**EFFICIENCY OF MULTIMEDIA CLOUD COMPUTING TO SAVE SMART PHONE ENERGY**

**ABSTRACT**

In spite of the dramatic growth in the number of smart phones in recent years, the challenge of limited energy capacity of these devices has not been solved satisfactorily. However, in the era of cloud computing, the limitation on energy capacity can be eased off in an efficient way by offloading heavy tasks to the cloud. In this project, we evaluate the energy cost of multimedia applications on smart phones that are connected to Multimedia Cloud Computing (MCC). In the proposed system we conducted an extensive set of experiments to measure the energy costs to investigate whether or not smart phones save energy by using MCC services.

In proposed system we compared the energy costs for uploading and downloading a multimedia file to and from MCC with the energy costs of encoding the same multimedia file on a smart phone. We propose a novel method for storing a multimedia file like image, audio, video, text on the cloud to reduce amount of energy. Our results show that MCC provides the smart phones with much multimedia functionality and saves smart phone energy from 30% to 70%.

|  |  |  |
| --- | --- | --- |
|  | **TABLE OF CONTENTS** |  |
| **CHAPTER NO.** | **TITLE** | **PAGE NO.** |
|  | **ABSTRACT** | **v** |
|  | **LIST OF FIGURES** | **viii** |
|  | **LIST OF ABBREVIATIONS** | **xi** |
| **1** | I**NTRODUCTION** |  |
|  | 1.1 Purpose | 1 |
|  | 1.2 Scope | 1 |
|  | 1.3 Product Features | 2 |
|  | 1.4 Existing System And Its Limitation | 2 |
|  | 1.5 Proposed System | 2 |
| **2** | **SYSTEM STUDY AND ANALYSIS** |  |
|  | 2.1 Literature Review | 3 |
|  | 2.2 Problem Description | 8 |
|  | 2.3 Feasibility Study | 8 |
|  | 2.4 Hardware And Software | 9 |
|  | 2.5 Technologies Used | 9 |
| **3** | **SYSTEM DESIGN** |  |
|  | 3.1 System Architecture | 12 |
|  | 3.2 Use Case Diagram | 14 |
|  | 3.3 Class Diagram | 16 |
|  | 3.4 Activity Diagram | 18 |
|  | 3.5 Sequence Diagram | 20 |
|  | 3.6 Collaboration Diagram | 22 |
| **4** | **SYSTEM IMPLEMENTATION** |  |
|  | 4.1 Modules Overview | 24 |
|  | 4.2 Login For Cloud Storage | 24 |
|  | 4.3 Uploading Multimedia Files In Cloud | 25 |
|  | 4.4 Searching File Using Date Filter | 26 |
|  | 4.5 View Files On Cloud | 26 |
| **5** | **TESTING** |  |
|  | 5.1 Testing Overview | 27 |
|  | 5.2 Type Of Testing | 27 |
|  | 5.3 Testing Plan | 28 |
|  | 5.4 Test Cases | 30 |
| **6** | **CONCLUSION** | **31** |
| **7** | **FUTURE ENHANCEMENTS** | **32** |
| **8** | **APPENDIX 1** |  |
|  | 8.1 Sample Coding | 33 |
|  | 8.2 Screen Shots | 67 |
| **9** | **REFERENCES** | **76** |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **Fig. No.** | **TITLE** | **PAGE NO.** |
| 2.1 | Running Java Byte Code on the Android | 8 |
| 2.2 | Android Software Stack | 10 |
| 2.3 | Android Compilation Process | 11 |
| 3.1 | System Architecture | 12 |
| 3.2 | Use Case Diagram | 14 |
| 3.3 | Class Diagram | 16 |
| 3.4 | Activity Diagram | 18 |
| 3.5 | Sequence Diagram | 20 |
| 3.6 | Collaboration Diagram | 22 |
| 4.1 | Login for Cloud Storage | 24 |
| 4.2 | Uploading Multimedia files in Cloud | 25 |
| 4.3 | View files on Cloud | 26 |

**LIST OF ABBREVIATIONS**

|  |  |  |
| --- | --- | --- |
| AAPT | - | Android Application Program Tool |
| ADB | - | Android Debug Bridge |
| ADT | - | Android Development Tools |
| APK | - | Android Package Kit |
| AVD | - | Android virtual devices |
| DEX | - | Dalvik Executable Files |
| Dalvik VM | - | Dalvik Virtual Machine |
| GUI | - | Graphical User Interface |
| ICT | - | Information and Communications Technology |
| JFC | - | Java Foundation Classes |
| JITT | - | Just In Time Translator |
| MCC | - | Multimedia Cloud Computing |
| NDK | - | Native Development Kit |
| SDK | - | Software Development Kit |
| SQA | - | Software Quality Assurance |
| URL | - | Universal Resource Locator |

**CHAPTER 1**

**1 INTRODUCTION**

**1.1 PURPOSE**

Our Project focus on the way to design a multimedia based service system that provides a security for the user multimedia files and also provides real time efficient results for storing and retrieving data from the cloud storage.

**1.2 SCOPE**

In spite of the dramatic growth in the number of smart phones in recent years, the challenge of limited energy capacity of these devices has not been solved satisfactorily. In this project, we evaluate the energy cost of multimedia applications on smart phones that are connected to Multimedia Cloud Computing (MCC). In the proposed system we conducted an extensive set of experiments to measure the energy costs to investigate whether or not smart phones save energy by using MCC services.

In proposed system we compared the energy costs for uploading and downloading a multimedia file to and from MCC with the energy costs of encoding the same multimedia file on a smart phone. We propose a novel method for storing a multimedia file like image, audio, video, text on the cloud to reduce amount of energy. Our results show that MCC provides the smart phones with much multimedia functionality and saves smart phone energy from 30% to 70%.

**1.3** **PRODUCT FEATURES**

This multimedia based service can be used to upload and download the multimedia data to cloud storage and user can be able to view the files on cloud.

**1.4 EXISTING SYSTEM AND ITS LIMITATION**

* In existing system we can store multimedia files on the internal or external storage.
* The user can be able to view the content only if storage is available.
* If internal or external storage get crashed or erased we can’t retrieve the data.
* User cannot be able to retrieve data if internal or external memory gets erased or crashed.
* There is no security for the user data and anybody can view the data.
* If internal or external memory get excess user may not able to save data.
* More amounts of data on smart phones will reduce performance and energy of the smart phones.

**1.5 PROPOSED SYSTEM**

* In proposed system the user can be used to store data in cloud storage.
* The user can open the data anywhere because we are uploading data's in cloud storage.
* In proposed system we are using Offloading method for uploading data into cloud storage.
* The user can be able to search the file on cloud storage using data filter.
* In our proposed system user can able to store and retrieve data without the use of mobile storage.

**CHAPTER 2**

**2 SYSTEM STUDY AND ANALYSIS**

**2.1 LITERATURE REVIEW**

**2.1.1 An Analysis of Power Consumption in a Smartphone**

Mobile consumer-electronics devices, especially phones, are powered from batteries which are limited in size and therefore capacity. This implies that managing energy well is paramount in such devices.

We develop a power model of the free runner device and analyze the energy usage and battery lifetime under a number of usage patterns. We discuss the significance of the power drawn by various components, and identify the most promising areas to focus on for further improvements of power management. We also analyze the energy impact of dynamic voltage and frequency scaling of the device’s application processor.

* + 1. **Cloud Computing for Mobile Users: Can Offloading Computation Save Energy**

The cloud heralds a new era of computing where application services are provided through the Internet. Cloud computing can enhance the computing capability of mobile systems, but is it the ultimate solution for extending such systems' battery lifetime.

* + 1. **Balancing Energy in Processing, Storage, and Transport**

Network-based cloud computing is rapidly expanding as an alternative to conventional office-based computing. As cloud computing becomes more widespread, the energy consumption of the network and computing resources that underpin the cloud will grow. This is happening at a time when there is increasing attention being paid to the need to manage energy consumption across the entire Information and Communications Technology (ICT) sector. While data center energy use has received much attention recently, there has been less attention paid to the energy consumption of the transmission and switching networks that are key to connecting users to the cloud.

* + 1. **On Effective Offloading Services for Resource-Constrained Mobile Devices Running Heavier Mobile Internet Applications**

Rapid advances in wireless mobile network technologies and mobile handsets (MHs) facilitate ubiquitous infrastructure that can support a range of mobile services and applications in addition to conventional mobile Internet access. One recent trend is to effectively run desktop PC-oriented heavier applications on MHs. However, due to their miniature, portable size, MHs are resource-constrained and therefore, running these applications directly on an MH is not satisfactory given a user's expectations.

* + 1. **Using Bandwidth Data to Make Computation Offloading Decisions**

We present a framework for making computation offloading decisions in computational grid settings in which schedulers determine when to move parts of a computation to more capable resources to improve performance. Such schedulers must predict when an offloaded computation will outperform one that is local by forecasting the local cost (execution time for computing locally) and remote cost (execution time for computing remotely and transmission time for the input/output of the computation to/from the remote system).

* + 1. **Energy-Efficient Platform-as-a- Service Security Provisioning in the Cloud**

In this paper we present ENUAGE, a platform as a service security framework for provisioning secure and scalable multi-layered services based on the cloud computing model. ENUAGE encapsulates the security procedures, policies, and mechanisms in these security associations at the service development stage to form a collection of isolated and protected policy-driven security domains. An energy analysis of the system shows, via real energy measurements, major savings in energy consumption on the consumer devices as well as on the cloud servers.

* + 1. **Energy Efficiency in Data Centers and Cloud-Based Multimedia Services: An Overview and Future Directions**

The expanding scale and density of data centers has made their power consumption an imperative issue. Data center energy management has become of unprecedented importance not only from an economic perspective but also for environment conservation. This paper provides a comprehensive overview of the techniques and approaches in the fields of energy efficiency for data centers and large-scale multimedia services.

**2.2** **PROBLEM DESCRIPTION**

* User cannot be able to retrieve data if internal or external memory gets erased or crashed.
* There is no security for the user data and anybody can view the data.
* If internal or external memory get excess user may not able to save data.
* More amounts of data on smart phones will reduce performance and energy of the smart phones.

**2.3 FEASIBILITY STUDY**

All projects are feasible given unlimited resources and infinite time. It is both necessary and prudent to evaluate the feasibility of the project at the earliest possible time.

The following feasibility techniques has been used in this project

* + - * **Operational Feasibility**
      * **Technical Feasibility**
      * **Economic Feasibility**

**2.3.1 Operational Feasibility**:

Operational feasibility is necessary as it ensures the success of the project. Certain tests have been carried out to ensure the operational feasibility of the system. All the capabilities work well if proper environment is provided.

**2.3.2 Technical Feasibility**:

Technical feasibility analysis makes a comparison between the levels of technology available that is needed for the development of the project. The level of technology consists of factors like software tools, machine environment, platform developed and so on.

**2.3.3 Economic Feasibility:**

This is the most important part of the project. As far as our project is concerned various software involved in our project are available freeware in the internet and can be easily downloaded. Thus the risk of finance doesn’t exist and hence economically feasible.

**2.4 HARDWARE AND SOFTWARE SPECIFICATION**

**2.4.1 Hardware Requirements**

* Processor : Pentium P4
* Motherboard : Genuine Intel
* RAM : Min 1 GB
* Hard Disk : 80 GB
* Mobile Phone : Android 2.2

**2.4.2 Software Requirements**

* Operating system : Windows XP
* Technology Used : Android 2.2
* IDE : Eclipse
* Emulators : AVD
* Plug-in : ADT plug-in
* Tools used : Android SDK

**2.5 TECHNOLOGIES USED**

* + 1. **Java**

**JAVA:**

Java is Platform Independent. Java application is being used in this project since android application is being run with java application.

**SWING:**

To create a Java program with a Graphical User Interface (GUI), Swing is essential. The Swing toolkit includes a rich set of components for building GUIs and adding interactivity to Java applications. Hence in our project we use swing in order to provide better page layout.

**Swing GUI Components:**

The Swing toolkit includes a rich array of components: from basic components, such as buttons and check boxes, to rich and complex components, such as tables and text. In our project we use swing components for login details and to create various buttons for uploading text, image, audio and video files.

**Running Java Byte Code on the Android:**

The Open JDK makes use of native code so it would be a non-trivial port. There is at least one VM that is written in Java; unfortunately it is not a completely working implementation of Java. Since DalvikVM runs classes that were converted from .class files it should be possible to convert the classes over.

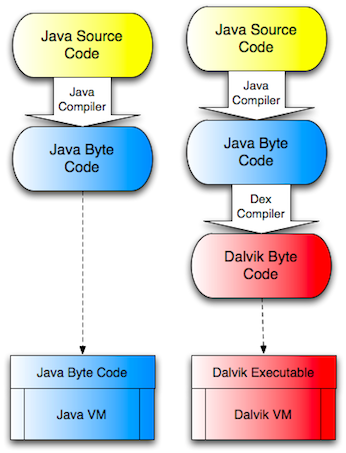
****

Fig 2.1 [Running Java Byte Code on the Android](http://stackoverflow.com/questions/670987/running-java-bytecode-on-the-android-sun-jvm-on-top-of-dalvikvm)

* + 1. **Android**

**2.5.2.1 Android Operating System**

Android is an operating system based on Linux with a Java programming interface. The Android Software Development Kit (Android SDK) provides all necessary tools to develop Android applications. This includes a compiler, debugger and a device emulator, as well as its own virtual machine to run Android programs. Our complete project runs in Android operating system.

**Android Development Tools**

**ANDROID:**

Android is a Linux-based open source platform.. Applications are developed in Android using a version of the Java programming language running on the Dalvik virtual machine.

**ANDROID SDK:**

The Android Software Development Kit (SDK) contains the necessary tools to create, compile and package Android application. Most of these tools are command line based.

The Android SDK also provides an Android device emulator, so that Android applications can be tested without a real Android phone. You can create Android virtual devices (AVD) via the Android SDK, which run in this emulator.

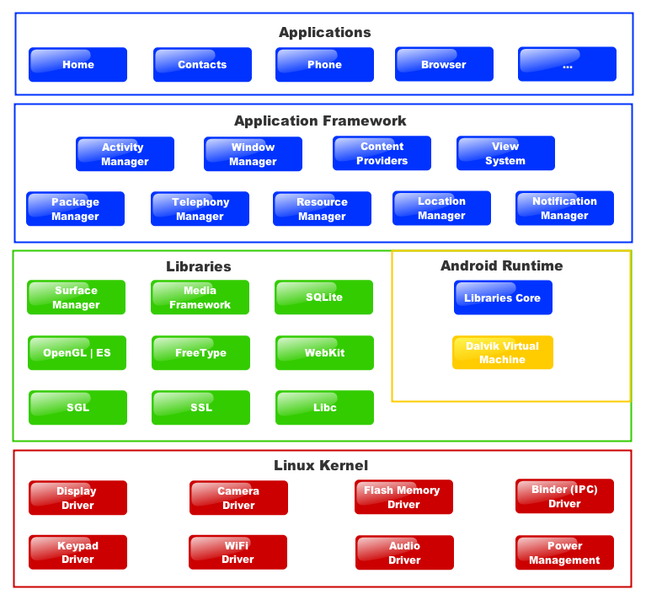
The Android SDK contains the Android Debug Bridge (ADB) tool which allows us connect with our android device.

**ANDROID DEVELOPMENT TOOLS:**

Google provides the Android Development Tools (ADT) to develop Android applications with Eclipse. ADT is a set of components (plug-ins) which extend the Eclipse IDE with Android development capabilities.

ADT contains all required functionalities to create, compile, debug and deploy Android applications from the Eclipse IDE. ADT also allows creating and starting AVDs. The Android Development Tools (ADT) is used for creating page layouts in our project.

**ANDROID SOFTWARE STACK:**

Fig 2.2 Android Software Stack

**Android Program Execution Process:**

1. A java file (ie)[ home.java ] file and other java files is stored in **SRC** folder , and an **XML** file is stored in **RES->layout** folder
2. After run a file in eclipse **XML** file is moved to **AAPT (Android Application Program Tool)** Which converts xml file into **R.Java** in the **GEN** folder.
3. Now the (home.java) file and other java file compile through java compiler **Javac** compile the program.
4. Now the .java file from **SRC** folder and .dex file from **Java Compiler** are moved to apk Builder Where it converts all the source code into final application with extension **(.apk)** which is compatible for android platform.

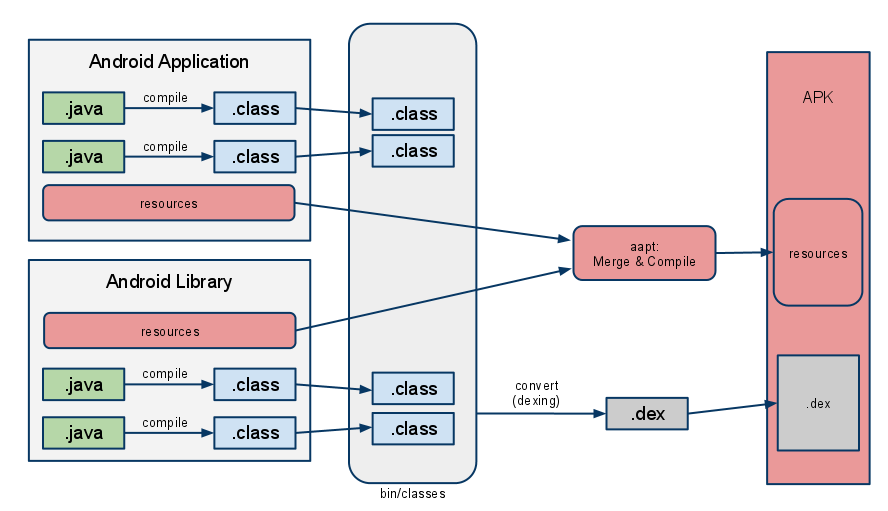
****

Fig 2.3 Android Compilation Process

**Dex File:**

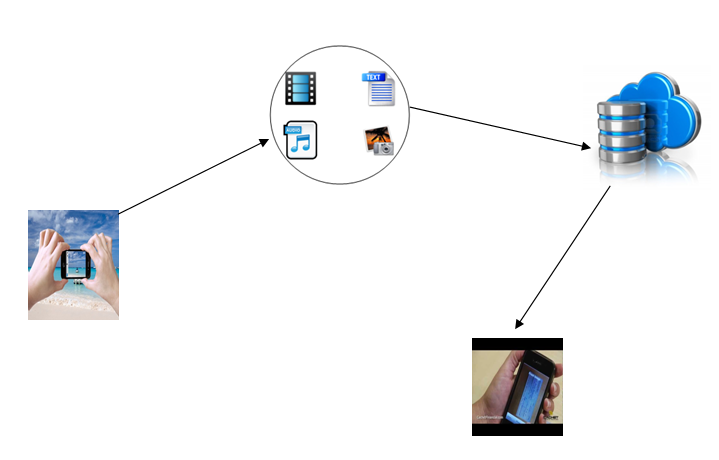
Java source code is converted into Java Byte Code using Java compiler, Java byte code is again converted to Dalvik byte using dex compiler.

**APK File:**Android Package Kit (APK) file an android archive, composed of Dalvik Executable and Resource file.

**CHAPTER 3**

**3 SYSTEM DESIGN**

**3.1 SYSTEM ARCHITECTURE**



Capturing and

Recording Multimedia file using application

Uploading the

multimedia files

to cloud storage

Downloading and

viewing files in mobile

Fig 3.1 System Architecture

In System Architecture refer to Fig: 3.1Uploading Multimedia files there are four main processes are given as follows initially capturing and recording multimedia files using android application. Then we must upload the file to cloud using Dropbox application. In our application after entering a text, capturing image, audio and video the application will ask us to link with dropbox. Then, we must link with dropbox by logging in with username and password. By giving valid username and password we can login into our account.

The next step is we have to select the files that are to be uploaded and then we can share our multimedia files to required persons. Then our recipients can view or download the files. They can even search the files by using time, date, keywords or place. Hence it will automatically store the gps position which will help us to search by the required position. This is an efficient cloud storage method which provides high security.

Finally the application can be installed in our android phones and now we can store our files in cloud rather than storing it in internal or external memory. Hence it will save energy and increases processing speed. This is an efficient cloud storage method to save multimedia files.

**3.2 USE CASE DIAGRAM**

A use case is a set of scenarios that describing an interaction between a user and a system. A use case diagram displays the relationship among actors and use cases. The two main components a user or another system that will interact with the system modeled. A use case is an external view of the system that represents some action the user might perform in order to complete a task.

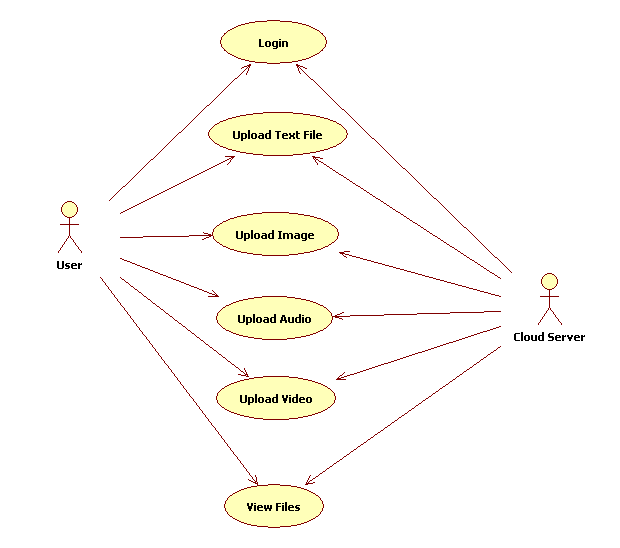


Fig: 3.2 Use Case Diagram

In the use case scenario refer to Fig: 3.2 there are two actors user and cloud server. They act has a key players in the complete process. Initially user must login with a valid username and password. If we provide authenticated username and password then we can login into dropbox. Then we have to enter a text and upload the text into cloud by linking it with dropbox. Next we can capture an image and upload an image into cloud by linking it with dropbox. Next we can record an audio file and upload an audio into cloud by linking it with dropbox. Next we can capture a video and upload video into cloud by linking it with dropbox. At last user can view the files in cloud and can even search the file using time, date, or gps location.

**3.3 CLASS DIAGRAM**

Class Diagram provides an overview of the target system by describing the objects and classes inside the system and the relationships between them. It provides a wide variety of usages; from modeling the domain-specific data structure to detailed design of the target system. With the share model facilities, you can reuse your class model in the interaction diagram for modeling the detailed design of the dynamic behavior.



Fig: 3.3 Class Diagram

In the class diagram refer to Fig: 3.3there are two classes namely user and cloud server.

The attributes given are as follows:

**Authenticate:** Here it authenticates the username and password.

**Valid:** It will validate our username and password.

**Filename:** Here we must provide valid files in order to upload multimedia files in cloud server.

The methods given are as follows:

**Authenticate ():** It will check whether a valid username or password is given or not.

**Upload text ():** Now we can enter the text and upload the text in cloud by linking with dropbox.

**Upload image ():** Now we can capture an image and upload an image in cloud by linking with dropbox.

**Upload audio ():** Now we can record an audio and upload the audio file in cloud by linking with dropbox.

**Upload video ():** Now we can record the video and upload the video file in cloud by linking with dropbox.

**Transfer file ():** Here all the files are being transferred to cloud account and can be shared to the required person.

**View file ():** Finally all the files can be viewed and downloaded in mobile or pc.

**3.4 ACTIVITY DIAGRAM**

Activity diagram are typically used for business process modeling for modeling the logic captured by a single use case or usage scenario, or for modeling the detailed logic of a business rule. Although UML activity diagrams could potentially model the internal logic of a complex operation it would be far better to simply rewrite the operation so that it is simple enough that you don’t requires an activity diagram.



Fig: 3.4 Activity Diagram

In the activity diagram refer to Fig: 3.4the first phase is user authentication where it authenticates the username and password. Here it will authenticates whether valid login details is given or not. If an invalid login details is given then it will request for username and password. If our username and password is authenticated then it will proceed for next step.

Now we can upload the multimedia files by linking with dropbox. Here we can enter the text and upload text, and then we can capture an image and upload image, and then we can record an audio and upload an audio file, and then we can capture a video and upload the video file.

Finally, we can view the files in cloud account and can share the files to required person. Hence they can view or download the files in mobile or pc.

**3.5 SEQUENCE DIAGRAM**

The Sequence Diagram models the collaboration of objects based on a time sequence. It shows how the objects interact with others in a particular scenario of a use case. With the advanced visual modeling capability, you can create complex sequence diagram in few clicks. Besides, VP-UML can generate sequence diagram from the flow of events which you have defined in the use case description.



Fig: 3.5 Sequence Diagram

The sequence diagram refer to Fig: 3.5 explains the collaboration with the objects.

Here user initially enters a username and password to the cloud. Then it validates the username and password, if it accepts the login details then we can proceed for next step or we it will request us to provide valid username and password.

Now we can write the content and upload the text file in cloud. Then we can capture the image and upload the image in cloud server. Then we can record an audio and upload the audio file in cloud. Then we can capture a video file and upload the video file in cloud server. Then it will request us to link with dropbox. Then we can transfer all the files to our cloud account. Now we can share our multimedia files and they can view or download all the files.

**3.6 COLLABORATION DIAGRAM**

A collaboration diagram, also called a communication diagram or interaction diagram, is an illustration of the relationships and interactions among software objects in the Unified Modeling Language (UML). The concept is more than a decade old although it has been refined as modeling paradigms have evolved.



Fig: 3.6 Collaboration Diagram

In the collaboration diagram refer to Fig: 3.6 it is used to communicate with the objects. Here there will be communication between user and cloud server. Here the user can login, record text, image, audio and video. Then we have can upload text, image, audio and video in cloud in cloud server. Now we can view the files in cloud. Hence the cloud server will validate the login details and stores multimedia files in cloud account. From this cloud account we can share the files and can view and download the files.

**CHAPTER 4**

**4 SYSTEM IMPLEMENTATION**

**4.1 MODULES OVERVIEW**

* Login for Cloud Storage
* Uploading Multimedia files in Cloud
* Searching the files using Date filter
* View file on cloud

**4.2 LOGIN FOR CLOUD STORAGE**

Cloud storage is used to store, access and manage file on online. To upload the multimedia file to cloud storage creates an account on cloud storage. There are different types of cloud storage is available. Here we are using Drop Box for storing the multimedia file in cloud storage. Create an account on the Drop Box and link it with our application to sign in for our application.

User Login Page

Login using DropBox Account

Access the Apps Folder

Fig: 4.1 Login for Cloud Storage

**4.3 UPLOADING MULTIMEDIA FILES IN CLOUD**

The main aim of our project is to upload multimedia files in cloud storage using offloading method .Offloading method is mainly used to upload data to server or cloud using 3G and Wi-Fi .Offloading is an effective method for extending the lifetime of handheld mobile devices by executing some components of applications remotely (e.g., on the server in a data center or in a cloud).For the end users the purpose for doing mobile data offloading is based on data service cost control and availability of higher bandwidth.

User Login into DropBox Account

Capturing Live Multimedia File

Uploading the Files

Storing in DropBox Cloud Storage

Fig: 4.2 Uploading Multimedia files in Cloud

**4.4 SEARCHING THE FILE USING DATE FILTER**

In this module the user can be able to filter the multimedia files which are stored on the cloud storage. To filter the files which is stored on the cloud we are using date filter once the user select the particular date the multimedia file . Which is uploaded on the particular date will be displayed.

**4.5 VIEW FILES ON CLOUD**

In this module the user can be able to view the multimedia files which is stored on the server. Offloading methods are used to fetch the data from the cloud storage to our application. When the user press the view button the data’s from cloud storage is listed and user can be able to view the multimedia file which is stored on the cloud storage.

User Login into DropBox Account anywhere in the internet

Access the Apps Folder

View the files in Online Cloud Storage

Fig: 4.3 View files on Cloud

**CHAPTER 5**

**5 TESTING**

**5.1TESTING OVERVIEW**

Testing is a set of activities that can be planned in advance and conducted systematically. For this reason a template for software testing, a set of steps into which we can place specific test case design techniques and testing methods should be defined for software process. Testing often accounts for more effort than any other software engineering activity. If it is conducted haphazardly, time is wasted, unnecessary effort is expanded, and even worse, errors sneak through undetected. It would therefore seem reasonable to establish a systematic strategy for testing software.

**5.2TYPE OF TESTING**

There are two type of testing according their behaviors

1. Unconventional Testing

2. Conventional Testing

**5.1.1Unconventional Testing**

Unconventional testing is a process of verification which is doing by SQA (Software Quality Assurance) team. It is a prevention technique which is performing from begging to ending of the project development. In this testing the SQA team follows these methods:

1. Peer review

2. Code walk and throw

3. Inspection

4. Document Verification

**5.1.2 Conventional Testing**

Conventional Testing is a process of finding the bugs and validating the project. Testing team involves in this testing process and validating that developed project is according to client requirement or not.

* 1. **TESTING PLAN**

A number of software testing strategies have been proposed in the literature. All provide the software developer with a template for testing and all have the following generic characteristics:

1. Testing begins at the component level and works “outward” toward the integration of the entire computer-based system.
2. Different testing techniques are appropriate at different points in time.
3. The developer of the s/w conducts testing and for large projects, independent test group.

**Integration Testing:**

The strategies for integrating software components into a functioning product include the bottom-up strategy, the top-down strategy and to ensure that modules will be available for integration into the evolving software product when needed.

**White Box Testing:**

It is just the vice versa of the Black Box testing. There we do not watch the internal variables during testing. This gives clear idea about what is going on during execution of the system. The point at which the bug occurs was all clear and was removed.

**Black Box Testing:**

In this testing we give input to the system and test the output. Here we do not go for watching the internal file in the system and what are the changes made on them for the required output.

**Interface Testing**

The Interface Testing is performed to verify the interfaces between sub modules while performing integration of sub modules aiding master module recursively.

**Module Testing**

Module Testing is a process of testing the system, module by module. It includes the various inputs given, outputs produced and their correctness. By testing in this method we would be very clear of all the bugs that have occurred.

**Maintenance**

The objectives of this maintenance work are to make sure that the system gets into work all time without any bug. In our project maintenance work is done regarding with proper execution of the project.

**5.4 TEST CASES**

**Unit Testing**

The primary goal of unit testing is to take the smallest piece of testable software in the application, isolate it from the remainder of the code, and determine whether it behaves exactly as you expect. Also the duplicate username is given and checked. For Example: Wrong email-id and web site URL (Universal Resource Locator) is given and checked.

**Integration Testing**

Testing is done for each module. After testing all the modules, the modules are integrated and testing of the final system is done with the test data, specially designed to show that the system will operate successfully in all its aspects conditions. Thus the system testing is a confirmation that all is correct and an opportunity to show the user that the system works.

**Validation Testing**

The final step involves Validation testing, which determines whether the software function as the user expected. Hence here Validation testing is made in order to check whether our project runs successfully.

**CHAPTER 6**

**6 CONCLUSION**

Our study clearly indicates that offloading heavy applications, namely multimedia applications, from smart phones to MCC is beneficial. MCC significantly reduces the energy consumption on smart phones by the EaaS service. Moreover, MCC enriches smart phones capabilities for multimedia applications. At this time when the CC is in its infant state, the importance of evaluating the benefit of MCC to overcome smart phone constraints motivates us to conduct this study. A large number of experiments has been performed for common network interfaces (3G and Wi-Fi) and protocols (HTTP and FTP).

The location of the original file has been considered. This paper provides a wide range of comparison between possible encoding location, original file place, encoding configuration parameters, network interfaces, and Internet protocols. The results reveal the potential of MCC by reducing smart phones energy consumptions on multimedia applications at least 30%.

**CHAPTER 7**

**7 FUTURE ENHANCEMENTS**

We would conduct more experiments on other multimedia types such as audio and images to generalize our finding. This study opens new opportunities to be investigated. Optimum algorithms, architectures, and implementations for this offloading technique are needed to reach best offloading case. Finally, modeling the MCC to handle the offloading method is important to implement efficient offloading.

This experiment can be even implemented in ios and black berry operating system.

**CHAPTER 8**

**8 APPENDIX 1**

**8.1 SAMPLE CODING**

**Design Part Coding**

**Main.xml**

<?xml version=*"1.0"* encoding=*"utf-8"*?>

<LinearLayout xmlns:android=*"http://schemas.android.com/apk/res/android"*

android:orientation=*"vertical"*

android:layout\_width=*"fill\_parent"*

android:layout\_height=*"wrap\_content"*

>

<Button

android:id=*"@+id/auth\_button"*

android:text=*"@string/lik"*

android:layout\_width=*"fill\_parent"*

android:layout\_height=*"wrap\_content"*

/>

<ScrollView xmlns:android=*"http://schemas.android.com/apk/res/android"*

android:layout\_width=*"fill\_parent"*

android:layout\_height=*"fill\_parent"*>

<LinearLayout

android:id=*"@+id/logged\_in\_display"*

android:orientation=*"vertical"*

android:layout\_width=*"fill\_parent"*

android:layout\_height=*"0dp"*

android:layout\_weight=*"1"* >

<TableLayout

android:layout\_width=*"fill\_parent"*

android:layout\_height=*"wrap\_content"*

android:orientation=*"vertical"* >

<TableRow >

<Button

android:id=*"@+id/store"*

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

android:background=*"@drawable/new\_button"* />

<TextView

android:id=*"@+id/tv1"*

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

android:text=*"Write Text"*

android:textSize=*"20sp"*

android:layout\_marginLeft=*"10dp"*

/>

</TableRow>

<TableRow >

<Button

android:id=*"@+id/photo\_button"*

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

android:background=*"@drawable/second"* />

<TextView

android:id=*"@+id/tv2"*

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

android:text=*"Take picture"*

android:textSize=*"20sp"*

android:layout\_marginLeft=*"10dp"*

/>

</TableRow>

<TableRow >

<Button

android:id=*"@+id/video\_button"*

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

android:background=*"@drawable/third"* />

<TextView

android:id=*"@+id/tv3"*

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

android:text=*"Record Video"*

android:textSize=*"20sp"*

android:layout\_marginLeft=*"10dp"*

/>

</TableRow>

<TableRow >

<Button

android:id=*"@+id/audio\_button"*

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

android:background=*"@drawable/third"* />

<TextView

android:id=*"@+id/tv3"*

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

android:text=*"Record Audio"*

android:textSize=*"20sp"*

android:layout\_marginLeft=*"10dp"*

/>

</TableRow>

<TableRow >

<Button

android:id=*"@+id/list\_button"*

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

android:background=*"@drawable/fourth"* />

<TextView

android:id=*"@+id/tv4"*

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

android:text=*"View File"*

android:textSize=*"20sp"*

android:layout\_marginLeft=*"10dp"*

/>

</TableRow>

</TableLayout>

<ImageView

android:id=*"@+id/image\_view"*

android:layout\_width=*"wrap\_content"*

android:layout\_height=*"wrap\_content"*

/>

</LinearLayout>

</ScrollView>

</LinearLayout>

**Background Coding**

**Home.Java**

\* Copyright (c) 2010-11 Dropbox, Inc.

package com.sweet.memories;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.OutputStreamWriter;

import java.text.DateFormat;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.List;

import android.annotation.SuppressLint;

import android.app.Activity;

import android.app.AlertDialog;

import android.app.Dialog;

import android.content.ActivityNotFoundException;

import android.content.Context;

import android.content.DialogInterface;

import android.content.Intent;

import android.content.pm.PackageManager;

import android.database.Cursor;

import android.location.Address;

import android.location.Geocoder;

import android.location.Location;

import android.location.LocationListener;

import android.location.LocationManager;

import android.media.MediaRecorder;

import android.net.Uri;

import android.os.Bundle;

import android.os.Environment;

import android.provider.MediaStore;

import android.util.Log;

import android.view.Menu;

import android.view.MenuInflater;

import android.view.MenuItem;

import android.view.View;

import android.view.View.OnClickListener;

import android.widget.Button;

import android.widget.EditText;

import android.widget.ImageView;

import android.widget.LinearLayout;

import android.widget.Toast;

import com.dropbox.client2.DropboxAPI;

import com.dropbox.client2.android.AndroidAuthSession;

@SuppressLint("ParserError")

public class Home extends Activity {

private static final String TAG = "Home";

private String addr = "Location Unknown";

final static private String APP\_KEY = "id8km2kwy3z5pn7";

final static private String APP\_SECRET = "glz4rcwbtqdk3nu";

final static private AccessType ACCESS\_TYPE = AccessType.APP\_FOLDER;

final static private String ACCOUNT\_PREFS\_NAME = "prefs";

final static private String ACCESS\_KEY\_NAME = "ACCESS\_KEY";

final static private String ACCESS\_SECRET\_NAME = "ACCESS\_SECRET";

final static int REQUEST\_VIDEO\_CAPTURED = 2;

private static int REQUEST\_AUDIO\_RECORDER = 3;

String audio\_path;

private static int MEDIA\_TYPE\_IMAGE = 1;

private static int MEDIA\_TYPE\_VIDEO = 2;

Uri uriVideo = null;

DropboxAPI<AndroidAuthSession> mApi;

private boolean mLoggedIn;

private Button mSubmit;

private LinearLayout mDisplay;

private Button mPhoto;

private Button Viewfile;

private Button mVideo;

private Button audio;

private ImageView mImage;

private final String PHOTO\_DIR = "/Photos/";

String data = null;

final static private int NEW\_PICTURE = 1;

private String mCameraFileName;

private String mVideoFileName;

MediaRecorder mRecorder;

LocationManager locationManager;

MyLocationListener locationListener;

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

if (savedInstanceState != null) {

mCameraFileName = savedInstanceState.getString("mCameraFileName");

}

if (savedInstanceState != null) {

mCameraFileName = savedInstanceState.getString("mVideoFileName");

}

// We create a new AuthSession so that we can use the Dropbox API.

AndroidAuthSession session = buildSession();

mApi = new DropboxAPI<AndroidAuthSession>(session);

setContentView(R.layout.main);

locationListener = new MyLocationListener();

locationManager = (LocationManager) getSystemService(Context.LOCATION\_SERVICE);

locationManager.requestLocationUpdates(LocationManager.NETWORK\_PROVIDER, 60000, 500, locationListener);

checkAppKeySetup();

mSubmit = (Button) findViewById(R.id.auth\_button);

mSubmit.setOnClickListener(new OnClickListener() {

public void onClick(View v) {

// This logs you out if you're logged in, or vice versa

if (mLoggedIn) {

logOut();

} else {

mApi.getSession().startAuthentication(Home.this);

}

}

});

mDisplay = (LinearLayout) findViewById(R.id.logged\_in\_display);

// This is where a photo is displayed

mImage = (ImageView) findViewById(R.id.image\_view);

// This is the button to take a photo

mPhoto = (Button) findViewById(R.id.photo\_button);

mPhoto.setOnClickListener(new OnClickListener() {

public void onClick(View v) {

Intent intent = new Intent();

// Picture from camera

intent.setAction(MediaStore.ACTION\_IMAGE\_CAPTURE);

Uri fileUri = getOutputMediaFileUri(MEDIA\_TYPE\_IMAGE);

intent.putExtra(android.provider.MediaStore.EXTRA\_OUTPUT,

fileUri);

Log.i(TAG, "Importing New Picture: " + mCameraFileName);

try {

startActivityForResult(intent, NEW\_PICTURE);

} catch (ActivityNotFoundException e) {

showToast("There doesn't seem to be a camera.");

}

}

});

mVideo = (Button) findViewById(R.id.video\_button);

mVideo.setOnClickListener(new OnClickListener() {

@Override

public void onClick(View v) {

Intent intent = new Intent();

intent.setAction(MediaStore.ACTION\_VIDEO\_CAPTURE);

Uri fileUri = getOutputMediaFileUri(MEDIA\_TYPE\_VIDEO);

mVideoFileName = fileUri.toString();

Log.v("Video uri", "Uri:" + fileUri);

startActivityForResult(intent, REQUEST\_VIDEO\_CAPTURED);

}

});

audio = (Button) findViewById(R.id.audio\_button);

audio.setOnClickListener(new OnClickListener() {

@Override

public void onClick(View arg0) {

startRecording();

AlertDialog.Builder alert = new AlertDialog.Builder(Home.this);

alert.setCancelable(false);

alert.setTitle("Recording audio");

alert.setMessage("Please start your speech to record");

alert.setPositiveButton("Stop",

new DialogInterface.OnClickListener() {

public void onClick(DialogInterface dialog, int which) {

stopRecording();

File file;

try {

file = new File(audio\_path);

UploadFile upload = new UploadFile( Home.this, mApi, PHOTO\_DIR, file);

upload.execute();

}

catch (Exception e) {

Log.e("Audio upload error:");

e.printStackTrace();

}

}

});

alert.setNegativeButton("Cancel",

new DialogInterface.OnClickListener() {

public void onClick(DialogInterface dialog,

int which) {

stopRecording();

}

});

alert.show();

}

});

// final EditText etext=(EditText)findViewById(R.id.etext);

Button store = (Button) findViewById(R.id.store);

store.setOnClickListener(new OnClickListener() {

public void onClick(View v) {

final Dialog d = new Dialog(Home.this);

d.setContentView(R.layout.dialog);

d.setCancelable(true);

Button b = (Button) d.findViewById(R.id.bt);

b.setOnClickListener(new OnClickListener() {

public void onClick(View v) {

d.dismiss();

EditText e = (EditText) d.findViewById(R.id.et);

data = e.getText().toString();

// Toast.makeText(Home.this,"Con: "+s,

// Toast.LENGTH\_SHORT).show();

storeText();

}

});

d.show();

// data=etext.getText().toString();

}});

// This is the button to take a photo

Viewfile = (Button) findViewById(R.id.list\_button);

Viewfile.setOnClickListener(new OnClickListener() {

public void onClick(View v) {

ViewFileList download = new ViewFileList(Home.this, mApi,

PHOTO\_DIR, mImage);

download.execute();

}

});

// Display the proper UI state if logged in or not

setLoggedIn(mApi.getSession().isLinked());

}

private void startRecording() {

audio\_path = Environment.getExternalStorageDirectory()

.getAbsolutePath() + File.separator;

String timeStamp = new SimpleDateFormat("yyyy-MM-dd-kk-mm")

.format(new Date());

audio\_path += "audio\_";

audio\_path += timeStamp + ".3gp";

mRecorder = new MediaRecorder();

mRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);

mRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE\_GPP);

mRecorder.setOutputFile(audio\_path);

mRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR\_NB);

try {

mRecorder.prepare();

} catch (IOException e) {

Log.e("Audio record error", "prepare() failed");

}

mRecorder.start();

}

private void stopRecording() {

mRecorder.stop();

mRecorder.release();

mRecorder = null;

}

private Uri getOutputMediaFileUri(int type) {

return Uri.fromFile(getOutputMediaFile(type));

}

/\*\* Create a File for saving an image or video \*/

private File getOutputMediaFile(int type) {

File mediaStorageDir = new File(Environment

.getExternalStoragePublicDirectory(Environment.DIRECTORY\_PICTURES),

"MyCameraApp");

if (!mediaStorageDir.exists()) {

if (!mediaStorageDir.mkdirs()) {

Log.d("MyCameraApp", "failed to create directory");

return null;

}

}

// Create a media file name

String timeStamp = new SimpleDateFormat("yyyy-MM-dd-kk-mm-ss")

.format(new Date());

File mediaFile;

if (type == MEDIA\_TYPE\_IMAGE) {

mediaFile = new File(mediaStorageDir.getPath() + File.separator+ "IMG\_" + timeStamp + ".jpg");

mCameraFileName = mediaFile.getPath();

} else if (type == MEDIA\_TYPE\_VIDEO) {

mediaFile = new File(mediaStorageDir.getPath() + File.separator+ "VID\_" + timeStamp + ".mp4");

mVideoFileName = mediaFile.getPath();

} else {

return null;

}

return mediaFile;

}

protected void onSaveInstanceState(Bundle outState) {

outState.putString("mCameraFileName", mCameraFileName);

outState.putString("mVideoFileName", mVideoFileName);

super.onSaveInstanceState(outState);

}

protected void onResume() {

super.onResume();

AndroidAuthSession session = mApi.getSession();

if (session.authenticationSuccessful()) {

try {

session.finishAuthentication();

TokenPair tokens = session.getAccessTokenPair();

storeKeys(tokens.key, tokens.secret);

setLoggedIn(true);

} catch (IllegalStateException e) {

showToast("Couldn't authenticate with Dropbox:"

+ e.getLocalizedMessage());

Log.i(TAG, "Error authenticating", e);

}

}

}

public void onActivityResult(int requestCode, int resultCode, Intent data) {

if (requestCode == NEW\_PICTURE) {

if (resultCode == Activity.RESULT\_OK) {

Uri uri = null;

if (data != null) {

uri = data.getData();

}

if (uri == null && mCameraFileName != null) {

uri = Uri.fromFile(new File(mCameraFileName));

Log.v("Picture Uri", uri.toString() + " " + uri.getPath());

}

File file = new File(mCameraFileName);

Log.v("Picture file", "" + file.getPath());

if (uri != null) {

UploadFile upload = new UploadFile(Home.this, mApi,PHOTO\_DIR, file);

upload.execute();

}

}

} else if (requestCode == REQUEST\_VIDEO\_CAPTURED) {

if (resultCode == RESULT\_OK)

{

uriVideo = data.getData();

Log.w(TAG, uriVideo.getPath() + "activity result video 1"

+ uriVideo.toString());

File file;

try {

file = new File(getRealPathFromURI(uriVideo));

Log.d("Video file", "File:" + file.getPath());

UploadFile upload = new UploadFile(Home.this, mApi,PHOTO\_DIR, file);

upload.execute();

File fi = new File(file.getPath());

fi.delete();

} catch (Exception e) {

Log.e("Video upload error:", "Video upload error:");

e.printStackTrace();

}

// showToast("till capture");

} else if (resultCode == RESULT\_CANCELED) {

uriVideo = null;

Toast.makeText(Home.this, "Cancelled!", Toast.LENGTH\_LONG)

.show();

} else {

Log.w(TAG, "Unknown Activity Result from mediaImport: "

+ resultCode);

}

}

}

**UploadFile.java**

package com.sweet.memories;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import android.app.ProgressDialog;

import android.content.Context;

import android.content.DialogInterface;

import android.content.DialogInterface.OnClickListener;

import android.util.Log;

import android.widget.Toast;

import com.dropbox.client2.DropboxAPI;

import com.dropbox.client2.DropboxAPI.UploadRequest;

import com.dropbox.client2.exception.DropboxException;

import com.dropbox.client2.exception.DropboxFileSizeException;

import com.dropbox.client2.exception.DropboxIOException;

import com.dropbox.client2.exception.DropboxServerException;

import com.dropbox.client2.exception.DropboxUnlinkedException;

public class UploadFile extends AsyncTask<Void, Long, Boolean> {

String TAG="UploadFile";

private DropboxAPI<?> mApi;

private String mPath;

private File mFile;

private UploadRequest mRequest;

private Context mContext;

private final ProgressDialog mDialog;

private String mErrorMsg;

public UploadFile(Context context, DropboxAPI<?> api, String dropboxPath,

File file) {

mContext = context.getApplicationContext();

mFileLen = file.length();

mApi = api;

mPath = dropboxPath;

mFile = file;

mDialog = new ProgressDialog(context);

mDialog.setMax(100);

mDialog.setMessage("Uploading " + file.getName());

mDialog.setProgressStyle(ProgressDialog.STYLE\_HORIZONTAL);

mDialog.setProgress(0);

mDialog.setButton("Cancel", new OnClickListener() {

public void onClick(DialogInterface dialog, int which) {

mRequest.abort();

}

});

mDialog.show();

}

**Display.java**

package com.sweet.memories;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.FileReader;

import android.app.Activity;

import android.app.ProgressDialog;

import android.content.Intent;

import android.os.Message;

import android.util.Log;

import android.widget.TextView;

import android.widget.Toast;

import com.dropbox.client2.exception.DropboxException;

public class DisplayText extends Activity {

ProgressDialog pd;

Intent intent;

private FileOutputStream mFos;

private String cachePath;

private final static String TEXT\_FILE\_NAME = "sweet.txt";

TextView textView;

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

textView = new TextView(this);

setContentView(textView);

intent = getIntent();

pd = ProgressDialog.show(DisplayText.this, "", "Importing text...", false, true);

new Thread() {

public void run() {

cachePath = getCacheDir().getAbsolutePath() + "/" + TEXT\_FILE\_NAME;

try {

mFos = new FileOutputStream(cachePath);

} catch (FileNotFoundException e) {

Log.e("Get Image error:", ""+e.toString());

}

try {

ViewFileList.mApi.getFile(intent.getStringExtra("TextPath"), null, mFos, null);

} catch (DropboxException e) {

handler.sendEmptyMessage(0);

}

handler.sendEmptyMessage(1);

}

}.start();

}

private Handler handler = new Handler() {

public void handleMessage(Message msg) {

pd.dismiss();

if (msg.what == 1) {

File file = new File(getCacheDir().getAbsolutePath(),TEXT\_FILE\_NAME);

Log.d("text file", ""+file.getPath());

StringBuilder text = new StringBuilder();

try {

BufferedReader br = new BufferedReader(new FileReader(file));

String line;

while ((line = br.readLine()) != null) {

text.append(line);

text.append('\n');

}

textView.setText(text.toString());

}

catch (Exception e) {

}

}

else if (msg.what == 0) {

Toast.makeText(DisplayText.this, "Error while retrieving image", Toast.LENGTH\_LONG).show();

DisplayText.this.finish();

}

}

};

}

**Displayimage.java**

package com.sweet.memories;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import android.app.Activity;

import android.app.ProgressDialog;

import android.content.Intent;

import android.graphics.drawable.Drawable;

import android.os.Bundle;

import android.os.Handler;

import android.os.Message;

import android.widget.ImageView;

import android.widget.LinearLayout;

import android.widget.TextView;

import android.widget.Toast;

import com.dropbox.client2.DropboxAPI.ThumbSize;

import com.dropbox.client2.exception.DropboxException;

public class DisplayImage extends Activity {

ProgressDialog pd;

ImageView imageView;

Intent intent;

String cachePath;

private final static String IMAGE\_FILE\_NAME = "dbroulette.png";

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

LinearLayout linearLayout = new LinearLayout(this);

linearLayout.setOrientation(LinearLayout.VERTICAL);

imageView = new ImageView(this);

TextView textView = new TextView(this);

linearLayout.addView(textView);

linearLayout.addView(imageView);

setContentView(linearLayout);

intent = getIntent();

textView.setText(intent.getStringExtra("ImagePath"));

pd = ProgressDialog.show(DisplayImage.this, "", "Importing image...", false, true);

new Thread() {

private FileOutputStream mFos;

public void run() {

cachePath = getCacheDir().getAbsolutePath() + "/" + IMAGE\_FILE\_NAME;

try {

mFos = new FileOutputStream(cachePath);

} catch (FileNotFoundException e) {

Log.e("Get Image error:", ""+e.toString());

}

try {

ViewFileList.mApi.getThumbnail(intent.getStringExtra("ImagePath"), mFos, ThumbSize.BESTFIT\_960x640,

ThumbFormat.JPEG, null);

} catch (DropboxException e) {

handler.sendEmptyMessage(0);

}

handler.sendEmptyMessage(1);

}

}.start();

}

private Handler handler = new Handler() {

public void handleMessage(Message msg) {

pd.dismiss();

if (msg.what == 1) {

imageView.setImageDrawable(Drawable.createFromPath(cachePath));

}

else if (msg.what == 0) {

Toast.makeText(DisplayImage.this, "Error while retrieving image", Toast.LENGTH\_LONG).show();

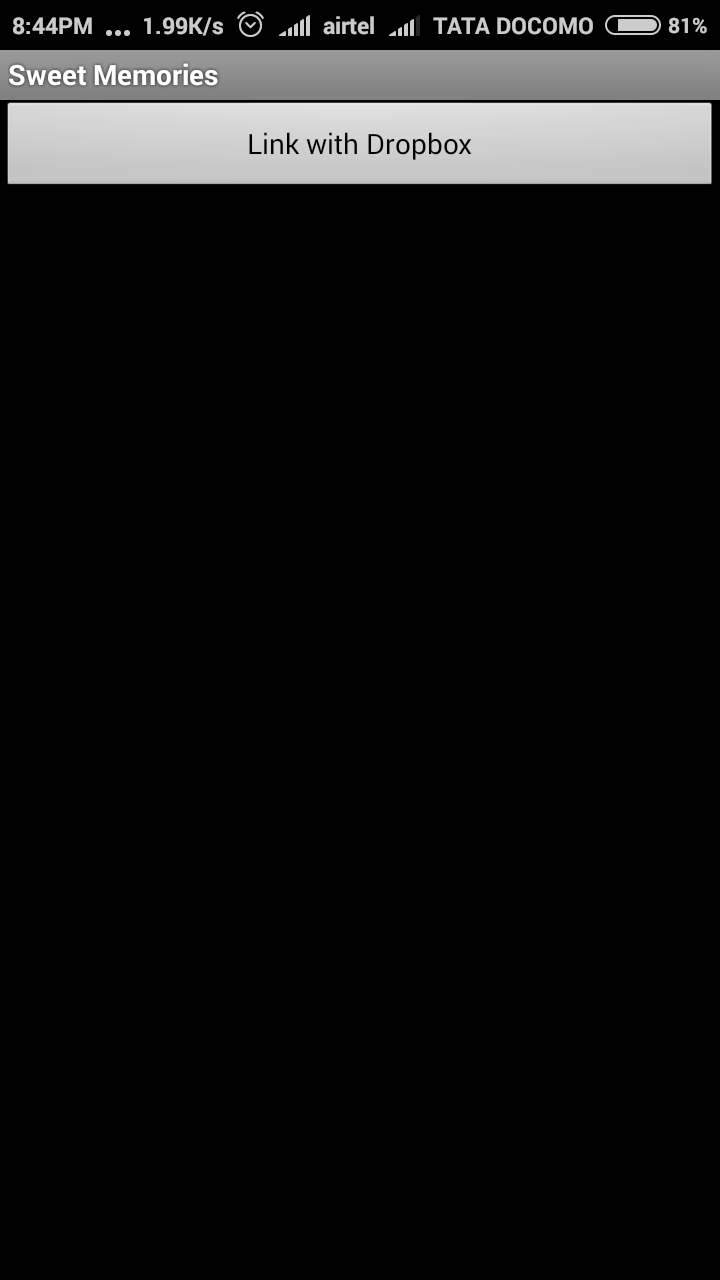
DisplayImage.this.finish();

}

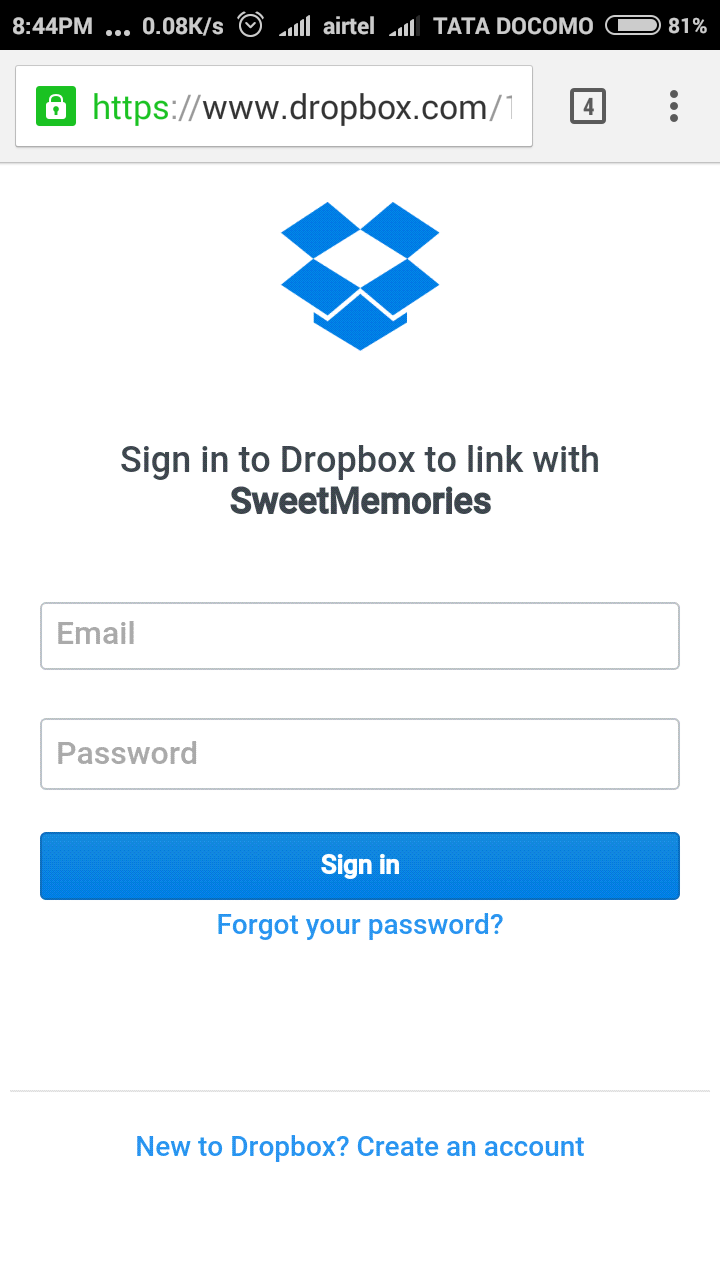
}

};}

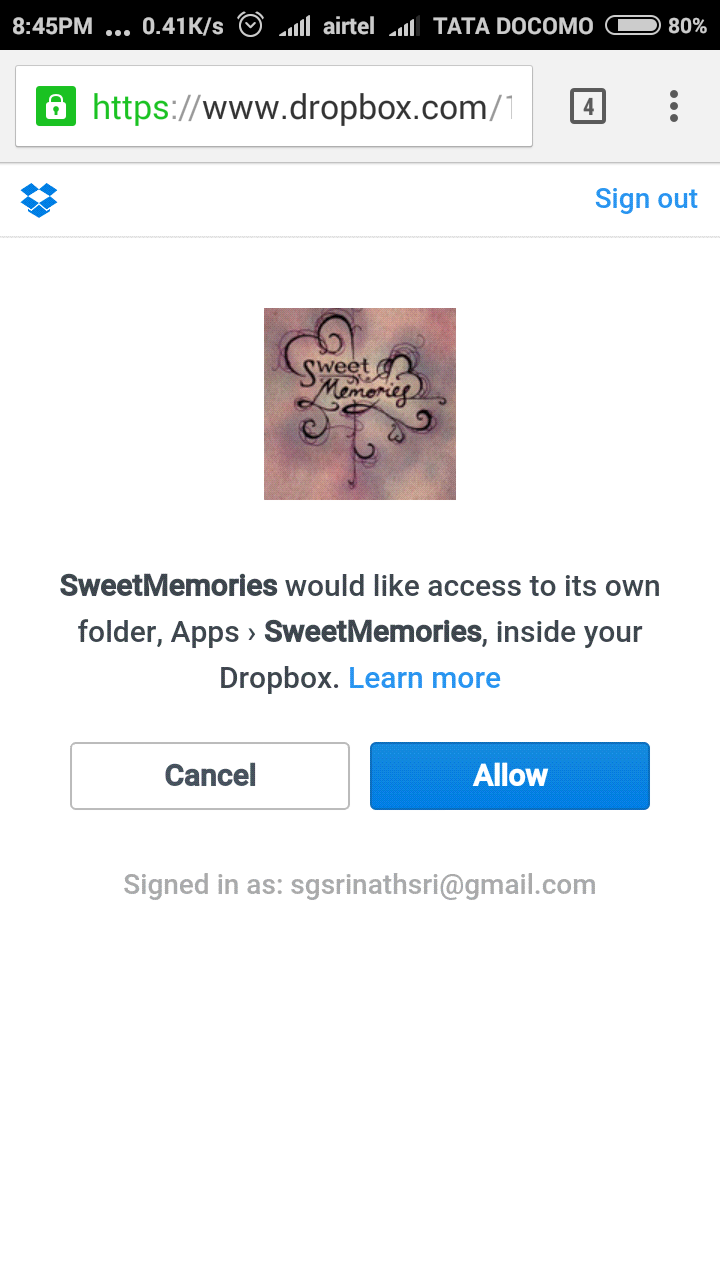
**8.2 SCREEN SHOTS**

****

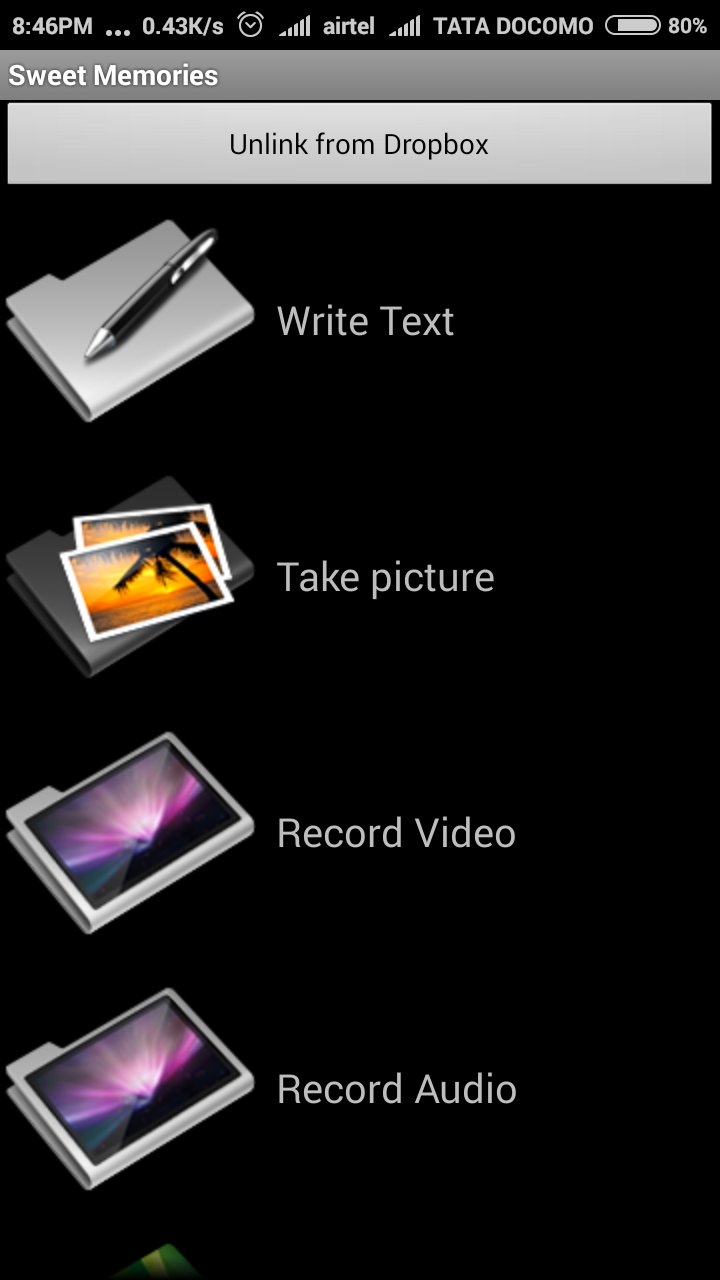
**Fig 1 Link with DropBox**

****

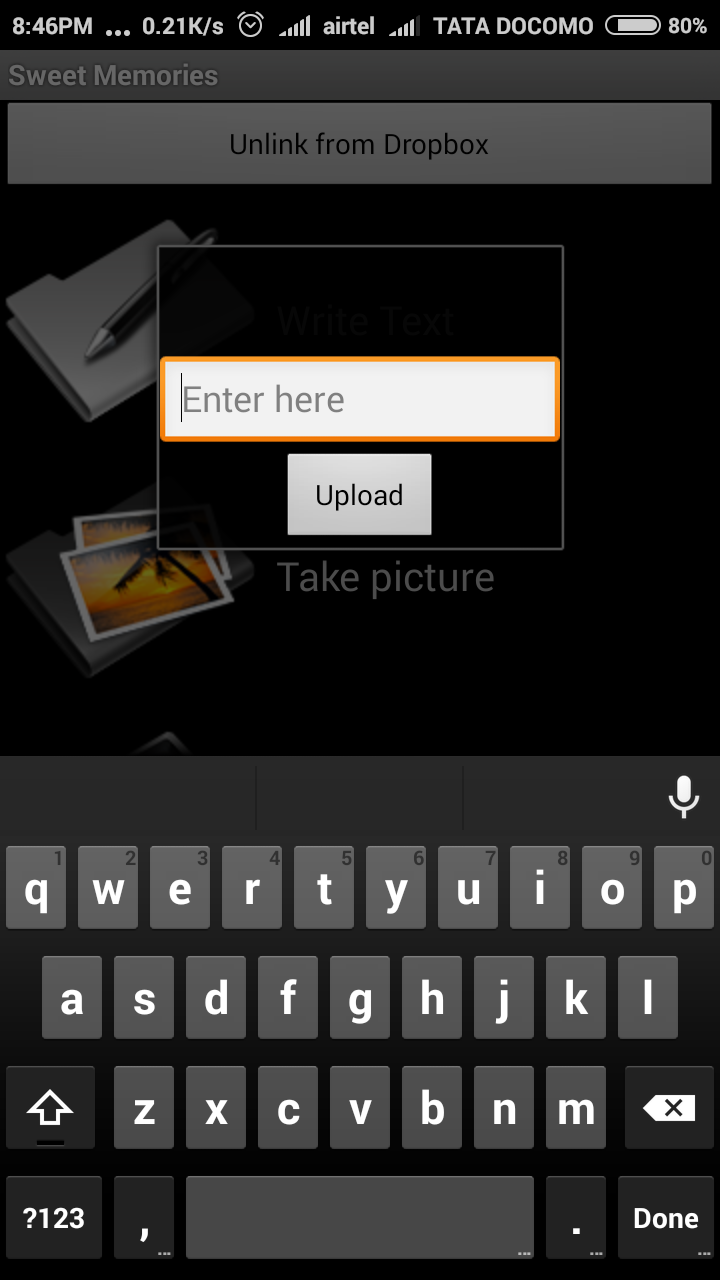
**Fig 2 DropBox Login page**

****

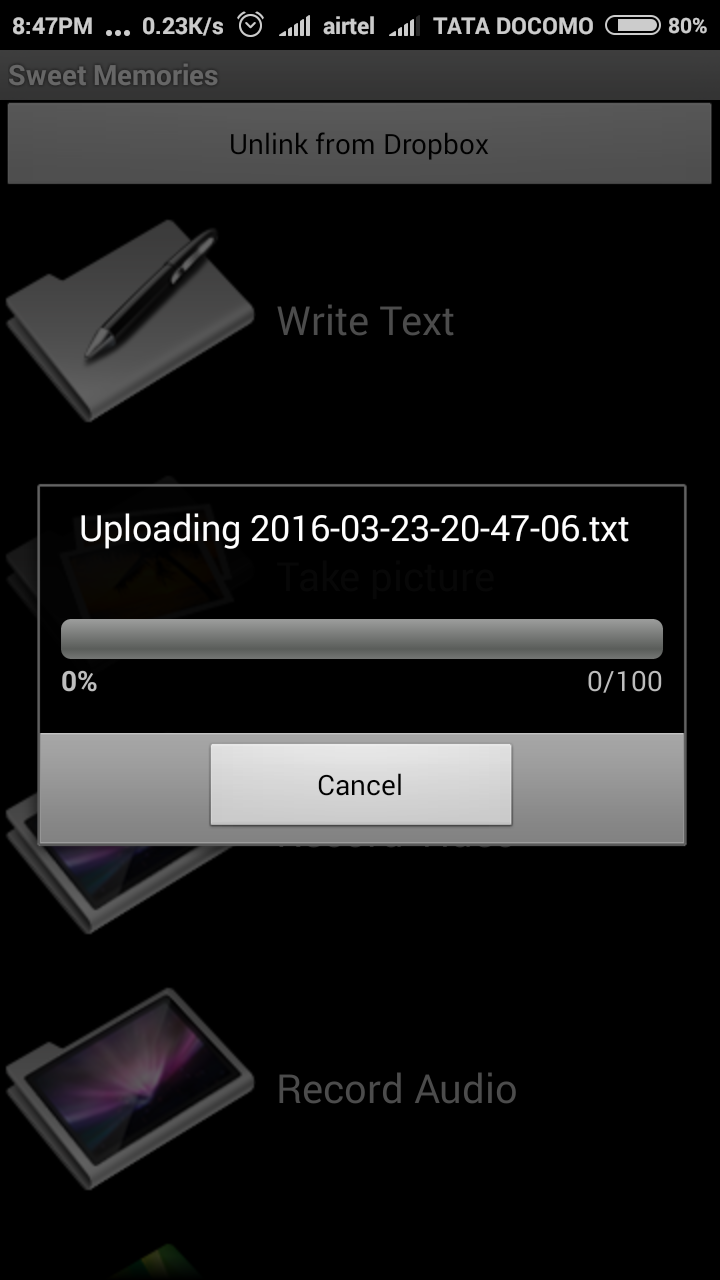
**Fig 3 Accessing DropBox Folder**

****

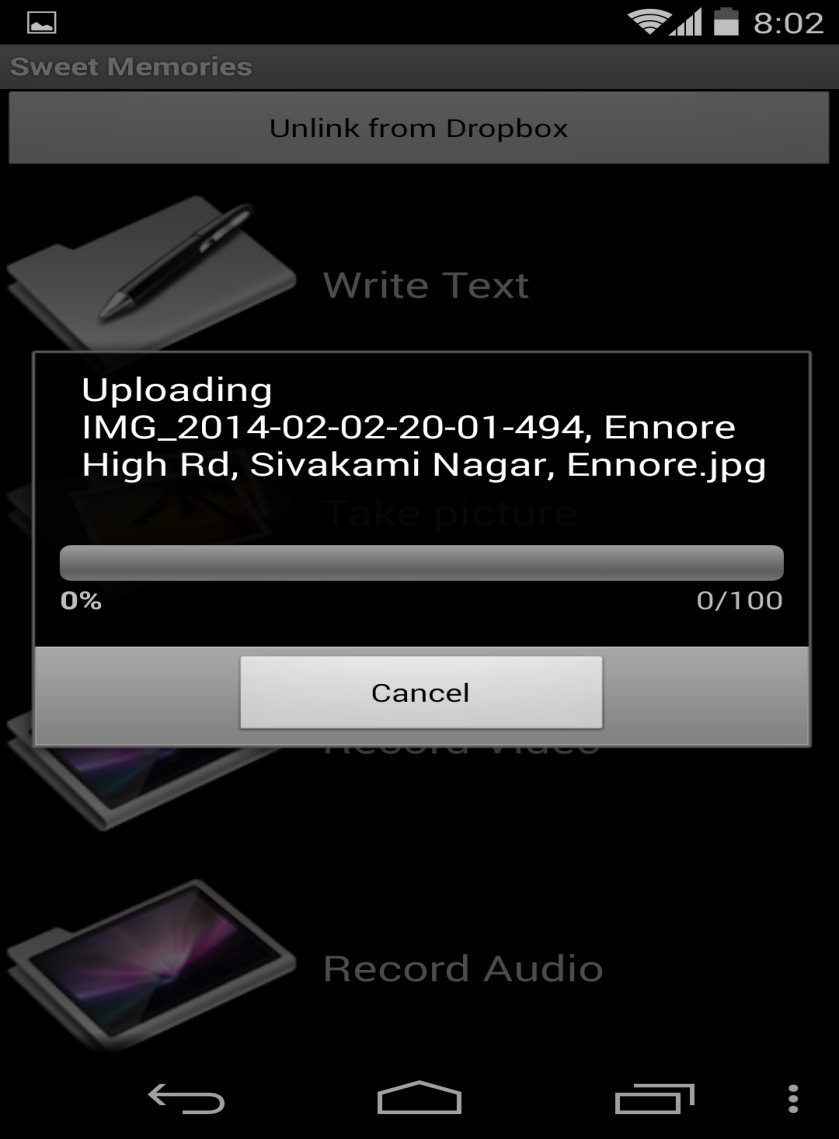
**Fig 4 Sweet Memories App list**

****

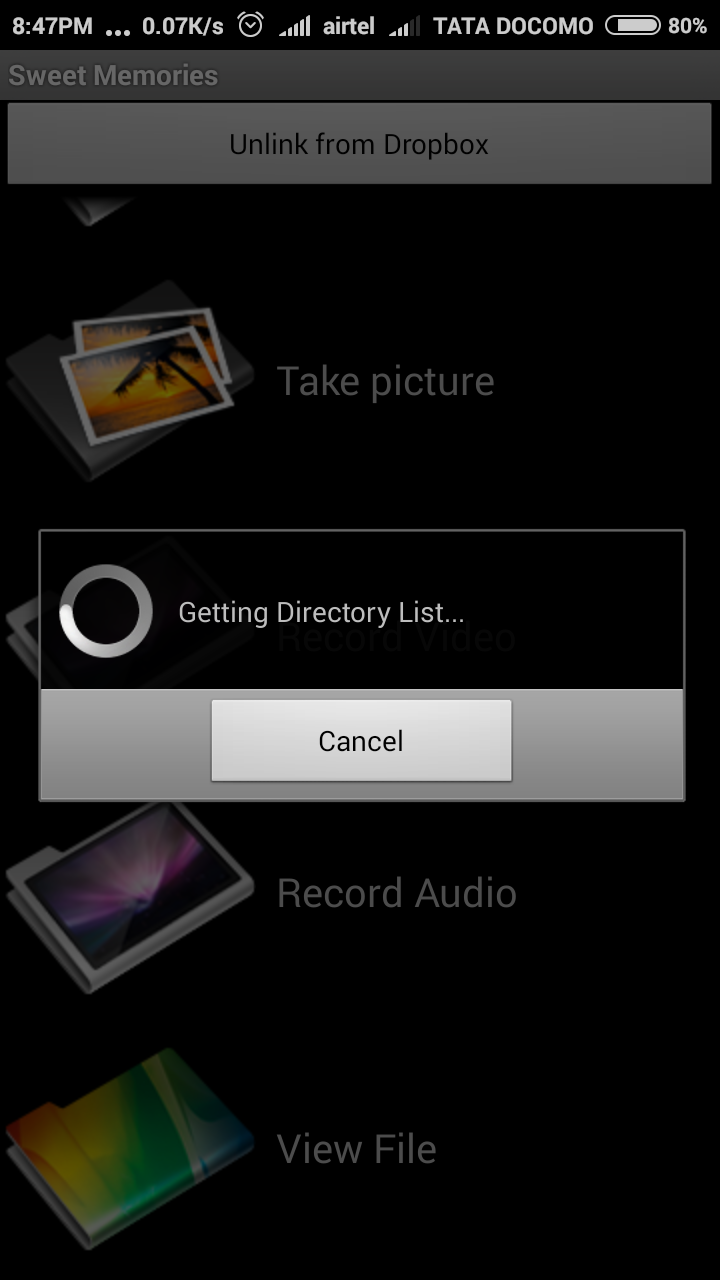
**Fig 5 Typing Text for upload**

****

**Fig 6 Uploading Text into DropBox Folder**

****

**Fig 7 Uploading Image into DropBox Folder**

****

**Fig 8 Retrieving Files from DropBox Folder**

****

**Fig 9 List of files in DropBox Folder**

**9 REFERENCES**

1. A. Carroll and G. Heiser, “An Analysis of Power Consumption in a Smartphone” in Proceedings of the 2010 USENIX conference on USENIX annual technical conference, 2010.
2. M. Altamimi and K. Naik, “The Concept of a Mobile Cloud Computing to Reduce Energy Cost of Smartphones and ICT Systems” in Proceedings of the First international conference on Information and communication on technology for the fight against global warming, ser. ICT-GLOW’11. Berlin, Heidelberg: Springer-Verlag, 2011.
3. K. Kumar and Y.-H. Lu, “Cloud Computing for Mobile Users: Can0020Offloading Computation Save Energy” Computer, 2010.
4. “Computing: Balancing Energy in Processing, Storage, and Transport” Proceedings of the IEEE, January 2011.
5. K. Yang, S. Ou, and H.-H. Chen, “On Effective Offloading Services for Resource-Constrained Mobile Devices Running Heavier Mobile Internet Applications” IEEE Communications Magazine, 2008.
6. R. Wolski, S. Gurun, C. Krintz, and D. Nurmi, “Using Bandwidth Data to Make Computation Offloading Decisions” in Proc. IEEE Int. Symp. Parallel and Distributed Processing, 2008.
7. W. Itani, A. Chehab, and A. Kayssi, “Energy-Efficient Platform-as-a- Service Security Provisioning in the Cloud” in Proc. Int Energy Aware Computing (ICEAC) Conf, 2011.
8. H. Yuan, C.-C. J. Kuo, and I. Ahmad, “Energy Efficiency in Data Centers and Cloud-Based Multimedia Services: An Overview and Future Directions” in Proc. Int. Green Computing Conf, 2010.

**LIST OF BOOKS AND WEBLINKS**

1. Android Programming, and Book! release.002 by **Nicolas Gramlich** .
2. 2. Professional Android Application Development by Perth, Western Australia, Reto Meier in London.
3. Android Open Source Project. Open Handset Alliance: <http://source.android.com/license>
4. Dalvik Debug Bridge Monitor Service: <http://developer.android.com/guide/developing/tools/ddms.html>
5. Official Android website: http://www.android.com/Open Handset Alliance, website: <http://www.openhandsetalliance.com/>
6. Java Virtual Machine Tool Interface Spec: <http://java.sun.com/javase/6/docs/platform/jvmti/jvmti.html>
7. Java Software, Sun Microsystems, Inc. Java Development Kit (JDK). <http://java.sun.com/products/jdk/1.2> .