets the chat which has the id *id.*

***getChatByName(String name):*** Through this method the server obtains the chat which has the name specified by *name.*

***getIdChat(String name):*** returns the *id* of the chat that has the name specified by *name.*

***getIdChatBySender(long number):*** returns the *id* of the chat whose sender has the phone specified by *number.*

***getChatName(long idChat):*** returns the name of the chat which has the id specified by *idChat.*

***getSender(long idChat):*** returns the sender number of the chat which has the id specified by *idChat.*

***updateChat(long id, String lastMessage, Calendar calendar):*** updates the message and the date which are going to appear in the description of the chat whose id is *id*.

**ChatsTab**

***openDatabases():*** opens the client databases which are going to be used in this class.

***closeDatabases():*** closes all the databases which have been opened.

***connectToService():*** binds the *ChatsTab* activity to the *MessagingAppService* service.

***disconnectFromService():*** unbinds the *ChatsTab* activity from the *MessagingAppService* service.

***onCreateOptionsMenu(Menu menu):*** creates an option menu with the option to create a new group chat.

***onMenuItemSelected(int featureId, MenuItem item):*** when the user clicks on the option to create a new group chat, this method launches the activity *CreateGroup*.

***onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo):*** creates a context menu with the option to delete a specific chat.

***onContextItemSelected(MenuItem item):*** deletes the selected chat from the database.

***onActivityResult(int requestCode, int resultCode, Intent data):*** when the user creates a group chat and selects the members, this method create the corresponding chat in the database and sends a message to the server.

***fillData():*** this method populates the menu with the existing chats.

***onListItemClick(ListView l, View v, int position, long id):*** when the user clicks on a chat, then this method launches the *MessagesChats* activity.

***onClick(View v):*** when the user clicks on the option to create a new chat, then this method launches the *ContactsTab* activity to pick a contact

***handleMessage(Message msg):*** through this method the *ChatsTab* activity receives the messages from the *MessagingAppService* service.

***onServiceConnected(ComponentName name, IBinder service):*** through this method we tell to the *MessagingAppService* service that the activity which has been connected is the *ChatsTab* activity.

**ContactsDB**

***open():***opens the client database and creates the table *contactsDB* if not exists.

***close():*** closes the database if it had been opened.

***createContact(int phoneNumber, String name):*** adds a new contact to the table *contactsDB* with the specified parameters.

***addContacts(Vector<String> names, Vector<Integer> phones):*** adds to the table *contactsDB* all the contacts who are in the vector *phones* with the names specified in the parameter *names.*

***deleteAllContacts():*** deletes all the contacts who are stored in the table *contactsDB.*

***deleteContact(long phoneNumber):*** deletes the contact who has the number *phoneNumber* from the table *contactsDB.*

***getAllContacts():*** returns all the contacts who are stored in the table *contactsDB.*

***getContact(long phoneNumber):*** returns the information of the contact whose phone number is *phoneNumber*.

***getContactName(long phoneNumber):*** returns the name of the contact whose phone number is *phoneNumber*.

***getContactsByName(String name):*** returns all the contacts whose names match with the patron specified by *name.*

**ContactsTab**

***onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo):*** create a context menu with the option to send a message to common friends.

***onContextItemSelected(MenuItem item):*** when the user clicks on the option to send a message to common friends, then this method launches the *MesssagesChat* activity.

***connectToService():*** binds the *ContactsTab* activity to the *MessagingAppService* service.

***disconnectFromService():*** unbinds the *ContactsTab* activity from the *MessagingAppService* service.

***getContactNames(Vector<Integer> numbers):*** by means of this method, the client obtains the names of the phone numbers specified in the vector *numbers*.

***fillData():*** this method populates the menu with the user contacts.

***onListItemClick(ListView l, View v, int position, long id):*** when the user clicks on a contact, then this method launches the *MessagesChats* activity.

***onClickSearch(View v):*** Through this method the user can search a specific contact.

***handleMessage(Message msg):*** through this method the *ContactsTab* activity receives the messages from the *MessagingAppService* service.

***onServiceConnected(ComponentName arg0, IBinder service):*** By means of this method we tell to the *MessagingAppService* service that the activity which has been connected is the *ContactsTab* activity.

**CreateGroup**

***openDatabases():*** opens the client databases which are going to be used in this class.

***closeDatabases():*** closes all the databases which had been opened.

***fillData():*** this method populates the list of the menu with the user contacts.

***onClick(View v):*** when the user clicks on the button to confirm the creation of new group, this method adds a group to the database with the specified name and members.

**GroupMembers**

***fillData():*** this method populates the list of the menu with the members of the selected group.

***onClick(View v):*** when the user clicks on the button to confirm the selected members, then this method launches the *MessagesChat* activity to send them a message.

**GroupsDB**

***open():*** opens the client database and creates the table *groupsDB* if not exists.

***close():*** closes the database if it had been opened.

***createGroup(String name):*** adds a new group with the name specified by *name* in the table *groupsDB*.

***deleteGroup(long id):*** deletes the group which has the identifier *id.*

***getAllGroups():*** returns all the groups stored in the table *groupsDB*.

***getGroup(long rowID):*** returns the information of the group whose id is *rowID*.

***getGroupsByName(String name):*** returns all the groups whose names match with the specified by the pattern *name*.

**GroupsTab**
***onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo):*** Creates a context menu with the option to delete a group.

***onContextItemSelected(MenuItem item):*** deletes the group selected by the user.

***openDatabases():*** opens the client databases which are going to be used in this class.

***closeDatabases():*** closes all the databases which had been opened.

***fillData():*** this method populates the list of the menu with the existing groups.

***onListItemClick(ListView l, View v, int position, long id):*** when the user clicks on a group, then this method launches the *GroupMembers* activity.

***onClick(View v):*** this method is used to search a group or create a new one by launching the *GroupTab* activity or *CreateGroup* activity.

**MembersGroupDB**
***open():*** opens the client database and creates the table *membersGroupDB* if not exists.

***close():*** closes the database if it had been opened.

***addMember(long idGroup, String name, long number):*** adds a new member to the group whose id is *idGroup* with the specified parameters.

***deleteMemberFromGroup(long id):*** deletes the member whose identifier is *id* from the group he belongs.

***deleteAllMembersGroup(long idGroup):*** deletes all the members who belong to the group with the id specified by *idGroup*.

***getMembersGroup(long idGroup):*** returns all the contacts who belong to the group whose identifier is *idGroup*.

***getMember(long id):*** returns the information of the member whose identifier is *id*.

**MessagesChat**
***openDatabases():*** opens the client databases which are going to be used in this class.

***closeDatabases():*** closes all the databases which had been opened.

***onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo):*** Creates a context menu with the option to delete a message from a chat.

***onContextItemSelected(MenuItem item):*** deletes the selected message from the chat.

***connectToService():*** binds the *MessagesChat* activity to the *MessagingAppService* service.

***disconnectFromService():*** unbinds the *MessagesChat* activity from the *MessagingAppService* service.

***fillData():*** this method populates the list of the menu with the messages of the chat.

***onClick(View v):*** when the user clicks on the button to send a message, then this method sends the corresponding message to the server.

***handleMessage(Message msg):*** through this method the *MessagesChat* activity receives the messages from the *MessagingAppService* service.

***onServiceConnected(ComponentName name, IBinder service):*** through this method we tell to the *MessagingAppService* service that the activity which has been connected is the *MessagesChat* activity.

**MessagesDB**
***open():*** opens the client database and creates the table *messagesDB* if not exists.

***close():*** closes the database if it had been opened.

***addMessageToChat(long idChat, long receiver, String senderName, int senderPhone, String message, Calendar calendar):*** adds a new message to the table *messagesDB* with the specified parameters.

***deleteMessage(long id):*** deletes the message whose identifier is *id* from the table *messagesDB*.

***deleteMessagesChat(long idChat):*** deletes all the messages from the chat whose id is *idChat*.

***getMessagesChat(long idChat):*** returns all the messages from the chat whose identifier is *idChat*.

***getMessagesReceiver(long idReceiver):*** returns all the messages where the number of the receiver is *idReceiver*.

**MessagingAppService**
***connectToServer():*** By means of this method the client connects to the server sending him his contacts.

***getContacts():*** returns all the contacts that the user has stored in his phone.

***sendMessage(String message):*** sends the specified message to the server.

***showNotification(String message):*** when the user receives a message, this method shows a notification in the notification bar of the phone.

***handleMessage(Message msg):*** through this method the *MessagingAppService* service receives the messages from the activities which are connected to him.

***isRunning():*** returns true if the service is running.

***run():*** manages the receipt of messages from the server.

## 3.4 Testing

The best way to test an application is through an example of the app running. To explain how MessagingApp works, I will use some screenshots for a better understanding and by this way I will explain all the functionalities.

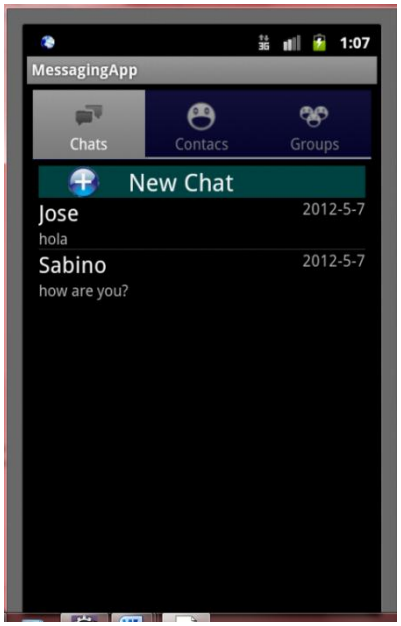
### 3.4.1 MessagingApp

The first time you launch the app it gets your phone number and registers it in the data base of the server. The app synchronizes with your phone contacts, by this way all your contacts who have installed MessagingApp will appear in the contacts of the application. Every time you start the app it checks if you have new contacts. All this processes is transparent to the user.

As I said before, my app simplifies the way to send messages and create groups as we can see in the next pictures:

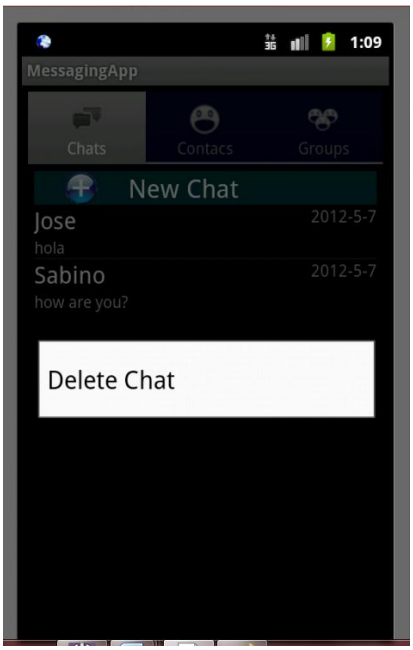
#### 3.4.1.1 Chats Menu

When you launch the app, the first thing you see is the last contacts/groups with which you have talked so that if you want to continue talking to them, you only have to click on the contact/group concerned.



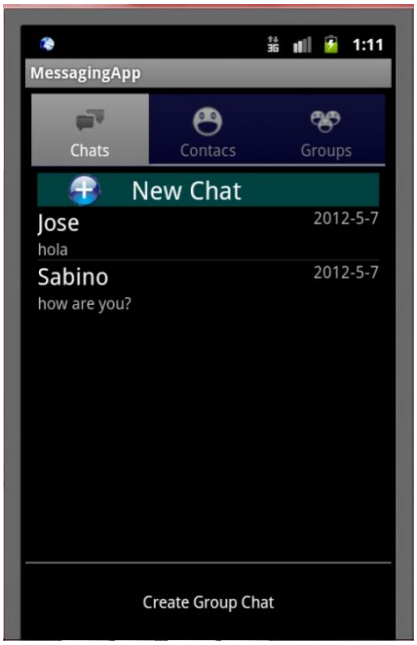

As we can see, we can create a new chat by clicking on the entry dedicated to this task. Once we have clicked, it will appear a menu with our contacts (right image) and here we only have to select one of them and start to talk.

If we want to delete a chat, we only have to keep pressed the chat concerned and it will appear a pop up menu with the option to delete the chat.



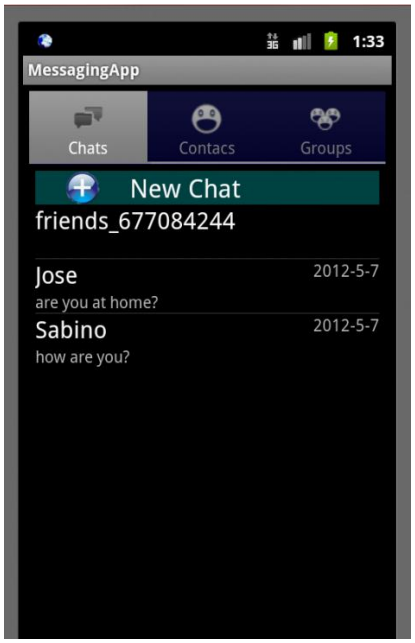

*(figure 4.4.1.1.3)*

The user can also create a group chat. To do this, he only has to click on the menu button and then will appear the option "*Create Group Chat*". After this, the client only has to choose the name of the group and select the members.




*(figure 4.4.1.1.4)*

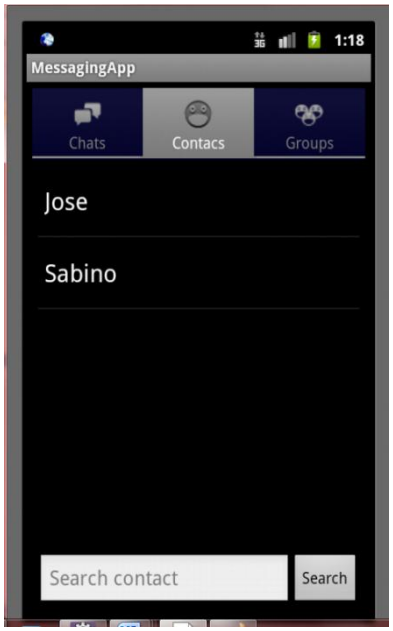

*(figure 4.4.1.1.5)*




*(figure 4.4.1.1.6)*

### 3.4.1.2 Menu of Contacts

In this menu we will see the list of contacts that we have in our application.

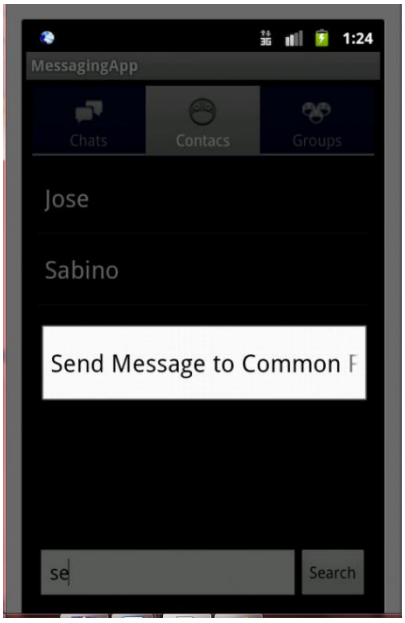



If we click on a specific contact, then we will start a conversation with him.

As we can see on the bottom of the first image, we can search a contact, to do this we only have to write the name (or part thereof) of the contact we want to find and then click on the *search* button. The app will show all the contacts whose names match with the inserted one.


*(figure 4.4.1.2.1)* *(figure 4.4.1.2.1)*

If we keep pressed a specific contact, then it will appear a pop up menu with the option "*Send Message to Common Friends*" that obviously it's used to send a message to all those contacts that the user has in common with the selected contact.

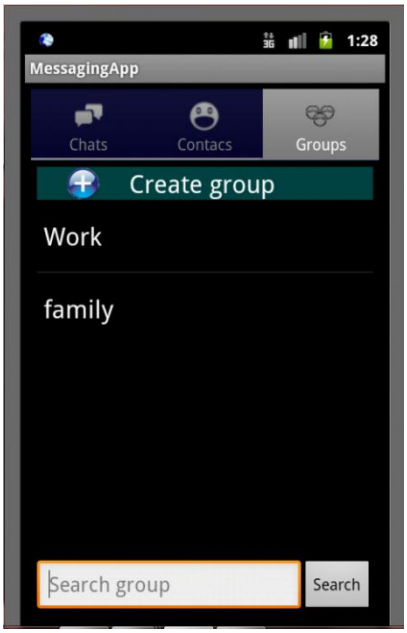




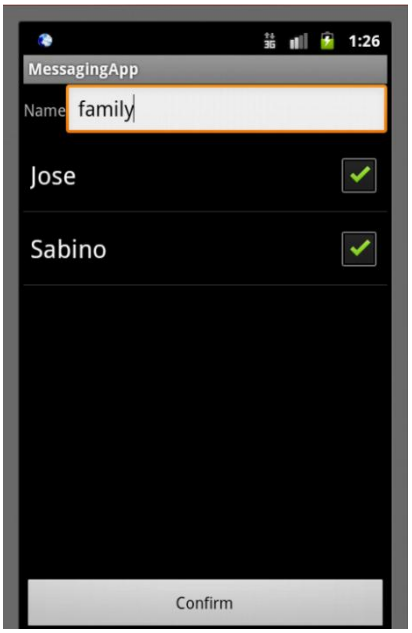
*(figure 4.4.1.2.3)* *(figure 4.4.1.2.4)*

### 3.4.1.3 Menu of Groups

In this menu we will see the list of groups that we have created in our application. We can create a new group by clicking on the entry "*Create Group*", after this it will appear a new menu where we have to choose a name for the group and select the members.

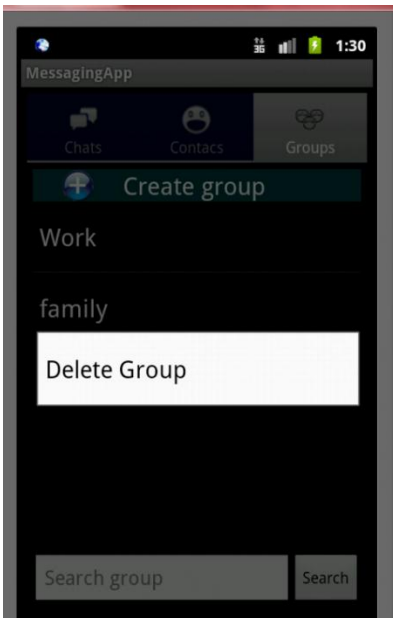



*(figure 4.4.1.3.1)*




*(figure 4.4.1.3.2)*

In the case that we want to delete a group, we only have to keep pressed the group concerned and it will appear a pop up menu with the option to delete the selected group (figure x).

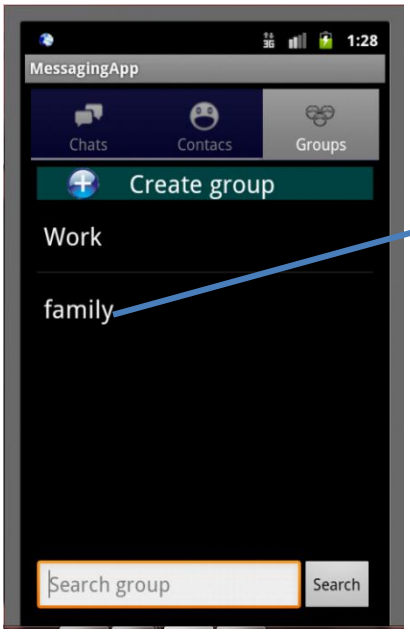



*(figure 4.4.1.3.3)*

We can also search groups by name, to do this we only have to type the name (or part thereof) of the group we want to find and then click on the *search* button. The app will show all the groups whose names match with the inserted one.


*(figure 4.4.1.3.4)*

If we click on a group, then we will see the members who compose the group and we will be able to select some of them (or all) to send them a message.



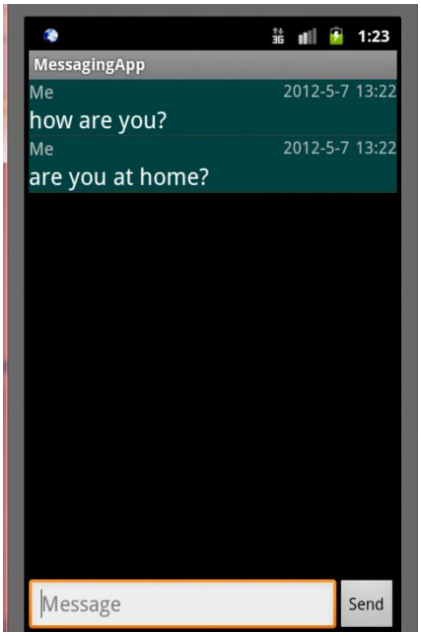

*(figure 4.4.1.3.5)*


*(figure 4.4.1.3.6)*

### 3.4.1.4 Sending Messages

Now I'm going to explain the most important part, sending messages. To send messages the user only has to select a chat, contact or group and then will appear the menu to write a message. If the user has chosen an only contact or chat, then it will also appear the previous conversations with that contact. On the other hand, if the user has

selected a group, then he can choose to send the message to all members of the group or only to some of them.



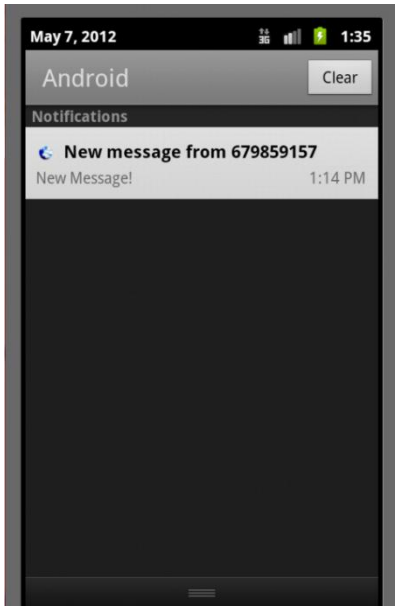

*(figure 4.4.1.4.1)*


*(figure 4.4.1.4.2)*

### 3.4.1.5 Notifications

When you receive a message, there may be two situations: the first one is that you have the app opened at this moment and the second one is that the app is running in background. Depending on the situation, the app will proceed in one way or another.

If the app is running in background and you receive a message, then in the notification bar will appear the icon of MessagingApp indicating that you have a new message. If the user clicks on this notification, then the app will be opened.




*(figure 4.4.1.5.1)*

# 4 Conclusion

The aim of this document has been showing the reader the process I have followed to develop MessagingApp. In a world where most people need to be connected with everybody around them, an app which allows them to achieve this objective for free has a high rate to get success. I know that currently there are more applications with this functionality and that these have been in the market since long time ago, but I think that I have done something to differentiate my app from the rest.

In fact, the success of this kind of applications is such that the mobile phone operators are offering to all those clients who have a data plan, the possibility of sending SMSs for free. Actually, in Spain the main mobile phone operators have made an agreement to develop their own application to compete against the current ones, the name of this app is "*Joyn*" and, in theory, has more functionalities than *Whatsapp, Viber*...and would be available for all operating systems, instead of only for most popular like the rest of applications.

Nowadays, most people with a smartphone has an app to send messages via the Internet like *Whatsapp*, *Viber*, *GroupMe* or *iMessage*, so, why should any of them change their current application by mine? To achieve this objective I have to offer them something that the rest of apps don't have, something new.

# 5 References

[1]Griffith, C. (2012) Chapter 3 - Introduction to Mobile Development. In: Christopher Griffith. Real-World Flash Game Development (Second Edition). Boston. Focal Press. Pages e29-e52.

[2]Hoog, A. (2011) Chapter 3 - Android software development kit and android debug bridge. In: Andrew Hoog. Android Forensics. Boston. Syngress. Pages 65-103.

[3]Hoog, A. and Strzempka, K. (2011) Chapter 1 – Overview. In: Andrew Hoog and Katie Strzempka. iPhone and iOS Forensics. Boston. Syngress. Pages 1-33.

[4]Iuppa, N. and Borst, T. (2010) Chapter Nineteen - Development and Delivery Platforms. In: Nick Iuppa and Terry Borst. End-to-End Game Development, Boston. Focal Press. Pages 227-256.

[5]AppBrain (2011) Number of available Android applications [online], available from: < http://www.appbrain.com/stats/number-of-android-apps > [accessed 22 November 2011].

[6]dotMobi (2011) Global mobile statistics 2011: all quality mobile marketing research, mobile Web stats, subscribers, ad revenue, usage, trends… [online], available from: < http://mobithinking.com/mobile-marketing-tools/latest-mobile-stats > [accessed 21 November 2011].

[7]GroupMe Inc. (2011) GroupMe [online], available from: < http://groupme.com/ > [accessed 23 November 2011].

[8] PCWorld Communications, Inc. (2011) At 40,000 Apps, Windows Phone Marketplace Still Lags | PCWorld [online], available from: <http://www.pcworld.com/article/244401/at_40000_apps_windows_phone_marketplace_still_lags.html> [accessed 22 November 2011].

[9]ReadWriteWeb (2011) Apple Reaches New Milestone: 500,000 iOS Apps (Infographic) [online], available from: <http://www.readwriteweb.com/archives/5000000_ios_apps_visualized.php > [accessed 22 November 2011].

[10]Viber Media Inc. (2011) Viber [online], available from: < http://www.viber.com/ > [accessed 23 November 2011].

[11]Weblogs SL (2011) WhatsApp vs. GroupMe: las dos apuestas más fuertes de mensajería instantánea en el móvil, frente a frente [online], available from:

< http://www.genbeta.com/mensajeria-instantanea/whatsapp-vs-groupme-las-dos-apuestas-mas-fuertes-de-mensajeria-instantanea-en-el-movil-frente-a-frente> [accessed 17 November 2011].

[12]WhatsApp Inc. (2011) WhatsApp FAQ [online], available from:
< http://www.whatsapp.com/faq/ > [accessed 23 November 2011].

[13]How SMS messaging is changing the world. (2011) [online image] available from:
< http://edudemic.com/wp-content/uploads/2011/11/planettext.jpg> planettext.jpg, [accessed 19 November 2011].

[14]The Android Story. (2011) [online image] available from:
< http://holaandroid.com/wp-content/uploads/2011/08/The-Andriod-Story.png > The-Andriod-Story.png, [accessed 20 November 2011].

# 1 Appendix

## 1.1 Code

### 1.1.1 Server

#### 1.1.1.1 ServerMA

```
public class ServerMA {
	private static final int PORT     = 11325;
	private static final int MAX_CONN = 50;
	private ServerSocket                  mServer;
	private Socket                        mClient;
	private boolean                       mExit;
	private HashMap<Integer, ClientThread> mMapClients = new HashMap<Integer, ClientThread>();
	private MessagesDB                                   mMessagesDB;
	private GroupsDB                                     mGroupsDB;
	private UsersDB                                              mUsersDB;

	public ServerMA() {
		try {
			mServer = new ServerSocket(PORT, MAX_CONN);
			mExit   = false;
			mMessagesDB = new MessagesDB();
			mUsersDB    = new UsersDB();
			mGroupsDB   = new GroupsDB();
			while(!mExit) {
				mClient = mServer.accept();
				new ClientThread(mMapClients, mClient, mMessagesDB, mUsersDB,
mGroupsDB);
			}
			mServer.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	private void closeConnection() {
		mExit = true;
	}

	public static void main(String[] args) {
		ServerMA server = new ServerMA();
	}
}
```

#### 1.1.1.2 ClientThread

```
public class ClientThread implements Runnable {
	private Integer                                                    mPhoneNumber;
	private HashMap<Integer, ClientThread> mMapClients;
	private Socket                         mClient;
	private BufferedReader                 mIn;
	private PrintWriter                                  mOut;
	private boolean                        mExit;
	private ProtocolMessage                                      mMessage;
	private MessagesDB                                           mMessagesDB;
	private GroupsDB                                             mGroupsDB;
	private UsersDB                                                      mUsersDB;

	public ClientThread(HashMap<Integer, ClientThread> mapClients, Socket client, MessagesDB
messagesDB, UsersDB usersDB, GroupsDB groupsDB) {
		try {
			mMapClients = mapClients;
			mClient     = client;
			mMessagesDB = messagesDB;
			mUsersDB    = usersDB;
			mGroupsDB   = groupsDB;
			mIn         = new BufferedReader(new
InputStreamReader(mClient.getInputStream()));
			mOut        = new PrintWriter(mClient.getOutputStream(), true);
			mOut.flush();
			mExit       = false;

			Thread thread = new Thread(this);
			thread.start();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	private void closeConnection() {
		mMapClients.remove(mPhoneNumber);
		mExit = true;
	}

	private void sendMessage(String message) {
		mOut.println(message);
		System.out.println("Enviando server: " + message);
	}

	private void notifyUsers(Vector<Integer> users) {
		for(int i=0; i < users.size(); i++) {
			ClientThread ct = mMapClients.get(users.get(i));
			if(ct != null)
				ct.sendMessage(mMessage.messToString());
			else
				mMessagesDB.addMessage(users.get(i),
mMessage.messToString());
		}
	}

	private void sendSimpleMessage() {
		Vector<String>  messages  = mMessage.toStringVector();
		Vector<Integer> receivers = mMessage.getReceivers();
		for(int i=0; i < mMessage.getNumReceivers(); i++) {
			ClientThread ct = mMapClients.get(receivers.get(i));
			if(ct != null)
				ct.sendMessage(messages.get(i));
			else
				mMessagesDB.addMessage(receivers.get(i), messages.get(i));
		}
	}

	private Vector<Integer> sendContacts() {
		Vector<Integer> contacts = mUsersDB.areRegistered(mMessage.getReceivers());
		ProtocolMessage message = new ProtocolMessage(ProtocolMessage.SCN,
mMessage.getSender(),contacts,Calendar.getInstance(),null,null);
		sendMessage(message.messToString());
		return contacts;
	}

	private void createGroup() {
		mGroupsDB.createGroup(mMessage.getGroupName() + "_" + mMessage.getSender(),
mMessage.getReceivers());
```

```
                    Vector<Integer> receivers = mMessage.getReceivers();
                    notifyUsers(receivers);
          }

          private void deleteGroup() {
                    Vector<Integer> members = mGroupsDB.getMembersGroup(mMessage.getGroupName());
                    mGroupsDB.deleteGroup(mMessage.getGroupName());
                    notifyUsers(members);
          }

          private void addMemberToGroup() {
                    mGroupsDB.addMemberToGroup(mMessage.getGroupName(),
mMessage.getReceivers().get(0));
                    Vector<Integer> members = mGroupsDB.getMembersGroup(mMessage.getGroupName());
                    notifyUsers(members);
          }

          private void deleteMemberFromGroup() {
                    Vector<Integer> members = mGroupsDB.getMembersGroup(mMessage.getGroupName());
                    mGroupsDB.deleteMemberFromGroup(mMessage.getGroupName(),
mMessage.getReceivers().get(0));
                    notifyUsers(members);
          }

          private void sendMessageToGroup() {
                    Vector<Integer> members = mGroupsDB.getMembersGroup(mMessage.getGroupName());
                    for(int i=0; i < mMessage.getNumReceivers(); i++)
                              members.remove(mMessage.getReceivers().get(i));
                    notifyUsers(members);
          }

          private void sendMessageToCommonFriends() {
                    HashMap<Integer, Void> user1 = mUsersDB.getContactsUser(mPhoneNumber);
                    Set<Integer>           user2 =
mUsersDB.getContactsUser(mMessage.getReceivers().get(0)).keySet();
                    Iterator<Integer>      it    = user2.iterator();
                    Vector<Integer>        commonFriends = new Vector<Integer>();
                    while(it.hasNext()) {
                              Integer phoneNumber = it.next();
                              if(user1.containsKey(phoneNumber));
                                        commonFriends.add(phoneNumber);
                    }
                    notifyUsers(commonFriends);
          }

          public void run() {
                    try {
                              while(!mExit) {
                                        String message = mIn.readLine();
                                        if(message != null) {
                                                  System.out.println("Recibiendo server: " +
message);
                                                  mMessage = new ProtocolMessage(message);
                                                  switch(mMessage.getType()) {
                                                            case ProtocolMessage.CNT: mPhoneNumber =
mMessage.getSender();

mMapClients.put(mPhoneNumber, this);

break;
                                                            case ProtocolMessage.DSC:
closeConnection(); break;
                                                            case ProtocolMessage.SCN:
mUsersDB.addUpdateUser(mPhoneNumber.intValue(), sendContacts()); break;
                                                            case ProtocolMessage.SSM:
sendSimpleMessage(); break;
                                                            case ProtocolMessage.CGR: createGroup();
break;
                                                            case ProtocolMessage.DGR: deleteGroup();
break;
                                                            case ProtocolMessage.ACG:
addMemberToGroup(); break;
                                                            case ProtocolMessage.DCG:
deleteMemberFromGroup(); break;
                                                            case ProtocolMessage.SMG:
sendMessageToGroup(); break;
                                                            case ProtocolMessage.SMC:
sendMessageToCommonFriends(); break;
                                                  }
                                                  if(mMessage.getType() != ProtocolMessage.DSC) {
                                                            Vector<String> messages =
mMessagesDB.getMessagesReceiver(mPhoneNumber);
                                                            for(int i=0; i < messages.size(); i++)
                                                                      sendMessage(messages.get(i));
                                                  }
                                        }
                              }
                              mIn.close();
                              mOut.close();
                              mClient.close();
                              mMapClients.remove(mPhoneNumber);
                    } catch (IOException e) {
                              e.printStackTrace();
                    }
          }

}
```

## 1.1.1.3 UsersDB

```
public class UsersDB {
          private static final String mTableUsers = "CREATE TABLE IF NOT EXISTS users ( " +

            "_id  integer PRIMARY KEY ," +

            "contacts      text);";
          private static final String mIDKey       = "_id";
          private static final String mContactsKey = "contacts";
          private Connection mConn;

          public UsersDB() {
                    try {
                              Class.forName("org.sqlite.JDBC");
                              mConn = DriverManager.getConnection("jdbc:sqlite:messagingApp.db");
                              Statement stat = mConn.createStatement();
                              stat.execute(mTableUsers);
                              stat.close();
                    } catch (Exception e) {
                              e.printStackTrace();
                    }
          }

          public void closeDB() {
                    try {
                              mConn.close();
                    } catch (SQLException e) {
                              e.printStackTrace();
                    }
          }

          public void addUpdateUser(int phoneNumber, Vector<Integer> contacts) {
                    StringBuilder sb = new StringBuilder();
                    sb.append("\'");
                    if(contacts.size() > 0)
                              sb.append(contacts.get(0));
                    for(int i=1; i < contacts.size(); i++)
                              sb.append('*').append(contacts.get(i));
                    sb.append("\'");
                    try {
                              String users = sb.toString();
                              Statement stat = mConn.createStatement();
```

```
                        stat.executeUpdate("INSERT OR REPLACE INTO users (" + mIDKey + ", " +
mContactsKey + " ) " +
                                                                        "VALUES ( " + phoneNumber +
", " + users + " );");
                        stat.close();
                } catch (SQLException e) {
                        e.printStackTrace();
                }

        }

        public Vector<Integer> areRegistered(Vector<Integer> contacts) {
                Vector<Integer> vUsers = new Vector<Integer>();
                try {
                        StringBuilder sb = new StringBuilder();
                        if(contacts.size() > 0) {
                                sb.append('(');
                                for(int i=0; i<(contacts.size() - 1); i++)
                                        sb.append(contacts.get(i) + ",");
                                sb.append(contacts.lastElement()).append(')');
                                Statement stat = mConn.createStatement();
                                ResultSet res  = stat.executeQuery("SELECT " + mIDKey + "
FROM users " +

                   "WHERE " + mIDKey + " IN " + sb.toString() + ";");
                                while(res.next())
                                        vUsers.add(new Integer(res.getInt(mIDKey)));
                                res.close();
                                stat.close();
                        }
                } catch (SQLException e) {
                        e.printStackTrace();
                }
                return vUsers;
        }

        public HashMap<Integer, Void> getContactsUser(int phoneNumber) {
                HashMap<Integer, Void> map = new HashMap<Integer, Void>();
                try {
                        Statement stat = mConn.createStatement();
                        ResultSet res  = stat.executeQuery("SELECT " + mContactsKey + " FROM
users " +

            "WHERE " + mIDKey + "=" + phoneNumber + ";");
                        if(res.next()) {
                                StringTokenizer st = new
StringTokenizer(res.getString(mContactsKey),"*");
                                while(st.hasMoreTokens())
                                        map.put(Integer.valueOf(st.nextToken()),null);
                        }
                        res.close();
                        stat.close();
                } catch (SQLException e) {
                        e.printStackTrace();
                }
                return map;
        }

        public void deleteAllUsers() {
                Statement stat;
                try {
                        stat = mConn.createStatement();
                        stat.executeUpdate("DELETE FROM users;");
                } catch (SQLException e) {
                        e.printStackTrace();
                }
        }

        public static void main(String[] args) {
                /*UsersDB usersDB = new UsersDB();
                Vector<Integer> receivers = new Vector<Integer>();
                usersDB.deleteAllUsers();
                usersDB.addUpdateUser(677084244, receivers);
                HashMap<Integer, Void> users = usersDB.getContactsUser(677084244);
                Set<Integer> u = users.keySet();
                Iterator<Integer> it = u.iterator();
                while(it.hasNext()) {
                        System.out.println("hola");
                        System.out.println(it.next());
                }*/
        }
}
```

## 1.1.1.4 MessagesDB

```
public class MessagesDB {
        private static final String mTableMessages = "CREATE TABLE IF NOT EXISTS messages ( "

           + "_id        integer PRIMARY KEY autoincrement, "

           + "receiver   integer NOT NULL, "

           + "message    text);";

        private static final String mIDKey        = "_id";
        private static final String mReceiverKey  = "receiver";
        private static final String mTypeKey      = "type";
        private static final String mSenderKey    = "sender";
        private static final String mReceiversKey = "receivers";
        private static final String mDateKey      = "date";
        private static final String mGroupNameKey = "group_name";
        private static final String mMessageKey   = "message";
        private Connection mConn;

        public MessagesDB() {
                try {
                        Class.forName("org.sqlite.JDBC");
                        mConn = DriverManager.getConnection("jdbc:sqlite:messagingApp.db");
                        Statement stat = mConn.createStatement();
                        stat.execute(mTableMessages);
                        stat.close();
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }

        public void addMessage(int receiver, String message) {
                try {
                        PreparedStatement pstmt = mConn.prepareStatement("INSERT INTO messages
(" + mReceiverKey + "," + mMessageKey + ") " +

                                                "VALUES ( ? , ?)");
                        pstmt.setInt(1, receiver);
                        pstmt.setString(2, message);
                        pstmt.executeUpdate();
                        pstmt.close();
                } catch (SQLException e) {
                        e.printStackTrace();
                }
        }

        public Vector<String> getMessagesReceiver(int receiver) {
                Vector<String> vMessages = new Vector<String>();
                try {
                        Statement stat = mConn.createStatement();
                        ResultSet res  = stat.executeQuery("SELECT * FROM messages " +
                                                                        "WHERE " + mReceiverKey
+ "=" + receiver + ";");
                        while(res.next())
                                vMessages.add(res.getString(mMessageKey));
                        stat.executeUpdate("DELETE FROM messages WHERE " + mReceiverKey + "=" +
receiver + ";");
```

```
                        res.close();
                        stat.close();
                } catch (SQLException e) {
                        e.printStackTrace();
                }
                return vMessages;
        }

        public void closeDB() {
                try {
                        mConn.close();
                } catch (SQLException e) {
                        e.printStackTrace();
                }
        }

        public static void main(String[] args) {
                MessagesDB mess = new MessagesDB();
                ProtocolMessage message = new ProtocolMessage(0, 677084244, new
Vector<Integer>(), Calendar.getInstance(), "hola", "me llamo jose luis");
                mess.addMessage(669238051, message.messToString());
                Vector<String> v = mess.getMessagesReceiver(669238051);
                for(int i = 0; i < v.size(); i++)
                        System.out.println("Message " + i + ": " + v.get(i));
        }
}
```

## 1.1.1.5 GroupsDB

```
public class GroupsDB {
        private static final String mGroupsTable = "CREATE TABLE IF NOT EXISTS groups ( "

         + "_id          integer PRIMARY KEY autoincrement, "

         + "group_name   text    NOT NULL, "

         + "members      text    NOT NULL);";
        private static final String mGroupNameKey      = "group_name";
        private static final String mMembersKey = "members";
        private Connection mConn;

        public GroupsDB() {
                try {
                        Class.forName("org.sqlite.JDBC");
                        mConn = DriverManager.getConnection("jdbc:sqlite:messagingApp.db");
                        Statement stat = mConn.createStatement();
                        stat.execute(mGroupsTable);
                        stat.close();
                } catch(Exception e) {
                        e.printStackTrace();
                }
        }

        private String membersToString(Vector<Integer> members) {
                StringBuilder sb = new StringBuilder();
                sb.append(members.get(0));
                for(int i=1; i<members.size(); i++)
                        sb.append('#').append(members.get(i));
                return sb.toString();
        }

        public void createGroup(String name, Vector<Integer> members) {
                try {
                        PreparedStatement pstmt = mConn.prepareStatement("INSERT INTO groups ("
+ mGroupNameKey + "," + mMembersKey + ") " +

                                                "VALUES (?, ?);");
                        pstmt.setString(1, name);
                        pstmt.setString(2, membersToString(members));
                        pstmt.executeUpdate();
                        pstmt.close();
                } catch (SQLException e) {
                        e.printStackTrace();
                }
        }

        public void deleteGroup(String groupName) {
                try {
                        Statement stat = mConn.createStatement();
                        stat.executeUpdate("DELETE FROM groups WHERE " + mGroupNameKey + "=" +
groupName + ";");
                        stat.close();
                } catch (SQLException e) {
                        e.printStackTrace();
                }
        }

        public Vector<Integer> getMembersGroup(String name) {
                Vector<Integer> members = new Vector<Integer>();
                try {
                        PreparedStatement pstmt = mConn.prepareStatement("SELECT " +
mMembersKey + " FROM groups " +

                                                "WHERE " + mGroupNameKey + " LIKE ?;");
                        pstmt.setString(1, name);
                        ResultSet res  = pstmt.executeQuery();
                        res.next();
                        StringTokenizer st = new StringTokenizer(res.getString(mMembersKey),
"#");
                        while(st.hasMoreTokens())
                                members.add(Integer.valueOf(st.nextToken()));
                        res.close();
                        pstmt.close();
                } catch (SQLException e) {
                        e.printStackTrace();
                }
                return members;
        }

        public void addMemberToGroup(String groupName,Integer member) {
                try {
                        if(!getMembersGroup(groupName).contains(member)) {
                                Statement stat = mConn.createStatement();
                                ResultSet res  = stat.executeQuery("SELECT " + mMembersKey +
" FROM groups " +

                "WHERE " + mGroupNameKey + "=" + groupName + ";");
                                res.next();
                                stat.executeUpdate("UPDATE groups " +
                                                                        "SET " +
mMembersKey + "=" + res.getString(mMembersKey) + "#" + member + " " +
                                                                "WHERE " + mGroupNameKey + "=" +
groupName + ";");
                                res.close();
                                stat.close();
                        }
                } catch (SQLException e) {
                        e.printStackTrace();
                }
        }

        public void deleteMemberFromGroup(String groupName, Integer member) {
                Vector<Integer> vMembers = getMembersGroup(groupName);
                vMembers.remove(member);
                try {
                        Statement stat = mConn.createStatement();
                        stat.executeUpdate("UPDATE groups " +
                                                "SET " + mMembersKey + "=" +
membersToString(vMembers) + " " +
                                                "WHERE " + mGroupNameKey + "=" + groupName +
";");
                        stat.close();
                } catch (SQLException e) {
```

```
                        e.printStackTrace();
                }
        }
}
```

### 1.1.1.6 ProtocolMessage

```
public class ProtocolMessage {
        public static final int CNT = 1;
        public static final int DSC = 2;
        public static final int SSM = 3;
        public static final int SCN = 4;
        public static final int CGR = 5;
        public static final int DGR = 6;
        public static final int ACG = 7;
        public static final int DCG = 8;
        public static final int SMG = 9;
        public static final int SMC = 10;
        private int             mType;
        private int             mSender;
        private Vector<Integer> mReceivers = new Vector<Integer>();
        private Calendar        mDate;
        private String          mGroupName;
        private String          mMessage;

        public ProtocolMessage(int type, int sender, Vector<Integer> receivers,
                               Calendar date, String groupName, String message) {
                mType      = type;
                mSender    = sender;
                mReceivers = receivers;
                mDate      = date;
                mDate      = Calendar.getInstance();
                mGroupName = groupName;
                mMessage   = message;
        }

        public ProtocolMessage(String message) {
                System.out.println(message);
                Gson    g  = new Gson();
                ProtocolMessage m  = g.fromJson(message,ProtocolMessage.class);
                mType      = m.getType();
                mSender    = m.getSender();
                mReceivers = m.getReceivers();
                mDate      = m.getDate();
                mGroupName = m.getGroupName();
                mMessage   = m.getMessage();
        }

        public int getType() {
                return mType;
        }

        public int getSender() {
                return mSender;
        }

        public int getNumReceivers() {
                return mReceivers.size();
        }

        public Vector<Integer> getReceivers() {
                return mReceivers;
        }

        public Calendar getDate() {
                return mDate;
        }

        public String getGroupName() {
                return mGroupName;
        }

        public String getMessage() {
                return mMessage;
        }

        public void printAll() {
                System.out.println("Type: " + mType);
                System.out.println("Sender: " + mSender);
                for(int i=0; i<mReceivers.size();i++) {
                        System.out.println("Receiver " + i + ": " + mReceivers.get(i));
                }
                System.out.println("Date: " + mDate.getTime());
                if(mGroupName != null)
                        System.out.println("Group Name: " + mGroupName);
                if(mMessage != null)
                        System.out.println("Message: " + mMessage);
        }

        public String messToString() {
                Gson g = new Gson();
                return g.toJson(this);
        }

        public Vector<String> toStringVector() {
                Vector<String> vString = new Vector<String>();
                for(int i = 0; i < mReceivers.size(); i++) {
                        Vector<Integer> v = new Vector<Integer>();
                        v.add(mReceivers.get(i));
                        vString.add(new ProtocolMessage(mType, mSender,v,mDate, mGroupName,
mMessage).messToString());
                }
                return vString;
        }

        public static void main(String[] args) {
                Vector<Integer> rec = new Vector<Integer>();
                rec.add(new Integer(606817088));
                rec.add(new Integer(669238051));
                ProtocolMessage message = new
ProtocolMessage(1,677084244,rec,Calendar.getInstance(),null,null);
                ProtocolMessage m2 = new ProtocolMessage(message.messToString());
                m2.printAll();
                Vector<String> v = m2.toStringVector();
                for(int i=0; i<v.size();i++)
                        System.out.println("Message " + i + '\n' + v.get(i) + '\n');
        }
}
```

## 1.1.2 Client

### 1.1.2.1 ChatsDB

```
public class ChatsDB {
        public static final String KEY_ROWID         = "_id";
        public static final String KEY_NAME          = "name";
        public static final String KEY_NUMBER        = "number";
        public static final String KEY_LAST_MESSAGE  = "message";
        public static final String KEY_DATE          = "date";

        public static final long   NO_CHAT           = -1;

        private static final String DATABASE_NAME    = "messagingAppDB";
    private static final String DATABASE_TABLE   = "chatsDB";
        private static final String TAG              = "ChatsDB";
        private static final String DATABASE_CREATE  = "CREATE TABLE IF NOT EXISTS " +
DATABASE_TABLE + " ("

                    + KEY_ROWID         + " integer PRIMARY KEY autoincrement, "
```

```
                    + KEY_NAME          + " text    NOT NULL, "
                    + KEY_NUMBER        + " integer NOT NULL, "
                    + KEY_LAST_MESSAGE  + " text, "
                    + KEY_DATE          + " date);";
    private static final int    DATABASE_VERSION = 1;

    private final Context  mCtx;
          private DatabaseHelper mDbHelper;
          private SQLiteDatabase mDb;

          private static class DatabaseHelper extends SQLiteOpenHelper {

        DatabaseHelper(Context context) {
            super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }

        @Override
        public void onCreate(SQLiteDatabase db) {
            db.execSQL(DATABASE_CREATE);
        }

        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
            if(db.getVersion() != newVersion) {
                    Log.w(TAG, "Upgrading database from version " + oldVersion + " to "
                        + newVersion + ", which will destroy all old data");
                    db.execSQL("DROP TABLE IF EXISTS " + DATABASE_TABLE + ";");
                onCreate(db);
            }
        }

        public void createTable(SQLiteDatabase db) {
          db.execSQL(DATABASE_CREATE);
        }
    }

          public ChatsDB(Context ctx) {
                    mCtx = ctx;
          }

          public ChatsDB open() throws SQLException {
        mDbHelper = new DatabaseHelper(mCtx);
        mDb = mDbHelper.getWritableDatabase();
        mDbHelper.createTable(mDb);
        return this;
    }

          public void close() {
        mDbHelper.close();
    }

          public long createChat(String name, long phoneNumber) {
                    ContentValues initialValues = new ContentValues();
                    initialValues.put(KEY_NAME, name);
                    initialValues.put(KEY_NUMBER, phoneNumber);
                    return mDb.insert(DATABASE_TABLE, null, initialValues);
          }

          public boolean deleteChat(long id) {
                    return mDb.delete(DATABASE_TABLE, KEY_ROWID + "=" + id, null) > 0;
          }

          public boolean deleteChatByName(String name) {
                    return mDb.delete(DATABASE_TABLE, KEY_NAME + "=" + name, null) > 0;
          }

          public Cursor getAllChats() {
                    return mDb.query(DATABASE_TABLE, new String[] {KEY_ROWID, KEY_NAME, KEY_NUMBER,
KEY_LAST_MESSAGE, KEY_DATE},
                                                                        null, null, null, null, KEY_DATE);
          }

          public Cursor getChat(long id) {
                    Cursor cursor = mDb.query(DATABASE_TABLE, new String[] {KEY_ROWID, KEY_NAME,
KEY_NUMBER, KEY_LAST_MESSAGE, KEY_DATE},
                                                                                                KEY_ROWID + "=" +
id, null, null, null, null);
                    if(cursor != null)
                              cursor.moveToFirst();
                    return cursor;
          }

          public Cursor getChatByName(String name) {
                    Cursor cursor = mDb.query(DATABASE_TABLE, new String[] {KEY_ROWID, KEY_NAME,
KEY_NUMBER, KEY_LAST_MESSAGE, KEY_DATE},
                                                                                                KEY_NAME + "=" +
name, null, null, null, null);
                    if(cursor != null)
                              cursor.moveToFirst();
                    return cursor;
          }

          public long getIdChat(String name) {
                    Cursor cursor = mDb.query(DATABASE_TABLE, new String[] {KEY_ROWID, KEY_NAME,
KEY_NUMBER, KEY_LAST_MESSAGE, KEY_DATE},
                                                                        KEY_NAME + "=" + name, null, null, null, null);
                    if(cursor != null)
                              cursor.moveToFirst();
                    if(cursor.getCount() != 0)
                              return cursor.getLong(cursor.getColumnIndex(KEY_ROWID));
                    else
                              return NO_CHAT;
          }

          public long getIdChatBySender(long number) {
                    Cursor cursor = mDb.query(DATABASE_TABLE, new String[] {KEY_ROWID, KEY_NAME,
KEY_NUMBER, KEY_LAST_MESSAGE, KEY_DATE},
                                                                        KEY_NUMBER + "=" + number, null, null, null, null);
                    if(cursor != null)
                              cursor.moveToFirst();
                    if(cursor.getCount() != 0)
                              return cursor.getLong(cursor.getColumnIndex(KEY_ROWID));
                    else
                              return NO_CHAT;
          }

          public String getChatName(long idChat) {
                    Cursor cursor = mDb.query(DATABASE_TABLE, new String[] {KEY_ROWID, KEY_NAME,
KEY_NUMBER, KEY_LAST_MESSAGE, KEY_DATE},
                                                                        KEY_ROWID + "=" + idChat, null, null, null, null);
                    cursor.moveToFirst();
                    return cursor.getString(cursor.getColumnIndex(KEY_NAME));
          }

          public long getSender(long idChat) {
                    Cursor cursor = mDb.query(DATABASE_TABLE, new String[] {KEY_ROWID, KEY_NAME,
KEY_NUMBER, KEY_LAST_MESSAGE, KEY_DATE},
                                                                        KEY_ROWID + "=" + idChat, null, null, null, null);
                    cursor.moveToFirst();
                    return cursor.getLong(cursor.getColumnIndex(KEY_NUMBER));
          }

          public boolean updateChat(long id, String lastMessage, Calendar calendar) {
                    ContentValues values = new ContentValues();
                    values.put(KEY_LAST_MESSAGE, lastMessage);
                    values.put(KEY_DATE, calendar.get(Calendar.YEAR) + "-" +
(calendar.get(Calendar.MONTH) + 1) + "-" + calendar.get(Calendar.DAY_OF_MONTH));
                    return mDb.update(DATABASE_TABLE, values, KEY_ROWID + "=" + id, null) > 0;
          }
```

```
}
```

## 1.1.2.2 ChatsTab

```
public class ChatsTab extends ListActivity implements ServiceConnection, Callback {
        public  static final String ID_CHAT               = "idChat";
        public  static final int    ACTIVITY_CREATE_GROUP = Menu.FIRST;
        private static final int    DELETE_ID             = Menu.FIRST;
        private static final int    CREATE_CHAT_ID        = Menu.FIRST + 1;
        private ChatsDB    chatsDB;
        private MessagesDB messagesChatDB;
        private ContactsDB contactsDB;
        private Messenger  mService;
        private Messenger  mActivity;
        private int        mSender = 677084244;

        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
            setContentView(R.layout.chats_list);
            mActivity      = new Messenger(new Handler(this));
            chatsDB        = new ChatsDB(this);
            messagesChatDB = new MessagesDB(this);
            contactsDB     = new ContactsDB(this);
            registerForContextMenu(getListView());
         }

        private void openDatabases() {
                chatsDB.open();
                messagesChatDB.open();
                contactsDB.open();
        }

        private void closeDatabases() {
                if(chatsDB != null)
                        chatsDB.close();
                if(messagesChatDB != null)
                        messagesChatDB.close();
                if(contactsDB != null)
                        contactsDB.close();
        }

        private void connectToService() {
                Intent intent = new Intent(this,MessagingAppService.class);
                intent.putExtra(MessagingAppService.ACTIVITY_ID,
MessagingAppService.ACT_CHATSTAB);
                if(!getApplicationContext().bindService(intent, this, Service.BIND_AUTO_CREATE))
                        Log.v("ConnectService", "Impossible connect to the service");
        }

        private void disconnectFromService() {
                if(mService != null) {
                        Message message = new Message();
                        message.what    = MessagingAppService.DISCONNECT;
                        try {
                                mService.send(message);
                        } catch (RemoteException e) {
                                e.printStackTrace();
                        }
                        getApplicationContext().unbindService(this);
                }
        }

        @Override
        protected void onResume() {
                super.onResume();
                connectToService();
                openDatabases();
                fillData();
        }

        @Override
        protected void onPause() {
                disconnectFromService();
                closeDatabases();
                super.onPause();
        }

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

        @Override
        public boolean onCreateOptionsMenu(Menu menu) {
                super.onCreateOptionsMenu(menu);
                menu.add(0, CREATE_CHAT_ID, 0, R.string.create_group_chat);
                return true;
        }

        @Override
        public boolean onMenuItemSelected(int featureId, MenuItem item) {
                switch(item.getItemId()) {
        case CREATE_CHAT_ID:
                Intent intent = new Intent(this, CreateGroup.class);
                intent.putExtra(ID_CHAT, 1L);
                startActivityForResult(intent, ACTIVITY_CREATE_GROUP);
                return true;
                }
                return super.onMenuItemSelected(featureId, item);
        }

        @Override
        public boolean onContextItemSelected(MenuItem item) {
                switch(item.getItemId()) {
        case DELETE_ID:
                AdapterContextMenuInfo info = (AdapterContextMenuInfo) item.getMenuInfo();
                if(chatsDB.getSender(info.id) == 0L) {
                        String   group_name = chatsDB.getChatName(info.id);
                        String[] name_parts = group_name.split("_");
                        if(name_parts[1].equals(Integer.toString(mSender))) {
                                ProtocolMessage message = new
ProtocolMessage(ProtocolMessage.DGR, mSender,
                                                                                        new
Vector<Integer>(), Calendar.getInstance(), group_name, null);
                                Message mess = new Message();
                                mess.what    = MessagingAppService.MESSAGE;
                                mess.obj     = message.messToString();
                                try {
                                                                        mService.send(mess);
                                                                } catch (RemoteException e) {
                                                                        e.printStackTrace();
                                                                }
                        }
                }
                messagesChatDB.deleteMessagesChat(info.id);
                chatsDB.deleteChat(info.id);

                fillData();
                return true;
                }
                return super.onContextItemSelected(item);
        }

        @Override
        public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo) {
                super.onCreateContextMenu(menu, v, menuInfo);
                menu.add(0, DELETE_ID, 0, R.string.delete_chat);
        }

        @Override
        protected void onActivityResult(int requestCode, int resultCode, Intent data) {
                super.onActivityResult(requestCode, resultCode, data);
```

```
                    openDatabases();
                    String          group_name    =
data.getExtras().getString(CreateGroup.GROUP_NAME);
                    long[]          phones        =
data.getExtras().getLongArray(CreateGroup.GROUP_MEMBERS);
                    Vector<Integer> group_members = new Vector<Integer>();
                    for(int i = 0; i < phones.length; i++)
                            group_members.add(new Integer ((int)phones[i]));
                    ProtocolMessage  message = new ProtocolMessage(ProtocolMessage.CGR, mSender,
group_members, Calendar.getInstance(), group_name, null);
                    chatsDB.createChat(group_name + "_" + Integer.toString(mSender), 0L);
                    fillData();
                    Message          mess    = new Message();
                    mess.what = MessagingAppService.MESSAGE;
                    mess.obj  = message.messToString();
                    closeDatabases();
                    try {
                            mService.send(mess);
                    } catch (RemoteException e) {
                            e.printStackTrace();
                    }
            }

            protected void fillData() {
                    Cursor chatsCursor = chatsDB.getAllChats();
                    startManagingCursor(chatsCursor);
                    String[] from = new String[]{ChatsDB.KEY_NAME, ChatsDB.KEY_LAST_MESSAGE,
ChatsDB.KEY_DATE};
                    int[]    to   = {R.id.row_chat_name, R.id.last_message_chat, R.id.date_chat};
                    SimpleCursorAdapter chats = new SimpleCursorAdapter(this, R.layout.chat_row,
chatsCursor, from, to);
                    setListAdapter(chats);
            }

        @Override
        protected void onListItemClick(ListView l, View v, int position, long id) {
            super.onListItemClick(l, v, position, id);
            Intent intent = new Intent(this,MessagesChat.class);
            intent.putExtra(ID_CHAT,id);
            startActivity(intent);
         }

            public void onClick(View v) {
                    switch(v.getId()) {
                            case R.id.android_new_chat_layout:
                                    Intent intent = new Intent(this,ContactsTab.class);
                                    startActivity(intent); break;
                    }
            }

            @Override
            public boolean handleMessage(Message msg) {
                    String          message     = (String) msg.obj;
                    ProtocolMessage protMessage = new ProtocolMessage(message);
                    long idChat;
                    switch(protMessage.getType()) {
                            case ProtocolMessage.CGR:
                                    idChat = chatsDB.createChat(protMessage.getGroupName(), 0L);
                                    String receiverName, contactName =
contactsDB.getContactName(protMessage.getSender());
                                    for(int i = 0; i < protMessage.getNumReceivers(); i++) {
                                            receiverName =
contactsDB.getContactName(protMessage.getReceivers().get(i));
                                            messagesChatDB.addMessageToChat(idChat, 0L,
contactName, protMessage.getSender(),

receiverName + " has been added by " + contactName, protMessage.getDate());
                                    }
                                    Toast.makeText(this, "The group chat " +
protMessage.getGroupName() + " has been created", Toast.LENGTH_LONG).show();
                                    break;
                            case ProtocolMessage.DGR:
                                    idChat = chatsDB.getIdChat(protMessage.getGroupName());
                                    messagesChatDB.deleteMessagesChat(idChat);
                                    chatsDB.deleteChatByName(protMessage.getGroupName());
                                    Toast.makeText(this, "The group chat " +
protMessage.getGroupName() + " has been deleted", Toast.LENGTH_LONG).show();
                                    break;
                    }
                    fillData();
                    return true;
            }

            @Override
            public void onServiceConnected(ComponentName name, IBinder service) {
                    mService        = new Messenger(service);
                    Message message = new Message();
                    message.what    = MessagingAppService.CONNECT;
                    message.arg1    = MessagingAppService.ACT_CHATSTAB;
                    message.replyTo = mActivity;
                    try {
                            mService.send(message);
                    } catch (RemoteException e) {
                            e.printStackTrace();
                    }
            }

            @Override
            public void onServiceDisconnected(ComponentName name) {
                    mService = null;
            }
}
```

## 1.1.2.3 ContactsDB

```
public class ContactsDB {
            public static final String KEY_NUMBER = "_id";
            public static final String KEY_NAME   = "name";

            private static final String DATABASE_NAME    = "messagingAppDB";
        private static final String DATABASE_TABLE   = "contactsDB";
            private static final String TAG              = "ContactsDB";
            private static final String DATABASE_CREATE  = "CREATE TABLE IF NOT EXISTS " +
DATABASE_TABLE + " ("

                        + KEY_NUMBER + " integer PRIMARY KEY,"

                        + KEY_NAME   + " text NOT NULL);";
        private static final int    DATABASE_VERSION = 1;

        private final Context  mCtx;
            private DatabaseHelper mDbHelper;
            private SQLiteDatabase mDb;

            private static class DatabaseHelper extends SQLiteOpenHelper {

            DatabaseHelper(Context context) {
                super(context, DATABASE_NAME, null, DATABASE_VERSION);
            }

            @Override
            public void onCreate(SQLiteDatabase db) {
                db.execSQL(DATABASE_CREATE);
                Log.v("ContactsDB", "Creating ContactsDB...");
            }

            @Override
            public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
                if(db.getVersion() != newVersion) {
                        Log.w(TAG, "Upgrading database from version " + oldVersion + " to "
                            + newVersion + ", which will destroy all old data");
                                db.execSQL("DROP TABLE IF EXISTS " + DATABASE_TABLE + ";");
```

```
                onCreate(db);
            }
        }

        public void createTable(SQLiteDatabase db) {
          db.execSQL(DATABASE_CREATE);
        }
    }

        public ContactsDB(Context ctx) {
                mCtx = ctx;
        }

        public ContactsDB open() throws SQLException {
        mDbHelper = new DatabaseHelper(mCtx);
        mDb = mDbHelper.getWritableDatabase();
        mDbHelper.createTable(mDb);
        return this;
    }

        public void close() {
                if(mDb != null)
                        mDb.close();
                mDbHelper.close();
    }

        private boolean createContact(int phoneNumber, String name) {
                ContentValues initialValues = new  ContentValues();
                initialValues.put(KEY_NUMBER, phoneNumber);
                initialValues.put(KEY_NAME, name);
                return mDb.insert(DATABASE_TABLE, null, initialValues) != -1;
        }

        public void addContacts(Vector<String> names, Vector<Integer> phones) {
                for(int i = 0; i < names.size(); i++)
                        createContact(phones.get(i).intValue(), names.get(i));
        }

        public void deleteAllContacts() {
                mDb.delete(DATABASE_TABLE, null, null);
        }

        public boolean deleteContact(long phoneNumber) {
                return mDb.delete(DATABASE_TABLE, KEY_NUMBER + "=" + phoneNumber, null) > 0;
        }

        public boolean updateContact(String name, long phoneNumber) {
                ContentValues values = new ContentValues();
                values.put(KEY_NAME, name);
                return mDb.update(DATABASE_TABLE, values, KEY_NUMBER + "=" + phoneNumber, null) >
0;
        }

        public Cursor getAllContacts() {
                return mDb.query(DATABASE_TABLE, new String[] {KEY_NAME, KEY_NUMBER}, null, null,
null, null, KEY_NAME);
        }

        public Cursor getContact(long phoneNumber) {
                Cursor cursor = mDb.query(DATABASE_TABLE, new String[] {KEY_NAME, KEY_NUMBER},
KEY_NUMBER + "=" + phoneNumber, null, null, null, null);
                if(cursor != null)
                        cursor.moveToFirst();
                return cursor;
        }

        public String getContactName(long phoneNumber) {
                Cursor cursor = mDb.query(DATABASE_TABLE, new String[] {KEY_NAME, KEY_NUMBER},
KEY_NUMBER + "=" + phoneNumber, null, null, null, null);
                if(cursor != null)
                        cursor.moveToFirst();
                if(cursor.getCount() != 0)
                        return cursor.getString(cursor.getColumnIndex(KEY_NAME));
                return Long.toString(phoneNumber);
        }

        public Cursor getContactsByName(String name) {
                return mDb.query(DATABASE_TABLE, new String[] {KEY_NAME, KEY_NUMBER}, KEY_NAME +
" LIKE \'%" + name + "%\'", null, null, null, KEY_NAME);
        }
}
```

## 1.1.2.4 ContactsTab

```
public class ContactsTab extends ListActivity implements ServiceConnection, Callback {
        public  static final String ID_CONTACT         = "idContact";
        public  static final String ID_COMMON_FRIENDS  = "commonFriends";
        private static final int    SEND_COMMON        = Menu.FIRST;
        private ContactsDB mDbHelper;
        private String     mSearchByName;
        private Messenger  mService;
        private Messenger  mActivity;

        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.contacts_list);
                mActivity = new Messenger(new Handler(this));
                mDbHelper = new ContactsDB(this);
                Bundle extras = getIntent().getExtras();
                if(extras != null)
                        mSearchByName = extras.getString(ContactsDB.KEY_NAME);
                registerForContextMenu(getListView());
        }

        @Override
        protected void onPause() {
                Log.v("ContactsTab", "Pausing activity");
                disconnectFromService();
                if(mDbHelper != null)
                        mDbHelper.close();
                super.onPause();
        }

        @Override
        protected void onResume() {
                super.onResume();
                Log.v("ContactsTab", "Resuming activity");
                connectToService();
                mDbHelper.open();
                fillData();
        }

        @Override
        protected void onDestroy() {
                Log.v("ContactsTab", "Destroying activity");
                super.onDestroy();
        }

        @Override
        public boolean onContextItemSelected(MenuItem item) {
                switch(item.getItemId()) {
        case SEND_COMMON:
                AdapterContextMenuInfo info    = (AdapterContextMenuInfo) item.getMenuInfo();
                Intent intent = new Intent(this, MessagesChat.class);
                intent.putExtra(ID_COMMON_FRIENDS, info.id);
                startActivity(intent);
                return true;
                }
                return super.onContextItemSelected(item);
        }

        @Override
```

```
	public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo) {
		super.onCreateContextMenu(menu, v, menuInfo);
		menu.add(0, SEND_COMMON, 0, R.string.send_common_friends);
	}

	private void connectToService() {
		Intent intent = new Intent(this,MessagingAppService.class);
		intent.putExtra(MessagingAppService.ACTIVITY_ID,
MessagingAppService.ACT_CONTACTSTAB);
		if(!getApplicationContext().bindService(intent, this, Service.BIND_AUTO_CREATE))
			Log.v("ConnectService", "Impossible connect to the service");
	}

	private void disconnectFromService() {
		if(mService != null) {
			Message message = new Message();
			message.what    = MessagingAppService.DISCONNECT;
			try {
				mService.send(message);
			} catch (RemoteException e) {
				e.printStackTrace();
			}
			getApplicationContext().unbindService(this);
		}
	}

	private Vector<String> getContactNames(Vector<Integer> numbers) {
		Vector<String> names = new Vector<String>();
		Uri uri;
		Cursor cursor;
		for(int i = 0; i < numbers.size(); i++) {
			uri    =
Uri.withAppendedPath(ContactsContract.PhoneLookup.CONTENT_FILTER_URI,
Uri.encode(numbers.get(i).toString()));
			cursor = getContentResolver().query(uri, new String[]
{PhoneLookup.DISPLAY_NAME}, null, null, null);
			cursor.moveToFirst();

	names.add(cursor.getString(cursor.getColumnIndex(PhoneLookup.DISPLAY_NAME)));
			cursor.close();
		}
		return names;
	}

	protected void fillData() {
		Cursor contactsCursor;
		if(mSearchByName == null)
			contactsCursor = mDbHelper.getAllContacts();
		else
			contactsCursor = mDbHelper.getContactsByName(mSearchByName);
		startManagingCursor(contactsCursor);
		String[] from = new String[]{ContactsDB.KEY_NAME};
		int[]    to   = {android.R.id.text1};
		SimpleCursorAdapter messages = new SimpleCursorAdapter(this,
android.R.layout.simple_list_item_1, contactsCursor, from, to);
		setListAdapter(messages);
	}

	@Override
    protected void onListItemClick(ListView l, View v, int position, long id) {
        super.onListItemClick(l, v, position, id);
        Intent intent = new Intent(this,MessagesChat.class);
        intent.putExtra(ID_CONTACT,id);
        startActivity(intent);
    }

	public void onClickSearch(View v) {
		switch(v.getId()) {
			case R.id.android_search_contact_button:
				EditText contactName =
(EditText)findViewById(R.id.android_search_contact_edittext);
				String   contact     = contactName.getText().toString();
				if(contact.length() != 0) {
					Intent intent = new
Intent(getBaseContext(),ContactsTab.class);
					intent.putExtra(ContactsDB.KEY_NAME, contact);
					startActivity(intent);
				}
				else
					Toast.makeText(this, "You need to type the name of
the contact", Toast.LENGTH_LONG).show();
				break;
		}
	}

	@Override
	public boolean handleMessage(Message msg) {
		ProtocolMessage message = new ProtocolMessage((String)msg.obj);
		Vector<String>  names   = getContactNames(message.getReceivers());
		mDbHelper.deleteAllContacts();
		mDbHelper.addContacts(names, message.getReceivers());
		fillData();
		return true;
	}

	@Override
	public void onServiceConnected(ComponentName arg0, IBinder service) {
		mService        = new Messenger(service);
		Message message = new Message();
		message.what    = MessagingAppService.CONNECT;
		message.arg1    = MessagingAppService.ACT_CONTACTSTAB;
		message.replyTo = mActivity;
		try {
			mService.send(message);
		} catch (RemoteException e) {
			e.printStackTrace();
		}
	}

	@Override
	public void onServiceDisconnected(ComponentName name) {
		mService = null;
	}
}
```

## 1.1.2.5 CreateGroup

```
public class CreateGroup extends ListActivity{
	public static final String GROUP_MEMBERS = "group_members";
	public static final String GROUP_NAME    = "group_name";

	private ContactsDB     mContactsDB;
	private GroupsDB       mGroupDB;
	private MembersGroupDB mGroupMembersDB;
	private boolean        mGroupChat = false;

	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
	    setContentView(R.layout.create_group);
	    Bundle extras   = getIntent().getExtras();
	    if(extras != null && extras.getLong(ChatsTab.ID_CHAT) == 1L)
	        mGroupChat = true;
	    mContactsDB     = new ContactsDB(this);
	    mGroupDB        = new GroupsDB(this);
	    mGroupMembersDB = new MembersGroupDB(this);
	    registerForContextMenu(getListView());
	    getListView().setChoiceMode(ListView.CHOICE_MODE_MULTIPLE);
	 }

	private void openDataBases() {
		mContactsDB.open();
		mGroupDB.open();
```

```
                mGroupMembersDB.open();
        }

        private void closeDatabases() {
                if(mContactsDB != null)
                        mContactsDB.close();
                if(mGroupDB != null)
                        mGroupDB.close();
                if(mGroupMembersDB != null)
                        mGroupMembersDB.close();
        }

        @Override
        protected void onResume() {
                super.onResume();
                openDataBases();
                fillData();
        }

        @Override
        protected void onPause() {
                closeDatabases();
                super.onPause();
        }

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

        protected void fillData() {
                Cursor contactsCursor = mContactsDB.getAllContacts();
                startManagingCursor(contactsCursor);
                String[] from = new String[]{ContactsDB.KEY_NAME};
                int[]    to   = {android.R.id.text1};
                SimpleCursorAdapter messages = new SimpleCursorAdapter(this,
android.R.layout.simple_list_item_multiple_choice, contactsCursor, from, to);
                setListAdapter(messages);
        }

        public void onClick(View v) {
                switch(v.getId()) {
                        case R.id.android_confirm_button_create_group:
                                EditText groupName =
(EditText)findViewById(R.id.android_group_name_edittext);
                                String   group     = groupName.getText().toString();
                                String contactName;
                                if(group.length() != 0) {
                                        ListView list       = getListView();
                                        ListAdapter adapter = getListAdapter();
                                        long ids[] = list.getCheckedItemIds();
                                        if(!mGroupChat) {
                                                long idGroup =
mGroupDB.createGroup(group);

                                                SparseBooleanArray posChecked =
list.getCheckedItemPositions();

                                                for(int j = 0,i = 0; i < list.getCount();
i++) {

                                                        if(posChecked.get(i)) {
                                                                contactName =
mContactsDB.getContactName(adapter.getItemId(i));

        mGroupMembersDB.addMember(idGroup, contactName, ids[j]);
                                                                j++;
                                                        }
                                                }
                                        }
                                        else {
                                                Intent intent = new Intent();
                                                intent.putExtra(GROUP_NAME, group);
                                                intent.putExtra(GROUP_MEMBERS, ids);
                                                setResult(RESULT_OK, intent);
                                                }
                                                finish();
                                        }
                                        else
                                                Toast.makeText(this, "You have to type the name of
the group", Toast.LENGTH_LONG).show();
                                        break;
                        }
                }
        }
```

## 1.1.2.6 GroupMembers

```
public class GroupMembers extends ListActivity{
        private MembersGroupDB mDbHelper;
        private long           idGroup;

        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
            setContentView(R.layout.group_members);
            Bundle extras = getIntent().getExtras();
            idGroup = extras.getLong(MembersGroupDB.KEY_ROWID);
            mDbHelper = new MembersGroupDB(this);
            registerForContextMenu(getListView());
            getListView().setChoiceMode(ListView.CHOICE_MODE_MULTIPLE);
         }

        @Override
        protected void onResume() {
                super.onResume();
                mDbHelper.open();
                fillData();
        }

        @Override
        protected void onPause() {
                if(mDbHelper != null)
                        mDbHelper.close();
                super.onPause();
        }

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

        protected void fillData() {
                Cursor membersCursor = mDbHelper.getMembersGroup(idGroup);
                startManagingCursor(membersCursor);
                String[] from = new String[]{MembersGroupDB.KEY_MEMBER_NAME};
                int[]    to   = {android.R.id.text1};
                SimpleCursorAdapter messages = new SimpleCursorAdapter(this,
android.R.layout.simple_list_item_multiple_choice, membersCursor, from, to);
                setListAdapter(messages);
        }

        public void onClick(View v) {
                switch(v.getId()) {
                        case R.id.android_confirm_button_members_group:
                                ListView list        = getListView();
                                long[]   selectedIds  = list.getCheckedItemIds();
                                if(selectedIds.length > 0) {
                                        long[]   membersPhone = new
long[selectedIds.length];

                                        Cursor   member;
                                        for(int i = 0; i < selectedIds.length; i++) {
                                                member =
mDbHelper.getMember(selectedIds[i]);
                                                membersPhone[i] =
member.getLong(member.getColumnIndex(MembersGroupDB.KEY_MEMBER_PHONE));
```

```
                                        }
                                        Intent intent = new Intent(this,
MessagesChat.class);
                                        intent.putExtra(MembersGroupDB.KEY_MEMBER_PHONE,
membersPhone);
                                        startActivity(intent);
                                        finish();
                                }
                                else
                                        Toast.makeText(this, "You have to select at least
one member", Toast.LENGTH_LONG).show();
                                break;
                        }
                }
}
```

## 1.1.2.7 GroupsDB

```
public class GroupsDB {
        public static final String KEY_ROWID        = "_id";
        public static final String KEY_NAME         = "name";

        private static final String DATABASE_NAME    = "messagingAppDB";
    private static final String DATABASE_TABLE   = "groupsDB";
        private static final String TAG              = "GroupsDB";
        private static final String DATABASE_CREATE  = "CREATE TABLE IF NOT EXISTS " +
DATABASE_TABLE + " ("
                    + KEY_ROWID        + " integer PRIMARY KEY autoincrement, "
                    + KEY_NAME         + " text    NOT NULL);";
    private static final int    DATABASE_VERSION = 1;

    private final Context  mCtx;
        private DatabaseHelper mDbHelper;
        private SQLiteDatabase mDb;

        private static class DatabaseHelper extends SQLiteOpenHelper {

        DatabaseHelper(Context context) {
            super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }

        @Override
        public void onCreate(SQLiteDatabase db) {

            db.execSQL(DATABASE_CREATE);
        }

        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
            if(db.getVersion() != newVersion) {
                    Log.w(TAG, "Upgrading database from version " + oldVersion + " to "
                        + newVersion + ", which will destroy all old data");
                    db.execSQL("DROP TABLE IF EXISTS " + DATABASE_TABLE + ";");
                onCreate(db);
            }
        }

        public void createTable(SQLiteDatabase db) {
          db.execSQL(DATABASE_CREATE);
        }
    }

          public GroupsDB(Context ctx) {
                    mCtx = ctx;
          }

          public GroupsDB open() throws SQLException {
        mDbHelper = new DatabaseHelper(mCtx);
        mDb = mDbHelper.getWritableDatabase();
        mDbHelper.createTable(mDb);
        return this;
    }

          public void close() {
        mDbHelper.close();
    }

          public long createGroup(String name) {
                    ContentValues initialValues = new ContentValues();
                    initialValues.put(KEY_NAME, name);
                    return mDb.insert(DATABASE_TABLE, null, initialValues);
          }

          public boolean deleteGroup(long id) {
                    return mDb.delete(DATABASE_TABLE, KEY_ROWID + "=" + id, null) > 0;
          }

          public Cursor getAllGroups() {
                    return mDb.query(DATABASE_TABLE, new String [] {KEY_ROWID, KEY_NAME}, null, null,
null, null, KEY_NAME);
          }

          public Cursor getGroup(long rowID) {
                    Cursor cursor = mDb.query(DATABASE_TABLE, new String [] {KEY_ROWID, KEY_NAME},
KEY_ROWID + "=" + rowID, null, null, null, null);
                    if(cursor != null)
                              cursor.moveToFirst();
                    return cursor;
          }

          public Cursor getGroupsByName(String name) {
                    return mDb.query(DATABASE_TABLE, new String[] {KEY_ROWID, KEY_NAME}, KEY_NAME + "
LIKE \'%" + name + "%\'", null, null, null, KEY_NAME);
          }
}
```

## 1.1.2.8 GroupsTab

```
public class GroupsTab extends ListActivity {
          private static final int DELETE_ID = Menu.FIRST;
          private GroupsDB       mGroupsDB;
          private MembersGroupDB mMembersGroupDB;
          private String   mSearchGroupByName;

          public void onCreate(Bundle savedInstanceState) {
                    super.onCreate(savedInstanceState);
              setContentView(R.layout.group_list);
              Bundle extras = getIntent().getExtras();
              if(extras != null)
                    mSearchGroupByName = extras.getString(GroupsDB.KEY_NAME);
              mGroupsDB       = new GroupsDB(this);
              mMembersGroupDB = new MembersGroupDB(this);
              registerForContextMenu(getListView());
           }

          @Override
          protected void onResume() {
                    super.onResume();
                    openDatabases();
                    fillData();
          }

          @Override
          protected void onPause() {
                    closeDatabases();
                    super.onPause();
          }
```

```
	@Override
	protected void onDestroy() {
		super.onDestroy();
	}

	@Override
	public boolean onContextItemSelected(MenuItem item) {
		switch(item.getItemId()) {
			case DELETE_ID:
				AdapterContextMenuInfo info = (AdapterContextMenuInfo)
item.getMenuInfo();
				mGroupsDB.deleteGroup(info.id);
				mMembersGroupDB.deleteAllMembersGroup(info.id);
				fillData();
				return true;
		}
		return super.onContextItemSelected(item);
	}

	@Override
	public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo) {
		super.onCreateContextMenu(menu, v, menuInfo);
		menu.add(0, DELETE_ID, 0, R.string.delete_group);
	}

	private void openDatabases() {
		mGroupsDB.open();
		mMembersGroupDB.open();
	}

	private void closeDatabases() {
		if(mGroupsDB != null)
			mGroupsDB.close();
		if(mMembersGroupDB != null)
			mMembersGroupDB.close();
	}

	protected void fillData() {
		Cursor groupsCursor;
		if(mSearchGroupByName == null)
			groupsCursor = mGroupsDB.getAllGroups();
		else
			groupsCursor = mGroupsDB.getGroupsByName(mSearchGroupByName);
		startManagingCursor(groupsCursor);
		String[] from = new String[]{GroupsDB.KEY_NAME};
		int[]    to   = {android.R.id.text1};
		SimpleCursorAdapter groups = new SimpleCursorAdapter(this,
android.R.layout.simple_list_item_1, groupsCursor, from, to);
		setListAdapter(groups);
	}

    @Override
    protected void onListItemClick(ListView l, View v, int position, long id) {
        super.onListItemClick(l, v, position, id);
        Intent intent = new Intent(this,GroupMembers.class);
        intent.putExtra(GroupsDB.KEY_ROWID,id);
        startActivity(intent);
     }

	public void onClick(View v) {
		Intent intent;
		switch(v.getId()) {
			case R.id.android_new_group_layout:
				intent = new Intent(this,CreateGroup.class);
				startActivity(intent);
				break;
			case R.id.android_search_group_button:
				EditText groupName =
(EditText)findViewById(R.id.android_search_group_edittext);
				String   group     = groupName.getText().toString();
				if(group.length() != 0) {
					intent = new Intent(this,GroupsTab.class);
					intent.putExtra(GroupsDB.KEY_NAME, group);
					startActivity(intent);
				}
				else
					Toast.makeText(this, "You need to type the name of
the group", Toast.LENGTH_LONG).show();
				break;
			}
		}
}
```

## 1.1.2.9 MembersGroupDB

```
public class MembersGroupDB {
	public static final String KEY_ROWID        = "_id";
	public static final String KEY_IDGROUP      = "group_id";
	public static final String KEY_MEMBER_PHONE = "number";
	public static final String KEY_MEMBER_NAME  = "name";

	private static final String DATABASE_NAME    = "messagingAppDB";
    private static final String DATABASE_TABLE   = "membersGroupDB";
	private static final String TAG              = "MembersGroupDB";
	private static final String DATABASE_CREATE  = "CREATE TABLE IF NOT EXISTS " +
DATABASE_TABLE + " ("

		+ KEY_ROWID        + " integer PRIMARY KEY autoincrement, "

		+ KEY_IDGROUP      + " integer NOT NULL, "

		+ KEY_MEMBER_NAME  + " text    NOT NULL, "

		+ KEY_MEMBER_PHONE + " integer NOT NULL);";
    private static final int    DATABASE_VERSION = 1;

    private final Context  mCtx;
	private DatabaseHelper mDbHelper;
	private SQLiteDatabase mDb;

	private static class DatabaseHelper extends SQLiteOpenHelper {

	DatabaseHelper(Context context) {
	    super(context, DATABASE_NAME, null, DATABASE_VERSION);
	}

	@Override
	public void onCreate(SQLiteDatabase db) {
	    db.execSQL(DATABASE_CREATE);
	}

	@Override
	public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
	    if(db.getVersion() != newVersion) {
	            Log.w(TAG, "Upgrading database from version " + oldVersion + " to "
	                + newVersion + ", which will destroy all old data");
	                    db.execSQL("DROP TABLE IF EXISTS " + DATABASE_TABLE + ";");
	        onCreate(db);
	   }
	}

	public void createTable(SQLiteDatabase db) {
	  db.execSQL(DATABASE_CREATE);
	}
    }

	public MembersGroupDB(Context ctx) {
		mCtx = ctx;
	}

	public MembersGroupDB open() throws SQLException {
    mDbHelper = new DatabaseHelper(mCtx);
```

```
        mDb = mDbHelper.getWritableDatabase();
        mDbHelper.createTable(mDb);
        return this;
    }

      public void close() {
        mDbHelper.close();
    }

        public long addMember(long idGroup, String name, long number) {
                ContentValues initialValues = new ContentValues();
                initialValues.put(KEY_IDGROUP, idGroup);
                initialValues.put(KEY_MEMBER_NAME, name);
                initialValues.put(KEY_MEMBER_PHONE, number);
                return mDb.insert(DATABASE_TABLE, null, initialValues);
        }

        public void addMembersVector(long idGroup, Vector<String> names, Vector<Integer> numbers) {
                for(int i = 0; i < names.size(); i++)
                        addMember(idGroup, names.get(i), numbers.get(i).intValue());
        }

        public boolean deleteMemberFromGroup(long id) {
                return mDb.delete(DATABASE_TABLE, KEY_ROWID + "=" + id, null) > 0;
        }

        public boolean deleteAllMembersGroup(long idGroup) {
                return mDb.delete(DATABASE_TABLE, KEY_IDGROUP + "=" + idGroup, null) > 0;
        }

        public Cursor getMembersGroup(long idGroup) {
                return mDb.query(DATABASE_TABLE, new String[] {KEY_ROWID, KEY_IDGROUP,
KEY_MEMBER_NAME,KEY_MEMBER_PHONE},
                                                                KEY_IDGROUP + "=" + idGroup, null, null,
null, KEY_MEMBER_NAME);
        }

        public Cursor getMember(long id) {
                Cursor cursor = mDb.query(DATABASE_TABLE, new String[] {KEY_ROWID, KEY_IDGROUP,
KEY_MEMBER_NAME,KEY_MEMBER_PHONE},
                                                                        KEY_ROWID + "=" +
id, null, null, null, null);
                if(cursor != null)
                        cursor.moveToFirst();
                return cursor;
        }
}
```

## 1.1.2.10 MessagesChat

```
public class MessagesChat extends ListActivity implements ServiceConnection, Callback{
        private static final int CHATS_ACTIVITY         = 0;
        private static final int CONTACTS_ACTIVITY      = 1;
        private static final int GROUPMEMBERS_ACTIVITY  = 2;
        private static final int COMMONFRIENDS_ACTIVITY = 3;
        private static final int DELETE_ID              = Menu.FIRST;

        private MessagesDB mMessagesDB;
        private ChatsDB    mChatsDB;
        private ContactsDB mContactsDB;
        private long       mIdChat;
        private int        mFromActivity;
        private long[]     mReceivers;
        private Messenger  mService;
        private Messenger  mActivity;
        private int        mSender = 677084244;

        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
            setContentView(R.layout.messages_list);
                mActivity     = new Messenger(new Handler(this));
                Bundle extras = getIntent().getExtras();
                mIdChat = extras.getLong(ChatsTab.ID_CHAT);
                if(mIdChat == 0L) {
                        mIdChat    = extras.getLong(ContactsTab.ID_CONTACT);
                        if(mIdChat == 0L) {
                                mIdChat = extras.getLong(ContactsTab.ID_COMMON_FRIENDS);
                                if(mIdChat == 0L) {
                                        mReceivers =
extras.getLongArray(MembersGroupDB.KEY_MEMBER_PHONE);
                                        mFromActivity = GROUPMEMBERS_ACTIVITY;
                                }
                                else
                                        mFromActivity = COMMONFRIENDS_ACTIVITY;
                        }
                        else
                                mFromActivity = CONTACTS_ACTIVITY;
                }
                else
                        mFromActivity = CHATS_ACTIVITY;
                mMessagesDB = new MessagesDB(this);
                mChatsDB    = new ChatsDB(this);
                mContactsDB = new ContactsDB(this);
                registerForContextMenu(getListView());
         }

        private void openDatabases() {
                mChatsDB.open();
                mMessagesDB.open();
                mContactsDB.open();
        }

        private void closeDatabases() {
                if(mChatsDB != null)
                        mChatsDB.close();
                if(mMessagesDB != null)
                        mMessagesDB.close();
                if(mContactsDB != null)
                        mContactsDB.close();
        }

        @Override
        protected void onResume() {
                super.onResume();
                connectToService();
                openDatabases();
                fillData();
        }

        @Override
        protected void onPause() {
                disconnectFromService();
                closeDatabases();
                super.onPause();
        }

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

        @Override
        public boolean onContextItemSelected(MenuItem item) {
                switch(item.getItemId()) {
                case DELETE_ID:
                        AdapterContextMenuInfo info = (AdapterContextMenuInfo)
item.getMenuInfo();
                        mMessagesDB.deleteMessage(info.id);
                        fillData();
                        return true;
                }
                return super.onContextItemSelected(item);
```

```
	}

	@Override
	public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo) {
		super.onCreateContextMenu(menu, v, menuInfo);
		menu.add(0, DELETE_ID, 0, R.string.delete_message);
	}

	private void connectToService() {
		Intent intent = new Intent(this,MessagingAppService.class);
		intent.putExtra(MessagingAppService.ACTIVITY_ID,
MessagingAppService.ACT_MESSAGESCHATTAB);
		if(!getApplicationContext().bindService(intent, this, Service.BIND_AUTO_CREATE))
			Log.v("ConnectService", "Impossible connect to the service");
	}

	private void disconnectFromService() {
		if(mService != null) {
			Message message = new Message();
			message.what    = MessagingAppService.DISCONNECT;
			try {
				mService.send(message);
			} catch (RemoteException e) {
				e.printStackTrace();
			}
			getApplicationContext().unbindService(this);
		}
	}

	protected void fillData() {
		if(mFromActivity == CHATS_ACTIVITY || mFromActivity == CONTACTS_ACTIVITY) {
			Cursor messagesCursor;
			if(mFromActivity == CHATS_ACTIVITY)
				messagesCursor = mMessagesDB.getMessagesChat(mIdChat);
			else
				messagesCursor = mMessagesDB.getMessagesReceiver(mIdChat);
			startManagingCursor(messagesCursor);
			String[] from = new String[]{MessagesDB.KEY_SENDER_NAME,
MessagesDB.KEY_DATE, MessagesDB.KEY_TIME, MessagesDB.KEY_MESSAGE};
			int[]    to   = {R.id.android_textview_message_name,
R.id.android_message_date_textview,

R.id.android_message_time_textview, R.id.android_message_text_textview};
			SimpleCursorAdapter messages = new SimpleCursorAdapter(this,
R.layout.message_row, messagesCursor, from, to);
			setListAdapter(messages);
		}
	}

	public void onClick(View v) {
		switch(v.getId()) {
			case R.id.android_send_message_button:
				EditText messageEditText  =
(EditText)findViewById(R.id.android_write_message_edittext);
				String   message          =
messageEditText.getText().toString();
				ProtocolMessage protMess  = null;
				Vector<Integer> receivers = new Vector<Integer>();
				if(message.length() > 0) {
					switch(mFromActivity) {
						case CHATS_ACTIVITY:
							long receiver =
mChatsDB.getSender(mIdChat);

	mMessagesDB.addMessageToChat(mIdChat, receiver, "Me", mSender, message,
Calendar.getInstance());
							mChatsDB.updateChat(mIdChat,
message, Calendar.getInstance());
							if(receiver == 0L) {
								String groupName =
mChatsDB.getChatName(mIdChat);
```

```
protMess = new
ProtocolMessage(ProtocolMessage.SMG, mSender, new Vector<Integer>(),

Calendar.getInstance(), groupName, message);
							}
							else {
								receivers.add(new
Integer((int)receiver));
protMess = new
ProtocolMessage(ProtocolMessage.SSM, mSender, receivers,
Calendar.getInstance(), null,
message);
							}
							break;
						case CONTACTS_ACTIVITY:
							long idChat =
mChatsDB.getIdChatBySender(mIdChat);
							if(idChat == ChatsDB.NO_CHAT)
								idChat =
mChatsDB.createChat(mContactsDB.getContactName(mIdChat), mIdChat);

	mMessagesDB.addMessageToChat(idChat, mIdChat, "Me", mSender, message,
Calendar.getInstance());
							mChatsDB.updateChat(idChat,
message, Calendar.getInstance());
							receivers.add(new
Integer((int)mIdChat));
protMess = new
ProtocolMessage(ProtocolMessage.SSM, mSender, receivers,

Calendar.getInstance(), null, message);
							break;
						case COMMONFRIENDS_ACTIVITY:
							receivers.add(new
Integer((int)mIdChat));
protMess = new
ProtocolMessage(ProtocolMessage.SMC, mSender, receivers,

Calendar.getInstance(), null, message);
							break;
						case GROUPMEMBERS_ACTIVITY:
							for(int i = 0; i <
mReceivers.length; i++)

	receivers.add((int)mReceivers[i]);
protMess = new
ProtocolMessage(ProtocolMessage.SSM, mSender, receivers,
Calendar.getInstance(), null,
message);
							break;
					}
					fillData();
					Message mess = new Message();
					mess.what    = MessagingAppService.MESSAGE;
					if(protMess != null) {
						mess.obj     = protMess.messToString();
						try {
							mService.send(mess);
						} catch (RemoteException e) {
							e.printStackTrace();
						}
					}
				}
				else
					Toast.makeText(this, "You have to write a message",
Toast.LENGTH_LONG).show();
				}
		}

		@Override
		public boolean handleMessage(Message msg) {
			ProtocolMessage message = new ProtocolMessage((String)msg.obj);
```

```
                    long chatId;
                    switch(message.getType()) {
                              case ProtocolMessage.SSM:
                                        chatId = mChatsDB.getIdChatBySender(message.getSender());
                                        if(chatId == ChatsDB.NO_CHAT)
                                                  chatId =
mChatsDB.createChat(mContactsDB.getContactName(message.getSender()), message.getSender());
                                        mMessagesDB.addMessageToChat(chatId, message.getSender(),
mContactsDB.getContactName(message.getSender()),
                                                                                                  message.getSender(),
message.getMessage(), message.getDate());
                                        mChatsDB.updateChat(chatId, message.getMessage(),
message.getDate());
                                        break;
                              case ProtocolMessage.ACG:
                                        chatId = mChatsDB.getIdChat(message.getGroupName());
                                        if(chatId == ChatsDB.NO_CHAT)
                                                  chatId =
mChatsDB.createChat(message.getGroupName(), 0L);
                                        mMessagesDB.addMessageToChat(chatId, 0L,
mContactsDB.getContactName(message.getSender()), message.getSender(),

mContactsDB.getContactName(message.getReceivers().get(0)) + " has been added to the group",
                                                                                        message.getDate());
                                        break;
                              case ProtocolMessage.DCG:
                                        chatId = mChatsDB.getIdChat(message.getGroupName());
                                        if(chatId == ChatsDB.NO_CHAT)
                                                  chatId =
mChatsDB.createChat(message.getGroupName(), 0L);
                                        mMessagesDB.addMessageToChat(chatId, 0L,
mContactsDB.getContactName(message.getSender()), message.getSender(),

mContactsDB.getContactName(message.getReceivers().get(0)) + " has been deleted from the group",
                                                                                        message.getDate());
                                        break;
                              case ProtocolMessage.SMG:
                                        chatId = mChatsDB.getIdChat(message.getGroupName());
                                        if(chatId == ChatsDB.NO_CHAT)
                                                  chatId =
mChatsDB.createChat(message.getGroupName(), 0L);
                                        mMessagesDB.addMessageToChat(chatId, 0L,
mContactsDB.getContactName(message.getSender()), message.getSender(),

message.getMessage(), message.getDate());
                                        mChatsDB.updateChat(chatId, message.getMessage(),
message.getDate());
                                        break;
                    }
                    fillData();
                    return true;
          }

          @Override
          public void onServiceConnected(ComponentName name, IBinder service) {
                    mService        = new Messenger(service);
                    Message message = new Message();
                    message.what    = MessagingAppService.CONNECT;
                    message.arg1    = MessagingAppService.ACT_MESSAGESCHATTAB;
                    message.replyTo = mActivity;
                    try {
                              mService.send(message);
                    } catch (RemoteException e) {
                              e.printStackTrace();
                    }
          }

          @Override
          public void onServiceDisconnected(ComponentName name) {
                    mService = null;
          }
}
```



## 1.1.2.11 MessagesDB

```
public class MessagesDB {
          public static final String KEY_ROWID         = "_id";
          public static final String KEY_CHATID        = "id_chat";
          public static final String KEY_RECEIVERID    = "id_receiver";
          public static final String KEY_SENDER_NAME   = "sender_name";
          public static final String KEY_SENDER_NUMBER = "sender";
          public static final String KEY_MESSAGE       = "message";
          public static final String KEY_DATE          = "date";
          public static final String KEY_TIME          = "time";

          private static final String DATABASE_NAME    = "messagingAppDB";
    private static final String DATABASE_TABLE   = "messagesDB";
          private static final String TAG              = "MessagesDB";
          private static final String DATABASE_CREATE  = "CREATE TABLE IF NOT EXISTS " +
DATABASE_TABLE + " ("

                    + KEY_ROWID         + " integer PRIMARY KEY autoincrement, "

                    + KEY_CHATID        + " integer NOT NULL, "

                    + KEY_RECEIVERID    + " integer NOT NULL, "

                    + KEY_SENDER_NAME   + " text    NOT NULL, "

                    + KEY_SENDER_NUMBER + " integer NOT NULL, "

                    + KEY_MESSAGE       + " text    NOT NULL, "

                    + KEY_DATE          + " date    NOT NULL, "

                    + KEY_TIME          + " time    NOT NULL);";
    private static final int    DATABASE_VERSION = 1;

    private final Context  mCtx;
          private DatabaseHelper mDbHelper;
          private SQLiteDatabase mDb;

          private static class DatabaseHelper extends SQLiteOpenHelper {

        DatabaseHelper(Context context) {
            super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }

        @Override
        public void onCreate(SQLiteDatabase db) {

            db.execSQL(DATABASE_CREATE);
        }

        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
            if(db.getVersion() != newVersion) {
                    Log.w(TAG, "Upgrading database from version " + oldVersion + " to "
                        + newVersion + ", which will destroy all old data");
                             db.execSQL("DROP TABLE IF EXISTS " + DATABASE_TABLE + ";");
                onCreate(db);
            }
        }

        public void createTable(SQLiteDatabase db) {
          db.execSQL(DATABASE_CREATE);
        }
    }

          public MessagesDB(Context ctx) {
                    mCtx = ctx;
          }
```

```
        public MessagesDB open() throws SQLException {
        mDbHelper = new DatabaseHelper(mCtx);
        mDb = mDbHelper.getWritableDatabase();
        mDbHelper.createTable(mDb);
        return this;
    }

        public void close() {
        mDbHelper.close();
    }

        public long addMessageToChat(long idChat, long receiver, String senderName, int
senderPhone, String message, Calendar calendar) {
                    ContentValues initialValues = new ContentValues();
                    initialValues.put(KEY_CHATID, idChat);
                    initialValues.put(KEY_RECEIVERID, receiver);
                    initialValues.put(KEY_SENDER_NAME, senderName);
                    initialValues.put(KEY_SENDER_NUMBER, senderPhone);
                    initialValues.put(KEY_MESSAGE, message);
                    initialValues.put(KEY_DATE, calendar.get(Calendar.YEAR) + "-" +
(calendar.get(Calendar.MONTH) + 1) + "-" + calendar.get(Calendar.DAY_OF_MONTH));
                    initialValues.put(KEY_TIME, calendar.get(Calendar.HOUR_OF_DAY) + ":"
+calendar.get(Calendar.MINUTE));
                    return mDb.insert(DATABASE_TABLE, null, initialValues);
          }

          public boolean deleteMessage(long id) {
                    return mDb.delete(DATABASE_TABLE, KEY_ROWID + "=" + id, null) > 0;
          }

          public boolean deleteMessagesChat(long idChat) {
                    return mDb.delete(DATABASE_TABLE, KEY_CHATID + "=" + idChat, null) > 0;
          }

          public Cursor getMessagesChat(long idChat) {
                    return mDb.query(DATABASE_TABLE, new String[] {KEY_ROWID, KEY_CHATID,
KEY_RECEIVERID, KEY_SENDER_NAME, KEY_SENDER_NUMBER, KEY_MESSAGE, KEY_DATE, KEY_TIME},
                                                                KEY_CHATID + "=" + idChat, null, null,
null,KEY_DATE + "," + KEY_TIME);
          }

          public Cursor getMessagesReceiver(long idReceiver) {
                    return mDb.query(DATABASE_TABLE, new String[] {KEY_ROWID, KEY_CHATID,
KEY_RECEIVERID, KEY_SENDER_NAME, KEY_SENDER_NUMBER, KEY_MESSAGE, KEY_DATE, KEY_TIME},
                                                                KEY_RECEIVERID + "=" + idReceiver, null,
null, null, KEY_DATE + "," + KEY_TIME);
          }
}
```

## 1.1.2.12 ProtocolMessage

```
public class ProtocolMessage {
          public static final int CNT = 1;
          public static final int DSC = 2;
          public static final int SSM = 3;
          public static final int SCN = 4;
          public static final int CGR = 5;
          public static final int DGR = 6;
          public static final int ACG = 7;
          public static final int DCG = 8;
          public static final int SMG = 9;
          public static final int SMC = 10;
          private int             mType;
          private int             mSender;
          private Vector<Integer> mReceivers = new Vector<Integer>();
          private Calendar        mDate;
          private String          mGroupName;
          private String          mMessage;

public ProtocolMessage(int type, int sender, Vector<Integer> receivers,
                                        Calendar date, String groupName, String message) {
          mType      = type;
          mSender    = sender;
          mReceivers = receivers;
          mDate      = date;
          mDate      = Calendar.getInstance();
          mGroupName = groupName;
          mMessage   = message;
}

public ProtocolMessage(String message) {
          Gson    g  = new Gson();
          ProtocolMessage m  = g.fromJson(message,ProtocolMessage.class);
          mType      = m.getType();
          mSender    = m.getSender();
          mReceivers = m.getReceivers();
          mDate      = m.getDate();
          mGroupName = m.getGroupName();
          mMessage   = m.getMessage();
}

public int getType() {
          return mType;
}

public int getSender() {
          return mSender;
}

public int getNumReceivers() {
          return mReceivers.size();
}

public Vector<Integer> getReceivers() {
          return mReceivers;
}

public Calendar getDate() {
          return mDate;
}

public String getGroupName() {
          return mGroupName;
}

public String getMessage() {
          return mMessage;
}

public void printAll() {
          System.out.println("Type: " + mType);
          System.out.println("Sender: " + mSender);
          for(int i=0; i<mReceivers.size();i++) {
                    System.out.println("Receiver " + i + ": " + mReceivers.get(i));
          }
          System.out.println("Date: " + mDate.getTime());
          if(mGroupName != null)
                    System.out.println("Group Name: " + mGroupName);
          if(mMessage != null)
                    System.out.println("Message: " + mMessage);
}

public String messToString() {
          Gson g = new Gson();
          return g.toJson(this);
}

public Vector<String> toStringVector() {
          Vector<String> vString = new Vector<String>();
          for(int i = 0; i < mReceivers.size(); i++) {
                    Vector<Integer> v = new Vector<Integer>();
```

```
                        v.add(mReceivers.get(i));
                        vString.add(new ProtocolMessage(mType, mSender,v,mDate, mGroupName,
mMessage).messToString());
                }
                return vString;
        }
}
```

## 1.1.2.13 MessagingAppService

```
public class MessagingAppService extends Service implements Runnable, Callback {
        public  static final String ACTIVITY_ID         = "activity";
        public  static final int    ACT_CHATSTAB        = 0;
        public  static final int    ACT_CONTACTSTAB     = 1;
        public  static final int    ACT_MESSAGESCHATTAB = 2;
        public  static final int    CONNECT             = 6;
        public  static final int    DISCONNECT          = 7;
        public  static final int    MESSAGE             = 8;
        public  static final int    CONNECTION_PROBLEM  = 11;
        private static final String SERVER_IP           = "192.168.1.100";
        private static final int    PORT                = 11325;
        private static final int    NUM_ACTIVITIES      = 3;
        private static boolean         mExit  = true;
        private static int             mPhone = 679859157;
        private Socket                 mSocket;
        private NotificationManager    mNM;
        private BufferedReader         mIn;
        private PrintWriter            mOut;
        private Messenger              mService;
        private Messenger              mActivity;
        private int                    mActivityID = -1;
        private Vector<Vector<String>> mActivityMessages = new Vector<Vector<String>>();

        @Override
        public void onCreate() {
                super.onCreate();
                try {
                        mNM = (NotificationManager)getSystemService(NOTIFICATION_SERVICE);
                        mSocket = new Socket(InetAddress.getByName(SERVER_IP),PORT);

                        mIn     = new BufferedReader(new
InputStreamReader(mSocket.getInputStream()));
                        mOut    = new PrintWriter(mSocket.getOutputStream(),true);
                        mOut.flush();
                        mExit   = false;

                        for(int i = 0; i < NUM_ACTIVITIES; i++)
                                mActivityMessages.add(new Vector<String>());
                        mService = new Messenger(new Handler(this));

                        connectToServer();

                        Thread thread = new Thread(this);
                        thread.start();

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

        @Override
        public void onDestroy() {
                //TelephonyManager telMan   =
(TelephonyManager)getSystemService(Context.TELEPHONY_SERVICE);
                //int              sender   =
Integer.getInteger(telMan.getLine1Number()).intValue();
                ProtocolMessage message = new ProtocolMessage(ProtocolMessage.DSC, mPhone, new
Vector<Integer>(), Calendar.getInstance(), null, null);
                sendMessage(message.messToString());
                try {
                        mIn.close();
                        mOut.close();
                        mSocket.close();
                } catch (IOException e) {
                        e.printStackTrace();
                }
                super.onDestroy();
                mExit = true;
        }

        public void connectToServer() {
                ProtocolMessage  message  = new ProtocolMessage(ProtocolMessage.CNT, mPhone, new
Vector<Integer>(), Calendar.getInstance(), null, null);
                sendMessage(message.messToString());
                Vector<Integer> receivers = getContacts();
                Vector<Integer> receivers2 = new Vector<Integer>();
                receivers2.add(new Integer(677084244));
                receivers2.add(new Integer(669238052));
                receivers2.add(new Integer(638357906));
                receivers2.add(new Integer(606817088));
                message  = new ProtocolMessage(ProtocolMessage.SCN, mPhone, receivers2,
Calendar.getInstance(), null, null);
                sendMessage(message.messToString());
        }

        @Override
        public int onStartCommand(Intent intent, int flags, int startId) {
                return super.onStartCommand(intent, flags, startId);
        }

        private Vector<Integer> getContacts() {
                Vector<Integer> contacts = new Vector<Integer>();
                Cursor cursor = getContentResolver().query(Phone.CONTENT_URI,new String[]
{Phone.NUMBER}, null, null, null);
                while(cursor.moveToNext()) {
                        String phoneNumber =
cursor.getString(cursor.getColumnIndex(Phone.NUMBER));
                        phoneNumber = phoneNumber.replace("-", "");
                        if(phoneNumber.startsWith("+34"))
                                phoneNumber = phoneNumber.replaceFirst("\\+34", "");
                        else {
                                if(phoneNumber.startsWith("0034"))
                                        phoneNumber = phoneNumber.replaceFirst("0034", "");
                                else {
                                        if(phoneNumber.startsWith("00353"))
                                                phoneNumber =
phoneNumber.replaceFirst("00353", "");
                                }
                        }
                        Log.v("Number",phoneNumber);
                        contacts.add(new Integer(phoneNumber));
                }
                cursor.close();
                return contacts;
        }

        private boolean sendMessage(String message) {
                Log.v("Send Client", message);
                mOut.println(message);
                return true;
        }

        private void showNotification(String message) {
                ProtocolMessage protMessage = new ProtocolMessage(message);
                String tittle, mess;
                if(protMessage.getMessage() == null) {
                        tittle = protMessage.getSender() + ": New";
                        mess = "New Message!";
                }
                else {
                        tittle = protMessage.getSender() + ": " +
((protMessage.getMessage().length() < 5) ?
```

```
                                        protMessage.getMessage() :
protMessage.getMessage().substring(0, 5)+ "...");
                        mess   = protMessage.getMessage();
                }
                Notification notification = new Notification(R.drawable.icon, tittle,
System.currentTimeMillis());
                Intent intent;
                switch(protMessage.getType()) {
                        case ProtocolMessage.CGR:
                        case ProtocolMessage.DGR:
                                intent = new Intent(this, ChatsTab.class);
                                break;
                        case ProtocolMessage.SCN:
                                intent = new Intent(this, ContactsTab.class);
                                break;
                        case ProtocolMessage.SSM:
                        case ProtocolMessage.ACG:
                        case ProtocolMessage.DCG:
                        case ProtocolMessage.SMG:
                                intent = new Intent(this, MessagesChat.class);
                                intent.putExtra(MessagesDB.KEY_RECEIVERID,
protMessage.getSender());
                                break;
                        default:
                                intent = new Intent();
                }
                PendingIntent contentIntent = PendingIntent.getActivity(this, 0, intent, 0);
                notification.setLatestEventInfo(this , "New message from " +
protMessage.getSender(), mess, contentIntent);
                mNM.notify(("New message from " + protMessage.getSender()).hashCode(),
notification);
        }

        @Override
        public IBinder onBind(Intent intent) {
                return mService.getBinder();
        }

        @Override
        public boolean handleMessage(Message msg) {
                Message mess;
                switch(msg.what) {
                        case CONNECT:
                                mActivity   = msg.replyTo;
                                mActivityID = msg.arg1;
                                Log.v("handleMessage", "Binding to the activity " +
mActivityID);
                                mess      = new Message();
                                for(int i = 0; i < mActivityMessages.get(mActivityID).size();
i++) {
                                        mess.obj =
mActivityMessages.get(mActivityID).remove(i);
                                        Log.v("handleMessage", "Enviando mensaje pendiente
" + i + ": " + (String)mess.obj);
                                        try {
                                                mActivity.send(mess);
                                        } catch (RemoteException e) {
                                                e.printStackTrace();
                                        }
                                }
                                break;
                        case DISCONNECT:
                                mActivity   = null;
                                mActivityID = -1;
                                break;
                        case MESSAGE:
                                String message = (String) msg.obj;
                                if(!sendMessage(message)) {
                                        mess      = new Message();
                                        mess.what = CONNECTION_PROBLEM;
                                        try {
                                                mActivity.send(mess);
                                        } catch (RemoteException e) {
                                                e.printStackTrace();
                                                return false;
                                        }
                                }
                        }
                        return true;
                }

                public static boolean isRunning() {
                        return !mExit;
                }

                @Override
                public void run() {
                        while(!mExit) {
                                try {
                                        String message = mIn.readLine();
                                        if(message != null) {
                                                Log.v("Receive client", message);
                                                ProtocolMessage protMessage = new
ProtocolMessage(message);
                                                Message         mess        = new Message();
                                                mess.obj = message;
                                                showNotification(message);
                                                switch(protMessage.getType()) {
                                                        case ProtocolMessage.SSM:
                                                        case ProtocolMessage.ACG:
                                                        case ProtocolMessage.DCG:
                                                        case ProtocolMessage.SMG:
                                                                if(mActivity != null &&
mActivityID == ACT_MESSAGESCHATTAB)
                                                                        mActivity.send(mess);
                                                                else

                mActivityMessages.get(ACT_MESSAGESCHATTAB).add(message);
                                                                break;
                                                        case ProtocolMessage.SCN:
                                                                if(mActivity != null &&
mActivityID == ACT_CONTACTSTAB)
                                                                        mActivity.send(mess);
                                                                else

                mActivityMessages.get(ACT_CONTACTSTAB).add(message);
                                                                break;
                                                        case ProtocolMessage.CGR:
                                                        case ProtocolMessage.DGR:
                                                                if(mActivity != null &&
mActivityID == ACT_CHATSTAB)
                                                                        mActivity.send(mess);
                                                                else

                mActivityMessages.get(ACT_CHATSTAB).add(message);
                                                                break;
                                                }
                                        }
                                } catch (Exception e) {
                                        e.printStackTrace();
                                }
                        }
                }
}
```

## 1.1.2.14 MessagingAppClient

```
public class MessagingAppClient extends TabActivity {
        private Intent mServiceIntent;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        Resources res = getResources(); // Resource object to get Drawables

              TabHost tabHost = getTabHost();  // The activity TabHost
              TabHost.TabSpec spec;  // Resusable TabSpec for each tab
              Intent intent;  // Reusable Intent for each tab

              mServiceIntent = new Intent(this, MessagingAppService.class);
              startService(mServiceIntent);

              intent = new Intent().setClass(this, ChatsTab.class);
              spec = tabHost.newTabSpec("chats").setIndicator("Chats",
res.getDrawable(R.drawable.ic_tab_chats)).setContent(intent);
              tabHost.addTab(spec);

              intent = new Intent().setClass(this, ContactsTab.class);
              spec = tabHost.newTabSpec("contacts").setIndicator("Contacs",
res.getDrawable(R.drawable.ic_tab_contacts)).setContent(intent);
              tabHost.addTab(spec);

              intent = new Intent().setClass(this, GroupsTab.class);
              spec = tabHost.newTabSpec("groups").setIndicator("Groups",
res.getDrawable(R.drawable.ic_tab_groups)).setContent(intent);
              tabHost.addTab(spec);

              tabHost.setCurrentTab(1);
    }

}
```

# 1.1.2.15 Layouts

## 1.1.2.15.1 Chat_row.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
         android:id="@+id/relativeLayout1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_toLeftOf="@+id/date_chat"
        android:gravity="left"
        android:orientation="vertical" >
        <TextView
            android:id="@+id/row_chat_name"
            android:layout_width="wrap_content"
            android:layout_height="0dip"
            android:text="@string/chat_row_chat_text"
            android:textColor="#FFFFFF"
            android:textSize="20sp"
            android:layout_weight="1"
            android:gravity="left"/>
        <TextView
            android:id="@+id/last_message_chat"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/chat_row_message_text"
            android:maxLines="1"/>
    </LinearLayout>
    <TextView
        android:id="@+id/date_chat"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_alignParentRight="true"
        android:text="@string/chat_row_date_text"/>
</RelativeLayout>
```

## 1.1.2.15.2 chats_list.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <LinearLayout
        android:id="@+id/android:new_chat_layout"
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:clickable="true"
        android:background="#004444">
        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:src="@drawable/ic_add"
            android:contentDescription="@string/desc_add_icon"/>
        <TextView
            android:id="@+id/android:new_chat"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textAppearance="?android:attr/textAppearanceLarge"
            android:text="@string/new_chat_text"/>
    </LinearLayout>
    <ListView
        android:id="@+id/android:list"
          android:layout_width="match_parent"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

## 1.1.2.15.3 contacts_list.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <ListView
        android:id="@+id/android:list"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:layout_alignParentTop="true"
          android:layout_above="@+id/android:search_contact"/>
          <LinearLayout
              android:id="@+id/android:search_contact"
              android:orientation="horizontal"
              android:layout_width="match_parent"
              android:layout_height="wrap_content"
              android:layout_alignParentBottom="true">
        <EditText
            android:id="@+id/android:search_contact_edittext"
            android:layout_width="0dip"
            android:layout_height="wrap_content"
            android:layout_gravity="left"
            android:maxLines="1"
            android:layout_weight="1"
            android:hint="@string/search_contact_text"/>
        <LinearLayout
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:gravity="right" >
```

```
                    <Button
                    android:id="@+id/android:search_contact_button"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:onClick="onClickSearch"
                    android:clickable="true"
                    android:text="@string/search_button"/>
        </LinearLayout>
          </LinearLayout>
</RelativeLayout>
```

### 1.1.2.15.4 Create_group.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
          android:id="@+id/relativeLayout1"
          android:layout_width="match_parent"
          android:layout_height="match_parent">
          <LinearLayout
              android:id="@+id/android:group_name_layout"
              android:orientation="horizontal"
              android:layout_width="match_parent"
              android:layout_height="wrap_content"
              android:layout_alignParentTop="true">
              <TextView
                  android:id="@+id/android:group_name_textview"
                  android:layout_width="wrap_content"
                  android:layout_height="wrap_content"
                  android:text="@string/group_name_text_textview"/>
              <EditText
                  android:id="@+id/android:group_name_edittext"
                  android:layout_width="match_parent"
                  android:layout_height="wrap_content"
                  android:maxLines="1"
                  android:hint="@string/group_name_hinttext"/>
          </LinearLayout>
          <ListView
              android:id="@+id/android:list"
              android:layout_width="match_parent"
              android:layout_height="wrap_content"
              android:layout_below="@+id/android:group_name_layout"
              android:layout_above="@+id/android:confirm_button_create_group"/>
          <Button
              android:id="@+id/android:confirm_button_create_group"
              android:layout_width="match_parent"
              android:layout_height="wrap_content"
              android:layout_alignParentBottom="true"
              android:onClick="onClick"
              android:clickable="true"
              android:text="@string/confirm_button_text"/>
</RelativeLayout>
```

### 1.1.2.15.5 Group_list.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <LinearLayout android:id="@+id/list_groups"
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_above="@+id/android:search_group">
          <LinearLayout
              android:id="@+id/android:new_group_layout"
          android:orientation="horizontal"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          android:onClick="onClick"
          android:clickable="true"
          android:background="#004444">
          <ImageView
                    android:layout_width="wrap_content"
                android:layout_height="match_parent"
              android:src="@drawable/ic_add"
                    android:contentDescription="@string/desc_add_icon"/>
          <TextView
                    android:id="@+id/android:new_group"
                android:layout_width="wrap_content"
              android:layout_height="wrap_content"
                    android:textAppearance="?android:attr/textAppearanceLarge"
                    android:text="@string/new_group_text"/>
          </LinearLayout>
          <ListView android:id="@+id/android:list"
                    android:layout_width="match_parent"
          android:layout_height="wrap_content"/>
          </LinearLayout>
          <LinearLayout
              android:id="@+id/android:search_group"
              android:orientation="horizontal"
              android:layout_width="match_parent"
              android:layout_height="wrap_content"
              android:layout_alignParentBottom="true">
        <EditText
            android:id="@+id/android:search_group_edittext"
            android:layout_width="0dip"
            android:layout_height="wrap_content"
            android:layout_gravity="left"
            android:maxLines="1"
            android:layout_weight="1"
            android:hint="@string/search_group_text"/>
        <LinearLayout
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:gravity="right" >
                    <Button
                    android:id="@+id/android:search_group_button"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:onClick="onClick"
                    android:clickable="true"
                    android:text="@string/search_button"/>
        </LinearLayout>
          </LinearLayout>
</RelativeLayout>
```

### 1.1.2.15.6 Group_members.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
          android:id="@+id/relativeLayout1"
          android:layout_width="match_parent"
          android:layout_height="match_parent">
          <ListView
              android:id="@+id/android:list"
              android:layout_width="match_parent"
              android:layout_height="wrap_content"
              android:layout_above="@+id/android:confirm_button_members_group"
              android:layout_alignParentTop="true"/>
          <Button
              android:id="@+id/android:confirm_button_members_group"
              android:layout_width="match_parent"
```

```
            android:layout_height="wrap_content"
            android:layout_alignParentBottom="true"
            android:onClick="onClick"
            android:clickable="true"
            android:text="@string/confirm_button_text"/>
</RelativeLayout>
```

## 1.1.2.15.7 Login.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
            android:id="@+id/relativeLayout1"
            android:layout_width="match_parent"
            android:layout_height="match_parent" >
            <TextView
                        android:id ="@+id/android:login_textview"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:layout_centerInParent="true"
                        android:layout_above="@+id/android:login_edittext"
                        android:text="@string/login_string_number"
                        android:textAppearance="?android:attr/textAppearanceLarge"/>
            <EditText
               android:id="@+id/android:login_edittext"
               android:layout_width="wrap_content"
               android:layout_height="wrap_content"
               android:layout_centerInParent="true"
               android:layout_alignParentRight="true"
               android:layout_alignParentLeft="true"
               android:layout_marginRight="20dip"
               android:layout_marginLeft="20dip"
               android:inputType="number" >
               <requestFocus />
            </EditText>
            <Button
                        android:id="@+id/android:login_button"
                        android:layout_width="match_parent"
                        android:layout_height="wrap_content"
                        android:layout_alignParentBottom="true"
                        android:clickable="true"
                        android:onClick="onClick"
                        android:text="@string/confirm_button_text"
            />
            <ImageView
               android:id="@+id/login_image"
               android:layout_width="wrap_content"
               android:layout_height="wrap_content"
               android:layout_alignParentTop="true"
               android:layout_centerHorizontal="true"
               android:src="@drawable/logo_messagingapp2" />

</RelativeLayout>
```

## 1.1.2.15.8 Message_row.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/linearLayout1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="5dip"
    android:background="#004444">
    <TextView
        android:id="@+id/android:textview_message_name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_toLeftOf="@+id/android:message_date_textview"
        android:textColor="#c0c0c0"
        android:textSize="15sp"
        android:text="@string/message_name"/>
    <TextView
        android:id="@+id/android:message_date_textview"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_toLeftOf="@+id/android:message_time_textview"
        android:layout_marginRight="5dip"
        android:text="@string/message_date"
        android:textColor="#c0c0c0"/>
    <TextView
        android:id="@+id/android:message_time_textview"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_alignParentRight="true"
        android:text="@string/message_time"
        android:textColor="#c0c0c0"/>
    <TextView
        android:id="@+id/android:message_text_textview"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/android:textview_message_name"
        android:text="@string/message_text"
        android:textSize="20sp"
        android:textColor="#ffffff"/>
</RelativeLayout>
```

## 1.1.2.15.9 Messages_list.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ListView
        android:id="@+id/android:list"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:layout_above="@+id/android:write_message"
          android:layout_alignParentTop="true"
          android:layout_marginBottom="3dip"/>
    <LinearLayout
        android:id="@+id/android:write_message"
              android:orientation="horizontal"
              android:layout_width="match_parent"
              android:layout_height="wrap_content"
              android:layout_alignParentBottom="true">
        <EditText
            android:id="@+id/android:write_message_edittext"
            android:layout_width="0dip"
            android:layout_height="wrap_content"
            android:layout_gravity="left"
            android:layout_weight="1"
            android:hint="@string/write_message_hinttext"/>
        <LinearLayout
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:gravity="right" >
                    <Button
                    android:id="@+id/android:send_message_button"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:onClick="onClick"
                    android:clickable="true"
                    android:text="@string/send_message_button_text"/>
```

```
        </LinearLayout>
          </LinearLayout>
</RelativeLayout>
```

## 1.1.2.15.10 Main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/tabhost"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:padding="5dp">
        <TabWidget
            android:id="@android:id/tabs"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:background="#000044"/>
        <FrameLayout
            android:id="@android:id/tabcontent"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:padding="5dp" />
    </LinearLayout>
</TabHost>
```