University of Strathclyde Engineering

Computer and Electronic Systems
Fourth Year Individual Project

Sustainable Energy for Development

The Gambia Solar Project

Remote data acquisition, communications and analysis for the utilisation and condition assessment of battery charging stations in developing countries.

Daniel Chakraverty
200906954
Dr Scott Strachan
April 2013

Declaration

I hereby declare that this work has not been submitted for any other degree/course at this University or any other institution and that, except where reference is made to the work of other authors, the material presented is original and entirely the result of my own work at the University of Strathclyde under the supervision of Dr Scott Strachan.

_____ Date: _____

Abstract

Remote condition monitoring has the potential to improve the lifespan of solar installation assets in developing countries. The lifespan of these solar installations is much lower than the design life due to their situation – systems can't be easily maintained, the lack of technical experience of users and the harsh environment conditions. Using a system such as the one proposed allows limited maintenance services to be directed to installations in need, asset life to be prolonged and data to be gathered that is essential for future designs.

This project develops a low cost remote data acquisition unit for a battery charging station in rural villages, which is capable of uploading data to a server situated in the United Kingdom over GSM/GPRS mobile internet networks. This server is capable of monitoring the batteries condition, and assisting the charging station operator in the management of their assets.

A web application has been developed to display information in a clear and user-friendly manner. An Android application has also been implemented with capability to scan QR codes on batteries and display relevant information to the user.

Acknowledgements

I would like to express my thanks to Dr Scott Strachan, Dr Michael Dolan, Puran Rakhra and Dr Tom Houghton for their advice, support and time during the research and development of this project.

I would also like to thank the Vertically Integrated Project on Sustainable Energy for Development for allowing me to contribute to their meetings and objectives, in particular David Smith and Jennifer Miller whose ideas and prototypes have supported the design of this project.

Finally I would like to thank Isaac Holeman for sharing his experience of using mobile networks in developing countries.

Contents

Table of Figures

1 Introduction

Solar energy is widely regarded as the best solution for generating electricity in many developing countries such as The Gambia, West Africa. However, implementing solutions that are sustainable i.e. developing solutions that can meet the needs of people not only in the present but also for future generations has been shown to be difficult, with many existing projects failing before the end of their design life. For example, only 38% of photovoltaic (PV) systems installed in Tunisia were fully operational after 7 years, with a UK system life expected to be around 30 years [1].

There are many projects that have involved installing solar systems in developing countries which have documented good results in the short term, but there is lack of understanding as to why the systems have failed, or even their condition just five or ten years after installation. As system designers, understanding the operation and usage of the systems in this environment is key to developing systems that are sustainable. Remote condition monitoring systems can facilitate the gathering of this information.

2 Project Objectives

This project will design and develop a low cost, low power prototype remote condition monitoring system to acquire battery condition and system usage from a battery charging station. It is required to be standalone (not require any local PC) and durable enough to withstand the extreme environment in which it will be situated.

A web application is to be developed to display data in a coherent and relevant manner for system operators/developers and offer decision support in the management of the charging station in order to optimize the station's design and lifetime, and minimize any maintenance requirements.

In addition, a smartphone app is to be developed to notify users of the condition of their battery, helping to engage users in the care and maintenance of the installation.

Further objectives include integrating the remote condition monitoring system with the existing prototype of the battery charging station developed as part of the Vertically Integrated Project on 'Sustainable Energy for Development'.

3 Project Motivation

Remote condition monitoring has many applications, not just in developing countries. As power generation becomes more decentralised, moving from large coal or nuclear power

stations to small pockets of renewable generation, there is a requirement for understanding what power is being generated, and where it is available.

A low cost, low power system that is capable of uploading data directly to the internet allows large amounts of data to be collected and analysed, giving the ability to measure spot and regional performance, which is essential for the modern power grid to become 'smarter'. As more consumers fit micro-generation to their houses, it is necessary to be able to monitor the behaviour of the network at a much lower level than has been previously carried out.

Most monitoring is currently carried out at a secondary substation level, which is not close enough to the consumer to understand the complex interactions between generation and usage within neighbourhoods. A system such as the one proposed could allow this data to become available, and the network managed in a much more efficient manner.

3.1.1 Solar System Monitoring

In developed countries PV installations are typically used in conjunction with a traditional grid connection. In the UK, feed-in tariffs have been introduced to incentivise the installation of these systems, where bills are reduced in return for generating renewable energy [2]. The combination of two power sources, and the extra income for power generated (even if this power is not consumed) often makes understanding the operation of the system difficult for the consumer.

This has led to the development of solar monitoring systems by manufacturers and third parties where, for an extra fee, a system will be installed that analyses the generation of power usage, and presents the data in an easily understandable format for the user. This data is usually presented on a small screen on a device in the home, or remotely via a web application. However many of these systems are expensive, and use proprietary technology [3] [4], with the cost of use being much higher than the cost of basic communication.

In developing countries, the lack of infrastructure provides issues with installing such systems, as most existing products use a hardwired network connection to pass data to a web application. In addition, cost is major issue with installing sustainable solutions. People in remote areas have no such infrastructure, earn very little financially, and struggle to afford the large capital cost associated with solar systems. Adding the extra cost of a remote monitoring system is not financially viable if not low and justified to the consumer.

These issues mean that any usage data gained is usually anecdotal, and relies on people without technical knowledge to log and report technical usage information back to system designers – who usually are situated in a different country. So any new developed solution often has the

same shortcomings as a system that has previously failed, due to a lack of understanding of the environment and usage, and fails in a similar way.

3.1.2 Accessing data from remote areas

Recent (2012) experiences in The Gambia showed the availability of GSM mobile networks in very remote areas without a grid connection. This is due to the differences in technology, with radio waves reaching much further with one base station. Many projects have taken advantage of this connectivity, using SMS text messages to relay data to a local base station, which then has the capability to upload the data to a web server – accessible across the world.

These solutions are not ideal due to the requirement of a local base station to receive the SMS messages and upload them to a web server. This avoids the cost of round the world text messages, but does add a cost of running a local base station in a larger town or city with a reliable power supply. Many projects are charitable or non-profit, and therefore cannot afford or don't have access to this type of facility.

As mobile technology improves, there is the opportunity to remove the need for a local base station, and upload data directly from a PV installation to a remote web server through mobile internet. This is possible over the 2G networks available, albeit at a lower speed and bandwidth than is used in modern systems. The amount of data that could be gathered and sent is relatively small compared to media streaming for example, so there is no need for high speed and bandwidth 3G or 4G networks. Removing the cost of running and maintaining a local base station means that the overall cost of remote condition monitoring can be lowered, making it more viable to developers and users, as it has been shown that cost is major factor in the success of these projects.

3.1.3 Human Centred Design

It is important to engage the end user in the design of solar systems. Many solar systems fail once out in the field due to the user having a lack of technical understanding of the system. These systems are installed without consulting the end user, and then not maintained once installed, with all the responsibilities placed on the end user. Having remote communication with the solar system could allow the designer to remotely manage the systems usage, and help the end user learn to maintain and prolong the system's lifespan.

The designer could feasibly show certain data to the end user, without overwhelming them with technical jargon and numbers, and then gradually increase the information shown as their technical understanding increases. This gives the technical support that many of the end users need with a new solar system, and would assist in its longevity, as it can be used in the way that it was designed.

This is 'Human Centred Design' [5], where the end user is consulted and becomes a part of the design and operation of the system. The user can be much more engaged in the development of a solution, allowing them to understand the possible benefits of a solar installation, and a remote monitoring system. It addresses the issue of the cost of remote condition monitoring being prohibitive to a user, if they are shown the benefits of such a system.

3.1.4 Improving limited maintenance facilities

In a system where support is offered to the end user by visits from technically proficient people who carry out maintenance and repairs on the installation, time can be wasted travelling to a system that has no problems. Having data available on a system's condition remotely allows a technician to only spend time travelling to a system that has a problem, saving them money, and decreasing any downtime a user has.

In areas where installed systems are many kilometres apart, and very few maintenance visits are made, this can save the operator travel time and costs, and also improve the user experience.

3.1.5 Optimising asset lifetime

Battery Charging Stations are a potential sustainable solution to meet the basic energy requirements of lighting and phone charging to rural communities who cannot afford a full solar home system. Through a central charging station at a local shop, users can exchange depleted batteries for a fully charged one for a small fee, providing a cheap, healthy alternative to kerosene lamps or candles.

In a traditional solar installation, the most expensive component across the system's lifespan is the batteries. With a charging station, battery lifetime is therefore an even greater issue. Using asset monitoring techniques, it is possible to assist the charging station operator (who may not be technically proficient) in the operation of their system to improve its lifespan.

Calculating a batteries state of health based on its use, manufacturer recommendations and internal condition measurements allows battery overuse to be prevented. Battery usage can therefore be assessed and managed to allow an even state of wear across the asset base. This could prevent the situation where the operator overuses some batteries, which then wear out and cannot be used to serve customers – impacting their income, and end user experience.

As the operator becomes more educated in the technical workings of the charging station, more control can be handed over to them. Integrating this asset optimisation with remote condition monitoring allows system designers and developers to have more access to the asset deterioration data, meaning they can design better systems in future, and assist the operator in the system's operation while not physically at the installation.

4 Background

In order to develop a successful design, it is necessary to understand in detail the environment in which the system will be situated – from a technical, social, and financial point of view in order to set a specification to which the system will have to adhere.

It is also important to understand the technologies that are currently available, as there is no need to spend time developing brand new technologies if there are tools already available. Doing so would slow development to the point that objectives could not be achieved within the time span of the project.

In addition, aligning the project to interface with other projects within the university facilitates easy integration with other solutions currently being developed. It also gives opportunities for the prototype system to be tested in the field, and actually be used in the environment for which it was designed.

4.1 Solar System Configurations

There are five main categories of solar installation, ranging from small solar lanterns to large micro-grid installations. This project mainly focuses on battery charging stations, as this configuration has been shown in many areas to give the best sustainable solution for people in developing countries with a low and unpredictable income. The main financial model usually associated with battery charging stations is the fee-for-service model, where a user is charged for the supply of electricity, and it is supplied and maintained by an electricity operator. Appendix 11.1 discusses different financial models in more detail, and their suitability for different solar system configurations.

4.1.1 Pico PV

These are small PV systems that have a power output of 1 to 10W, mainly used for lighting. Often seen as handheld solar lanterns, they often have the solar panel built into the device and replace inefficient light sources such as candles or kerosene lamps.

4.1.2 Solar Home Systems

Solar Home Systems are larger than Pico PV lanterns, offering a power output of up 200W.

They generally constitute a layout as seen in Figure 1, with one solar panel charging a battery, which can then power loads such as lighting, radios, DC televisions etc. Usually a charge controller is incorporated to protect the battery from over discharging and charging.

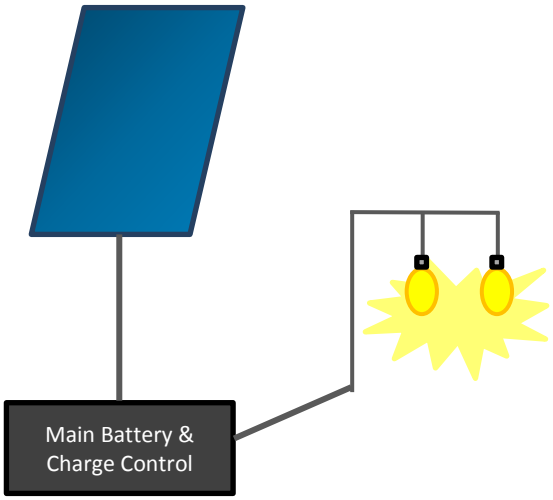


Figure 1 | Layout of a traditional Solar Home System

4.1.3 Solar Residential Systems

Solar Residential Systems are similar to Solar Home Systems in terms of layout, as shown in Figure 2; however they are generally larger, in the order of 1000W to 4000W. Multiple solar panels are usually present, used to feed multiple batteries and loads.

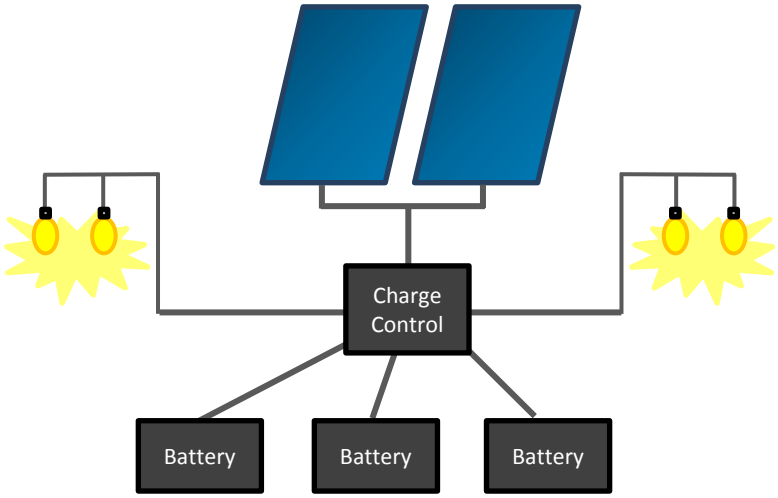


Figure 2 | Layout of a traditional Solar Residential System

Often an inverter is used to power AC loads. Because of the larger size and therefore cost these systems are usually present at schools or health clinics in developing countries.

This is the type of system the Gambia Solar Project introduced in section 4.4 normally installs.

4.1.4 Battery Charging Stations

A large central station, with solar panels, charge control electronics and batteries is used to charge small portable batteries that users can take home to power smaller loads like lighting or mobile phone charging. The standard layout for a charging station is shown in Figure 3.

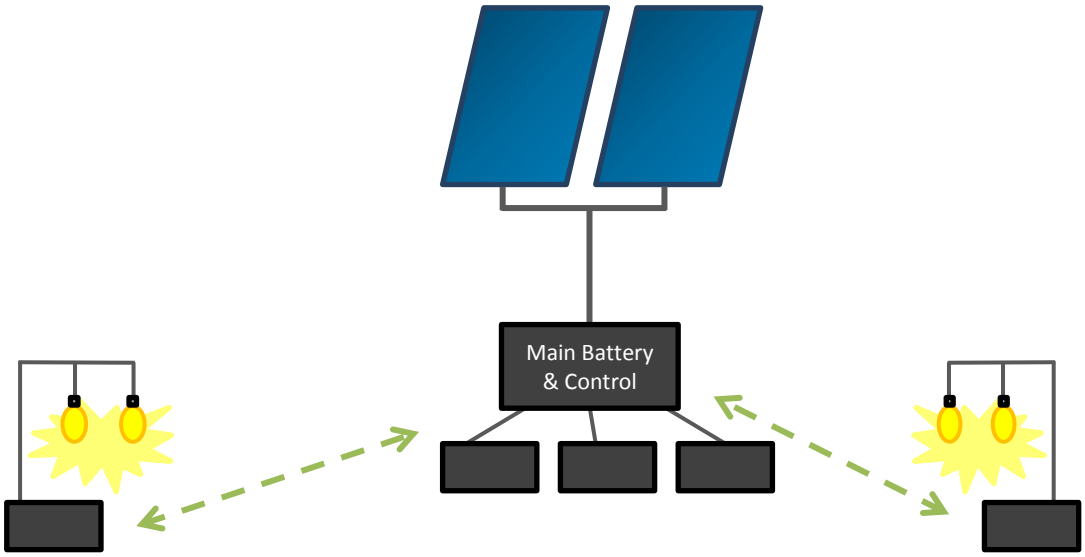


Figure 3 | Layout of a standard Battery Charging Station

The flexibility offered to the end user with battery charging stations is particularly suited to the unpredictable financial incomes of farmers in rural areas of developing countries.

4.1.5 Micro-grids

A micro-grid uses small pockets of distributed generation to supply customers who are hardwired to each other and to the generation. This usually gives a very good security of supply and often a lower cost over a number of years. However the large capital cost incurred with the initial construction of the grid and generation puts this configuration outside the financial means of most rural villages in developing countries.

If the micro-grid is run by a local electricity supply non-governmental organisation (NGO), having the means to monitor the condition and operation of the system would be extremely useful.

Even though this project focuses on battery charging stations, it could be perceived that the system proposed has applications across most of the configurations along with other areas of the engineering industry.

4.2 Review of Existing Condition Monitoring Systems

Mobile technology is advancing at a very high rate, and with this designs are possible that perhaps were not feasible two or even three years ago. Because of this it is necessary to develop

using technologies that will be available after development and testing time in a project. Developing using technologies that are currently available means that once a product is ready for deployment it is already outdated.

In most areas, this type of development is not possible, however as developing countries are often two or three generations behind the cutting edge in technology, it is possible to develop using technologies that are slightly ahead of what is currently available – with the understanding that they will become available in the near future.

This project uses 2G mobile networks to communicate directly to a web server, many other projects that are available to research use a local base station to receive data and update a web application. Using the proposed approach simplifies the hardware required in country.

Presented here is a critical review of some existing projects attempting to use mobile communications to assist in the deployment of new technologies in developing countries.

4.2.1 Medic Mobile

Medic Mobile is an organisation that is currently using SMS messages to assist healthcare in very remote areas [6]. Hospitals and clinics are generally located in larger towns and cities, meaning that patients and health workers need to travel large distances (over 10km) for appointments and treatment. Using open source tools such as Frontline SMS [7] and SMSsync [8], Medic Mobile have developed systems that allow for information to be sent over the mobile networks, and health workers to report on patients without travelling back to the clinic. This has proved to be successful in pilot projects in Malawi [9].

Part of Medic Mobile's methodology includes using open source and existing technologies where available to develop their projects. SIM apps have been used on mobile phones to facilitate cross platform development. SIM apps will run on any phone, from old 'dumb-phones' to top of the range smart-phones, meaning that the functionality can reach many more users than if the system ran on one specific piece of hardware.

As technology advances, the functionality offered by modern smartphones becomes more necessary, and SIM apps become more obsolete. However developing for specific hardware and operating systems is very complex, expensive and time consuming. The mobile operating system Android has many different versions of its Software Development Kit (SDK) for different devices, and similar experiences are had with iOS, Windows Phone, Symbian and Blackberry. Instead of developing for native platforms, there is the option of developing browser based applications, as most modern phones will include a browser.

4.2.2 SIMbaLink

This project has identified the issue of long-term maintenance and monitoring being critical to the longevity of a solar installation [10]. In the challenging environment of developing countries such as Uganda, Kenya and Tanzania the high cost of travelling in-person maintenance to remote areas means that most installations are not maintained. This has led to the conclusion that this is one of the largest barriers to the widespread adoption of solar power in sub-Saharan Africa [11].

The system developed is aimed at local solar installers who send technicians out to monitor and maintain the PV installations. GSM modems are connected to small low power microcontrollers, and text messaging is used to send information back to the headquarters about the status of the system. This means that a technician only has to travel to a system that they know for sure is in need of maintenance.

Once SMS text messages reach the base station, they are decoded and uploaded to a web server containing a database. Here the data is analysed and a diagnosis of the system's state of health is made. The results are viewable using a web application.

In a project where customers are charged on a fee-for-service basis, the remote communications could be used to monitor the energy drawn by the solar product, allowing the operator to charge the customer accordingly [10].

SIMbaLink has been prototyped and tested in Brooklyn, New York with positive results. Investigations have been carried out into how the SMS messaging system can be exploited to get as much data into one message as possible, along with what data indicates an installation in need of maintenance.

4.2.3 An Open Source Monitoring System for Remote Solar Power Applications

Using only open-source tools, this project at the University of Florida had the objectives of showing that solar remote monitoring technology is neither expensive nor difficult to implement [12]. A low power microcontroller is capable of relaying information from a solar home system to a web server to present historical data.

During the project, a remote communications system was implemented to send text messages using a microcontroller and a basic mobile phone. The microcontroller communicates with the phone by electronically 'pressing' the buttons on the phone. Instead of using SMS text messages, the microcontroller makes a phone call, and messages are relayed by pressing buttons during the call resulting in tones heard at the receiver. The receiver then decodes these tones back to data which can then be uploaded.

The remote server uses Voice over IP (VoIP) services to call the receiver, so no local base station is required. The justification for this is that in a system where the remote system updates the server, the server must be contactable remotely (i.e. have a phone number). In 2009, when the project was completed, this was not as feasible without the extra hardware at a local base station, whereas modern socket programming allows this to be achieved fairly easily.

All decisions have been made in order to make the system as cheap as possible, along with using components and designs that are freely available. While this approach results in an easily replicated prototype in developing countries, as components can be sourced as required, it means that technologies are perhaps not used to their full potential.

For example using a mobile phone (with a modem integrated) presents many limitations that would not be present if the modem was connected directly to the microcontroller. However it does mean that the system has an abstraction layer that makes it very simple to understand and replicated. In systems that are often using very complex circuitry for charge controllers anyway, it could be proposed that a better approach is making robust, reliable hardware that can be sealed and presented as one component that can then be used in larger systems.

4.2.4 Field Level Operations Watch (FLOW)

This is an Android app made available for volunteers in developing countries to allow them to update a large database of the location and condition of water pumps throughout developing countries [13]. It uses 2G and 3G networks to update data remotely.

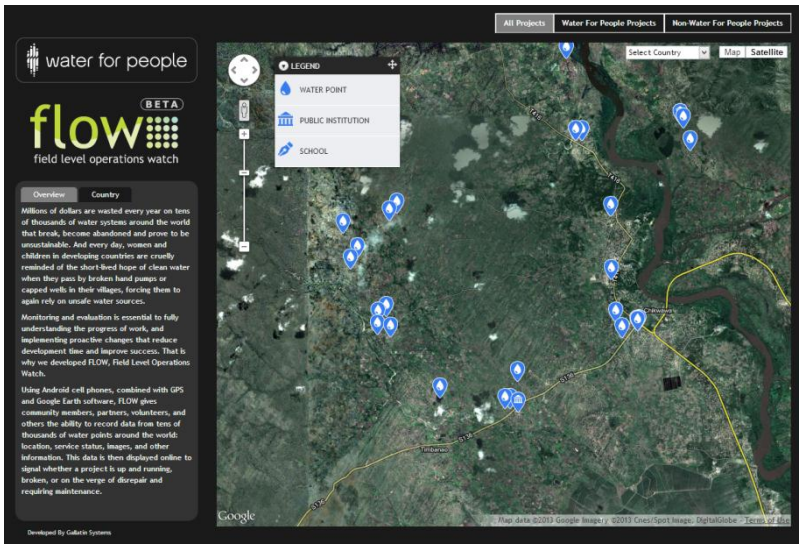


Figure 4|FLOW main screen - Water pumps can be seen as water droplet. [14].

The data is presented on a freely available web application, with pump locations marked on a map as can be seen in Figure 4. This map can then be used to select and show more detail of a pump's condition, allowing charities and non-profit organisations to only visit and maintain pumps that require it, therefore improving the effectiveness of these limited services.

It is encouraging to see mobile networks being used in this way, even though the proposed project is slightly different in operation to the proposed system. The same technologies are being used to great effect, and shows that mobile networks are available across many developing countries.

A similar approach could be applied to visualising the battery charging stations as more are deployed. This gives a good visualisation as to their geographical locations, and comparisons could be easily made by region or country.

4.2.5 Remote Energy Monitoring in the Developing World

This was a 5th year individual project carried out by an Electrical and Mechanical Engineering student at the University of Strathclyde in 2009. Also working underneath the Gambia Solar Project introduced on page 19, research was carried out into available technologies in developing countries. As with this project, GSM networks were chosen as the communication link and, differing from other products and projects researched, GPRS technologies were used to send data rather than using text messaging. This would potentially allow the remote system to be located in a different country to the server.

An existing solar installation on the roof of the university was monitored using two GSM modems to send information such as panel voltage and load currents. Two SIM cards were purchased, along with two plans from a mobile network operator in the UK for testing the prototype developed.

The requirement to have two Teltonika modems was due to their 'transparent RS232' serial port function, so a receiver is required to read the data that was sent by the remote transmitter. Mentioned in the report [15] is that the modems were 'a large percentage of the system cost, and as such a final system design would consider alternative components like isolated GSM modules'. As discussed in this report, these objectives can be achieved by using just one modem, as the receiver can be connected to the internet like a standard client and directly communicated with.

Low cost voltage and current sensors were developed, and data was sent over the GPRS link to a remote computer, which then displayed the information. Plans were made for creating a user-friendly interface for viewing the information, but this was not completed in the timespan of the project.

4.3 Vertically Integrated Projects (VIP)

In the second semester of the 2011-2012 session the University of Strathclyde set up a Vertically Integrated Project (VIP) tasked with researching and developing a prototype solution for

providing sustainable energy to rural communities. This project involved students from within the Electronic and Electrical Engineering department, post-graduates and academics.

The outputs and achievements of this project included a hardware prototype of a battery charging station being developed, and much research carried out into the loads that are typically required to be powered.

The project continues this year, including business and DMEM students alongside a much larger team of electrical engineering students. The main objective of the project in the long term is to develop a battery charging station that can be deployed in The Gambia.

A small group of engineering students have been assigned to the task of gathering data from the portable batteries, and storing it within the charging station. This report details where the project has interfaced with this area of the VIP, and objectives aligned so that the two projects integrate successfully.

4.4 The Gambia Solar Project

The Gambia Solar Project at the University of Strathclyde has been sending teams of staff and students out to rural schools in The Gambia to carry out installations of solar systems every year since 2006. This has had a positive impact on the schoolchildren and communities in these rural areas. Attempts have been made to ensure that this impact is sustainable, by revisiting installations during following trips to carry out maintenance and repairs.

Members of the Gambia Project have indicated that using remote monitoring on existing and new installations would be beneficial to the long term sustainability. Remote condition monitoring becomes more and more important as the number of installations increases, making travelling to each installation during the yearly trip less feasible.

5 Design

In order to develop a high level framework of the system it was necessary to understand exactly what was to be required from the system.

5.1 Calculations in the 'Cloud'

Simulations carried out on the operation of a battery charging station as discussed in section 7.1 supported the understanding that calculating the batteries state of health, and using that figure to influence the order in which batteries are handed out means that the entire asset life can be improved. It also gives the opportunity to add weightings so that only a set percentage of the portable batteries fail, minimising and managing the replacement cost to the manager.

Carrying out these calculations on the server allows values to be managed and batteries that are unsafe to be disabled remotely by those who have the technical knowledge to do so effectively. In addition doing these calculations away from the remote system means that the processing (that could be quite intensive with multiple large battery charging stations) is carried out on machines that are not limited by such extreme cost and environment requirements. The algorithms can also be altered remotely, with the only requirement being that more data is passed to the server if required.

This approach requires that any communication link is bi-directional, so that values can be passed back to the remote installation. The remote installation would then recommend batteries to be handed out based on their state of health, helping the operator manage the condition of their asset base.

5.2 Required Battery Information/Integration with VIP

Two students were assigned from the Vertically Integrated Project to research and implement solutions for acquiring data from the portable batteries, and aggregating the information at the charging station. It was important to work together to understand exactly how and what data could be acquired, and make sure that both projects were aligned and working to the same interface.

It was decided that environment information at the charging station would be limited to irradiance and air temperature, and this would be monitored by the remote monitoring system, leaving the charging station to gather data about the batteries.

Each battery would have a unique alphanumeric ID containing location information, the type of battery, the manufacturer recommended cycles, the initial battery capacity and the number of the battery within the charging station. This ID would be hardcoded and would not change. An example of a battery ID is shown overleaf.

SK05PB12008

SK – Location of the battery, Sambel Kunda (a rural town in The Gambia).

05 – Number of the battery at the charging station, number 5.

PB – Battery type, PB is Lead Acid.

1200 – Manufacturer's predicted number of cycles, 1200.

8 – Initial capacity of the battery, 8Ah. The available capacity will decrease as the battery ages.

For condition information, the number of cycles that the battery has been through and the time taken to charge the battery would be recorded. In addition the time that the battery was last charged would be recorded.

Load information would be provided by a number of samples taken over the batteries discharge cycle. When a battery is switched on, the current time and date is recorded along with the initial charge level of the battery. When it is switched off, the charge level is recorded, along with the number of minutes that the battery has been on. Recording the number of minutes on rather than the date and time when switched off was decided purely because of memory limitations (less memory is required to store one integer than a full date and time). In addition each sample would have the average temperature during that sample recorded.

This information allows the average load during the sample to be calculated using equation 1, given the battery's available capacity.

$$Average\ Load\ (A) = \frac{(startChargeLevel - endChargeLevel) \times availableCapacity}{timeLoadAppliedFor} \quad (1)$$

As the charge level is read as a percentage, this gives the percentage of the battery's available capacity used. Once divided by the time load is applied for, the load in Amps can be found. The available capacity is calculated by multiplying the battery's current state of health, equation 3, with the initial capacity, as seen in equation 2.

$$Available\ Capacity\ (Ah) = initialCapacity\ \times stateOfHealth \quad (2)$$

This is the procedure used by many laptop battery monitors [16] to calculate available capacity, however further research will be required into the validity of this statement. The issue is that the battery does not have a linear state of condition decay, however if the state of health figure is calculated more accurately equation 3 could prove to be valid. Again, this was outside the scope of this project, and facilities were made to alter the calculations that are used if required.

A manufacturer of a battery will recommend how many cycles the battery is expected to go through during its usable lifespan, and by keeping track of the number of cycles that it has

currently experienced, an indication of its state of health (SOH) can be calculated. This is shown in equation 4.

$$\text{SOH } \% = \left(1 - \frac{\text{numberOfCycles}}{\text{manufacturerCycles}}\right) \times 100 \quad (3)$$

In practice, many other factors influence a battery's condition, and many assumptions are present within this calculation. However calculating this value accurately is outside the scope of this project, and is left to future work within the Vertically Integrated Project.

As the VIP was working towards acquiring this data, this project could then move forward assuming the successful gathering of this information. This led to a natural interface between the charging station and the remote condition monitoring system.

5.3 System Framework

As in the project objectives, a low power remote system is required, with a bi-directional communication link as discussed in section 5.1. The remote system is to be an interface to the existing charging station, receiving data from and sending data to it to assist in the management of assets.

At the server, several technologies are required. A database is required to store all the information from the charging stations. Data needs to be received from the remote system, statistics calculated, data stored in the database, and information sent back to the remote installation. A server backend is to be developed to carry out these operations and act as an interface between the database and the communications link.

A web application is to be implemented; this will read data from the database and display it in a user-friendly manner. The application will also be capable of updating certain fields in the database so operators can manage assets remotely.

Users will be able to view information about their battery through a mobile application. This application will query the database and then display relevant information to the user.

Figure 5 shows a block diagram of the system design, with arrows depicting the direction of data flows. The sections that reside in the UK and those which sit with the installation can be seen. The interface between this project and the VIP is between the remote system and the charging station.

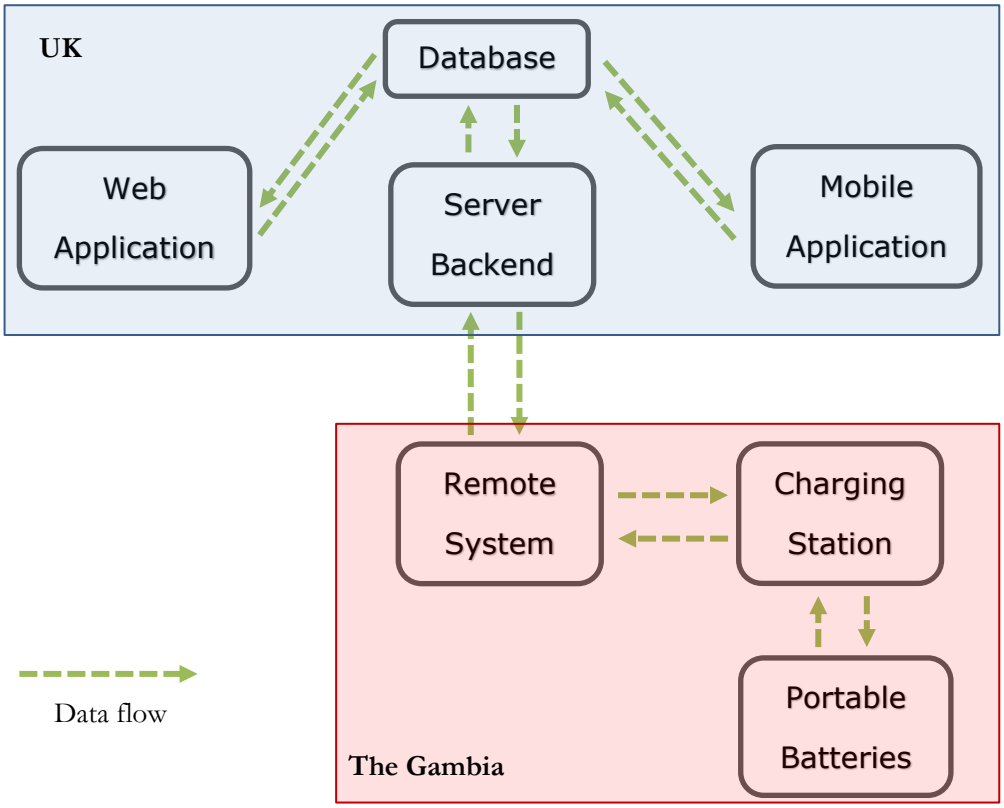


Figure 5 | High level block diagram of system framework.

5.3.1 Remote System

The remote system is required to have low power consumption and be robust, allowing it to be left unattended for long periods of time without maintenance. A microcontroller was a logical choice for this, as they can be programmed to carry out a single function indefinitely. Many microcontrollers also offer analogue inputs to measure their surroundings.

5.3.1.1 Hardware Selection

The Arduino system was chosen as the microcontroller due to experience gained developing the original VIP charging station prototype. In addition the Arduino is open source, allowing it to be replicated anywhere in the world. The use of the microcontrollers for hobby electronics also means that many add on components are available and well documented.

Other microcontrollers were considered, such as Microchip PICs. Previous experience gained with these suggested that, although possibly lower power and slightly cheaper, the added complexity in development was considered too high for this proof of concept project.

Research concluded that the main hardware requirements of the microcontroller were to:

- Be as inexpensive and low power as possible.
- Include two serial ports, one to communicate with the communication system, and one to communicate with the charging station.
- Be able to run a LCD screen and status LEDs for feedback to the user as the system's operations are carried out.

Originally an Arduino Uno was selected and purchased. The Uno has a single hardware UART chip, only allowing serial communication to only one device. However, using the SoftwareSerial [17] library included with the Arduino distribution, two digital ports can also be used to transmit serial data. This is at a lower rate than is possible with a hardware port.

Testing carried out with the chosen GSM modem discovered that although sending data was possible, receiving data back from the modem was not reliable using the SoftwareSerial library. This is due to the modem sending received data too quickly back to the Arduino for it to read.

Another issue discovered using the Uno was the lack of internal memory. The Arduino microcontrollers have three types of memory: Flash, SRAM (static random access memory) and EEPROM [18]. Flash memory is non-volatile and is where the Arduino code is stored, so that it is available on a reset. SRAM is where the code stores variables while running and EEPROM is available memory to store long term memory.

It was discovered during implementation that one day's information for a battery charging station with 80 batteries could require upwards of 26kB. In terms of communication this is very little over data networks; however it is more than is available on the Uno for variable storage. This can result in random resets and memory overflows, causing unpredictable errors. Table 1 shows the sizes of memory available on the Uno, and the Arduino Mega – a larger device.

Table 1 | Arduino Memory Sizes

Type	Arduino Uno (ATmega328)	Arduino Mega (ATmega2560)
Flash	32kB (0.5kB used for bootloader)	256kB (8kB used for bootloader)
SRAM	2kB	8kB
EEPROM	1kB	4kB

On both devices, storing all the charging station information in SRAM as normal is not possible. Using C syntax, it is possible to access and store data in the Flash memory on the microcontroller chip (the ATmega… is the device used to power the Arduino board). Even with this, there is not much room for data in the Flash memory on the Uno, so the decision was made to move to the Arduino Mega with the 256kB of Flash memory available.

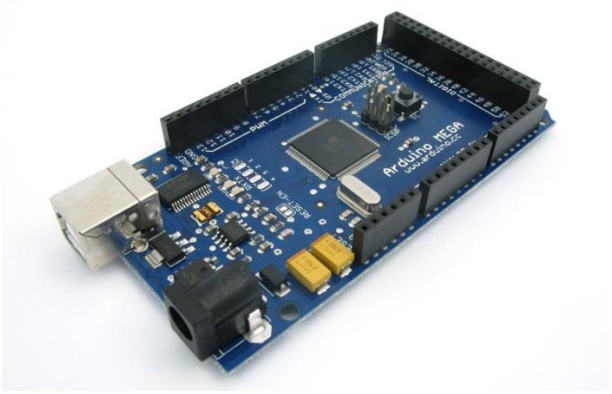


Figure 6|Arduino Mega2560

In the future it may not be necessary to store the information on the remote communications device; it could be possible to stream the data across the two serial ports. However this project is working independently from the development of the charging station at this time and therefore needs to be demonstrated separately, so it is necessary to store the information on device.

The Arduino Mega seen in Figure 6 also includes three extra UART chips, allowing hardware serial communication with the modem and with the charging station.

5.3.1.2 VIP DMEM Product Designs

The system is to be as robust as possible and include feedback to users if the system is in operation. The VIP project includes a sub-team of DMEM (Design Manufacture and Engineering Management) students. One of their objectives was to design concepts for the remote communications system, protecting it from the heat, insects and to make the system secure against tampering. Specifications of the hardware requirements were passed across and feedback was given as to why certain designs were better or worse.

This part of the project is still on-going, and a prototype casing will be manufactured in time for deployment in The Gambia this year.

5.3.2 Communications Link

The GPRS protocol was only added into GSM in 1997 [19], meaning after time for the standard to filter down, it is only fairly recently that sending internet packets across 2G networks has become feasible in developing countries. Other wireless technologies are being developed and deployed rapidly like Wi-Fi, WiMAX and CDMA 450. Scanbi-Invest also suggests that technologies like WCDMA 'would be a cheaper technology to deploy in many green field operations and would be more appropriate for carrying data traffic' [20]. However in Africa, GSM voice networks are already heavily invested in and available as can be seen in Figure 7, so piggybacking data on top of these voice services makes internet very accessible. As a comparison, GSM has an 80-85% market share worldwide, and CDMA has around 10-15%. This is due to the lower cost of GSM-based modules [21].

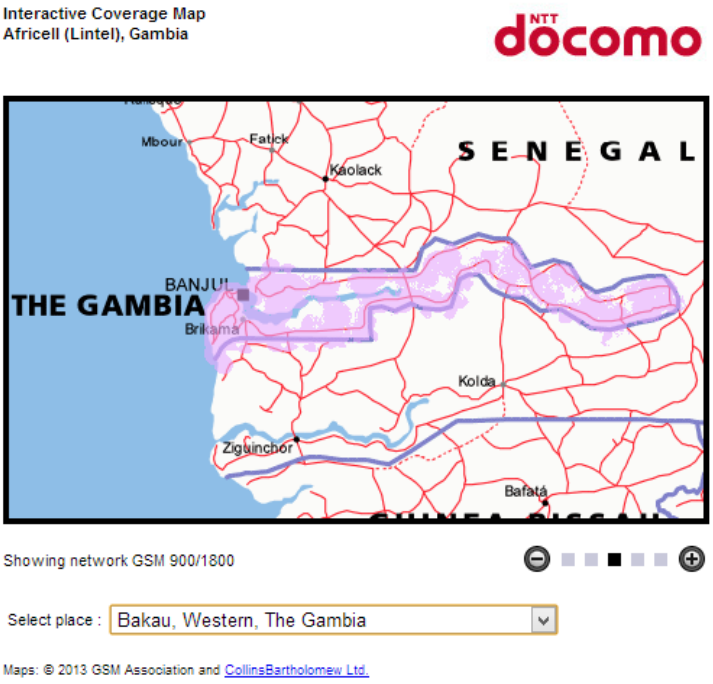


Figure 7 | GSM coverage in The Gambia. Purple areas show GSM availability.

Gamcell, QCell and Africell in are the main mobile operators in The Gambia. Africell operates a GSM 900 MHz and GSM 1800 MHz network [22], the same as many operators in the United Kingdom, so their networks were selected for use, giving a maximum download and upload speed of 85.6kbps. This is slower than modern standards, but fast enough for this purpose. Any modem purchased has to be able to operate on these frequencies. Africell offers communications to GCF (Global Certification Forum) modules and devices [23], so any modem chosen will also have to comply with this specification. The GCF motto is "Test Once, Use Anywhere" which highlights the broad acceptance of GCF approved devices [24].

5.3.2.1 SIM Data Plan

A data plan is required for this project, so a free SIM was chosen from the network operator T-Mobile. This company has good coverage around the University of Strathclyde for 2G (all that is required). In addition, a bolt on pack of £25 for internet use for 6 months is used that to cover the planned time span of the project. The use of a pay as you go SIM will also allow the project to be demonstrated and used for £1 a day after the 6 month bolt on expires, or more data can be added as required. Unlimited data was purchased, however the actual system is unlikely to require this much. Part of the objectives of this project is to understand how much data would be required for a charging station in a developing country.

5.3.2.2 Modem Choice

Research was done into how to operate GSM modules interfaced to microcontrollers. Most modems are controlled using the industry standard Hayes command set. This command set consists of short text strings that are combined to produce commands for operations such as dialling, hanging up and configuring the modem. These commands are then sent to the module over a serial connection, so it is possible for them to be generated dynamically on a microcontroller.

Various modules are available for Arduino microcontrollers, with the chosen modem being a Spreadtrum SM5100B, broken out to the Arduino by Sparkfun Electronics.

5.3.2.2.1 Spreadtrum SM5100-B [25]

This shield was chosen due to the manufacturer's (SparkFun Electronics) experience in developing components for Arduino boards. In addition a supplier (Proto-Pic) was found that is based in Scotland.

The green block seen in Figure 8 is the GSM modem, as found in most mobile phones behind the battery. This is a GPRS capable modem that is GCF approved. It has an extended AT (Hayes) command set to include TCP/IP protocols. When in sleep mode it draws a minimum of 2mA, and during data transmission it draws 400mA, with peaks of 2A in bursts. It has a temperature operation range of -20°C to 55°C [26]. The maximum temperature is close to Gambian temperatures during the summer, testing will be carried out into whether cooling is required in the future. It is suggested that some variant of a heat sink/cooling fins would be more suitable than a fan, with dust and insects being an issue, along with the power consumption issues.

Figure 8|SparkFun Cellular Shield with SM5100B

Evidence was also found that this component has worked in the UK for other users, and there is much documentation from other projects that could be consulted to get the module running [27].

5.3.2.2.2 Simcom SIM900 [28]

Figure 9|Seeed Electronics GPRS Shield

The module shown in Figure 9 had plenty of documentation on the manufacturer's website and its functionality seemed to be as required. However further investigation found that this was a Chinese manufacturer, and due to the cost of shipping and lack of support if there is an issue this option was disregarded. The modem on the board is suitable for requirements.

5.3.2.2.3 GPRS/GSM QUADBAND MODULE FOR ARDUINO (SIM900) [29]

This was the first component researched; however this had to be bought from Germany. There is a huge amount of documentation on the website, tutorials and example code etc. but not

much evidence of it being used in other projects. As seen in Figure 10, the shield only inserts into those pins necessary, which makes the system look very tidy. Again, the SIM900 is a suitable module for the project.

Figure 10|SIM900 Quadband Module

5.3.2.2.4 Other Options

Other projects in this area have used mobile phones to access GSM networks [12]. This option was disregarded to the added complexity and having redundant hardware on site. This also makes it more difficult to access the functions required for data upload.

Previous projects [15] have also used Teltonika Wireless Modems [30] to provide a transparent serial port that utilises GSM/GPRS networks to send and receive data. It was decided that the simplicity of this solution is not ideal, as two modems are required to send and receive due to the abstraction of the channel to appear as a standard serial port.

5.3.2.3 Data formatting

The battery charging station data is required to be formatted in such a way that facilitates easy transmission over the GPRS networks. The modem allow strings of data to be sent, so it is necessary to set how the data is laid out, so that the transmitter understands what to send, and the receiver knows what to expect.

During testing the use case of an eighty battery charging station was used, with four load samples per battery, adding up to a send string 23kB long. Over a year this would be 5MB of data costs, which is minimal on most mobile networks. By logging into the T-Mobile online account provided with the SIM card, it could be seen that the estimations of data usage were correctly logged by the operator.

5.3.2.3.1 Transmission Format

The data to be transmitted every day from the charging station is to include:

- Location information (name, number of batteries etc.)
- Environment data (irradiance and temperature for 24 hours)

- Battery information
    - Load Information per battery

The information is variable in size, for example a battery may have very little load activity. There are two solutions to this. The first is to set the size of the transmitted data initially, and send null values if the information is not there. This has the issue of sending redundant bytes, therefore costing more, and also setting a limit, so if there is more information than can be sent data is either not sent or another sending operation has to be initiated to send a little more data. The second solution is to use variable sizes of strings. The number of batteries and load data needs to be included near the start of the string so the receiver knows when to stop parsing the string, and what values correspond to what information. This is the solution chosen.

An example of the sent data string is included below. Data was minimised and compressed as much as possible, for example multiplying by 10 to remove decimal points.

"[SK100413:80[0004:258,0003:264,0004:281, x24 samples]…

- SK – Code for Sambel Kunda
- 100413 – Date that environment data refers to, 10/04/2013.
- 80 – Number of batteries registered at charging station, 80.
- 0004 – Irradiance at midnight on the date specified, 4W/m^2.
- 258 – Temperature at midnight on the date specified, multiplied by 10 to avoid sending unnecessary decimal point, 25.8°C.

The two values above are repeated 24 times for the 24 hours in the day, and closed with a ']' character.

…[SK01LI19005:04:0413:0104132245:566:1[(1004130000:452:053:012:269), x number of load samples]] x number of batteries at charging station"

- SK01LI19005 – Battery ID, as discussed in section 3.2.
- 04 – Number of load samples contained in battery.
- 0413 – Number of cycles battery has experienced.
- 0104132245 – Date and time that battery was last charged, 1/4/2013 at 22:45.
- 566 – Number of minutes it took to charge the battery, 566 mins.
- 1 – With customer or not, 1 = with customer, 0 = not with customer.

It is not necessary to send the battery's disabled status, as this is always calculated on the server, so if a battery's condition variables change, it could be possible for a battery to become re-enabled after being disabled.

Enclosed in `(` and `)` is a single load activity:

- 1004130000 – Date and time load activity started, 10/4/2013 at 00:00
- 452 – Number of minutes load was on for, 452, this allows the end time to be calculated on the server without sending the full date and time.
- 053 – Charge level at the start of load activity, 53%.
- 012 – Charge level at the end of load activity, 12%.
- 269 – Average temperature during activity, again multiplied by 10, 26.9°C.

The load samples are then repeated for the number specified in the battery information, and then the whole thing is repeated for the number of batteries specified at the start of the transmitted string.

5.3.2.3.2 Receiving Format

The return message needs to return the updated state of health and disabled status to the charging station. These values are calculated on the server, and then formatted as below, including the number of the battery that the values relate to. This again is to cut down on the amount of data transferred, as information such as location of the battery is already known from the received string, so only the battery number is required.

An example of the returned string is shown below.

...[15,67.8,1][16,92.3,0]...repeated for all batteries in the charging station

- 15 – Battery number, 15.
- 67.8 – Battery state of health, 67.8%
- 1 – Disabled status, 1 = disabled, 0 = not disabled.

These values can then be transferred to the charging station, parsed and the appropriate battery can be updated. If a battery is disabled, it will not be possible to switch it on to a load, unless it is re-enabled by a developer from the web application.

5.3.3 Server Backend

At the server, a relational database management system is required to manage the information and structure it in a manner that is easily read. MySQL was chosen as the system to do this, as it is very widely used.

Other database management systems include CouchDB [31], a flexible management system built around JavaScript Object Notation (JSON) strings. This is the system upon which much of Medic Mobile's development is being based [9], however it was decided to use the more structured, more widely used SQL system to allow facilitate development in the time available.

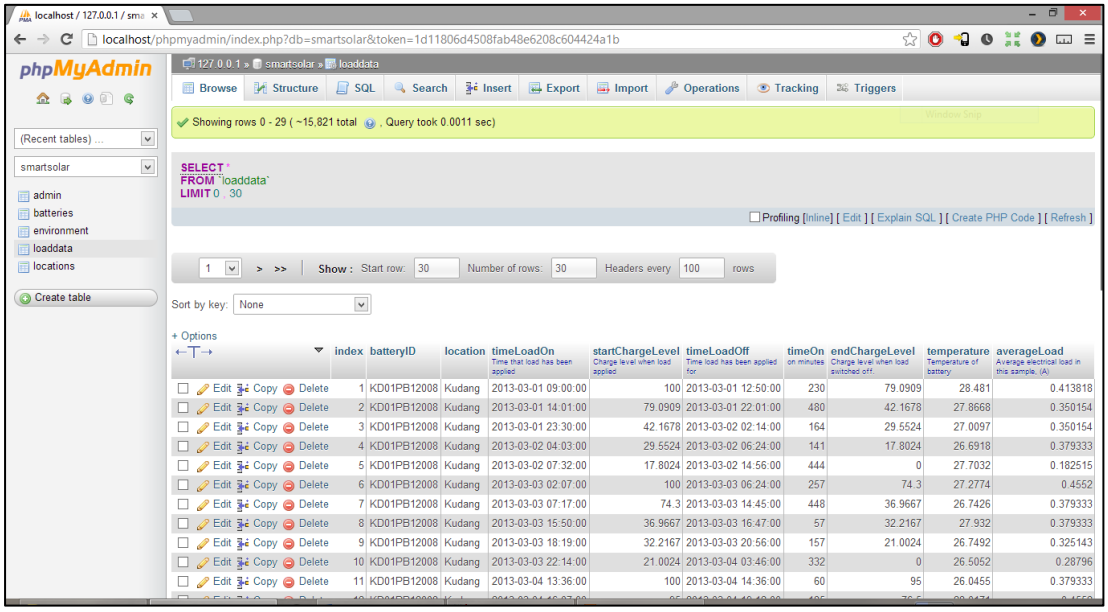


Figure 11 | PHPMyAdmin Graphical User Interface

Research could be carried out into using this more flexible system in the future.

The development package XAMPP [32] was used as an easy installation of the main components required for the project, Apache, MySQL and PHP. Without this, installing these components was very time consuming and error prone, especially when working across multiple machines. XAMPP also includes a software package called PHPMyAdmin, which allows the SQL database to be viewed and managed in an easy to used visual interface as seen in Figure 11, rather than interacting through the command line.

5.3.3.1 Database Structure Design

Within the database, tables are used to aggregate and store information. In order to store the

admin

Table comments: admin

Column	Type	Null	Default	Comments
id	int(11)	No		
username	varchar(30)	No		
password	varchar(120)	No		
approved	int(11)	No		Allows users into areas of site 0 = not allowed, 1=user, 2=operator, 3=designer

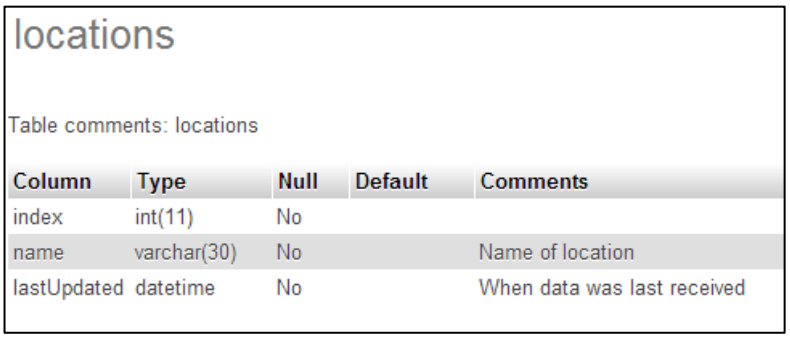
Figure 12 | Administrative table

information from the charging station in a structured manner, time was spent designing the relationships between different tables.

The first table is an administrative table, named 'admin' as seen in Figure 12. This is used to store information about users, and their status within the system. This table will be queried to discover whether the user is registered with the system, and whether they are approved to have access to certain features. The levels of access are:

- 0 – Not approved to use the system.
- 1 – Approved as a user.
- 2 – Approved as an operator.
- 3 – Approved as a developer/designer.

Location information is stored in the table 'locations'. This stores the location names, and when they were last updated as seen in Figure 14. An improvement could be to store the codes associated with the location, for example 'SK' for Sambel Kunda, and then load the associated name dynamically, rather than hardcoding the location codes into the system. This would aid maintenance. The date and time is stored as the SQL format datetime. This is the format "YYYY-MM-DD HH:MM:SS".

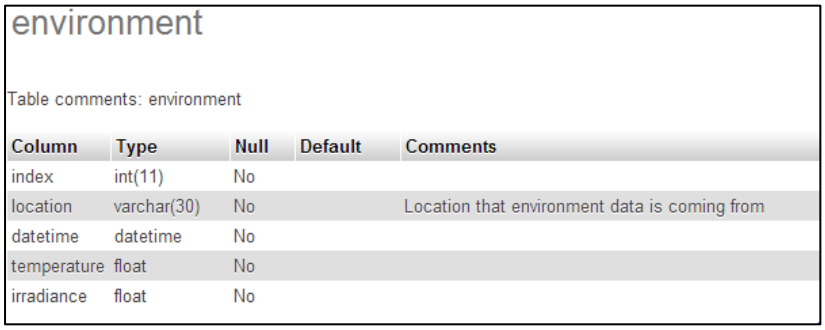


locations

Table comments: locations

Column	Type	Null	Default	Comments
index	int(11)	No		
name	varchar(30)	No		Name of location
lastUpdated	datetime	No		When data was last received

Figure 14|Location information table



environment

Table comments: environment

Column	Type	Null	Default	Comments
index	int(11)	No		
location	varchar(30)	No		Location that environment data is coming from
datetime	datetime	No		
temperature	float	No		
irradiance	float	No		

Figure 13|Environment information table

Environment information is stored in the table 'environment' as seen in Figure 13. This includes all historical samples for all locations. This can then be filtered using SQL queries to only show certain dates or only certain locations.

batteries

Table comments: batteries

Column	Type	Null	Default	Comments
index	int(11)	No		
batteryID	varchar(11)	No		String ID
location	varchar(30)	No		Where the battery is used e.g. Sambel Kunda
number	int(11)	No		Index of battery at location
type	varchar(30)	No		E.g. Lead acid
numberOfCycles	int(11)	No		Number of Cycles on battery
recommendedCycles	int(11)	No		Manufacturer recommended cycles
initialCapacity	float	No		Initial Battery Capacity
timeCharged	datetime	No		When battery was last charged
timeTakenToCharge	int(11)	No		in minutes
stateOfHealth	float	No		Batteries state of health as a percentage
availableCapacity	float	No		Available capacity after wear
withCustomer	tinyint(1)	No		True for with customer, false for in charging station
disabled	int(11)	No	0	Developer intervenes, set to 1 to disable battery.

Figure 15 | Battery information table

Battery data is stored across two tables. Current information is stored in the table 'batteries' shown in Figure 15, and load information is stored in the table 'loaddata' shown in Figure 16.

loaddata

Table comments: loaddata

Column	Type	Null	Default	Comments
index	int(11)	No		
batteryID	varchar(11)	No		
location	varchar(30)	No		
timeLoadOn	datetime	No		Time that load has been applied
startChargeLevel	float	No		Charge level when load applied
timeLoadOff	datetime	No		Time load has been applied for
timeOn	int(11)	No		on minutes
endChargeLevel	float	No		Charge level when load switched off.
temperature	float	No		Temperature of battery
averageLoad	float	No		Average electrical load in this sample, (A)

Figure 16 | Load information table

The load data table stores all historical samples of load information, alongside the battery ID that the load information refers to. This battery ID has to exist in the battery list, or an error will be thrown. The battery ID in the battery list is also unique, and no duplicates can exist within the table.

5.3.3.2 Language choice

Java was chosen as the language to implement the server backend due to familiarity with the language and the fact that it runs in virtual machine – allowing it run in the same environment on any operating system – meaning that the server functionality can be ported to Linux after development has finished. In addition connectors are available to connect the application to a MySQL database, and networking is possible through the ServerSocket class.

5.3.4 Web Design

The web application is designed to be as user-friendly as possible, with summaries of historical data alongside the functionality to generate reports for specific dates or locations. The design is to be clean and clear, with a recognisable brand and consistent style across pages.

5.3.4.1 Prototyping

Four main technologies are used within the web application:

- HTML – Defines the content of the page, along with the structure.
- CSS – Sets the style of the page, including colours and hover animations.
- PHP – Generates HTML dynamically, often from results after querying a MySQL database. Runs on the server.
- JavaScript – Used in a client's browser to apply dynamic content including animations and plugins to improve the user experience.

HTML is used to define the semantic content of a page. All static content is hardcoded between opening and closing tags. Text, input forms, tables and lists can all be declared, along with the hierarchy of headers, headings, paragraphs and footers. These are split up into divisions or 'divs', which can then be named to have styles applied to them with CSS, or content generated for them using JavaScript.

CSS, or Cascading Style Sheets is a language used to apply styles to HTML pages, allowing the separation of page semantics and style. This also allows the same style to be applied to multiple pages, meaning they all share the same look.

The 960 grid system [33] is used to streamline web development. It allows commonly used CSS dimensions to be applied up to a width of 960 pixels, a standard size for web pages. When writing HTML, classes are assigned as to the divisions expected size and positioning on the

page. The framework is imported at the top of the page, and it takes care of pixel sizes etc. This allows web pages to be developed much more quickly than would otherwise be possible, especially with inexperience in web design.

More modern complex designs use fluid designs, where the page is responsive in its generation based on the size of screen that is viewing it. This was decided to be too complex, and the standard size of 960 pixels wide was used. This width is just lower than the minimum resolution, 1024 x 768 pixels, of modern monitors, so should display fully on most computers. Future work could be carried out to make the page viewable for mobile devices.

PHP can be used to load information from a server using MySQL queries, and then generate HTML content by using 'echo' statements. It is also used to check user logins, and if they are not stored in the current session, redirecting the user to the login page.

JavaScript is a client side scripting language. Many APIs are available to display interactive charts on a web page, including Google Charts [34] and HighCharts [35]. HighCharts was selected as more customisation is available, allowing the charts to be skinned to match the colour scheme of the rest of the site. Data for the charts can be generated manually, using PHP statements to load the data, or by an AJAX call to a PHP function on the server which returns JSON encoded data. Licensing is free for student projects, if there was any commercialisation of a product, a fee would have to be paid.

The website wire-framing tool 'Mockingbird' [36] was used to develop some initial plans for early site prototypes. These initial prototypes were then developed in order to assess different plugins, and test the web design concepts discussed above.

5.3.4.2 Brand

While not a primary objective, having a clear brand adds to the user experience. The name 'Smart Solar' was proposed, conveying the fact that the system is focused on solar systems, and adds intelligent or 'smart' asset management algorithms to a traditional system. The name was stylised as 'smartSOLAR', providing an eye catching layout, even when written in plain text.

5.3.4.2.1 Colour Scheme

The colour scheme uses greens to convey the environmental element of the system, and blues to reflect the usual colour of solar panels or the sky. Other colours within the application were kept to different shades of grey, ensuring that highlight colours stood out.

5.3.4.2.2 Logo Design

Several iterations of a logo were designed. The final design uses a blue rectangle to represent a solar panel, with radio waves emanating from one corner, representing the remote communication. The stylised name was written with 'SOLAR' in green for emphasis. The

capitalisation was also designed so that if the logo was presented in greyscale, the emphasis would still be present. The logo was developed using Adobe Photoshop CS6, using the same colours chosen for the colour scheme and is shown in Figure 17.

Figure 17 | Final smartSOLAR logo

This logo is used throughout the web and mobile application to provide a clear and consistent brand.

5.3.5 Mobile Application

Most phones used in developing countries are donated by western workers or charities, and therefore are two or three generations behind the most modern phones. The proposed design was to develop a native Android application, as it is more likely the cheaper and widely available Google operating system will penetrate the market before Apple's iOS, or Microsoft's Windows Phone.

Developing for Android natively can be a complex problem, with many different devices and versions of operating systems available. A solution is to use 'PhoneGap', a framework used to move browser based apps onto native platforms. Browser based apps, as discussed in section 4.2.1, allow cross platform compatibility as older SIM apps also offer. The issue with this is that the browser does not have access to the device's hardware, such as camera, GPS or accelerometer. PhoneGap offers this native functionality through plugins that are device specific, meaning that very few alterations have to be made to an application for it to be deployed across all mobile operating systems. In addition, website code and styles can be reused to create the browser application, saving time for developers, as they do not need to learn new languages and development environments.

The mobile application was to use a QR code reader to scan batteries. QR codes are a special type of barcode which allows any information to be embedded within it, including text. A battery QR code would have the relevant ID embedded into it, the app would scan a battery and

the server queried to return important non-technical battery information back to the user, again using mobile internet networks.

The system needs to offer a consistent user experience with the web application, this was achieved by designing the application using the same colours and styling, and prominently displaying the 'smartSOLAR' logo.

Initially the application is to be deployed on an Android phone, as a Google Nexus 4 is available running the latest version of the Android operating system. If required, future objectives could include improving the functionality, and making the app available across other platforms.

6 Implementation

Hardware was purchased, and a prototype system was developed.

This is capable of receiving data from a remote microcontroller over GSM/GPRS networks, calculating batteries state of health, disabled status, average loads over time, and updating a database of information for multiple charging stations. The Java server is then capable of sending data back to the charging station so that batteries can be updated.

A web application has been developed that can show the information in the database in a user-friendly manner, using charts and tables where appropriate. Facilities are also available to manually disable a battery if required. This feature is limited to users who are logged in as a developer/designer.

Finally, a mobile application has been developed and tested to scan a QR code, and display information from the server's database about the battery ID embedded within the code.

6.1 Arduino and GSM Modem

The Arduino Mega was connected to the GSM module using soldered on headers to match up with the pins on the microcontroller. The GSM modem on the shield has it's transmit and receive pins connected to digital pins 2 & 3 on the microcontroller. This was fine on the Arduino Uno using the SoftwareSerial library (as discussed in section 5.3.1.1), however when using the Mega, it was necessary to disconnect digital pins 2 & 3, and directly connect them to pins 19, RX1, and 20, TX1. These are the pins for the second hardware UART chip available on the Mega.

A bench power supply is used to supply power to the prototype, as the modem during transmission can draw up to 2 Amp. The Arduino is unable to supply this amount of current without causing damage to the board. Even though a large amount of current is required, it is in very short bursts. In addition, the system will be located at the charging station, so there will be a semi-permanent supply of power from the solar panels. This power supply is connected to VIN on the shield, and ground. There is a regulator on the shield which converts the input voltage to 3.3V for the modem, and also a regulator on the Arduino, converting the input voltage to 5V for the ATmega2560 microcontroller. It is important to note that connecting the input power to the 5V output on the Arduino/shield will cause damage to the Arduino.

A 16 x 2 character LCD screen was connected to digital pins 12, 11, 10, 9, 8 and 7 referring to the Arduino LCD library example [37]. The backlight pins, A and K on the screen were connected to +V and ground respectively. The contrast pin is to be connected to the wiper end of a 10kΩ potentiometer to adjust the contrast of the screen. In the prototype this is connected

to the middle of a simple voltage divider circuit to minimise components, however in the final system, a potentiometer will be required to adjust the contrast as changes in temperature affect the screen's operation. This screen is used to display status information about the transmitting and receiving process, along with configuration parameters during operation. More detailed information is also sent to the first serial port, so that important parameters and responses can be viewed on a computer during testing and demonstration. In practice this serial port would be connected to the charging station to update batteries.

A high gain antenna is connected to the output on the GSM shield to enable connection to mobile networks. A yellow LED is connected to digital pin 13, used to indicate when a server is connected, and a red LED is connected between +V and ground to indicate when power has been applied.

The final prototype remote system can be seen in Figure 18 and Figure 19.

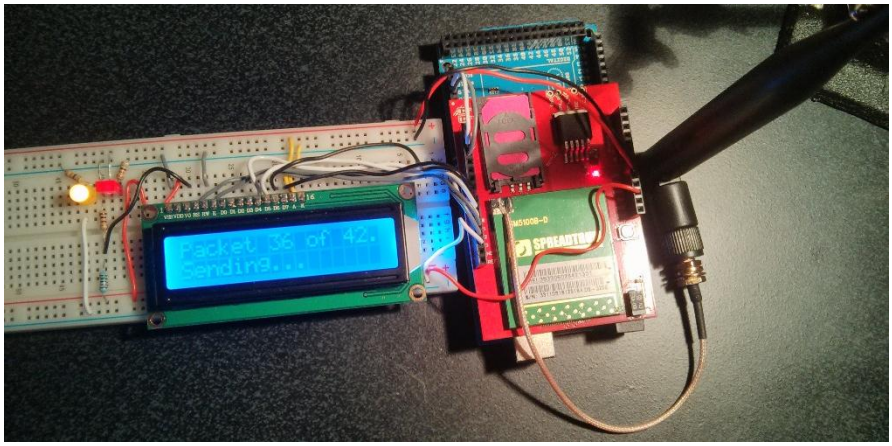


Figure 18|Remote System Prototype

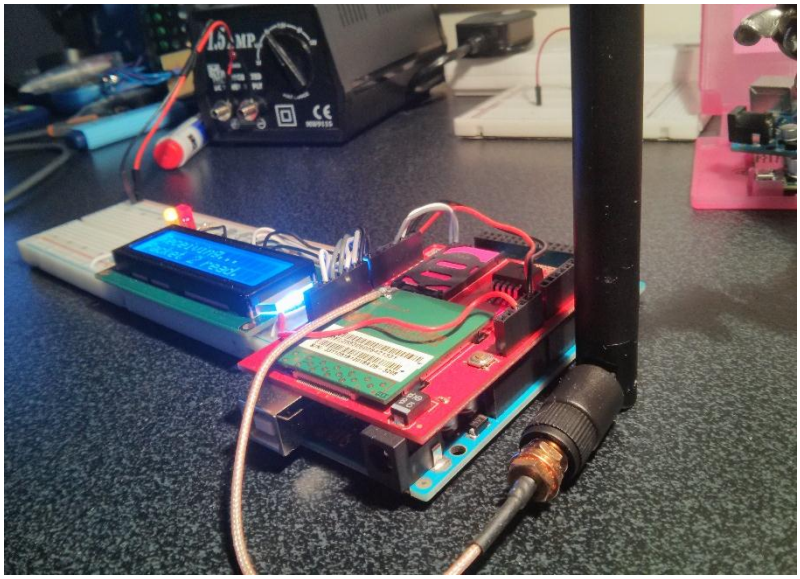


Figure 19|Remote System Prototype, with power supply.

The wiring diagram is shown in Figure 20, and the schematic shown in Figure 21.

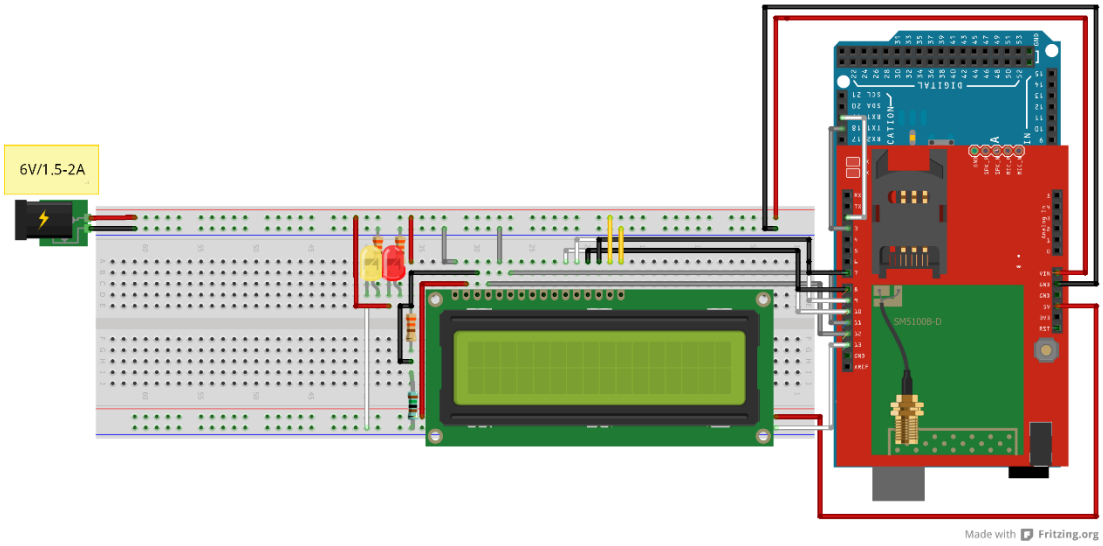


Figure 20 | Hardware Wiring Diagram

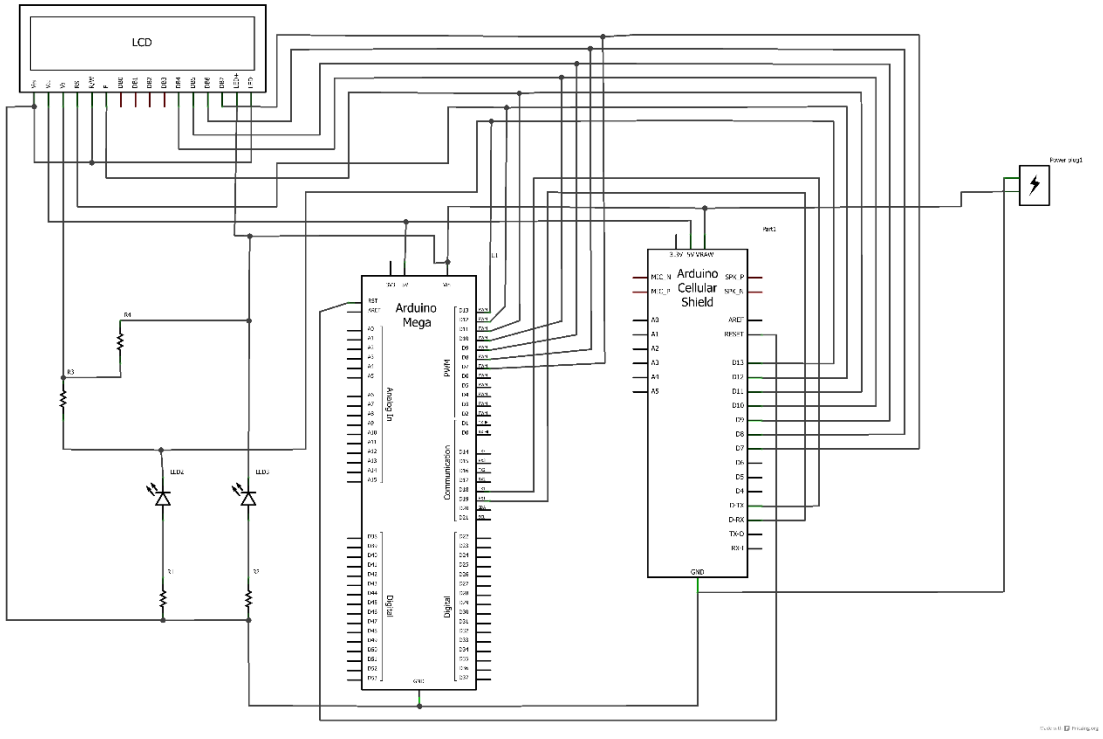


Figure 21 | Hardware Schematic

While this schematic looks complex, most of the connections are connected through header pins, as the module is stacked on top of the Arduino Mega. This means that the actual prototype has very little wiring in comparison.

The schematic for the SM5100B modem is included in appendix 11.2. It can be seen that there a few components on the shield, and the open source nature of the Arduino system would allow this to be replicated for very little cost in future.

6.1.1 Configuring Modem

As discussed earlier, the modem responds to the Hayes command set. These commands are sent through the serial connection, and begin with the characters "AT+". The datasheets for the modem [38] lists these commands in full for reference. Also available is a document/tutorial to assist in using AT commands to control the TCP/IP stack on these modules [39].

The commands allow UDP and TCP sockets to be created and destroyed, data to be transferred to and from the modem, and IP status obtained from the active link (for example check that the system is connected).

In order to test the modem, a sketch was uploaded to the Arduino which mirrors any input bytes appearing at the main serial port to the serial port connected to the modem and vice versa. Once plugged in and run, the following was returned from the modem in a serial monitor on a PC.

```
1. +SIND : 1
2. +SIND: 10,"SM",1,"FD",1,"LD",1,"MC",1,"RC",1,"ME",1
3. +SIND : 11
4. +SIND : 3
5. +SIND : 4
```

Line 1 indicates that a SIM card has been inserted, line 2 gives the status of the modules phone book – indicating that the SIM card is ready. Line 3 is the status code for registered with network, and lines 4 and 5 are call and SMS ready respectively. If all of these codes do not appear, there has been an issue in the modem power on.

The module was tested by calling the modem from a mobile phone. The following output was returned:

```
1. RING
2. RING
3. NO CARRIER
4. +SIND: 6,1
```

"NO CARRIER" was returned once the phone was hung up, and line 4 was returned to indicate that the call has been ended, and the SIM is ready to use again.

The serial program Terminal was downloaded to offer full two way communications over the serial port, as the serial monitor in the Arduino IDE is fairly limited. It was possible to make phone calls from the modem to test connections using the commands:

```
1. ATDxxxxxxxxxxx
2. ATH
```

Replacing the x's with a phone number dialled the number, and the command on line 2 hangs up the call.

The Access Point Network (APN) settings for T-Mobile were obtained from the product page [25] and are shown below.

Network: general.t-mobile.uk

Username: tmobile

Password: tm

These settings vary from network to network, so if using a different SIM card, APN settings will need to be obtained for that carrier's network to use GPRS technologies.

The next set of commands is used to check for GPRS connectivity, and configuring the modem with the APN settings shown above. The module will work without checking for GPRS; however the system will not connect to a remote server if GPRS is not available. It is possible to be connected to the mobile network without GPRS, this was found during testing, and therefore it is recommended to wait for this to become true before proceeding. Responses from the modem are shown in italics.

1. Query and wait for GPRS connectivity, the response will be "+CGATT: 1" for connected, and "+CGATT: 0" when not connected.
    - `AT+CGATT?`
    - *`+CGATT: 1`*
    - *`OK`*
2. Set up PDP (Packet Data Protocol) context with APN settings shown above. Returns an OK if successful.
    - `AT+CGDCONT=1,"IP","general.t-mobile.uk"`
    - *`OK`*
3. Configure the PDP context parameters if a username and password is required for the GPRS APN (which it is on T-Mobile's network).

- `AT+CGPCO=0,"tmobile","tm",1`
- *OK*

4. Activate the PDP context.

    - `AT+CGACT=1,1`
    - *OK*

6.1.2 Sending Data

This section assumes the commands detailed previously have been successfully carried out. The next steps required a server to be implemented. As discussed earlier, this will be a Java implementation of a socket, described in more detail in section 6.2.

1. Configure remote host and port to open a TCP connection. TCP connections are inherently more reliable than UDP connections, using handshaking and acknowledgements to guarantee data transmission to a large extent.

    - `AT+SDATACONF=1,"TCP","90.205.204.45",6100`
    - *OK*

Note the IP address here is a public IP, meaning traffic will arrive at the external router. This router needs to be configured to allow data arriving on port 6100 to be sent onto a local IP address. This is done through a method known as port forwarding, and is not covered in this report due to the differences in method based on manufacturer of router, and the availability of tutorials online. In the future it is planned to run the server on a Raspberry Pi minicomputer, situated in the university with a static IP address assigned to it that does not change.

2. Start TCP connection (last value should be a 0 to close the connection).

    - `AT+SDATASTART=1,1`
    - *OK*

At this point the server recognises a connection being made to it. It is possible to check that the socket is connected by using the following command:

3. Query socket connection status.

    - `AT+SDATASTATUS=1`
    - *+SOCKSTATUS:1,x,0102,0,0,0*

The x in the returned string is a 0 for not connected, and a 1 when the socket is connected. The other values refer to the number of TCP packets sent, and number of TCP packets acknowledged by the receiver.

To actually send information, there are two methods possible. The first is to use "AT+SDATASEND". This sends the data as hex values, meaning that the information has to be cast back into ASCII characters at the server. It also has a more complicated sending procedure, requiring a CTRL+Z character to be sent to terminate the data to send, and trigger the load into the sending buffer.

The second method is to used "AT+SSTRSEND". This does not require pre-declaration of the data size, as with the previous method, which allows some flexibility. In addition this sends the information as a String data type, or a set of ASCII characters, which are easily parsed at the server.

4. Sending the string "helloworld!" to the server.

    - `AT+SSTRSEND=1,"helloworld!;"`
    - *OK*

Reading the first byte on the server as an integer gives the value 104, which is the ASCII value for 'h'. Reading the entire string gives the string "helloworld!" as required. It is worth noting that this method does not send carriage return or line break characters within the message, meaning that any receiving method on the server needs to read the string until it reaches an end of file character in this case set to be ';'. This character is not used in the send string format, and therefore can be used in this way without prematurely breaking the connection. Using the end of file character means that multiple messages can be sent, and concatenated at the server until the last message finishes with the end of file character.

6.1.3 Receiving Data

Receiving data on the modem is slightly more complicated. As the modem transmits data back to the microcontroller very quickly, it is necessary to read the information extremely quickly or it will be lost. To do this the hardware UART chip on the Arduino Mega was required. The speed of this serial port was also increased from 9600bps to 115200bps in the Arduino code:

```
Serial1.begin(115200);
```

And the modem was configured to operate at this speed using the following command when connected at 9600bps.

```
AT+IPR=115200
```

This was only required to be done once, as this configuration is stored in non-volatile memory within the modem.

There are two methods using the command "AT+SDATAREAD" to receive data. The first is to receive in hex format, and the second to receive in ASCII string format. Testing was carried out, and data was successfully received from the server, however it was in hex format. On the Arduino this isn't a simple task to convert from a hex value to an ASCII string, so an alternative was sought.

To change the receive mode to ASCII the command below is sent to the modem. This command is sent during modem configuration, as it is important to read data immediately once received in order to avoid buffers filling up and data loss.

```
AT+SDATARXMD=1
OK
```

When data is available from the receive buffer, the following status code will be sent from the modem.

```
+STCPC: 1
```

This should be caught and the data is then read using:

```
AT+SDATAREAD=1
+SSTR:1,testmessage
```

The return message can then be extracted out of the returned string from the modem. It is important that the string message and only that message is sent, as any encapsulation inside objects from the Java server will result in an inability to read the string, even though the data is received.

6.1.4 Arduino Control and Flow Diagram

Arduino code was written to construct AT commands and control the modem to configure, send and receive data.

A method was written to read in the messages returned from the modem. The returned message can then be checked against the message codes expected for a successful operation before moving forward with operation. This method can then be reused when receiving data from the server. The method reads bytes in the Arduino's receive buffer into a new string variable, until a new line character is encountered. This is the character that is always sent at the end of a modem return message. The String method trim() was used on the received string before returning to ensure that white space doesn't cause issues with recognising codes.

When sending a command to the modem, the main algorithm used is to send the command to the serial port, wait a short amount of time, read the next message available from the modem, if it is the expected success code move on, if not resend the message and loop until the expected

message is returned. This solves some issues with the modem returning error messages, when the repeated command is sent; often there are no repeating errors.

The Flash library [40] is used to store the long string with charging station and environment data in the flash memory on the Mega, which is large enough. This library allows large strings to be constructed, stored and smaller strings extracted to deal with underneath the limit imposed by the Mega's small SRAM size.

As the SSTRSEND method of sending data is limited to 1000 bytes at a time, an algorithm was developed to split the charging station data into smaller packets of data, which could then be sent using a send method developed using the AT commands detailed in section 6.1.2. The algorithm runs through the string 1000 characters at a time, until there are less than 1000 bytes left in the string, at which point it appends a ';' character to the remainder of the data, and then sends it, therefore terminating the transmission at the server.

During testing erratic behaviour was observed when extracting data from the string stored in flash memory. This was due to the maximum packet size of 1000 bytes being larger than the Arduino could cope with in SRAM memory. The packet size was reduced to a maximum of 300 bytes, and the long string of information could be successfully received on the server.

It was observed that strings of only 50 bytes could be received at a time from the remote server, so a similar system to the sending of information was adopted. The message was split up on the server into small packets of 50 bytes, with the final one terminated with a ';' character. The Arduino code listens for the signal that information is available, and parses the data into a string. This section loops until the end of file character is received, signalling that no more data has been sent from the server, so the final string can be returned.

The left hand side of Figure 22 shows the control flow for the Arduino code.

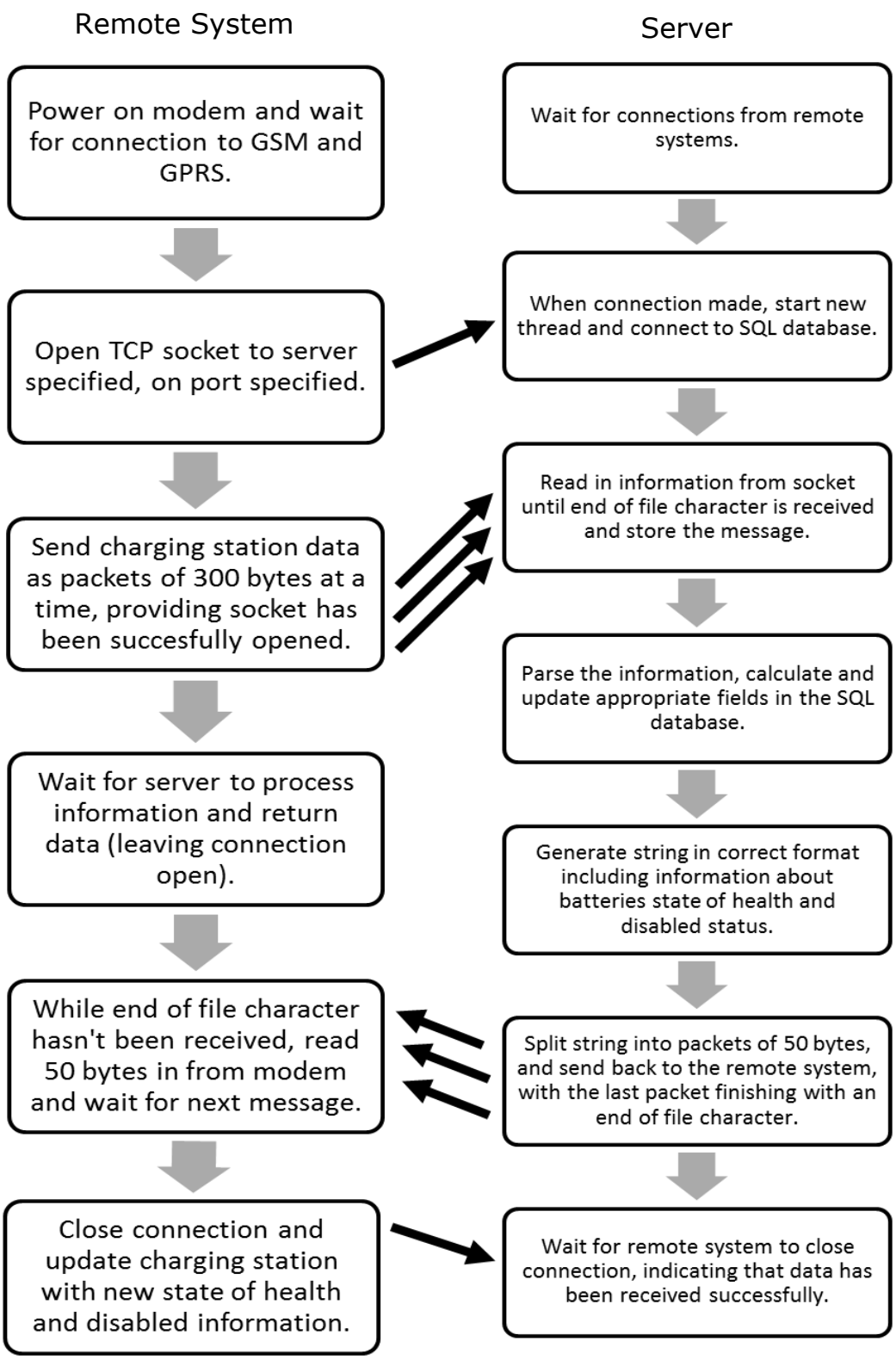


Figure 22|Process Flow Diagram

6.2 Java Server

The operation of this system is based on a standard multithreaded Java server, following the flow diagram in Figure 22. Using multithreading allows the server to process multiple connections at once, without a new connection having to wait until the processing has finished for the last connection. This means that it is possible to receive data from multiple charging stations at once. Testing was carried out using multiple Java clients to ensure that multiple connections could be handled, however it could not be tested using multiple hardware prototypes due to budget limitations.

Once a new connection is detected on the ServerSocket class, a new thread is started with the client socket passed as a parameter. This allows any new connection to be handled as a new client socket. The Java Database Connector detailed in section 6.3.1 is used to connect the Java application to a MySQL database running on the localhost. Once successful, the server starts waiting to receive data from the remote system.

Once send and receive operations have been carried out, the server waits for the remote system to close the connection. Closing the connection automatically causes data to be lost if the modem has not received all the information in time. Leaving this to the slower device (the microcontroller) is a solution to this problem.

6.2.1 Receiving Data

Initially the Java class BufferedReader was used to read information in from the socket, using the readLine() method. This method blocks until it receives a new line, '\n', or a carriage return, '\r' character. Issues were found with this, as the method for sending from the modem does not send new line or carriage return characters meaning that even though data was received, this method would block indefinitely.

Instead the decision was made to use a more modern version of BufferedReader, the Scanner class. This class allows the data to be read until a specified character is received, in this case the ';' end of file character. This proved to work, as a 23kB message could be received from the modem and parsed using string methods without any issues.

6.2.2 Sending Data

Once the information has been parsed and the database updated using SQL queries (section 6.3), a string can be constructed in the format required to send back battery state of health information, along with the disabled status from the server.

As discussed in section 6.1.4, the modem is only capable of receiving messages shorter than 50 bytes at a time. The same algorithm used to split up the data string sent from the charging station was used to split up the return string, albeit this time written in Java.

The PrintStream class was used to send the string back to the socket each time, with the last message terminating with the end of file character ';'. When using classes to send information back to the socket, it is necessary to check that the class is not encapsulating the information within a Java object. Even though this is fine when communicating with standard Java clients, the modem does not have the capabilities of decoding this format. A standard class should be used that writes the raw bytes to the socket, and therefore the modem can read them without issue.

6.3 SQL Queries

The SQL database forms the backbone of the entire prototype, allowing data to be managed efficiently, and values to be passed to and from various systems in a manner that is flexible. Data is found within the database by using query statements. These queries can be constructed and sent to the database in many languages, in this project queries are executed using PHP and Java. MySQL is built into modern versions of PHP, making it very easy to carry out operations on the database. The Android application calls a PHP file on the server, which then carries out the queries required and returns the information.

The standard query format is shown below:

```
SELECT x FROM y WHERE z='true' AND w=5 ORDER BY DATE(v)
```

This returns all the values of column x from table y that column z is true and column w is equal to 5. The results are then ordered by the date values in the column v. Various data types are available, including integer, string (varchar), double and date and time formats.

Many SQL queries are constructed and carried out in this prototype, and are self-explanatory thanks to the human readable syntax.

To change information in a table row, the syntax UPDATE is used as below:

```
UPDATE y SET z=8 WHERE t=1
```

This updates the table y, setting the column z to 8 where the column t is equal to 1.

To insert information into a database the INSERT statement can be used:

```
INSERT INTO x (y,z) VALUES(8, 9)
```

This inserts a new row into table x, with columns y and z being assigned the values 8 and 9 respectively. It can be seen that using these queries can allow complex data results to be returned and the database manipulated programmatically. SQL statements are used within the Java server to update and insert new values into the database, and on the web application to display live data on screen.

6.3.1 Java Database Connector

To connect a Java application to a MySQL database, the Java database connector (JDBC) is used [41]. This is an executable jar file that can be downloaded, and added to the project build path in Eclipse by right clicking on the project, selecting build path, configure build path, libraries and then add external jars. This is then linked to the database running on the localhost on XAMPP; however it is possible to use an IP address and port to connect to a database running on a remote server if required. The username, password and database name are also required to set up a connection.

6.4 Web Application

A content plan was designed for the landing page for the website. This was required to set the tone for the website, and show what and how the system works in a clear manner. A large 'smartSOLAR' logo, a clear button link to the login page along with pictures and information about how the system operates are all included on this page.

The plan is shown in Figure 23.

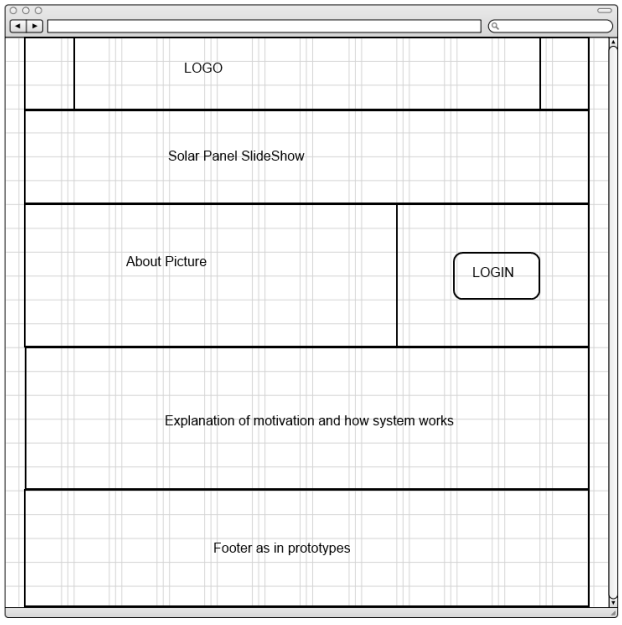


Figure 23 | Landing page mockup

This page was then constructed as seen in Figure 23 using the JQuery cycle JavaScript plugin. This executes on document ready in order to fade between the images describing the operation of the system. A CSS button was implemented, with a slight gradient that inverted on mouse hover, and position moving down when clicking on it by one pixel, giving feedback to the user. Four paragraphs were included to discuss the project motivations in more detail.

Figure 24 | Landing page

Other pages were designed in a similar way. The summary page is the first page that the user sees after logging in, and therefore shows a list of all locations monitored, and when they last updated the database. An embedded Google map is shown to give an idea of where the charging stations are updated. At the moment this system has to be updated manually, but future work could be carried out into updating this map based on the locations stored in the database.

The design plan is shown in Figure 25, and the constructed page is shown in Figure 26.

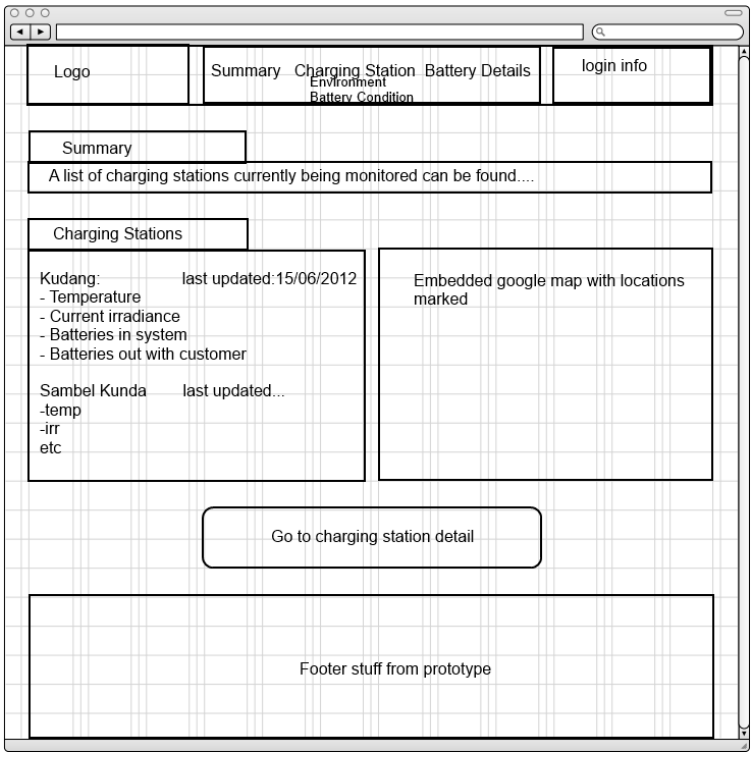


Figure 25 | Summary page mockup

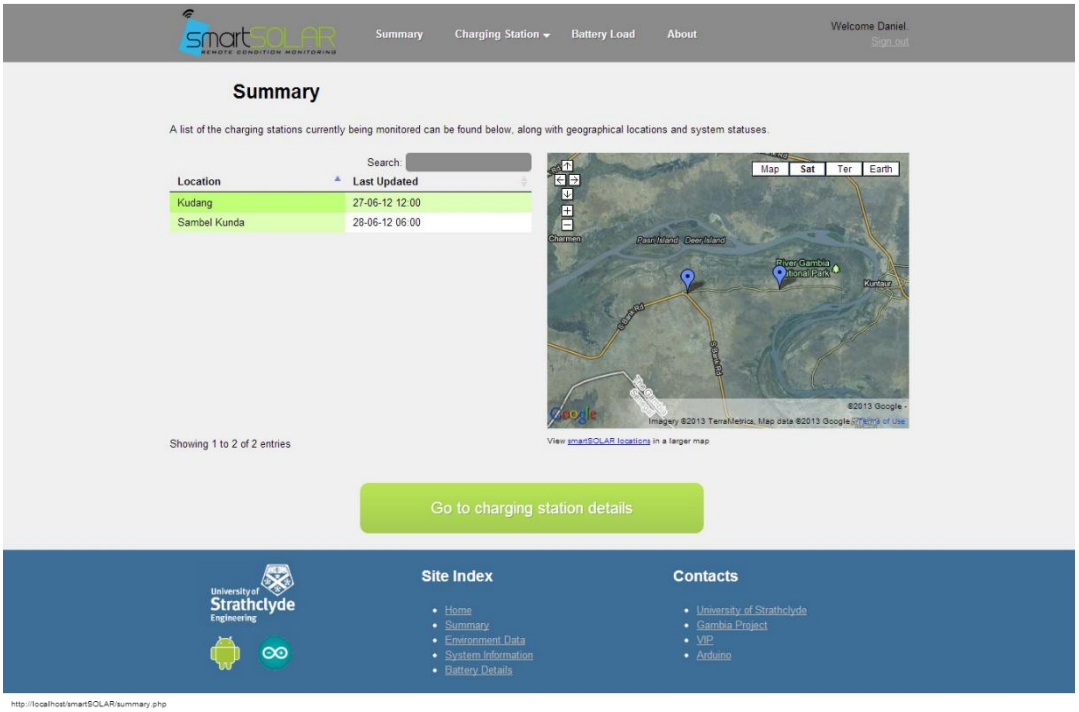


Figure 26 | Summary page

The datatables JQuery plugin was used to skin a HTML table populated through PHP executing MySQL queries to read the location information from the database. The dates were made more human readable by using the strtotime function in PHP. This function outputs the time in milliseconds (since January 1st 1970) and then the date format functions are used to format the date in a suitable manner.

A free CSS menu was sourced online [42], and skinned to match the site colour scheme. This CSS styles the list of links at the top of the page so that the second level of links is only visible when hovered over, giving a dropdown menu effect without using JavaScript.

The environment page was designed and constructed using the 960.gs framework, and a JQuery date picker to a HTML text input box [43]. This allows a box to pop up and the user to select a date for a report to be generated without having to enter the date in manually in the correct format. The maximum date on the date picker is initialised to a new JavaScript Date object, which initialises as the current date – meaning that all dates in the future cannot be selected. The fields' altDateFormat and altField where used to allow the displayed date to be more human friendly, with the values passed to the database are already formatted as a SQL datetime.

A dropdown box is constructed with a SQL query for all the locations and the last time they were updated, therefore avoiding hardcoded values in the site, which would hinder maintenance.

HTML divisions are only loaded when the button "Load data" is pressed using the PHP line:

```
IF($_SERVER['REQUEST_METHOD'] == "POST"){
```

A Highcharts chart is then initialised and data acquired by using an AJAX call, $.getJSON, with the targeted PHP file opening the database, reading the results into an array, and printing the JSON encoded result. The date and location parameter is passed to the PHP file by reading the HTML POST variables, and then using the format shown below within the PHP file call.

```
environmentJSON.php?locParam=LOCATION&dateParam=DATE
```

The values passed to the file can be accessed within the PHP file using the syntax GET['locParam'].

Two y axes are declared with different scales for temperature and irradiance, and the second set of results in the JSON string was assigned to the second y axis using the JavaScript syntax shown below in the chart initialisation.

```
options.series[1].yaxis = 1;
```

PHP code was written to display maximum, minimum and average temperatures and irradiance for the specified day.

A JavaScript Highstocks [44] chart was also added to show all historical data and trends for the selected location. PHP structure was added so that if the user leaves the date selector blank, only historical trends are shown.

The completed page is shown in Figure 27.

Figure 27 | Environment page

The system information page is for operators of battery charging stations, and as such displays asset condition information in a simpler, easier to digest fashion, with large graphics to show the operation of the system. A dropdown box is generated as with the environment page. Once clicked, a report is generated about the selected location.

A design plan was developed as with the other pages to understand what content was to be displayed. The reports include a Highcharts pie chart showing the proportion of batteries in set bands of condition and a column graph showing the state of health of the worst ten performing batteries. These results are generated using AJAX calls to a PHP file, which again JSON encodes the result array in the format required for the charts.

Also included are summaries of the asset base usage, conditions and relatively simple load information. At the bottom of the report is a searchable table allowing the operator to call up information about a specific battery, for example whether it is disabled, with a customer or its state of health.

The completed system information page is shown in Figure 28.

The final page is aimed at system developers with technical knowledge. Information such as charge levels, load information and times of usage is presented, along with the ability to disable a battery manually if a data is seen that makes a battery unsafe.

All batteries are listed in scrollable and searchable table. The developer can click on a battery row, which then offers feedback by changing the background colour to confirm selection, and then click "load battery data" to load technical information about the battery.

The label on the disable/enable button is changed based on the battery's disabled status read from the database, and the tick/cross image is also based on this value to give immediate feedback as to the battery's status.

When clicking the disable/enable button, a new page is called, with the battery ID to be updated as the POST variables. This page then updates the database, and gives feedback to the user as to the success of the operation. If the battery's state of health is below 10% the system has automatically disable the battery, therefore it cannot be re-enabled from the web interface.

It is necessary to POST the current disabled status through to the new page along with the ID, otherwise refreshing the page would toggle the disabled status, resulting in confusing behaviour to the user.

The technical battery reports include a Highstocks chart showing charge level and average load over time, and a table of all historical load samples for that battery.

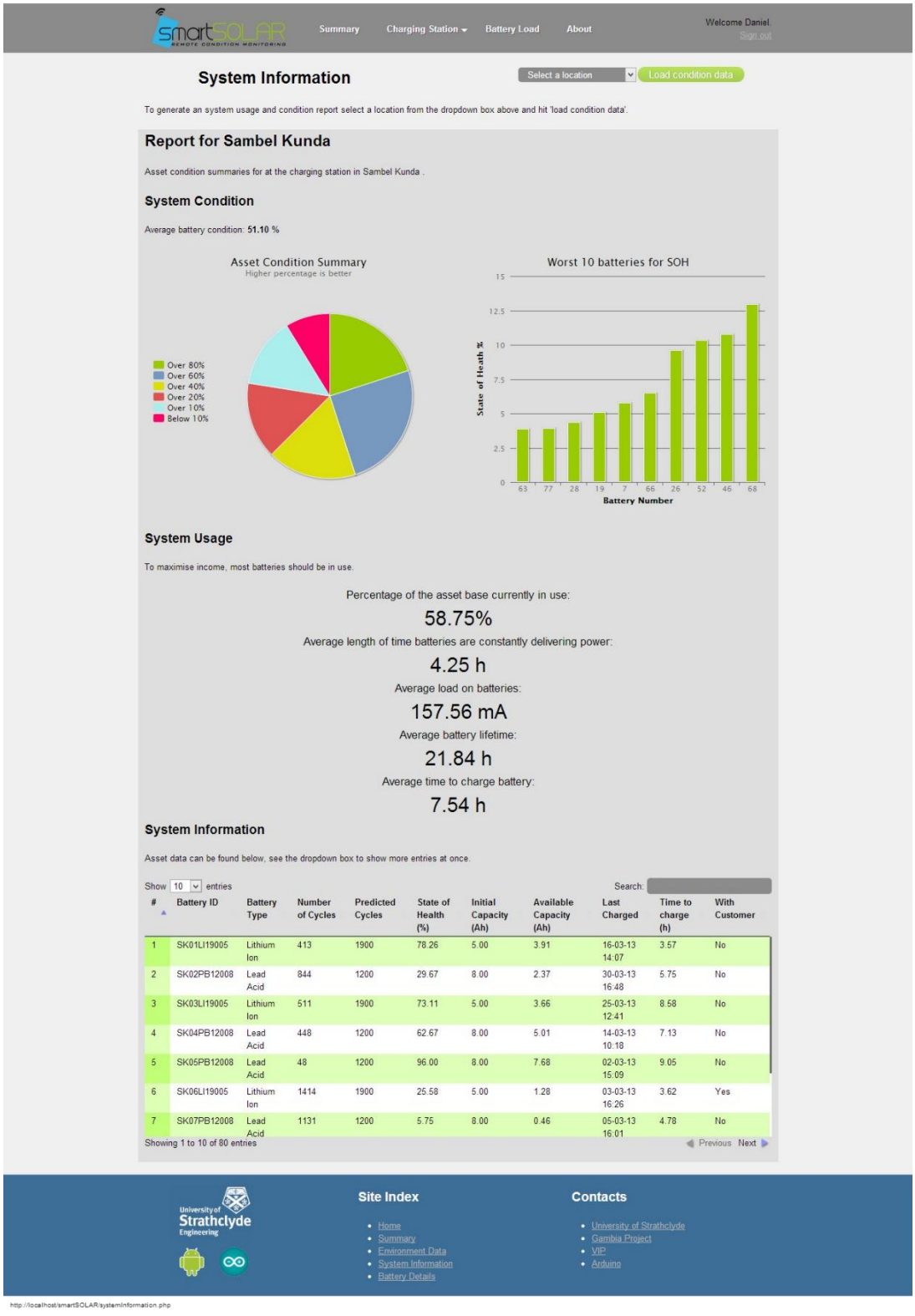


Figure 28|System Information Page

The technical battery information is shown in Figure 29, with the confirmation page that shows when a battery's disabled status is changed shown in Figure 30.

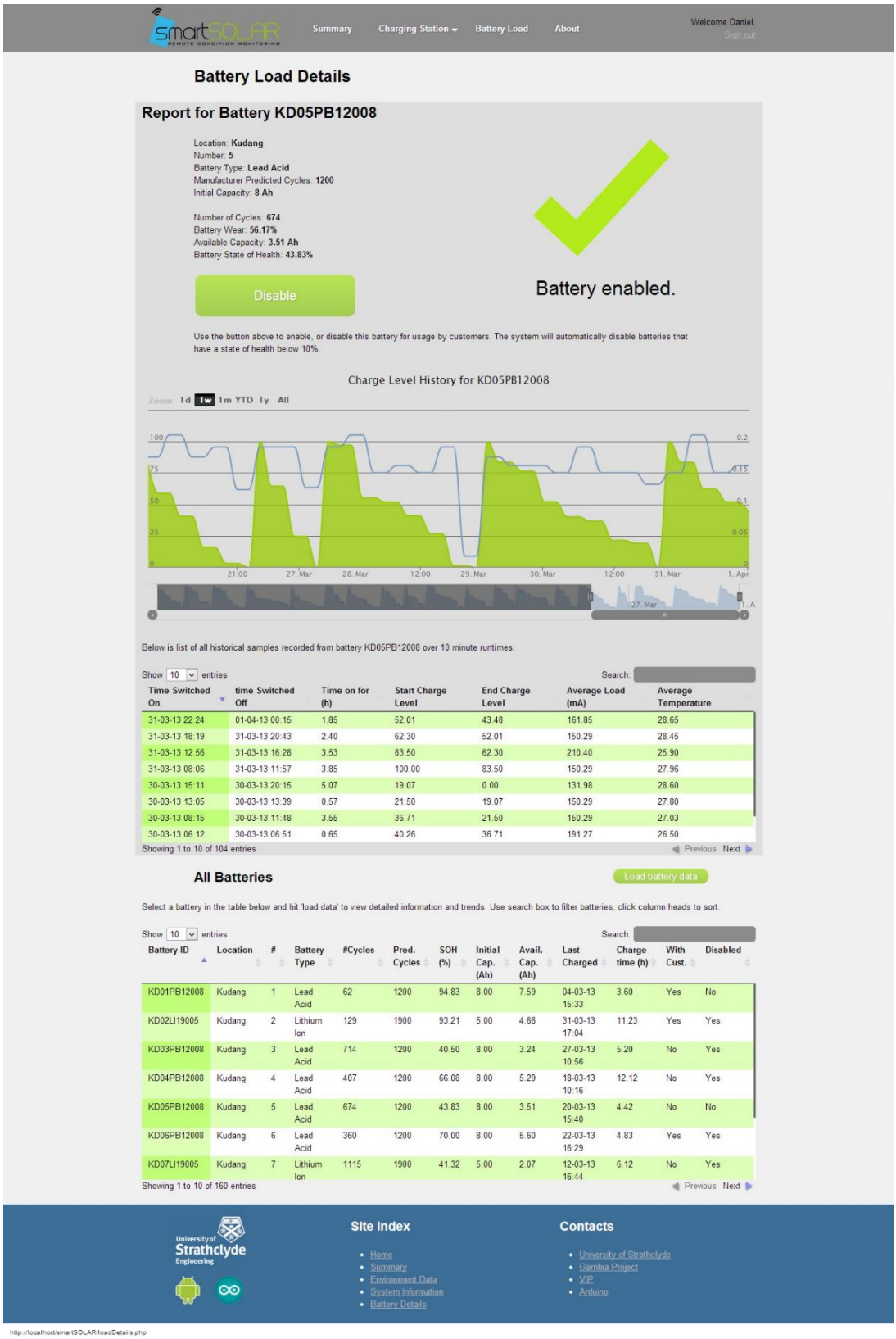


Figure 29 | Battery Load Information page

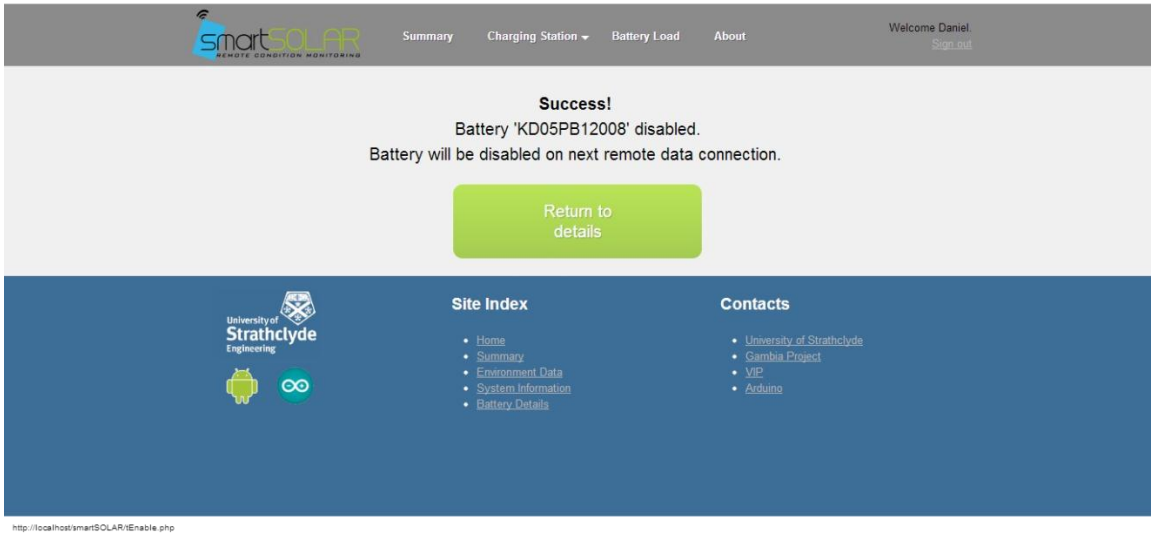


Figure 30|Disabled Confirmation page

6.4.1 Security

As this site will be hosted live on the internet, some level of security is required to prevent unauthorised users from accessing the system. A login system was implemented using PHP and a MySQL table to store the usernames and passwords of registered users and their level of approval. The default level of approval is 0, which does not let the user login to the site, so an admin with access to the database needs to change the approval level before a new user can log in.

Usernames and passwords are sent to the server using HTTP POST variables, and are escaped before writing to the database to prevent SQL injections, where a third party could alter the database layout without permission through the text input fields..

Passwords are hashed using the SHA1 algorithm implemented by PHP; this prevents the plain text password from being viewed by an admin, or any intruder. This makes it more secure. The hashing algorithm is a one way function, so there is no way of obtaining the plain text password given the hashed password. In order to check passwords, the given input is also hashed, and the hash values compared.

The login and register pages were skinned using CSS to appear consistent with the rest of the website. Error messages are returned from the server if the users do not match, and confirmation messages shown when registering. HTML5 input boxes are used to give feedback to the user if boxes are left empty or confirmation passwords do not match for example, and the user is alerted before the values are sent to the server.

The login and register pages are included in Figure 31 and Figure 32 respectively.

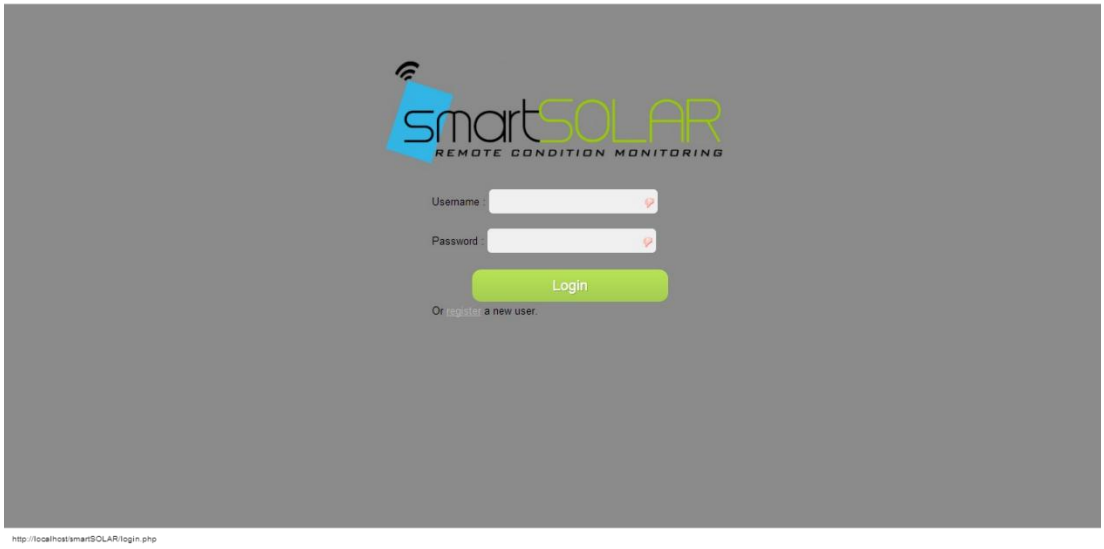


Figure 31 | Login page

Figure 32 | Register page

6.5 Mobile Application

The PhoneGap framework was used to develop for Android devices, using the AppLaud plugin from Mobile Developer Solutions [45]. This is an Eclipse plugin to allow development using web technologies such as HTML, CSS and JavaScript.

The Barcode Scanner PhoneGap plugin [46] was imported into the project and the appropriate JavaScript was linked within the main page.

The first page offers a large button to start the barcode scanning operation. Once found, the resulting text is passed to the next page using the hash URL notation (test.html#foundtext). This can then be parsed by the JavaScript on the second page, and an AJAX request is sent to a PHP file on the server given the IP address of the server. The androidJSON.php file then

returns a summary of non-technical battery information to the device over any internet connection available.

The flag 'async:false' is used in the AJAX request to make the function wait until data is returned from the server. This is required so that the HTML page can be rendered after the information is available.

Some error checking was carried out on the application to only allow text strings form QR codes to be sent to the server, so encoded URLs and different barcode types show an error message. If the batteryID (or text string) is not found in the database, a null message is returned so that the mobile application can show a relevant message.

The application was skinned to provide a consistent interface across all the devices as shown in Figure 33. Highcharts were tested within the application, and was proved to work, however to keep the application as simple to use as possible this was not included.

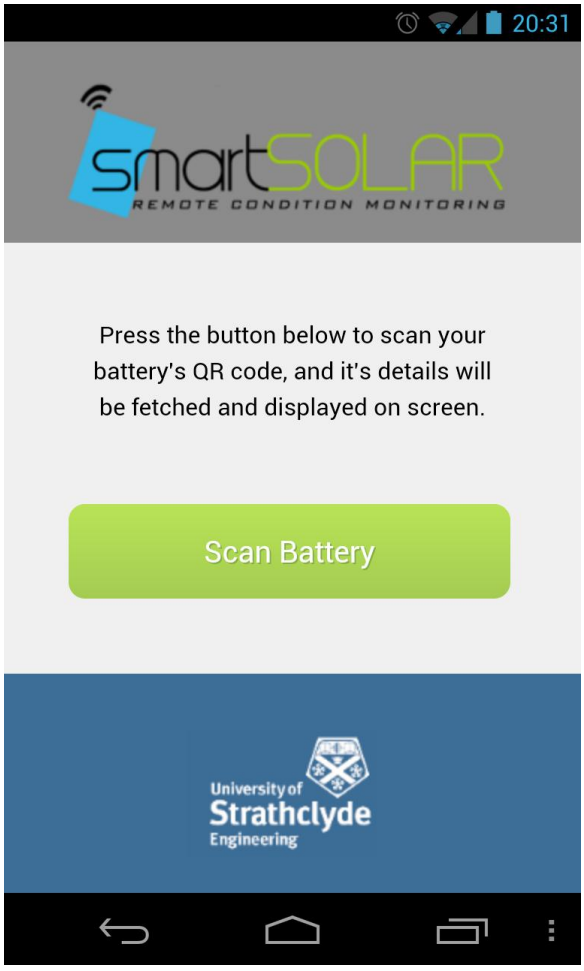


Figure 33|Android App Main Screen

A QR code with the battery ID 'KD01PB12008' encoded in it is shown in Figure 34.

Figure 34|QR Code with the ID KD01PB2008 embedded.

6.6 Development & Prototype Costs

See Table 2|Development CostsTable 2 for the table of costs and equipment required to develop the prototype. Note that the Arduino Uno was purchased initially, but replaced with an Arduino Mega when the Uno did not meet requirements through testing.

Table 2|Development Costs

Product	**Cost**	**Supplier**	**Part Number**	**Reason**
Cellular Shield	£59.55	Proto-Pic	PPCEL-09607	Connecting Arduino to Cellular Networks
Quad-band SMA Antenna	£3.95	Proto-Pic	PPWRL-00558	Antenna for the cellular shield
Arduino Headers	£1.19	Proto-Pic	PPPRT-10007	Allows stacking of shield directly on Arduino
Arduino Uno	£17.10	RS	715-4081	Control for remote system
Arduino Mega	£30	RS	715-4084	Replacement control for remote system
SIM card with data plan	£25.00	T-Mobile		Allow use of mobile networks for data upload
Bench Power Supply	£0.00	University of Strathclyde		Powering prototype.
LCD Screen	£0.00	Personal		Showing operation of modem.

Windows machine	£0.00	Personal		Running the Java and Web server on an accessible network.
Google Nexus 4 Android Phone	£0.00	Personal		Android Application development.
Total development cost	**£136.79**			

Even with unforeseen issues like the microcontroller not meeting requirements, this is minimal extra cost when compared with the usual cost of a solar installation.

7 Testing

Testing was carried out throughout implementation, and also during prototyping to ensure that such a design was possible. Simulations were carried out to assist in design of the system, and realistic data was generated so that the various applications could be developed.

This dummy data was created using small Java applications to generate appropriate SQL statements which could then update the database through the Java Database Connector. This code was then adapted to generate dummy charging station data to test the remote system, with variables taken in at the start to specify from when and how long the data should be generated.

The web application was tested to work across most recent iterations of browsers: Google Chrome version 26.0.1410.64 m, Mozilla Firefox version 20.01 and Microsoft Internet Explorer 9. Older versions of browsers may cause issues, especially with client side JavaScript and modern CSS styling with HTML5.

7.1 Simulations

Initial simulations were carried out in Java to determine whether having assistance in the distribution of portable batteries affects the life span of the batteries. A framework was built to allow batteries to be created, and cycles simulated until they reached their recommended number of cycles by the manufacturer.

The system was regarded to have failed when there were no longer enough healthy batteries to serve the entire village.

Tests were carried out with different algorithms:

1. A random allocation of battery to customer when multiple charged batteries are available.
2. Allocation based on the best state of health, so there is an even wear across the system.

These results showed a consistent improvement in asset life of 3-4% using algorithm 2. It was proposed that the random allocation in algorithm 1 was actually best case, with the technically inexperienced operator more likely to overuse the same batteries.

This led to the conclusion that implementing a two way communications, with the system calculating battery state of health was a good objective. Furthermore any transfer of education from technically experienced developers to inexperienced operators in the efficient operation of the system is useful in the long term.

8 Future Direction

As discussed previously, this project has been running in parallel with areas of the Vertically Integrated Project. The VIP continues every year, so the next step for this project in the immediate future is to integrate the systems developed in both projects.

In addition, the system is to be adapted to operate on the Solar Residential Systems installed by the Gambia Solar Project in recent years, and will be tested in the environment for which it was designed this year. Before this can be possible however, the system has to be made much more robust, both physically in an enclosure and be made able to deal with connection dropouts during transmission.

8.1 Raspberry Pi

A Raspberry Pi minicomputer has been made available as part of the project, and the plan is to run the Java, SQL and web servers on this as part of a LAMP (Linux, Apache, MySQL and PHP) installation. This will be run within the university, and will be assigned a static IP address, removing the need to update the hardcoded addresses within the remote system and the Android application. Since this computer has a very low power consumption, and is very cheap to purchase, it fits the requirement for the entire system to be as low cost as possible.

8.2 State of Health Calculations

Many assumptions are made in the calculations of the state of health of batteries, and many more variables are involved in accurately assessing their condition. More research is required into this, and how much of a difference there is between simply counting number of cycles, and getting more information such as depth of discharge, or internal resistance measurements.

This is a huge research area, and outside the scope this project. However the framework is designed to be very easily modified to change thresholds, or add more information gained from batteries.

8.3 Financial Options

If the charging station operator operates a fee-for-service model, where the users pay for electricity on a pay as you go system, it can be seen that having a communications framework in place can facilitate this. A user could scan their battery to see how much money is left on the battery, and the operator can see the usage of the battery and bill accordingly. There is not a huge gap between the functionality offered within this project, and adding a financial model.

8.4 Security Issues

While a login system was implemented, hashing passwords as required, it is noted that there are several security holes in the prototype.

No login system was included in the Android application to make the application as easy to use as possible for extremely non-technical users. There aren't huge security concerns, as any battery hardware has to be present for the application to be useful. In the future such a system can be developed, with the approved level 1 in the database relating to users, if for example the batteries are assigned to users, and only batteries that are assigned to the logged in user can be scanned.

All usernames and passwords are sent to the server using a HTTP POST, in plain text, so any intruder using packet sniffing software can intercept them. The solution to this is to use SSL/HTTPS protocols; however this needs to be set up on the server, and is outside the scope of this project.

For ease of setup and demonstration, the database has been left with the default username and password, 'root' and a blank password. This will be required to be changed as soon as the website is deployed.

9 Conclusion

Due to the rural situation of many of solar installations in developing countries, the lack of road infrastructure makes long-term maintenance of the system very difficult, requiring technicians to spend days travelling to carry out repairs. Furthermore, many solar installations are donated by charities or non-profit organisations in order to improve the villager's quality of life, however the lack of technical experience of the users often leads to system misuse, lowering the system's lifespan. Finally the weakest component within solar installations has been identified as the batteries used that store power when no sunlight is present. Monitoring and managing the condition of these batteries is important for the long term sustainability of the system.

Condition monitoring has huge benefits when applied to solar installations in developing countries. GSM networks have a very high penetration in Africa, and as such can be used to facilitate data transmission from very remote areas. Using GPRS technologies on top of the existing GSM networks allows packet based data to be sent across the world. This means that a remote system can communicate with a system that can be maintained by technically experienced developers. Opportunities are therefore available for international developers to design solar installations for rural areas, but then also manage the operation of the system without being physically present. Any data gathered will be invaluable in the design of future systems, and can also be used to improve the lifespan of the installation.

Using a system such as the one developed during this project can allow assistance to be given to an operator of a battery charging station, and more control is gradually handed over from the developers as the operator becomes more proficient in the management of their assets. This significantly improves the long term sustainability the installation.

The prototype system uses many different languages and tools at different levels of abstraction, from gathering data from batteries to web design, highlighting its inter-disciplinary nature. Both top-down design, where the functionality is defined and available technologies are used for implementation, and bottom-up design, where technology has been made available allowing the functionality to be realised, have been used within the project. An example of this is the main requirement for the need for remote condition monitoring having been developed through the course of the Vertically Integrated Project last year, and the availability of GSM networks allowing this functionality to be realised.

10 References

[1] F. Niewwenhout, A. v. Dijk, P. Lasschuit, G. v. Roekel, V. A. P. v. Dijk, D. Hirsch, H. Arriaza, M. Hankins, B. D. Sharma and H. Wade, "Experience with Solar Home Systems in Developing Countries: A Review," *Progress in Photovoltaics: Research and Applications,* vol. 9, pp. 455-474, 2001.

[2] Energy Saving Trust, "Feed-Tariffs scheme (FITs)," March 2013. [Online]. Available: http://www.energysavingtrust.org.uk/Generating-energy/Getting-money-back/Feed-In-Tariffs-scheme-FITs. [Accessed 26 October 2012].

[3] SMA, "Monitoring Systems," [Online]. Available: http://www.sma.de/en/products/monitoring-systems.html. [Accessed 26 November 2012].

[4] Fat Spaniel, "How It Works," Fat Spaniel, [Online]. Available: http://www.fatspaniel.com/fat-spaniel-in-action/. [Accessed 3 November 2012].

[5] IDEO, "Human-Centered Design Toolkit," IDEO, 2009. [Online]. Available: http://www.ideo.com/work/human-centered-design-toolkit/. [Accessed 30 March 2013].

[6] Medic Mobile, "Impact," [Online]. Available: http://medicmobile.org/impact/. [Accessed 20 March 2013].

[7] FrontlineSMS, "Frontline SMS: Using mobile technology to promote positive social change," [Online]. Available: http://www.frontlinesms.com/. [Accessed 16 February 2013].

[8] Ushahidi, "The free and open source SMS gateway for Android," 20 March 2013. [Online]. Available: http://smssync.ushahidi.com/. [Accessed 27 March 2013].

[9] I. Holeman, Interviewee, *Co-founder, Medic Mobile.* [Interview]. 29 March 2013.

[10] N. Schelling, M. J. Hasson, S. L. Houng, A. Neveraz, P. W.-C. Lu, M. Tierney and L. S. H. Schutzeichel, "SIMbaLink: Towards a Sustainable and Feasible Solar Rural Electrification System," SIMbaLink, New York, 2006.

[11] SIMbaLink, "Using mobile-phone technology to make solar a better energy solution," [Online]. Available: http://simbalink.com/. [Accessed 7 March 2013].

[12] N. Wolfe, "An Open-Source Monitoring System for Remote Solar Power Applications," 12 November 2009. [Online]. Available: https://code.google.com/p/solar-monitoring-project/wiki/Motivation. [Accessed 5 November 2012].

[13] Water for People, "FLOW," [Online]. Available: http://www.waterforpeople.org/programs/field-

level-operations-watch.html. [Accessed 15 December 2012].

[14] Water for People, "FLOW Application," [Online]. Available: http://watermapmonitordev.appspot.com/. [Accessed 4 10 2013].

[15] R. Sims, "Remote Energy Monitoring in the Developing World," University of Strathclyde, Glasgow, 2009.

[16] Osiris Development, "Battery Bar Pro," 2012. [Online]. Available: http://batterybarpro.com/. [Accessed 5 February 2013].

[17] M. Hart and T. Igoe, "SoftwareSerial Library," Arduino, 5 May 2012. [Online]. Available: http://arduino.cc/en/Reference/SoftwareSerial. [Accessed 15 April 2013].

[18] Arduino, "Memory," 2011. [Online]. Available: http://arduino.cc/en/Tutorial/Memory. [Accessed 6 February 2013].

[19] F. Hillebrand, "Milestones of the GSM/UMTS Development," 3gpp, 2007.

[20] O. Hesselmark and A. Engvall, "Internet for Everyone in African GSM Networks," Scanbi-Invest, Stockholm, 2005.

[21] M2M Support, "A Guide to Module Selection," [Online]. Available: http://m2msupport.net/m2msupport/a-guide-to-module-selection/. [Accessed March 2013].

[22] FreeGSMPlus, "Gambia GSM Coverage," [Online]. Available: http://www.freegsmplus.com/gsm-coverage/gambia.html. [Accessed 8 November 2012].

[23] M2M Support, "Carrier Search - Africell," 2012. [Online]. Available: http://m2msupport.net/m2msupport/africell-gambia-m2m-modules-certification-sim-data-plan/. [Accessed March 2013].

[24] M2M Support, "GCF Introduction," [Online]. Available: http://m2msupport.net/m2msupport/gcf-process-costs-timeframe-and-labs/. [Accessed March 2013].

[25] Proto-Pic, "Cellular Shield with SM5100B," [Online]. Available: http://proto-pic.co.uk/cellular-shield-with-sm5100b/.

[26] Spreadtrum, "SM5100B-D Specification," SendTrue, 200. [Online]. Available: http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Cellular/SM5100B-D_HW_spec_V1.0.0.pdf.

[27] J. Boxall, "Tutorial: Arduino and GSM Cellular – Part One," TronixStuff, 19 January 2011. [Online]. Available: http://tronixstuff.wordpress.com/tag/sm5100b/. [Accessed 2 November 2012].

[28] SEEED Electronics, "GPRS Shield," [Online]. Available: http://www.seeedstudio.com/depot/gprs-

shield-p-779.html?page=4&r_q=r.

[29] Cooking Hacks, "GPRS/GSM Quadband Module for Arduino," [Online]. Available: http://www.cooking-hacks.com/indexa.php/gprs-quadband-module-sim900-for-arduino.html.

[30] Teltonika, "Wireless COM/G10," [Online]. Available: http://www.teltonika.lt/en/pages/view/?id=16. [Accessed 15 April 2013].

[31] Apache, "CouchDB," [Online]. Available: http://couchdb.apache.org/. [Accessed 15 March 2013].

[32] Apache Friends, "XAMPP," [Online]. Available: http://www.apachefriends.org/en/xampp.html. [Accessed 12 November 2012].

[33] N. Smith, "960.gs," 2012. [Online]. Available: http://960.gs/. [Accessed 2 November 2012].

[34] Google, "Google Chart Tools," [Online]. Available: https://developers.google.com/chart/. [Accessed 26 November 2012].

[35] Highcharts, "Highcharts & Highstocks," [Online]. Available: http://www.highcharts.com/. [Accessed 7 Febrary 2013].

[36] Mockingbird, "Wire-framing," [Online]. Available: https://gomockingbird.com/. [Accessed 28 October 2012].

[37] Arduino, "LiquidCrystal Tutorial," [Online]. Available: http://arduino.cc/en/Tutorial/LiquidCrystal. [Accessed 17 March 2013].

[38] Buddy and Steven, "SM5100B-D AT Command," 25 December 2008. [Online]. Available: https://www.sparkfun.com/datasheets/CellularShield/SM5100B%20AT%20Command%20Set.pdf. [Accessed January 2013].

[39] S. Shao, "Using AT commands to control TCP/IP stack on SM5100B-D modules," 20 September 2008. [Online]. Available: https://www.sparkfun.com/datasheets/CellularShield/SM5100B%20TCPIP%20App%20Note.pdf. [Accessed 27 November 2012].

[40] M. Hart, "Flash," 28 January 2012. [Online]. Available: http://arduiniana.org/libraries/flash/. [Accessed 15 March 2013].

[41] Oracle, "JDBC Overview," [Online]. Available: http://www.oracle.com/technetwork/java/overview-141217.html. [Accessed 21 March 2013].

[42] CSS Menu Maker, "Make Awesome Menus," [Online]. Available: http://cssmenumaker.com/. [Accessed 25 March 2013].

[43] T. Kang, "Passing a Date Parameter to Highcharts line chart," 12 March 2013. [Online]. Available: http://blueflame-software.com/blog/pass-date-parameter-to-highcharts-line-chart/. [Accessed 15 March 2013].

[44] HighCharts, "Highstock API Reference," 2012. [Online]. Available: http://api.highcharts.com/highstock. [Accessed 26 March 2013].

[45] Mobile Developer Solutions, "AppLaud," 2013. [Online]. Available: http://www.mobiledevelopersolutions.com/. [Accessed 30 March 2013].

[46] PhoneGap, "Plugins," March 2013. [Online]. Available: https://github.com/phonegap/phonegap-plugins. [Accessed 31 March 2013].

11 Appendices

Presented here are summaries and extracts of information that supports the ideas and proposals in this report.

11.1 Comparison of financial models for PV installations in developing countries

It is generally accepted that models are country and economy specific, and every part of the model has to function, otherwise the system will not function. There are four main models that were analysed in the literature review [1]: donations, cash sales, consumer credit and fee-for-service.

Rental schemes were categorized under fee-for-service where maintenance and repair service are provided and under consumer credit otherwise.

11.1.1 Donations

Many modern projects use this method. This means that the donor provides the hardware free/almost free. Social objectives provide the main motivation for donations, trying to make the largest immediate impact.

Users who have donated to generally feel less involved and responsible for donated systems compared to systems that they have had to contribute a considerable part of the costs. Most donations are also limited to hardware only – resulting in a neglect of maintenance and service requirements.

Advantages:

- Low initial costs for the user (often zero)
- Rapid Deployment/Immediate impact

Disadvantages:

- Lack of user commitment
- Failure due to lack of maintenance as no one feels responsible for it.

Donation is a good model for bringing SHS to large amount of customers that cannot afford or be reached by commercial SHS or grid electrification. It can be concluded that a financial contribution of the user of a SHS provides an incentive for **sustained** operation of the system.

11.1.2 Cash Sales

Advantages:

- Easy financing

- High flexibility in consumer choice

Disadvantages:

- Many cannot afford to pay for systems up front
- After sales service is non-existent in many projects, due to dealers living in the city and the buyer in the countryside
- Tendency to buy undersized systems and cheap replacement parts to save money

Systems paid for in cash are liable for failure due to the tendency to go for cheap under designed systems with lower costs. There is therefore a need for standards for components and system designs. Reliable after sales service is also required to remedy failures quickly. For those with the lowest incomes, cash purchase is usually not feasible.

11.1.3 Consumer Credit

Due to negative experience with donations, and the need for finance, more structured delivery modes have emerged. For widespread deployment, credit schemes have been considered essential. Funding can come from banks or private companies.

Advantages:

- Investment costs are spread out over a number of years.
- Improves affordability for a large proportion of the population.

Disadvantages:

- Local banks and other financing institutions are often reluctant to provide loans for non-productive investments to rural population, who are generally perceived as **'non-bankable'**.
- Credit is very vulnerable to unpredictable external financial shocks, as shown in Sudimara in Indonesia (7000 systems in 2.5 years on a 4 year term).

Credit schemes can work reasonably well provided good quality systems have been installed. The repayment discipline is strongly related to the technical performance of the system. If the system does not work, people stop their loan repayments.

11.1.4 Fee-for-Service

This is a system where the user pays a small amount for a supply. This means that operation and maintenance is the responsibility of the supply company rather than the user. If the user decides to terminate the supply, they no longer pay the fee, and the service is removed. The equipment is owned by the operating company.

Advantages:

- Costs to the user are spread over a very long period (10 years or more) with costs being paid back by multiple users.
- Public utilities are used to operating on a fee for service basis

Disadvantages

- Public utilities are not eager to get involved with solar PV due to the high costs of rural electrification, and unfamiliarity with the new technology which is different from grid technology. This is true to some extent with the public utilities in this country (UK).
    - It is worth noting that the reference for this section [1] is old, and the story is different now with a number of companies making a profit from sustainable energy on a fee for service basis.

Examples of this model in operation:

Kiribati – Solar Energy Company (SEC)

- Converted to become a rural utility in 1989
- Set up districts (each having 50 -125 systems)
    - These are each serviced by one SEC technician
- Installation fee of A$50 with a monthly fee of A$10-50 (depending on system size) to cover cost of operation of system in full.
- SEC technician visits every month to collect fees and perform system checks
- Evaluation teams in 1998 and 2000 reported that this scheme had been successful.

Dominican Republic – "Soluz Dominicana" (established 1996)

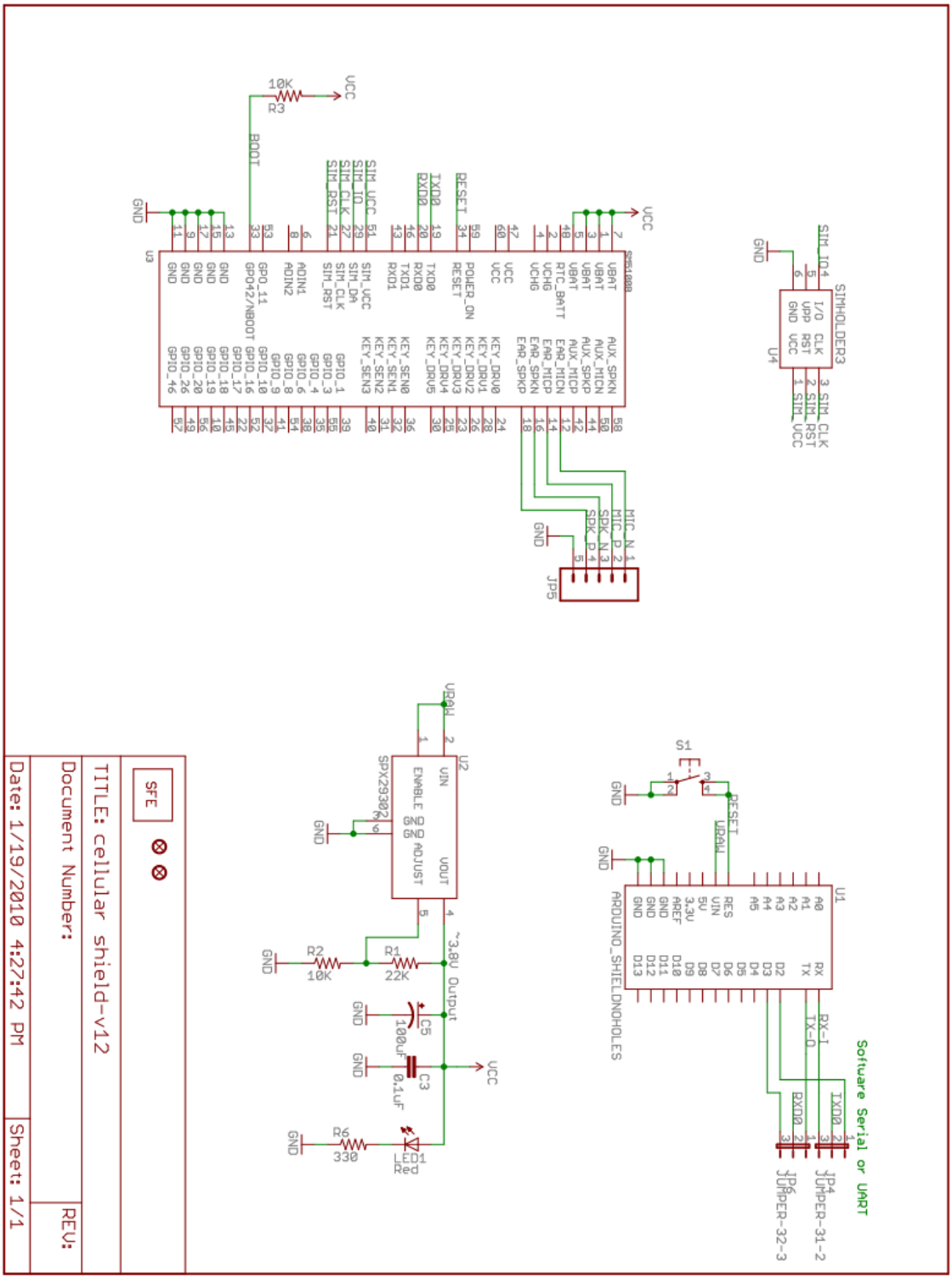
- Cash flow positive with 1000 units
- Credible repossession programme, non-repayment leads to quick and visible removal

Fee for service arrangements show opportunities to combine subsidies and market operation. The main challenge is to organise operation and maintenance of the systems and fee collection in a financially sustainable fashion.

11.2 Modem Shield Schematic

Available online at:

https://www.sparkfun.com/datasheets/DevTools/Arduino/cellular%20shield-v12.pdf



11.3 Developers Guide and CD contents

Presented here is a guide to the contents of the CD included with the report, so that code can be viewed. The CD contents are also available online at: https://github.com/chakkums/smart-solar.

The CD contents are as shown below:

Arduino

Contains all code used to control a 2G SM5100B modem, and send and receive data to a remote server.

Java Server

Multi-threaded server used to receive data from remote system, parse and update SQL database, and send messages containing asset condition back to remote system.

Web Application

Includes: JavaScript charts and tables to show data, PHP login/registration system, and user enabling/disabling of assets.

Android

Application developed using PhoneGap to scan a battery's barcode, and then show data loaded from remote server.

Testing

Files used for prototyping, and testing certain features of system.

11.3.1 Development Environments

Many development environments were used throughout the project, this section details their set up and configuration.

11.3.1.1 Java Development

Eclipse was used as the Java development IDE, this is available at http://www.eclipse.org/.

In order to connect Java applications to an SQL database, the Java Database Connector was set up following the tutorial available at http://docs.oracle.com/javase/tutorial/jdbc/basics/connecting.html.

11.3.1.2 Web Development

Within this project, Microsoft Expression Web 4 was used, as it was available through the university's MSDNAA subscription. Other development environments are available, such as

Adobe Dreamweaver. The website was coded from scratch, using the 'split' view to preview style changes.

As discussed in the report, XAMPP was used as the development server for the website. It is also possible to forward port 80 through the router, so the website can be viewed externally. XAMPP is available at http://www.apachefriends.org/en/xampp.html, and an installer is available for Windows machines.

11.3.1.3 Arduino Development

The Arduino IDE was used for this, available at: http://arduino.cc/, with the external Flash library required as discussed in the report. This library is available at: http://arduiniana.org/libraries/flash/, and can be installed by extracting the downloaded folder to the libraries folder in the Arduino installation directory.

11.3.1.4 Android Development

Eclipse was used for the Android development, with the Mobile Developer Solutions AppLaud PhoneGap plugin. Tutorials are available for the setup of the plugin at their website: http://www.mobiledevelopersolutions.com/. Android SDKs must be downloaded and can be found here: http://developer.android.com/sdk/index.html.

11.3.2 Set up of demonstration

XAMPP should be started, and PHPMyAdmin (included in XAMPP) used to import the prepared database, **smartsolar.sql.**

The website files need to be copied to the 'htdocs' folder in the XAMPP installation directory, and can then be accessed in a web browser by going to the address http://localhost/smartSOLAR/, or by replacing the keyword localhost with the server's IP address. The website files are found in the web application folder on the CD.

The server's public IP needs to be changed within the Arduino Mega code, near the top of the file MegaModemControl_v2_0.ino in the Arduino folder on the CD.

In a similar fashion, the Android application's server IP needs to be changed. This is done by importing the two projects found in the Android folder on the CD into Eclipse, as set up above. The IP address needs to be changed in the file:

Android/barcodeScanner/assets/www/batteryInfo.html.

Use of a static IP address would negate the requirement for these alterations.

The string that is sent to the server can be generated by using the Java application found in Testing/Create Dummy String, and altering the dates required within the source code. This generates random information in the correct format that can be parsed with the server.

The Java server can be run in the command line, providing the Java runtime environment is installed using the command:

```
java –jar javaServer.jar
```

This runnable jar file is found in the Java Server folder on the CD.