VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Adeoye, Adebayo Adebowale

ShareBox: An Android-based Multi-user and Multi-device File Syncing Application

Technology and Communication
2011

ACKNOWLEDGEMENT

My first thanks go to Almighty God who has afforded me the opportunity to start and conclude this project in sound health. I am also using this medium to appreciate my friends and family for their words of encouragement every now and then since I began this final work in my degree program.

My thanks also go to Google Inc. for giving us Android and for being a blessing to our generation. Lastly, I would like to appreciate my supervisor, Dr Liu Yang and every other teacher who has impacted on me since I started this degree.

VAASAN AMMATTIKORKEAKOULU

UNIVERSITY OF APPLIED SCIENCES

Information Technology Degree Programme

ABSTRACT

Author	Adeoye, Adebayo Adebowale
Title	ShareBox: An Android-based Multi-user and Multi-device File Syncing Application
Year	2011
Language	English
Pages	44
Name of Supervisor	Liu Yang

With the rate at which the usage of smart phones and devices increases nowadays, there is more pressing need for mobile applications that would make life a lot easier for end users. One of the platforms that make the usage and development of such applications easy is the Android platform. Because it is open source and also easy to use and develop for, the Android OS is fast becoming the most popular operating system for smart phones.

The basic objective of this project is to develop an application that would enable users to be able to share, modify, update and sync files on multiple Android devices.

Keywords: Android, XML, class, application, side, Client, Server

CONTENTS

ACKNOWLEDGEMENT 8

ABBREVIATIONS 12

1 INTRODUCTION 7

1.1 Android 8

1.2 Project Objective 9

1.3 Some Issues on the Proposed Application 9

1.4 Alternative Application 10

2 DESIGN AND DEVELOPMENT APPROACH 11

2.1 Requirements 11

2.1.1 Application's Functional Requirements 11

2.1.2 Application's Non-functional Requirements 11

2.2 System Architecture 11

2.2.1 Simple Case Scenario 13

2.3 Android SDK 14

2.4 Server Side Technologies 15

2.5 Data Structures for transmitting data 16

3 IMPLEMENTATION 18

3.1 Eclipse 18

3.2 Android Virtual Machine 18

3.3 Structure of the Project on Eclipse 19

3.3.1 /Android 2.2 Directory 21

3.3.2 /src 21

3.3.3 /gen 21

3.3.4 /assets 21

3.3.5 /res 21

3.3.6 /res/drawable 21

3.3.7 /res/drawable-hdpi 22

3.3.8 /res/drawable-ldpi 22

3.3.9 /res/drawable-mdpi 22
3.3.10 /res/layout 22
3.3.11 /res/menu 22
3.3.12 /res/values 22
3.3.13 AndroidManifest.xml 22
3.4 Server Side of the Application 23
3.5 Client Side of the Application 30
4 TESTING THE APPLICATION 39
4.1 Testing the Server Side 39
4.2 Testing the Client Side 41
4.3 Testing in the Real World 41
5 CONCLUSION 42
5.1 Evaluation 42
5.2 Challenges 43
5.3 Possible Future Expansions 43
REFERENCES 44

ABBREVIATIONS

GPS	GLOBAL POSITIONING SYSTEM
IDE	INTEGRATED DEVELOPMENT ENVIRONMENT
SMS	SHORT MESSAGE SERVICE
OS	OPERATING SYSTEM
GSM	GLOBAL SYSTEM FOR MOBILE COMMNUNICATIONS
SDK	SOFTWARE DEVELOPMENT KIT
XML	EXTENSIBLE MARKUP LANGUAGE
JSON	JAVASCRIPT OBJECT NOTATION
TCP	TRANSMISSION CONTROL PROTOCOL
IP	INTERNET PROTOCOL
URL	UNIVERSAL RESOURCE LOCATOR

1 INTRODUCTION

A smart phone is a sophisticated mobile phone that serves both as a typical mobile phone and a personal digital assistant. A personal digital assistant (PDA) also called a palmtop is a small handy mobile device that provides computing functionalities and information storage and retrieval capacities for business or personal use [1]. Therefore, a smart phone can be used as a video player, a multimedia player, a game box, a navigator/GPS and also as a web tablet, alongside its primary role as a phone.

Given the rate at which more people use smart phones and devices, there's more need for mobile applications than ever before. Also, more software companies are trying to lay their hands on mobile application development in order to utilise this increase in the usage of smart phones and devices. Even websites today are now being designed with the ability to adapt to mobile architectures and work on mobile devices. The gaming industry is also not left out of this mobile revolution given the number of mobile games that are now available on the mobile technology market.

Because smart phones are also computers, they cannot be built without an operating system (OS). Therefore, giant software companies like Google, Apple and Microsoft as well as mobile technology companies like Nokia, Palm and Research in Motion (RIM) have been at the forefront of this mobile revolution. As an example, Apple's iPhones use iOS as their operating systems while phone giants like HTC, Samsung etc also use the Android OS which is Google's mobile platform.

Since it came out in 2007[2], Google's Android has grown to become the predominant operating system for mobile phones and devices. In a recent survey done by The Pew Internet and American Life Project on the use of smart phones [3], 35% of surveyed American adults said they personally have a smart phone. According to the study, out of those that own a smart phone, a whopping 35% use phones that have the Android OS, 24% use iPhones, 24% use Blackberry, 6% use Palm while 4% use Windows-based phones.

1.1 Android

Android is a mobile operating system developed by the Open Handset Alliance led by Google [2]. Based on the Linux kernel that includes middleware, libraries and key applications, it is a customised version of the Java programming language [2] [4].

Because it is open-source and relatively easy to learn and master, it has gained the support of developers to become the number one mobile operating system today. It affords programmers the freedom to develop various kinds of applications. It also gives unparalleled access to both first and third party applications which means more in-depth integration between components in varied programs and software sharing and reuse.

Some of the major features of the Android architecture include the Application framework, the Dalvik virtual machine, optimized graphics, SQLite, media support, integrated browser and support for GSM telephony, Bluetooth, WiFi, Camera, GPS and others [4].The architecture is as shown in Figure 1.1 below.

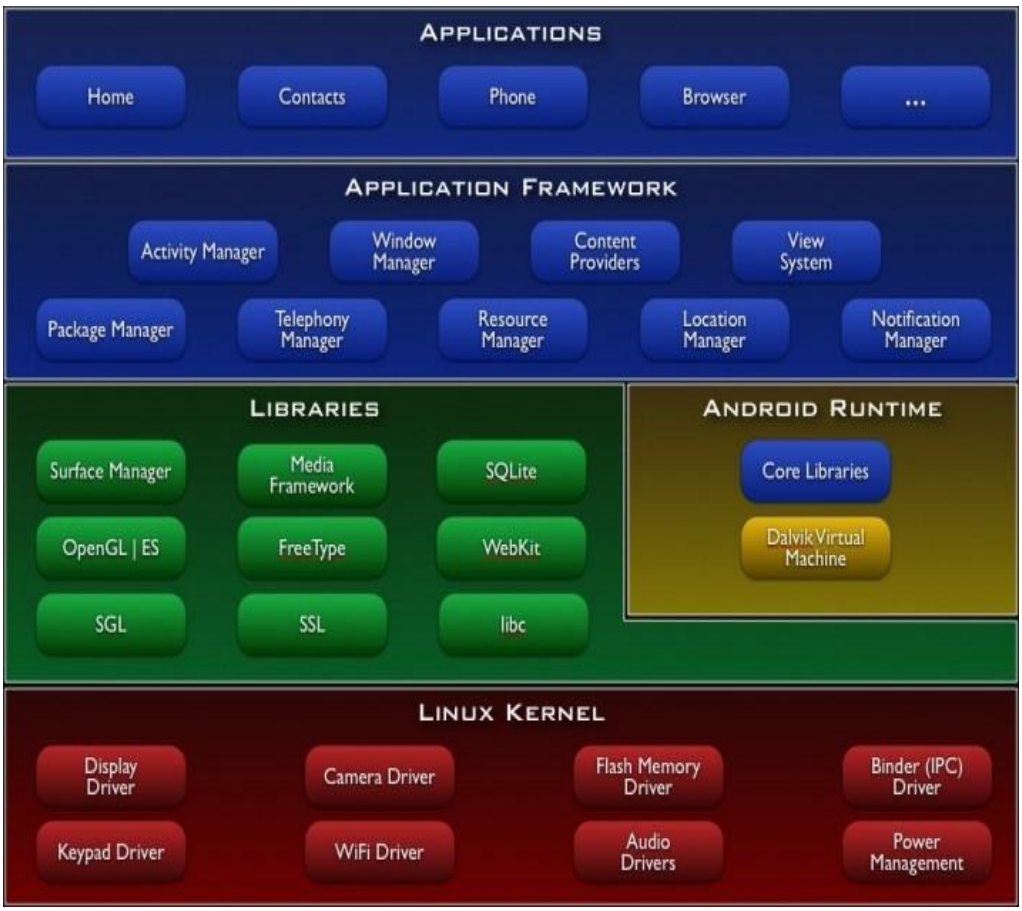

Figure 1.1 – Android Architecture [4]

1.2 Project Objective

The objective of this project is to design and implement a multi-user and multi-device file syncing application for Android. A research into the availability of such an application already released shows that there are a few and limited number of such applications on the Android market. Also, a few Android applications were targeted at single users trying to sync files between multiple devices. However, this project will allow multiple users to share files to multiple devices. In other words, this project would serve the purpose of enabling collaboration by more than one user of android devices through file sharing.

To put it in simple terms, this application seeks to achieve the following:
(i)A user of the app will be able to create a box whereby files can be uploaded
(ii)The creator of the box can add other users to the box so that they can have access to the box and its content
(iii) Everybody in the group has access to the box and its constituent files
(iv) Any of the users in the group can open any of the files, modify it and re-upload it to the shared box
(v) Everybody in the group gets alerted whenever there is an update to the box.

1.3 Some Issues on the Proposed Application

Since it will be used mostly by non-technical users, it is important for the application to be very user-friendly. As such, it should have a good graphical user interface (GUI) that a user can easily understand and use. In order to ensure this, it should follow the Android developer's application user interface guidelines. This would enable it to be in line with a user's previous experience of the Android platform.

Because the application is targeted at multiple users wanting to share and collaborate on files, it is good to have some form of identification that would provide security and privacy to user's documents. Users who have been given access to a box would be able to download and open any of the files, modify it and re-upload it to the shared box under a different name so as to give a kind of versioning that would enable users to revert to older copies if they so desire.

The ability to create more than one box would be very beneficial to users because it would afford them the freedom of having different sets of files for different purposes and different sets of people having access to them. As the files would end up being replicated on a server, it is very much likely that some form of limit will be set on the size of files allowed to be uploaded into a box given the fact that we are using a service with limited free resources.

1.4 Alternative Application

The major alternative Android applications to this application are Dropbox and SugarSync. With Dropbox and SugarSync, it is also possible to sync files between multiple devices and users. However, neither of them has an alert system that notifies the users of a box of any change to the box or its content. This is where this application, ShareBox, has an edge as it comes with a good alert system.

2 DESIGN AND DEVELOPMENT APPROACH

2.1 Requirements

After carefully reviewing the project objective and the issues on the proposition, a list of the requirements for the application were arrived at. These requirements are broadly divided into two categories which are the functional requirements and the non-functional requirements. A functional requirement is a requirement that defines specific functions or behaviour while a non-functional requirement is the one that specifies the criteria that can be used to judge how the application operates as opposed to its specific behaviour [5]. Both requirements are listed below.

2.1.1 Application's Functional Requirements

(i) It should be able to find other android devices

(ii) It must have a Graphical User Interface

(iii) It must be able to duplicate files

(iv) It must be able to duplicate to the server

(v) It must be built with push updates

(vi) It must allow the sharing of boxes with others

(vii) It must be built with context menus

(viii) It must have logins for users

2.1.2 Application's Non-functional Requirements

(i) It must have scalability i.e. the application can be expanded to support larger number of users without reducing its performance.

2.2 System Architecture

An Android application is a collection of one or more classes called *activities*. An activity is a single, focused thing a user can do. As an example, a user might want to create a folder. To achieve this in android, a Create folder activity is written. An android application usually has several activities. Even though activities usually work together to make an application, they are, in the real sense of it, independent of each other.

To communicate between activities and the OS, Android makes use of *intents*. Intent is composed of data and an action that it needs to perform e.g. edit, view etc. The action is generally the action to be performed when the initial intent is received while the data is the

data to operate on. Intents are used to start activities and to communicate among various parts of the Android system [6].

The design diagram for the client side of the application is shown in Figure 2.1 below. All the possible paths of the application together with the activities used for the different functions are shown on the diagram.

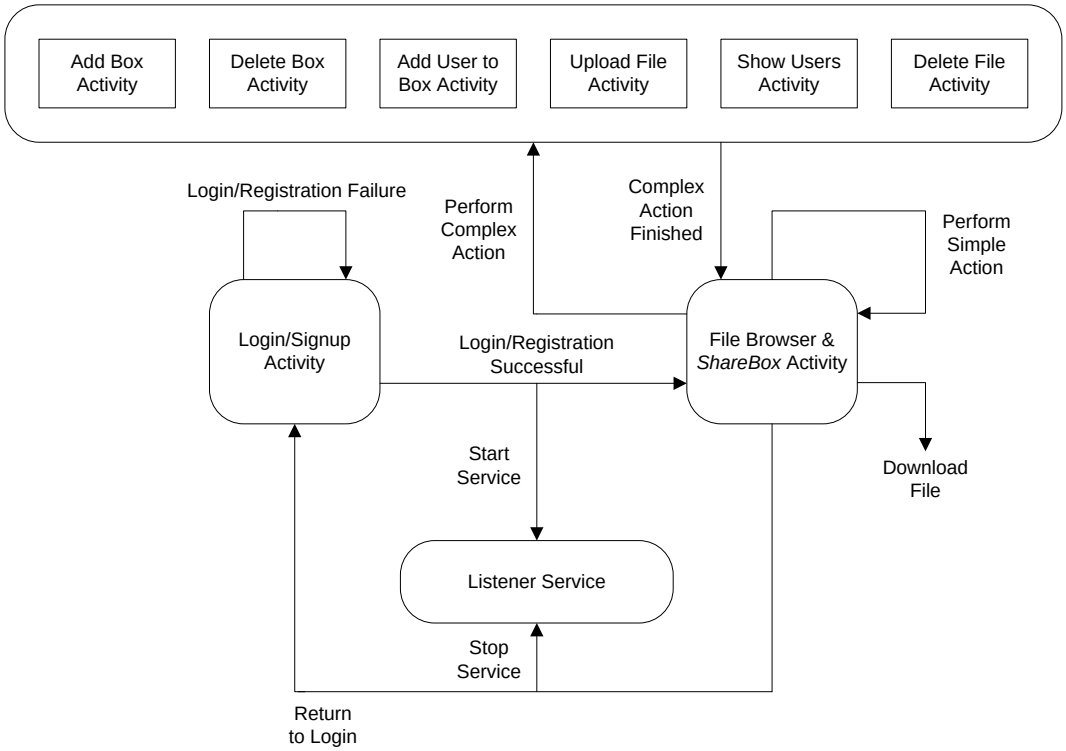

Figure 2.1 – The Client Side of the Application

The design for the server side of the application is as shown in Figure 2.2 below.

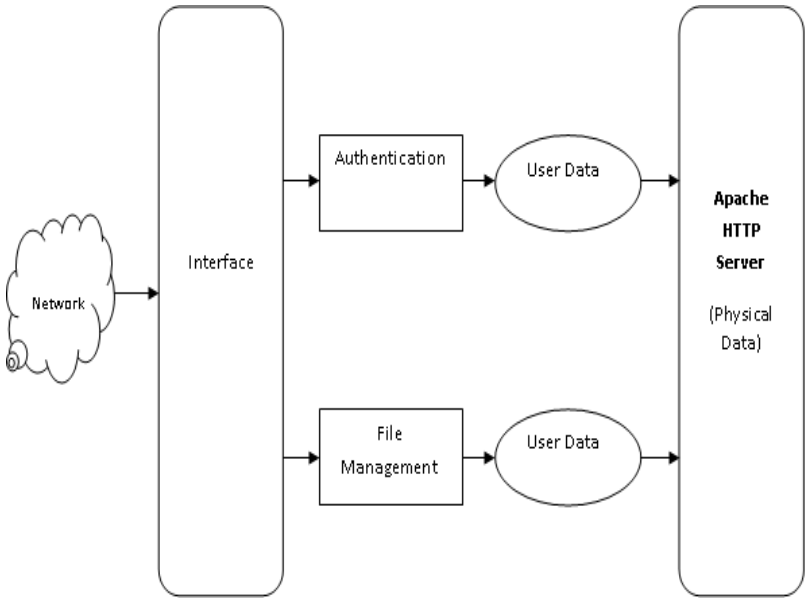

Figure 2.2 – The Server Side of the Application

2.2.1 Simple Case Scenario

A typical case scenario of how the application is used is as in the steps below and the Figure 2.3 that follows.

(i)A user tries to login or sign up into ShareBox

(ii)The user is granted access to the application

(iii)The user decides to open one of the boxes

(iv)The user uploads a file into the box

(v)The user tries to download a file and the application prompts the user to locate where the file would be downloaded to

(vi)User then attempts to leave ShareBox

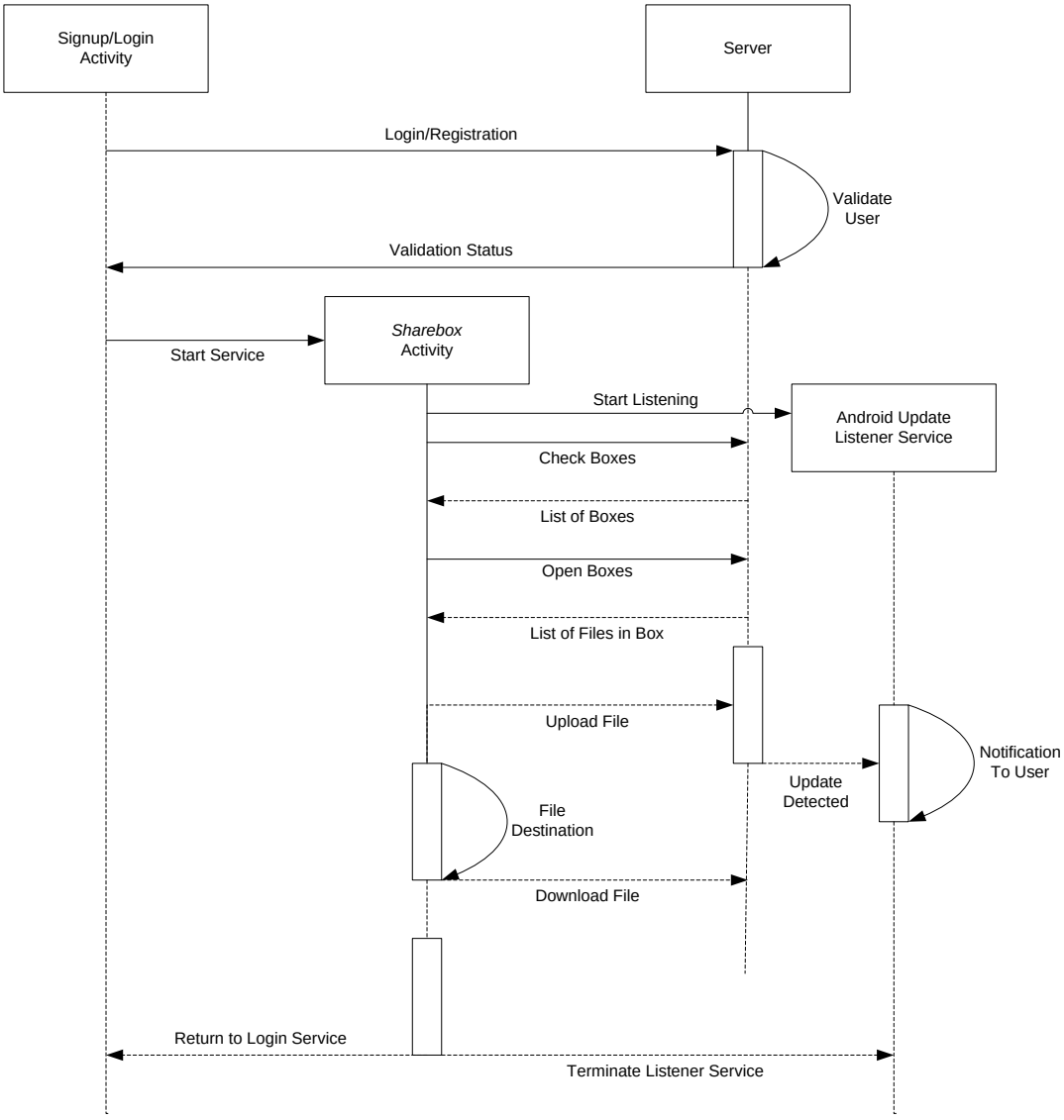

Figure 2.3 – A Typical Sequence for Interaction between Client and Server

2.3 Android SDK

The Android SDK is a Java based software development kit that enables developers to build applications for the Android platform. It includes development tools, source codes for sample projects, an emulator and the necessary libraries to create Android applications. Applications are run on Dalvik which is a custom-built virtual machine that runs on top of a Linux kernel [7].

The graphical user interface of an Android application is built from the libraries in the SDK using XML and the graphical input. This ensures that the scheme created by the developer is uniform for the application and between applications. In an Android application the user

interface is created using View and ViewGroup objects. View objects are the fundamental units of user interface expression on the Android platform [8].

On Android, there are some permissions that are required when trying to perform certain tasks. If for instance the application that is being built requires access to the internet to function, the developer must define it in the configuration file of the application that it would like to use the concerned permission. When a user wants to install an Android application, he or she gets a screen whereby the required permissions for the application must be confirmed.

Android has been updated severally since its release. The newer versions always contain a number of updates each fixing some bugs and adding new features. Each version is developed under a code name e.g. the first version released, version 1.0, was also known as Cupcake. This was followed by Donut, Eclair, Froyo, Gingerbread, Honeycomb and Ice Cream Sandwich [9]. In order to ensure the best compatibility with Android devices in the market now, this application has been built on the Froyo (Version 2.2).

2.4 Server Side Technologies

For the server side of this application, a generic web host is required for the storage of data and interaction with the client. For this purpose, one can either use a private server or rent a server from a hosting company. If the server is rented, the setup and maintenance of the server are done by the hosting company thereby decreasing the complexity and time spent trying to enable the server to run the required software. There are numerous hosts that are available. Some of them charge fees for their services while there are some free ones too. The hosts usually have many options depending on the services required from them.

Many of the free hosting sites often times require placements of adverts on web pages. This might be a drawback especially for an application that needs to connect to pages in order to retrieve data simply because with such an advertisement link placed in between, it might be difficult for the application to really understand the response it receives and therefore might not function as expected. Also, free hosts usually have restrictions on the free services they provide and would charge some form of payment if extra services are required.

With private servers on the other hand, these problems are taken care of but at the expense of setup cost. To run a private server having a database, time is required to setup the server and

to install the database. In case anything goes wrong, there is no assurance of backup support. However, if everything is fine, the quality of service would most likely be higher when compared to a rented server. To modify the requirements e.g. extra storage space or services, again, more cost would be incurred.

For the purpose of this project, a private Apache http web server that runs a MySQL database has been used.

2.5 Data Structures for transmitting data

Because this project requires the transfer of information between the client and server, it is very important to make the right decision as to which data structure technology to use. Data structure technology might affect the data transmission time between the client and server. It can also affect the load which an application exerts on the devices on which it runs. This obviously would affect the longevity of the battery on such devices. There are some options available as to the data structure to use for an android application.

The first technology which is capable of storing data in an organised way is the XML. This device and platform independent technology affords the developer a hierarchical structure in a way that is both machine and human readable. XML makes use of tags to define relations between objects and their corresponding contents. It does this by building XML schema which is a file that provides the ability to define the type and some other constraints that an element can have in a flexible way.

Another option that could be used as a data structure technology for data transmission is JSON. Simply put, JSON is built on a collection of name/value pairs which are in the form of arrays. It has a very simple structure which is also easy to implement. Despite its simplicity however, it has some issues. With JSON, all data must be parsed into strings before they could be sent. This format is not always appropriate when it comes to some information. Also, because of the repeated parsing of data involved, a lot of strain computationally is mounted on the devices on which the application runs thereby leading to increase in the time required to process actions and also by consequence affecting the battery life of the devices involved.

The last option is the Protocol buffer from Google. Its structure is built in C++, Python and Java. Even though it has an advantage over XML and JSON as far as binary data and positional binding are concerned, it has the problem of not being human-readable.

For this project, the XML option was chosen because Android has in-built XML libraries which process the responses from the server and XML data is easier to generate on the server.

3 IMPLEMENTATION

3.1 Eclipse

One of the requirements for developing a software project like this one is an integrated development environment (IDE). IDEs make life a lot easier for software developers simply because everything required for running codes are already integrated in them. There are many different IDEs available some of which are free and some of which are at a fee. This project would need an IDE that was targeted primarily at Java. This is where Eclipse comes in. For android developers, Google provides plugins for Eclipse in order to have access to easy integration for the development of Android applications. Apart from that, Eclipse also comes as a free multi-platform IDE which provides easy portability and compatibility. Figure 3.1 below shows the Eclipse IDE with Google Plugins

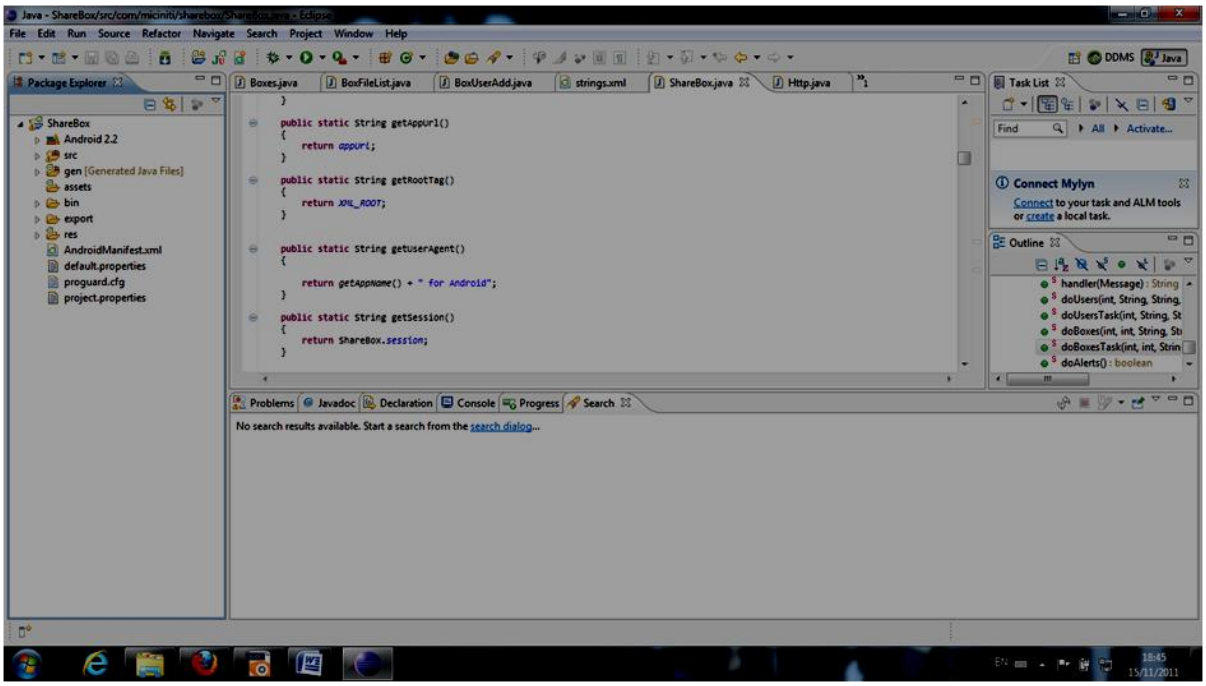

Figure 3.1 – The Eclipse SDK with Plugins for Android

3.2 Android Virtual Machine

The Android Virtual Machine is an emulator which runs on the computer. Android applications could be installed on the virtual machine to see how the application would run on a real android device. With Google's plugins for eclipse, using the emulator is a lot easier because as soon as an application is run on the eclipse, the emulator is prompted to open and

when it's ready, the exact replica of how the application behaves on real android phones is displayed. The only difference between the emulator and a real android phone is that phone calls cannot be made with it and also some services like the Android Market are absent on it. The emulator is as shown in figure 3.2 below.

Figure 3.2 – The Android Virtual Machine

3.3 Structure of the Project on Eclipse

After having created the project ShareBox in eclipse, the figures 3.3 and 3.4 below show the project and the constituent top-level folders as they appear in the package explorer. In the figure 3.3, under the first folder i.e. ShareBox (which is the name of the project as created on Eclipse), one can easily see Android 2.2 and the other folders in the project. The details of the folders are as follows.

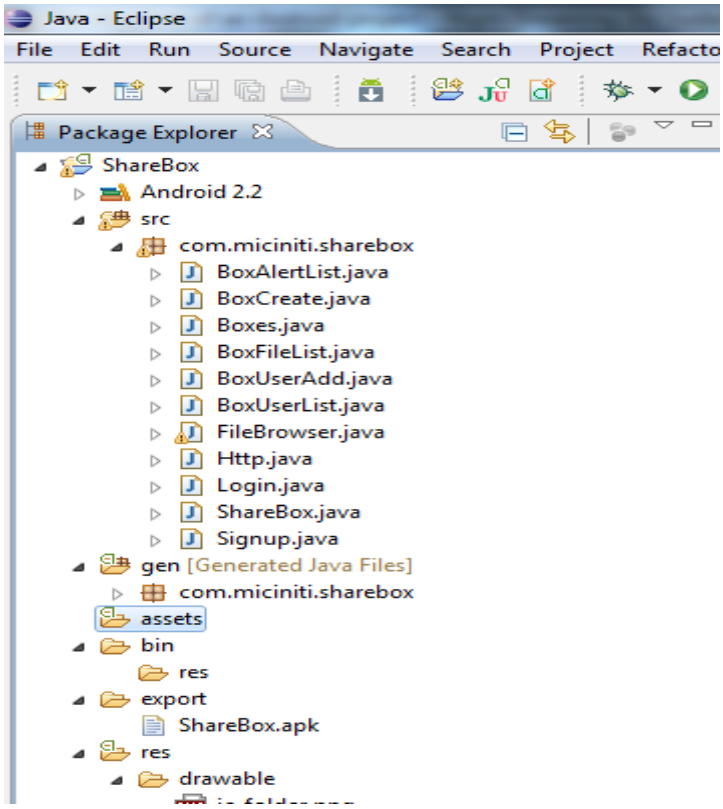

Figure 3.3 – The First View of the Package Explorer for ShareBox

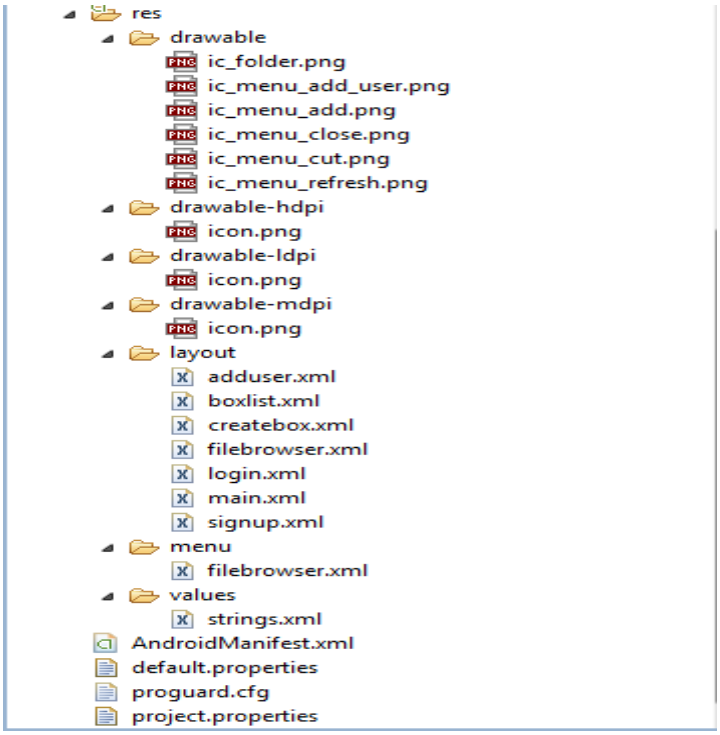

Figure 3.4 – The Second View of the Package Explorer for ShareBox

3.3.1 /Android 2.2 Directory

The folder shows that this project was created for the version 2.2 of Android. It contains the libraries (android.jar) that are needed for the project.

3.3.2 /src

This folder usually contains the Java source files that a developer is creating or has created for the project. In the figure 3.3, you can see the classes created for the project named ShareBox.

3.3.3 /gen

This folder is also a source folder. However, it usually contains Java source files that are generated automatically by the android platform. The most important of these files is the R.java file.

3.3.4 /assets

Like the /res folder, this folder also contains external resources used in the application but it's different from the /res folder in that the resources stored here are stored in raw format i.e. raw asset files. For this project, the folder was not used at all and it is empty.

3.3.5 /res

This important folder contains the external resources like images and strings that are used by the android application. The contents of these resources are usually referenced in the android application. This directory has 7 subdirectories which are:

(i) drawable
(ii) drawable-hdpi
(iii) drawable-ldpi
(iv) drawable-mdpi
(v) layout
(vi) menu
(vii) values

3.3.6 /res/drawable

This is the folder that contains the .png icons used for the project. As could be seen from the above figures, some of these icons include ic_folder.png, ic_menu_add_user.png etc.

3.3.7 /res/drawable-hdpi

This is the folder that contains the ShareBox icon used in the application for high-resolution screens. The qualifier “hdpi” refers to high-density screens [6].

3.3.8 /res/drawable-ldpi

This is the folder that contains the ShareBox icon used in the application for low-resolution screens. The qualifier “ldpi” refers to low-density screens. This is the same as the res/drawable-hdpi sub-directory except that the bitmap stored here is compiled into low-resolution drawable resources.

3.3.9 /res/drawable-mdpi

This is the folder that contains the ShareBox icon used in the application for medium-resolution screens. The qualifier “mdpi” refers to medium-density screens. This is the same as the res/drawable-hdpi sub-directory except that the bitmap stored here is compiled into medium-resolution drawable resources.

3.3.10 /res/layout

This is the folder that contains XML files that were used to define the user interface layout. From the figure 3.4, it could be seen that the files under this folder include adduser.xml, boxlist.xml, createbox.xml, filebrowser.xml, login.xml, main.xml and signup.xml.

3.3.11 /res/menu

This folder usually contains XML files that are used to represent application menus. For this project, it contains *filebrowser.xml* which was used to represent the menus of this application.

3.3.12 /res/values

This folder also contains XML files but as opposed to the XML files in the layout sub-directory used to define a single resource based on the names of the files, the XML files here are used to define multiple resources for many uses. For this project, the /values directory contains the strings.xml file which is used for string values and accessed using Android’s R.strings class.

3.3.13 AndroidManifest.xml

The package explorer also has under it, the *AndroidManifest.xml*, which is a very important file for any Android project. This XML file contains information about the Android application. It has information on various services, activities etc used for the application. It is also inside this file that permissions needed to run this application are defined. For this

application, two permissions are defined. These are android.permission.INTERNET and android.permission.WRITE_EXTERNAL_STORAGE. This means that for this application internet is required and data would be written to an external storage.

3.4 Server Side of the Application

The server part of this project was basically written by making use of the HttpURLConnection class which is simply a URLConnection for HTTP which is used to send and receive data over the web [10]. When using this class, it is important that the following steps are followed:

(a) First, a new HttpURLConnection is obtained by calling URL.openConnection() and then casting the result to HttpURLConnection

(b) Next, the request is prepared

(c) To upload a request body, instances must be configured with setDoOutput(true). This makes use of POST. However, if data is to be transmitted i.e. if data is being requested from the server e.g. to download a file, the stream returned by getOutputStream() must be written to. This on the other hand makes use of GET.

(d) Next, the response is read. The body of the response may be read from the stream returned by getInputStream(). However, if the response does not have a body, the method returns an empty stream.

(e) Lastly, it is very important to close HttpURLConnection by calling disconnect().This ensures that resources held by a connection are released so that they could be reused or closed.

The above steps were followed for this project. Http GET was used in all the client's requests to the server except for the file upload request which definitely requires POST. The server imposes a limit of 1024KB on the size of files that could be uploaded to a box. The class written to achieve this server side of the application was the *Http.java* which is as shown below.

```
package com.miciniti.sharebox;

import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
```

```
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.params.HttpConnectionParams;
import org.apache.http.params.HttpParams;
import org.apache.http.HttpResponse;
import org.apache.http.HttpStatus;
import org.w3c.dom.Document;

import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;

public class Http
{
    private static final String TAG = "HttpTask" ;

    private Document          doc    = null;
    private InputStream               is           = null;
      private DataOutputStream        dos    = null;

    private HttpClient                httpClient        = null;
    private HttpResponse              httpResponse      = null;

    public String                           responseError     = null;;
    public String                           responseMsg       = null;
    public int                              responseCode      = 0;

    private boolean                   parse                     = false;
    private boolean                         uploadMode        = false;

      private Handler                       handler = null;

      private String                        file   = null;

    public Http(Handler handler)
    {
      this.handler = handler;
    }

      public Document getDOM()
      {
            return doc;
      }

      public String getResponse()
      {
            return this.responseMsg;
      }

      public String getError()
      {
            return this.responseError;
      }


      public int getCode()
      {
            return this.responseCode;
      }
```

```
    public void setParse(boolean _parse)
    {
      parse = _parse;
    }

      public void setFile(String file)
      {
            this.file = file;
      }

      public void setUploadMode(boolean mode)
      {
            this.uploadMode = mode;
      }

    public boolean Connect(String urls, String post)
    {
        HttpURLConnection     con = null;
            boolean                 ret = false;

            Log.e(TAG, "url: " + urls);


            responseError = "";
            try
            {
                  httpClient = ShareBox.getHttpClient();

            postMessage("Connecting...");

                  if(uploadMode)
                  {
                        URL         connectURL;
                        String            lineEnd           = "\r\n";
                        String            twoHyphens   = "--";
                        String            boundary     = "*****";

                        FileInputStream fileInputStream = null;

                        File rFile = new File(this.file);
                        if (rFile.exists() && rFile.canRead())
                        {
                              fileInputStream = new FileInputStream(rFile);
                        }
                        else
                        {
                              responseError = "File not found / file not
readable";
                              return false;
                        }

                        connectURL = new URL(urls);
                        HttpURLConnection conn =
(HttpURLConnection)connectURL.openConnection();
                        conn.setDoInput(true);
                        conn.setDoOutput(true);
                        conn.setUseCaches(false);

                        conn.setRequestMethod("POST");
                        conn.setRequestProperty("Connection", "Keep-
Alive");
                        conn.setRequestProperty("Content-Type",
"multipart/form-data;boundary=" + boundary);
                  conn.setRequestProperty("User-Agent",
ShareBox.getUserAgent());

                        dos = new DataOutputStream(conn.getOutputStream());

                        dos.writeBytes(twoHyphens + boundary + lineEnd);
```

```
                        dos.writeBytes("Content-Disposition:form-
data;name=\"sharefile\";filename=\"" + this.file + "\"" + lineEnd);

                        dos.writeBytes(lineEnd);

                        int   bytesAvailable    =
fileInputStream.available();
                        int maxBufferSize = 1024;
                        int bufferSize          = Math.min(bytesAvailable,
maxBufferSize);

                        byte[] buffer           = new byte[bufferSize];

                        long total = rFile.length();
                        long count = 0;
                        int bytesRead = fileInputStream.read(buffer, 0,
bufferSize);

                        while(bytesRead > 0)
                        {
                              dos.write(buffer, 0, bufferSize);

                              bytesAvailable = fileInputStream.available();
                              bufferSize = Math.min(bytesAvailable,
maxBufferSize);

                              bytesRead = fileInputStream.read(buffer, 0,
bufferSize);
                              bufferSize = bytesRead;

                        postMessage("Uploading " + (count/1024) + "kB of "
+ (total / 1024) + "kB...");

                        }

                        dos.writeBytes(lineEnd);
                        dos.writeBytes(twoHyphens + boundary + twoHyphens +
lineEnd);

                        fileInputStream.close();
                        dos.flush();
                  responseCode = conn.getResponseCode();
                        is = conn.getInputStream();
                  }
                  else
                  {
                        HttpGet request = new HttpGet(urls);

                    HttpParams params = request.getParams();

                    HttpConnectionParams.setSoTimeout(params,
ShareBox.HTTP_TIMEOUT);
                    HttpConnectionParams.setConnectionTimeout(params,
ShareBox.HTTP_TIMEOUT);
                    request.setParams(params);


                        request.setHeader("Content-Type", "text/xml;
charset=utf-8");
                        request.setHeader("Content-Language", "en-US");

//                  request.setHeader("IF-Modified-Since", date.toString());

                  request.setHeader("Connection", "Keep-Alive");
                  request.setHeader("User-Agent", ShareBox.getUserAgent());

                  httpResponse = httpClient.execute(request);
```

```
                    responseCode =
httpResponse.getStatusLine().getStatusCode();
                    is = httpResponse.getEntity().getContent();
                    }

              if (responseCode == HttpStatus.SC_OK)
              {
                    postMessage("Loading...");

                          if(parse)
                          {
                                ret = parse2DOM(is);
                          }
                          else
                          {
                          if(file == null || uploadMode)
                          {
                                responseMsg = readData(is);
                                      Log.i("Response", responseMsg);
                                ret = true;
                          }
                          else
                          {
                                Log.d(TAG, "get file : " + this.file);

                                FileOutputStream fos = null;

                                Log.d(TAG, "New file: " + this.file);

                                      File rFile = new File(this.file);
                                      rFile.delete();
                                      if(!rFile.createNewFile())
                                      {
                                            responseError = "File already
exists " + this.file;
                                            ret = false;
                                      }
                                      else
                                      {

                                      Log.d(TAG, "Writing file: " +
this.file);

                                      fos = new FileOutputStream(rFile);

                                            int bufferSize    = 1024;

                                            byte[] buffer     = new
byte[bufferSize];

                                            int count = 0;
                                            int bytesRead;

                                            do
                                            {
                                                  bytesRead = is.read(buffer,
0, bufferSize);
                                                  if(bytesRead > 0)
                                                  {
                                                        Log.d(TAG, "size: " +
bytesRead);

                                                        fos.write(buffer, 0,
bytesRead);

      postMessage("Downloading " + (count/1024) + "kB");
                                                        count += bytesRead;
                                                  }
```

```
                                                }
                                                while(bytesRead > 0);

                                                fos.flush();
                                                fos.close();

                                        ret = true;
                                        }
                                }
                                }
                        }
                else
                {
                        responseError = "Invalid response (" +
httpResponse.getStatusLine().getStatusCode() + " : " +
httpResponse.getStatusLine().getReasonPhrase() + ")";
                    ret = false;
                }
                }
                catch(MalformedURLException ex)
                {
                ex.printStackTrace();
                        Log.e(TAG, "MalException" , ex);
                        responseError = "MalUrl Error (" + ex.getMessage() + ")";
                }
        catch (IOException e)
        {
                e.printStackTrace();
                        Log.e(TAG, "IOException" , e);
                        responseError = "IO Error (" + e.getMessage() + ")";
        }
                catch (Exception e)
                {
                        e.printStackTrace();
                        Log.e(TAG, "Exception" , e);
                        try
                        {
                                responseError = "Error (" + e.getMessage() + ")";
                        }
                        catch (Exception e1)
                        {

                                e1.printStackTrace();
                        }
                }
                finally
                {
                        try
                        {
                        if(is != null)
                        {
                                        is.close();
                                }
                          if(con != null)
                          {
                                con.disconnect();
                          }
                        }
                        catch (IOException e)
                        {
                                e.printStackTrace();
                                responseError = "Error on close (" + e.getMessage()
+ ")";
                        }
                }
                return ret;
        }

        public String readData(InputStream is) throws Exception
```

```
    {
            BufferedReader reader = new BufferedReader(new
InputStreamReader(is));

            String line;
            StringBuffer buffer = new StringBuffer();
      String NL = System.getProperty("line.separator");
            while((line = reader.readLine()) != null)
            {
                  buffer.append(line);
                  buffer.append(NL);
            }
            reader.close();

            return buffer.toString();
    }


      public boolean parse2DOM(InputStream in)
      {
            try
            {
                  DocumentBuilderFactory dbf =
DocumentBuilderFactory.newInstance();
                  DocumentBuilder db = dbf.newDocumentBuilder();

            doc = (Document) db.parse(in);

            if(doc != null) return true;
            }
          catch (IOException ioe)
          {
                  Log.e(TAG, "IO Exception" , ioe);
                  responseError = "IO Error (" + ioe.getMessage() + ")";
              doc = null;
          }
            catch (Exception e)
            {
                  Log.e(TAG, "Exception" , e);
                  responseError = "Error (" + e.getMessage() + ")";
              doc = null;
            }
            return false;
      }

      public void setHandler(Handler _handler)
      {
            this.handler = _handler;
      }


      public void postMessage(String str)
      {
            postMessage(0, "msg", str);
      }

      public void postMessage(int what, String str)
      {
            postMessage(what, "msg", str);
      }

      public void postMessage(String key, String str)
      {
            postMessage(0, key, str);
      }

      public void postMessage(int what, String key, String str)
      {
            if(handler == null) return;
```

```
            Message msg = handler.obtainMessage();

            Bundle b = new Bundle();
            b.putString(key, str);
            msg.setData(b);
            msg.what = what;
            handler.sendMessage(msg);
      }

}
```

3.5 Client Side of the Application

The client side of the application is made up of the main class which is *ShareBox.java* and other classes. These other classes are *Signup.java*, *Login.java*, *Boxes.java*, *BoxCreate.java*, *BoxUserAdd.java*, *BoxUserList.java*, *BoxFileList.java*, *FileBrowser.java* and *BoxAlertList.java*. The first activity implemented was the Signup/Login activity. This is normally the first page that a user sees once they launch the application on their phones. This activity was built using two different classes which are *Signup.java* and *Login.java*. The Signup requires a new user to enter two pieces of information which are username and password. The user is actually prompted to enter the password twice for confirmation. The Signup class was written such that it is compulsory for a user to enter at least 6 characters for both the username and password. It is also important that the user enters the same password twice. Otherwise, the application returns with a password mismatch error or alert.

The *Login.java* class was used for the other part of this first Signup/Login activity. It was written to authenticate a returning user of the application. It also made use of the same error checking approach used by the *Signup.java* class. Here is the segment of the *Signup.java* class where the authentication described above is done

```
public void doSignup()
      {
            String signupMsg = null;
            String userPass2;

            userName = signupName.getText().toString();
            userPass = signupPass.getText().toString();
            userPass2 = signupPass2.getText().toString();

            if (userName.equals("") || userName.length() < 6)
            {
                  signupMsg = "Please enter the user name (6 chars. min)";
                  signupName.requestFocus();
            }
            else if (userPass.equals("") || userPass.length() < 4)
            {
                  signupMsg = "Please enter the password (6 chars. min)";
                  signupPass.requestFocus();
```

```
            }
            else if (userPass2.equals("") || userPass2.length() < 4)
            {
                  signupMsg = "Please enter the password (6 chars. min)";
                  signupPass2.requestFocus();
            }
            else if (!userPass.equals(userPass2))
            {
                  signupMsg = "Password mismatch!";
                  signupPass2.requestFocus();
            }
            else
            {
                  progressDialog.setMessage("Starting...");
                  progressDialog.show();

                  signupThread = new Thread(this);
                  signupThread.start();
            }

            if (signupMsg != null)
            {
                  toast.setText(signupMsg);
                  toast.show();
            }
      }
```

Also for the *Login.java* class, the authentication of the user is done with this code segment:

```
public void doLogin()
      {
            String      loginMsg = null;

            userName = loginName.getText().toString();
            userPass = loginPass.getText().toString();

            if (userName.equals("") || userName.length() < 6)
            {
                  loginMsg = "Please enter the user name (6 chars. min)";
                  loginName.requestFocus();
            }
            else if (userPass.equals("") || userPass.length() < 4)
            {
                  loginMsg = "Please enter the password (6 chars. min)";
                  loginPass.requestFocus();
            }
            else
            {
                  if              (userName.equals("appdemo")              &&
userPass.equals("appdemo"))
                  {
                        doReady();
                        return;
                  }
                  else
                  {
                        progressDialog.setMessage("Starting...");
                        progressDialog.show();

                        loginThread = new Thread(this);
                        loginThread.start();
                  }
            }

            if (loginMsg != null)
            {
                  toast.setText(loginMsg);
                  toast.show();
```

```
            }
        }
```

The Signup/Login page is as shown in the Figure 3.5 below.The user can login right on that page.

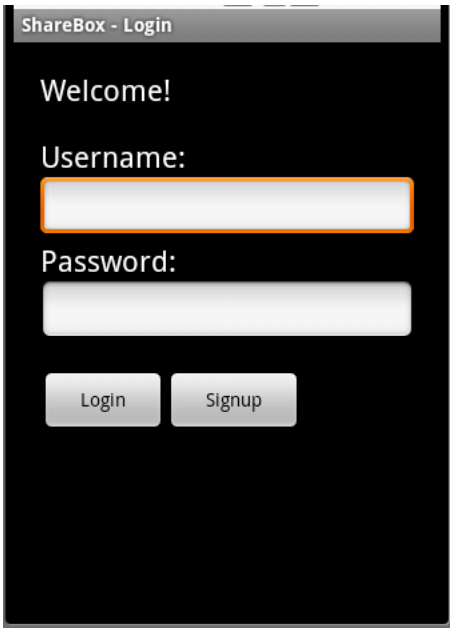

Figure 3.5 – Login/Signup Page for *ShareBox*

After the user, on the other hand, has clicked the Signup button, the Signup page comes up and it is as shown in figure 3.6 below.

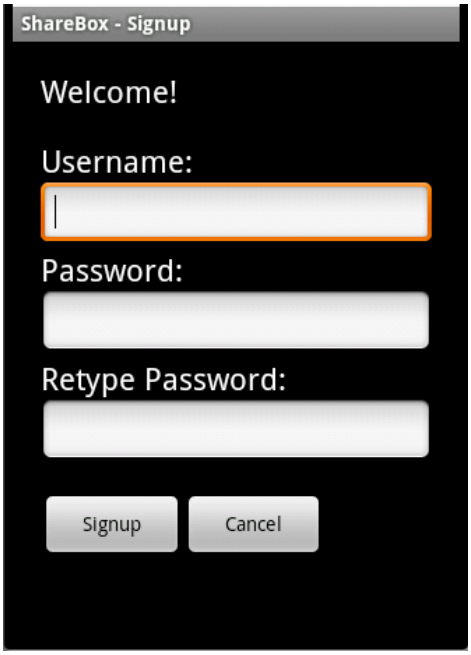

Figure 3.6 – Signup Page for *ShareBox*

Once the user is logged in, the next activity which they would want to do is to create a box. This is taken care of by the *BoxCreate.java* class. To create a box, all a user needs to do is to tap the menu button on the phone and a bar showing the options possible for the user is shown. This is as shown in the figure 3.7 below.

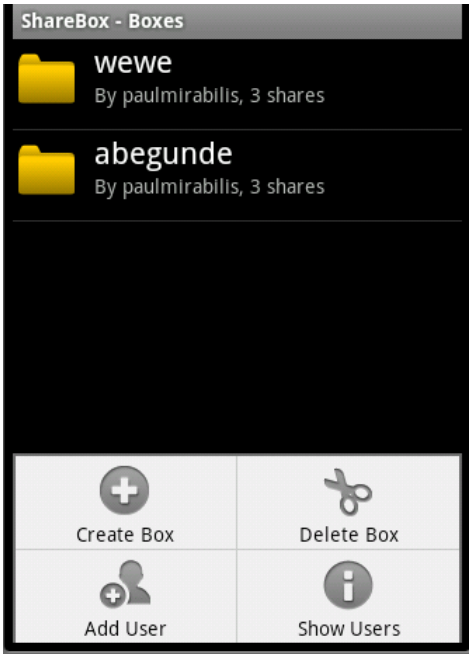

Figure 3.7 – Page Showing boxes and menu bar

Once the user clicks the Create Box button, the next page that comes up is the one shown in figure 3.8 below.

Figure 3.8 – Page for Box Creation

The user can also select the Add User button as shown in the menu in order to add a user that already exists on the ShareBox database so as to give them rights to one of the boxes belonging to the user. This functionality is taken care of by the *BoxUserAdd.java* class. To add a user to a box, the page looks like that in Figure 3.9 below.

Once the user is in a box, they would be able to see the files present in the box. At this point also, another set of menu buttons are shown when the menu button is tapped as shown in figure 3.10. If a user clicks the Show Users button, they get a page that looks like that in Figure 3.11 showing the users added to the box. In order to take care of this functionality i.e. to show the users of a particular box, the class *BoxUserList.java* was written. To display the list of files in a box as well as the users that uploaded them as shown in Figure 3.10, the class *BoxFileList.java* was written.

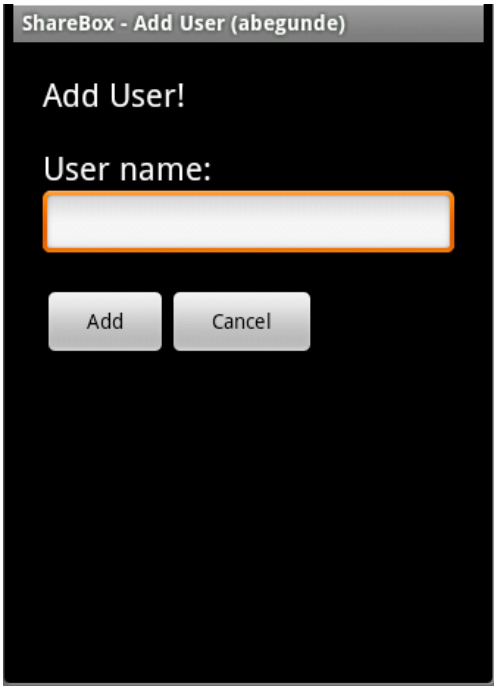

Figure 3.9 – Page showing how to add a user to a box

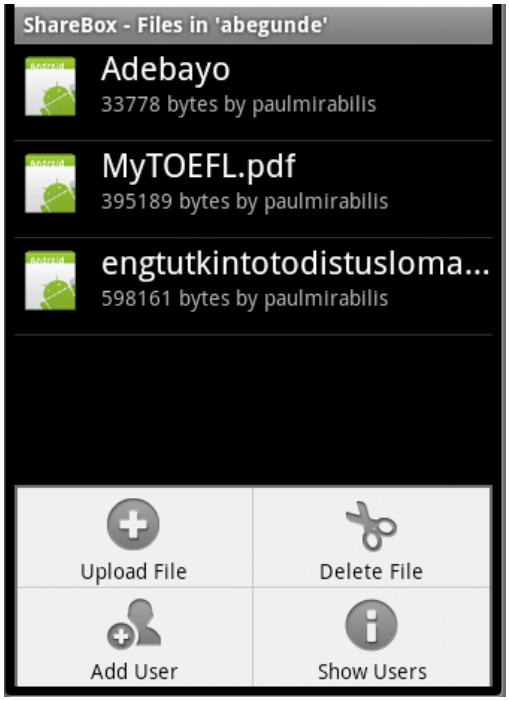

Figure 3.10 – Page Showing Files in a Box and menu bar

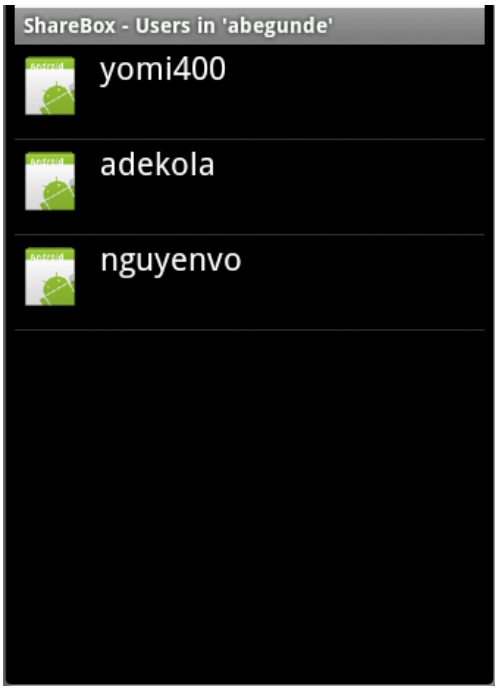

Figure 3.11 – Page Showing Users of a Box

The actions of the *BoxCreate.java*, *BoxUserAdd.java*, *BoxUserList.java* and *BoxFileList.java* classes are all coordinated using the *Boxes.java* class. Also, in order to perform some actions on the files e.g. upload file, delete file, download file etc, the class *FileBrowser.java* was written. For example, Figure 3.12 shows what the page looks like when a file is long-pressed.

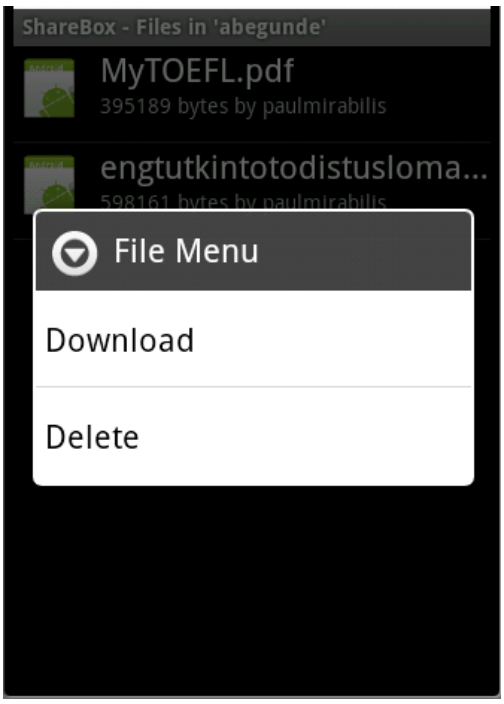

Figure 3.12 – Page showing what happens when a file is long-pressed

For the purpose of alerting users of a particular box of any new change or update to the content of the box, the class *BoxAlertList.java* was written. This class is very important because its function is one of the reasons why this application is unique in a way when compared to similar applications on the Android Market. This application was designed to use push notification for the alerts. There are three possible ways to implement a push notification system. These are poll notification, SMS method and Persistent TCP/IP method.

The SMS method usually comes with a cost both to the developer and the user. The Persistent TCP/IP method, on the other hand, has some reliability issues. It was therefore decided at the beginning of this project that the poll notification system is more suitable for the purpose of this application simply because it is easy to implement and solves the problems associated with the other methods mentioned above. The application requests update alerts from the server at regular intervals. The code for this class is as shown below.

```
package com.miciniti.sharebox;

import com.miciniti.sharebox.R;
import com.miciniti.sharebox.ShareBox.ShareAlerts;
import android.app.ListActivity;
import android.os.Bundle;
import android.content.Context;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

public class BoxAlertList extends ListActivity
{
      String TAG = "AlertsTask";

    public static final int MENU_ITEM_DELETE    = Menu.FIRST + 3;
    public static final int MENU_ITEM_CLEAR     = Menu.FIRST + 4;

    public Toast              toast;
    private int                     alertPos;

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

        setTitle(getTitle() + " Alerts");

        setDefaultKeyMode(DEFAULT_KEYS_SHORTCUT);

            toast = Toast.makeText(this, "", Toast.LENGTH_LONG);

        refresh();

        ShareBox.alertList = this;

    }

    @Override
      protected void onDestroy()
    {
            super.onDestroy();
            ShareBox.alertList = null;
      }

      @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        super.onCreateOptionsMenu(menu);

        menu.add(0, MENU_ITEM_CLEAR, 0, R.string.menu_clear_alerts)
              .setShortcut('2', 'c')
              .setIcon(R.drawable.ic_menu_cut);

        return true;
    }

      @Override
    public boolean onOptionsItemSelected(MenuItem item)
    {
      alertPos = getSelectedItemPosition();

        switch (item.getItemId())
```

```
        {
              case MENU_ITEM_DELETE:
                  if(alertPos >= 0 && alertPos < ShareBox.files.size())
                  {
                        ShareBox.alerts.remove(alertPos);

                        refresh();
                  }
                  return true;

              case MENU_ITEM_CLEAR:
                  ShareBox.alerts.clear();

                  refresh();
                        return true;
        }
        return super.onOptionsItemSelected(item);
    }

      @Override
      protected void onListItemClick(ListView l, View v, int position, long
id)
      {

            super.onListItemClick(l, v, position, id);
            if(position >= 0 && position < ShareBox.alerts.size())
            {
                  ShareAlerts alert = ShareBox.alerts.get(position);
                  toast.setText(alert.getMsg());
                  toast.show();
            }
      }

      public class AlertAdapter extends BaseAdapter
      {
          public AlertAdapter(Context c)
          {
          }

          public int getCount()
          {
              return ShareBox.alerts.size();
          }

          public Object getItem(int position)
          {
              return null;
          }

          public long getItemId(int position)
          {
              return 0;
          }

            public View getView(int position, View convertView, ViewGroup
parent)
            {
                  View row = convertView;

                  if (row == null)
                  {
                        LayoutInflater              inflater              =
BoxAlertList.this.getLayoutInflater();
                        row = inflater.inflate(R.layout.boxlist, null);
                  }

                  ShareAlerts alert = ShareBox.alerts.get(position);

                  TextView    text = (TextView)row.findViewById(R.id.text);
```

```
                TextView    info = (TextView)row.findViewById(R.id.info);


                text.setText(alert.getMsg());
                info.setText(alert.getTime());
                return(row);
            }

        }

        public void refresh()
        {
            setListAdapter(new AlertAdapter(BoxAlertList.this));
        }

        public void update()
        {
            this.runOnUiThread(new Runnable()
            {
                @Override
                public void run()
                {
                    refresh();
                }
            });
        }

}
```

Finally, the main class i.e. *ShareBox.java* is the class used to coordinate the server side of the application with all the classes of the client side.

The XML files for all the pages of the application earlier explained are defined under the res folder of the project on Eclipse. Also, it is important to mention that the URL of the web server used for the project was referenced in the *strings.xml* file which is located in the *values* folder under the *res* folder.

4 TESTING THE APPLICATION

In order to be able to ensure that the set objectives for a project are achieved, it is very important to test it thoroughly. It is imperative that this testing is done at each stage of the implementation i.e. testing each of the constituent parts before being incorporated into the final system. The testing carried out for this project are categorised into three basic subheadings. These are testing of the Server Side, testing of the Client Side and real world testing.

4.1 Testing the Server Side

Testing the server side was simple because of the nature of the architecture of the server. As stated earlier, the server side involved methods connecting with servlets. Each of the methods used for particular actions was made to be a singular entity which was encapsulated with the

other methods. It was therefore kind of easy to test each of the methods individually before connecting to the whole system.

In order to test the individual methods, a class was used to call the method so as to ensure that the output is as expected. Also, since most of the actions affect the MySQL database and the data stored on the server, it was easy to see the outputs and changes to the database tables. The outputs of methods written to return data were also easily verified by comparing the returned values to the expected values. The correctness of the connection between the Java methods and servlets were also confirmed using an internet connection via a browser to test and also by doing some debugging.

It was also important to test the integrity of the data requested from server and the data input or uploaded to the server. Because MySQL is a relational database, this was easy to check within the database system. The figure 4.1 below shows how the server looks with the boxes and files as uploaded by users.

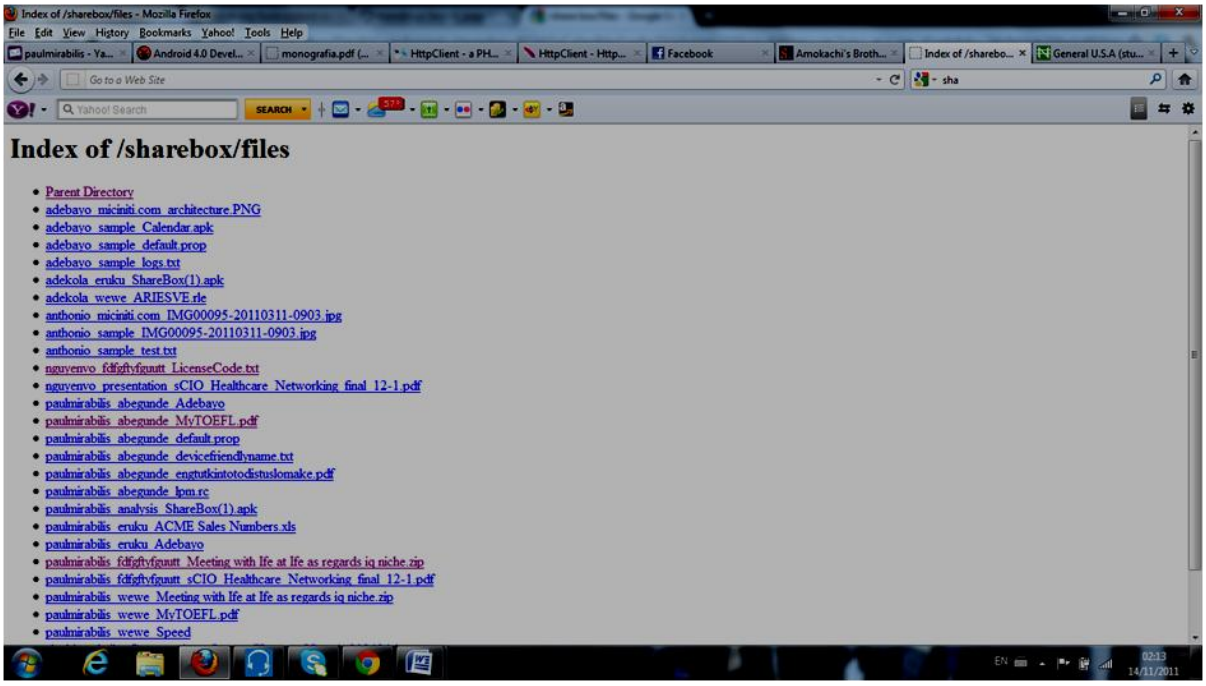

Figure 4.1 – The MySQL database view showing boxes, files and users who have rights to them.

4.2 Testing the Client Side

This part of the testing was more complex than that of the server described above. The constituent components were built separately and then later incorporated into the whole system. For the Client Side of the application, the testing was in two stages.

The first stage of the test was carried out during the development of the constituent components. Each component was first built as a new project before being copied into the overall project. Testing under this stage involved calling methods within each component and watching the end output. Since most of the client actions require communication with the server side of the application, it was decided that allowing the components the freedom to access the actual server was important. It was therefore important to make available all the services from the server part required by a component of the client side before writing the component itself and this explains why a greater percentage of the server was built before starting the client side.

Incorporating each component of the client part into the project and then testing the overall system was what the second stage of the testing was all about. The Dalvik Virtual Machine that comes with the android SDK takes care of this part. Running the application prompts the virtual machine to come up and then it was very easy to see how the application functions on the emulator which is a simulation of how the application behaves on real android devices. Even though this test stage was a lot easier, it was more time consuming than the first stage because this is where the behaviour of the application is actually ascertained.

4.3 Testing in the Real World

The last form of testing of any application is usually the real world testing. This application was also made to pass through this stage by giving at least three android phone users the .apk file of the application to install on their devices. Each of the testers ran and used the application to check how it actually behaves when used on real devices by end users. The testers were tasked to check the usability of the application and also suggest any necessary change deemed fit by them. The testing also involved trying to break into the application. For example, trying to log into the application with wrong password or attempting to add a user that was non-existent on the database of the application.

5 CONCLUSION

5.1 Evaluation

Having thoroughly followed the development lifecycle of a normal software project from conception of ideas to design and implementation to testing, this project has achieved its set objectives. This can only be ascertained by revisiting the suggested requirements for the project described earlier on and comparing them with the functionalities that the final application is capable of providing the user.

The first requirement stated was that the application must be able to find other android devices. This was confirmed to have been achieved by the application while it was tested on many devices during the real world testing stage. Also, the application has been built with an interface which is very easy for any user to follow, thereby fulfilling its requirement of having a graphical user interface. The application has also successfully achieved the third and fourth requirements because not only can it duplicate files on the physical android device, it also duplicates to the server.

The application comes with update notification or alert system which sends the users of a box alerts in case any change or update happens to the box and its content. This fulfils the fifth requirement. With the application, users have been able to share their boxes and files with their friends and colleagues without any problem. The application also comes with context menu which allows users to do some extended functionalities like showing the users of a box, uploading files etc. The login and signup part of the application have been built in a user-friendly and secured way, thereby achieving the last functional requirement.

The issue of scalability which is the only non-functional requirement stated at the beginning of the project, has also been taken care of by the application. This is because the server used for the application is one that the developer has control over and also, the application has been built in such a way that will allow expansions in the future.

In all, the application has been able to fulfil its requirements and successfully achieved all its objectives.

5.2 Challenges

To build this kind of software application requires overcoming a lot of challenges and bottlenecks. The first major challenge encountered in the course of developing the application was learning how to program Android. Despite the fact that Android code has the same syntax as Java, it is worthy of note that there are some differences as to the libraries present in the two and so a lot of adjustments had to be made. However, because of the availability of numerous helpful tutorial websites and links coupled with a lot of books that have already been written on the platform, the challenge of learning the platform was overcome.

Other challenges faced while developing the application included creating the user interface, parsing the XML response from the server and file transfer between the server and client.

5.3 Possible Future Expansions

While the application has achieved the objectives set for it at the start of the project, it is important to say that there is room for other additions in the future. Some of the possible features that could be added to the application in the future include adding or implementing a quota on users, virus scanning of files being uploaded and searching and filtering of files.

In conclusion, this project has been an eye-opener to a lot of technologies as far as information technology and software development is concerned. The experience which the project has afforded is a really enjoyable and worthwhile one. It would be a thing of joy to see people work on expanding the project in the future.

REFERENCES

[1]SearchMobileComputing. Personal digital assistant (pda). Website.
http://searchmobilecomputing.techtarget.com/definition/personal-digital-assistant

[2] Wikipedia Foundation. Android (operating system). Website.
http://en.wikipedia.org/wiki/Android_%28operating_system%29#cite_note-google_code-6

[3] Pew Research Centre. Smartphone Adoption and Usage. Website.
http://pewinternet.org/Reports/2011/Smartphones/Summary/Key-Findings.aspx

[4] Android developers. What is Android? Website.
http://developer.android.com/guide/basics/what-is-android.html

[5] Wikipedia Foundation. Non-functional requirement. Website.
http://en.wikipedia.org/wiki/Non-Functional_Requirements

[6] Donn Felker. 2011. Android Application Development For Dummies. Wiley.

[7] Webopedia Foundation. Android SDK. Website.
http://www.webopedia.com/TERM/A/Android_SDK.html

[8] Android developers. User Interface. Website.
http://developer.android.com/guide/topics/ui/index.html

[9] Wikipedia Foundation. List of Android operating system versions. Website.
http://en.wikipedia.org/wiki/Android_version_history

[10] Android developers. HttpURLConnection. Website.
http://developer.android.com/reference/java/net/HttpURLConnection.html