SMART- A LEARNING APPLICATION FOR ANDROID BASED DEVICES

A Project

Presented to the faculty of the Department of Computer Science
California State University, Sacramento

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

by

Rohit Agashe

SPRING
2014
SMART- A LEARNING APPLICATION FOR ANDROID BASED DEVICES

A Project

by

Rohit Agashe

Approved by:

__________________________________, Committee Chair
Du Zhang, Ph.D.

__________________________________, Second Reader
Bill Mitchell, Ph.D.

__________________________
Date
Student:  **Rohit Agashe**

I certify that this student has met the requirements for format contained in the University format manual, and that this project is suitable for shelving in the Library and credit is to be awarded for the project.

________________________________________, Graduate Coordinator  ___________________________
Nikrouz  Faroughi, Ph.D.  

Date

Department of Computer Science
Abstract

of

SMART- A LEARNING APPLICATION FOR ANDROID BASED DEVICES

by

Rohit Agashe

The objective of the project is to design a learning application for android-based devices that will track the user’s behavior pattern and allow the user to manage his device smartly. There are a few existing apps that are already exhibiting this behavior for example App Usage Tracker [5]. However, Smart app does provide more knowledgeable actions compared to others (for example: comparison between user’s top favorites and user’s calculated favorites, easy access to the favorite apps by creating shortcuts on the home screen etc.). App Usage tracker [5] application is also capable of learning the user’s mobile behavioral patterns. The behavioral patterns captured would include number of things that would help the mobile users to combine statistics and provide insights into mobile device usage. For instance, Smart will tell which applications the user uses frequently on a certain period like daily, weekly and monthly by which the user will come to know which apps he/she uses regularly. That would be the user’s favorite app- “the app used most frequently”. There would also be comparison between the user’s top rated apps (wherein user gives initial rating to the apps according to his choice) v/s user’s
calculated favorite rating (which would be coming from the learning done on the user’s behavior patterns). Along with this learned data of user’s behavior patterns, the application also provides an insight into the average power consumption for the apps. This app is also intended to use the data gathered from learning a user's patterns to convert them into some knowledgeable actions that would enable more effective device use. For instance, creating a shortcut of the learned favorite apps to the home screen. This list will be refreshed every time the application is run. In this way, the old favorite list would be refreshed automatically.

The user will also be notified for the maximum battery using app during the course of his usage based on a certain set threshold.

______________________, Committee Chair
Du Zhang, Ph.D.

______________________
Date
DEDICATION

To my Parents
ACKNOWLEDGEMENTS

I would like to thank Dr. Du Zhang, my advisor and guide for giving me this opportunity to work on this project. Without his support and invaluable time spent on this project, it would have been difficult for me to accomplish the goal of this project. Dr. Du Zhang always provided me the knowledge required to make this project a success. His expert advice made it easier for me to understand the project and finish the project successfully on time.

In addition, I would like to thank Dr. Bill Mitchell for his willingness to serve on the committee.

I would like to thank my family, for their love and support.

In addition, I would like to thank the faculty and staff of Department of Computer Science as California State University, Sacramento.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dedication</td>
<td>vii</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>viii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>xi</td>
</tr>
<tr>
<td>Chapter</td>
<td></td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Report Organization</td>
<td>2</td>
</tr>
<tr>
<td>2. RELATED WORK</td>
<td>3</td>
</tr>
<tr>
<td>3. TECHNOLOGY</td>
<td>5</td>
</tr>
<tr>
<td>3.1 Android</td>
<td>5</td>
</tr>
<tr>
<td>3.2 Main Blocks of Android</td>
<td>6</td>
</tr>
<tr>
<td>3.2.1 Activity</td>
<td>7</td>
</tr>
<tr>
<td>3.2.2 Broadcast Receivers</td>
<td>9</td>
</tr>
<tr>
<td>3.2.3 Intent</td>
<td>10</td>
</tr>
<tr>
<td>3.2.4 Service</td>
<td>11</td>
</tr>
<tr>
<td>3.2.5 Shared Preferences</td>
<td>14</td>
</tr>
<tr>
<td>3.2.6 Manifest File</td>
<td>14</td>
</tr>
<tr>
<td>3.3 Architecture of Android</td>
<td>16</td>
</tr>
<tr>
<td>3.4 Tools Required</td>
<td>17</td>
</tr>
</tbody>
</table>

ix
4. DESIGN ........................................................................................................18
   4.1 Functional Requirements ........................................................................18
   4.2 Use Case Diagram ..................................................................................21
   4.3 Data Flow Diagram ................................................................................22
5. IMPLEMENTATION ........................................................................................24
   5.1 Algorithm Approach ..............................................................................24
   5.2 Favorite Apps .......................................................................................25
      5.2.1 Code Snippet for Favorite App List ...................................................26
      5.2.2 Screenshot for Favorite App List ......................................................28
   5.3 Comparison ............................................................................................29
      5.3.1 Code Snippet for Comparison ..........................................................30
      5.3.1 Screenshot for Comparison ................................................................32
   5.4 Average Battery Use ..............................................................................32
      5.4.1 Code Snippet for Average Battery Usage ............................................33
   5.5 Shortcut Icons on Home Screen ..............................................................34
      5.5.1 Code Snippet for Shortcut Icons .......................................................36
   5.6 Notification .............................................................................................38
      5.6.1 Code Snippet for Notification ............................................................39
      5.6.1 Screenshot for Notification ...............................................................41
6. CONCLUSION ...............................................................................................42
   6.1 Conclusion ..............................................................................................42
6.2 Performance Analysis .................................................................43
6.3 Future Work .............................................................................43
Appendix Source Code ....................................................................45
Bibliography .....................................................................................86
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figures</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Android Activity Life Cycle</td>
</tr>
<tr>
<td>2</td>
<td>Android Service Life Cycle</td>
</tr>
<tr>
<td>3</td>
<td>Architecture of Android</td>
</tr>
<tr>
<td>4</td>
<td>Use Case Diagram of Application</td>
</tr>
<tr>
<td>5</td>
<td>Data Flow Diagram</td>
</tr>
<tr>
<td>6</td>
<td>Flow Chart for Favorite Apps</td>
</tr>
<tr>
<td>7</td>
<td>Screenshot for Favorite App List</td>
</tr>
<tr>
<td>8</td>
<td>Flow Chart for Comparison</td>
</tr>
<tr>
<td>9</td>
<td>Screenshot for Comparison</td>
</tr>
<tr>
<td>10</td>
<td>Flow Chart for Average Battery Usage</td>
</tr>
<tr>
<td>11</td>
<td>Flow Chart for Shortcut Icons</td>
</tr>
<tr>
<td>12</td>
<td>Flow Chart for Notification</td>
</tr>
<tr>
<td>13</td>
<td>Screenshot for Notification</td>
</tr>
</tbody>
</table>
Chapter 1
INTRODUCTION

The objective of the project is to design a learning application for android-based devices that will track the user’s behavior pattern and allow the user to manage his device smartly. There are a few existing apps that are already exhibiting this behavior for example App Usage Tracker [5]. However, Smart app does provide more knowledgeable actions compared to others (for example: comparison between user’s top favorites and user’s calculated favorites, easy access to the favorite apps by creating shortcuts on the home screen, etc). App Usage Tracker [5] application is also capable of learning the user’s mobile behavioral patterns. The behavioral patterns captured would include number of things that would help the mobile users to combine statistics and provide insights into mobile device usage. For instance, Smart will tell which applications the user uses frequently on a certain period like daily, weekly and monthly by which the user will come to know which apps he/she uses regularly. That would be the user’s favorite app—“the app used most frequently”. There would also be comparison between the user’s top rated apps (wherein user gives initial rating to the apps according to his choice) v/s user’s calculated favorite rating (which would be coming from the learning done on the user’s behavior patterns). Along with this learned data of user’s behavior patterns, the application also provides an insight into the average power consumption for the apps. This app also uses the data gathered from learning user's patterns to convert them into some knowledgeable actions that would enable effective device use. For instance,
creating a shortcut of the learned favorite apps to the home screen. The list will be refreshed every time the application is run. In this way, the old favorite list will be refreshed automatically.

The user will also be notified for the maximum battery-using app during the course of his usage based on a certain set threshold.

1.1 Report Organization

The project report is organized into six chapters.

First chapter comprises of introduction of the project. Second chapter comprises of the related work done. Third chapter comprises of technology used for building this project. Fourth chapter comprises of the functional requirement, use case diagram and the data flow diagram. Fifth chapter deals with the implementation details. The last chapter comprises of the conclusion, performance analysis and the future work of our project.
Chapter 2

RELATED WORK

There are a few learning apps that track the user’s behavior pattern. App Usage Tracker [5] is an app on android platform, which tracks the user’s behavior pattern and uses the data gathered to monitor regularly which apps are used often. The same information is depicted graphically. It also has the feature to export out the usage stats. The application starts tracking as soon as the app is launched. For the very first time the usage report which will show the usage analysis will be empty. The time-series graph showing the usage pattern for a particular app can be viewed by tapping on the app from the list. It also exports usage through email. It also notifies the user with the most used app daily. The daily notification is enabled by default. In this way, this app is useful in tracking the most used apps on daily basis.

BreakFree[6] is an app, that will help you maintain a controlled digital lifestyle. It monitors phone and app usage and tracks how addicted you are to your phone and apps. With visually appealing characters, graphs and stats, the app, in its own way will help you control phone usage. The app in the background studies usage pattern and warns you with timely notifications whenever you go overboard. For example, if you use a particular app for too long, if you use the phone over an hour, etc. With these notifications, you know it is time to slow down. The app calculates an addiction score for you in real-time and maintains a history for the same. The app keeps the user engaged by
communicating results via a friendly but a tad bit exaggerated character called Sato. The app also provides phone management tools to help you relax. Tools such as disabling the internet, rejecting phone calls, sending auto text messages, etc. will provide peace of mind when you need it.

You can also view call stats and stats for your top used apps. This will indicate how much time you are spending on calls and apps and where you can reduce. For example, if you have launched WhatsApp 30 times a day, then it’s time to take a step back and cut down on that.
Chapter 3

TECHNOLOGY

3.1 Android

Android is a Linux-based mobile phone OS developed through Open Handset Alliance led by Google [3]. It comes with open source libraries for application development. Google is developing the platform but giving it free to hardware manufacturers and phone carriers. Android relies on Linux kernel version 2.6.x for system services such as process management, memory management, security, and driver model. Android consists of a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language [3].

Android's user interface is based on human-computer interaction style which involves continuous representation of objects of interest, and incremental actions and feedback using touch inputs that correspond to real-world actions, like swiping and, tapping to manipulate on-screen objects[3].

Android is made of the following important parts such as, Libraries, Linux Kernel, Android run time, Application framework, software development kit and native and third party applications. It consists of various open source libraries for application development like SQLite, OpenGL and media manager. The kernel also acts as a separation between the hardware and the rest of the software. Android run time includes Android core libraries and Dalvik Virtual Machine. Dalvik has been written so that a
device can run multiple VMs efficiently. The Dalvik VM executes files in the Dalvik Executable (.dex) format. The VM is register-based, and runs classes compiled by a Java compiler that have been transformed into the .dex format by the included "dx" tool. Application framework includes APIs and classes for developing native and third party applications [2].

The framework API consists of [4]

- A core set of packages and classes
- A set of XML elements and attributes for declaring a manifest file
- A set of XML elements and attributes for declaring and accessing resources.
- A set of permissions that application can request.

A software development kit used to create applications, including the tools, plug-ins, and documentation.

### 3.2 Main Blocks of Android

The main building block of android consists of the following

- Activity
- Broadcast Receivers
- Intent
- Service
- Shared Preference
- Manifest File
3.2.1 Activity

Activity in android is something that interacts with the user. Activity class is responsible for creating a window where in you the user can place all of your UI screens. An activity can be embedded inside another activity. That means a new activity can be started from the current activity. Each time a new activity starts, the previous activity is stopped. The system preserves the activity in a stack. When a new activity starts, it is pushed onto the back stack. Activity can make use of the Fragment class to make the UI better for large screens, and help scale the application between small and large screens. An application usually consists of multiple activities that are loosely bound to each other. The main activity is the activity, which runs while launching the application for the first time. When an activity is stopped because a new activity starts, it is notified of this change through the activity's lifecycle callback methods. All activities of an application are registered in the manifest file [1].

Life Cycle of Activity

An activity is made up of four states[1]

- If an activity is on the top of stack, it is active or running.
- If an activity has lost focus but is still visible, it is paused. A paused activity is completely alive, but can be killed.
- If an activity is interrupted by another activity, it is stopped. It retains the state but is no longer visible to the user.
If an activity is paused or stopped, the system can either ask it to finish it or can kill it. If the activity has to be displayed to the user again, it must be restarted and fully restored to its previous state.

The following diagram shows the important state paths of an Activity:

**Fig 1. Android Activity Life Cycle [1]**
OnCreate() is called when the activity is created. OnStart() and OnResume() will be called after the activity is created. OnDestroy() will be called when the activity is being destroyed.

### 3.2.2 Broadcast Receivers

A broadcast receiver is an Android component, which allows you to register for application events. All registered receivers for an event are notified by the Android runtime whenever this event happens. For example, applications can register for the ACTION_BOOT_COMPLETED system event, which is fired once the Android system has completed the boot process. If the event for which the broadcast receiver has registered happens, the onReceive() method of the receiver is called by the Android system. A broadcast receiver is declared in the manifest file. There are two major classes of broadcasts that can be received[1].

- **Normal Broadcasts**: They are completely asynchronous. They are sent with Context.sendBroadcast.
- **Ordered Broadcasts**: They are delivered to one receiver at a time. They are sent with Context.sendOrderedBroadcast in a specified order.
3.2.3 Intent

Intent gives description of an operation to be performed. Intent is an object you can use to request an action from another app component. The three main uses of intent are as follows[1]

- To start an activity: A new instance of an activity can be started by passing intent to `startActivity()`. If some result is expected from the activity when it is finished call `startActivityForResult()`.
- To start a service: You can start a service by passing intent to `startService()`. If the service is designed with a client-server interface, you can bind to the service from another component by passing an intent to `bindService()`.
- To deliver a broadcast: A broadcast can be delivered to other apps by passing an intent to `sendBroadcast()`, `sendOrderedBroadcast()`, or `sendStickyBroadcast()`.

There are two types of intent[1]

**Explicit intents**: Explicit intents specify the component to start by name. We use an explicit intent to start a component in our own app, because we know the class name of the activity we want to start.

**Implicit intents**: Implicit intents do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it. For example, if you want to show which user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map.
3.2.4 Service

A Service is a component that runs in background without user interaction. If the application is to run for a longer time without much user interaction, it can be implemented as service. A service is not bound to the lifecycle of an activity. Service is used for process like internet downloading, data processing, etc. Android system usually does not terminate a service[1].
Fig 2. Android Service Life Cycle [1]
Service can be started using the startService() method. This method takes the intent as argument. It can be stopped using stopService() method.

You can also specify that your service runs in a different process through the android:process=":process_description" attribute.

```xml
<service
android:name="wordService"
android:process=":my_process"
android:icon="@drawable/icon"
android:label="@string/service_name"
/>
</service>
```

If the colon prefix is used in front of the name then the service is private to its declaring application. If colon is not used then the service is global and could be used by other applications [1].
3.2.5 Shared Preferences

Shared preference is one of the ways to store data of an application. They save and retrieve the data in the form of key value pair. To use shared preference, getSharedPreferences() method has to be called.

```
SharedPreferences shpref = getSharedPreferences(MyPref, Context.MODE_PRIVATE)
```

The first parameter is the key and second parameter is the MODE. Each shared preferences file is handled by the framework. They can be either private or shared. Shared preferences can be used to save primitive data type like int, string, boolean, float, long. To write values call the edit() method of the sharedPreference.Editor class. To write the values you can use putString() or putBoolean() method. You can commit the added values with the commit() method [1].

3.2.6 Manifest file

A Manifest file is the file that represents information about the application to android. Typically, all the configuration of the android application goes in the Android Manifest.xml. This file is also known as Manifest. Manifest also has some additional metadata for the applications such as icons, version number of the application, and so on. This file is read by the Android system during the app installation. The android system checks the file configuration to know what the application can do. Manifest file also tells about the different components of the application like activities, services, and broadcast receivers. Broadcast receivers can be declared statically or dynamically at runtime of application. Manifest file also consists of all the permissions required for the application.
For example, if the application requires network access, it must be declared in the manifest file. It also declares the permissions that others must have in order to interact with the application. Manifest also names the Java package for the application. The package name remains unique for the application. If the java object lies in a different package, it must be used with full-qualified package name. Manifest file also declares the Android API that the application requires. Below is a code snippet from the manifest file of the application[1].

```xml
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.smart.PowerTutor"
    android:versionCode="15"
    android:versionName="1.5" >

    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name" >
        <activity
            android:name="ui.UMLogger"
            android:label="@string/app_name"
            android:screenOrientation="portrait" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

The above snippet tells about the app configuration as the unique package name used. It also tells the image icon used for the app, app name given to our application, screen orientation and so on.
3.3 Architecture of Android

Below is the architecture of android [7]

![Diagram of Android Architecture](image)

Fig 3. Architecture of Android [7]
3.4 Tools Required

- Android SDK 22.3 or higher
- ADT Plugin 20.0.0 or higher
- Eclipse 4.2.1.v2013
- Using SDK Manager download latest SDK tools
- Java SE 1.6.0
4.1 Functional Requirements

1) Favorite Apps: Once the app is installed and you start the profiler, the app starts learning the user’s usage pattern. There is an option for the user to view all the apps that are installed on the device. There is also a provision made for the user to submit his own rating to the apps based on his choice. For example, the user can rate Facebook as his top rated app. He will also be able to view his own created favorite app list.

Based on the learned knowledge of user’s user pattern, the app will generate a calculated favorite list for the user. This list is based on the most often apps used by the user but a more advanced scoring system might be more useful – for example, checking the weather once per day for 5 minutes is much less of an app use/preference than once a day spending an hour viewing/studying lengthy tutorials on software. In other words, scoring is typically implemented as a weighted sum over the different categories of measure like frequency of use, time duration of use, etc. The user can then view the calculated list of favorite apps, which he most often uses. This learning would be broken down into daily, weekly and monthly analysis. Therefore, in this way, the user can see his most often app usage on a daily, weekly and monthly basis.

This will give the user a picture of how he/she uses the device (apps) over a certain period. The user can then decide on how to cut-down the usage on what apps, how long to use a particular app, etc.
2) Comparison: This feature of the app will give the user an insight of his own personal favorite against his calculated favorite. The training data here selected is the list of all application on the device. Every application has been given a default ranking/rating. User then labels the data (apps) as per his choice (favorites). In this way, the user can set his favorites by rating the apps. The user can view the comparison between the apps he rated as his top favorites and the apps which he actually is being using. The user would be given a provision to rate a particular app according to his choice. The app then learns the usage pattern and scores accordingly on top the score that the user initially provides, if any. For example, if user rate Facebook with the highest score of 100 the app will start scoring on top the score, which the user has initially provided (training data).

An example below gives an overview of the comparison feature.

User has rated “XYZ” app as 500 and “ABC” as 480. Therefore, His top would be XYZ and then ABC.

<table>
<thead>
<tr>
<th>Initial</th>
<th>TOP CALCULATED</th>
<th>TOP FAVORITE</th>
</tr>
</thead>
<tbody>
<tr>
<td>XYZ</td>
<td>XYZ</td>
<td></td>
</tr>
<tr>
<td>ABC</td>
<td>ABC</td>
<td></td>
</tr>
</tbody>
</table>

Then user starts using ABC. Smart will record the activity and score it based on the algorithm. Therefore, after learning the usage pattern it knows that the usage score for ABC has gone above 500 so it would change the list accordingly.
3) Average Battery Usage: This section of the app would show the average battery usage of the apps. Using the information gathered from learning the usage pattern along with the services provided by android.

4) Shortcut Icons on Home Screen: This section of the app is to provide easy access to the favorite apps of the user. The idea here is to use the learned knowledge form the user’s usage pattern and use it into a knowledgeable action. Every time the app is run, shortcut icons for the top favorite apps are created on the home screen. In this way, the user can access the apps he/she likes easily. The list will be refreshed every time the app is run. If the favorites change over the period this list will be replaced by the new list. For example, if the app learns the favorite list of the user as Facebook, Gmail, Subway Surfers, it would create the shortcut for those applications. However, later if the list changes to something else like Chess, Whatsapp, Facebook then the old list shortcuts would be replaced by the new ones.

5) Notification for maximum battery using app: This section of the app deals with notifying the user whenever he/she goes overboard in using a particular app. The user will be notified with a toast message saying that the app is using more battery. The notification is based on a certain threshold value set. Once that threshold value is crossed, the user will be notified with the maximum battery using app message. For example, if the threshold value is set to be 2% of the total battery and user starts using gmail application. Once the gmail application consumes more than 2% of the overall battery,
the user will be shown a toast message on the screen saying gmail application uses more battery. It will also be shown as a notification in the notification bar of the device. In this way, the user will know that it is time to stop using the app and save some battery.

4.2 Use Case Diagram

Once the user starts the profiler, the application will start learning the usage behavior of the user. Based on the learning we provide various knowledgeable actions like tracking user’s favorite app, Comparison, Average Battery Usage, Shortcut Icons for favorite apps on the home screen, and Notification. Below is the use case diagram for the application.
4.3 Data Flow Diagram

Install the Smart application first. Then we need to start the profiler to begin with the learning process. Based on the learned knowledge we provide some knowledgeable
actions. In order to stop the learning process, we need to stop the profiler. Below is the data flow diagram for the application

Fig 5. Data Flow Diagram
Chapter 5

IMPLEMENTATION

5.1 Algorithm Approach

The approach used for this process tends towards machine-learned ranking model (MLR) [8]. We have used ordinal classification [8] supervised learning where the goal is to construct the ranking model from the training data. The training data here selected is the list of all applications on the device. Every application is given a default ranking/rating. User then labels the data (apps) as per his choice (favorites). In this way, the user can set his favorites by rating the apps. A provision has been made to change the label of the training data, so as for user to change his favorites.

The training sample contains the data related to number of hits, usage of the apps installed on the device and the ranking (in the initial case we consider default ranking). All the three parameters of the training data sample changes based on usage behavior of the user. Ordinal classification is applied on this training data set. The output of training data set will be classified into three categories like “Calculated top favorite”, “Calculated favorite”, and “Calculated least favorite”. Now the algorithm considers the “Calculated top favorite” classified data set as a reference to the construct the ranking model. The ranking is typically induced by giving a numerical score.
The score computation takes place as follows

- Whenever a new activity (app) is launched. The Smart application starts recording the activity.
- It gives (increments) the score for that activity (app). Records two seconds of the activity for 1 point.
- Adds up this newly recorded score on top of the rating (label) given to our training data.
- In this way, the usage pattern would be recorded and the data gathered is used to produce knowledgeable actions.

5.2 Favorite Apps

Using the algorithm discussed above in section 5.1, we calculate the favorite apps for the user. We are keeping track of all application in the phone, which are in running state. After calculation, these values are maintained in database in which all values are stored as well as which app was in foreground and background, and how many times the application was on screen. Below is the flow chart for Favorite Apps
5.2.1 Code Snippet for Favorite Apps

```java
public List getCurrentRunningActivities(){

    ActivityManager am = (ActivityManager)
    this.getSystemService(ACTIVITY_SERVICE);
    // get the info from the currently running task
    List<ActivityManager.RunningTaskInfo> taskInfo = am.getRunningTasks(1);
    Log.d("topActivity", "CURRENT Activity ::" +
    taskInfo.get(0).topActivity.getClassName());
    ComponentName componentInfo = taskInfo.get(0).topActivity;
    componentInfo.getPackageName();
    return taskInfo;
}

    // get the info from the currently running task

    case CAlculatedFavPage:
        prevPage = CAlculatedFavPage;
```
final Intent mainIntent = new Intent(Intent.ACTION_MAIN, null);
mainIntent.addCategory(Intent.CATEGORY_LAUNCHER);
final List pkgAppsList = this.getPackageManager().queryIntentActivities(mainIntent, 0);

//((Button)findViewById(R.id.buttonFrmBeg)).setOnClickListener(this);
((Button)findViewById(R.id.buttonFrmMonth)).setOnClickListener(this);
((Button)findViewById(R.id.buttonFrmWeek)).setOnClickListener(this);
((Button)findViewById(R.id.buttonLastDay)).setOnClickListener(this);
((Button)findViewById(R.id.buttonBackCalculated)).setOnClickListener(this);
(ListView)findViewById(R.id.listViewCalculatedApp)).setOnItemClickListener(this);

try {
    db = this.openOrCreateDatabase(DatabaseProvider.DATABASE_NAME, Context.MODE_PRIVATE, null);
    Cursor c = null;
    switch(periodSelected) {
        case TOTAL_PERIOD:
            c = db.rawQuery("SELECT * FROM " + DatabaseProvider.TABLE_NAME + " ORDER BY " + DatabaseProvider.COLUMN_NAME4UsageCount + " DESC ;",null);
            break;
        case MONTH_PERIOD:
            c = db.rawQuery("SELECT * FROM " + DatabaseProvider.TABLE_NAME + " ORDER BY " + DatabaseProvider.COLUMN_NAME9MONTHRATING + " DESC ;",null);
            break;
        case WEEK_PERIOD:
            c = db.rawQuery("SELECT * FROM " + DatabaseProvider.TABLE_NAME + " ORDER BY " + DatabaseProvider.COLUMN_NAME8WEEKRATING + " DESC ;",null);
            break;
        case LAST_DAY_PERIOD:
            c = db.rawQuery("SELECT * FROM " + DatabaseProvider.TABLE_NAME + " ORDER BY " + DatabaseProvider.COLUMN_NAME10PER_DAY_RATING + " DESC ;",null);
            break;
        }
    return c;
}
5.2.2 Screenshot for Favorite App List

![Fig 7. Screenshot of Favorite App List](image)
5.3 Comparison

As discussed in the above section 5.2, the top favorite apps for a user is calculated which is compared against the top favorite apps that user rates as his favorites. User can submit his rating for the app. Below is the flow chart for Comparison

![Flow chart for Comparison](image-url)

Fig 8. Flow chart for Comparison
5.3.1 Code Snippet for Comparison

```java
public void displayCalculatedFirstThree()
{
    try {
        db = this.openOrCreateDatabase(DatabaseProvider.DATABASE_NAME, Context.MODE_PRIVATE, null);
        Cursor c = db.rawQuery("SELECT * FROM "+ DatabaseProvider.TABLE_NAME + " ORDER BY "+ DatabaseProvider.COLUMN_NAME4UsageCount + " DESC ;", null);
        c.moveToFirst();
        String [] lstData1 = new String[c.getCount()];
        Log.e("Count",""+c.getCount());
        for(int i = 0 ;i < 3 ;i++)
        {
            switch(i)
            {
                case 0:
                    ((TextView)findViewById(R.id.textViewCalculated1)).setText(c.getString(c.getColumnIndex(DatabaseProvider.COLUMN_NAME1AppName))) ;// c.getInt(c.getColumnIndex(DatabaseProvider.COLUMN_NAME4UsageCount));
                    break;
                case 1:
                    ((TextView)findViewById(R.id.TextViewCalculated2)).setText(c.getString(c.getColumnIndex(DatabaseProvider.COLUMN_NAME1AppName))) ;// c.getInt(c.getColumnIndex(DatabaseProvider.COLUMN_NAME4UsageCount));
                    break;
                case 2:
                    ((TextView)findViewById(R.id.TextViewCalculated3)).setText(c.getString(c.getColumnIndex(DatabaseProvider.COLUMN_NAME1AppName))) ;// c.getInt(c.getColumnIndex(DatabaseProvider.COLUMN_NAME4UsageCount));
                    break;
            }
            c.moveToNext() ;
        }
        db.close() ;
    } catch (Exception e) {
        // TODO: handle exception
    }
```
public void displayCustomFirstThree() {
    try {
        SQLiteDatabase db = this.openOrCreateDatabase(DatabaseProvier.DATABASE_NAME, Context.MODE_PRIVATE, null);
        Cursor c = db.rawQuery("SELECT * FROM "+DatabaseProvier.TABLE_NAME + " ORDER BY "+DatabaseProvier.COLUMN_NAME6CustomUsageCount +" DESC ;", null);
        c.moveToFirst();
        for (int i = 0; i < 3; i++) {
            switch (i) {
            case 0:
                ((TextView)findViewById(R.id.textViewFav1)).setText(c.getString(c.getColumnIndex(DatabaseProvier.COLUMN_NAME1AppName)));
                c.getInt(c.getColumnIndex(DatabaseProvier.COLUMN_NAME4UsageCount));
                break;
            case 1:
                ((TextView)findViewById(R.id.TextViewFav2)).setText(c.getString(c.getColumnIndex(DatabaseProvier.COLUMN_NAME1AppName)));
                c.getInt(c.getColumnIndex(DatabaseProvier.COLUMN_NAME4UsageCount));
                break;
            case 2:
                ((TextView)findViewById(R.id.textViewFav3)).setText(c.getString(c.getColumnIndex(DatabaseProvier.COLUMN_NAME1AppName)));
                c.getInt(c.getColumnIndex(DatabaseProvier.COLUMN_NAME4UsageCount));
                break;
            }
        }
    } catch (Exception e) {
        db.close();
    }
}

5.3.2 Screenshot for Comparison

![Screenshot for Comparison](image)

**Fig 9. Screenshot for Comparison**

5.4 Average Battery Use

For the battery usage calculation we consider CPU usage and screen (LCD/LED) usage, a relative calculation is done by what % of CPU usage consumes one level of battery, and same for screen. For all the above we maintain a file which stores all the information.
Resources we calculate for battery usage are Audio, CPU, GPS, LCD/LED, Sensors, cellular and WIFI. Below is the flow chart for Average Battery Usage

![Flow Chart for Average Battery Usage](image)

**Fig 10. Flow Chart for Average Battery Usage**

### 5.4.1 Code Snippet for Average Battery Usage

```java
private double calcPower(PowerEstimator p, int powId) {
    switch(powId) {
    case POWER_INSTANT: {
        int count = 0;
        int[] history = p.getComponentHistory(5 * 60, -1,
                                               SystemInfo.AID_ALL, -1);
        double weightedAvgPower = 0;
        for(int i = history.length - 1; i >= 0; i--) {
            if(history[i] != 0) {
```

count++;
weightedAvgPower *= 1.0 - POLY_WEIGHT;
weightedAvgPower += POLY_WEIGHT * history[i] / 1000.0;
}
}
if(count == 0) return -1.0;
return weightedAvgPower / (1.0 - Math.pow(1.0 - POLY_WEIGHT, count));
} case POWER_MINUTE:
  case POWER_HOUR:
  case POWER_DAY:
  case POWER_TOTAL: {
    int wind = 0;
    if(powId == POWER_MINUTE) wind = Counter.WINDOW_MINUTE;
    if(powId == POWER_HOUR) wind = Counter.WINDOW_HOUR;
    if(powId == POWER_DAY) wind = Counter.WINDOW_DAY;
    if(powId == POWER_TOTAL) wind = Counter.WINDOW_TOTAL;
    double total = 0;
    for(long x : p.getMeans(SystemInfo.AID_ALL, wind)) {
      total += x / 1000.0;
    }
    return total;
  } case POWER_SENSOR: {
    BatteryStats bst = BatteryStats.getInstance();
    double curr = bst.getCurrent();
    if(curr >= 0) return -1.0;
    return curr * bst.getVoltage();
  }
}
return -1.0;

5.5 Shortcut Icons on Home Screen

Based on the learned top favorite’s shortcut icons are created on the home screen every
time the app is run. This list will be refreshed every time there is a change in the top
favorites. Below is the flow diagram for shortcut icons
Fig 11. Flow Chart for Shortcut Icons
5.5.1 Code Snippet for Shortcut Icons

```java
public void createShortcuts() {
    try {
        BatteryMonitor.db = openOrCreateDatabase(
                DatabaseProvider.DATABASE_NAME, MODE_WORLD_READABLE, null);
        final PackageManager pm = getPackageManager();
        Intent mainIntent = new Intent(Intent.ACTION_MAIN, null);
        mainIntent.addCategory(Intent.CATEGORY_LAUNCHER);
        List<ResolveInfo> appList = pm.queryIntentActivities(mainIntent, 0);
        Collections
                .sort(appList, new ResolveInfo.DisplayNameComparator(pm));
        Cursor c = BatteryMonitor.db.rawQuery("SELECT * FROM "
                + DatabaseProvider.TABLE_NAME + " ORDER BY "
                + DatabaseProvider.COLUMN_NAME4UsageCount + " DESC ;",
                null);
        c.moveToFirst();
        for (int i = 0; i < 3; i++) {
            for (ResolveInfo temp : appList) {
                if (c.getString(
                        c.getColumnIndex(DatabaseProvider.COLUMN_NAME2PackageName))
                        .equals(temp.activityInfo.packageName)) {
                    if (temp.activityInfo.packageName
                            .contains("com.smart.PowerTutor")) {
                        BatteryMonitor.db
                                .execSQL("CREATE TABLE IF NOT EXISTS "+
                                        DatabaseProvider.TABLE_ICON_CREATED
                                        + "(" + DatabaseProvider/icon1PackageName+ " VARCHAR, "+
                                        DatabaseProvider/icon2AppName+ " VARCHAR, "+
                                        DatabaseProvider/icon3IconName+ " VARCHAR, "+
                                        DatabaseProvider/icon4IconId
                                        + " INT) ;");
                        BatteryMonitor.db.execSQL("INSERT INTO "+
                                DatabaseProvider.TABLE_ICON_CREATED+ " VALUES ("+
                                temp.activityInfo.packageName + ",","+ temp.activityInfo.name + ","+ R.drawable.icon + ");");
```
// temp.activityInfo.loadIcon(getPackageManager());

createShortcutOnDesktop(temp.activityInfo.packageName,
    temp.activityInfo.name, "Smart", R.drawable.icon);

} else {

BatteryMonitor.db
    .execSQL("CREATE TABLE IF NOT EXISTS "+
DatabaseProviHerd.TABLE_ICON_CREATED
+ "("+ DatabaseProviHerd.ICON_COLUMN1Package+ " VARCHAR, "
+ DatabaseProviHerd.ICON_COLUMN2AppName+ " VARCHAR, "
+ DatabaseProviHerd.ICON_COLUMN3IconName+ " VARCHAR, "
+ DatabaseProviHerd.ICON_COLUMN4IconId
+ " INT) ;");
BatteryMonitor.db
    .execSQL("INSERT INTO "+ DatabaseProviHerd.TABLE_ICON_CREATED
+ " VALUES ("+ temp.activityInfo.packageName+ ","+ temp.activityInfo.name
+ ","+ c.getString(c.getColumnIndex(DatabaseProviHerd.COLUMN_NAME1AppName))
+ ","+ R.drawable.icon + ");");

createShortcutOnDesktop(temp.activityInfo.packageName, temp.activityInfo.name,
c.getString(c.getColumnIndex(DatabaseProviHerd.COLUMN_NAME1AppName)),
    R.drawable.icon);
}

Log.v("my logs", "package and activity name =" + temp.activityInfo.packageName + ""
    + temp.activityInfo.name);

c.moveToNext();
}
BatteryMonitor.db.close();

} catch (Exception ex) {
Log.e("Exception", "" + ex);
}
5.6 Notification

A certain threshold value is set for the user to be notified for a certain application using more battery once this level is crossed the user will be notified with a notification saying that the app is using more battery. Below is the flow chart for Notification

Fig 12. Flow Chart for Notification
public static void setBatteryLevel(int level) {
    try {
        ActivityManager am = (ActivityManager) UMLogger.cContext
                .getSystemService(UMLogger.cContext.ACTIVITY_SERVICE);
        // get the info from the currently running task
        List<ActivityManager.RunningTaskInfo> taskInfo = am
                .getRunningTasks(1);
        Log.d("topActivity", "CURRENT Activity ::"
                + taskInfo.get(0).topActivity.getClassName());
        ComponentName componentInfo = taskInfo.get(0).topActivity;
        componentInfo.getPackageName();
        // //////////////////////////////////
        BatteryMonitor.db = BatteryMonitor.AppContext.openOrCreateDatabase(
                DatabaseProvioder.DATABASE_NAME,
                UMLogger.cContext.MODE_PRIVATE, null);
        // //////////////
        Cursor c = BatteryMonitor.db.rawQuery("SELECT * FROM 
                + DatabaseProvioder.TABLE_NAME + " WHERE 
                + DatabaseProvioder.COLUMN_NAME2PackageName + " LIKE " 
                + componentInfo.getPackageName() + ",;", null);
        c.moveToFirst();
        Log.i("cursor data ", c.toString());
        level = c
                .getInt(c
                .getColumnIndex(DatabaseProvioder.COLUMN_NAME3BatteryUsage)) + 1;
        BatteryMonitor.db.execSQL("UPDATE " + DatabaseProvioder.TABLE_NAME 
                + " SET " + DatabaseProvioder.COLUMN_NAME3BatteryUsage 
                + "=" + level + " WHERE 
                + DatabaseProvioder.COLUMN_NAME2PackageName + "=" 
                + componentInfo.getPackageName() + ",; ");
        // //notification code
        // Toast.makeText(cContext, "hi123 1 "+level,
        // Toast.LENGTH_SHORT).show();
Toast.makeText(cContext,
c.getString(c.getColumnIndex(DatabaseProvoider.COLUMN_NAME1AppName)) + "
application uses more energy."
" + ".Toast.LENGTH_LONG).show();
if (level > 1) {
Log.i("my info", level + "level");
// Displays a toast which will notify user about the application
// is used more battery

Intent intent = new Intent(cContext, UMLogger.class);
PendingIntent pIntent = PendingIntent.getActivity(cContext, 0,
intent, 0);
intent.putExtra("KILL", true);
PendingIntent pIntentkill = PendingIntent.getActivity(cContext,
0, intent, 0);

String temp=" "
+ c.getString(c.getColumnIndex(DatabaseProvoider.COLUMN_NAME1AppName))
+ " application uses more energy. ";
NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(cContext)
.setContentTitle("More Battery Use")
.setSmallIcon(R.drawable.icon)
.setContentText(temp)
.addAction(R.drawable.power_on, "Skip", pIntent)
.addAction(R.drawable.power_off, "Kill", pIntentkill);

NotificationManager mNotifyMgr = (NotificationManager) cContext
.getSystemService(NOTIFICATION_SERVICE);
mNotifyMgr.notify(0, mBuilder.build());
Toast.makeText(cContext,temp, Toast.LENGTH_LONG).show(

// Toast.makeText(cContext, "hi123 1 " + level, Toast.LENGTH_SHORT).show();
Log.e("appUsedBaterry", "" + level);
BatteryMonitor.db.close();
5.6.2 Screenshot for Notification

![Screenshot for Notification](image)

Fig 13. Screenshot for Notification
Chapter 6

CONCLUSION

6.1 Conclusion

Based on the learned knowledge of user’s behavior pattern, Smart provides a way for the mobile user to manage his device wisely.

With the use of this app, user can manage his time of the mobile device usage. He will know on how to cut down his usage of a particular app, so that he can save both time and battery. The breakdown of the period into daily, weekly, and monthly basis, will tell the user the apps he has been using frequently; along with the apps he stopped using. In this way, the user can go ahead and delete the apps which he stopped using and save some space on the device.

We have used the learned data to provide some knowledgeable actions, such as comparison, easy access, notification, which will help the user use his device smartly and with ease.
6.2 Performance Analysis

Using Smart application, we can in a way increase the performance of the device by killing the app consuming more power. With the notification for maximum battery use, the app will give an option of killing the power consuming app. Smart only uses cpu as the resource for the calculation and does not use other resources like sensors, wifi etc. Normally Smart app uses less than 3 % resource. For example, if at this time the overall cpu utilization of the device is around 76% than our app would be using less than 3 % out of it. In comparison to App Usage Tracker [5] which normally consumes less battery. The accuracy of tracking can also be controlled. There is a tradeoff here. If fine accuracy is required it would use more CPU and battery. Other levels would be a little less accurate but would be far on CPU and battery [5]. The user interface for Smart is more interactive than App Usage Tracker [5], while the user interface for BreakFree[6] is more interactive than Smart because of the friendly character called Sato through which you get the notification [6].

6.3 Future Work

The current Smart application is an application demonstrating a decent amount of knowledgeable actions on the learned data. For future work, we can add more knowledgeable actions to it and make it even smarter then what it is currently now. The following features can be added as future work. Recommendation is one of the features, which can be implemented. Based on the learned data for user’s top favorite over a monthly period, the user would be given recommendation on the apps of his favorite
category in the Android store that he can install and use. For example, the user’s top favorite is a puzzle game; he would be recommended a similar game from the app store. Reminders and customized notifications can be other features that can be implemented as future work to the existing app.
package com.smart.PowerTutor.service;
import com.smart.PowerTutor.R;
import com.smart.PowerTutor.ui.PowerTabs;
import com.smart.PowerTutor.ui.UMLogger;
import com.smart.PowerTutor.util.BatteryStats;
import com.smart.PowerTutor.util.SystemInfo;

import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.app.Service;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.BatteryManager;
import android.os.Bundle;
import android.os.IBinder;
import android.telephony.PhoneStateListener;
import android.telephony.ServiceState;
import android.telephony.TelephonyManager;
import android.util.Log;

import java.io.ByteArrayOutputStream;
import java.io.ObjectOutputStream;
import java.io.IOException;
import java.lang.reflect.Method;
import java.lang.reflect.InvocationTargetException;

public class UMLoggerService extends Service{
    private static final String TAG = "UMLoggerService";
    private static final int NOTIFICATION_ID = 1;
    private static final int NOTIFICATION_ID_LETTER = 2;
}
private Thread estimatorThread;
private PowerEstimator powerEstimator;

private Notification notification;
private NotificationManager notificationManager;
private TelephonyManager phoneManager;

@Override
public IBinder onBind(Intent intent) {
    return binder;
}

@Override
public void onCreate() {
    powerEstimator = new PowerEstimator(this);
    /* Register to receive phone state messages. */
    phoneManager = (TelephonyManager) this.getSystemService(TELEPHONY_SERVICE);
    phoneManager.listen(phoneListener, PhoneStateListener.LISTEN_CALL_STATE |
                         PhoneStateListener.LISTEN_DATA_CONNECTION_STATE |
                         PhoneStateListener.LISTEN_SERVICE_STATE |
                         PhoneStateListener.LISTEN_SIGNAL_STRENGTH);

    /* Register to receive airplane mode and battery low messages. */
    IntentFilter filter = new IntentFilter();
    filter.addAction(Intent.ACTION_AIRPLANE_MODE_CHANGED);
    filter.addAction(Intent.ACTION_BATTERY_LOW);
    filter.addAction(Intent.ACTION_BATTERY_CHANGED);
    filter.addAction(Intent.ACTION_PACKAGE_REMOVED);
    filter.addAction(Intent.ACTION_PACKAGE_REPLACED);
    registerReceiver(broadcastIntentReceiver, filter);

    notificationManager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
}

@Override
public void onStart(Intent intent, int startId) {
    super.onStart(intent, startId);
    //android.os.Debug.startMethodTracing("pt.trace");
if(intent.getBooleanExtra("stop", false)) {
    stopSelf();
    return;
} else if(estimatorThread != null) {
    return;
}
showNotification();
estimatorThread = new Thread(powerEstimator);
estimatorThread.start();
}

@Override
public void onDestroy() {
    //android.os.Debug.stopMethodTracing();
    if(estimatorThread != null) {
        estimatorThread.interrupt();
        while(estimatorThread.isAlive()) {
            try {
                estimatorThread.join();
            } catch(InterruptedException e) {
            }
        }
    }
    unregisterReceiver(broadcastIntentReceiver);

    /* See comments in showNotification() for why we are using reflection here.
    */
    boolean foregroundSet = false;
    try {
        Method stopForeground = getClass().getMethod("stopForeground",
            boolean.class);
        stopForeground.invoke(this, true);
        foregroundSet = true;
    } catch (InvocationTargetException e) {
    } catch (IllegalAccessException e) {
    } catch(NoSuchMethodException e) {
    }
    if(!foregroundSet) {
        // stopForeground(false);
        notificationManager.cancel(NOTIFICATION_ID);
    }
    super.onDestroy();
}
/** This function is to construct the real-time updating notification*/
public void showNotification(){
  int icon = R.drawable.level;

  // icon from resources
  CharSequence tickerText = "BatteryUse";       // ticker-text
  long when = System.currentTimeMillis();         // notification time
  Context context = getApplicationContext();    // application Context
  CharSequence contentTitle = "SmartBatteryUse"; // expanded message title
  CharSequence contentText = "";                // expanded message text

  Intent notificationIntent = new Intent(this, UMLgger.class);
  notificationIntent.putExtra("isFromIcon", true);
  PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
              notificationIntent,
              PendingIntent.FLAG_UPDATE_CURRENT);

  /* the next two lines initialize the Notification, using the
   * configurations above.
   */
  notification = new Notification(icon, tickerText, when);
  notification.iconLevel = 2;
  notification.setLatestEventInfo(context, contentTitle,
                                   contentText, contentIntent);

  /* We need to set the service to run in the foreground so that system
   * won't try to destroy the power logging service except in the most
   * critical situations (which should be fairly rare). Due to differences
   * in apis across versions of android we have to use reflection. The newer
   * api simultaneously sets an app to be in the foreground while adding a
   * notification icon so services can't 'hide' in the foreground.
   * In the new api the old call, setForeground, does nothing.
   */
  boolean foregroundSet = false;
  try {
    Method startForeground = getClass().getMethod("startForeground",
                 int.class, Notification.class);
    startForeground.invoke(this, NOTIFICATION_ID, notification);
    foregroundSet = true;
  } catch (InvocationTargetException e) {
  } catch (IllegalAccessException e) {
  
} catch (InvocationTargetException e) {
  } catch (IllegalAccessException e) {
catch(NoSuchMethodException e) {
}
if(!foregroundSet) {
    // stopForeground(true);
    notificationManager.notify(NOTIFICATION_ID, notification);
}

/** This function is to update the notification in real time. This function
* is apparently fairly expensive cpu wise. Updating once a second caused a
* 8% cpu utilization penalty.
*/
public void updateNotification(int level, double totalPower) {
    notification.icon = R.drawable.level;
    notification.iconLevel = level;

    // If we know how much charge the battery has left we'll override the
    // normal icon with one that indicates how much time the user can expect
    // left.
    BatteryStats bst = BatteryStats.getInstance();
    if(bst.hasCharge() && bst.hasVoltage()) {
        double charge = bst.getCharge();
        double volt = bst.getVoltage();
        if(charge > 0 && volt > 0) {
            notification.icon = R.drawable.time;
            double minutes = charge * volt / (totalPower / 1000) / 60;
            if(minutes < 55) {
                notification.iconLevel = 1 +
                (int)Math.max(0, Math.round(minutes / 10.0) - 1);
            } else {
                notification.iconLevel = (int)Math.min(13,
                6 + Math.max(0, Math.round(minutes / 60.0) - 1));
            }
        }
    }

    CharSequence contentTitle = "Smart";
    CharSequence contentText = "Total Power: " + (int)Math.round(totalPower) +
    " mW";

    /* When the user selects the notification the tab view for global power
    * usage will appear. */
Intent notificationIntent = new Intent(this, UMLogger.class);
notificationIntent.putExtra("isFromIcon", true);
PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
        notificationIntent, 0);
notification.setLatestEventInfo(this, contentTitle, contentText,
        contentIntent);
notificationManager.notify(NOTIFICATION_ID, notification);
}

private final ICounterService.Stub binder =
    new ICounterService.Stub() {
        public String[] getComponents() {
            return powerEstimator.getComponents();
        }

        public int[] getComponentsMaxPower() {
            return powerEstimator.getComponentsMaxPower();
        }

        public int getNoUidMask() {
            return powerEstimator.getNoUidMask();
        }

        public int[] getComponentHistory(int count, int componentId, int uid) {
            return powerEstimator.getComponentHistory(count, componentId, uid, -1);
        }

        public long[] getTotals(int uid, int windowType) {
            return powerEstimator.getTotals(uid, windowType);
        }

        public long getRuntime(int uid, int windowType) {
            return powerEstimator.getRuntime(uid, windowType);
        }

        public long[] getMeans(int uid, int windowType) {
            return powerEstimator.getMeans(uid, windowType);
        }

        public byte[] getUidInfo(int windowType, int ignoreMask) {
            UidInfo[] infos = powerEstimator.getUidInfo(windowType, ignoreMask);
            ByteArrayOutputStream output = new ByteArrayOutputStream();
try {
    new ObjectOutputStream(output).writeObject(infos);
} catch (IOException e) {
    return null;
}

for (UidInfo info : infos) {
    info.recycle();
}
return output.toByteArray();

public long getUidExtra(String name, int uid) {
    return powerEstimator.getUidExtra(name, uid);
}
};

BroadcastReceiver broadcastIntentReceiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        if (intent.getAction().equals(Intent.ACTION_AIRPLANE_MODE_CHANGED)) {
            Bundle extra = intent.getExtras();
            try {
                if ((Boolean) extra.get("state")) {
                    powerEstimator.writeToLog("airplane-mode on\n");
                } else {
                    powerEstimator.writeToLog("airplane-mode off\n");
                }
            } catch (ClassCastException e) {
                // Some people apparently are having this problem. I'm not really
                // sure why this should happen.
                Log.w(TAG, "Couldn't determine airplane mode state");
            }
        } else if (intent.getAction().equals(Intent.ACTION_BATTERY_LOW)) {
            powerEstimator.writeToLog("battery low\n");
        } else if (intent.getAction().equals(Intent.ACTION_BATTERY_CHANGED)) {
            powerEstimator.plug(
                intent.getIntExtra("plugged", -1) != 0);
else if(intent.getAction().equals(Intent.ACTION_PACKAGE_REMOVED) || intent.getAction().equals(Intent.ACTION_PACKAGE_REPLACED)) {
    // A package has either been removed or its metadata has changed and we
    // need to clear the cache of metadata for that app.
    SystemInfo.getInstance().voidUidCache(intent.getIntExtra(Intent.EXTRA_UID, -1));
}

};

PhoneStateListener phoneListener = new PhoneStateListener() {
    public void onServiceStateChanged(ServiceState serviceState) {
        switch(serviceState.getState()) {
            case ServiceState.STATE_EMERGENCY_ONLY:
                powerEstimator.writeToLog("phone-service emergency-only\n");
                break;
            case ServiceState.STATE_IN_SERVICE:
                powerEstimator.writeToLog("phone-service in-service\n");
                switch(phoneManager.getNetworkType()) {
                    case(TelephonyManager.NETWORK_TYPE_EDGE):
                        powerEstimator.writeToLog("phone-network edge\n");
                        break;
                    case(TelephonyManager.NETWORK_TYPE_GPRS):
                        powerEstimator.writeToLog("phone-network GPRS\n");
                        break;
                    case 8:
                        powerEstimator.writeToLog("phone-network HSDPA\n");
                        break;
                    case(TelephonyManager.NETWORK_TYPE_UMTS):
                        powerEstimator.writeToLog("phone-network UMTS\n");
                        break;
                    default:
                        powerEstimator.writeToLog("phone-network " +
                            phoneManager.getNetworkType() + "\n");
                        break;
                }
                break;
            case ServiceState.STATE_OUT_OF_SERVICE:
                powerEstimator.writeToLog("phone-service out-of-service\n");
                break;
            case ServiceState.STATE_POWER_OFF:
                powerEstimator.writeToLog("phone-service power-off\n");
                break;
        }
public void onCallStateChanged(int state, String incomingNumber) {
    switch(state) {
    case TelephonyManager.CALL_STATE_IDLE:
        powerEstimator.writeToLog("phone-call idle\n");
        break;
    case TelephonyManager.CALL_STATE_OFFHOOK:
        powerEstimator.writeToLog("phone-call off-hook\n");
        break;
    case TelephonyManager.CALL_STATE_RINGING:
        powerEstimator.writeToLog("phone-call ringing\n");
        break;
    }
}

public void onDataConnectionStateChanged(int state) {
    switch(state) {
    case TelephonyManager.DATA_DISCONNECTED:
        powerEstimator.writeToLog("data disconnected\n");
        break;
    case TelephonyManager.DATA_CONNECTING:
        powerEstimator.writeToLog("data connecting\n");
        break;
    case TelephonyManager.DATA_CONNECTED:
        powerEstimator.writeToLog("data connected\n");
        break;
    case TelephonyManager.DATA_SUSPENDED:
        powerEstimator.writeToLog("data suspended\n");
        break;
    }
}

public void onSignalStrengthChanged(int asu) {
    powerEstimator.writeToLog("signal " + asu + "\n");
}

}
import java.io.FileWriter;
import java.io.IOException;
import java.sql.Date;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;
import com.smart.PowerTutor.R;
import com.smart.PowerTutor.ui.UMLogger;

import android.annotation.SuppressLint;
import android.app.Activity;
import android.app.ActivityManager;
import android.app.ActivityManager.RecentTaskInfo;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.pm.ApplicationInfo;
import android.content.pm.PackageInfo;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.ColorFilter;
import android.graphics.drawable.BitmapDrawable;
import android.graphics.drawable.Drawable;
import android.net.ConnectivityManager;
import android.net.TrafficStats;
import android.os.AsyncTask;
import android.os.BatteryManager;
import android.os.Bundle;
import android.os.Environment;
import android.preference.PreferenceManager;
import android.text.format.DateFormat;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageButton;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

@SuppressLint("NewApi")
public class BatteryMonitor extends Activity implements OnClickListener,
OnItemClickListener{

public static BatteryMonitor AppContext=null;
public static SQLiteDatabase db ;
static BatteryMonitor mainContext;
public static int bateryLevel = 0;
static int periodSelected = 0;
public static int TOTAL_PERIOD = 0;
public static int WEEK_PERIOD = 1;
public static int MONTH_PERIOD = 2;
public static int LAST_DAY_PERIOD = 3;
@Override
public void onCreate(Bundle icicle) {

    super.onCreate(icicle);
    AppContext = this ;
    page = MainPage ;
    loadPages();

    db =
    openOrCreateDatabase(DatabaseProvoider.DATABASE_NAME,MODE_WORLD_READABLE , null );

    db.execSQL("CREATE TABLE IF NOT EXISTS "+
DatabaseProvider.TABLE_NAME +"("+
DatabaseProvider.COLUMN_NAME1AppName +" VARCHAR,"+
DatabaseProvider.COLUMN_NAME2PackageName +" VARCHAR,"+
DatabaseProvider.COLUMN_NAME3BatteryUsage +" INT,"+
DatabaseProvider.COLUMN_NAME4UsageCount +" INT,"+
DatabaseProvider.COLUMN_NAME5DataUsage +" INT,"+

}
db.execSQL("CREATE TABLE IF NOT EXISTS " +
DatabaseProvider.TABLE_NAME_DATE +"( " +
DatabaseProvider.DATE_COLUMN_NAME1 +" INT, " +
DatabaseProvider.DATE_COLUMN_NAME2 +" INT, " +
DatabaseProvider.DATE_COLUMN_NAME3 +" INT ) ;");

db.execSQL("ALTER TABLE " + DatabaseProvider.TABLE_NAME + " ADD " +
DatabaseProvider.COLUMN_NAME6 + " INT ");

lastDataUsage = TrafficStats.getTotalRxBytes();
    PInfo temp = new PInfo();
    temp.appname = "MobileAppAnalyser";
    temp.icon = new Drawable() {
        @Override
        public void setColorFilter(ColorFilter cf) {
            // TODO Auto-generated method stub
        }
        @Override
        public void setAlpha(int alpha) {
            // TODO Auto-generated method stub
        }
        @Override
        public int getOpacity() {
            // TODO Auto-generated method stub
            return 0;
        }
        @Override
        public void draw(Canvas canvas) {
            // TODO Auto-generated method stub
        }
    };
    temp.pname = "com.mobware4u.batterymonitor";
    temp.versionName = "1";
    db = this.openOrCreateDatabase(DatabaseProvider.DATABASE_NAME, Context.MODE_PRIVATE, null);
Cursor c = db.rawQuery("SELECT * FROM " + DatabaseProvoider.TABLE_NAME
, null);
if (c.getCount() < 2) {
    ArrayList<PInfo> allApps = getPackages();
    allApps.add(temp);

    db = openOrCreateDatabase(DatabaseProvoider.DATABASE_NAME, MODE_PRIVATE
, null);
    for (ApplicationInfo app:
        getPackageManager().getInstalledApplications(0)) {
        if(app.packageName!=null) {
            db.execSQL("INSERT INTO " + DatabaseProvoider.TABLE_NAME + "
VALUES ('" + app.packageName.substring(app.packageName.lastIndexOf(".")) + 1,
app.packageName.length()) + "," + app.packageName + "," + 0 + "," + 0 + "," + 0 + "," + 0 + "," + app.uid + "," + 0 + "," + 0 + ");");
            Log.e("**APP","LIST" + app.uid
            + app.packageName.substring(app.packageName.lastIndexOf(".")) + 1,
            app.packageName.length());
        }
    }

    db.execSQL("UPDATE " + DatabaseProvoider.TABLE_NAME + " SET
" + DatabaseProvoider.COLUMN_NAME6CustomUsageCount + " = " + 0 + ";");

    // for(int i= 0 ;i < allApps.size();i++)
    // {                           
    //     db.execSQL("INSERT INTO " + DatabaseProvoider.TABLE_NAME + " VALUES
('" + allApps.get(i).appname.toString() + "," + allApps.get(i).pname + "," + 0 + "," + 0 + ");");
    //     Log.e("LIST",allApps.get(i).appname);
    // }
    // }
    db.execSQL("INSERT INTO " + DatabaseProvoider.TABLE_NAME_DATE + "
VALUES (" + 0 + "," + 0 + "," + 0 + ");");
    db.close();
public String getBatterStatus() {
    String status = ";
    SharedPreferences settings = PreferenceManager
        .getDefaultSharedPreferences(getApplicationContext());
    status = settings.getString("BatteryStatus", ";");
    return status;
}

public void setBatterStatus(String status) {
    SharedPreferences settings = PreferenceManager
        .getDefaultSharedPreferences(getApplicationContext());
    SharedPreferences.Editor editor = settings.edit();
    editor.putString("BatteryStatus", status);
    editor.commit();
}

public String getONOFF() {
    String status = ";
    SharedPreferences settings = PreferenceManager
        .getDefaultSharedPreferences(getApplicationContext());
    status = settings.getString("OnOff", "OFF");
    return status;
}

public void setONOFF(String status) {
    SharedPreferences settings = PreferenceManager
        .getDefaultSharedPreferences(getApplicationContext());
    SharedPreferences.Editor editor = settings.edit();
    editor.putString("OnOff", status);
    editor.commit();
}
public void onClick(View v) {
    int id = v.getId();
    switch(id) {
    case R.id.startB:
        String status = getONOFF();
        if(status.equalsIgnoreCase("ON")) {
            setONOFF("OFF");
            ((Button) findViewById(R.id.startB)).setText("Start Monitor");
            //stop service
            stopService(new Intent(BatteryMonitor.this, BatteryService.class));
            Toast.makeText(this, "Stopped", Toast.LENGTH_SHORT).show();
        } else {
            setONOFF("ON");
            ((Button) findViewById(R.id.startB)).setText("Stop Monitor");
            //start service
            startService(new Intent(BatteryMonitor.this, BatteryService.class));
            Toast.makeText(this, "Started", Toast.LENGTH_SHORT).show();
        }
        break;
    case R.id.backupB:
        String backupFilePath = Environment
                                   .getExternalStorageDirectory()
                                   .getAbsoluteFile().getAbsolutePath();
        backupFilePath += "/battery_usage.xls";

        String battery = getBatterStatus();

        try {
            FileWriter fw = new FileWriter(backupFilePath);
            fw.write(battery);
            fw.close();
        } catch (IOException e) {
            e.printStackTrace();
        }

        Toast.makeText(this,
                        "Saved to " + backupFilePath,
                        Toast.LENGTH_SHORT).show();
        break;
    }
case R.id.resetB:
    setBatterStatus("");
    Toast.makeText(this, "Reset", Toast.LENGTH_SHORT).show();
    break;

case R.id.allAppListB:
    page = AllAppListPage;
    loadPages();
    break;

case R.id.calculatedFB:
    page = CalculatedFavPage;
    loadPages();
    break;

case R.id.customFB:
    page = CustomFavPage;
    loadPages();
    break;

case R.id.buttonBackCustom:
    page = MainPage;
    loadPages();
    break;

case R.id.buttonBackAllApp:
    page = MainPage;
    loadPages();
    break;

case R.id.buttonFrmBeg:
    periodSelected = TOTAL_PERIOD;
    loadPages();
    break;

case R.id.buttonFrmMonth:
    periodSelected = MONTH_PERIOD;
    loadPages();
    break;

case R.id.buttonfrmWeek:
    periodSelected = WEEK_PERIOD;
    loadPages();
    break;

case R.id.buttonLastDay:
periodSelected = LAST_DAY_PERIOD;
loadPages();
break;

// case R.id.buttonBackCalculated:
//
// page = MainPage;
// loadPages();
// break;
//
case R.id.buttonSubmitRating:

String customRating =
((EditText)findViewById(R.id.editTextCustomFavRating)).getText().toString();

db =
openOrCreateDatabase(DatabaseProvider.DATABASE_NAME,MODE_PRIVATE, null);

db.execSQL("UPDATE "+ DatabaseProvider.TABLE_NAME +" SET "+DatabaseProvider.COLUMN_NAME6CustomUsageCount +" = "+customRating +", "+ DatabaseProvider.COLUMN_NAME4UsageCount +" = "+customRating +" WHERE "+ DatabaseProvider.COLUMN_NAME1AppName +" = "+(TextView)findViewById(R.id.textViewAppName).getText().toString() + " ;");

db.close();
((EditText)findViewById(R.id.editTextCustomFavRating)).setText(" ");

setDescriptionPage(((TextView)findViewById(R.id.textViewAppName)).getText().toString());
break;

case R.id.appIconImageButton:

// Intent intent = new Intent(Intent.ACTION_MAIN);
//    intent.getComponent(new ComponentName(packageName, packageName +
//    ((TextView)findViewById(R.id.textViewAppName)).getText().toString()));
//    startActivity(intent);
//    
//    Intent LaunchIntent = 
//    getPackageManager().getLaunchIntentForPackage(packageName +
//    ((TextView)findViewById(R.id.textViewAppName)).getText().toString());
//    startActivity( LaunchIntent );

PackageManager pmi = getPackageManager();
Intent intent = null;
Log.e("Pack name Launched", packageName);
intent = pmi.getLaunchIntentForPackage(packageName);
if (intent != null){
    startActivity(intent);
}
break;

case R.id.CompareBtn:

    page = ComparePage;
    setContentView(R.layout.compare);
    displayCalculatedFirstThree();
    displayCustomFirstThree();
    break;

case R.id.buttonNextMenu:

    page = NextMenuPage;
    loadPages();
    break;

case R.id.btnWifiUsage:
    page = WifiUsagePage;
    loadPages();
    break;
case R.id.buttonRefreshNetworkStatistics:
    refreshNetworkStatistics();
    break;

case R.id.btnBatteryUse:
    Intent intentBatteryUsage = new Intent(Intent.ACTION_POWER_USAGE_SUMMARY);
    startActivity(intentBatteryUsage);
    break;
}

public void displayCalculatedFirstThree()
{
    try
    {
        db = this.openOrCreateDatabase(DatabaseProvider.DATABASE_NAME, Context.MODE_PRIVATE, null);
        Cursor c = db.rawQuery("SELECT * FROM " + DatabaseProvider.TABLE_NAME + " ORDER BY " + DatabaseProvider.COLUMN_NAME4UsageCount + " DESC ;", null);
        c.moveToFirst();
        String[] lstData1 = new String[c.getCount()];
        Log.e("Count","" + c.getCount());
        for(int i = 0; i < 3; i++)
        {
            switch(i)
            {
                case 0:
                    ((TextView)findViewById(R.id.textViewCalculated1)).setText(c.getString(c.getColumnIndex(DatabaseProvider.COLUMN_NAME1AppName)));
                    c.getInt(c.getColumnIndex(DatabaseProvider.COLUMN_NAME4UsageCount));
                    break;
            }
        }
    }

    ((TextView)findViewById(R.id.textViewCalculated1)).setText(c.getString(c.getColumnIndex(DatabaseProvider.COLUMN_NAME1AppName)));
    c.getInt(c.getColumnIndex(DatabaseProvider.COLUMN_NAME4UsageCount));
break;

case 1:

((TextView)findViewById(R.id.TextViewCalculated2)).setText(c.getString(c.getColumnIndex(DatabaseProvoirer.COLUMN_NAME1AppName))) ;
c.getInt(c.getColumnIndex(DatabaseProvoirer.COLUMN_NAME4UsageCount));
break;

case 2:

((TextView)findViewById(R.id.textViewCalculated3)).setText(c.getString(c.getColumnIndex(DatabaseProvoirer.COLUMN_NAME1AppName))) ;
c.getInt(c.getColumnIndex(DatabaseProvoirer.COLUMN_NAME4UsageCount));
break;
}
c.moveToNext();
}
db.close();

// SimpleDateFormat dateFormat = new SimpleDateFormat(
// "yyy-MM-dd HH:mm:ss");
// Calendar cal = Calendar.getInstance();
// System.out.println("time => " + dateFormat.format(cal.getTime()));
//
//
// ((TextView)findViewById(R.id.textViewCalculated3)).setText("t:"+dateFormat.format(cal.getTime()));
//
// Calendar currentDate = Calendar.getInstance();
// SimpleDateFormat sdf = new SimpleDateFormat("MM/dd/yyyy");
//
// String dateNow = sdf.format(currentDate.getTime());
// Date today = new Date(dateNow);
// Date finalDay = null;
// finalDay = (Date) sdf.parse("MM/dd/yyyy");
// int numberOfDays=(int)((finalDay.getTime()-today.getTime())/(3600*24*1000));
//
}
catch (Exception e) {

}
public void displayCustomFirstThree()
{
    try {
        db = this.openOrCreateDatabase(DatabaseProvider.DATABASE_NAME, Context.MODE_PRIVATE, null);
        Cursor c = db.rawQuery("SELECT * FROM " + DatabaseProvider.TABLE_NAME + " ORDER BY " + DatabaseProvider.COLUMN_NAME6 + " DESC;", null);
        c.moveToFirst();
        for (int i = 0; i < 3; i++) {
            switch (i) {
                case 0:
                    ((TextView)findViewById(R.id.textViewFav1)).setText(c.getString(c.getColumnIndex(DatabaseProvider.COLUMN_NAME1))); //
                    c.getInt(c.getColumnIndex(DatabaseProvider.COLUMN_NAME4));
                    break;
                case 1:
                    ((TextView)findViewById(R.id.TextViewFav2)).setText(c.getString(c.getColumnIndex(DatabaseProvider.COLUMN_NAME1))); //
                    c.getInt(c.getColumnIndex(DatabaseProvider.COLUMN_NAME4));
                    break;
                case 2:
                    ((TextView)findViewById(R.id.textViewFav3)).setText(c.getString(c.getColumnIndex(DatabaseProvider.COLUMN_NAME1))); //
                    c.getInt(c.getColumnIndex(DatabaseProvider.COLUMN_NAME4));
                    break;
                default:
                    break;
            }
        }
    }
} // TODO: handle exception
}
DatabaseProvider.COLUMN_NAME1AppName)) ; //
c.getInt(c.getColumnIndex(DatabaseProvider.COLUMN_NAME4UsageCount));
    break;
  }
c.moveToNext() ;
}

db.close();
}
catch (Exception e) {
  // TODO: handle exception
}

}

public ArrayList<PInfo> get_packages() {
  ArrayList<PInfo> apps = getInstalledApps(false); /* false = no system packages */
  final int max = apps.size();
  for (int i=0; i<max; i++) {
    apps.get(i).prettyPrint();
  }
  return apps;
}

public ArrayList<PInfo> getInstalledApps(boolean getSysPackages) {
  ArrayList<PInfo> res = new ArrayList<PInfo>();
  List<PackageInfo> packs = getPackageManager().getInstalledPackages(0);
  for(int i=0;i<packs.size();i++) {
    PackageInfo p = packs.get(i);
    if (!getSysPackages && (p.versionName == null)) {
      continue ;
    }
    PInfo newInfo = new PInfo();
    newInfo.appname = p.applicationInfo.loadLabel(getPackageManager()).toString();
    newInfo.pname = p.packageName;
    newInfo.versionName = p.versionName;
    newInfo.versionCode = p.versionCode;
newInfo.icon = p.applicationInfo.loadIcon(getPackageManager());
res.add(newInfo);
}
return res;
}

///////////////////////////////most used
/////////////////////////////////

public static final int MainPage = 0;
public static final int AllAppListPage = 1;
public static final int CustomFavPage = 2;
public static final int CAIculatedFavPage = 3;
public static final int DescriptionPage = 4;
public static final int ComparePage = 5;
public static final int NextMenuPage = 6;
public static final int WifiUsagePage = 7;

public static int page = 0;
public int prevPage = 0;

public void loadPages()
{
    switch(page)
    {
    case MainPage:
    setContentView(R.layout.main1);
    //views on main screen
    ((Button) findViewById(R.id.startB)).setOnClickListener(this);
    ((Button) findViewById(R.id.backupB)).setOnClickListener(this);
    ((Button) findViewById(R.id.resetB)).setOnClickListener(this);
    ((Button) findViewById(R.id.calculatedFB)).setOnClickListener(this);
    ((Button) findViewById(R.id.customFB)).setOnClickListener(this);
    ((Button) findViewById(R.id.allAppListB)).setOnClickListener(this);
    ((Button) findViewById(R.id.CompareBtn)).setOnClickListener(this);
    ((Button) findViewById(R.id.buttonNextMenu)).setOnClickListener(this);

    String status = getONOFF();
if(status.equalsIgnoreCase("ON")) {
    ((Button) findViewById(R.id.startB)).setText("Stop Monitor");
} else {
    ((Button) findViewById(R.id.startB)).setText("Start Monitor");
}

break;
case AllAppListPage:
    prevPage = AllAppListPage;
    setContentView(R.layout.applist);
    ((Button)findViewById(R.id.buttonBackAllApp)).setOnClickListener(this);
    // ((ListView)findViewById(R.id.listViewAllApp)).setOnItemClickListener(this);
    final ArrayList<PInfo> allApps = getPackages();
    ListView listOfAllApp = ((ListView)findViewById(R.id.listViewAllApp));
    String [] lstData = new String[1000];
    for(int i = 0 ;i <allApps.size();i++) {
        Log.e("LIST",allApps.get(i).appname);
        lstData[i] = allApps.get(i).appname;
    }
    listOfAllApp.setAdapter(new ArrayAdapter(this,android.R.layout.simple_list_item_1, lstData));
    listOfAllApp.setOnItemClickListener(new OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
            PackageManager pmi = getPackageManager();
            Intent intent = null;
            //Toast.makeText(getApplicationContext(), allApps.get(arg2).pname, Toast.LENGTH_LONG).show();
            Intent = null;
            pmi .getLaunchIntentForPackage(allApps.get(arg2).pname);
            if (intent != null) {
                // Do something with the intent
            }
        }
    };
}
startActivity(intent);
}
// TODO Auto-generated method stub
});

break;

case CustomFavPage:
    prevPage = CustomFavPage;
    setContentView(R.layout.customfapplist);
    ((ListView)findViewById(R.id.listViewCustomSelected)).setOnItemClickListener(this);
    try
    {
        db =this.openOrCreateDatabase(DatabaseProvoider.DATABASE_NAME, Context.MODE_PRIVATE, null);
        Cursor c = db.rawQuery("SELECT * FROM "+ DatabaseProvoider.TABLE_NAME + " ORDER BY "+DatabaseProvoider.COLUMN_NAME6CustomUsageCount +" DESC ";",null);
        ListView listOfAllCalApp =
        ((ListView)findViewById(R.id.listViewCustomSelected));
        c.moveToFirst();
        String [] lstData1 = new String[c.getCount()];
        Log.e("Count",""+c.getCount());
        for(int i= 0 ;i <c.getCount() ;i++)
        {
            lstData1[i] = c.getString(c.getColumnIndex(DatabaseProvoider.COLUMN_NAME1AppName));//+
c.getInt(c.getColumnIndex(DatabaseProvoider.COLUMN_NAME4UsageCount));
            c.moveToNext();
        }
        listOfAllCalApp.setAdapter(new
        ArrayAdapter(this,android.R.layout.simple_list_item_1, lstData1));
    }
    catch (Exception e)
    {
    }
    db.close();
}
catch (Exception e) {
    // TODO: handle exception
}

//((Button)findViewById(R.id.buttonBackCustom)).setOnClickListener(this);

break;

case CAlculatedFavPage:
    prevPage = CAlculatedFavPage;

    //
    //    final Intent mainIntent = new Intent(Intent.ACTION_MAIN, null);
    //    mainIntent.addCategory(Intent.CATEGORY_LAUNCHER);
    //
    //    final List pkgAppsList = this.getPackageManager().queryIntentActivities(mainIntent, 0);
    //
    //    for(int j=0;j<pkgAppsList.size();j++)
    //    {
    //        Log.e("EEEE",pkgAppsList.get(j).toString());
    //    }
    //
    //
    setContentView(R.layout.calculatedfapplist);
    ((Button)findViewById(R.id.buttonFrmBeg)).setOnClickListener(this);
    ((Button)findViewById(R.id.buttonFrmMonth)).setOnClickListener(this);
    ((Button)findViewById(R.id.buttonfrmWeek)).setOnClickListener(this);
    // ((Button)findViewById(R.id.buttonBackCalculated)).setOnClickListener(this);
    ((ListView)findViewById(R.id.listViewCalculatedApp)).setOnItemClickListener(this);
    try {
        db =this.openOrCreateDatabase(DatabaseProvier.DATABASE_NAME, Context.MODE_PRIVATE, null);
        Cursor c = null;
    }
switch(periodSelected)
{
    case TOTAL_PERIOD:
        c = db.rawQuery("SELECT * FROM "+ DatabaseProvoider.TABLE_NAME + "
ORDER BY "+DatabaseProvoider.COLUMN_NAME4UsageCount +" DESC ;",null );
        break;

    case MONTH_PERIOD:
        c = db.rawQuery("SELECT * FROM "+ DatabaseProvoider.TABLE_NAME + "
ORDER BY "+DatabaseProvoider.COLUMN_NAME9MONTHRATING +" DESC ;",null );
        break;

    case WEEK_PERIOD:
        c = db.rawQuery("SELECT * FROM "+ DatabaseProvoider.TABLE_NAME + "
ORDER BY "+DatabaseProvoider.COLUMN_NAME8WEEKRATING +" DESC ;",null );
        break;
    case LAST_DAY_PERIOD:
        c = db.rawQuery("SELECT * FROM "+ DatabaseProvoider.TABLE_NAME + "
ORDER BY "+DatabaseProvoider.COLUMN_NAME10PER_DAY_RATTING +" DESC ;",null );
        break;
}

    //  c = db.rawQuery("SELECT * FROM "+ DatabaseProvoider.TABLE_NAME + "
ORDER BY "+DatabaseProvoider.COLUMN_NAME4UsageCount +" DESC ;",null );

    ListView listofAllCalApp =
((ListView)findViewByld(R.id.listViewCalculatedApp));

    c.moveToFirst();
    String [] lstData1 = new String[c.getCount()];
    Log.e("Count",""+c.getCount());
    for(int i= 0 ;i <c.getCount() ;i++)
    {
}
lstData1[i] =
c.getString(c.getColumnIndex(DatabaseProvier.COLUMN_NAME1AppName)) ;
+ c.getInt(c.getColumnIndex(DatabaseProvier.COLUMN_NAME4UsageCount));
  c.moveToNext() ;
}

listOfAllCalApp.setAdapter(new
ArrayAdapter(this,android.R.layout.simple_list_item_1, lstData1));
db.close() ;

}
catch (Exception e) {

// TODO: handle exception
}

//
// Intent intent = new Intent(Intent.ACTION_MAIN);
// intent.setComponent(new
ComponentName("com.android.quicksearchbox","com.android.quicksearchbox.SearchA
tivity"));
// startActivity(intent);
//
// Intent LaunchIntent =
getPackageManager().getLaunchIntentForPackage("com.android.quicksearchbox.Search
Activity");
// startActivity( LaunchIntent );

break;

case DescriptionPage:
  setContentView(R.layout.description);
  ((ImageButton)findViewById(R.id.appIconImageButton)).setOnClickListener(this);
  ((Button)findViewById(R.id.buttonSubmitRating)).setOnClickListener(this);
  break;

case NextMenuPage:
  setContentView(R.layout.more_menu);
  ((Button)findViewById(R.id.btnWifiUsage)).setOnClickListener(this);
  break;
((Button)findViewById(R.id.btnBatteryUse)).setOnClickListener(this);
break;

case WifiUsagePage:
setContentView(R.layout.wifi_usage);

((Button)findViewById(R.id.buttonRefreshNetworkStistics)).setOnClickListener(this);
refreshNetworkStatistics();
break;

default:
}

@Override
public void onBackPressed() {
switch(page)
{

case AllAppListPage:
page = MainPage ;
loadPages();
break;
case CustomFavPage:
page = MainPage ;
loadPages();
break;
case DescriptionPage:

page = prevPage;
// page = MainPage ;
loadPages();
break;

case CAIculatedFavPage:
page = MainPage ;
loadPages();
break;
}
case NextMenuPage:
    page = MainPage;
    loadPages();
    break;

case WifiUsagePage:
    page = NextMenuPage;
    loadPages();
    break;

case MainPage:

    Intent myIntent = new Intent(this, UMLogger.class);
    startActivityForResult(myIntent, 0);

    // PackageManager pm = getPackageManager();
    // Intent tintent = null;
    // tintent = pm.getLaunchIntentForPackage("edu.support.Power");
    // if (tintent != null){
    //    startActivity(tintent);
    // }
    break;

default:
    page = MainPage;
    loadPages();

}

public static String packageName = "";

public void setDescriptionPage(String appName){
// String appName = ((ListView)findViewById(R.id.listViewAllApp)).getItemAtPosition(position).toString();

    Log.e("AppName", appName); 
    db = this.openOrCreateDatabase(DatabaseProvider.DATABASE_NAME, Context.MODE_PRIVATE, null);
    Cursor c = db.rawQuery("SELECT * FROM " + DatabaseProvider.TABLE_NAME + " WHERE " + DatabaseProvider.COLUMN_NAME1AppName + " like " + appName + "]", null);
    if(c.getCount() >= 1)
    {
        c.moveToFirst();
        Log.e("AppName", c.getString(c.getColumnIndex(DatabaseProvider.COLUMN_NAME1AppName)));
        ((TextView)findViewById(R.id.textViewAppName)).setText("" +
        c.getString(c.getColumnIndex(DatabaseProvider.COLUMN_NAME1AppName)));
        ((TextView)findViewById(R.id.textViewBatteryUsage)).setText("" +
        c.getInt(c.getColumnIndex(DatabaseProvider.COLUMN_NAME3BatteryUsage)));
        ((TextView)findViewById(R.id.TextViewFavPesentage)).setText("" +
        c.getInt(c.getColumnIndex(DatabaseProvider.COLUMN_NAME6CustomUsageCount)));
        // ((Button)findViewById(R.id.appIconImageButton)).setText("" +
        c.getInt(c.getColumnIndex(DatabaseProvider.COLUMN_NAME1AppIcon)));
        switch(periodSelected)
        {
            case TOTAL_PERIOD:
                ((TextView)findViewById(R.id.textViewCalculatedFavRatting)).setText("" +
                c.getInt(c.getColumnIndex(DatabaseProvider.COLUMN_NAME4UsageCount)));
                break;
            case MONTH_PERIOD:
                ((TextView)findViewById(R.id.textViewCalculatedFavRatting)).setText("" +
                c.getInt(c.getColumnIndex(DatabaseProvider.COLUMN_NAME9MONTHRATING)));
                break;
            case WEEK_PERIOD:
                ((TextView)findViewById(R.id.textViewCalculatedFavRatting)).setText("" +
                c.getInt(c.getColumnIndex(DatabaseProvider.COLUMN_NAME8WEEKRATING)));
                break;
            case LAST_DAY_PERIOD:
((TextView)findViewById(R.id.textViewCalculatedFavRating)).setText("\n" +
c.getInt(c.getColumnIndex(DatabaseProvider.COLUMN_NAME10PER_DAY_RATING)));
break;
}

//((TextView)findViewById(R.id.textViewCalculatedFavRating)).setText("\n" +
c.getInt(c.getColumnIndex(DatabaseProvider.COLUMN_NAME4UsageCount)));

int uid= c.getInt(c.getColumnIndex(DatabaseProvider.COLUMN_NAME7UID));
packageName =
c.getString(c.getColumnIndex(DatabaseProvider.COLUMN_NAME2PackageName));

double tx=TrafficStats.getUidTxBytes(uid)/(1024*1024);
double rx=TrafficStats.getUidRxBytes(uid)/(1024*1024);

((TextView)findViewById(R.id.textViewDataUsage)).setText("\n"+rx+ "MB");

for (ApplicationInfo app :
getPackageManager().getInstalledApplications(0)) {

if(app.packageName.substring(app.packageName.lastIndexOf("." ) +1,
app.packageName.length()).equals(appName)){
Log.e("icon",""+app.icon);

((ImageButton)findViewById(R.id.appIconImageButton)).setBackgroundDrawable(app.loadIcon(getPackageManager()));
}
}

db.close();

try {
// ApplicationInfo app =
this.getPackageManager().getApplicationInfo(packageName, 0);
Drawable icon = this.getPackageManager().getApplicationIcon(packageName);
    // String name = this.getPackageManager().getApplicationLabel(app);
    // Bitmap bitmap = ((BitmapDrawable)icon).getBitmap();
    //
    ((ImageButton)findViewById(R.id.appIconImageButton)).setBackgroundDrawable(icon);

    } catch (Exception e) {
        
    } ////////////////

}

@Override
public void onItemClick(AdapterView<?> parent, View v, int position, long listId) {
    int id = v.getId();
    Log.e("New PAge","" + v);
    //
    // switch(id)
    // {
    // case R.id.listViewAllApp:
    // //
    // String appName ="";
    // if(page==CAlculatedFavPage)
    // {
    //     appName = ((ListView)findViewById(R.id.listViewCalculatedApp)).getItemAtPosition(position).toString();
    // }
    // else if(page == CustomFavPage)
    // {
    //     appName = ((ListView)findViewById(R.id.listViewCustomSelected)).getItemAtPosition(position).toString();
    // }
    // page = DescriptionPage;
    loadPages();
    setDescriptionPage(appName);
}
public void refreshNetworkStatistics()
{
    checkNetworkStatus();
    if(BatteryMonitor.page == BatteryMonitor.WifiUsagePage){
        long tx = TrafficStats.getTotalTxBytes();
        long rx = TrafficStats.getTotalRxBytes();
        ((TextView)findViewById(R.id.textViewTransmitted)).setText(""+tx+ "bytes");
        ((TextView)findViewById(R.id.textViewRecieved)).setText(""+rx + "bytes");
    }
}

public void checkNetworkStatus()
{
    final ConnectivityManager connMgr = (ConnectivityManager) this.getSystemService(Context.CONNECTIVITY_SERVICE);
    final android.net.NetworkInfo wifi = connMgr.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
    final android.net.NetworkInfo mobile = connMgr.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);
    if( wifi.isAvailable() ){
        ((TextView)findViewById(R.id.textViewConnType)).setText("WI-FI");
        // Toast.makeText(this, "Wifi" , Toast.LENGTH_SHORT).show();
    }
    else if( mobile.isAvailable() ){
        ((TextView)findViewById(R.id.textViewConnType)).setText("Mobile Packet Data");
        //Toast.makeText(this, "Mobile Packet Data ", Toast.LENGTH_SHORT).show();
    }
    else {
        ((TextView)findViewById(R.id.textViewConnType)).setText("No Network");
        Toast.makeText(this, "No Network ", Toast.LENGTH_SHORT).show();
    }
public void analyseUSage(){
}
public void dataUsage(){
    Log.e("trafic usage",""+TrafficStats.getTotalRxBytes());
}
}

class PInfo {
    String appname = "";
    String pname = "";
    String versionName = "";
    int versionCode = 0;
    Drawable icon;
    void prettyPrint() {
        Log.v("App Info",appname + "\t" + pname + "\t" + versionName + "\t" + versionCode);
    }
}

///:com.android.settings.SubSettings

class MyAsynchronous extends Thread {
    static int lastBatteryLevel = 0;
    public void resetWeekUsage(){
        BatteryMonitor.db = BatteryMonitor.AppContext.openOrCreateDatabase(DatabaseProvider.DATABASE_NAME, BatteryMonitor.AppContext.MODE_PRIVATE, null);
        Cursor c = BatteryMonitor.db.rawQuery("SELECT * FROM "+ DatabaseProvider.TABLE_NAME_DATE +";",null);
        c.moveToFirst();
        int dbWeek = c.getInt(c.getColumnIndex(DatabaseProvider.DATE_COLUMN_NAME1START_WEEK_DATE)) ;
        SimpleDateFormat dateFormat = new SimpleDateFormat("dd");
        Calendar cal = Calendar.getInstance();
        System.out.println("time => " + dateFormat.format(cal.getTime()));
        int currntWeek = Integer.parseInt(dateFormat.format(cal.getTime())) / 7 ;
        if(dbWeek != currntWeek){
            //
        }
    }
}
BatteryMonitor.db.execSQL("UPDATE "+ DatabaseProvider.TABLE_NAME_DATE +" SET "+ DatabaseProvider.DATE_COLUMN_NAME1START_WEEK_DATE +" ="+ currntWeek +";");

BatteryMonitor.db.execSQL("UPDATE "+ DatabaseProvider.TABLE_NAME +" SET "+ DatabaseProvider.COLUMN_NAME8WEEKRATING +" ="+ 0 +";");
dbWeek = currntWeek;
}
BatteryMonitor.db.close();

// SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
// Calendar cal = Calendar.getInstance();
// System.out.println("time => " + dateFormat.format(cal.getTime()));
//
// ((TextView)findViewById(R.id.textViewCalculated3)).setText("t:"+dateFormat.format(cal.getTime()));

Calendar currentDate = Calendar.getInstance();
SimpleDateFormat sdf = new SimpleDateFormat("MM/dd/yyyy");

String dateNow = sdf.format(currentDate.getTime());
Date today = new Date(dateNow);
Date finalDay = null;
finalDay = (Date) sdf.parse("MM/dd/yyyy");
int numberOfDays=(int)((finalDay.getTime()-today.getTime())/3600*24*1000);
}

public void resetMonthUsage(){
BatteryMonitor.db = BatteryMonitor.AppContext.openOrCreateDatabase(DatabaseProvier.TABLE_NAME, BatteryMonitor.AppContext.MODE_PRIVATE, null);
Cursor c = BatteryMonitor.db.rawQuery("SELECT * FROM "+ DatabaseProvier.TABLE_NAME_DATE +";",null);
c.moveToFirst();
int dbMonth = c.getInt(c.getColumnIndex(DatabaseProvier.DATE_COLUMN_NAME2START_MONTH_DATE)) ;
Log.e("MONTH",""+dbMonth);
SimpleDateFormat dateFormat = new SimpleDateFormat("MM");
Calendar cal = Calendar.getInstance();
System.out.println("time => " + dateFormat.format(cal.getTime()));
int currntMonth = Integer.parseInt(dateFormat.format(cal.getTime())) ;
Log.e("MONTH",""+currntMonth);
if(dbMonth != currntMonth ){
    BatteryMonitor.db.execSQL( "UPDATE " + DatabaseProvoider.TABLE_NAME_DATE 
                        +" SET "+ DatabaseProvoider.DATE_COLUMN_NAME2START_MONTH_DATE 
                        +"="+ currntMonth + " ;");
    BatteryMonitor.db.execSQL( "UPDATE " + DatabaseProvoider.TABLE_NAME +" SET 
                        + DatabaseProvoider.COLUMN_NAME9MONTHRATING +"="+ 0 +";" );
    dbMonth = currntMonth ;
}
BatteryMonitor.db.close() ;
}
public void resetPerDayUsage(){
    BatteryMonitor.db =
    BatteryMonitor.AppContext.openOrCreateDatabase(DatabaseProvoider.DATABASE_N
AME, BatteryMonitor.AppContext.MODE_PRIVATE , null );
    Cursor c = BatteryMonitor.db.rawQuery("SELECT * FROM "+
    DatabaseProvoider.TABLE_NAME_DATE +";",null );
    c.moveToFirst();
    int dbDay =
    c.getInt(c.getColumnIndex(DatabaseProvoider.DATE_COLUMN_NAME3START_DA
Y)) ;

    SimpleDateFormat dateFormat = new SimpleDateFormat( 
    "dd");
    Calendar cal = Calendar.getInstance();
    System.out.println("time => " + dateFormat.format(cal.getTime()));
    int currntDay = Integer.parseInt(dateFormat.format(cal.getTime())) ;
    Log.e("DAY",""+currntDay);
    Log.e("DB DAY",""+dbDay);
    if(dbDay != currntDay ){
        BatteryMonitor.db.execSQL( "UPDATE " + DatabaseProvoider.TABLE_NAME_DATE 
                        +" SET "+ DatabaseProvoider.DATE_COLUMN_NAME3START_DAY 
                        +"="+ currntDay + " ;");
        BatteryMonitor.db.execSQL( "UPDATE " + DatabaseProvoider.TABLE_NAME +" SET 
                        + DatabaseProvoider.COLUMN_NAME10PER_DAY_RATTING +"="+ 0 +";" );
        dbDay = currntDay ;
    }
    BatteryMonitor.db.close();
public void resetUsageCount()
{
    BatteryMonitor.db =
    BatteryMonitor.AppContext.openOrCreateDatabase(DatabaseProvider.DATABASE_NAME, BatteryMonitor.AppContext.MODE_PRIVATE, null);
    
    int usageCount = 0;
    Cursor c = BatteryMonitor.db.rawQuery("SELECT * FROM " +
    DatabaseProvider.TABLE_NAME + " ORDER BY
    " + DatabaseProvider.COLUMN_NAME4UsageCount + " DESC ;", null);
    try
    {
        c.moveToFirst();
        for(int i = 0; i < c.getCount(); i++)
        {
            usageCount =
            c.getInt(c.getColumnIndex(DatabaseProvider.COLUMN_NAME4UsageCount)) - 90;
            
            if(usageCount <= 90)
            {
                usageCount = 0;
            }
            
            BatteryMonitor.db.execSQL("UPDATE " + DatabaseProvider.TABLE_NAME +
            " SET " + DatabaseProvider.COLUMN_NAME4UsageCount + "+ usageCount +" +
            DatabaseProvider.COLUMN_NAME2PackageName + " LIKE '' +
            c.getColumnIndex(DatabaseProvider.COLUMN_NAME2PackageName) + '';''");
            
        } 
        c.moveToNext();
    }catch (Exception e) {
        // TODO: handle exception
    }
} 

public int getBatteryLevel(Intent intent) {
    int rawlevel = intent.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
    int scale = intent.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
    int level = -1;
    
    if (rawlevel >= 0 && scale > 0) {
        level = (rawlevel * 100) / scale;
    }
public void run(){
    while(true){
        try {
            resetMonthUsage();
            resetWeekUsage();
            resetPerDayUsage();
            this.sleep(10000);
            Log.e("**************trafic usage*", "+TrafficStats.getTotalRxBytes());

            long appDataUsage = TrafficStats.getTotalRxBytes() - BatteryMonitor.lastDataUsage;
            int rawlevel = BatteryMonitor.AppContext.getIntent().getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
            int scale = BatteryMonitor.AppContext.getIntent().getIntExtra(BatteryManager.EXTRA_SCALE, -1);
            int level = -1;
            if (rawlevel >= 0 && scale > 0) {
                level = (rawlevel * 100) / scale;
            }
            Log.e("BAttery Level",""+level);

            int batteryUsage = 0;
            if(lastBatteryLevel!=0)
            {
                if((lastBatteryLevel - level) > 0){
                    batteryUsage = lastBatteryLevel - level ;
                }
            }
            lastBatteryLevel= level ;

            ActivityManager am = (ActivityManager) BatteryMonitor.AppContext.getSystemService(BatteryMonitor.AppContext.ACTIVITY_SERVICE);
            // get the info from the currently running task
            List< ActivityManager.RunningTaskInfo > taskInfo = am.getRunningTasks(1);
            Log.d("topActivity", "CURRENT Activity ::"+
                taskInfo.get(0).topActivity.getClassName());
ComponentName componentInfo = taskInfo.get(0).topActivity;
componentInfo.getPackageName();

BatteryMonitor.db = BatteryMonitor.AppContext.openOrCreateDatabase(DatabaseProvider.DATABASE_NAME, BatteryMonitor.AppContext.MODE_PRIVATE, null);

int usageCount = 0;
int weekUsage = 0;
int monthUsage = 0;
int dayUsage = 0;

Cursor c = BatteryMonitor.db.rawQuery("SELECT * FROM " + DatabaseProvider.TABLE_NAME + " WHERE " + DatabaseProvider.COLUMN_NAME2PackageName + " LIKE " + componentInfo.getPackageName() + " ;",null);
try{
    c.moveToFirst();
    usageCount = c.getInt(c.getColumnIndex(DatabaseProvider.COLUMN_NAME4UsageCount)) + 1;
    weekUsage = c.getInt(c.getColumnIndex(DatabaseProvider.COLUMN_NAME8WEEKRATING)) + 1;
    monthUsage = c.getInt(c.getColumnIndex(DatabaseProvider.COLUMN_NAME9MONTHRATING)) + 1;
    dayUsage = c.getInt(c.getColumnIndex(DatabaseProvider.COLUMN_NAME10PER_DAY_RATING)) + 1;
    boolean isUsageCountLarge = usageCount >= 100;
    Log.e("USageCount","" + usageCount + "" + componentInfo.getPackageName());
    BatteryMonitor.db.execSQL("UPDATE " + DatabaseProvider.TABLE_NAME + " SET " + DatabaseProvider.COLUMN_NAME4UsageCount + " = " + usageCount + " , " + DatabaseProvider.COLUMN_NAME8WEEKRATING + " = " + weekUsage + " , " + DatabaseProvider.COLUMN_NAME9MONTHRATING + " = " + monthUsage + " , " + DatabaseProvider.COLUMN_NAME10PER_DAY_RATING + " = " + dayUsage + " WHERE " + DatabaseProvider.COLUMN_NAME2PackageName + " = " + componentInfo.getPackageName() + " ; " );
    BatteryMonitor.db.close();
    if(isUsageCountLarge){
        
    }
}
resetUsageCount() ;
}

} catch (Exception e) {
    // TODO: handle exception
}
}

} catch (Exception e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

} } }
Bibliography

[1] Android Components
http://developer.android.com/reference/android/app/Activity.html#ActivityLifecycle
http://www.vogella.com/tutorials/AndroidBroadcastReceiver/article.html
http://developer.android.com/reference/android/content/Intent.html
http://www.vogella.com/tutorials/AndroidServices/article.html
http://www.tutorialspoint.com/android/android_shared_preferences.htm
http://www.vogella.com/tutorials/Android/article.html


http://en.wikipedia.org/wiki/Android_(operating_system)


[6] BreakFree


[8] Learning Algorithm Approach

http://en.wikipedia.org/wiki/Supervised_learning

http://en.wikipedia.org/wiki/Learning_to_rank