DESIGN AND IMPLEMENTATION OF A FILE SHARING APPLICATION FOR ANDROID

[1]Alatishe A.A, [2]Adegbola M.A, [3]Dike U. Ike

[1,2,3]Department of Electrical and Information Engineering, Covenant University, Ota Ogun State, Nigeria.

ABSTRACT: Over the last few years, there has been a drastic change in information technology. This includes the various ways in which files can be shared and stored. Android OS is a relatively new mobile OS which has been steadily taking over more and more market share. Easy to use, easy to develop for, and open-source, it has picked up a following of developers who want to create content for the masses. Cloud computing is publicized as the next major step for all forms of typical information technology use. From businesses, to non-profit organisations, to single users, there seems to be various applications which can use cloud computing to offer better, faster, and smarter computing. This paper aims to combine the two, building a cloud based application for Android, offering users the power of cloud computing in the palm of their hand.

Keywords- Cloud computing, Android, Android SDK, Information Technology.

I INTRODUCTION

File sharing is the practice of distributing or providing access to digitally stored information, such as computer programs, multimedia (audio, images and video), documents, or electronic books. It may be implemented through a variety of ways.

Android is a relatively new mobile operating system developed by Google and the Open Handset Alliance. [1] Officially released in October 2008, it has revolutionized mobile application development due to the fact that it is open source. It allows developers unparalleled freedoms to create varied and interesting applications. Based on the Java programming language, it is touted as being easy to pick up and master, whilst the underlying is a modified Linux kernel. Some of Android's biggest draws for developers include the relative simplicity of developing using Java syntax, which means quickly producing applications. Also, Android provides easy yet secure access to first and third party applications, allowing deeper integration between components in different programs, and encourages software sharing and reuse. The user interface can be built quickly and simply through XML or graphically, and once an application has been finished it can be submitted to Android market, a portal through which developers can make their creations available to Android users, either free or for profit. [2]

Cloud computing has been viewed in several forms. There has been no single view that has decidedly become the obvious candidate; however there are some common elements between them all. The most glaring of these is that it is a form of distributed computing, in that distinctly separate systems link together to form a cloud. [3] Also, there is an idea of on-the-fly scalability, that machines can join and leave the cloud as required.

One definition of cloud computing is that of a pool of computational resources, linked together to provide a greater processing power. [4] These are projects which involve supporters installing software on computers at home, which connect, when idle, to their respective clouds over the internet and compute small parts of complex scientific calculations. Another use of the term is to provide some form of data syncing.

One more idea of cloud computing is that of peer-to-peer systems. This form has been used for many years for file sharing, recently implemented for services such as Skype. These services reduce load on their servers by passing data directly from user to user.

Java is a general-purpose, concurrent, class-based, object-orientedcomputer programming language that is specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that code that runs on one platform does not need to be recompiled to run on another. Java applications are typically compiled to bytecode (class file) that can run on any Java virtual machine (JVM) regardless of computer architecture. Java is, as of 2012, one of the most popular programming languages in use, particularly for client-server web applications, with a reported 10 million users. Java was originally developed by James Gosling at Sun Microsystems (which has since merged into Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them.[5]

The aim of this project is to design and implement a file sharing application for Android based devices. This project will allow multiple users to share files to multiple devices. This project would provide a stable platform

to enable collaboration through file sharing. To this end, files may be uploaded by one user and available to another, all simplified through an easy to use application on an Android device.

II DESIGN REQUIREMENTS

These are broken into functional requirements, which explain what the application should accomplish, and non-functional requirements, which explain or communicate how the application should work, listed in order of descending practical functionality and desirability.

Functional Requirements

1. Must have a graphic user interface (GUI)
2. Ability to find other handsets
3. Replication of files
4. Regular updates of content
5. Replication to server
6. Push updates
7. Individual boxes
8. Sharing boxes with friends
9. User logins
10. Content menus

Non Functional Requirements

1. Must be scalable
2. Must be accessible by as many android users as possible.

III SYSTEM ARCHITECTURE

Android applications are made up of classes called activities. Activities are classes which encapsulate the user interface and the program execution. An activity generally has a specific task to perform, and an application can be made of several activities, each performing part of the job of the application. For communication between activities and to the OS, Android provides intents. These are simple objects which carry information around the system, and can be used to start an activity, pass along application data, or request the OS to open a file. [6]

The application design diagram which shows possible paths through the application, and the activities used to act on the requested functions is shown below. As shown, should a user request to open a file, the application send a request to the Android OS to display the contents using a relevant program. Other actions such as uploading a file or saving one to the device start off activities internal to the application, but separate from the main thread of execution.

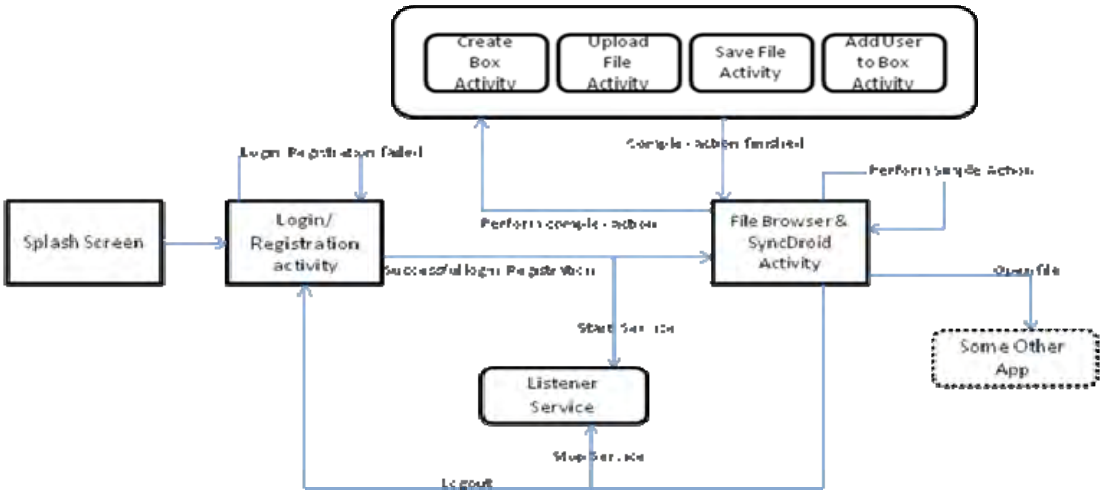

Figure 1.0 The Diagram for the Client Side Application

The next diagram shows the high level design for the server supporting the application. As can be seen, actions are broadly separated into two main functions: user authentication or file management and manipulation. These actions are separated to enable reuse of security functions by several different file management functions, to avoid recreation of code and possible security holes.

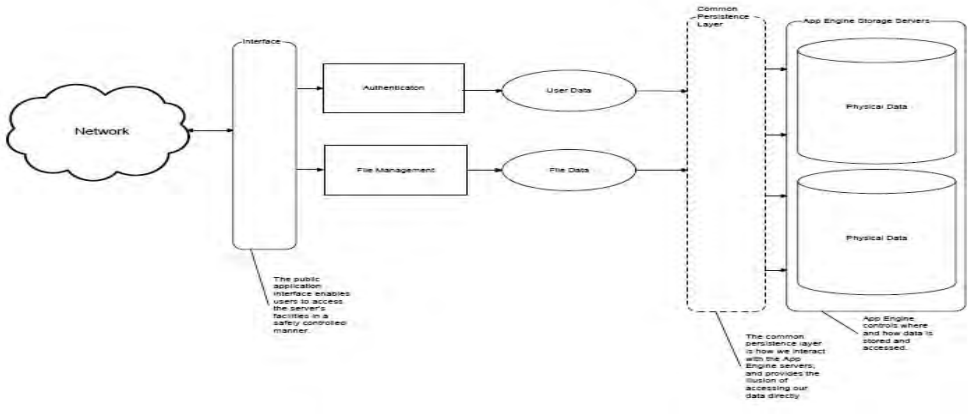

Figure 2.0 the Design Diagram of the Server Side Application

The UML diagram shows the design of the server side of the application. It demonstrates the interaction between classes and the breakup of the classes into the different roles as seen in the preceding server side diagram. The external API to the server side application is made up of a number of servlets, each performing a different action. These interact with the utility classes, which form the user and file management sections. Finally these manipulate the persist able Java objects, which are stored by the server on disk.

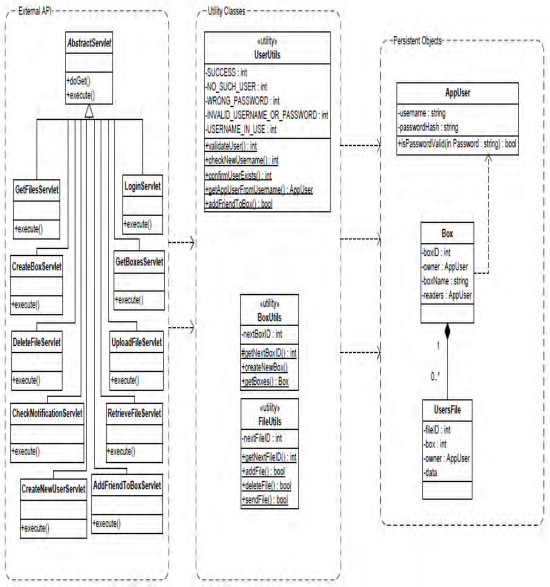

Fig 3.0 A UML Diagram of the Server Side Architecture

The following figure is a sequence diagram of interactions between the client and the server. It details the events that occur when a user requests a certain action, and the messages sent between the application and the server.

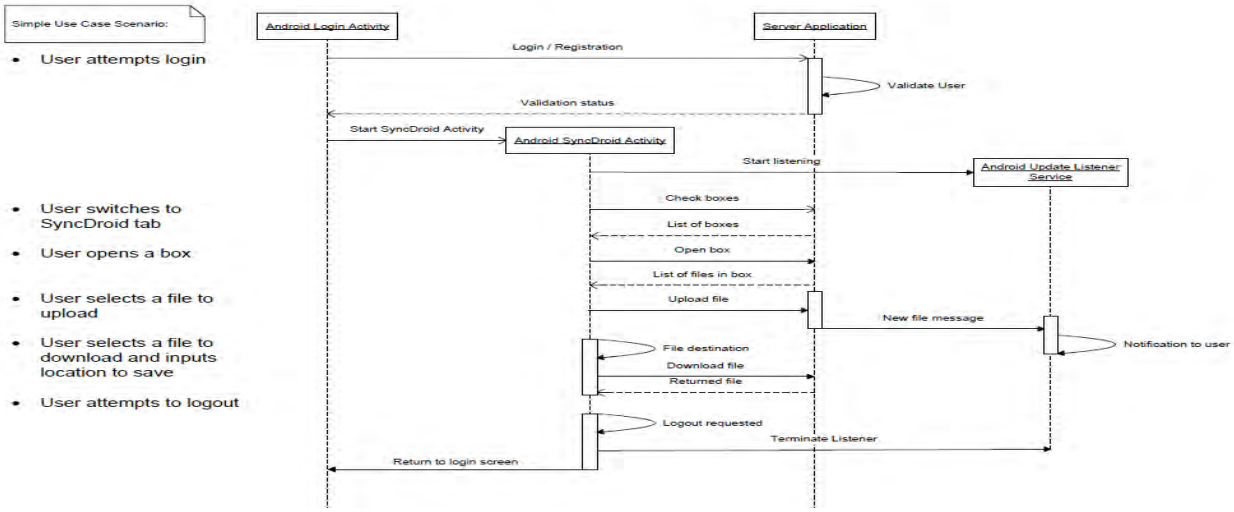

Fig 4.0 A Sequence Diagram showing interactions between the Client and the Server

IV DEVELOPMENT APPROACH

This will consider the technical options available to implement the project application. By discussing the merits and weaknesses of each approach, an informed decision can be made as to how to build a professional program. This chapter will focus on the Android SDK, possible technologies for server side support, and data structures for transmitting information.

Android SDK

The Android SDK is a Java based language, which enables developers to build powerful programs in a neat, object oriented design. Whilst it is a relatively compact development kit, it contains libraries for many varied functions, allowing access to virtually every component of the Android handset and operating system.

Whilst offering such unseen control over the handset, Android enforces rigorous security protocols on developers to protect users from unsavory or malicious code. Users are alerted to all required access to handset operational equipment on installation, and so are informed of such uses before accepting any software.

Androids graphical user interfaces are built from the libraries included in the SDK, using XML or graphical input. This enables developers to create a scheme that is consistent throughout the application, as well as between applications.

Android has been updated constantly since its release, reaching version 2.1 in under two years. As such new functions are available every few months, and development and maintenance of applications requires much more attention than longer lifecycle systems. To ensure maximum compatibility with handsets currently in the market, this project will be built on Android 1.5.

As this technology is a requirement for this project, there does not need to be an in depth review of alternative systems. As a matter of record, operating systems such as Windows Mobile, iPhone OS, Symbian or Palm OS could have been used, though most of these are not as open as Android (Symbian became open-source in February 2010)[7], neither are they based in such a widely studied language as Java.

Server Side Technologies

Using a generic web host to serve the storage and interaction with the client would give great freedom as to how to develop the application as a whole. This option could be run on a private machine, or rented from a hosting company. By renting a server, setup and maintenance are left to the supplier. This reduces complexity and time spent on enabling the machine to run the required software, for example Apache. Many hosts are available at competitive prices, with some even offering free hosting. Most have options available as to services required, including extra storage space, or some form of database, generally some favour of SQL.

However, free hosting sites often require placement of adverts on pages, such as click through links. This would pose a problem for an application simply connecting to pages to retrieve data. If such a link were placed in between, the application would not understand the response it received and so not react as expected. Also, by using third party services, only those offered are available. There can be restrictions on the services able to be provided from a given host. Often the required extras would require some form of payment.

By running a private machine as a web host, these issues are resolved, at the cost of ease of setup. Running a private server with a database involves time to setup the underlying server and installing the database. If something goes wrong there is no guaranteed external support. Nonetheless, once up and running the integrity and quality of service can be higher than an external server as it is maintained personally. Modifications can be

made to requirement, such as extra storage space or installation of any further required software. Again, this would cost even more money to run, especially if a computer were obtained or built simply for the purpose of being a dedicated server.

One large obstacle to either of these options is scalability. The application is expected to be released to the public through Android Market, and as such will be available to many thousands of users. Should reception be positive, and uptake high, many users would upload and store many bytes of data, sharing with many friends and this would therefore lead to many accesses

to the server at any given time. To allow for such an eventuality, the server side of the application should be scalable, so load could be balanced between several different machines serving the application, acting as one server.

This could easily be achieved using a cloud computing server. Such a server would provide a minimum level of service, and should there be an increase in requirements, further services may be made available. These requirements may include an increase in storage, in which case more disk space may be made accessible. If more processing power is needed to serve requests or more bandwidth to receive them, more machines can be loaded with the server application and the load balanced between them. Should the demand fall, resources can be released for use by other applications. All of this is done automatically, and is known as elastic cloud computing.

Data Structures for Data Transmission

The data structure and technology used to transfer information between client and server is a very important decision. It can affect the work load put on the devices to encode and decode data, which can affect battery life on mobile handsets. It would also alter transmission times, which would bear upon the quality of service to the user. One option of technology which could be used as a data structure for transmission is JSON. This is simply an array of key-value pairs. Data is parsed into strings, stored in the array, and transmitted as a single string. At the other end the string is parsed as an array, and then each pair must be parsed again into the required types.

This is a simple structure, which is relatively easy to implement. However, there are several issues with this format. First, all data to be sent must be parsed to strings; this is not always a suitable format for certain information.

Secondly, there is a computational strain put on the devices to parse data to and from strings. This can increase the time required to act on information, and use more battery, which reduces quality of service to the user.

XML is another technology capable of storing data in a structured manner. It provides a simple, hierarchical structure to information that is human readable. Using generic tags similar to HTML, it is extensible with user created tags to define relations between obaajects and their contents. This involves the use of some software to build the XML schema, a file which lays

out a definition of acceptable XML documents which could be used with an application.

Again, however, XML uses text to transfer data, which as stated is not always suitable for all data types. It is also not built for efficiency, which is a major concern when using mobile devices. A third option to consider is Google's own protocol buffers. They offer a data structure built natively in C++, Java or Python. For this project Java would make the most sense.

V DEVELOPMENT METHODOLOGY

The methodology used to develop an application can greatly affect the resulting product. A single run through of development can lead to a product which does not work correctly or does not cover requirements as expected. Conversely, too much time testing may result in an application which is not finished. The Waterfall method would provide a clear set of requirements up front, and reduce the amount of time not in implementation, so more focus could be put on development, therefore most likely leading to greater output of functionality. However, there are known weaknesses of this methodology, such as clients not having a fixed set of requirements, and issues with integration of components.

For this project, an agile process seemed to be best suited to development. As the supervisor could be seen as the client, requirements maybe checked with him. Weekly meetings provide a chance to update the client on activity and progress. This weekly cycle ensures that, should the project start heading in the wrong direction, it is caught early and can be corrected. This way, the project would provide a correct piece of software, in terms of the clients requests.

Eclipse

To develop a large scale application, an integrated development environment provides an easier way to manage and create software. There are many available, for free or for a fee, each offering different properties. For this project, an IDE aimed at Java would be most suitable. Google offers plugins for Eclipse to enable easier

integration for development, debugging and releasing for its Android and App Engine products. This IDE is available for free on all major platforms, providing easy portability between development hardware.

Android Virtual Machine

The Android virtual machine is an Android emulator which runs on a computer. Once an application is ready to be tested, it can be installed on the AVM, which acts almost like an actual hardware device. It is limited as it cannot make phone calls, and is restricted in other services such as Android Market, but it provides a next best alternative to see how an application might run on a real handset and enables developers and testers to interact with the software as a user might.

Server Side Application

The first stage of development involved building the server side application to which the handset application would connect. As seen in previous chapters, the server application classes were split into several groups: the external API, the management classes and the persist able objects.

First the objects had to be developed, which involved defining Java objects compatible with Google's JDO database system.

Client Side Application

The client application was originally planned to be a single Android activity with several pages. However, as the design process continued it became apparent that this was not the best approach to take. Android is designed in such a way that each activity should perform a specific action, or several actions, all based on the same functionality. It is encouraged for applications to be made up of several activities, each leading from one to the next.

The first activity to implement was the login activity. This was the first point at which a user has access to SyncDroid, the application. Users need the ability to login to an existing account, or register a new one. The activity was relatively simple to implement, as it required only two pieces of information to be entered, a username and password. It then connected via a GET command to the relevant servlet, to either check the credentials or create a new user. Once these steps were completed, it would start off the application's main class, and finish itself in the background.

As part of the method to start the main class, it also stored the user's credentials in a Shared Preferences object belonging to the application. This is Android's way of transferring application data between an application's activities safely and securely. This information is then used in methods in the main class to call other servlet functions.

The main class of the application, named SyncDroid.java, is where the user accesses most of their data. It acts as the central hub of the application, with other activities branching off to perform specific actions. When logged in, the SyncDroid class offers a view with two tabs. The first is a file browser of the users handset, with files stored locally. The second shows the boxes the user can access. By selecting a box, a user can view the files within it, and download them if they wish.

The file browser view is based on a tutorial from anddev.org,[8] with changes to enable it to function as required for the application. By selecting a directory, users can access them, and by selecting files, they can be uploaded. There is also a context menu for users to opt to open a file. This is achieved by simply giving the Android OS the URI of the data to be viewed, and requesting it opened. The system attempts to automatically work out the best way to open a file, and if it cannot work it out, offers the user a list of programs to select from to view the file in.

To display the list of boxes and files, the most effective way was to implement an adapter. This enabled the visual object, the List View, to accept the list of boxes or files, and display them on screen as a list of names. Hidden within the model was the other relevant data, such as the owners and the ids of the objects in question. Protocol buffers transferred the data to and from the client. When a user requested the list of files within a box, a request was sent including the box's id. In return the server sent back a protocol buffer containing a list of file names and ids. It did not contain the file data itself. This was to reduce the amount of network traffic in use, as the data would not necessarily be required at that stage. The file names would be displayed to the user, and if one was selected to be downloaded, this would start off the file download activity. Within this activity a protocol buffer would be received containing the file's name, id and data.

From the list of files a user again had the option of viewing a context menu, with the option of deleting a file. This would only be carried out if the user owned the file that is the user was the uploader of the file. Otherwise an error message would be displayed.

To ensure security, users were offered the option to logout of the application when the wanted to exit. This deleted their credentials from the system's memory and returned them to the login screen. This way, any other person to use the user's handset would not be able to gain unauthorized access to their Sync-Droid account.

For actions such as uploading, downloading and adding friends to a box, other activities would be launched.

Protocol Buffers

To send data between the client and the server application, protocol buffers were designed for the boxes and files. These enabled the use of objects to send data over the connection, which was preferred as this allowed direct manipulation of the data within the code, rather than having to externally pack and parse it.

The class used to define the protocol buffer for a box is shown below. It is very short and simple, only listing the instance variables of an object and their types. Each value defined as 'optional' is not required; therefore to build a Box-Message protocol buffer there is no obligation to fill in any fields.

The protocol buffer definition file is compiled by Protoc, which outputs a Java file. This class can be used to create protocol buffers as native Java objects, and manipulate them as such.

VI CONCLUSION

This project has followed the development cycle of a cloud based Android application, from inception of the idea through to implementation.

Recommendations

There are several expansions to the application which, while unnecessary under the requirements outlined previously, would offer greater functionality to the user and more depth in the application. Some of these are changes to existing code to increase functionality, whilst others are simply adding new components.

First, implementing a file splitting algorithm when uploading and downloading files. Currently, the application takes a whole file and sends it to the server for storage. This only allows for small files to be sent and stored.

By splitting larger files, these could also be stored, allowing the potential for large images and video to be stored and shared. Another extension would be to implement a quota on users. Currently there is no limit, and so users may upload as much as they want. Obviously, this is unsustainable due to the cost of storing data on the server. Further to this extension is another, developing levels of users. By offering a pay-for service with more storage space, users may opt to buy more or use a smaller amount for free.

Virus scanning of files being uploaded would be a useful extension to add. As so many files belonging to many users would be uploaded, scanning them would provide security to all users, as well as the application, by preventing viruses from being shared or stored on the server.

A final idea could be to offer searching and filtering of files. As users begin to own and gain access to several boxes, it is quite feasible they may lose track of where a given file is, or wish to see a list of all files of a given file type that they have access to. By offering some form of searching and filtering, they would be able to access this information.

Of course these are only a few ideas of extensions, there are many more ways to broaden the application and offer more functionality, depending on what is deemed desirable.

REFERENCES

[1] Android Dev. Team. Android Developers. Website. http://developer.android.coml. [Accessed: January 15, 2013].
[2] Wikipedia Foundation. Android (operating system).Website http://en.wikipedia.org/wiki/Android_(operating_system).[Accessed: January 15, 2013].
[3] Berkeley University. Seti@Home. Website http://setiathome.ssl.berkeley.edu/. [Accessed: January 15, 2013].
[4] Stanford University. Folding@Home. Website http://folding.stanford.edu/, [Accessed: January 15, 2013].
[5] Java, *https://en.wikipedia.org/wiki/Java_(programming_language)*.[Accessed: January 22 2013].
[6] Chris Haseman, Android Essentials. United States of America: Apress, First press 2008.
[7] Wikipedia Foundation. Symbian OS. Website. http://en.wikipedia.org/wiki/Symbian_OS. [Accessed: February 17, 2013].
[8] Nicolas Gramlich. Android File Browser V2.0. Website. http://www.anddev.org/android_filebrowser__v20-t101.html. [Accessed: April 3, 2013].