Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

MASTER THESIS

TITLE: Application for monitoring mobiles batteries

MASTER DEGREE: Master in Science in Telecommunication Engineering & Management

AUTHOR: Sergio Ortega Parrado

DIRECTOR: Ángel Cuadras

DATE: May, 11th 2015

Title: Application for monitoring mobiles batteries

Author: Sergio Ortega Parrado

Director: Àngel Cuadras

Date: May, 11th 2015

Overview

The master thesis focuses on the development of a monitoring system of mobile devices' batteries. The main achievement of the project is the implementation of a functional version of the monitoring system which retrieves information of the mobile device, synchronizes it with the server, processes the information and finally stores it on a server.

The monitoring system consists mainly of two parts, which are the mobile application and the server. The mobile application (hereafter, app) has been developed in Java by using the Android programming tools and SQLite database. Its main functions are the monitoring and the storage of the changes in the mobile device battery. The app retrieves battery's parameters, such as, voltage, current, temperature, battery level and status, among others. It periodically sends this information using a web service to the server, which is the other main component of the system. The server consists of an Apache Web server developed with PHP. Its main functions are the management of the information sent by the app and the storage of the processed data into MySQL database. Additionally, the server publishes a web platform or dashboard, where users can search for specific information that has been previously saved. The monitoring system has been developed based on the market-leading Android mobile platform. However, the designed architecture can easily be adapted to other mobile platforms.

The other goal of the project is obtaining the battery's generated entropy in real time. This parameter is a thermodynamics function of state that describes irreversible processes. The entropy calculation is based on the open circuit voltage, which cannot be directly measured; hence, estimation methods are required.

The application has been installed, tested and evaluate in more than 20 devices. Voltages, currents, temperatures and entropies have been saved for more than two months. Although it is beyond the scope of the project, information collected via the application along with the estimate of entropy can lead to additional and future research.

Título: Aplicación para la monitorización de baterías en móviles

Autor: Sergio Ortega Parrado

Director: Ángel Cuadras

Fecha: 11 de Mayo del 2015

Resumen

Este proyecto de final de master presenta un sistema desarrollado para la monitorización de baterías en dispositivos móviles.

El objetivo principal es tener una versión funcional del sistema capaz de recuperar la información, sincronizarla, procesarla y almacenarla en un servidor. Aunque el desarrollo del sistema se basa en la plataforma móvil Android, líder en el mercado, se puede transportar fácilmente la arquitectura diseñada a otras plataformas.

El sistema consta de dos partes principales. La primera de ellas consiste en una aplicación móvil desarrollada en Java, utilizando las herramientas de programación de Android y una base de datos SQLite. Esta aplicación monitoriza y almacena todos los cambios producidos en la batería del móvil de manera autónoma. Recupera parámetros proporcionados por Android como el voltaje, la intensidad, la temperatura, el nivel de batería y su estado, entre otros. La aplicación envía periódicamente esta información a un servidor que proporciona un servicio web. La otra parte del sistema, está formada por un servidor web Apache desarrollado principalmente con PHP que gestiona la información enviada por la aplicación y la guarda en una base de datos MySQL. Adicionalmente, este servidor publica una plataforma web de consulta que muestra la información que el usuario seleccione y lo muestra en forma de gráficos.

Otros de los objetivos era el procesado de dicha información para poder obtener la generación de entropía de la batería en tiempo real. Este parámetro es un indicador de los efectos termodinámicos irreversibles en estos elementos. El cálculo de entropía se basa en el voltaje en circuito abierto y por limitaciones para obtener este valor se ha debido realizar estimaciones. Esta información conjuntamente con la recuperada podrá ser utilizada para futuras líneas de investigación.

Durante el desarrollo del proyecto, se ha instalado la aplicación en más de 20 dispositivos, que desde hace más de 2 meses se ha estado almacenando sus voltajes, intensidades, temperaturas y entropías.

INDEX

INTRODUCTION

Batteries are part of our everyday life. In this actual age of smartphones, tablets, wearable technology and electric vehicles, battery lifetime has become one of the top usability concerns.

Since battery life directly impacts the extent and duration of mobility, one of the key considerations in the design of a mobile embedded system should be to maximize the energy delivered by the battery, and hence the battery lifetime.

While many endeavors have been devoted to improving battery lifetime, they have fallen short in understanding how users interact with batteries. The main aim of this master thesis is to develop a system for monitoring mobiles batteries. Through this system, real time information about user's mobile battery is collected and processed. In addition, data are stored with the purpose of analyze historical statistics and perhaps predict future behaviors.

For this reason, another objective proposed is to calculate the entropy generation in the battery component. Entropy is a thermodynamics function of state that describes irreversible processes and can evaluate ageing and degradation of these components. Therefore, observing entropy generation could give information about the battery life. Entropy calculation requires knowledge of the open circuit voltage. To obtain this voltage, battery must be disconnected from the device and that is not possible while the application is running. Consequently, two alternative methods to estimate this property have been used.

Monitoring system consists of two main parts. The first one is an Android application developed in Java, using Android Software Development Kit (SDK) and SQLite database. This application handles each battery's changes and it retrieves battery's parameters, such as, voltage, current, temperature, battery level and status, among others. The application runs without user interaction and periodically synchronizes battery data with the server. As well, application provides a visual interface to display current information and where user can modify some settings. The other part of the system consists of an Apache Web Server developed with PHP. It provides a web service which is used by the app to send the data. Moreover, it processes data and it performs needed calculations and estimations. After that, server stores information into MySQL database. This database supports data from multiple devices.

Additionally, a web platform or dashboard has been developed, where users can search for specific information saved on the server. Results are displayed in visual graphs, and it is possible to export the data in a CSV file. It is important to note that a login form exists. Therefore, users have to authenticate with valid credentials to access to the dashboard.

The present report is divided according to these two parts of the system. It contains six chapters in total. Firstly, mobile battery chapter introduces some mobile batteries concepts, types of technologies, state of charge concept and

entropy definition. In the second chapter, the technology used to develop the system is presented. Third and fourth chapters explain in depth the implementation of the mobile application and the server, respectively. Next chapter displays the obtained results and presents a complete test. Finally, conclusions are exposed in chapter six.

CHAPTER 1.MOBILE BATTERY

1.1 Mobile Battery Technologies

Mobile phones run on a variety of different batteries depending on the phones manufacturer, its size, shape, and features. Principally, there are four different technologies of battery available for mobile phones. Table 1.1 presents main characteristics of these mobile battery technologies.

Table 1.1 Characteristics of mobile battery technologies [1]

	Li-Poly	**Li-Ion**	**NiMH**	**NiCd**
Cycle Life (to 80% of initial capacity)	300-500	500-1000	300-500	1500
Cell voltage (nominal)	3.6V	3.6V	1.25V	1.25V
Memory effects	No	No	Yes	Yes
Maintenance requirement	Not req.	Not req.	60-90 days	30-60 days
Gravimetric Energy Density (Wh/kg)	100-130	110-160	60-120	45-80
Operating Temperature	0 to 60 ºC	-20 to 60ºC	-20 to 60ºC	-40 to 60ºC
Self-discharge / Month	~ 10 %	10%	30%	20%
Load Current (peak)	>2C	>2C	5C	20C
Fast Charge Time	2-4h	2-4h	2-4h	1h

1.1.1 Lithium Polymer (Li-Poly)

This is one of the more modern types of mobile phone batteries. It is extremely light and is very strong and resistant. The battery is also very safe as due to the casing surrounding it there is no risk of fire or explosion even if the battery is punctured. A lithium polymer battery has no memory effects which means it can be recharged to full capacity regardless of how much charge is still held in the battery. A lithium polymer battery also is able to retain its charge for a long period of time.

1.1.2 Lithium Ion Battery (Li-Ion)

This is the current and most popular technology for mobile phone batteries. A lithium-ion phone battery is a lightweight battery, which, because of its high energy density is able to operate at a higher voltage than many other types of rechargeable phone battery. A lithium-ion battery is usually 3.6 volts while some

other rechargeable phone batteries may be as low as 1.2 volts. This means Lithium-ion batteries will normally only be suitable to use in newer models of mobile. In the same way that Li-Poly battery, this battery does not suffer from the memory effect and can be charged at any time.

1.1.3 Nickel Metal Hydride Battery (NiMH)

The main advantage of using a nickel metal hydride battery is that the fact that it is possible to recharge the battery at great speed. In some cases it may be possible to fully recharge the battery in an hour. This is slightly outweighed by the fact that nickel metal hydride batteries have a fairly poor power retention rate. This means that although the battery can be quickly charged, recharging it again may have to take place more frequently than with other types of phone battery. NiMH and NiCd batteries are prone to the memory effect, thus, good practice is to discharge these batteries entirely before recharging them, and to always recharge them fully.

1.1.4 Nickel Cadmium Battery (NiCd)

This was the original type of rechargeable phone battery and it has largely been superseded by the other types of battery, they are seldom used nowadays. Cadmium is a toxic substance and its presence in the batteries led to general concern over their usage and their disposal because of the potential environmental impact. The use of nickel cadmium batteries is now restricted in the EU and it would be very unusual to find nickel cadmium phone batteries available to buy.

1.2 Fuel Gauge and State of Charge

Battery level estimation in mobile phones is usually performed by a “fuel gauge” chip. The basic function of fuel gauge is to monitor the voltage, charge/discharge current and battery temperature, and to estimate the battery’s SoC (State of Charge) and Full Charge Capacity (FCC) of battery.

SoC is an important parameter, which reflects the battery performance, so accurate estimation of SoC cannot only protect battery, prevent overcharge or discharge, and improve the battery life, but also let the application make rationally control strategies to achieve the purpose of saving energy.

Different phone models use different chips. And each chip provides different information about battery. For instance, some Samsung models uses Maxim MAX17043 chip (Fig. 1.1), which, according to its datasheet [2], it has only a battery voltage sensor. As said by the manufacturer's description, this chip uses a sophisticated Li+ battery-modeling scheme, called ModelGauge to track the

battery's relative state-of-charge (SoC) continuously over a widely varying charge/discharge profile.

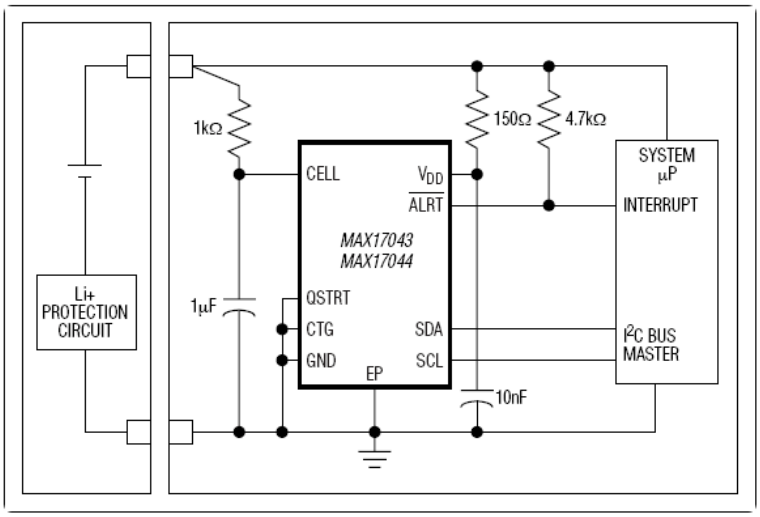


Fig. 1.1 Simplified operating circuit for MAX17043 and MAX17044

Other fuel gauge chips like MAX17047 [3] provides precision measurements of current, voltage, and temperature. This chip incorporates the Maxim ModelGauge m3 algorithm (Fig. 1.2) that combines the Open Circuit Voltage state estimation with the coulomb counter, along with temperature compensation to provide best accuracy. Please refer to annex 1 for additional fuel gauge examples.

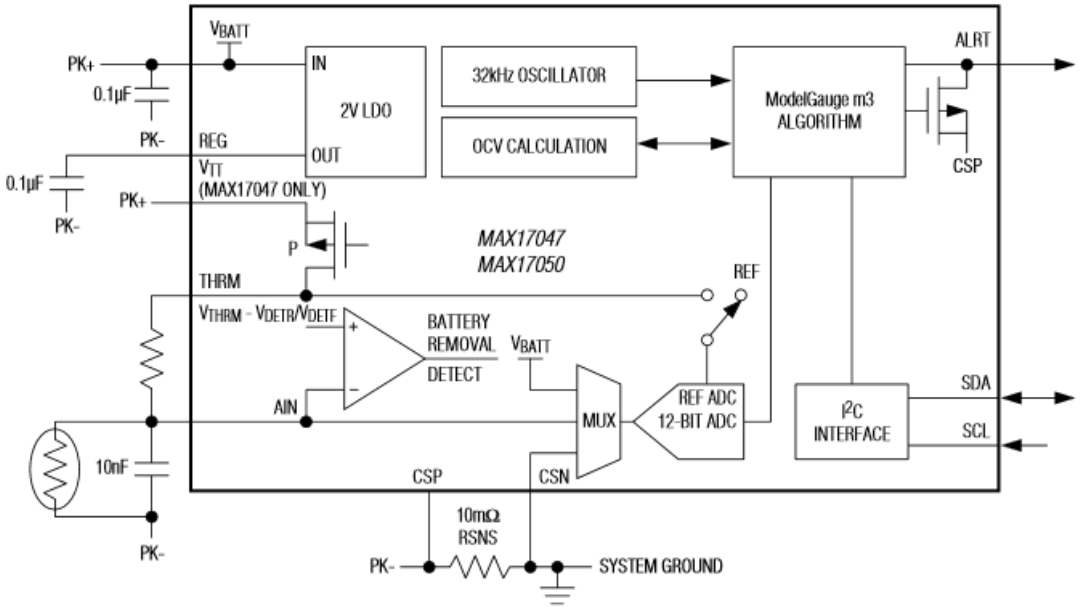


Fig. 1.2 MAX14047 fuel gauge chip

There are two classic methods to do the SOC estimation, OCV and Coulomb Counter.

1.2.1 Open circuit voltage method

It has been shown that there is a linear relationship between the state of charge of the battery and it is open circuit voltage given by [4]

$$V_{oc}(t) = a_{100}SOC(t) + a_0 \quad \textbf{(1.1)}$$

Where $SOC(t)$ is the SOC of the battery at t, a_0 is the battery terminal voltage when SOC is 0%, and a_{100} is obtained from knowing the value of a_0 and $V_{oc}(t)$ at 100% of SOC. The OCV method based on the OCV of batteries is proportional to the SoC when they are disconnected from the loads for a period longer than two hours. However, such a long disconnection time may be too long to be implemented for battery.

In addition, this method can be inaccurate. Cell types have dissimilar chemical compositions that deliver varied voltage profiles. Temperature also plays a role. Higher temperature raises the open-circuit voltage, a lower temperature lowers it.

1.2.2 Coulomb Counter method

This method calculates the state of charge by periodically measuring the current flowing into the battery through a sense resistor and integrating it in time. Accumulation of these current samples gives the total charges that have flown through the battery.

$$SoC = 100\,\frac{Q_a}{Q_{max}} = \frac{\int_{t_0}^{t} I_b(t)dt}{Q_{max}} \quad \textbf{(1.2)}$$

Where Q_a represents the available charge of the battery (expressed in coulomb), Q_{max} represents the maximum charge of the battery and I_b the battery charging current.

Coulomb counters suffers from long-term drift (small ADC offset errors that accumulate indefinitely) and lack of a reference point. To correct these offset accumulation drifts, a large and expensive sense resistor is needed.

1.3 Battery life

The battery life is defined as the number of complete charge and discharge cycles a battery can perform before its nominal capacity falls below 80% of its initial rated capacity. Lithium- and nickel-based batteries deliver between 300 and 500 full discharge/charge cycles before the capacity drops below 80%. Cycling is not the only cause of capacity loss; keeping a battery at elevated temperature also induces stress.

However, counting cycles is not conclusive because a discharge may vary in depth and there are no clearly defined standards of what constitutes a cycle. Rather than establish a discharge cycle as a 100 percent depth of discharge (the DoD is the complement of SoC), manufacturers prefer rating the batteries at 80 percent DoD, meaning that only 80 percent of the available energy is being delivered and 20 percent remains in reserve. A less-than-full discharge increases the battery life, and manufacturers argue that this is closer to a field representation because batteries are seldom fully discharged before recharge [5].

As previously said, there are no standard definitions to define what constitutes a discharge cycle. A smart battery keeping track of cycle count may require a depth of discharge of 70 percent to qualify for a discharge cycle. A battery in a satellite has a typical DoD of 30 or 40 percent before the batteries are recharged during the satellite day.

Notwithstanding, Apple establishes that one charge cycle is when use has used (discharged) an amount that equals 100 per cent of his battery's capacity but not necessarily all from one charge [6]. For instance, user might uses 75 per cent of his battery's capacity one day, then recharge it fully overnight. If he uses 25 per cent the next day, he will have discharged a total of 100 per cent, and the two days will add up to one charge cycle, it is shown in Fig. 1.3 .It could take several days to complete a cycle.

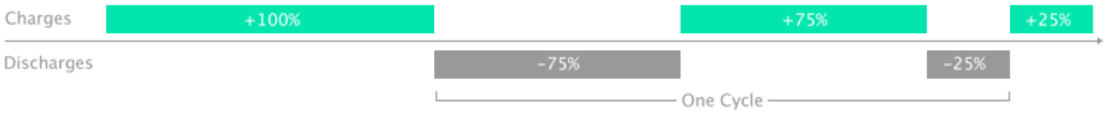


Fig. 1.3 Apple's definition for one complete discharge cycle

Consequently, the depth of discharge (DoD) determine the cycle count. In Fig. 1.4, it can be observe the shorter the discharge (low DoD), the longer the battery will last. If at all possible, avoid full discharges and charge the battery more often between uses. Mobile phone users typically recharge their batteries when the DoD is only about 25 to 30 percent. At this low DoD a lithium-ion battery can be expected to achieve between 5 and 6 times the specified cycle life of the battery which assumes complete discharge every cycle.

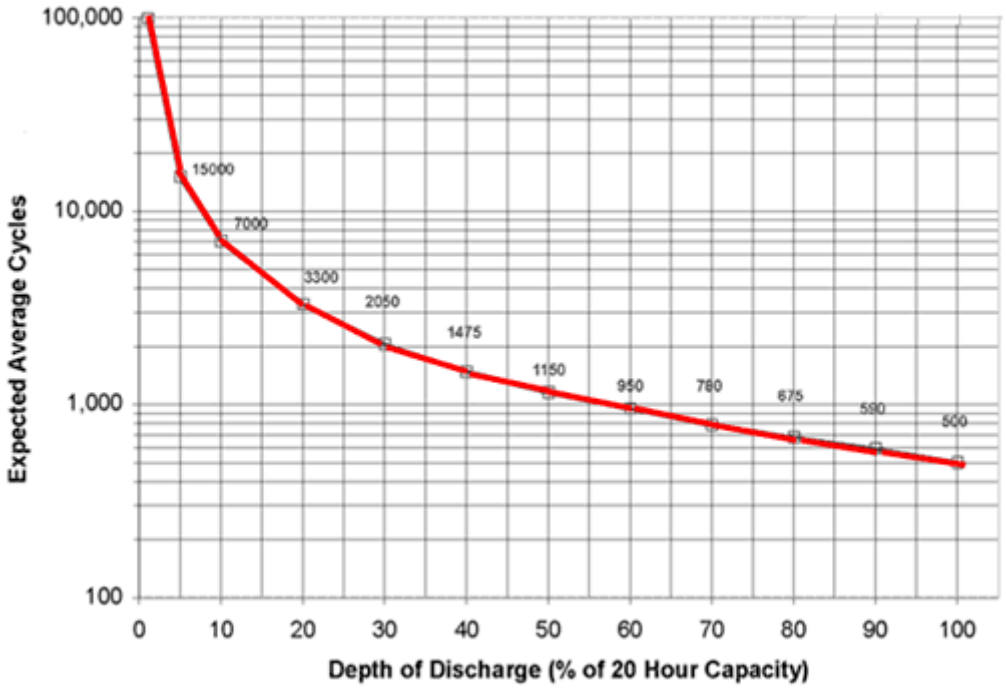


Fig. 1.4 Depth of Discharge vs Cycle Life [7]

Batteries are electrochemical devices which convert chemical energy into electrical energy or vice versa by means of controlled chemical reactions between a set of active chemicals. Unfortunately the desired chemical reactions on which the battery depends are usually accompanied by unwanted, parasitic chemical reactions which consume some of the active chemicals or impede their reactions. Even if the cell's active chemicals remain unaffected over time, cells can fail because unwanted chemical or physical changes to the seals keeping the electrolyte in place.

Chemical reactions internal to the battery are driven either by voltage or temperature. The hotter the battery, the faster chemical reactions will occur. High temperatures can thus provide increased performance, but at the same time the rate of the unwanted chemical reactions will increase resulting in a corresponding loss of battery life.

Most Li-ions are charged to 4.20V/cell and every reduction of 0.10V/cell is said to double cycle life [8]. For instance, if charged a lithium-ion cell to only 4.10V/cell, the life can be prolonged to 600 or 1000 cycles. On the contrary, if cut off voltage is increased, capacity will be reduced as it is noted in the Fig. 1.5.

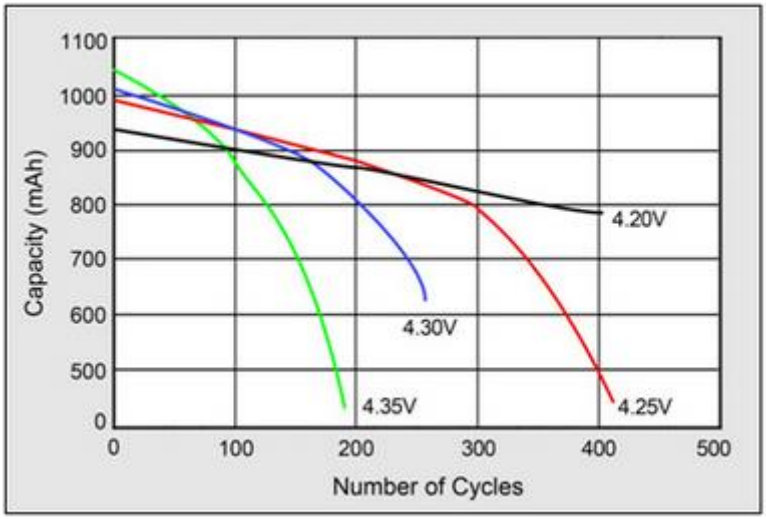


Fig. 1.5 Effects on cycle life at elevated charge voltages.

1.4 Entropy

The word entropy was first used by Rudolf Clausius. It is taken from the Greek word "tropee" which means transformation. Entropy (usual symbol S) is a measure of the number of specific ways in which a thermodynamic system may be arranged, commonly understood as a measure of disorder.

The second law of thermodynamics asserts that dissipative processes of the systems must generate entropy. This entropy is a thermodynamics function of state that describes irreversible processes and can clarify ageing and degradation of electric systems.

The change in entropy (ΔS) of a system was originally defined for a thermodynamically reversible process as

$$\Delta S = \int \frac{\delta Q}{T} \qquad \textbf{(1.3)}$$

Where T is the absolute temperature of the system, dividing an incremental reversible transfer of heat into that system δQ.

CHAPTER 2.APPLIED TECHNOLOGY

2.1 Mobile application

An Android mobile application is developed in order to fulfill the main objective of the master thesis. Android is a mobile operating system (OS) based on the Linux kernel and currently developed by Google. Android is the leading mobile platform worldwide, as can be observe from Fig. 2.1. In addition,

Table 2.1 presents the evolution of the market share and unit shipments for the last two years.

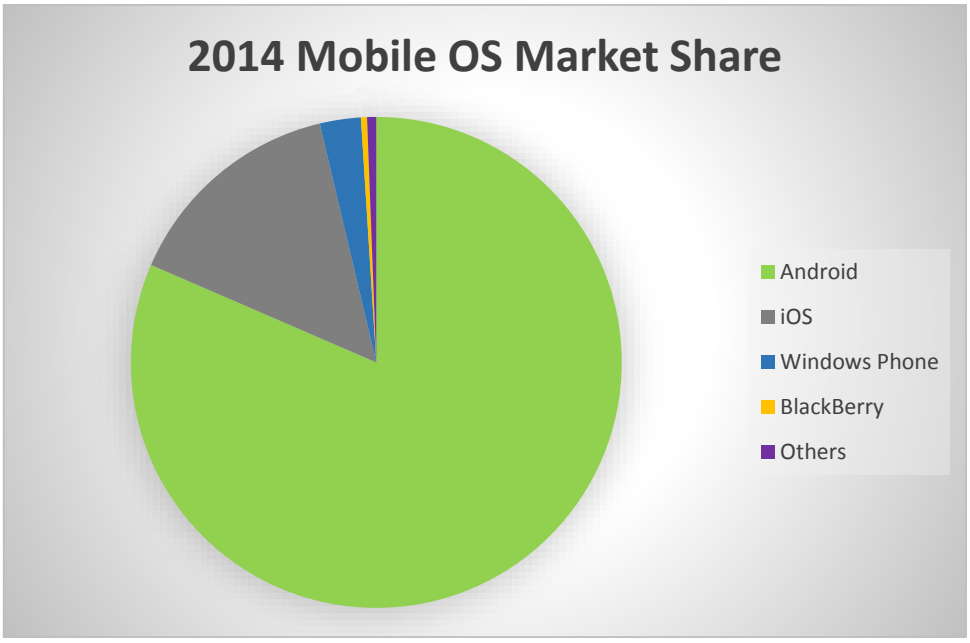


Fig. 2.1 Mobile platform market share in 2014

Table 2.1 Top Four Smartphone Operating Systems, Unit Shipments, Market Share, and Year-Over-Year Growth (Units in Millions) [9]

Operating System	2014 Unit Volumes	2014 Market Share	2013 Unit Volumes	2013 Market Share	Year-Over-Year Change
Android	1,059.3	81.5%	802.2	78.7%	32.0%
iOS	192.7	14.8%	153.4	15.1%	25.6%
Windows Phone	34.9	2.7%	33.5	3.3%	4.2%
BlackBerry	5.8	0.4%	19.2	1.9%	-69.8%

Others	7.7	0.6%	2.3	0.2%	234.8%
Total	1,300.4	100.0%	1,018.7	100.0%	27.7%

The application is developed using Java, and is built with the latest available Android SDK version. Additionally, it takes advantage of SQLite database already installed on Android operating system.

2.1.1 Android SDK

Android applications are developed using the Java language programming and Android provides a software development kit (SDK), which is a set of tools and APIs (*Application Programming Interface)* for developers [10].

The Android SDK separates tools, platforms, and other components into packages that can be downloaded using the SDK Manager [11]. It can be observed in the Fig. 2.2 that the latest API version (API 21) is used to develop this android application.

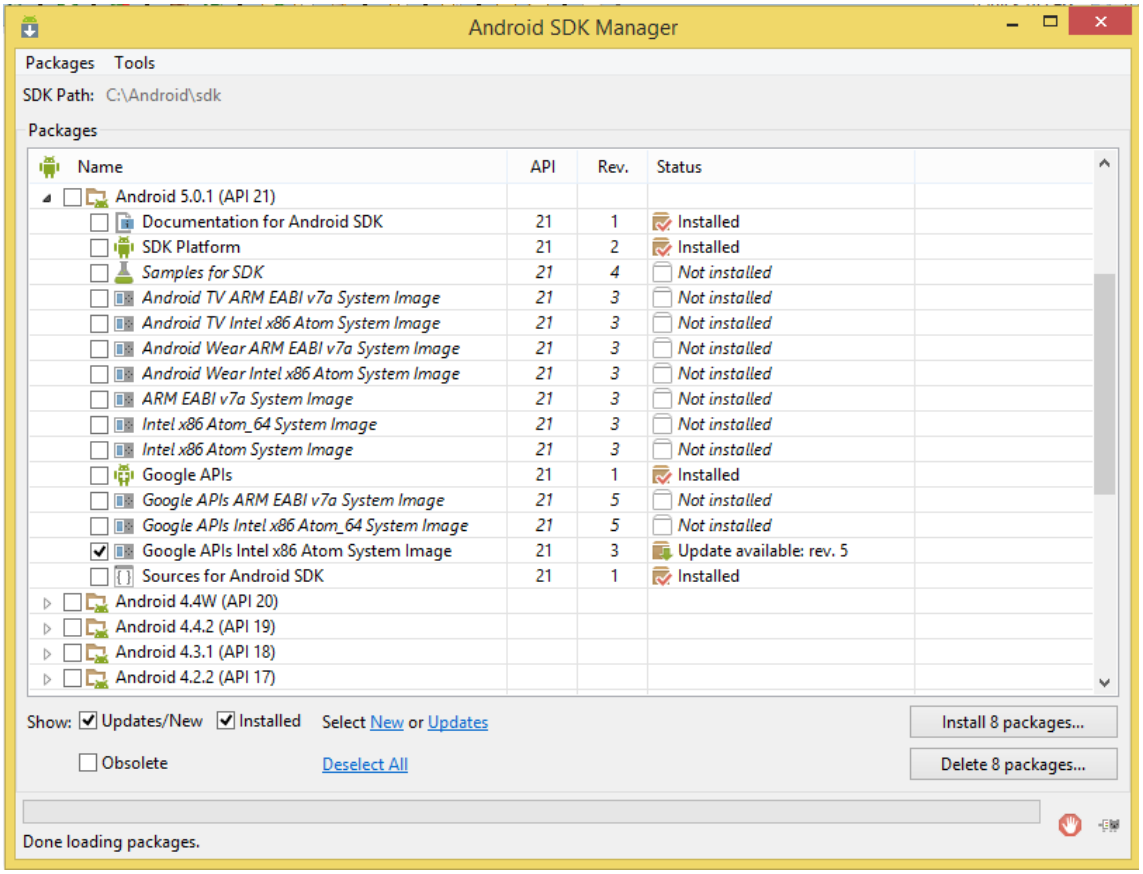


Fig. 2.2 Android SDK Manager

Android application is built using blocks or in other words, application components. There are four different types of app components: activities, services, broadcast receivers and content providers. The first three components are activated by an asynchronous messages called *intent.* Intents bind individual components to each other at runtime and facilitate communication between them.

2.1.1.1 Activities

An activity represents a single screen with which users can interact in order to do something.

For example, an email app might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. Although the activities work together to form a cohesive user experience in the email app, each one is independent of the others.

2.1.1.2 Services

A service is a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface.

Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it.

For example, a service might fetch data over the network without blocking user interaction with an activity.

2.1.1.3 Content providers

A content provider manages a shared set of app data. Data can be stored in the file system, in SQLite database, on the web, or any other persistent storage location that the app can access. Through the content provider, other apps can query or even modify the data.

Content providers are also useful for reading and writing data that is private to the app and not shared.

For example, the Android system provides a content provider that manages the user's contact information.

2.1.1.4 Broadcast receivers

A broadcast receiver is a component that responds to system-wide broadcast announcements. Many broadcasts originate from the system, for instance, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured.

Apps can also initiate broadcasts, for example, to let other apps know that some data has been downloaded to the device and is available for them to use.

2.1.2 SQLite

Android provides several options to save persistent application data. It has been decided to use SQLite databases to store structured private data.

SQLite is an Open Source database. It supports standard relational database features like SQL syntax, transactions and prepared statements. The database requires limited memory at runtime which makes it a good candidate from being embedded into small devices. Android OS comes with SQLite Database already built-in.

2.1.3 Development environment

Eclipse is an IDE (integrated development environment) for Java and other programming languages like C, C++, PHP, and Ruby etc. Development environment provided by Eclipse includes the Eclipse Java development tools (JDT) for Java, Eclipse CDT for C/C++, and Eclipse PDT for PHP, among others. Eclipse is free and open source software. The Eclipse IDE can be extended with additional software components called plug-ins.

Android offers a customized plugin for the Eclipse IDE, called Android Development Tools (ADT). This plugin provides a powerful, integrated environment in which to develop Android applications. It extends the capabilities of Eclipse to let quickly set up new Android projects, build an application User Interface, debug the application, and export signed (or unsigned) application packages (APKs) for distribution.[12]

Eclipse installation must meet the following requirements to be compatible with the latest version of ADT:

- Eclipse 3.7.2 (Indigo) or greater
- Eclipse JDT plugin (included in most Eclipse IDE packages)
- JDK 6 (JRE alone is not sufficient).

Eclipse Luna version 4.4.1 is the IDE selected to develop the application. Following screenshot Fig. 2.3 shows the user interface of the IDE.

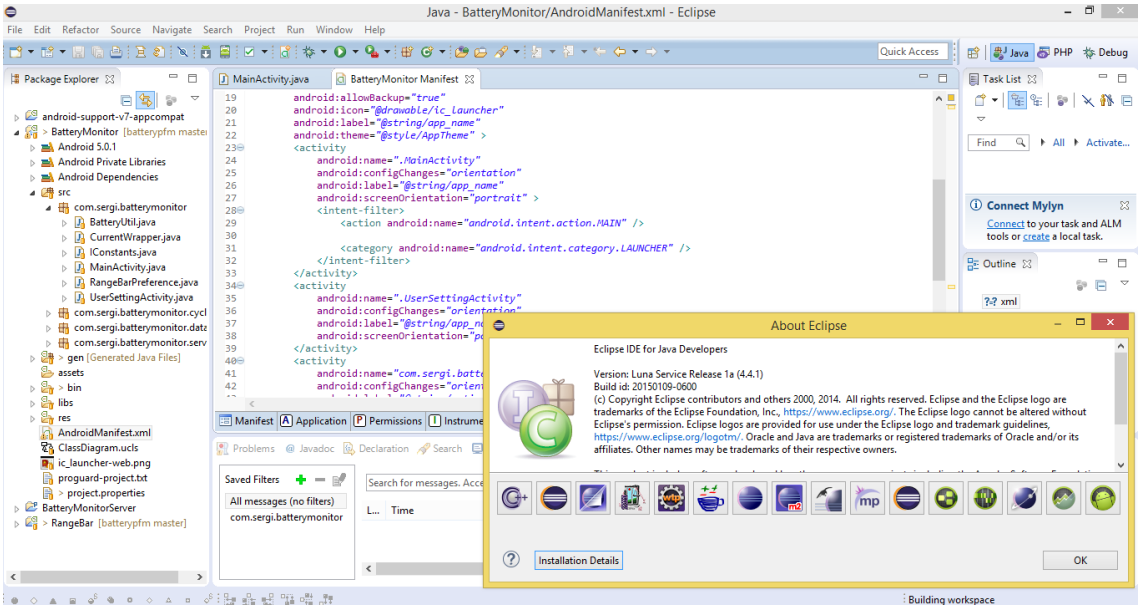


Fig. 2.3 Eclipse Luna with ADT plugin

Moreover, the developed source code is integrated into version control system in order to track and provide control over changes.

2.1.3.1 Version control system.

At the simplest level, developers could simply retain multiple copies of the different versions of the program, and label them appropriately. This simple approach has been used on many large software projects. While this method can work, it is inefficient as many near-identical copies of the program have to be maintained. This requires a lot of self-discipline on the part of developers, and often leads to mistakes. Consequently, systems that automate some or all of the revision control process have been created. There are different software tools for version control, one of the most famous is Git.

Git is a free and open source distributed version control system (DVCS) designed to handle everything from small to very large projects with speed and efficiency. Git has become the most widely adopted version control system for software development. [13]

Development process of the application has been integrated with Git and using Bitbucket as a source code repository web-server. Bitbucket is a hosting site for the distributed version control systems, such as Git, and it provides free plans to private repositories.

There are different ways to connect to this private repository, via web (*https://bitbucket.org/srg-io/batterypfm*), using desktop application called SourceTree (Fig. 2.4) or using an Eclipse Git plug-in.

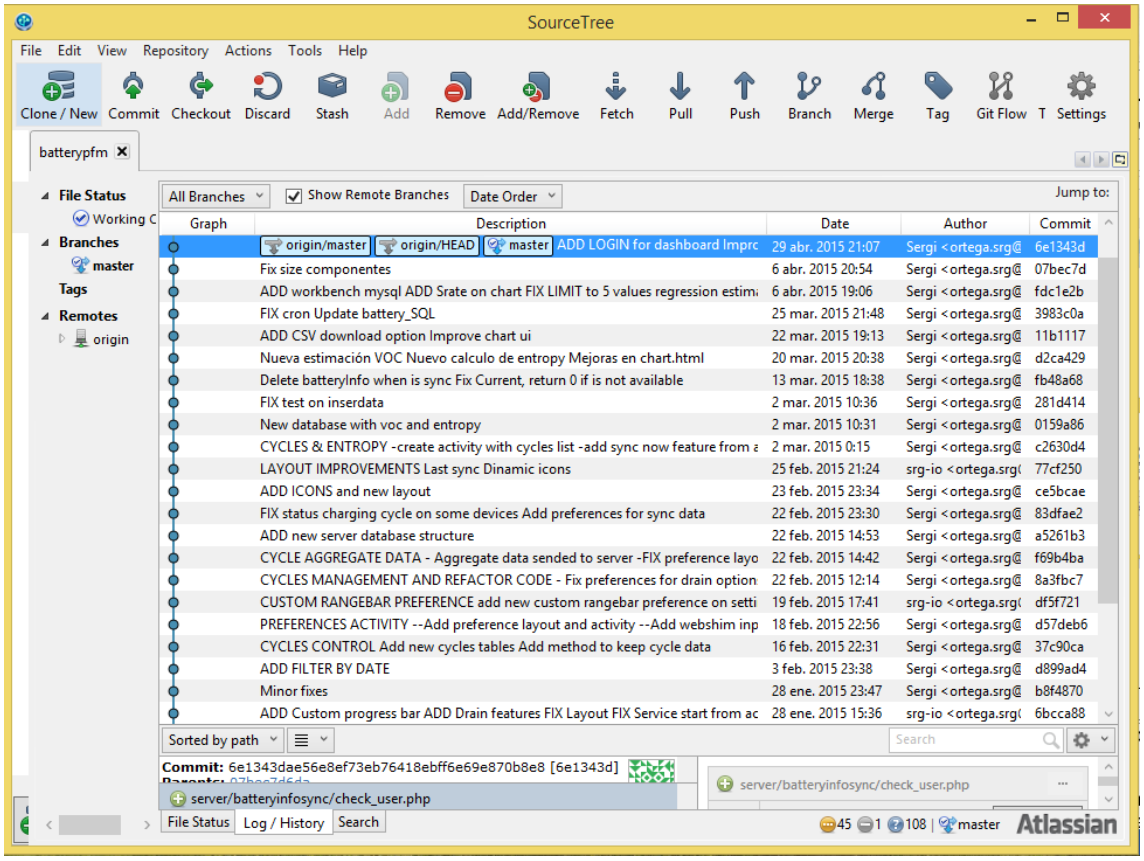


Fig. 2.4 SourceTree desktop application to connect with online repository

2.2 Web Server

The application transfers information collected in the device to a web service provided by a server. Server implementation is built using XAMPP. XAMPP is a free, cross-platform web server, consisting mainly of the Apache, MySQL, PHP and Perl programming languages. X in XAMPP denotes that it can be run on different operating system like Windows, Linux, Mac OS X, etc.

Officially, XAMPP intended it for use only as a development tool, to allow website designers and programmers to test their work on their own computers without any access to the Internet. Fig. 2.5 shows local web for XAMPP control panel. In practice, however, XAMPP is sometimes used to actually serve web pages on Internet. XAMPP also provides support for creating and manipulating databases in MySQL.

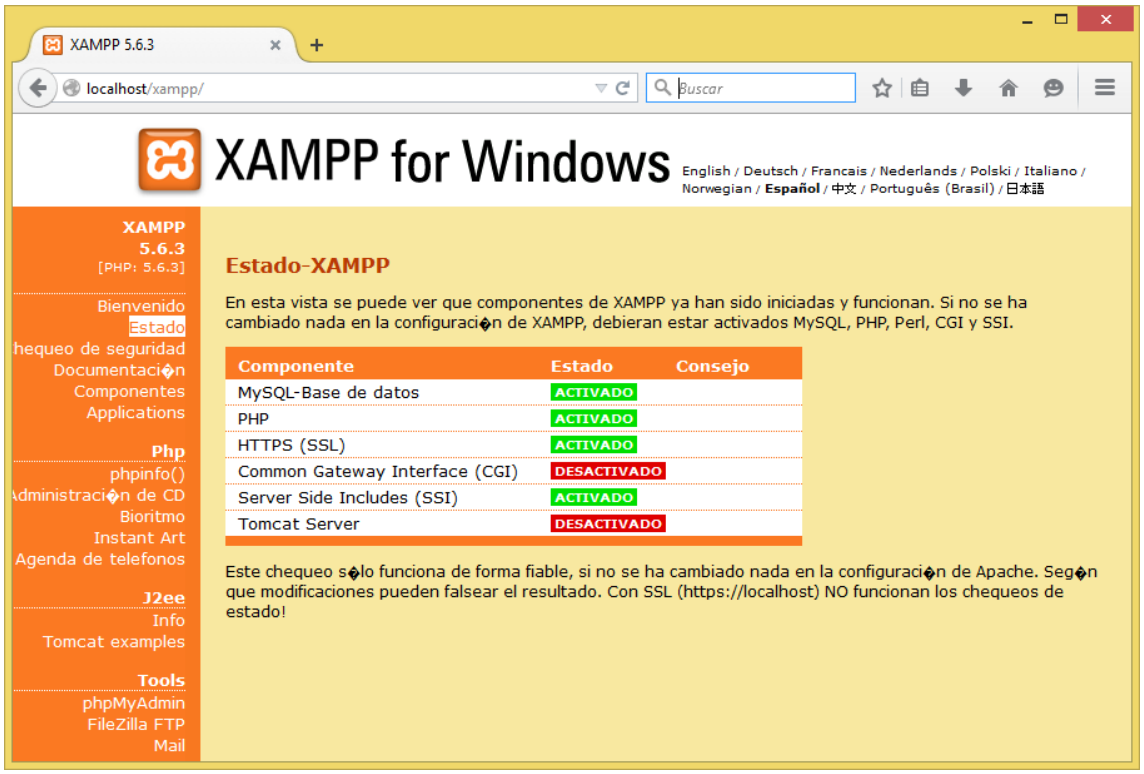


Fig. 2.5 XAMPP status web for Windows

2.2.1 PHP

Application connects to server through web service developed with PHP. The main reason for using this language is because server have installed XAMPP, is the most common server-side programming language used (Fig. 2.6) and is the cheapest option on hosting providers.

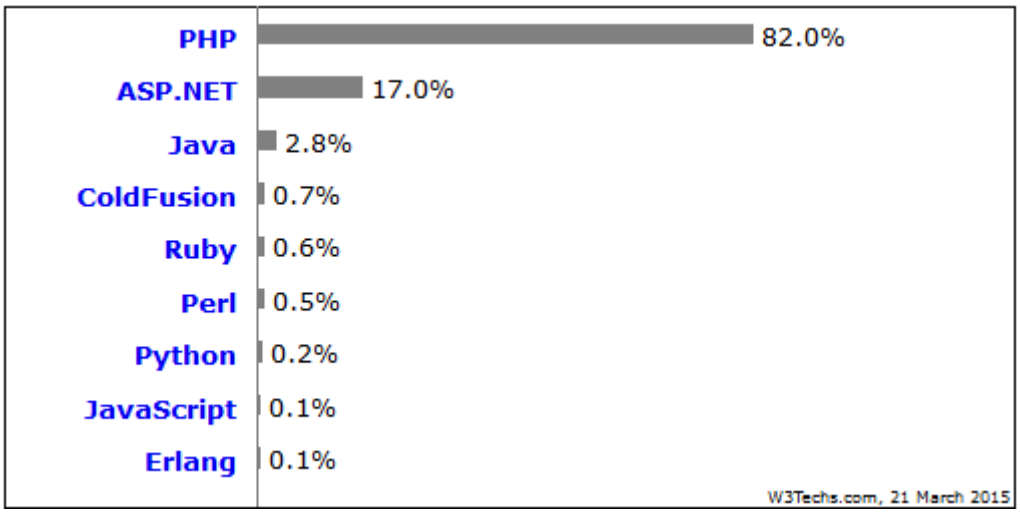


Fig. 2.6 Percentages of various server-side programming languages used actually by websites.

PHP is a general-purpose scripting language that is especially suited to server-side web development, in which case PHP generally runs on a web server. Any PHP code in a requested file is executed by the PHP runtime, usually to create dynamic web page content or dynamic images used on websites. PHP can be deployed on most web servers, many operating systems and platforms, and can be used with many relational database management systems. Most web hosting providers support PHP for use by their clients. It is available free of charge, and the PHP Group provides the complete source code for users to build, customize and extend it to their own use.

2.2.2 MySQL

MySQL is the world's most popular open source database, enabling the cost-effective delivery of reliable, high-performance and scalable Web-based and embedded database applications.

XAMPP installation also includes phpMyAdmin tool. PhpMyAdmin is a tool written in PHP intended to handle the administration of MySQL servers over the Web. Currently it can create and drop databases, tables and views, fields, execute any SQL statement, manage keys on fields, manage privileges, manage triggers and stored procedures and export data into various formats.

2.2.3 Apache

The Apache HTTP Server, colloquially called Apache. This server is an open-source HTTP server for modern operating systems including UNIX, Microsoft Windows, Mac OS/X and Netware. The goal is to provide a secure, efficient and extensible server that provides HTTP services observing the current HTTP standards. Apache has been the most popular web server on the Internet since April 1996.

2.2.4 Dashboard

Web dashboard is based on HTML5 with JQuery 1.10.2 as a JavaScript framework and Bootstrap 3.3.4 as a CSS framework. In addition, uses GoogleChart, a library from Google to create and embed charts on websites.

jQuery is the most popular JavaScript library in use today. jQuery is free, open-source software licensed under the MIT License. Bootstrap is a free and open-source collection of tools for creating websites and web applications. It contains HTML and CSS design templates for typography, forms, buttons, navigation and other interface components. It is one of the most popular frameworks used today.

2.2.5 Development environment

These web program languages do not need a specific IDE. The code can be written only using a text editor application. However, advanced text editor or source editor software is highly recommended.

In this case, Notepad++ editor and Eclipse IDE are used to write PHP, HTML, JavaScript and CSS code. The administration of the MySQL database is performed by PhpMyAdmin tool. Moreover, source code is integrated with the same version control system explained in the section 2.1.3.1.

CHAPTER 3. BATTERY MONITOR APPLICATION

The aim of this chapter is to describe the implementation of the android application to monitor battery evolution using the technology already explained in the previous section. Therefore, architecture is firstly presented. Afterwards, the way in which the application gets battery's information and how it keeps these data is detailed. The following section is about synchronization requests and responses. Finally, user interface and the project structure are explained.

3.1 Architecture

Battery monitor application includes different components based on primary building blocks of Android applications (see 2.1.1) that interact with each other as can be seen in Fig. 3.1.

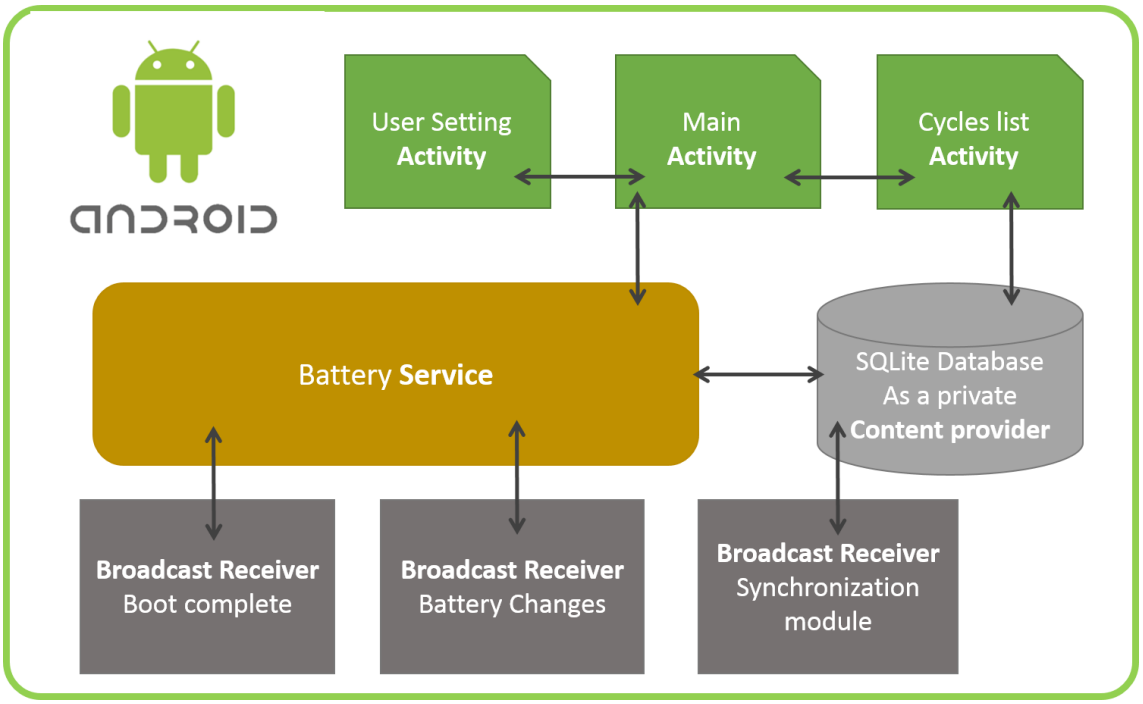


Fig. 3.1 Architecture application based on Android components

Battery monitor Android is composed by three activities (screens): Main, user settings and cycles' list. The main component of the application is the battery service, which is connected to the broadcast receivers to obtain battery changes and to detect when device is started. Additionally, there is a broadcast receiver that is connected to the synchronization module to send data to the server every fifteen minutes.

Application is built to maintain compatibility with older Android versions, minimum compatible version is 2.3.3, that means, application can be run on 99,6% of android devices according to Fig. 3.2.

Version	Codename	API	Distribution
2.2	Froyo	8	0.4%
2.3.3 - 2.3.7	Gingerbread	10	6.9%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	5.9%
4.1.x	Jelly Bean	16	17.3%
4.2.x		17	19.4%
4.3		18	5.9%
4.4	KitKat	19	40.9%
5.0	Lollipop	21	3.3%

Data collected during a 7-day period ending on March 2, 2015.
Any versions with less than 0.1% distribution are not shown.

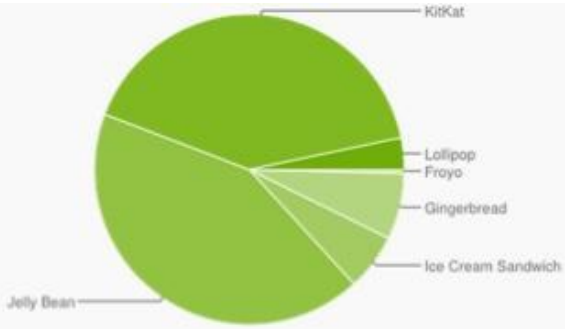


Fig. 3.2 Android device distribution by version system.

3.2 Battery Service component

Battery Service is the core application. This service component is created when device starts, since it is active listening and waiting for “BOOT_COMPLETED” system event. In case that the service was not previously created, it is generated when user interface of the application starts, because the main activity needs to bind to the service to obtain data.

“ACTION_BATTERY_CHANGED” broadcast receiver provides battery changes information. Battery service registers this information and saves it in a table called “battery” on SQLite database.

Additionally, service manages charge and discharge cycles. It compares every battery changed received with previous information, so it detects complete cycles. It keeps cycle data on a different table in the database. By default, the minimum and maximum cycle’s thresholds are 5% and 95% battery level, respectively. However, user can customize these cycle’s thresholds. Service sends cycle data to server after the cycle is completed.

Moreover, service sets an alarm, which perform time-based operations outside the lifetime of the application. In this case, alarm sends data from battery table every 15 minutes to synchronize information with server database. This synchronizing process will be explained in next section. Now, information obtained from battery will be detailed.

3.3 Battery data

Android SDK broadcasts the following battery's data and it is accessible by using Android's *BatteryManager* class [14]:

- EXTRA_STATUS: integer containing the current status constant.

Table 3.1 Extra_status Android battery values

Constant name	Constant value
BATTERY_STATUS_UNKNOWN	1
BATTERY_STATUS_CHARGING	2
BATTERY_STATUS_DISCHARGING	3
BATTERY_STATUS_NOT_CHARGING	4
BATTERY_STATUS_FULL	5

- EXTRA_HEALTH: integer containing the current health constant. It should be note that Android only provides causes of failure. This parameter is not related with battery degradation or state of health (SoH).

Table 3.2 Extra_health Android battery values.

Constant name	Constant value
BATTERY_HEALTH_UNKNOWN	1
BATTERY_HEALTH_GOOD	2
BATTERY_HEALTH_OVERHEAT	3
BATTERY_HEALTH_DEAD	4
BATTERY_HEALTH_OVER_VOLTAGE	5
BATTERY_HEALTH_UNESPECIFIED_FAILURE	6
BATTERY_HEALTH_COLD	7

- EXTRA_PLUGGED: integer indicating whether the device is plugged in to a power source; 0 means it is on battery, other constants are different types of power sources.

Table 3.3 Extra_plugged Android battery values

Constant name	Constant value
BATTERY_PLUGGED_AC	1
BATTERY_PLUGGED_USB	2
BATTERY_PLUGGED_WIRELESS	4

- EXTRA_PRESENT: boolean indicating whether a battery is present.

- EXTRA_LEVEL: integer field containing the current battery level, from 0 to EXTRA_SCALE.

- EXTRA_SCALE: integer containing the maximum battery level.

- EXTRA_VOLTAGE: integer containing the current battery voltage level.

- EXTRA_TEMPERATURE: integer containing the current battery temperature.

- EXTRA_TECNHOLOGY: String describing the technology of the current battery.

If application runs on android system version 5.0 (lollipop) or later, following data are accessible:

- BATTERY_PROPERTY_CAPACITY: Remaining battery capacity as an integer percentage of total capacity (with no fractional part).

- BATTERY_PROPERTY_CHARGE_COUNTER: Battery capacity in microampere-hours, as an integer.

- BATTERY_PROPERTY_CURRENT_AVERAGE: Average battery current in microamperes, as an integer. Positive values indicate net current entering the battery from a charge source, negative values indicate net current discharging from the battery. The time period over which the average is computed may depend on the fuel gauge hardware and its configuration.

- BATTERY_PROPERTY_CURRENT_NOW: Instantaneous battery current in microamperes, as an integer. Positive values indicate net current entering the battery from a charge source, negative values indicate net current discharging from the battery.

- BATTERY_PROPERTY_ENERGY_COUNTER: Battery remaining energy in nanowatt-hours, as a long integer.

However, these last properties can return empty values in Android 5.0 or later, because the information is provided from the fuel gauge chip (section 1.2), and depending on the model chip these information are exposed or not.

Next section will explain how to access to the current information in Android versions prior to 5.0, subject to fuel gauge provides it.

3.3.1 Battery current information

Although fuel gauge chip used in a particular phone can measure the current as explained in previous chapter, accessing that information from Android might be impossible for several reasons.

One reason could be that the chip only uses the current measurement internally and does not expose it to the outside. This is not the case of MAX17047 described above, which has registers holding current measurement values, and these values can be read over the I2C bus. Inter-Integrated Circuit bus allows easy communication between components which reside on the same circuit board.

Another reason could be that kernel driver for the fuel gauge chip does not have code to expose the current measurement information to user applications. That means, Android does not have a standard API to access to this information. This is the main problem in Android versions prior to 5.0. However, alternative technique exists to access to the information.

Android uses Linux core, thus, kernel driver can make the current measurement information available through some sysfs files. Sysfs is a virtual file system provided by the Linux kernel. By using virtual files, sysfs exports information about various kernel subsystems, hardware devices and associated device drivers from the kernel's device model to user space that refers to all code which runs outside the operating system's kernel.

Battery information is located under path:

/sys/class/power_supply

Current information is differently exposed according each kernel manufacturer. For instance, that information could be found in the following file for a LG Nexus device:

/sys/class/power_supply/battery/current_now

Where the current value is on the first line. However, in other devices, the information is located on a different path and the current value is a key-value property. For that reason, a large list with possible locations for current values has been created.

It is hardly difficult to create a complete list of locations since there are near to ten thousand different android devices. So, there will be cases for which current value will not be found.

3.4 Database

Fig. 3.3 shows the entity relationship diagram of SQLite database. It is a simple database and contains three tables. One table called “battery” where data collected from battery are inserted and two relational tables for the data of the cycles captured.

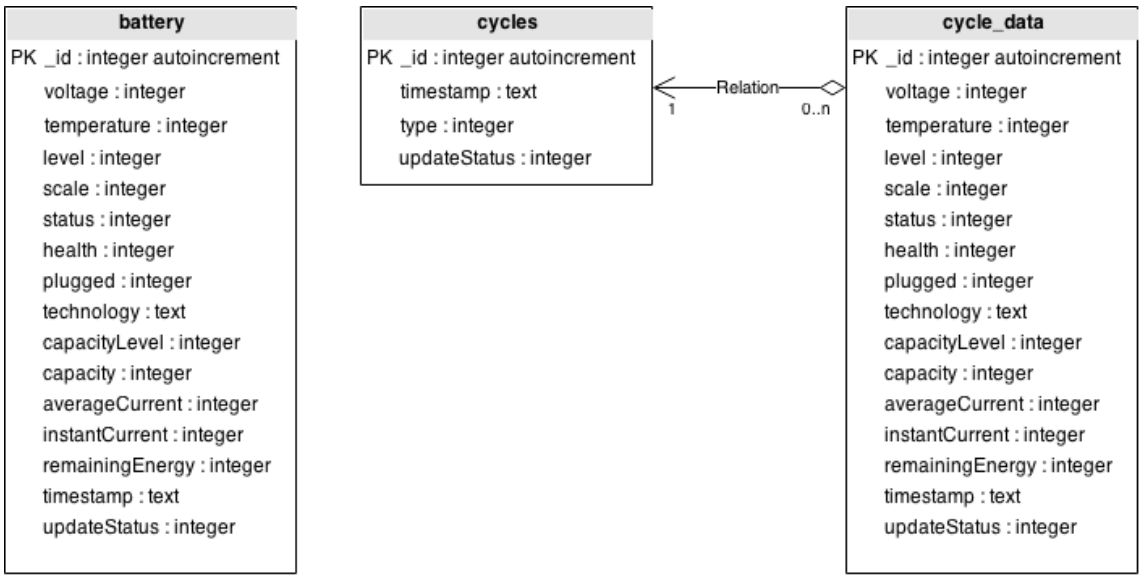


Fig. 3.3 ERD of SQLite mobile database

There is a datasource class that provides the connection to database and contains the needed queries for the application. Main CRUD (Create, Read, Update and Delete) operations are described below:

- Create: insert with correct format new battery registers or cycles.
- Read: obtain unsynchronized data to send to server. Moreover, convert information to JSON format. Data recovered from battery table are limited to 1,000 registers to avoid server’s saturation. Data from cycles are aggregated to obtain average temperature, voltage, current values for each SoC level. So, cycles are composed by 100 registers.
- Update: synchronization module receives status information from server and updates updateStatus field for synchronized registers.
- Delete: by default, remove data successfully synchronized.

3.5 Synchronization module

Synchronization module is responsible to communicate with server and send information stored on device database. It manages http requests and responses and it decides the actions to be made in each situation.

This process is performed by asynchronous post request to following PHP class in the web server which represents a Web Service. That means, web server provides functions that can be accessed by other programs over the web. The communication between app and web service is performed using the JSON standard format.

http://host/batteryinfosync/insertdata.php

The following Table 3.4 shows the parameters sent by the module:

Table 3.4 Parameters sent to server.

Parameter name	Parameter value
datatype	Data battery type: -1: Data send periodically from Battery table. 0: data for charging cycle 1: data for discharging cycle
batteryJSON	All the battery data explained in the previous section.
deviceInfo	Manufacturer and model device
deviceId	A hardware serial number
cycle_id	Cycle identifier stored in cycle table (only for Cycle datatype)
timestamp	Timestamp when cycle was complete (only for Cycle datatype)

This is an example request for a periodically data:

```
datatype=-1&deviceId=047f6af3c998ed60&deviceInfo=LGE Nexus
4&batteryJSON=[{technology":"Li-ion","timestamp":"2015-03-31
10:25:46","_id":7,"instantCurrent":450,"remainingEnergy":0,"averageCurrent
":0,"capacity":0,"capacityLevel":40,"health":2,"level":40,"plugged":2,"pre
sent":false,"scale":100,"status":2,"temperature":256,"updateStatus":0,"vol
tage":3730},
{"technology":"Li-ion","timestamp":"2015-03-31
10:26:56","_id":8,"instantCurrent":369,"remainingEnergy":0,"averageCurrent
":0,"capacity":0,"capacityLevel":40,"health":2,"level":40,"plugged":2,"pre
sent":false,"scale":100,"status":2,"temperature":260,"updateStatus":0,"vol
tage":3746},
{..},]
```

Response from server contains the registers identifier related to data sent with the status information about synchronization, 1 success and 0 fail.

```
[{"type":"-1","id":7,"status":"1"},
{"type":"-1","id":8,"status":"1"},
{"type":"-1","id":9,"status":"1"},
,..,
{"type":"-1","id":47,"status":"1"}]
```

Moreover, this is an example request for charging cycle:

```
datatype=0&deviceId=047f6af3c998ed60&deviceInfo=LGE Nexus
4&cycle_id=32&timestamp:2015-03-31 11:43:00&batteryJSON=[{technology":"Li-
ion","timestamp":"2015-03-31
10:25:46","_id":7,"instantCurrent":450,"remainingEnergy":0,"averageCurrent
":0,"capacity":0,"capacityLevel":40,"health":2,"level":40,"plugged":2,"pre
sent":false,"scale":100,"status":2,"temperature":256,"updateStatus":0,"vol
tage":3730},
{"technology":"Li-ion","timestamp":"2015-03-31
10:26:56","_id":8,"instantCurrent":369,"remainingEnergy":0,"averageCurrent
":0,"capacity":0,"capacityLevel":41,"health":2,"level":41,"plugged":2,"pre
sent":false,"scale":100,"status":2,"temperature":260,"updateStatus":0,"vol
tage":3746},
{..},]
```

In this case, response only contains the identifier related to the cycle.

```
[{"type":"0","id":32,"status":"1"}]
```

By default, data synchronized successfully are deleted on SQLite database, in order to free device memory space. Moreover, a study about data usage in these networks messages has been performed. Each register of battery information is approximately 280 bytes. The frequency of the batteries' changes updates depends on the fuel gauge chip. However, on average there are three updates in one minute. Therefore, 12600 bytes are sent every 15 minutes approximately. That implies a data usage equal to 1.15 Mb in one day.

3.6 User Interface

User can see and manage the information recovered by Battery Service using a visual interface. When user presses application icon from their Android smartphone, main activity starts.

Main activity class binds to Battery Service each time is started to obtain battery information in real time (Fig. 3.4). Activity requests new information each five seconds using a handler when it is in foreground. Handlers are implemented in the main thread of an application and are primarily used to make updates to the user interface in response to messages sent by another thread running within the application's process.

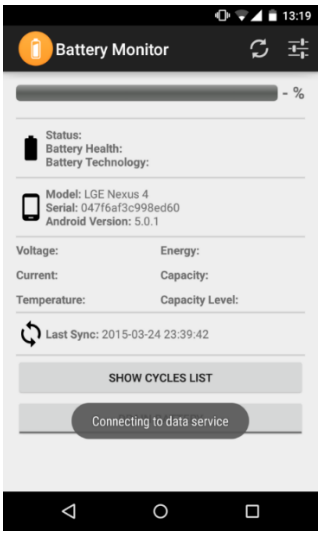


Fig. 3.4 Starting application main activity

This main screen contains the most important information for user about their device battery. Following screenshots, Fig. 3.5, show the activity with loaded information in the latest Android version and for Android version 2.3.7. Furthermore, they show the application layout in different screen sizes.

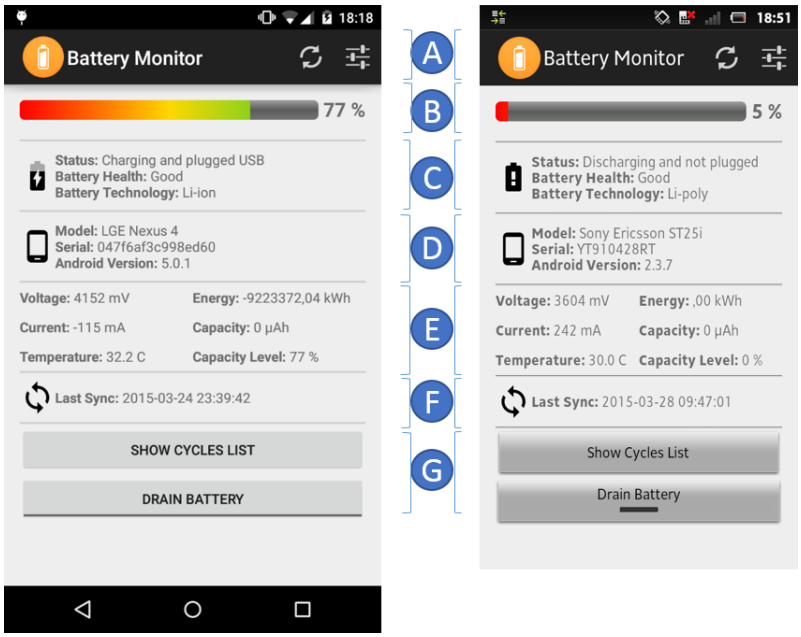


Fig. 3.5 Information in main screen

A. Android action bar contains icon, name and two action buttons.
    1. Sync button: User can synchronize current data from table Battery without waiting 15 minutes.
    2. Preferences button: Navigate to user settings activity.

B. Custom progress bar. Specific design for identify current level of battery with colour gradient and with percentage number.

C. Status Battery section. The icon image is generated dynamically depending on status and level battery. This section displays information about status, plugged type, health and technology.

D. Device section. It shows manufacturer, model, serial and android version related to device.

E. Battery stats section. Voltage, current, temperature, energy remaining, capacity and capacity level are displayed in this section.

F. Indicates to user when last data synchronization with server has been performed.

G. “Show cycles list” button navigates to list activity screen and it displays cycles information. “Drain battery” button is a toggle button for activate fast discharge, that is described below.

3.6.1 Fast discharge option

Application not only shows information about battery but also includes an option to perform a constant fast battery discharge.

It has been developed to perform tests more quickly. This button enables the user to drain the battery.

It is well-known that the most draining battery components are:

- CPU
- Brightness screen
- Vibration
- WIFI
- GPS
- Bluetooth

Applications uses full wake lock, which is a mechanism to indicate that it needs to have the device stay CPU on and screen on. It turns GPS on with zero time polling intervals. Also it turns Bluetooth on and it continuously scan for new devices (Fig. 3.6). The remaining two components have been discarded, since vibration may disturb the user and WIFI can interfere with synchronization process.

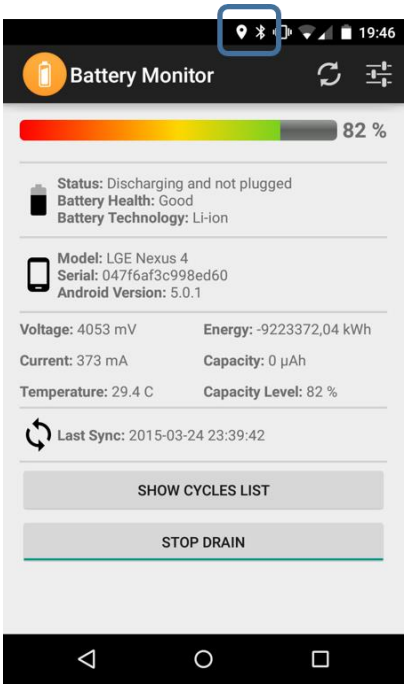


Fig. 3.6 Fast discharge option activated

The following graphs in Fig. 3.7 show how devices are discharged in approximately 4 hours in a constant way, when this option is active.

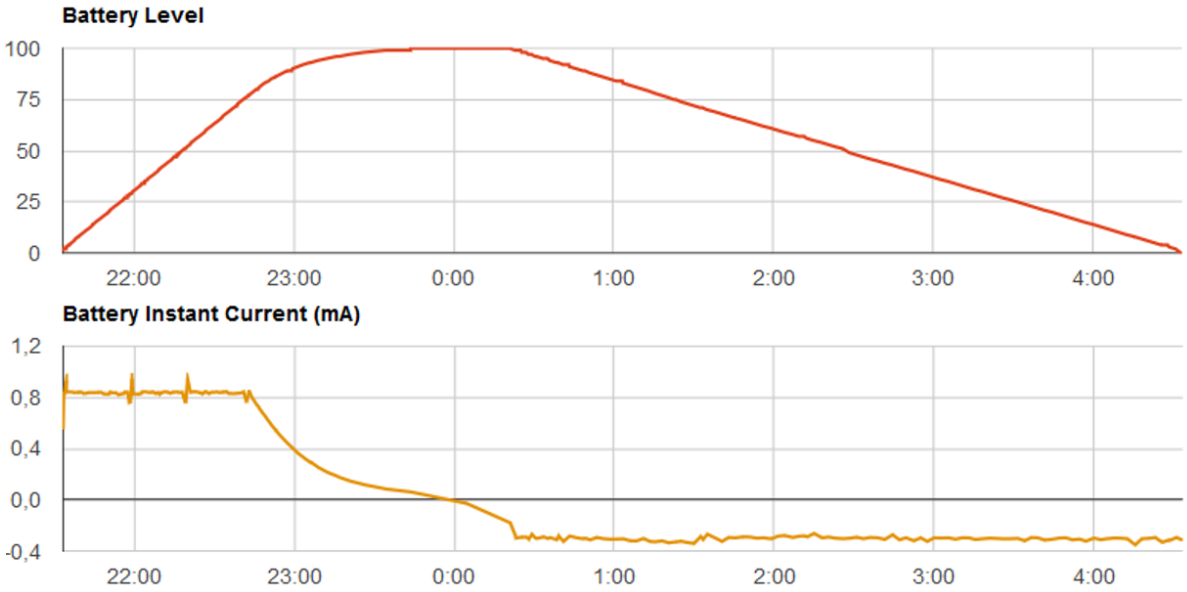


Fig. 3.7 Discharge graph when fast discharge is active

3.6.2 User Preferences screen

User preference activity is based on Android settings screen. It contains few configurations that can be modified by user.

As it is shown in Fig. 3.8, it is divided in three categories:

- Cycles Settings: it is possible to modify maximum and minimum level's threshold in order to detect complete cycles.
- Drain Settings: user can deactivate components used when fast discharge option is turned on.
- Database Settings: By default, synchronized data with server are deleted. However, user can change this rule with the purpose of keeping data on mobile database.

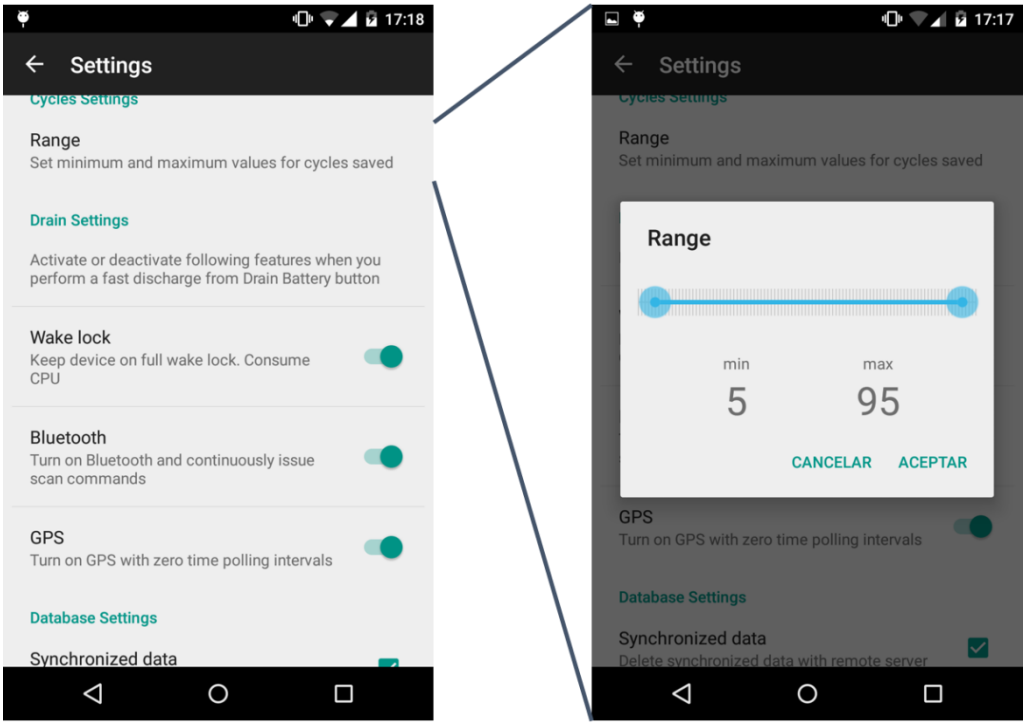


Fig. 3.8 User preferences screen

3.6.3 Cycles list screen

By pressing "show cycles list" button on main screen, user accesses to the historical list of complete charge and discharge cycles.

Data from cycles are removed when they are synchronized. Nonetheless, cycle's table keeps registers to inform user about complete cycles (number, type and time). It is observed in Fig. 3.9.

Fig. 3.9 Cycles' List screen

3.7 Project structure

The Eclipse plug-in Android Developer Tools (section 2.1.3) defines a basic project structure for Android applications (**Fig. 3.10**).

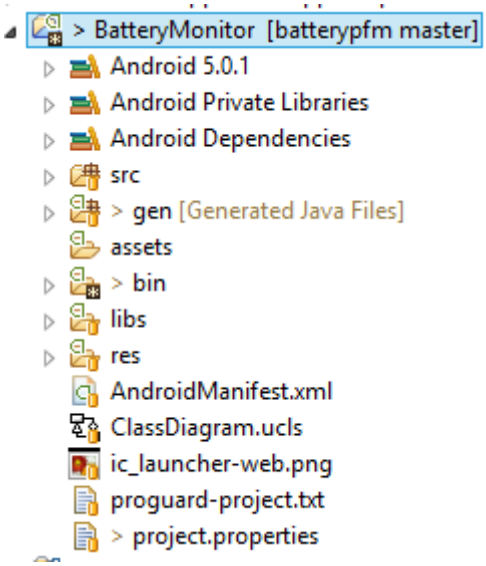


Fig. 3.10 Project structure for Android applications

- *src* folder: It contains the Java source files.
- *gen* folder: This folder contains java files generated by ADT. These files have references to various resources placed in the application.
- *bin* folder: It is the area used by the compiler to prepare the files to be finally packaged to the application's APK file.
- *res* folder: It contains all the resources required like images, layouts and values.
- *Libs* folder: It contains precompiled third-party libraries (JAR archives). Battery monitor application uses two external libraries:
    - android-async-http-1.4.6: it is useful to perform asynchronous requests to the web server.
    - gson-2.2.4: it is a Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object.

Furthermore, basic structure includes important files for the Android projects:

- *AndroidManifest.xml*: It is the Android definition file. It contains information about the Android application such as minimum Android version, permission to access Android device capabilities, definition of the application components (activities, services, receivers, etc). Battery monitor application requires permission to Internet, GPS, Bluetooth, boot complete and wake lock capabilities.

```
<uses-sdk
        android:minSdkVersion="10"
        android:targetSdkVersion="21" />

 <uses-permission android:name="android.permission.INTERNET" />
 <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
 <uses-permission android:name="android.permission.WAKE_LOCK" />
 <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
 <uses-permission android:name="android.permission.BLUETOOTH" />
 <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

- *Project.properties*: It is the main project's properties file that contains information such as the build platform target (API 21) and the library dependencies.

3.7.1 Source folder

Developed java classes are divided in different packages depending on their function. Annex 3 contains the app diagram class. **Fig. 3.11** shows the source folder structure.

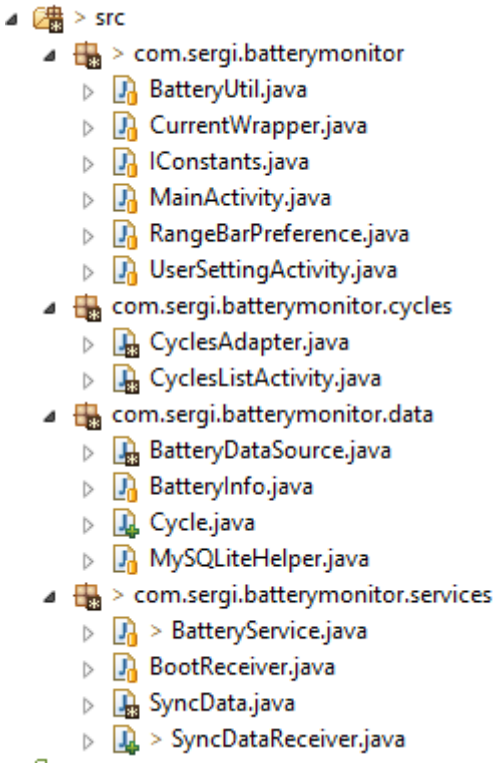


Fig. 3.11 Application java files

com.sergi.battery package contains classes related to the main activity, user settings activity, constants and additional classes to obtain the current. Further information is provided below.

- *MainActivity.java:* It is the entry point when user press application icon. It manages the information and options presented on the main screen (see section 3.6).

- *UserSettingActivity.java:* It controls user preferences view (see 3.6.2).

- *RangeBarPreference.java:* It handles the specific component created to select minimum and maximum values for the cycle's thresholds. This component is displayed on Fig. 3.8.

- *CurrentWrapper.java:* Android devices with version prior to 5.0 use this class to look up the file location where current information is placed. It contains specific locations for each device model.

- *BatteryUtil.java:* It groups support functions, such as, access to files to read current information.

- *IConstants.java:* Constants values are placed on this class. For instance, it has web server address and the period time for each synchronization.

```
WS_HOST = "http://192.168.1.132";
URL_insertdata = WS_HOST + "/batteryinfosync/insertdata.php";
SYNC_TIME = 15;
```

There are two classes into the cycles' package. *CyclesListActivity.java* obtains the data for the cycles list screen. *CyclesAdapter.java* manages these data and defines the format for each row of the list.

com.sergi.battery.data package has two classes (*BatteryInfo.java* and *Cycles.java*), which contains the battery and cycle properties' definition. Moreover, classes to manage SQLite database are located here:

- *MySQLiteHelper.java*: This class provides the connection to the Android SQLite database and it is extended to add the script for creating the tables.

- *BatteryDataSource.java*: It contains the queries to interact with the database (see 3.4).

Finally, services' package encloses java files for battery service, broadcast receivers and synchronization module.

- *BatteryService.java*: It is the main component of the application. It is described in section 3.2.

- *BootReceiver.java*: This class is the broadcast receiver for the boot event. It starts battery service when device is turned on.

- *SyncDataReceiver.java*: Every fifteen minutes, an event is captured by this class in order to activate the synchronization module.

- *SyncData.java*: It contains Java code for the synchronization module explained in section 3.5.

3.7.2 Resources folder

In order to maintain resources independently, it is necessary to externalize them from the code. Externalizing application resources allows to provide alternative resources that support specific device configurations such as different languages, screen size or system version. As it can be seen in the **Fig. 3.12**,

the res directory contains all the resources that groups them in sub-directories by type and configuration.

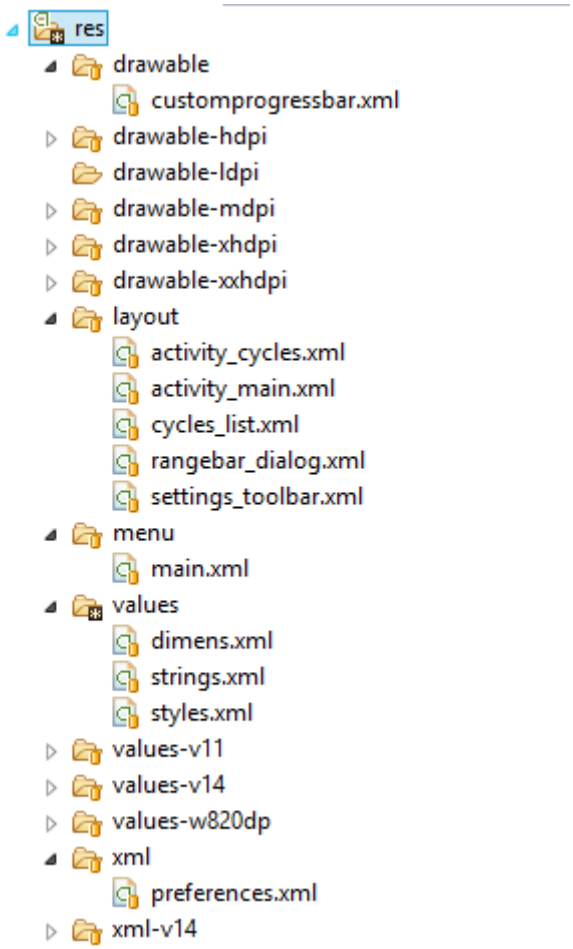


Fig. 3.12 Resource's folder

- *Drawable* folder: A drawable resource is a general concept for a graphic that can be drawn to the screen, in this case contains and XML file that defines a geometric shape, colors and gradients of the custom progress bar displayed in the main screen (Fig. 3.5).

- *Drawable-*dpi* folders: This folders contain icons and images used in the user interface. They are divided into sub-directories depending on screen density [15]. **Fig. 3.13** shows the relation between screen size and screen density.

    - *ldpi*: Low-density screens; approximately 120dpi.
    - *mdpi*: Medium-density (on traditional HVGA) screens; approximately 160dpi.
    - *hdpi*: High-density screens; approximately 240dpi.
    - *xhdpi*: Extra-high-density screens; approximately 320dpi.
    - *xxhdpi*: Extra-extra-high-density screens; approximately 480dpi

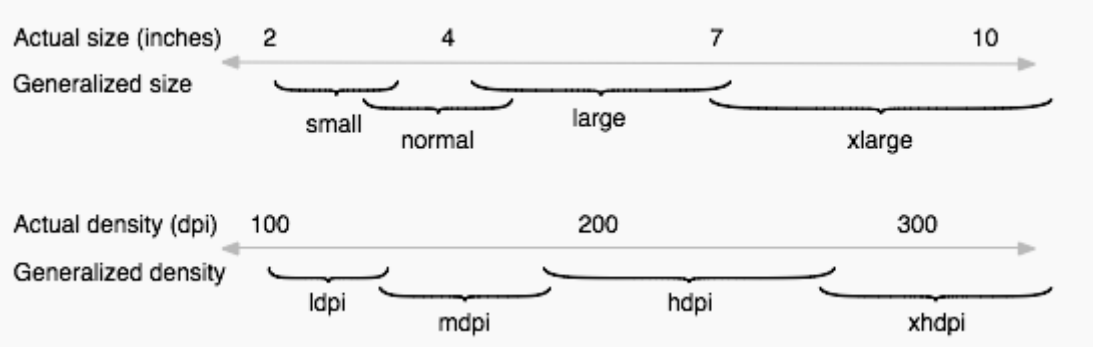


Fig. 3.13 Map of actual screen sizes and densities.

- *Layout* folder: Layout resource defines the architecture for the user interface in the activities or components.

- *Menu* folder: XML file that define application menu placed on the top bar, which contains the application icon and name, and the options buttons.

- *Value* folder: It defines values of the application for the text strings, style and dimensions. This XML files are divided depending on the platform version, for instance –V11 for API level 11 (devices with Android 1.6 or higher). Please refer to annex 4 for further information about these values. Furthermore, it can contain sub-directories depending on the screen width, for devices with 820dpi exists a specific dimension file.

- *Xml* folder: It contains XML configuration files, such as, preference screen configurations. Preference.xml defines categories and properties that user can modify in the preference activity.

CHAPTER 4.SERVER

Server provides a web service where the application connects to, in order to send the information using its synchronization module (section 3.5). Server classifies received data, processes entropy results and stores all into a database. Finally, server makes available this information in a website. As it is shown in **Fig. 4.1**, server is composed for three modules: web service, database and dashboard.

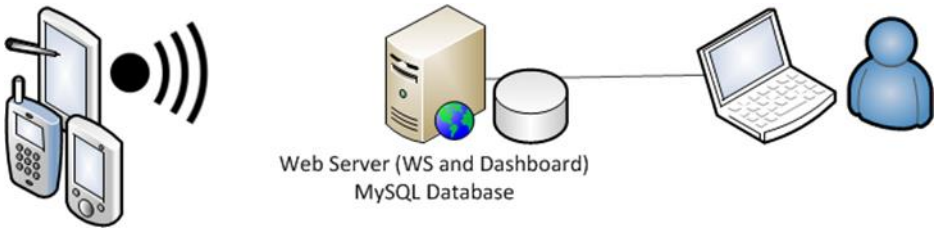


Fig. 4.1 Server's modules

In this chapter, server implementation is defined. Firstly, MySQL database is presented. After that, the entropy calculation and the techniques to obtain the open circuit voltage of the battery are explained. These calculation and methods are applied on the web service workflows subsequently detailed. Furthermore, website where user can obtain data captured by the mobile application, is described on dashboard's section.

4.1 Database

Server's database supports data from multiple devices. All devices registered are on devices' table, which primary key is *_id*. The other tables relate to this one through the *device_id*.

Furthermore, the server's database includes extra fields and tables for calculated data, such as open voltage circuit or entropy.

The following Fig. 4.2 shows the entity relationship diagram of MySQL database:

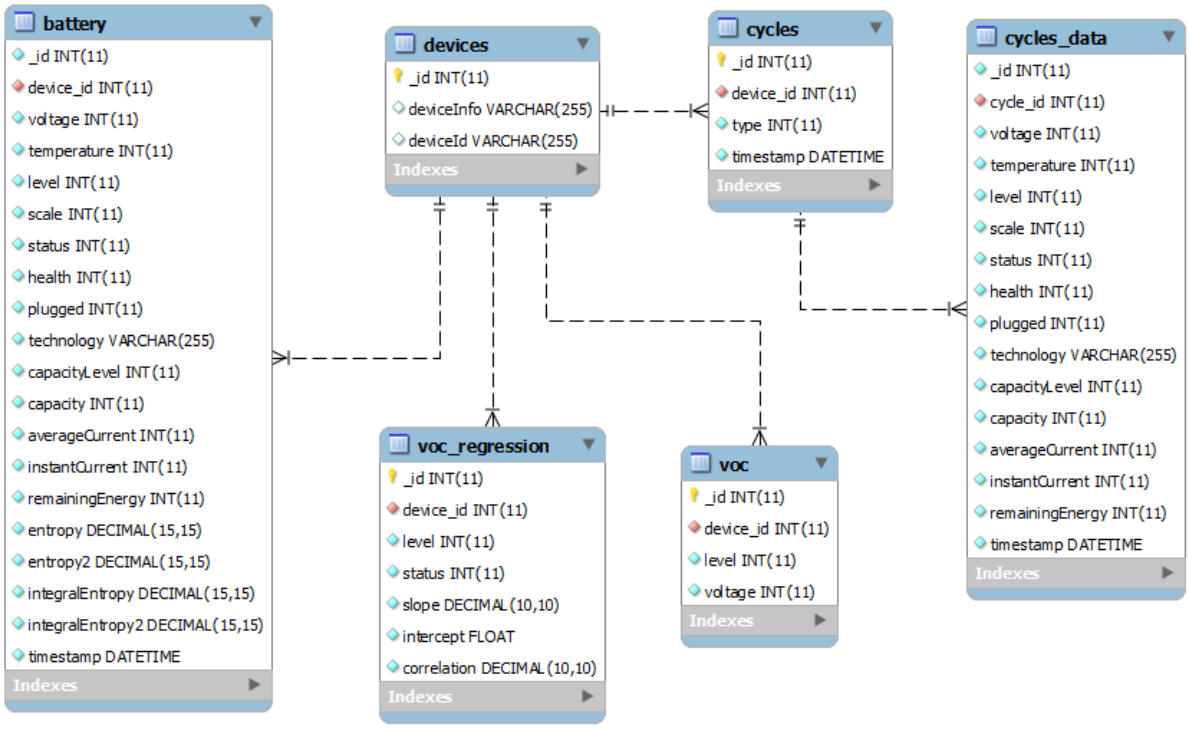


Fig. 4.2 ERD of MySQL server database

In addition, there is another table unrelated to battery data. The user's table contains the username and password of the users who can access to dashboard web.

4.2 Entropy Calculation

Calculation of battery entropy is based on patent [16]. Entropy equation is described by:

$$\Delta S = \int S_{rate}\, dt = \int \frac{P}{T} dt \qquad \textbf{(4.1)}$$

Where S_{rate} is rate of entropy generation in the system, T is measured temperature and P is power consumption that under adiabatic or isometric conditions can be approximated by:

$$P = (V_{oc} - V_{bat})I \qquad \textbf{(4.2)}$$

Where V_{oc} is the open circuit voltage (OCV), V_{bat} battery voltage and I is charging or discharging battery current. However, in order to measure the open circuit voltage V_{oc}, the battery must be disconnected from the device.

Since this is not possible while the application is running, estimation methods for this property are required.

Altogether, S_{rate} is calculated with information provided from battery monitor application and OCV estimations. Finally, trapezoidal rule is used to integrate the rate of entropy generation. The trapezoidal rule is a technique for approximating the definite integral. It approximates the region under the graph of the function to a trapezoid (Fig. 4.3) and it calculates the area. The trapezoidal rule follows the equation (4.3):

$$\int_a^b f(x)\,dx \approx (b-a)\frac{f(a)+f(b)}{2} \tag{4.3}$$

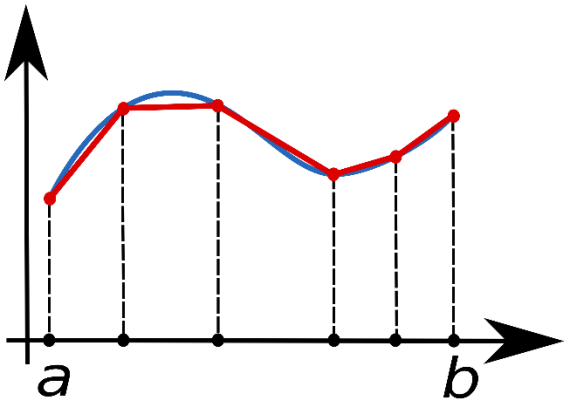


Fig. 4.3 Graph divided in trapezoids

Applying this technique to entropy equation (4.1):

$$\Delta S = \int_{t_1}^{t_2} S_{rate}(t)\,dt \approx (t_2-t_1)\frac{S_{rate}(t_1)+S_{rate}(t_2)}{2} = (t_2-t_1)\frac{\frac{(V_{oc|t_1}-V_{bat|t_1})I_{t_1}}{T_{t_1}}+\frac{(V_{oc|t_2}-V_{bat|t_2})I_{t_2}}{T_{t_2}}}{2} \tag{4.4}$$

Two different approaches regarding OCV estimation are used for the entropy calculation. Both of them are described below.

4.2.1 OCV with mean estimation

The open-circuit voltage of the battery cannot be directly measured because the battery needs to be disconnected for three hours minimum. However, according to [17] , the OCV can be found in the area between charging line and

discharging line (Fig. 4.4). Therefore, OCV is calculated from mean voltage values of each SoC level.

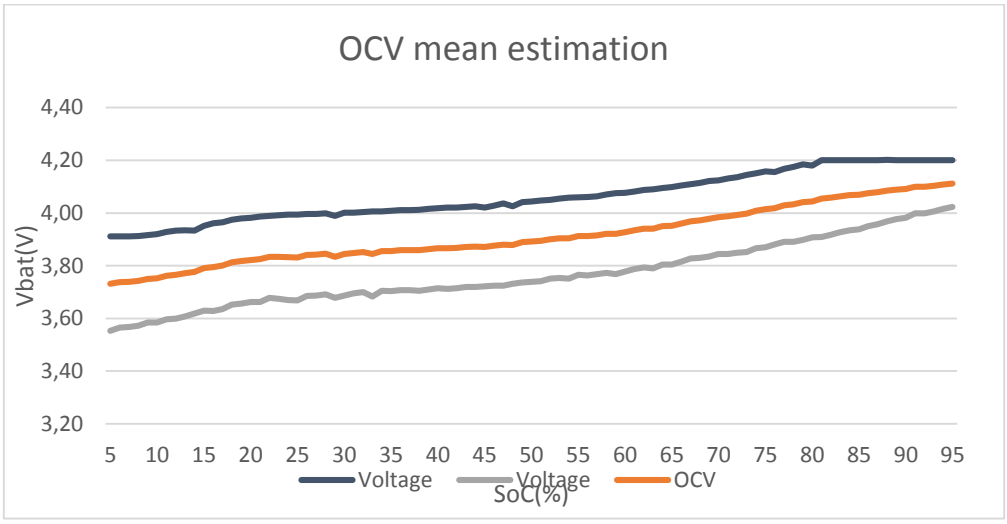


Fig. 4.4 OCV region limited by charge and discharge battery voltage (data in annex 2)

4.2.2 OCV with linear regression method

This OCV estimation can be performed using a simple battery model, as shown in Fig. 4.5. It consists of an ideal battery with open-circuit voltage V_{oc}, a constant equivalent internal resistance R_{int} and the battery voltage represented by V_b.[4]

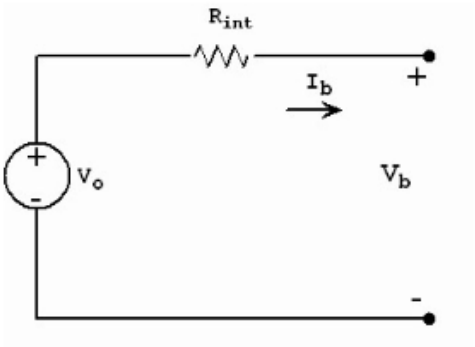


Fig. 4.5 Simple battery model

I_b is obtained from:

$$I_b = \frac{v_{oc} - v_b}{R_{int}} \quad \textbf{(4.5)}$$

Hence, it is possible to transform it to straight line equation where y-intercept is v_{oc}

$$y = ax + b \tag{4.6}$$

$$v_b = -R_{int}I_b + v_{oc} \tag{4.7}$$

Current $I_b$and battery voltage are collected from battery monitor application. Therefore, a regression line can be estimated with some points of each SoC. Only last five points are entered into regression because aging is an important factor on battery values and it could introduce a bias. Fig. 4.6 shows linear regression for 30% of SoC and when battery was in charging stage. It has been found that the internal resistance is different under discharge and charge conditions, consequently, it is necessary to create different regression lines according to battery status and SoC.

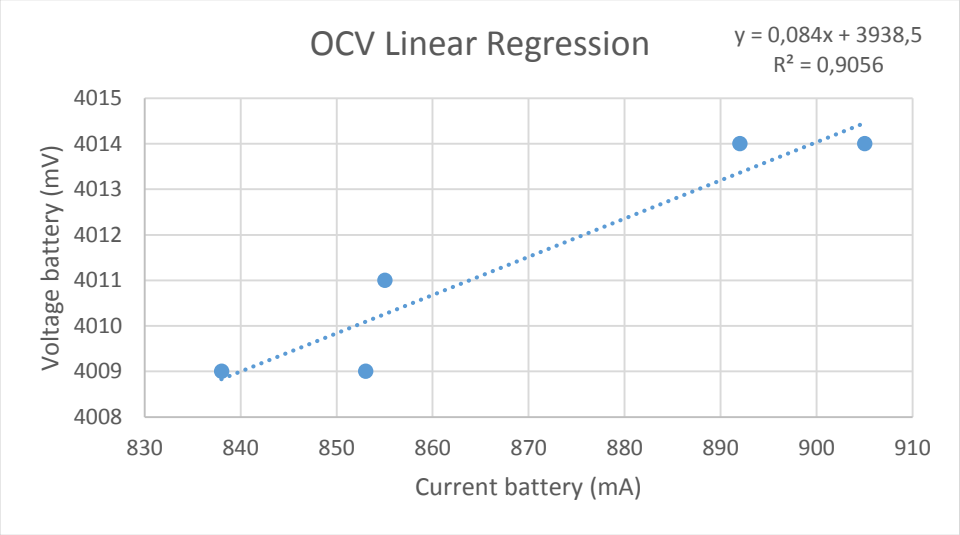


Fig. 4.6 Linear regression for 30% SoC in charging status. Please, note that in equation slope be negative. Indicate, if so, that charging current is negative and thus, slope is positive.

Mathematically, ordinary least squares (OLS) approach is used to fit the regression line. This method calculates the best-fitting line for the observed data by minimizing the sum of the squares (Q) of the vertical deviations from each data point to the line.

$$Q = \sum_{i=1}^{n}(y_i - y)^2 = \sum_{i=1}^{n}(y_i - ax_i - b)^2 \tag{4.8}$$

Then, Q will be minimized at the values of a and b for which $\frac{\partial Q}{\partial b} = 0$ and $\frac{\partial Q}{\partial a} = 0$. The first of these conditions is,

$$\frac{\partial Q}{\partial b} = \sum_{i=1}^{n} -2(y_i - ax_i - b) = 2(nb + a\sum_{i=1}^{n} x_i - \sum_{i=1}^{n} y_i) = 0 \quad \textbf{(4.9)}$$

Which, solving b,

$$b = \bar{Y} - a\bar{X} \quad \textbf{(4.10)}$$

The second condition for minimizing Q is,

$$\frac{\partial Q}{\partial a} = \sum_{i=1}^{n} -2x_i(y_i - b - ax_i) = \sum_{i=1}^{n} -2(x_i y_i - bx_i - ax_i^2) = 0 \quad \textbf{(4.11)}$$

Now, the expression for b from (4.10) could be used in (4.11),

$$\sum_{i=1}^{n}(x_i y_i - x_i\bar{Y} + ax_i\bar{X} - ax_i^2) = \sum_{i=1}^{n}(x_i y_i - x_i\bar{Y}) - a\sum_{i=1}^{n}(x_i^2 - x_i\bar{X}) = 0 \quad \textbf{(4.12)}$$

Finally, solving a,

$$a = \frac{\sum_{i=1}^{n}(x_i y_i - x_i\bar{Y})}{\sum_{i=1}^{n}(x_i^2 - x_i\bar{X})} = \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i)}{n\sum_{i=1}^{n} x_i^2 - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i} \quad \textbf{(4.13)}$$

In conclusion, it is possible to calculate the slope (a), which is equal to internal resistance (4.7), by using values measured by the monitor application. Afterwards, the intercept (b) is obtained (4.10). Finally, it should be borne in mind that intercept is equal to open circuit voltage (OCV) in the simple battery model previously defined. **Fig. 4.7** shows an example of estimated OCV values using linear regression.

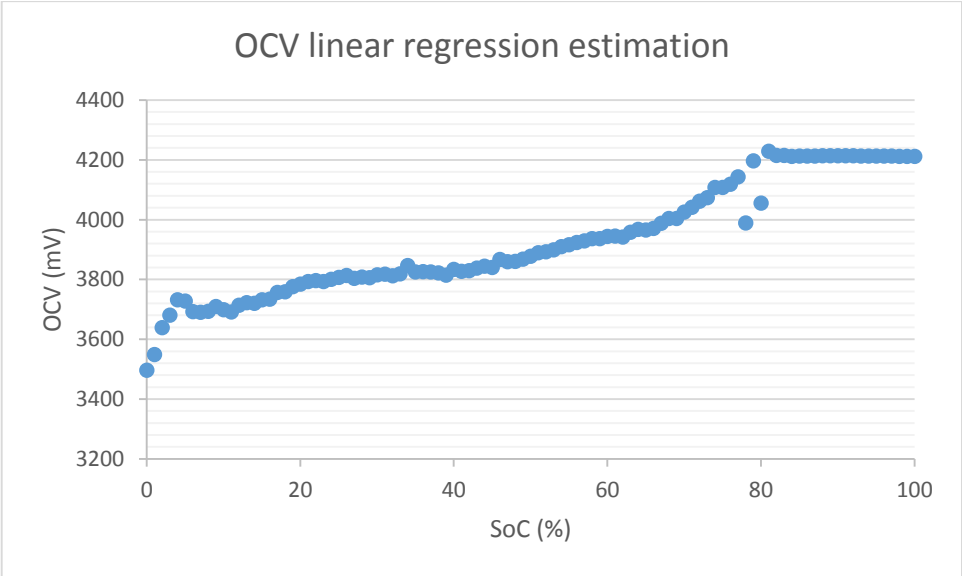


Fig. 4.7 OCV values using linear regression estimation

There is a process implemented on SQL that takes last five points for each SoC level and calculates parameters of regression line based on what was described above:

```
SELECT device_id ,level,status,
cast((N * Sum_XY - Sum_X * Sum_Y)/(N * Sum_X2 - Sum_X * Sum_X)as decimal(10,10)) AS Slope,
(Mean_Y - (N * Sum_XY - Sum_X * Sum_Y)/(N * Sum_X2 - Sum_X * Sum_X) * Mean_X) AS intercept,
(N * Sum_XY - Sum_X * Sum_Y)/SQRT((N*Sum_X2 - Sum_X*Sum_X) * (N*Sum_Y2 - Sum_Y*Sum_Y)) AS
correlation
FROM (
SELECT level,device_id, status,
COUNT(*) AS N,
AVG(instantCurrent) AS Mean_X,
SUM(instantCurrent) AS Sum_X,
SUM(instantCurrent * instantCurrent) AS Sum_X2,
AVG(voltage) AS Mean_Y,
SUM(voltage) AS Sum_Y,
SUM(voltage*voltage) AS Sum_Y2,
SUM(instantCurrent*voltage) AS Sum_XY
FROM  (
SELECT *
FROM battery
WHERE device_id = $device_id
and level = $i
and status = $status
order by timestamp desc limit 5) A
GROUP BY level ) B
```

4.3 Web Service

As explained above, server is implemented with PHP programming language and MySQL and there are different workflows depending on data type sent by the mobile application using the server's web service (*http://host/batteryinfosync/insertdata.php)*. Furthermore, there is a specific web service method to perform OCV linear regression estimation.

4.3.1 Battery information data workflow

Fig. 4.8 shows the sequence diagram when monitor application sends information every 15 minutes from battery table to server (or is manually synchronized by user). The request format is presented in section 3.5.

First, device identifier key is recovered from database. If the device does not exist on server, a new register is created into device's table. Next, the server calculates two slightly different entropy values for each register inserted on database, according to each OCV estimation method. Server always has a reference of the last information inserted in this workflow, with the purpose of calculating the integral.

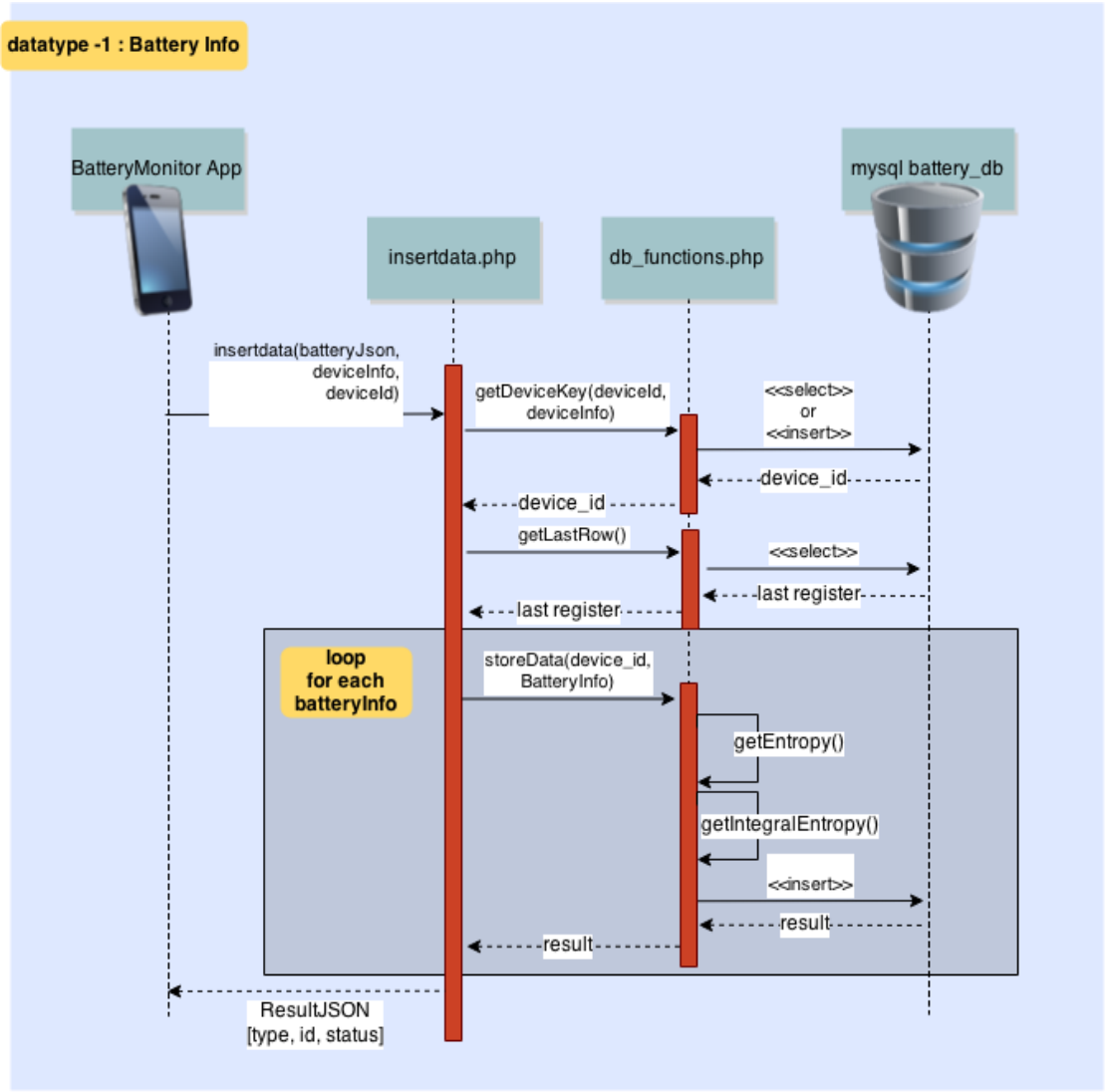


Fig. 4.8 Sequence diagram for battery data

4.3.2 Battery cycle data workflow

Application sends cycle information to server using synchronization module when each cycle is completed. Server performs following workflow for this type of data (Fig. 4.9). The request format is presented in section 3.5.

As described above, first, it searches the key of device and if it does not exist, a new register is created. Next, it inserts a new register into cycle table, including the type of cycle (charging or discharging) and timestamp.

Afterwards, server inserts data information for each battery level of the cycle into *cycles_data* table. This process is transactionally performed to ensure that all records are inserted correctly. If not, the cycle register saved before is deleted.

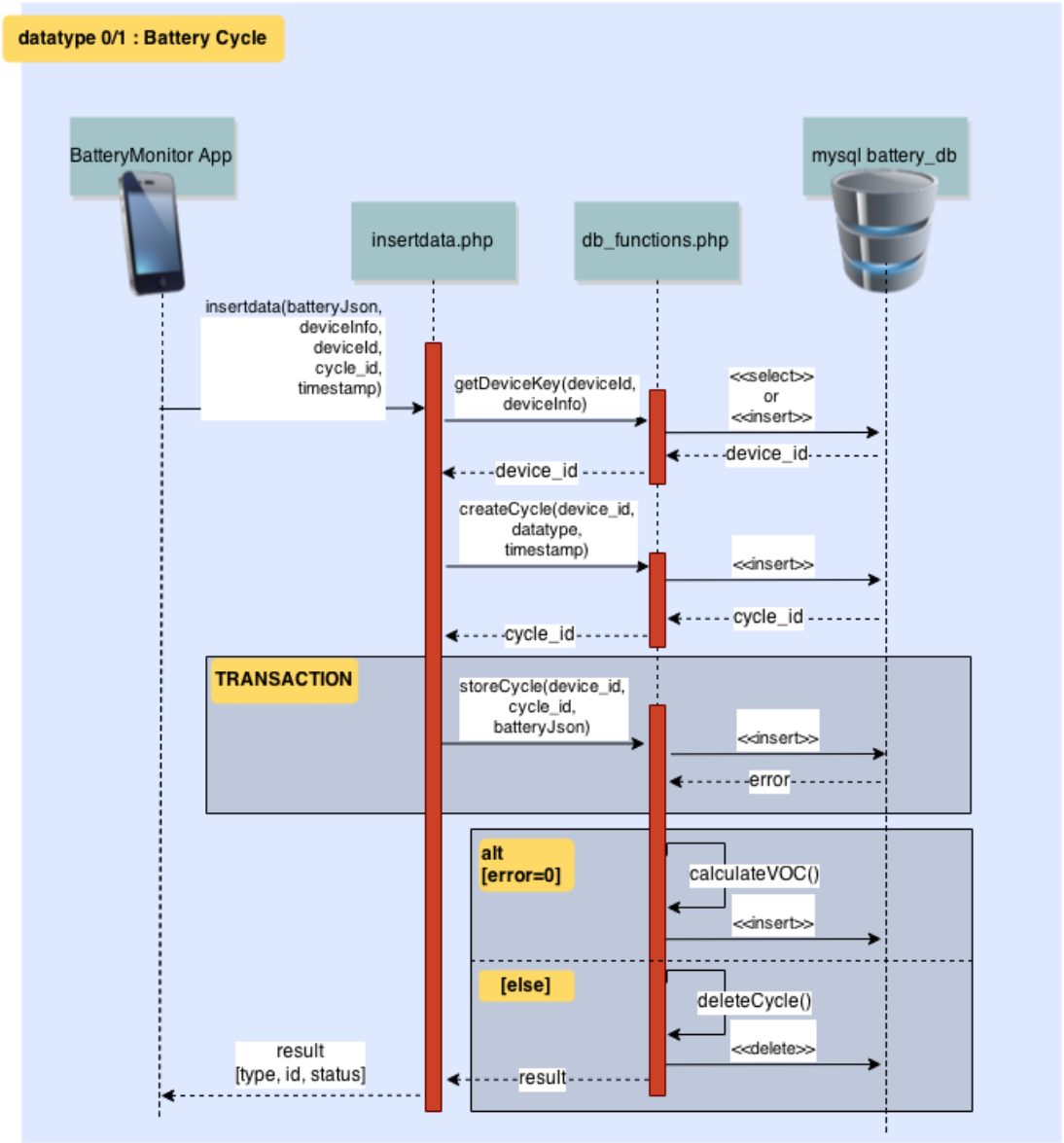


Fig. 4.9 Sequence diagram for cycle data

At the end, OCV with mean estimation is calculated using last charge and discharge cycles as explained above.

4.3.3 Cron for OCV linear regression estimation

In order to calculate OCV with linear regression estimation (see section 4.2.2), a workflow is created to access it by task scheduler or job and it is observed in Fig. 4.10). The task uses this server's web service:

http://host/batteryinfosync/cron.php

This task gets a list with all the devices in the database and calculates the linear regression for each SoC level and charging or discharging status, using the algorithm explained above.

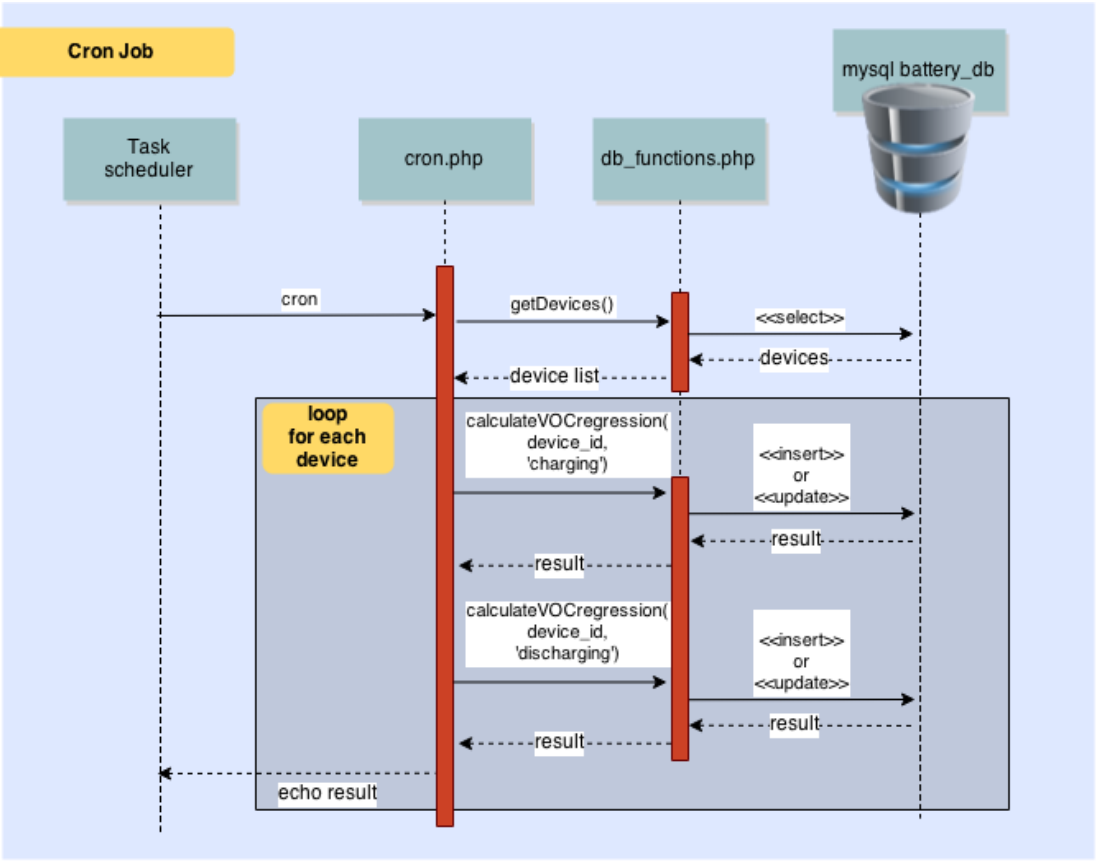


Fig. 4.10 Sequence diagram for perform OCV linear regression estimation

4.4 Dashboard

4.4.1 Login page

Users can get battery information in a dashboard web located in the main page:

http://host/index.php

Dashboard is secured and thus a properly authenticated access is required. Although it only contains information about user's battery, these data can be used in order to identify user's routines thanks to predictive analytics techniques. In other words, through information about battery so many users' actions could be identified. For instance, when user wakes up, goes to bed or even when is driving.

Authenticated access is performed by login form (Fig. 4.11), where users have to introduce valid credentials (username and password) to access the data.

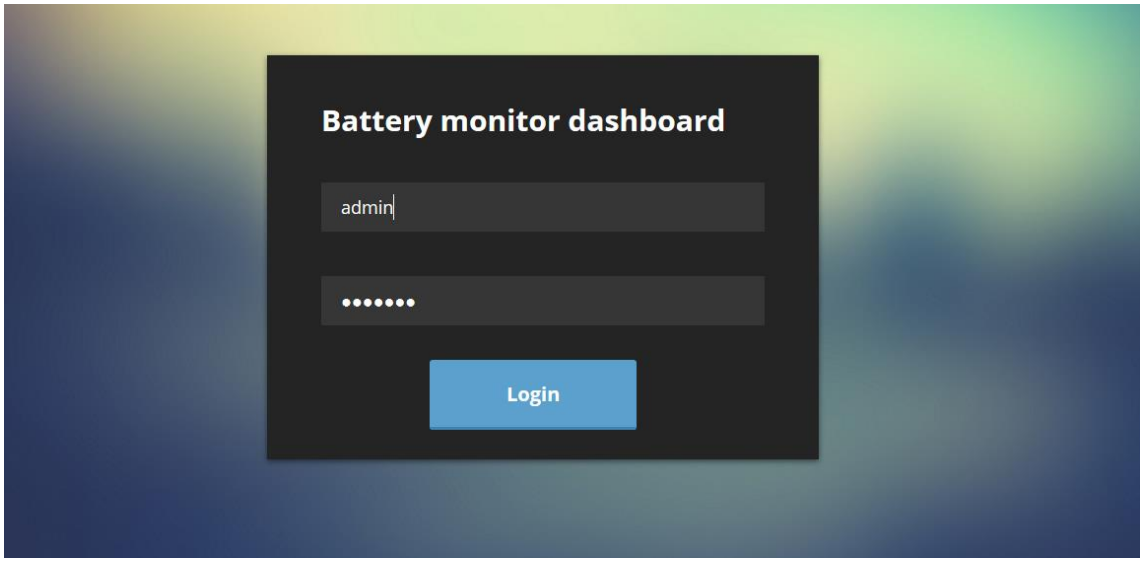


Fig. 4.11 Login page

Login form checks user's credentials on users' table in the database. If the user signed in successfully, login page establishes a session and puts username into this server's session. Finally, user is redirected to main page.

Fig. 4.12 Page top bar

Main page includes a bar on top with information about the user signed (Fig. 4.12) and contains a sign out link to finalize the session and be redirected to the login page.

4.4.2 Search form

Dashboard contains a search form, it can be observe in the next Fig. 4.13.

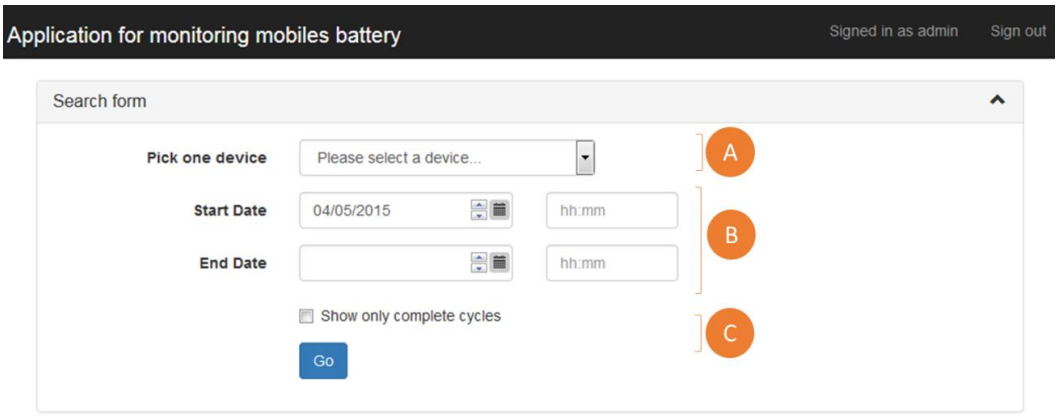


Fig. 4.13 Search Form

A. When page is loaded is connected to PHP server with method *getdevices* and load all devices registered in the database (from devices' table), and put information into this select such as options.

B. User can select a range of dates (date and time). By default, start date is one month before from today, and end date is the current time.

C. If this checkbox is selected, data is recovered from cycles' table instead of battery's table.

When user submits form, it appears new panel for filter results and also shows charts displaying the data selected. It can be observed in Fig. 4.14.

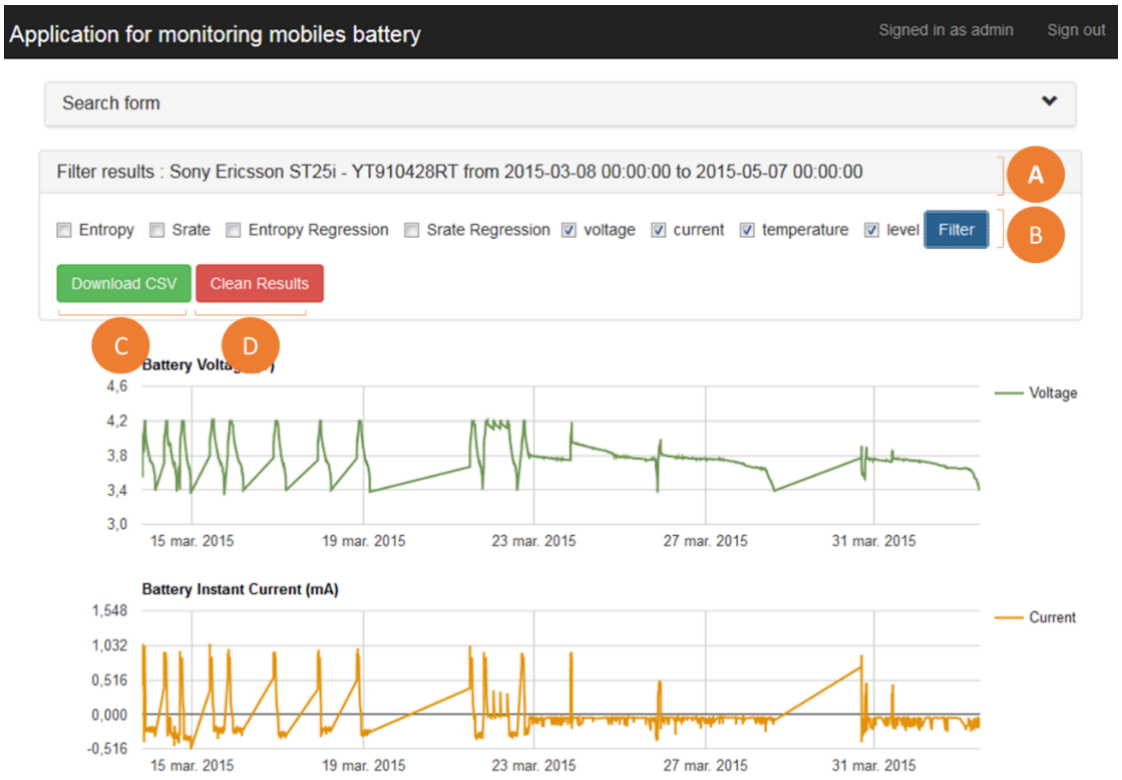


Fig. 4.14 Results displayed in the dashboard

A. Display information about criteria selected on search form.

B. All avalaible charts. User can select the charts that wants to be shown.

C. Download a CSV with all selected data. User can import to excel or another application to work with these data.

D. Clean results and restart dashboard to perform a new search.

4.5 Server files

XAMPP server defines a system folder where files have to be placed. By default, in Windows OS is :

C:\xampp\htdocs

Fig. 4.15 shows structured files for web server. *Batteryinfosync* folder contains PHP files, each one has a specific role.

Fig. 4.15 Server files

- *config.php*: Variables to connect to MySQL database are defined here.
- *db_connect.php*: It establishes connection to database.
- *db_functions.php*: It contains SQL queries to insert and get data and functions to calculate the entropy.
- *Insertdata.php*: It is the PHP class which mobile application connects to send battery data. It reads received data and uses *db_funcions.php* class to insert the data in the database. Finally, it creates a response according to the insert results.
- *Getdevices.php*: Dashboard web loads available devices in its selector component using this class.
- *Getdata.php*: User submits the search form to this class which obtains the information using *db_functions.php*. It returns data with specific format in order to supply the charts data.
- *Cron.php*: It is the file used by a daily process to perform the linear regression calculation, in order to estimate the OCV.
- *Check_user.php:* This PHP class is called from login page to validate user credentials.

Css, *js* and *img* folders contain the resources used by dashboard, such as images, style or JavaScript functions. *Sql* folder has the initial database script.

Finally, *index.php* defines the search form and search results website described in section 4.4.2. *Login.php* has the code for login page and *logout.php* invalidates the session established by the user.

CHAPTER 5.RESULTS

5.1 Set up the test environment

5.1.1 Generating the application APK file

Android application package (APK) is a compressed file and it is used to install and distribute Android applications. Eclipse ADT plugin can export the Battery Monitor application to generate APK file. Android online documentation has a high level of detail regarding this topic. [18]

This procedure should only be performed once, since APK file can be distributed to unlimited android devices.

5.1.2 Installing the battery monitor application

Firstly, APK file must be on the mobile device. It can be passed via through USB connection or as an attachment to an email. It should be borne in mind that to assure the proper functioning of the application, the device has to comply with the following minimum requirements:

- Android version is 2.3.3 or later
- Device connected to the Internet through 3G/4G or WIFI.

Google Play is the official Android application market. Normally, users download and install Android applications from this location. In order to install apps from sources other than Google Play Store, it is necessary to turn on the "Unknown sources" feature on the device's settings, as it is shown in Fig. 5.1.

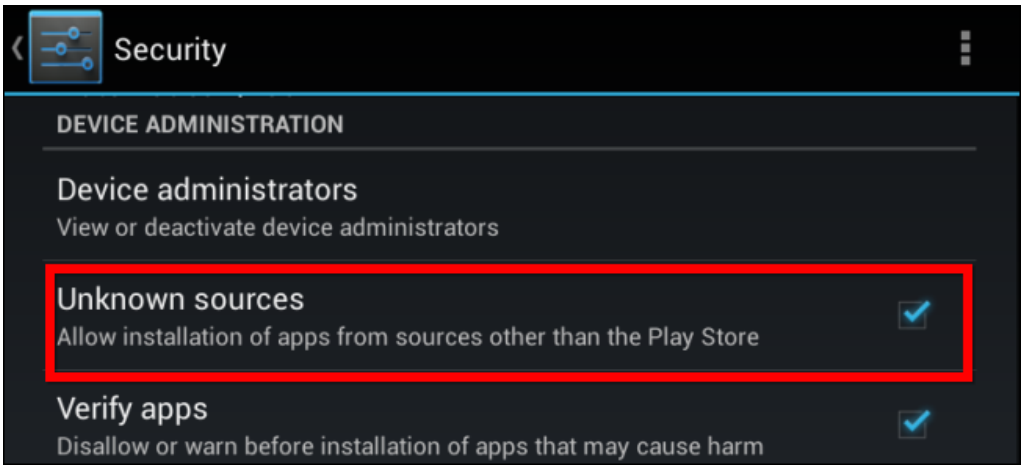


Fig. 5.1 Android security settings

After checking these requirements, the application can be installed opening the APK file in the device. Then platform prompts you to accept the terms (Fig. 5.2) in order to start the installation. It is important to emphasize that the size of the application is only 1.06 Mb.

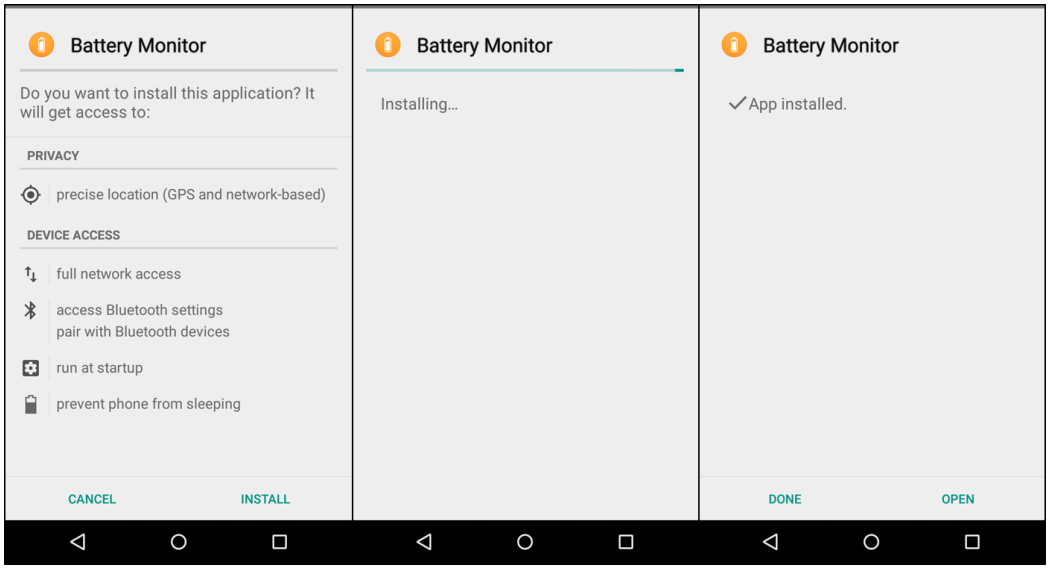


Fig. 5.2 Installation process

5.1.3 Starting the Web Server

The XAMPP Control Panel (Fig. 5.3) allows starting and stopping the Apache Web server and MySQL database. This control panel is located on the windows start menu, all programs, XAMPP, XAMPP Control Panel.

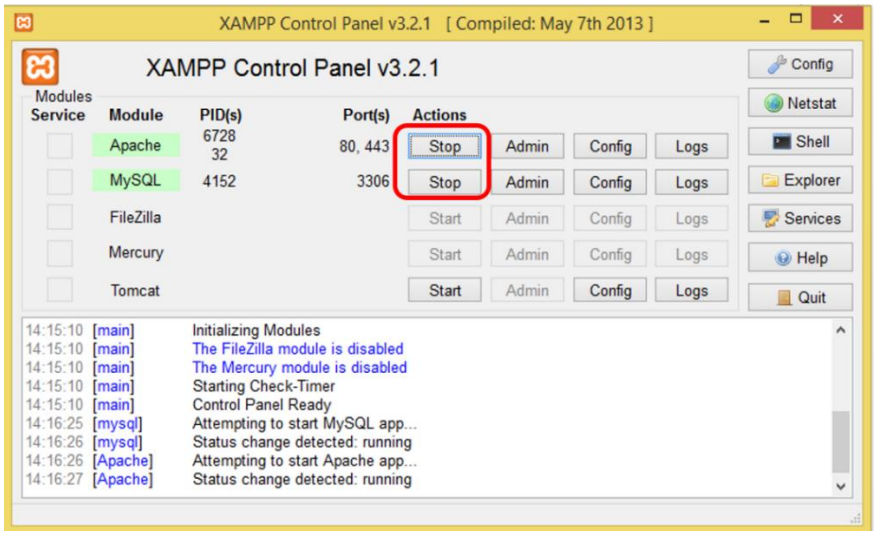


Fig. 5.3 XAMPP Control Panel

5.1.4 Creating the Server Database

It is possible to access to MySQL database through phpMyAdmin tool included in the XAMPP installation. phpMyAdmin is accessible by entering the following address in the web browser:

http://host/phpmyadmin/

This tool requires authenticated access. By default, credentials are username "root" without password. Once inside, it should be pasted and submitted the script generation (provided in the annex 5) into the SQL tab, as it is show on Fig. 5.4.

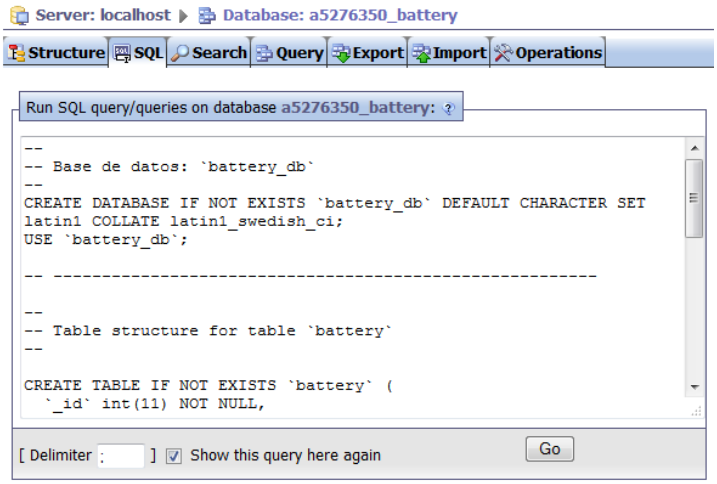


Fig. 5.4 Submitting script in order to generate MySQL database

5.1.5 Creating cron job on Windows server

Estimating OCV with linear regression approach requires a hard processing. For that reason, this work should be performed out of the system routine, in order to avoid delays on the system.

It is possible to program jobs in Windows using the Task Scheduler feature included in the operating system.

Creating a basic task (Fig. 5.5) consists on the setting the following parameters:

- Trigger time: Daily. It is recommended to fix the running time at night.
- Action: Execute cron.php file. It should be placed the following line:

```
C:\xampp\php\php.exe -f c:/xampp/htdocs/batteryinfosync/cron.php
```

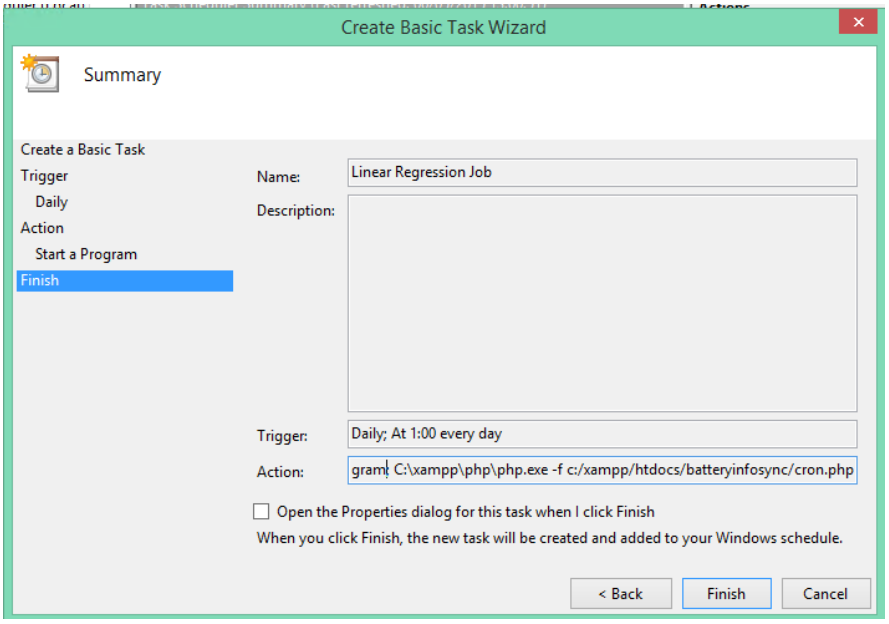


Fig. 5.5 Task properties

5.2 Collecting information

After the mobile application has been installed, the user has to start it in order to activate Battery Service. Once the Battery Service is activated, data from the battery is collected from now on. After that, Battery Service will always run in background, and if device reboots, it starts itself. Fig. 5.6 shows the battery service running in the execution applications menu.

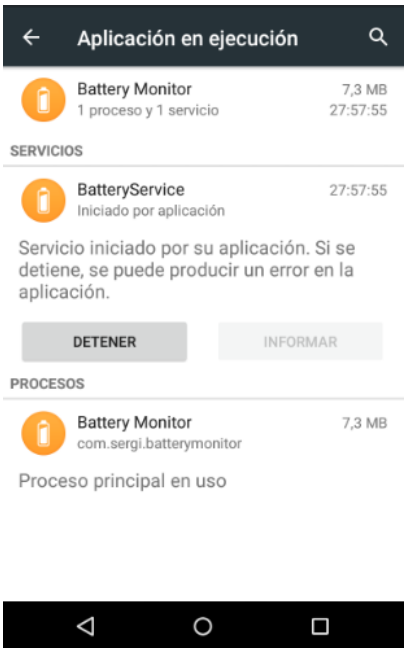


Fig. 5.6 Battery service is running on the smartphone

The application runs without user interaction and periodically synchronizes battery data with the server using the specific web service. Through user interface it is possible to check the current data and the last synchronization performed. As it is shown in Fig. 5.7, application monitors the discharge of the battery by using the drain option, and it sends the data every fifteen minutes.

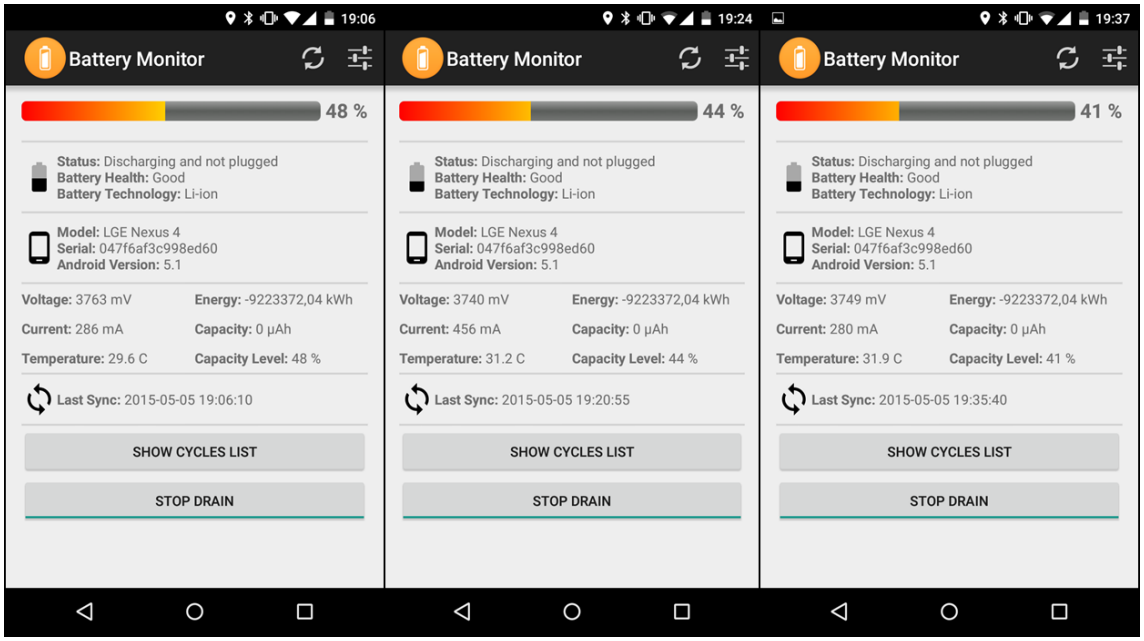


Fig. 5.7 The app is monitoring the battery

Moreover, the application manages the collected data in order to determine a complete cycle. Following the same case presented in the figure above, if the device is continuously discharged from a level of 95% to 5%, the application assimilates it to a complete discharging cycle and it sends the average measurements of the each cycle's level to the server. Finally, a new register referring to this new cycle on the cycles' list screen is added (Fig. 5.8).

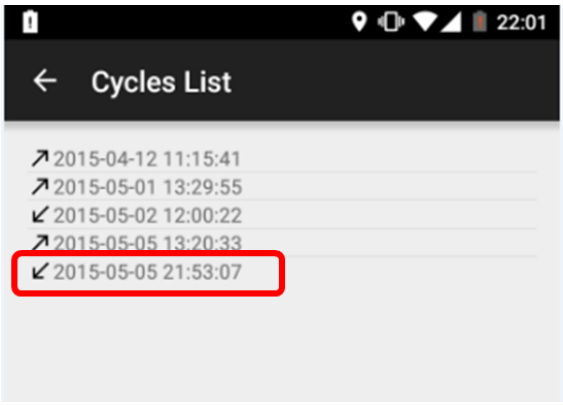


Fig. 5.8 Adding the new completed cycle

5.3 Displaying results

The ultimate aim is that users can check the information sent to the server via battery monitor app. Once users have inserted valid credentials on login page, they access to dashboard page and they can seek data from a particular device in a certain period of time.

For instance, according to the collected information presented in the previous section, one user fills the search form in order to obtain data from LGE Nexus 4 on May 5th from 10 am to 10:30 pm. The search criteria is shown in the following Fig. 5.9.

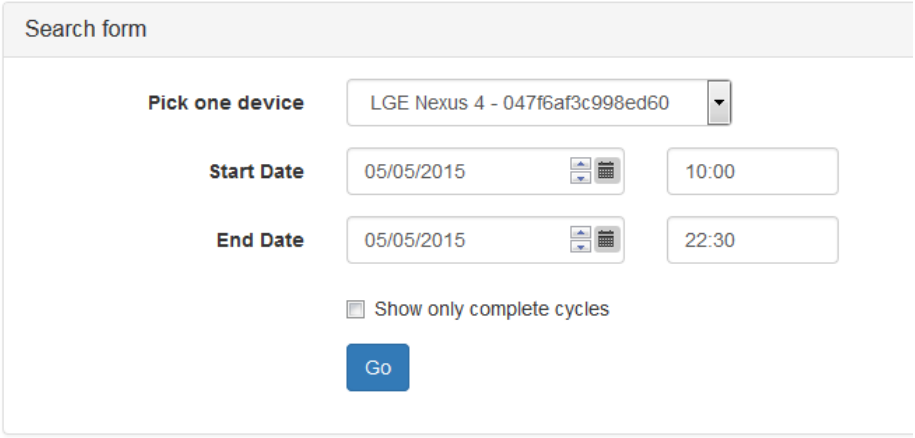


Fig. 5.9 Filled with searching criteria

After few seconds, dashboard displays a set of graphs with the data requested. Fig. 5.10 displays the results in voltage, current, temperature and level graphs. It can be observed that user performed a complete charging cycle and discharging cycle during this time range.

Moreover, it is stated that the charging cycle took less than three hours to be finalized, which is a typical average time in Li-ion batteries. It can be seen that the discharge cycle began with a common use of the smartphone, then about 85% of battery level, the option of fast discharge was activated in order to accelerate the process. It should be appreciated that at level 52% (at time 6:47pm) there was a peak load which caused an increase of temperature and current.

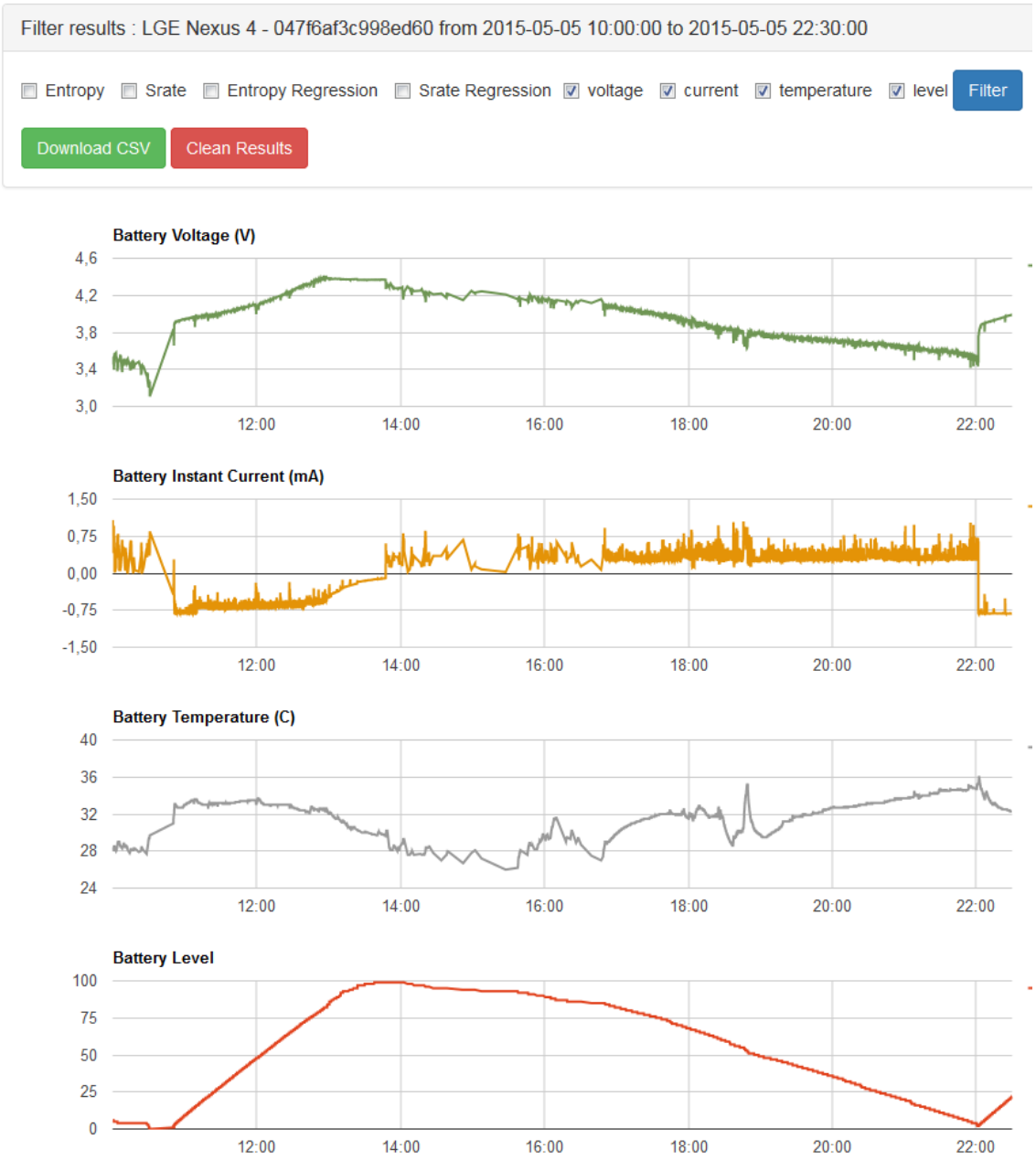


Fig. 5.10 Voltage, current, temperature and level results displayed

In the search form, user has the option to obtain only the complete cycles through the searching criteria. In that case, dashboard obtains the data from cycles' table instead of battery's table. Fig. 5.11 displays the result for the same search performed above but taking into account this option. It should be noted that only the charging and discharging cycles appear and that the graph contains fewer points than before, since the cycles' table only contains the average values for each battery level.

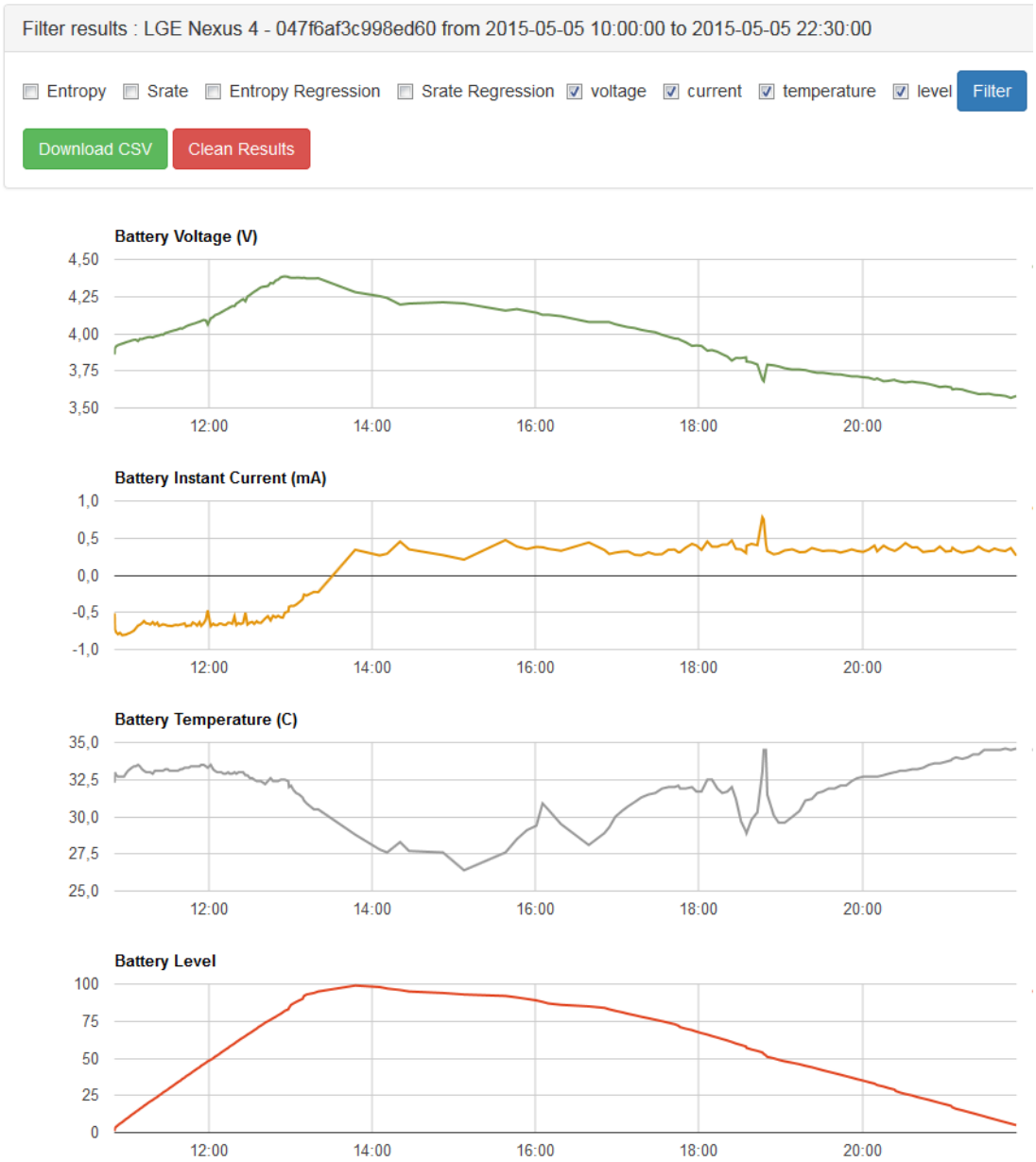


Fig. 5.11 Showing cycles in results

Finally, the calculated entropy and rate of entropy generation (S_{rate}) parameters can be observed. Both are displayed in the Fig. 5.12 and Fig. 5.13, respectively. There are two graphs for each parameter according to the OCV estimations and both are slightly different. However, the peak load detected at time 6:47 pm is exhibited in both graphs. Entropy's graphs have an especial component to control and select the time range, with the purpose of focusing on these specific data.

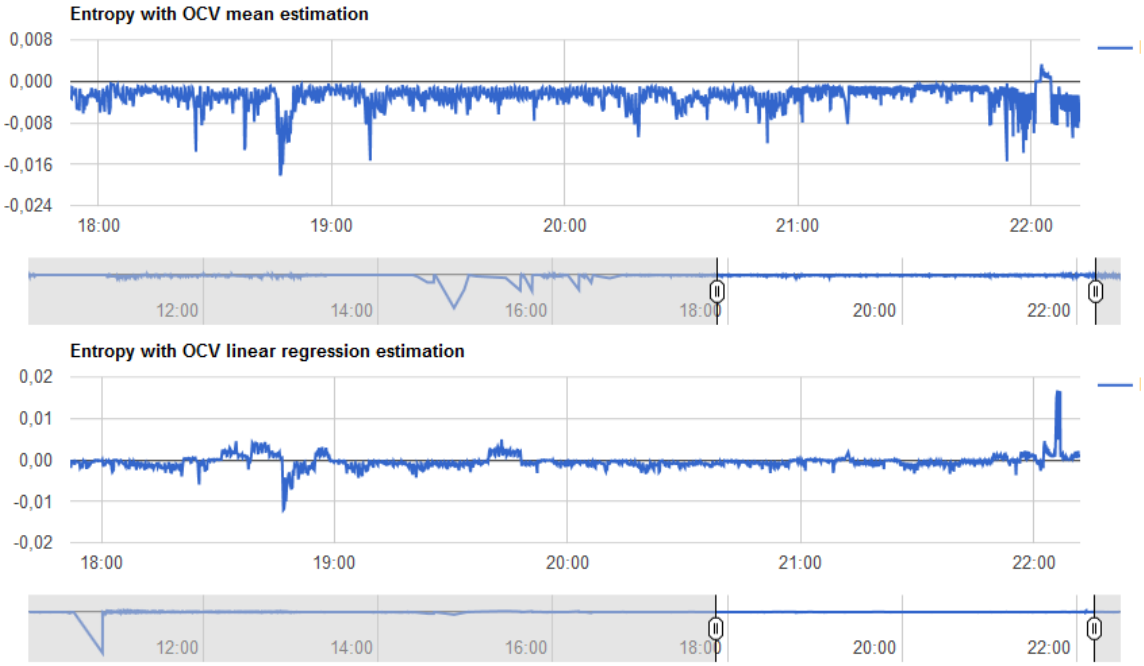


Fig. 5.12 Entropy graphs

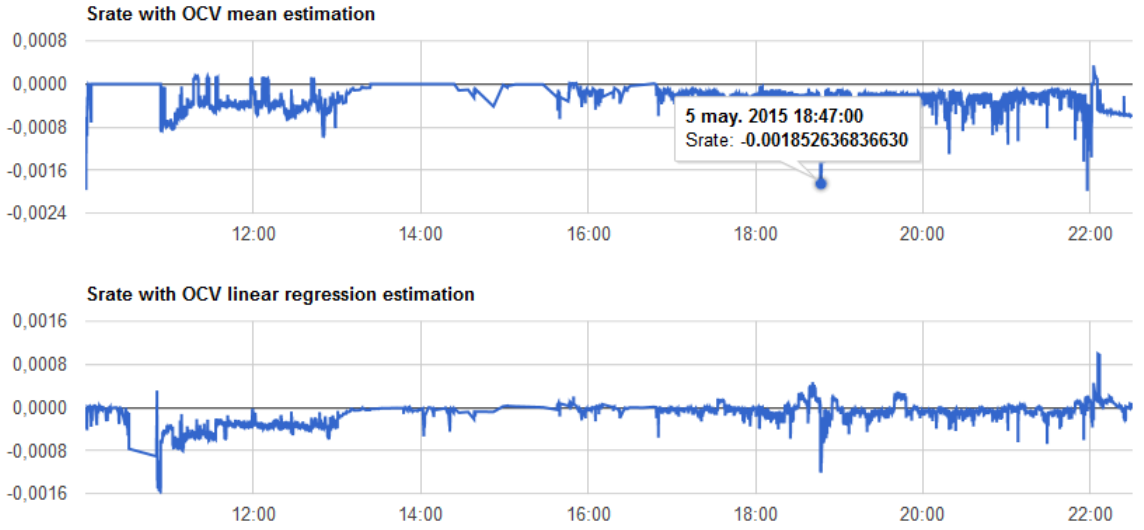


Fig. 5.13 Rate of entropy generation graphs

CHAPTER 6. CONCLUSIONS

A functional system to monitor batteries in mobile phones and transfer information to a server has been created and deployed. The system is developed using the most popular, powerful, free and open source tools and frameworks. For that reason, system is reliable and stable without costs. Actually, this system is monitoring around twenty batteries of different users' smartphones and tablets. It is collecting all the data available and processing this information. Moreover, users have a clear and user-friendly web platform to search for desired information.

The key milestones of the project are detailed below:

- A mobile phone application has been developed using Android SDK and SQLite database. Battery Monitor app is compatible with the 99.6% of Android devices and the last year (2014), Android was the leading mobile platform with 81.5% of the market share. Consequently, the app is available for the 81.17% of the mobile devices. The app has been tested in more than 20 devices, such as, different models of Samsung, LG, HTC, Sony, BQ and Motorola which implies different screen sizes and Android versions. The app takes current, voltage, temperature, status, level, technology, capacity and energy values from batteries. The processes of collecting information and send it by the app are transparent to the user, who does not care about anything. Processes run always on the background and start themselves when device is booted. The Battery Monitor app is able to withstand a server failure since the app keeps unsynchronized data into its database and it retries to send it again in the future.

- Another fundamental part of the system is the Web Service, which enables the app to send the information to the server. The Web Service is capable of handling the information in real time, process it and store it in a structured database. It has been developed using PHP and MySQL and is hosted by XAMPP web server.

- One of the main objectives of the data processing is the entropy calculation. Entropy parameter requires battery voltage, temperature, current and open circuit voltage values in every measure of the battery. Some problems have been encountered with current and OCV values throughout the development stage. The problems together with their solutions are summarized below.

    - First of all, some of the fuel gauge chips do not provide the current property. In addition, Android SDK only exposes this property in its last version 5.0 (API 21), which is a major problem since the latest Android version is only distributed to the 5% of the Android devices. Therefore, a workaround has been implemented to solve this circumstance, which consists in accessing directly to files of the Android operating system. However, the information is placed

on different paths and in different format depending on each manufacturer. For that reason, a large list with possible locations for current values has been created.

- The other troublesome is OCV, which cannot be measured while the application is running because the battery must be disconnected from the device. Hence, two different methods to estimate this property have been used, in order to compare the results and discover similar behaviors.

- The dashboard web is the last but not least important part of the system. Thanks to it, users can manage the information by searching for specific battery stored data and also they can filter different parameters in the displayed results. Additionally, the dashboard has the functionality of exporting the resulting data into a CSV file. It is developed using JQuery, Boostrap and GoogleChart libraries.

Further steps and future work:

- As it is mentioned in the introduction chapter, this system is a useful tool to perform studies about rechargeable batteries. The information collected via the application along with the estimate of entropy can lead to additional and future research about battery degradation and lifetime.

- In order to create value from the large amount of collected data, big data techniques and predictive analytics or other certain advanced methods shall be employed in the future. For instance, Data Mining is one analytic process that is designed to explore data in search of consistent patterns and systematic relationships between variables, and then to validate the findings by applying the detected patterns to new subsets of data. The goal of these processes is prediction. Finally, it should be considered that prediction is not only related with battery lifetime, but also with the identification of user's routines and habits.

- This system has a scalable and modular design. Server is composed of different modules: web service, database and dashboard. These modules can be placed on different networked computers and can be replicated in order to build a system that can scale up to meet increased workloads. Fig. 6.1 displays a system where web service is located in three different servers. These servers connect to different database servers which replicate database changes between them. Dashboard has a dedicated web server in order to improve the searches performance. Finally, apps send the data to load balancer, this element distributes the workload of incoming requests onto multiple computers.

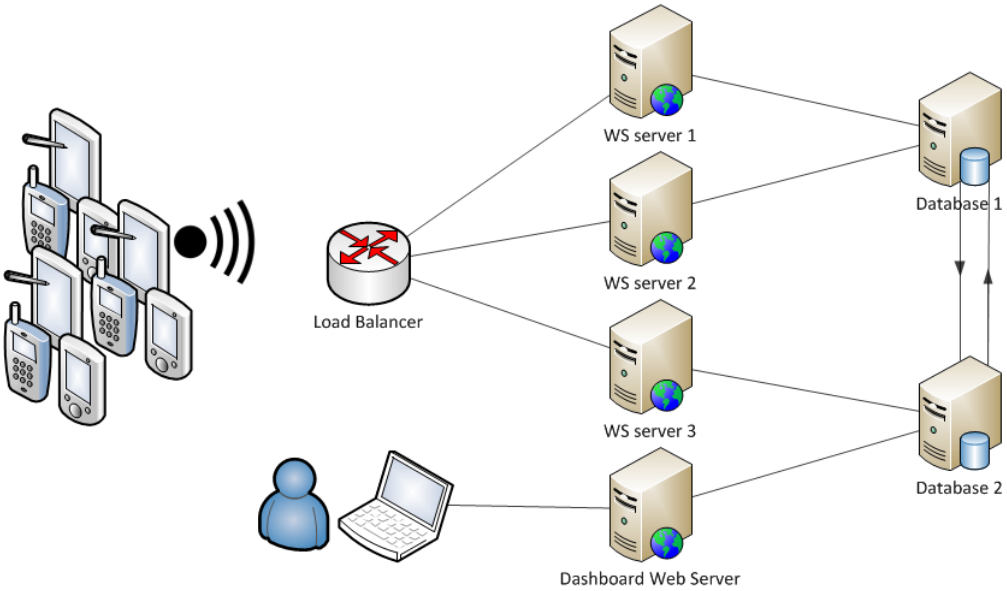


Fig. 6.1 Distributed system

- The system is running for an Android app, nevertheless, system could be adapted in the future to work with another platforms. In that sense, specific apps for other platforms should be created, such as iOS, which will send the data using the same server's web service.

BIBLIOGRAPHY

[1]. What's the Best Battery?. Retrieved from: http://batteryuniversity.com/learn/article/whats_the_best_battery

[2]. MAX17043 datasheet. Retrieved from: http://datasheets.maximintegrated.com/en/ds/MAX17043-MAX17044.pdf

[3]. MAX17047 datasheet. Retrieved from: http://datasheets.maximintegrated.com/en/ds/MAX17047-MAX17050.pdf

[4]. John Chiasson; Baskar Vairamohan. "Estimating the State of Charge of a Battery". Electrical and Computer Engineering Department. The Universitiy of Tennessee, Knoxville.

[5]. Basics About Discharging. Retrieved from: http://batteryuniversity.com/learn/article/discharge_methods

[6]. Why lithium-ion?. Retrieved from: https://www.apple.com/batteries/why-lithium-ion/

[7]. Battery Life (and Death). Retrieved from: http://www.mpoweruk.com/life.htm#dod

[8]. How to Prolong Lithium-based Batteries. Retrieved from: http://batteryuniversity.com/learn/article/how_to_prolong_lithium_based_batteries

[9]. Smartphone OS Market Share. Retrieved from: http://www.idc.com/prodserv/smartphone-os-market-share.jsp

[10]. Android SDK. Retrieved from: https://developer.android.com/sdk/index.html

[11]. Android SDK Manager. Retrieved from: https://developer.android.com/tools/help/sdk-manager.html

[12]. ADT. Retrieved from: https://developer.android.com/tools/help/adt.html

[13]. GIT. Retrieved from: http://git-scm.com/

[14]. BatteryManager. Retrieved from: http://developer.android.com/reference/android/os/BatteryManager.html

[15]. Supporting Multiple Screens. Retrieved from: http://developer.android.com/guide/practices/screens_support.html

[16]. Cuadras Tomàs, Àngel; Ovejas Benedicto, Victòria Júlia; Quílez Figuerola, Marcos. "Método para determiner la degradación de sistemas

con efecto Joule a partir del increment de entropía". Universitat Politècnica de Catalunya.

[17].Suleiman Abu-Sharkh; Dennis Doerffel. "Rapid test and non-linear model characterization of solid-state lithium-ion batteries". School of Engineering Science, University of Southampton, Highfield, Southampton.

[18].Signing Your Applications from Eclipse with ADThttps://developer.android.com/tools/publishing/app-signing-eclipse.html

ANNEX 1. FUEL GAUGE SPECIFICATIONS

Model	*Nexus 5*
Fuel Gauge	Maxim MAX17048 fuel gauge (ModelGauge™, no coulomb counter)
Properties	BATTERY_PROPERTY_CAPACITY
Measurements	The fuel gauge does not support any measurements other than battery State Of Charge to a resolution of %/256 (1/256th of a percent of full battery capacity).

Model	*Nexus 6*
Fuel Gauge	Maxim MAX17050 fuel gauge (a coulomb counter with Maxim ModelGauge™ adjustments), and a 10mohm current sense resistor.
Properties	BATTERY_PROPERTY_CAPACITY BATTERY_PROPERTY_CURRENT_NOW BATTERY_PROPERTY_CURRENT_AVERAGE BATTERY_PROPERTY_CHARGE_COUNTER BATTERY_PROPERTY_ENERGY_COUNTER
Measurements	CURRENT_NOW resolution 156.25uA, update period is 175.8ms. CURRENT_AVERAGE resolution 156.25uA, update period configurable 0.7s - 6.4h, default 11.25 secs. CHARGE_COUNTER (accumulated current, non-extended precision) resolution is 500uAh (raw coulomb counter read, not adjusted by fuel gauge for coulomb counter offset, plus inputs from the ModelGauge m3 algorithm including empty compensation). CHARGE_COUNTER_EXT (extended precision in kernel) resolution 8nAh. ENERGY_COUNTER is CHARGE_COUNTER_EXT at nominal voltage of 3.7V.

Model	*Nexus 9*
Fuel Gauge	Maxim MAX17050 fuel gauge (a coulomb counter with Maxim ModelGauge™ adjustments), and a 10mohm current sense resistor.
Properties	BATTERY_PROPERTY_CAPACITY BATTERY_PROPERTY_CURRENT_NOW BATTERY_PROPERTY_CURRENT_AVERAGE BATTERY_PROPERTY_CHARGE_COUNTER BATTERY_PROPERTY_ENERGY_COUNTER
Measurements	CURRENT_NOW resolution 156.25uA, update period is 175.8ms. CURRENT_AVERAGE resolution 156.25uA, update period configurable 0.7s - 6.4h, default 11.25 secs. CHARGE_COUNTER (accumulated current, non-extended precision) resolution is 500uAh.

	CHARGE_COUNTER_EXT (extended precision in kernel) resolution 8nAh. ENERGY_COUNTER is CHARGE_COUNTER_EXT at nominal voltage of 3.7V. Accumulated current update period 175.8ms. ADC sampled at 175ms quantization with a 4ms sample period. Can adjust duty cycle.

Model	*Nexus 10*
Fuel Gauge	Dallas Semiconductor DS2784 fuel gauge (a coulomb counter), with a 10mohm current sense resistor.
Properties	BATTERY_PROPERTY_CAPACITY BATTERY_PROPERTY_CURRENT_NOW BATTERY_PROPERTY_CURRENT_AVERAGE BATTERY_PROPERTY_CHARGE_COUNTER BATTERY_PROPERTY_ENERGY_COUNTER
Measurements	Current measurement (instantaneous and average) resolution is 156.3uA. CURRENT_NOW instantaneous current update period is 3.5 seconds. CURRENT_AVERAGE update period is 28 seconds (not configurable). CHARGE_COUNTER (accumulated current, non-extended precision) resolution is 625uAh. CHARGE_COUNTER_EXT (extended precision in kernel) resolution is 144nAh. ENERGY_COUNTER is CHARGE_COUNTER_EXT at nominal voltage of 3.7V. Update period for all is 3.5 seconds.

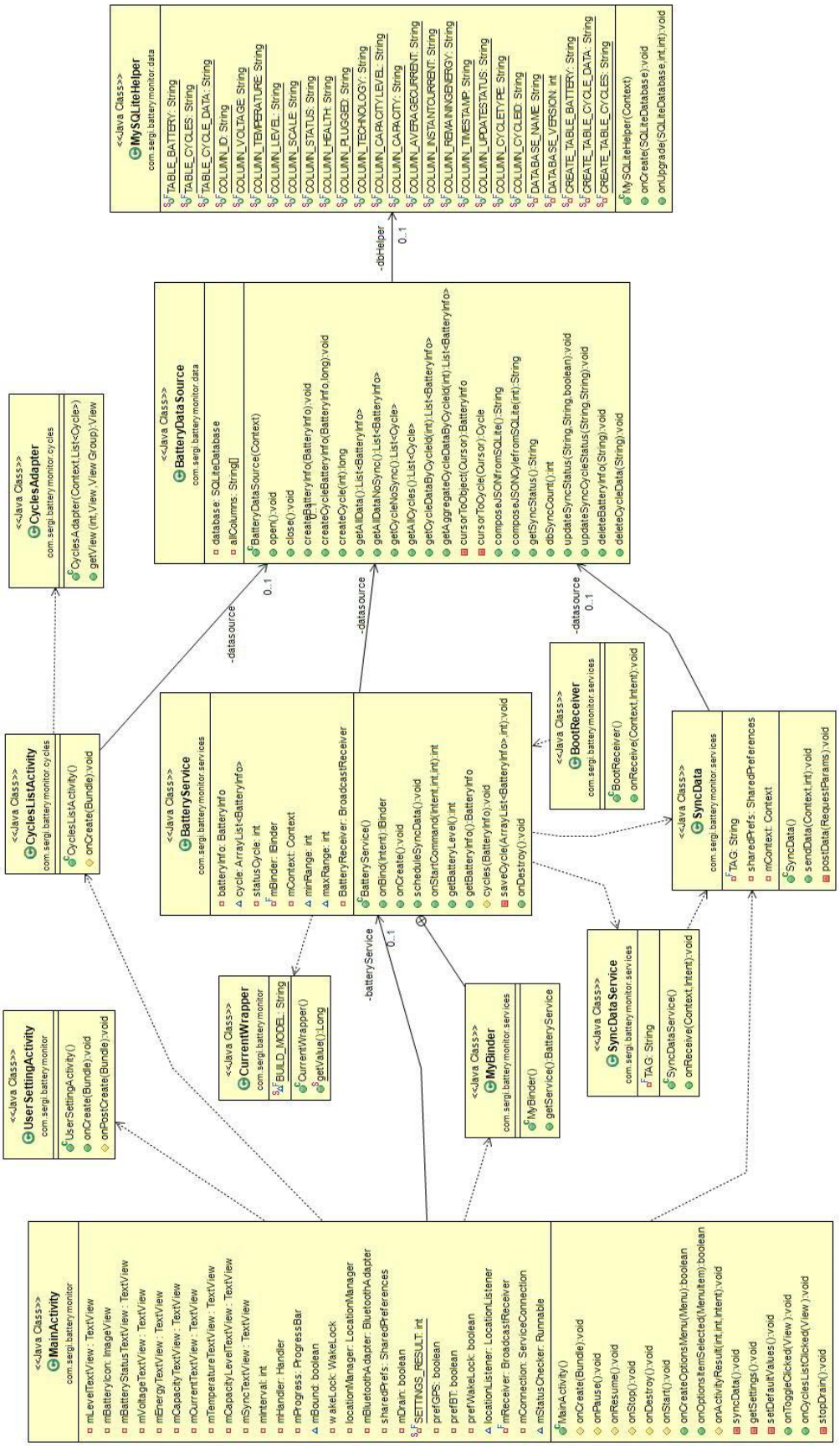
ANNEX 2. OCV MEAN ESTIMATION WORKSHEET

CHARGING				DISCHARGING				MEAN
Date	Level	Voltage	Current	Date	Level	Voltage	Current	OCV
21 mar. 2015 11:55:54	5	3,911	0,816	21 mar. 2015 18:46:53	5	3,553	-0,36	3,7320
21 mar. 2015 11:56:24	6	3,911	0,832	21 mar. 2015 18:44:23	6	3,565	-0,354	3,7380
21 mar. 2015 11:57:24	7	3,911	0,821	21 mar. 2015 18:41:53	7	3,567	-0,331	3,7390
21 mar. 2015 11:58:24	8	3,913	0,837	21 mar. 2015 18:39:23	8	3,572	-0,352	3,7425
21 mar. 2015 11:59:24	9	3,916	0,835	21 mar. 2015 18:37:23	9	3,584	-0,339	3,7500
21 mar. 2015 12:00:24	10	3,920	0,834	21 mar. 2015 18:34:53	10	3,584	-0,319	3,7520
21 mar. 2015 12:01:24	11	3,928	0,832	21 mar. 2015 18:31:53	11	3,597	-0,287	3,7625
21 mar. 2015 12:02:24	12	3,933	0,837	21 mar. 2015 18:29:23	12	3,599	-0,271	3,7660
21 mar. 2015 12:02:54	13	3,935	0,837	21 mar. 2015 18:26:23	13	3,608	-0,283	3,7715
21 mar. 2015 12:03:54	14	3,933	0,819	21 mar. 2015 18:23:53	14	3,619	-0,275	3,7760
21 mar. 2015 12:04:54	15	3,952	0,837	21 mar. 2015 18:21:23	15	3,629	-0,269	3,7905
21 mar. 2015 12:05:54	16	3,961	0,836	21 mar. 2015 18:18:23	16	3,628	-0,314	3,7945
21 mar. 2015 12:06:54	17	3,965	0,834	21 mar. 2015 18:15:53	17	3,636	-0,292	3,8005
21 mar. 2015 12:07:54	18	3,974	0,821	21 mar. 2015 18:12:53	18	3,653	-0,273	3,8135
21 mar. 2015 12:08:54	19	3,979	0,835	21 mar. 2015 18:10:23	19	3,656	-0,28	3,8175
21 mar. 2015 12:09:24	20	3,982	0,822	21 mar. 2015 18:07:53	20	3,662	-0,285	3,8220
21 mar. 2015 12:10:24	21	3,987	0,824	21 mar. 2015 18:04:53	21	3,662	-0,321	3,8245
21 mar. 2015 12:11:24	22	3,989	0,817	21 mar. 2015 18:02:23	22	3,678	-0,325	3,8335
21 mar. 2015 12:12:24	23	3,992	0,821	21 mar. 2015 17:59:53	23	3,675	-0,3	3,8335
21 mar. 2015 12:13:24	24	3,994	0,824	21 mar. 2015 17:56:53	24	3,670	-0,273	3,8320
21 mar. 2015 12:14:24	25	3,994	0,837	21 mar. 2015 17:54:23	25	3,668	-0,281	3,8310
21 mar. 2015 12:15:24	26	3,996	0,822	21 mar. 2015 17:51:23	26	3,685	-0,279	3,8405
21 mar. 2015 12:15:54	27	3,996	0,821	21 mar. 2015 17:48:53	27	3,687	-0,274	3,8415
21 mar. 2015 12:16:54	28	3,999	0,838	21 mar. 2015 17:45:53	28	3,692	-0,282	3,8455
21 mar. 2015 12:17:54	29	3,989	0,837	21 mar. 2015 17:43:23	29	3,678	-0,294	3,8335
21 mar. 2015 12:18:54	30	4,001	0,838	21 mar. 2015 17:40:53	30	3,687	-0,273	3,8440
21 mar. 2015 12:19:54	31	4,001	0,819	21 mar. 2015 17:37:53	31	3,695	-0,264	3,8480
21 mar. 2015 12:20:54	32	4,004	0,837	21 mar. 2015 17:35:23	32	3,700	-0,278	3,8520
21 mar. 2015 12:21:54	33	4,006	0,837	21 mar. 2015 17:32:23	33	3,683	-0,33	3,8445
21 mar. 2015 12:22:24	34	4,006	0,838	21 mar. 2015 17:29:53	34	3,705	-0,354	3,8555
21 mar. 2015 12:23:24	35	4,009	0,834	21 mar. 2015 17:26:53	35	3,703	-0,272	3,8560
21 mar. 2015 12:24:24	36	4,011	0,83	21 mar. 2015 17:24:23	36	3,707	-0,269	3,8590
21 mar. 2015 12:25:24	37	4,011	0,838	21 mar. 2015 17:21:23	37	3,707	-0,291	3,8590
21 mar. 2015 12:26:24	38	4,012	0,831	21 mar. 2015 17:18:53	38	3,705	-0,273	3,8585
21 mar. 2015 12:27:24	39	4,016	0,812	21 mar. 2015 17:15:53	39	3,710	-0,292	3,8630
21 mar. 2015 12:28:24	40	4,018	0,819	21 mar. 2015 17:13:23	40	3,714	-0,271	3,8660
21 mar. 2015 12:28:54	41	4,021	0,825	21 mar. 2015 17:10:53	41	3,712	-0,331	3,8665
21 mar. 2015 12:29:54	42	4,021	0,822	21 mar. 2015 17:07:53	42	3,714	-0,302	3,8675
21 mar. 2015 12:30:54	43	4,023	0,824	21 mar. 2015 17:05:23	43	3,719	-0,305	3,8710
21 mar. 2015 12:31:54	44	4,026	0,824	21 mar. 2015 17:02:53	44	3,719	-0,302	3,8725

21 mar. 2015 12:32:54	45	4,021	0,822	21 mar. 2015 17:00:23	45	3,722	-0,312	3,8715
21 mar. 2015 12:33:54	46	4,028	0,834	21 mar. 2015 16:57:53	46	3,724	-0,304	3,8760
21 mar. 2015 12:34:54	47	4,036	0,832	21 mar. 2015 16:55:23	47	3,724	-0,317	3,8800
21 mar. 2015 12:35:24	48	4,026	0,812	21 mar. 2015 16:52:53	48	3,732	-0,307	3,8790
21 mar. 2015 12:36:24	49	4,041	0,827	21 mar. 2015 16:50:23	49	3,737	-0,305	3,8890
21 mar. 2015 12:37:24	50	4,044	0,827	18 mar. 2015 16:49:23	50	3,739	-0,3	3,8915
21 mar. 2015 12:38:24	51	4,047	0,835	21 mar. 2015 16:47:53	51	3,741	-0,321	3,8940
21 mar. 2015 12:39:24	52	4,050	0,827	21 mar. 2015 16:45:23	52	3,751	-0,279	3,9005
21 mar. 2015 12:40:24	53	4,055	0,832	21 mar. 2015 16:42:23	53	3,754	-0,277	3,9045
21 mar. 2015 12:41:24	54	4,058	0,838	21 mar. 2015 16:39:23	54	3,751	-0,282	3,9045
21 mar. 2015 12:42:24	55	4,060	0,838	21 mar. 2015 16:36:53	55	3,766	-0,313	3,9130
21 mar. 2015 12:42:54	56	4,061	0,826	21 mar. 2015 16:34:23	56	3,763	-0,305	3,9120
21 mar. 2015 12:43:54	57	4,063	0,835	21 mar. 2015 16:31:53	57	3,768	-0,315	3,9155
21 mar. 2015 12:44:54	58	4,070	0,825	21 mar. 2015 16:29:23	58	3,773	-0,33	3,9215
21 mar. 2015 12:45:54	59	4,075	0,827	21 mar. 2015 16:26:53	59	3,768	-0,326	3,9215
21 mar. 2015 12:46:54	60	4,077	0,838	21 mar. 2015 16:24:23	60	3,778	-0,344	3,9275
21 mar. 2015 12:47:54	61	4,082	0,834	21 mar. 2015 16:22:23	61	3,788	-0,339	3,9350
21 mar. 2015 12:48:54	62	4,087	0,838	21 mar. 2015 16:19:53	62	3,793	-0,33	3,9400
21 mar. 2015 12:49:24	63	4,090	0,838	21 mar. 2015 16:17:53	63	3,790	-0,336	3,9400
21 mar. 2015 12:50:24	64	4,095	0,838	21 mar. 2015 16:15:23	64	3,805	-0,356	3,9500
21 mar. 2015 12:51:24	65	4,099	0,822	21 mar. 2015 16:13:23	65	3,805	-0,321	3,9520
21 mar. 2015 12:52:24	66	4,104	0,821	21 mar. 2015 16:10:53	66	3,815	-0,333	3,9595
21 mar. 2015 12:53:24	67	4,109	0,825	21 mar. 2015 16:08:23	67	3,827	-0,339	3,9680
21 mar. 2015 12:54:24	68	4,114	0,829	21 mar. 2015 16:06:23	68	3,830	-0,32	3,9720
21 mar. 2015 12:55:24	69	4,121	0,838	21 mar. 2015 16:03:53	69	3,835	-0,333	3,9780
21 mar. 2015 12:55:54	70	4,124	0,832	21 mar. 2015 16:01:23	70	3,844	-0,338	3,9840
21 mar. 2015 12:56:54	71	4,131	0,829	21 mar. 2015 15:59:23	71	3,844	-0,33	3,9875
21 mar. 2015 12:57:54	72	4,136	0,829	21 mar. 2015 15:56:53	72	3,849	-0,33	3,9925
21 mar. 2015 12:58:54	73	4,144	0,827	21 mar. 2015 15:54:53	73	3,852	-0,33	3,9980
21 mar. 2015 12:59:54	74	4,151	0,837	21 mar. 2015 15:52:23	74	3,866	-0,32	4,0085
21 mar. 2015 13:00:54	75	4,158	0,845	21 mar. 2015 15:49:53	75	3,870	-0,325	4,0140
21 mar. 2015 13:01:54	76	4,156	0,83	21 mar. 2015 15:47:53	76	3,881	-0,334	4,0185
21 mar. 2015 13:02:24	77	4,168	0,83	21 mar. 2015 15:45:23	77	3,891	-0,32	4,0295
21 mar. 2015 13:03:24	78	4,175	0,822	21 mar. 2015 15:42:53	78	3,891	-0,346	4,0330
21 mar. 2015 13:04:24	79	4,185	0,842	21 mar. 2015 15:40:53	79	3,898	-0,331	4,0415
21 mar. 2015 13:05:24	80	4,180	0,783	21 mar. 2015 15:38:23	80	3,908	-0,317	4,0440
21 mar. 2015 13:06:24	81	4,200	0,822	21 mar. 2015 15:35:53	81	3,909	-0,346	4,0545
21 mar. 2015 13:07:24	82	4,200	0,781	21 mar. 2015 15:33:53	82	3,917	-0,356	4,0585
21 mar. 2015 13:08:24	83	4,200	0,736	21 mar. 2015 15:31:23	83	3,927	-0,328	4,0635
21 mar. 2015 13:09:24	84	4,200	0,71	21 mar. 2015 15:28:53	84	3,935	-0,323	4,0675
21 mar. 2015 13:10:24	85	4,200	0,676	21 mar. 2015 15:26:53	85	3,938	-0,33	4,0690
21 mar. 2015 13:11:24	86	4,200	0,626	21 mar. 2015 15:24:23	86	3,950	-0,341	4,0750
21 mar. 2015 13:12:54	87	4,200	0,596	21 mar. 2015 15:21:53	87	3,957	-0,323	4,0785
21 mar. 2015 13:13:54	88	4,202	0,567	21 mar. 2015 15:19:53	88	3,968	-0,329	4,0850
21 mar. 2015 13:15:24	89	4,200	0,512	21 mar. 2015 15:17:23	89	3,977	-0,315	4,0885

21 mar. 2015 13:16:54	90	4,200	0,486	21 mar. 2015 15:14:53	90	3,982	-0,328	4,0910
21 mar. 2015 13:18:54	91	4,200	0,438	21 mar. 2015 15:12:23	91	3,999	-0,347	4,0995
21 mar. 2015 13:20:24	92	4,200	0,404	21 mar. 2015 15:10:24	92	3,999	-0,321	4,0995
21 mar. 2015 13:22:24	93	4,200	0,364	21 mar. 2015 15:07:53	93	4,006	-0,312	4,1030
21 mar. 2015 13:24:54	94	4,200	0,321	21 mar. 2015 15:05:23	94	4,016	-0,314	4,1080
21 mar. 2015 13:27:24	95	4,200	0,279	21 mar. 2015 15:02:53	95	4,023	-0,313	4,1115

ANNEX 3. APP DIAGRAM CLASS



ANNEX 4. ANDROID VERSION LEVELS

Platform Version	API Level	VERSION_CODE
Android 5.0	21	LOLLIPOP
Android 4.4W	20	KITKAT_WATCH
Android 4.4	19	KITKAT
Android 4.3	18	JELLY_BEAN_MR2
Android 4.2, 4.2.2	17	JELLY_BEAN_MR1
Android 4.1, 4.1.1	16	JELLY_BEAN
Android 4.0.3, 4.0.4	15	ICE_CREAM_SANDWICH_MR1
Android 4.0, 4.0.1, 4.0.2	14	ICE_CREAM_SANDWICH
Android 3.2	13	HONEYCOMB_MR2
Android 3.1.x	12	HONEYCOMB_MR1
Android 3.0.x	11	HONEYCOMB
Android 2.3.4 Android 2.3.3	10	GINGERBREAD_MR1
Android 2.3.2 Android 2.3.1 Android 2.3	9	GINGERBREAD
Android 2.2.x	8	FROYO
Android 2.1.x	7	ECLAIR_MR1
Android 2.0.1	6	ECLAIR_0_1
Android 2.0	5	ECLAIR
Android 1.6	4	DONUT
Android 1.5	3	CUPCAKE
Android 1.1	2	BASE_1_1
Android 1.0	1	BASE

ANNEX 5. MYSQL DATABASE SCRIPT

```
-- phpMyAdmin SQL Dump
-- version 4.2.11
-- http://www.phpmyadmin.net
--
-- Servidor: 127.0.0.1
-- Tiempo de generaciÃ³n: 15-01-2015 a las 08:39:37
-- VersiÃ³n del servidor: 5.6.21
-- VersiÃ³n de PHP: 5.6.3

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET time_zone = "+00:00";


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;

--
-- Base de datos: `battery_db`
--
CREATE DATABASE IF NOT EXISTS `battery_db` DEFAULT CHARACTER SET
latin1 COLLATE latin1_swedish_ci;
USE `battery_db`;

-- --------------------------------------------------------

--
-- Table structure for table `battery`
--

CREATE TABLE IF NOT EXISTS `battery` (
  `_id` int(11) NOT NULL,
  `device_id` int(11) NOT NULL,
  `voltage` int(11) NOT NULL,
  `temperature` int(11) NOT NULL,
  `level` int(11) NOT NULL,
  `scale` int(11) NOT NULL,
  `status` int(11) NOT NULL,
  `health` int(11) NOT NULL,
  `plugged` int(11) NOT NULL,
  `technology` varchar(255) NOT NULL,
  `capacityLevel` int(11) NOT NULL,
  `capacity` int(11) NOT NULL,
  `averageCurrent` int(11) NOT NULL,
  `instantCurrent` int(11) NOT NULL,
  `remainingEnergy` int(11) NOT NULL,
  `entropy` decimal(15,15) NOT NULL,
  `entropy2` decimal(15,15) NOT NULL,
  `integralEntropy` decimal(15,15) NOT NULL,
  `integralEntropy2` decimal(15,15) NOT NULL,
  `timestamp` datetime NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

-- --------------------------------------------------------

--
-- Table structure for table `cycles`
```

```
--

CREATE TABLE IF NOT EXISTS `cycles` (
  `_id` int(11) NOT NULL AUTO_INCREMENT,
  `device_id` int(11) NOT NULL,
  `type` int(11) NOT NULL,
  `timestamp` datetime NOT NULL,
  PRIMARY KEY (`_id`)
) ENGINE=MyISAM  DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci;

-- --------------------------------------------------------

--
-- Table structure for table `cycles_data`
--

CREATE TABLE IF NOT EXISTS `cycles_data` (
  `_id` int(11) NOT NULL,
  `cycle_id` int(11) NOT NULL,
  `voltage` int(11) NOT NULL,
  `temperature` int(11) NOT NULL,
  `level` int(11) NOT NULL,
  `scale` int(11) NOT NULL,
  `status` int(11) NOT NULL,
  `health` int(11) NOT NULL,
  `plugged` int(11) NOT NULL,
  `technology` varchar(255) COLLATE latin1_general_ci NOT NULL,
  `capacityLevel` int(11) NOT NULL,
  `capacity` int(11) NOT NULL,
  `averageCurrent` int(11) NOT NULL,
  `instantCurrent` int(11) NOT NULL,
  `remainingEnergy` int(11) NOT NULL,
  `timestamp` datetime NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci;

-- --------------------------------------------------------

--
-- Table structure for table `devices`
--

CREATE TABLE IF NOT EXISTS `devices` (
  `_id` int(11) NOT NULL AUTO_INCREMENT,
  `deviceInfo` varchar(255) DEFAULT NULL,
  `deviceId` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`_id`)
) ENGINE=MyISAM  DEFAULT CHARSET=latin1;

-- --------------------------------------------------------

--
-- Table structure for table `voc`
--

CREATE TABLE IF NOT EXISTS `voc` (
  `_id` int(11) NOT NULL AUTO_INCREMENT,
  `device_id` int(11) NOT NULL,
  `level` int(11) NOT NULL,
  `voltage` int(11) NOT NULL,
  PRIMARY KEY (`_id`)
) ENGINE=MyISAM  DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci;
```

```
-- --------------------------------------------------------

--
-- Table structure for table `voc_regression`
--

CREATE TABLE IF NOT EXISTS `voc_regression` (
  `_id` int(11) NOT NULL AUTO_INCREMENT,
  `device_id` int(11) NOT NULL,
  `level` int(11) NOT NULL,
  `status` int(11) NOT NULL,
  `slope` decimal(10,10) NOT NULL,
  `intercept` float NOT NULL,
  `correlation` decimal(10,10) NOT NULL,
  PRIMARY KEY (`_id`)
) ENGINE=MyISAM  DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci;

-- --------------------------------------------------------

--
-- Table structure for table `users`
--
CREATE TABLE IF NOT EXISTS `users` (

`_id` int(11) NOT NULL AUTO_INCREMENT,
`username` varchar(20) NOT NULL,

`password` varchar(20) NOT NULL,
PRIMARY KEY (`_id`)
) ENGINE=MyISAM  DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci;
```