A low complexity, low infrastructure solution to per appliance Energy Monitoring

James Devine

School of Computing and Communications, Lancaster University

MSci (Hons) Computer Science

March 19, 2015

Abstract

Energy monitoring has come to the fore in recent years due to rapid climate change and as a result energy companies have begun distributing free home energy monitoring kits. Because of this, many homes have monitoring devices that provide an overall view of the energy consumption of a household, but do not have the ability to view energy consumption per device.

This project investigates a solution to the problem of per appliance energy monitoring using a low infrastructure, low cost approach, utilising the Earthed infrastructure of a building as a means of communication.

An application allows Users to interface with the data obtained by the system, as well as external price, energy and carbon information, with the hope of allowing Users to make informed decisions on the usage of their appliances.

The results of this project show that the Earthed internals of a buildings can indeed be utilised as means of communication, and that the reliability and dependability of such a system are high.

Working Documents `http://scc-intranet.lancs.ac.uk/Projects/2014/SmartMonitoring`

Must login with valid Lancaster University Credentials.

Declaration

I certify that the material contained in this dissertation is my own work and does not contain unreferenced or unacknowledged material. I also warrant that the above statement applies to the implementation of the project and all associated documentation. Regarding the electronically submitted version of this submitted work, I consent to this being stored electronically and copied for assessment purposes, including the Departments use of plagiarism detection systems in order to check the integrity of assessed work. I agree to my dissertation being placed in the public domain, with my name explicitly included as the author of the work.

Date: March 19, 2015

Signed: James Devine

Contents

1. Introduction & Motivation

The world is increasingly looking for new ways to become more aware about the amount of energy that is actively being used everyday. From a high abstraction level, energy can be monitored from power plants as the energy grid is actively feeding back information to controllers that determine how much energy should be generated at various times throughout the day.
However, the people that are responsible for drawing on the power from the energy grid aren't consciously aware of how much energy they are using in any detail. The only detail that the average energy conscious household owner has is an indication of how much energy is being used as a **household** - not per **device** or **appliance**.
Being aware of how much energy one is consuming has benefits for both a persons economy and the environment on a global scale. Lowering energy consumption would also be beneficial to the National Grid and the energy companies, as they are looking to conserve energy as resources become scarcer.
There are existing products that allow users' to break down energy consumption per device, but none of these existing products are both concealed from sight and are simple to setup and use. Other existing products provide accurate data without the ease of accessibility, whilst some provide accessible less accurate data. Many of the solutions are quite expensive for the features they offer, which is a barrier to consumers.
There has also been a lot of research into per appliance energy monitoring, but in most cases the end result was a bulky device that would intrude in many environments.
The methods of communication used by many of the research projects and existing products require a high level of infrastructure and setup.
Therefore, there is a need for a solution that provides a low cost, low infrastructure solution to per appliance energy monitoring.

1.1. Existing Products

There are a few products that already exist in the market that allow users to become more energy conscious. In this section these solutions will be analysed listing the pros and cons of each, and conclude what qualities an ideal solution would have.

1.1.1. Meter Plug

Meter Plug [44] is a crowdfunded idea to provide real time information to your Smartphone in order to aid energy consciousness. The device works by acting as a proxy to the main socket, the appliance requiring monitoring is plugged into the meter plug, which in turn plugs into the main socket. The user then receives real time updates via Bluetooth to their smart phone.
The design of the plug is minimalist, but this doesn't detract from its size: 37mm X 37 mm X 29mm. In figure 1 you can see the app and the plug side by side. The key aim of the Meter Plug is to save money. Along with the real time power consumption figures, the user

is presented with real time estimations of how much money is being spent per hour for the appliance.
The Meter Plug states that you can have more than one plug in the household, and the user is able to remotely turn appliances on and off remotely from up to 100 metres. There is also the ability to automatically turn off an appliance if it is in Vampire mode, where the appliance is consuming power even when it isn't being used.

Pros

- Real time - The Meter Plug allows real time data to be beamed directly from the plug.
- Easily accessible - The data supplied by the plug is easily accessible through smart phones, and the data displayed by the app is easy to understand and digest.
- Control over appliances - The Meter Plug enables the user to remotely turn appliances on and off.

Cons

- Bulky - The Meter Plug bulks up existing sockets, it isn't discrete
- Expensive - The Meter Plug prices up at £40, which for one plug is extremely expensive.
- Wireless - The Meter Plug uses Bluetooth LE, which has a range of around 100 metres in an open environment. However, in a household with various other technology and building materials, the range will be severely reduced.
- Locality - The basic premise of the device is that you need to be in range to see how much energy you are using, which in the common use case where the user is in the household, that would be fine. This product doesn't take into account the use case where the user is not in the house but still wants to monitor energy usage.

Figure 1: Meter Plug in the environment

1.1.2. Neurio

Neurio [24] is a product that makes monitoring power more accessible.
Neurio does breakdown data per appliance. It does this by analysing the consumption patterns exhibited by common household devices, matching them to a predetermined pattern, and is then able to determine the difference between an air conditioner, and a refrigerator for example.
Unfortunately, the granularity of this product does not allow it to detect all appliances, though it claims to "account for more up to 80% of the consumption in your home" [12].
Neurio has an open API that allows product developers to take advantage of the energy monitoring technology, and allows for the expansion of control features from the users' smartphone.
Neurio works through attaching a Wifi enabled sensor to the main power junction of the household. This sensor then feeds all it's data to the Neurio servers so that it can then be accessed on the users' device.

Figure 2: The sensor used for Neurio

Pros

- Easily accessible - The data supplied by Neurio is easily accessible through smart phones, and the data displayed by the app is easy to understand and digest.
- Minimal - The Neurio sensor is hidden from view, and is relatively small.
- Reasonably Priced - Neurio prices up at £55, which is a reasonable price for the ability to look at the overall energy consumption for a household.
- Customisable - Neurio is extensible though their development toolkit.
- Real Time Data - Data is transmitted every split second.
- Customisable - Neurio can be integrated into IFTT (If then this that) and SmartThings to add additional features, i.e. when the washing machine turns on, you receive a push notification.

Cons

- Granularity - Neurio can detect only large appliances which according to Neurio make up 80% of household consumption [11].

Figure 3: The app for the end user.

1.1.3. Kill A Watt

Kill A Watt [13] is a simplistic solution to the energy monitoring issue. It's a very similar to the Meter Plug, without the connectivity. The user knows how much energy an appliance is used through the LCD display immediately above the plug.

Figure 4: The Kill A Watt package

Pros

- Simple - The Kill A Watt is simple to install and the data it produces is easy to digest.
- Real Time - The Kill A Watt is always shows the real time energy consumption of the appliance it's connected to.

Cons

- Large - The Kill A Watt is bulky and would stand out in your home environment.
- Expensive - The Kill A Watt prices up at £20 which is expensive for a device which offers limited functionality.
- Not easily accessible - In order to view the data, the user will need to be in the vicinity of the plug.

1.2. Related Research

There is a lot of related work in this field with the aims of aiding a user to be more energy conscious, thereby reducing their energy consumption.
This section provides a brief overview of this related work and possible applications to this project.

The field of power measurement for home appliances is not a new field, it has been around for a long time.
As a result, there has been a huge amount of research into different methods of analysing appliances and displaying this data in an easy to understand format for a user to digest.
Power signature analysis [28] is a concept whose results can be seen in Neurio, discussed in existing products.
Laughman and the team theorised that a distributed network of sensors may not be the best way of measuring current of various appliances, especially in large buildings where noise is unpredictable. Instead, they proposed an improvement to non-intrusive load monitoring whereby the harmonic characteristics of appliances were used as an identifier to detect and measure voltage at the electric utility service entry.
This concept has been tried and tested, the results showing that the granularity of a system utilising this methodology will only be accurate with a small number of concurrent devices [29].
The alternative to the former, is a distributed network of sensors that feedback to a centralised server. In 2008, Bai and Hung investigated the capabilities of a low power embedded system for appliance control and current measurement using Zigbee for communicating data [7]. Their work concluded with a low cost, low power, wireless embedded system that pertained to this architecture.
There are a couple of issues with this solution. Firstly, the outlined architecture would not be able to scale to larger buildings. Larger buildings are subject to huge amounts of noise either corrupting packets or blocking them completely. The range of Zigbee would also be a factor in why this solution would not scale favourably, as an extreme example, a 50 storey building would not be a suitable deployment location. Moreover, the final solution was quite bulky, which would be an important factor in many deployment scenarios.
Rather than using Zigbee for communication, Lien and the team investigated the use of communication over power line [30]. Their research lead them to a web based monitor and control platform for their distributed sensors. However, the sensors produced were large and

obtrusive in their environment due to the demands and requirements of power line communication. On the other hand, their results were accurate, and integrated extremely well using the inbuilt architecture of the home.
Lien and the team also investigated the applications of a remotely controlled outlet system in the household [31]. This research involved the use of Bluetooth to control appliances, and GSM to send commands to the controller. The downfall of their approach being that the Bluetooth modules consumed a lot of power, had a limited range, and were affected by the noise of the environment.
The importance of current monitoring is paramount, but it is also important to relay this information to a user in an easy to understand format so that a user can then amend their behaviour dependant on the information displayed.
There have been a number of studies that have investigated the repercussions of energy consumption feedback for users.
Costanza in 2012 investigated the importance of Interactive Visualisation in aiding the conservation of energy [14]. The paper illustrated that the participants of the study found the engaging nature of the software helped them to identify heavy consumption appliances. In the paper the notion of an activity was introduced, as the participants went about their daily tasks, they were able to associate appliance usage with their activity.
A paper was also written on the openness and visibility of energy data, and detailed the methodology of how to design digital technologies to better describe the energy their home is consuming [39]. Various different interfaces were trialed and tested, and the results showed that these interfaces benefited households positively and affected their day to day activies.
These papers show that engaging visualisations affect user behavior, and these visualisations could be part of the scope of this project.

1.3. Conclusions

The selected existing products are from the same area of the market, but each performs the task of energy measurement very differently.
The apparent conclusion from the existing products section is there isn't a product on the market that is low cost, low power, unobtrusive in its environment, accurate and has a per appliance granularity combined with the convenience of accessing data through the use of mobile devices.
The related work section highlights areas which a high concentration of research and time has been invested. The research conducted in this field is invaluable, but there are areas where the research is lacking.
Laughman's work determined that using the electric utility service entry as a single point of monitoring was an extremely successful approach to the issue at hand. The downsides of the approach being accuracy, as it didn't scale to a large number of devices.
Bai's work investigated the use of a network of sensors reporting to a centralised hub, to produce monitoring data for a large number of appliances. This architecture would scale really well if the range of the Zigbee's was unaffected by noise.

Lien investigated the possible applications of Bluetooth with a similar architecture to that of Bai's. However, Bluetooth was shown to be too short range and consumed a comparatively large amount of power.
Lien also investigated the use of the homes internal architecture, the power line. The work produced a distributed network of sensors that fed back to a centralised point on the power line. There were a couple of drawbacks to this approach, the most important drawback being the size of the sensors.
The overall conclusion to be drawn from this selection of papers is that the correct architecture to a successful system is a distributed network of sensors with a centralised point of contact for the sensors.
In these papers, power line, Zigbee and Bluetooth communication have been covered extensively. However, there is an important, less traditional medium of communication that has been overlooked, the Earth line. This medium will be one of the main focuses of this project.
The final two papers covered in the related work section detailed the importance of visualising the data gathered by the sensors. Therefore data visualisation will be another key aspect of this project.

1.3.1. Top Level Requirements

Through the analysis provided in the previous section, there are a number of requirements that can be generated:

- Small - The device needs to be small and unobtrusive and if possible, it should be hidden.
- Minimalist- The devices design should be simple and not overly complex.
- Low power - The device shouldn't consume a lot of energy, as that would defeat the aim of the device.
- Low cost - The device should be inexpensive, so that the technology is easily accessible to all.
- Simple to install and use - The device should work out of the box and require minimal setup and installation.
- Accessible data - The power consumption data should be instantly accessible through either a web interface or smartphone app.
- Real time - The data produced and displayed should be real time and up to date.
- Breaks down data per appliance - The data produced should be tied to a particular appliance so that the user can visually identify each device.
- Reliable - The device should be reliable and require minimal maintenance.

2. Background

2.1. System Outline

Using the information gathered about existing research into this space, a few architectural components can be derived:

- Appliance sensor - There will need to be a small device used to sense the current being used by the appliance. This device will need to be able to communicate using a wired medium.
- Central hub - There will need to be a device that listens for packets being transmitted by the current sensor, and it must be able to process them and forward them onto a server.
- Server - There will need to a server, that will be used to store, access and manipulate the data provided by the central hub.
- Application - There will also need to be an application so that the user can see the data reported by the current sensor.

In the Related Research section, investigations have already been performed on the use of Zigbee, Bluetooth and Powerline communications as a method of reporting the sensor data. However, these solutions were all bulky and consumed quite a lot of power. One method of communication that wasn't considered was the use of the Earth line of a building.
The use of the Earth line would provide a network of distributed sensors, all communicating towards a centralised exit point. Placing the central hub at the exit point would enable the sensing of packets transmitted from the current sensors.
A visualisation of this proposed system can be seen in Figure 5.

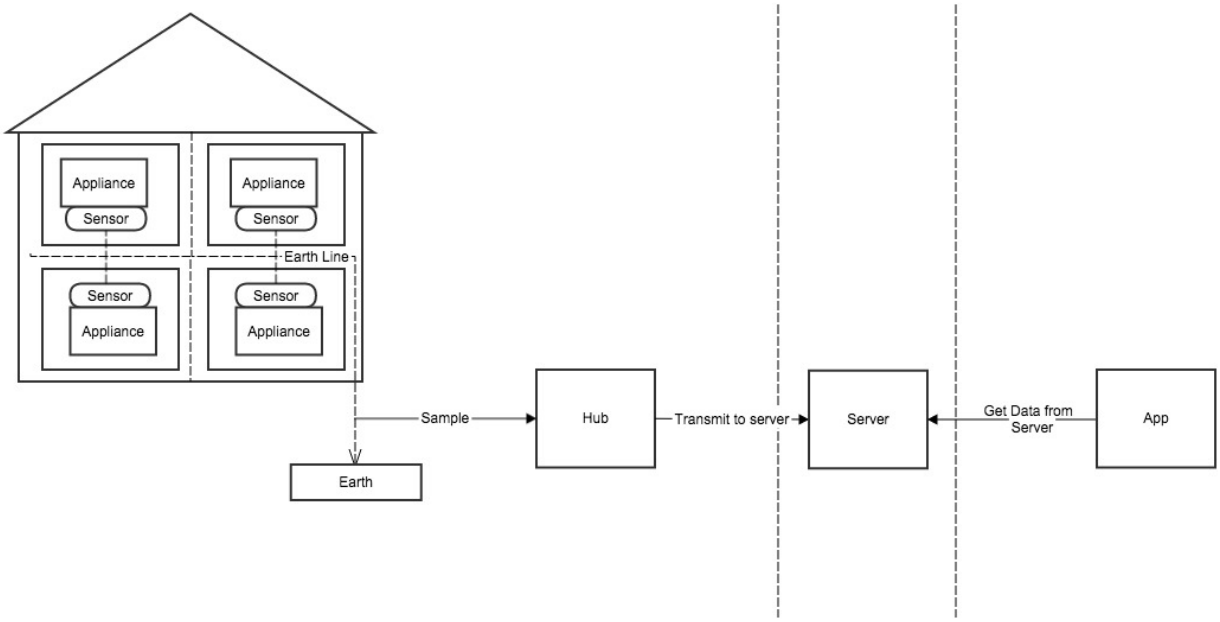


Figure 5: The proposed system.

The Appliance sensors will report power consumption for appliances using the Earth line as the communication medium.
The Central hub will be placed at, or near the exit point of the Earth line, so to gather packets transmitted on the Earth line as all packets will travel in the same direction, towards Earth.
The Central hub will then be connected to the Server through the Internet using Wifi or Ethernet as the communication medium. When a hub detects a packet, it will then transmit the received data to the Server.
The Server contains all data generated by the sensors used in the platform. The Server is also able to serve the information contained to clients.
The Application provides core visualisations and insights into the data gathered by the sensors. The application will access the data stored on the Server and present it to Users in an easy to digest format.
The proposed to system adheres to all of the top level requirements generated in the Introduction as will be described in the following sections.

2.2. Appliance sensor

In the System Outline section a method of monitoring the power consumption of an appliance was described as a core component. This section briefly discusses the options for a small, low cost embedded device to perform this operation, which aligns with the overall top level requirements.

2.2.1. Hardware Selection

Based on the requirements, an embedded system seems to be the most applicable technology to this aspect of the project

Todays market consists of a large number of embedded devices built for a huge range of tasks. There are many manufacturers of these chips such as Freescale, ARM and Texas Instruments, and there are far too many different chips to compare.

This section will determine the chip best suited to the task from one manufacturer, Microchip.

2.2.1.1 PIC10F322

The 10F322 [35] is a small surface mount PIC. The 10F322 is smaller, and is relatively cheap costing 56 pence for amounts over 100. The 10f322 also draws a small amount of power around 0.8 µA at 32khz.

The downsides to using this pic would be that it has no on board serial module. This would increase development time, which may skew other factors of the project.

However this PIC would be more suited to a production environment because of its size and price.

2.2.1.2 PIC16F1788

The 16F1788 [34] is a large surface mount PIC. The 16F1788 has lots of program memory, RAM and carries a lot of on board components including an ADC, and Serial I/O. This makes it great for testing and experimenting with new circuits.

Conversely, this PIC has a large number of on board components, which increases the price (£1.91), and the size of the PIC.

The power draw of the circuitry is around 8 µA at 32khz, which would be quite high for a final product.

2.2.2. Conclusions

The 16f1788 would be the ideal PIC for testing and developing the circuitry for the Appliance sensor. However, when entering production, the 10f322 would be the PIC of choice as it aligns far better with the top level requirements due to its size, price and selection of components.

2.3. Hub

In the System Outline section, a Hub that detects packets transmitted by an Appliance Sensor was described. This component would listen for packets transmitted on the Earth Line, processes and validates the received packets, and stores them on a server.
This section provides a brief overview of the different possible options in terms of hardware for this specific component.

2.3.1. Hardware Selection

The requirements dictate that something more powerful than a traditional embedded system will need to be used to process and store the data generated by the Appliance sensors.
The hardware will need to be able to have either wired or wireless Internet connectivity, as well as being low cost, low power and able to process quite a lot of data.
Over recent years, a lot of investment has been placed with the aim of creating affordable, small, powerful computers that are widely available, which would be suited to this aspect of the project.
This section will compare key players in this market to determine the hardware that will be utilised in this project.

2.3.1.1 Raspberry Pi

The Raspberry Pi [22] is the most well known member of this market, and with the most recent introduction of the Raspberry Pi 2, is a strong market leader as well.
The Raspberry Pi has a number of hardware components that can be controlled by a user, including a Serial I/O port. This Serial I/O port can be used to communicate with external embedded systems, which could be useful for the project.
The Raspberry Pi costs around £30 depending on the model. It comes with Ethernet as standard, but external modules can be added to introduce other technologies such as WiFi and Bluetooth. This will obviously incur additional costs.
The power consumed by the Pi on average is around 700 mA which is really low for such a powerful computer.

2.3.1.2 Hummingboard

The Hummingboard [40] is a relatively new player in the market.
It's more expensive (£15 more) than the Raspberry Pi, but offers more RAM, and a faster processor.
Like the Raspberry Pi, it offers GPIO and UART built in. Ethernet is also offered as standard, but again like the Raspberry Pi, for an additional amount, WiFi and Bluetooth can be added.
The operating system is Linux based, so it has support for quite a lot of the standard Linux distributions, much like the Raspberry Pi.
The Hummingboard consumes slightly more power than the Raspberry Pi, approximately 780 mA in total, but this would make a negligible difference overall.

2.3.1.3 Beaglebone Black

The Beagle Bone [21] prices £5 above the Raspberry Pi and offers a 1gHz processor instead of the Pi's 700 mHz one.

The power consumed by the Beaglebone is half that of the Raspberry Pi, with 350 mA and it has a large number of GPIO pins. However, it doesn't have inbuilt UART for serial communications.

2.3.1.4 Summary Table

	Raspberry Pi	Hummingboard	Beaglebone Black
Small	✓	✓	✓
Low power	✓	✓	✓
Low cost	✓	✗	✗
Simple to install and use	✓	✓	✓
Reliable	✓	✓	✓

2.3.2. Conclusions

It appears that the best candidate for the hub element of this project is the Raspberry Pi. It is the cheapest, and offers a wide range of features as well consuming a small amount of energy. The measurements taken were from the original Model B, and at the time of writing, the Raspberry Pi 2 has just been released, offering improved features for the same price.
The Raspberry Pi aligns with most of the applicable top level requirements, except for the requirement of minimalism, as the Raspberry Pi is quite complex.

2.4. Server

A Server was described as a core component in the System Outline section. The Server would act as a central point for data storage and retrieval.
This section compares the options for both the Web Server and Database aspects of the server, and concludes with the selected Database and Web Server software.

2.4.1. Web Server

There are a number of web server packages available, the most popular being Apache.
Apache has been around for a long time, and the most common stack for an Apache server is LAMP: Linux, Apache, MySQL and PHP.
Considering that the system will be receiving a large amount of data, will be used in conjunction with a cross platform app and the system will be event driven, the aforementioned Apache LAMP stack may not necessarily be the best choice.
There are many modern variations and implementations of web servers around today, developers no longer need to transition between different languages. Python developers can transition to Django, a Python web server. They can build apps using python, transition to the server side development environment, and build the API. The story is the same with Ruby, Perl and more recently, JavaScript.

2.4.1.1 MEAN Stack

The MEAN [33] stack is built from four components: MongoDB - the database, Express - the web application framework, Angular - the Client side framework and finally Node to create the MEAN stack.
MongoDB is a NoSQL database and will be discussed later on in this document, when determining the correct database software.
Express [17] [8] is a Web Application framework which manages the routing for the application, as well as simplifying the overall process getting request data. It acts as the glue between Node, and MongoDb.
Angular is a language built by Google, based on Javascript. It implements the MVC architecture, and is implemented client side, using HTTP requests or local storage for data.
Node.js [50] is an open source runtine environment for server side applications, and provides an event driven architecture and non blocking I/O making it extremely scalable.
Node replaces the Apache element of the LAMP stack, and uses the Google V8 JavaScript engine [52] used in Google's popular Chrome browser. The v8 engine compiles and executes JavaScript, rather than the more traditional model of interpretation on the fly.
When comparing Node to the traditional Apache LAMP stack [23] [26] there are immense speed improvements due to the non-blocking nature of Node. This is extremely important for a number of reasons:

- Scalable - Node works concurrently, the number of users makes no difference to the page load time, improving the aspect of scalability.

- Lower Costs- Nodes concurrency removes the issue of concurrent users impacting load times, this means that less physical resources will be required.
- Rapid - As Node is concurrent, pages load lightning quick, which allows data to be served extremely quickly to the client.

2.4.1.2 Django

Django [48], unlike the MEAN stack, can be configured to be layered upon a number of base web servers, and is a Web Framework.
In the traditional LAMP model, Django takes the place of PHP, and instead uses Python to construct the application.
As PHP is a language, it can only be compared to Python in this case. The performance of Python [15] is 2-3 times faster than PHP, but also used 2 times as much memory. So there is a trade off between performance and server resources which would be important when scaling the solution.

2.4.1.3 Ruby on Rails

Ruby on Rails [51] is similar to Django, it is a web framework, and it acts as a replacement for the PHP element of the LAMP stack.
As PHP is a language, it can only be compared to Ruby in this case. The performance of Ruby [16] is 2-4 times faster than PHP, but again uses more memory, and thusly there is yet again a trade off between performance and server resources, affecting scalability.

2.4.1.4 Summary Table

	Mean Stack	Django	Ruby on Rails
Scalable For A Huge Number of Connections	✓	✗	✗
Rapid Development	✓	✓	✓
Event Driven	✓	✗	✗
Minimalist	✓	✗	✗

2.4.2. Database

The database is an important consideration in any project. In the Web Server section, it was determined that a solution involving MEAN stack was preferable over the more traditional LAMP stack solution.
The database layer is separate, but at the same time is intertwined with the server layer. This means that there needs to be a relatively easy way of communicating between the two layers.
This section will compare various database solutions applicable to the kind of project discussed in this document: the Internet of things.

2.4.2.1 MongoDB

MongoDB [20] harnesses the benefits of a NoSQL database, but at the same time retains the relational aspects of SQL.
Documents are stored using JSON documents, and B-Trees are used for quick look up times. MongoDB also scales across a number of servers using Passive server replication.
It's best used for real time systems, with rapidly changing data [27], which is directly applicable to the project.
It's also worth considering that the rest of the stack uses the *EAN model. Thus the majority of development will be in Javascript, implying that it might be best to store objects in JSON for easy storage and retrieval. MongoDB's main storage format is JSON, it will therefore be easy to integrate with the rest of the stack.

2.4.2.2 Redis

Redis [9] is an advanced key-value cache and store. It stores data in traditional data structures, like Strings, Hashes and many more.
Redis stores all objects in memory, which may make it unsuitable for the project in the long term. A user can however partition memory across server machines [10], so that Redis can expand beyond the limits of a single machines.
The issue with redis is that there isn't a direct translation between database and server like there is in MongoDB, which may make storage and retrieval difficult. It is also far easier to increase disk space than it is RAM.
Redis is best used for projects with rapidly changing data with a foreseeable database size [27]. In this project, the database size is not fully predictable due to a number of variables, like the number of Appliance sensors and the number of Hubs a user may own.

2.4.2.3 CouchDB

CouchDB [19] is written in Erlang, and stores data in JSON format. CouchDB uses HTTP requests to perform CRUD (Create, Read, Update, Delete) operations on the data. The interesting concept behind this is that CouchDB can actually act as the applet, and allows easy content management using a control panel integrated into the database.
CouchDB unfortunately doesn't offer dynamic queries, instead offering map/reduce functions. It's also not relational, both of which will impact performance in the long term.
CouchDB is best used for accumulating occasionally changing data, where queries have to be predefined [27]. In the project, data is dynamic and constantly changing, CouchDB is not. This factor, in combination with the non-relational nature of CouchDB, renders it unusable in this project.

2.4.2.4 Summary Table

	MonogoDB	Redis	CouchDB
High Performance	✓	✓	✗
Suitable for Dynamic Data	✓	✓	✗
Easy Storage and Retrieval	✓	✗	✓
Compatible with the MEAN stack	✓	✓	✓

2.4.3. Conclusions

The MEAN stack seems to be the obvious conclusion from the comparison above. Scalability and performance are key factors in this decision. As the system becomes more widely used, more requests will be received. A stack like the MEAN stack where the number of requests makes little or no impact is ideal.
Development time is also another key factor. Using the MEAN stack will result in faster development, due to the rapid nature of Node and the rest of the stack.
The MEAN stack also aligns with the top level requirements. It is scalable, reliable, minimalist, simple to install and use, and it is real time due to its event driven nature.

2.5. Monitoring Visualisation Application

In the System outline, the Application acts as the bridge between the data aggregation and data visualisation.
The Application would provide visualisations of the data gathered by the sensors, and display it in an easy to digest format for the User.
This section will compare and contrast the different options for creating the application, and conclude with the chosen technology.

2.5.1. Frameworks

Accessibility is an important requirement for this project, and as such an application that is available to only one platform would tarnish this requirement.
To complete two fully featured applications native to each of the key platforms (iOS and Android) would take a long period of time. Because of this, a cross platform app would be the best choice, and would allow the development of the application for multiple platforms simultaneously. In the previous section, the MEAN stack was chosen as the preferable application stack. The element applicable to client-side applications in the MEAN stack is Angular. A cross platform solution that integrates with Angular would be ideal and would decrease the development time for the project further.

2.5.1.1 Ionic

Ionic [43] is a framework created by a company called Drifty, and is powered by Angular JS, conforming to the MEAN stack requirement.
It is a high performance, native focused, cross platform framework allowing for rapid yet stable development.
Currently, the cross platform element supports only iOS and Android, but Windows phone and FirefoxOS are coming soon.
Ionic utilises Angular lending itself to the MVC architecture, which allows the compartmentalisation of the app.
Like many mobile frameworks, it has a vast amount of built in components, to make the final app look and feel native to the platform.
PhoneGap [38] and Cordova [18] are used to wrap the HTML, CSS and JavaScript produced by Ionic and presents them in a full screen web view.

2.5.1.2 Ratchet

Ratchet [41] was developed by Connor Sears who worked at twitter and need a way of rapidly prototyping new UI ideas for the native Twitter app.
Unlike Ionic, this framework doesn't use Angular, and uses native javascript to perform UI interactions.
This may make app development more difficult due to the lack of compartmentalisation, and as the project scales, source code will be harder to maintain and update.
Ratchet also has a number of built in UI components, however it seems like Ratchet doesn't

make any effort to appear native to the platform.
It also doesn't adhere to the MEAN stack requirement as it doesn't intertwine with Angular, and it seems like this framework is more focused on prototyping than on formal app development.

2.5.1.3 Famo.us

Famo.us [25] is an extremely new framework, and is in very early stages of Beta, the current version being 0.3 at the time of writing.
This framework can easily be integrated with Angular, matching the MEAN stack requirement.
This framework claims to be a platform for building high-performance user interfaces that can be used to display complex animations and has an inbuilt physics engine to support games development.
Although this framework simplifies UI development, it doesn't simplify the overall app development like the Ionic framework does.
It's also in early development, and appears to not offer as many features as Ionic or Ratchet.

2.5.1.4 Summary Table

	Ionic	Ratchet	Famo.us
Stable	✓	✓	✗
Native Look and Feel	✓	✗	✓
Feature Rich	✓	✗	✗
Compatible with the MEAN stack	✓	✗	✓

2.5.2. Conclusions

The framework that meets the most requirements appears to be the Ionic Framework.
It offers great performance, a huge range of UI tools, Angular at it's core and cross platform support.
It has a great support community behind it, and it is well established in the development world. These combined factors make it in obvious choice for the basis of the application for the project.
The Ionic framework neatly aligns with the top level requirements, and its use would provide an accessible, real time, per appliance view of the data gathered by the application.

2.6. Existing APIs

As mentioned in the related research section, data visualisation can influence user behaviour. As such, it would be interesting to include external APIs to the project to enhance the information provided to the User.
Dr. Adrian Friday recommended 4 APIs that would provide real time information that could influence users to change their behaviour when it comes to turning on large appliances.

2.6.1. Carbon Intensity

The first API that was recommended was the Carbon Intensity API, which is an xml file that details the current carbon intensity level, as well as the historic carbon intensity level for the UK. This can be accessed using the following url:

`http://www.earth.org.uk/_gridCarbonIntensityGB.xml`

The insights provided by this API are numerous. The information could potentially influence users to change the times at which they use large appliances through the presentation of the level of carbon in the UK. If the carbon level is high or medium, a user might be dissuaded from putting on large appliances.
Damon Hart-Davis, the creator of this API, was contacted to ensure that the harvesting of this API would not affect the service offered by the full site:

`http://www.earth.org.uk/_gridCarbonIntensityGB.html`

2.6.2. Energy Prices

Another data set that could be relevant to Users is energy prices.
The current energy prices can be obtained from the following URL:

`http://www.nordpoolspot.com/api/marketdata/page/1639`

Historic energy prices can be obtained from this URL:

`http://www.nordpoolspot.com/api/marketdata/page/4708`

The ability to view live and historic energy prices could influence power consumption by a User who is on a variable price tariff.
A variable price tariff is where the price of the energy billed by the supplier reflects the actual price of the energy market at that time. The presentation of this data may influence users to wait until energy prices are not at their peak.
Peaks in the prices occur when the grid is under the most load. For example, at 7pm after various TV shows have finished, there is a peak in the energy prices due to everyone boiling their kettles.
Energy price also correlates with the historic carbon levels in the UK, therefore the presentation of this information would also influence Carbon Intensity, even if the User was concerned only with the energy prices.
NordPoolSpot [3], is Europe's leading power market, and it offers a fully open API to fetch market data historically or predictions for the day ahead.

2.6.3. Generation Statistics

The use of generation statistics would be more for interest than influencing user behaviour. From the following URL, current generation and historic generation details can be obtained:

`http://www.bmreports.com/bsp/additional/soapfunctions.php?element=generationbyfueltypet`

The details returned by the API provide insights into energy generation by energy sources for the UK. BMReports [2] provides an open API to query data sets pertaining to energy generation. The data is generated by the Balancing Mechanism used by the National Grid, which controls the flow on and off the electricity grid system.

2.6.4. Conclusions

The use of the APIs mentioned in this section, used in conjunction with a good graphical user interface, would benefit the Users of the application through providing access to data that isn't widely used by front facing applications.
It would also provide data visualisation opportunities that could influence user behaviour, through the provision of this real time information.
The use of the APIs also aligns with the top level requirement of accessible data, and meets the visualisation aims of the project.

3. Design & Implementation

The section outlines the design and implementation of each component discussed in the background section the overall system architecture is also described.

3.1. Overall System Architecture

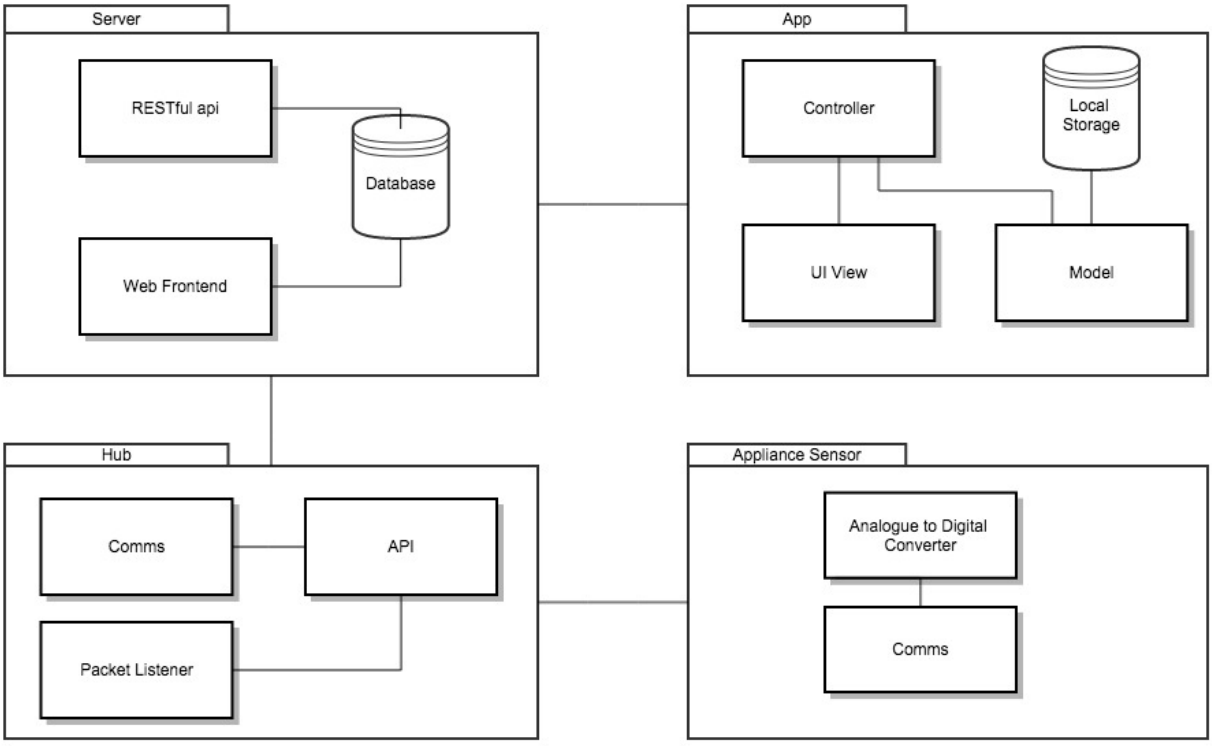


Figure 6: Overall System Architecture

Figure 6 is the proposed system architecture for the project.
There are four key components: the Server, the Cross Platform Application, the Hub and the Appliance Sensor. These four components are based upon the components derived in the Background section.
The architecture diagram describes the compartmentalisation of the four different components and the connections between each component.
The appliance sensor can only communicate with the Hub, the Hub can only communicate with the server, and the application can only communicate with the server.
The Appliance Sensor consists of two subsystems, the analogue to digital converter which is responsible for monitoring the the current consumed by an appliance, and the communications subsystem which is responsible for communicating data using the ground line.
The Hub has three subsystems. The Comms subsystem is responsible for communicating

received packets to the server, the API subsystem is responsible for determining what to communicate to the server via the Comms subsystem. The Packet Listener subsystem listens for packets communicated over the Earth line, the Packet Listener will then pass the data received to the API subsystem which will communicate the data to the server using the Comms system.
The Server also has three subsystems. The RESTful API subsystem offers data through a well-defined API adhering to the RESTful principles. The Database subsystem offers a way of storing and retrieving information provided by the RESTful API and Web Frontend subsystems. The Web Frontend subsystem serves simple web pages that provide information about the project, it talks to the database subsystem to retrieve required information.
The Cross Platform App has four subsystems. It adheres to the MVC architecture, imposed by the Ionic framework as discussed in the background section. The Controller subsystem handles actions performed by the user and communicates with the View and Model subsystems to impose updates. The Model subsystem handles the storage and retrieval of data used in the application, it communicates with the Database and provides a layer of abstraction. The UI View subsystem is the visible interface, which is updated using the controller.

3.2. Appliance sensor

The Appliance sensor consists of a PIC16f1788 running custom assembly software. The schematics of the circuit, and the overall software architecture of the Appliance sensor are described in this section.

3.2.1. Requirements

- Transmit Data - The Appliance sensor should transmit data using the Earthed line.
- Accurate - The Appliance sensor should be accurate with its samples.
- Small - The Appliance sensor should be small.
- Low Cost - The Appliance sensor should be low cost.
- Low Power - The Appliance sensor should not consume a large amount of power.

The PIC16f1788 matches all of the applicable overall top level requirements as well as those specified above, aside from one aspect, the PIC16f1788 isn't minimalist.
The PIC16f1788 as discussed in the Background is more suited to testing and development than production. If the system were to move to production, the PIC10f322 would be used, which would match all of the requirements.

3.2.2. Software

The software for the Appliance sensor is written in assembly.
Assembly is used for extremely low level embedded devices. The reason assembly has been chosen is because it offers a level of accuracy that is extremely hard to achieve through using a higher level language like C.

3.2.2.1 Current Sensing

The Appliance sensor generates data that can be used to generate an average over time.
When a sample is collected by the ADC, the sample count is incremented and the ADC result is added to a sum.
When the Appliance sensor transmits, the packet contains the sample count and the sum of all of the samples currently held in memory. The average voltage is then calculated on the Hub, and sent to the server.

3.2.3. Communication Principles

Referring back to Figure 5, communication occurs over the Earth line. The Hub is placed towards the end of the Earth line, and uses a current sensor to detect packets.
To provide robust transmission down a usually noisy line, there needs to be a form of error correction in place.
The chosen form of error correction is Extended Hamming Code, due to it's extensive use

throughout the industry.
Each byte will be encoded before transmission in a 4:8 encoding ratio. This means every 4 bits will produce a byte when encoded.
Unfortunately, this does produce more bytes, which is the downfall of this encoding, however it will make the transmission more robust and more impervious to errors.
In Extending Hamming Code, each nibble is encoded using a look up table:

Hex Value	Encoded Result
0	0x15
1	0x02
2	0x49
3	0x5e
4	0x64
5	0x73
6	0x38
7	0x2F
8	0xD0
9	0xC7
A	0x8C
B	0x9B
C	0xA1
D	0xB6
E	0xFD
F	0xEA

Table 1: Hamming code translations

Once the nibble is encoded, the byte is transmitted using RS232 via the Earth Line. Hamming encoding will be discussed again later on in this document, with a further explanation of how it works.

3.2.3.1 Random Transmission

The timings between each transmission is 'random'. This randomness is generated through the incrementation of a variable throughout various points in the code. When this variable hits a specific value, data will then be transmitted.
The reason for random transmission is to reduce collisions on the line. The hope is that through using a randomised variable, Appliance sensors will transmit at different times.

3.2.3.2 Packet Structure

The following details the packet structure that is transmitted by the Appliance sensor.
The packet consumes 7 bytes. Due to hamming code, this packet size is doubled, making the final packet size 14 bytes.

Label	Bits	Address
Device ID Lower	8	0x26
Device ID Higher	4	0x27
Sample Count Lower	4	0x27
Sample Count Higher	8	0x28
Total Lower	8	0x29
Total Middle	8	0x2A
Total Higher	8	0x2B
Checksum	8	0x2C

Table 2: The break down of the packet

Packet Description

- Device ID - The Appliance sensor's ID generated at the factory. It can have a value anywhere between 0 and 4095.
- Sample Count - This variable is incremented each time a sample is taken, and allows the hub to work out the average voltage consumed by the connected appliance. The maximum value being 4095.
- Total - This variable is the result of the summing of ADC values after a new sample is taken. It allows the hub to work out the average voltage consumed by the connected appliance. The maximum value for these 24 bits is 16,777,216.
- Checksum - Before the packet is transmitted, each register is added to the checksum register, ignoring overflow. This variable will then be used Hub-side in conjunction with hamming code, to determine if the packet has been received correctly.

3.2.3.3 Sampling

Sampling will occur very regularly, depending on the final clock rate of the PIC.
The PIC will continuously sample the ADC until the random transmission variable is matched.
In software, each time a sample is captured, it will be added to the Total Count, which is contained in 3 bytes as detailed in table 2. At the same time, the sample count will also be incremented.
When a hub receives the packet, it will be able to determine the average voltage being consumed by that appliance by dividing the total by the number of samples.

3.2.4. Hardware

The PIC that is currently in use is the 16F1788. This PIC has simply been chosen for the amount of on board components, and ease of integration between these various components. The PIC itself is far too big for the final product and there therefore should be a contingency strategy in place for the eventuality of moving to a smaller chip like the 10F322.

3.2.4.1 USART

The 16F1788 has an on board serial chip. This chip will be used to transmit the packet using the USART standard in serial communications.
Depending on the clock rate, the serial communication will occur at different baud rates. This will be a factor for testing in the evaluation section.

3.2.5. Schematics

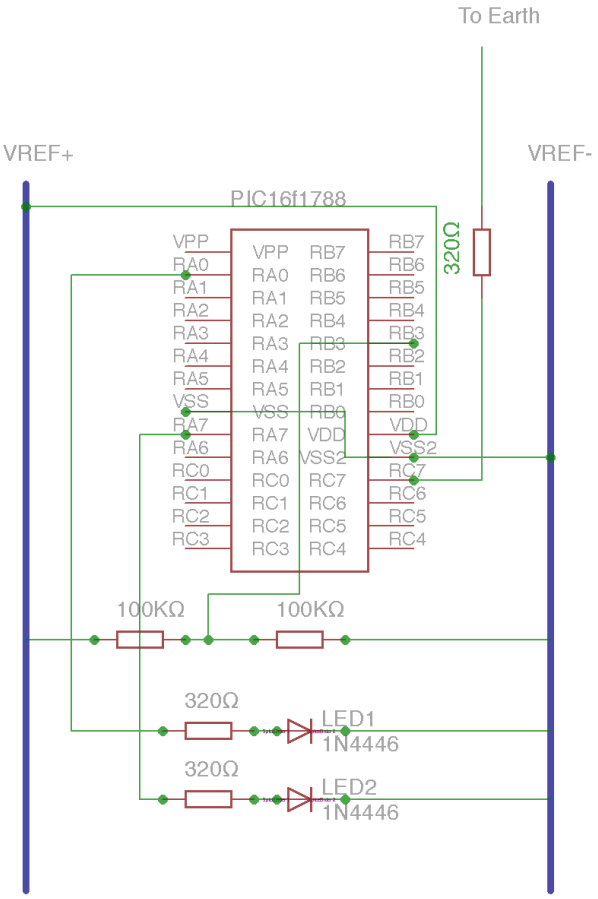


Figure 7: The schematic for the Appliance sensor.

Figure 7 shows the schematic for the current implementation of the Appliance sensor.
Currently the input for the Appliance sensor is a voltage divider using two 100K resistors across VREF+, which is then connected to RB3, the input for the ADC.
There are two status LEDs. LED1 shows the on state of the Appliance sensor, when the Appliance sensor transmits LED2 will light up for the duration of transmission.
The Appliance sensor transmits using pin RC7 to a faux Earth line connected to the hub. This line is protected using a 320 ohm resistor.
As mentioned, the input for the Appliance sensor is currently a voltage divider. This is

purely for testing, and to make it ready for appliances, the only change that would be required is the addition of a current sensor as the input to the ADC. The current sensor would then be connected to the appliance.

3.2.6. Architecture Diagram

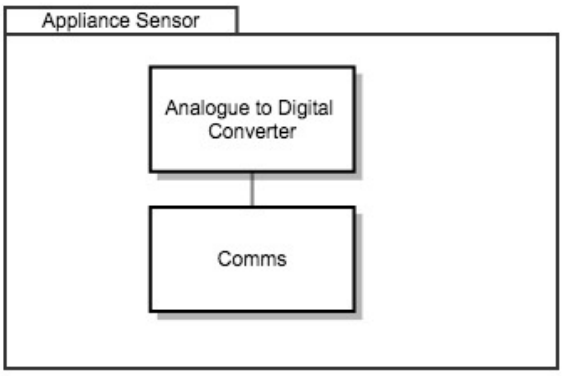


Figure 8: Appliance Sensor Architecture Diagram

3.3. Hub

The Hub consists of a Raspberry Pi connected to a custom circuit used to intercept messages from the Appliance sensor before they reach Earth. In this section, the additional circuitry is described as well as the software architecture of the Hub.

3.3.1. Requirements

- Transmit Data - The Hub should transmit received data to the server.
- Receive Data - The Hub should be able to detect packets being transmitted on the Earthed line.
- Small - The Appliance sensor should be small.
- Low Cost - The Appliance sensor should be low cost.
- Low power - The Appliance sensor should be low power, and not consume large amounts of electricity.
- Process Data - The Hub should be capable of processing data with ease.

The applicable top level requirement are matched precisely through the use of the Raspberry Pi. As well has the top level requirements, the specific requirements listed above match directly onto the features offered by the Raspberry Pi.

3.3.2. Software

The software for the Hub is written in Python.
Python is a really useful language, especially on the Raspberry Pi, where there are a lot of translations from lower level C to high level python.
This section covers the different aspects of the software: a class diagram, the event driven architecture, and the reception of packets

3.3.2.1 Class Diagram

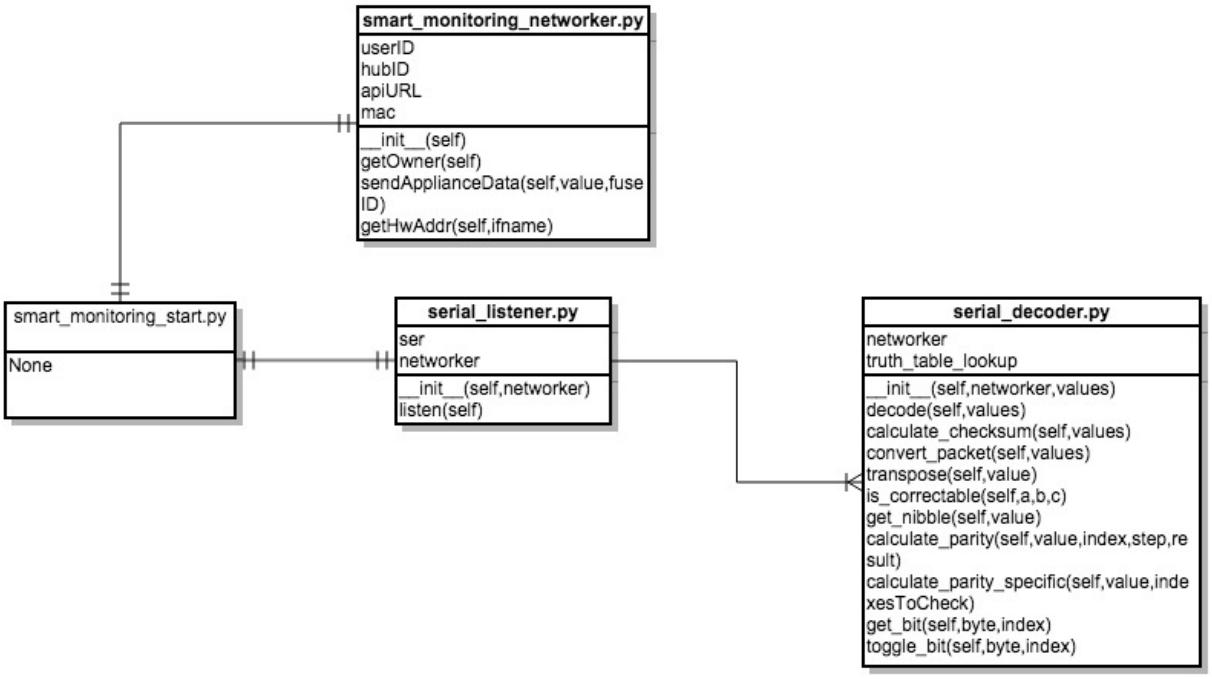


Figure 9: The class diagram for the Hub software.

Figure 9 shows the final class diagram for the hub software.
Each class is instantiated using the smart_monitoring_start.py file. When this file is executed, the smart_monitoring_networker class is instantiated, and getOwner is called. This prompts the hub to talk to the server and obtain its credentials and also the user account it's linked to. If no user account is linked, the hub will check at regular 30 second intervals until a user account has been found.
Once the hub has obtained all of the required details, the serial_listener class is instantiated with the instance of networker previously instantiated. The listen function is called on the serial listener, which will listen forever. This is the main loop for the program.
When the serial_listener class successfully receives a full packet, it passes the packet and an instance of networker to the serial_decoder and begins execution in a separate thread. If the serial_decoder successfully decodes the packet, it is transmitted to the server using the sendApplianceData function of the networker class passed in by the serial_listener.

3.3.2.2 Obtaining Credentials

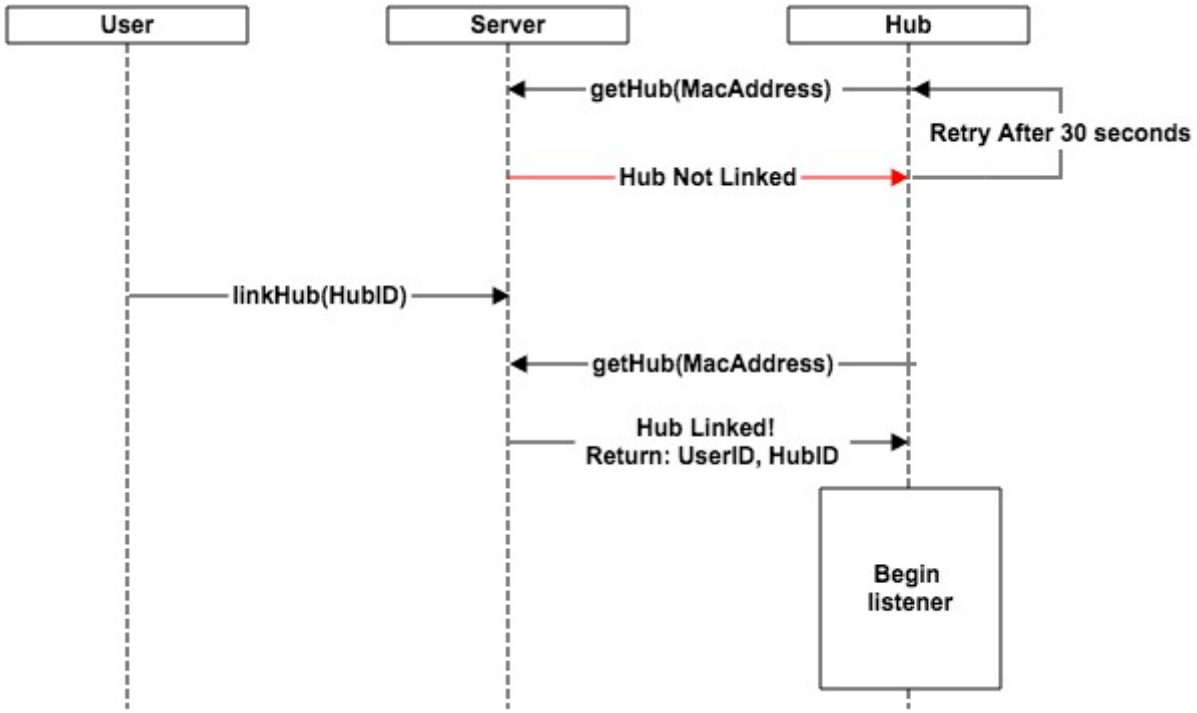


Figure 10: How the Hub obtains Credentials

Figure 10 shows a sequence diagram of how a Hub obtains credentials.
The Hub will continuously try and obtain API credentials until it receives an acknowledgement that the Hub is linked to a User account. A Hub is considered linked when the user adds a new hub to their account.
The Hub will not receive any packets for the duration that it is not linked, as there is a requirement imposed by the API, that each request must be associated with a user account.

3.3.2.3 Event Driven Architecture

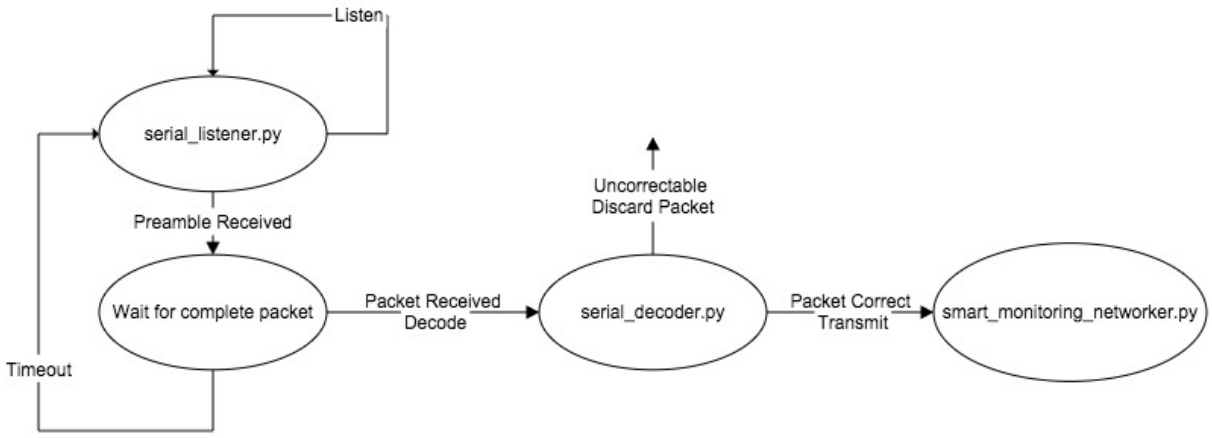


Figure 11: The event driven architecture for the Hub.

Figure 11 shows the event driven architecture used by the Hub.
The serial listener listens for the preamble which is 0x55. When It successfully receives the preamble, it listens for the next 14 consecutive bytes, if there is more than a millisecond gap between bytes, the whole packet is discarded, and the listener is reset.
If the listener receives all 14 bytes, it then starts up a Python Process to handle the decoding and transmission of the received data. Once the process has started, the listener immediately resumes listening for incoming data.
The serial decoder operates in a separate thread (or Process) and as the name suggests, decodes the packet using Extended Hamming Code. If the packet is deemed correct, the data is sent to the server and the Process terminates. If the packet is uncorrectable, the packet is discarded and the Process terminates.

3.3.2.4 Packet Reception

As the Appliance sensor is using Extended Hamming Code to communicate, the Hub must be able to decode the messages received.
Extended Hamming Code allows the receiver to determine the number of incorrect bits, and also the location of the incorrect bits. If the number of incorrect bits is one, then the receiver is able to correct this value, and proceed with processing the packet.
Unfortunately if there is more than one error, the packet is unable to be corrected, and therefore must be discarded.
On top of hamming code, there is a check sum. So even if the hamming code check succeeds, there is another layer of checking before the packet is determined to be valid.

Extended Hamming Code Example:

Say the following byte is received:

b7	b6	b5	b4	b3	b2	b1	b0	Hex
1	1	1	1	1	1	1	1	0xFF

Table 3: An incorrect byte received by the hub.

However the byte transmitted from the Appliance sensor was the following:

b7	b6	b5	b4	b3	b2	b1	b0	Hex
1	1	1	1	1	1	0	1	0xFD

Table 4: The actual byte sent by the hub.

The hub would go through the following process to determine if the byte requires correction:

$$\text{overall parity} = b7 \hat{} b6 \hat{} b5 \hat{} b4 \hat{} b3 \hat{} b2 \hat{} b1 \hat{} b0$$
$$\text{check } 0 = b7 \hat{} b5 \hat{} b1 \hat{} b0$$
$$\text{check } 1 = b7 \hat{} b3 \hat{} b2 \hat{} b1$$
$$\text{check } 2 = b5 \hat{} b4 \hat{} b3 \hat{} b1$$

Figure 12: The XOR calculations used in Extended Hamming Code [5]

If the overall parity variable is 1, 0 or 2 errors occurred. If checks 0-2 are equal to one, then the byte was received intact, otherwise it was damaged beyond repair.
If the overall parity variable is zero it means that there is a single bit error that can be corrected.
The following look up table can be used to identify the incorrect bit:

Check 0	Check 1	Check 2	Meaning
1	1	1	error in bit b6
1	1	0	error in bit b4
1	0	1	error in bit b2
0	1	1	error in bit b0
0	0	1	error in bit b7
0	1	0	error in bit b5
1	0	0	error in bit b3
0	0	0	error in bit b1

Table 5: The look up table used in Extended Hamming Code to identify incorrect bits [5]

Using the example of the incorrect byte in table 3, the checks can be evaluated:

$$\text{overall parity} = 1 \hat{} 1 \hat{} 1 \hat{} 1 \hat{} 1 \hat{} 1 \hat{} 1 \hat{} 1 = 0$$
$$\text{check } 0 = 1 \hat{} 1 \hat{} 1 \hat{} 1 = 0$$
$$\text{check } 1 = 1 \hat{} 1 \hat{} 1 \hat{} 1 = 0$$
$$\text{check } 2 = 1 \hat{} 1 \hat{} 1 \hat{} 1 = 0$$

Figure 13: Attempting to correct the incorrect packet

Figure 13 shows the resulting values from processing the incorrect byte from table 3.
The overall parity check indicates that there is a one bit correctable error. Then the checks 0-2 are calculated and subsequently used in the lookup table 5. The lookup table indicates that bit one is incorrect, bit one is flipped, and the corrected byte is:

b7	b6	b5	b4	b3	b2	b1	b0	Hex
1	1	1	1	1	1	0	1	0xFD

Table 6: The corrected byte.

3.3.3. Schematics

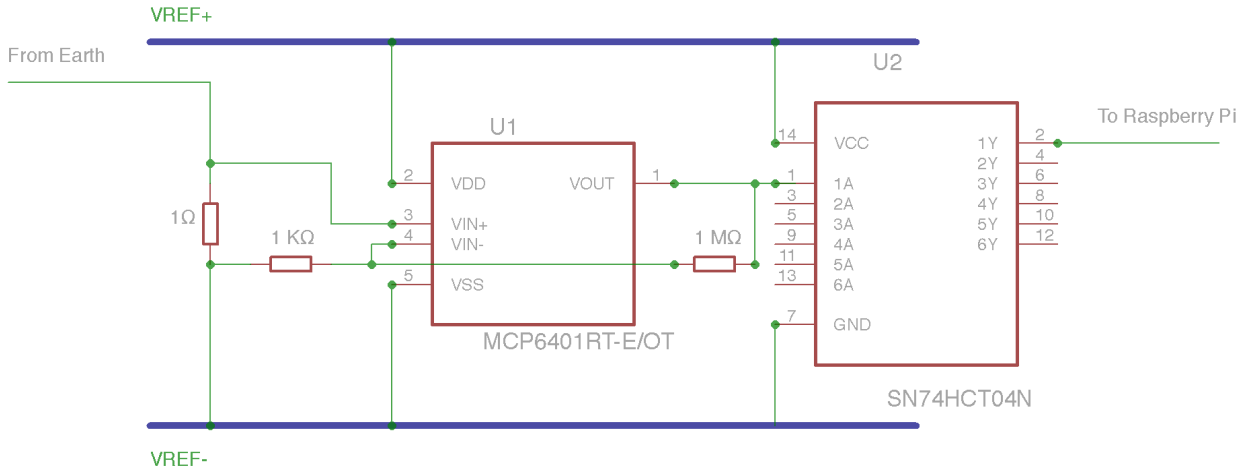


Figure 14: The schematic for the Hub.

The additional circuitry in figure 14 is used to detect messages sent from the Appliance sensor before they hit actual Earth.
A precise OpAmp (MCP6401RT) is used to amplify the packets, the output of which is used in conjunction with a Hex Inverter to transpose the packets from inverted RS232 to non-inverted RS232 to be processed by the Pi.
A 1 Ohm resister is used to provide a small difference between VIN+ and VIN- without distorting the signal. The OpAmp then uses a closed loop to determine the level of gain it should apply to the signal, which is calculated by dividing 1 megohm by 1 kilohm resulting in a gain of 1,000.

3.3.4. Architecture Diagram

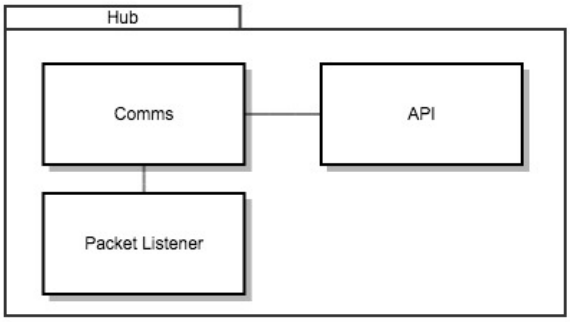


Figure 15: Hub Architecture Diagram

3.4. Cross Platform Application

The Cross Platform Application uses the Ionic framework as detailed in the Background section. This section describes the different aspects considered when creating the application, and provides a walk-through of the different views of the application

3.4.1. Requirements

- Cross Platform - The application should be Cross Platform to allow for rapid development.
- Usability - The application should maintain similar UI components to that of native apps, and should be easy to use.
- Performance - The performance of the application should be fast, and shouldn't be greatly affected by large datasets.
- Integrate into the MEAN stack - The application should display integrate into the MEAN stack with ease.
- Real Time - The application should display user information in real time.
- Consistent - The application should have a consistent state over different platforms.
- Visualisation - The application should provide a good visualisation of data held on the server.

The top level requirements applicable to the application are that it is real time, simple to install and use, and breaks down data per appliance.
The implementation of this application matches these top level requirements, and also those listed above.

3.4.2. Use Case Diagram

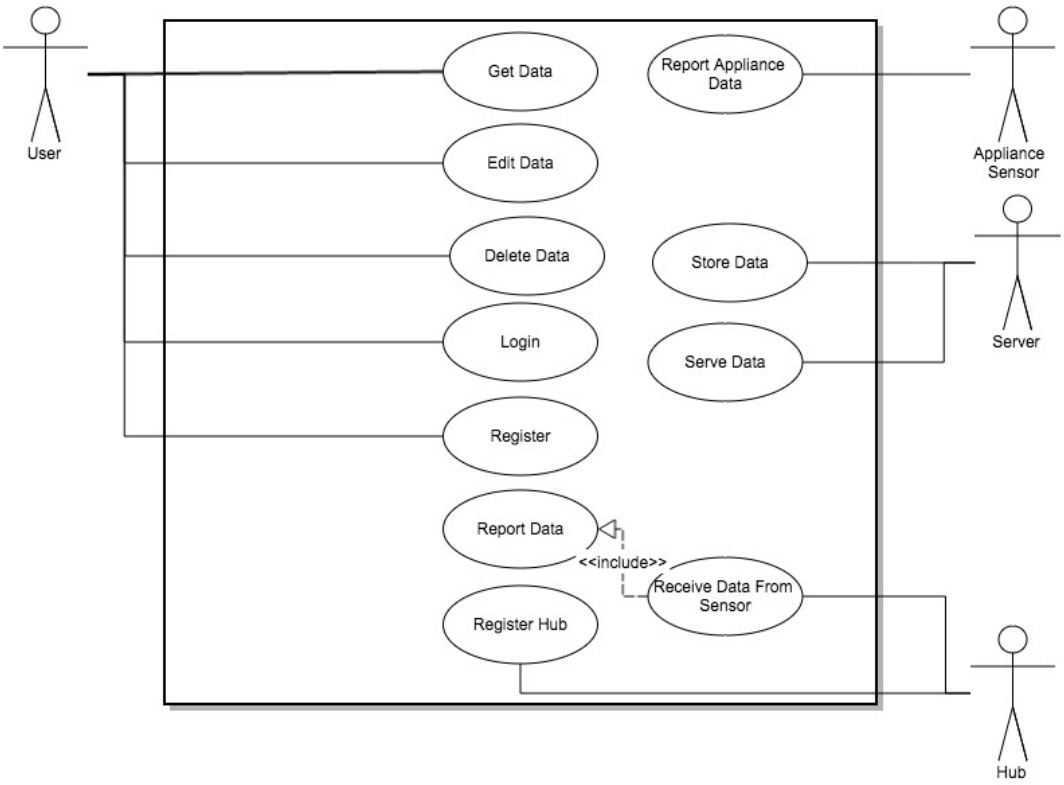


Figure 16: Use Case Diagram

Figure 16 shows the use case diagram for the system.
A **User** can perform various actions, they can get, edit and delete data, and they can also login and register.
An **Appliance sensor** can only report appliance data, and transmit it over the Earth line.
A **Hub** can receive data from an **Appliance sensor**, report data and also register with the **Server**.
The **Server** can store and serve the data stored in the database.

3.4.3. Architecture Diagram

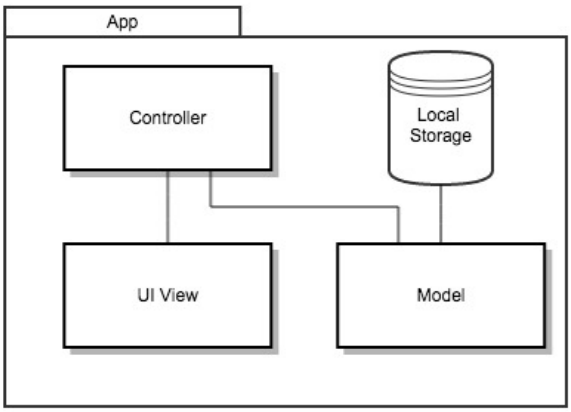


Figure 17: Application Architecture Diagram

3.4.4. Storage

There is a caching model in place for the application. When a controller is loaded, it asks the model to retrieve local data for the current day. If the model has data for the current data, the view will be updated. Otherwise, the model will fetch fresh data from the api.
Every single view implements a way of refreshing the data stored on the device. In the application a user can do this by pulling down on the view, triggering a controller event which forces the application to ask for fresh data from the server. The cached data is also updated when the data is returned from the API.
An example of this functionality can be seen in Figure 18

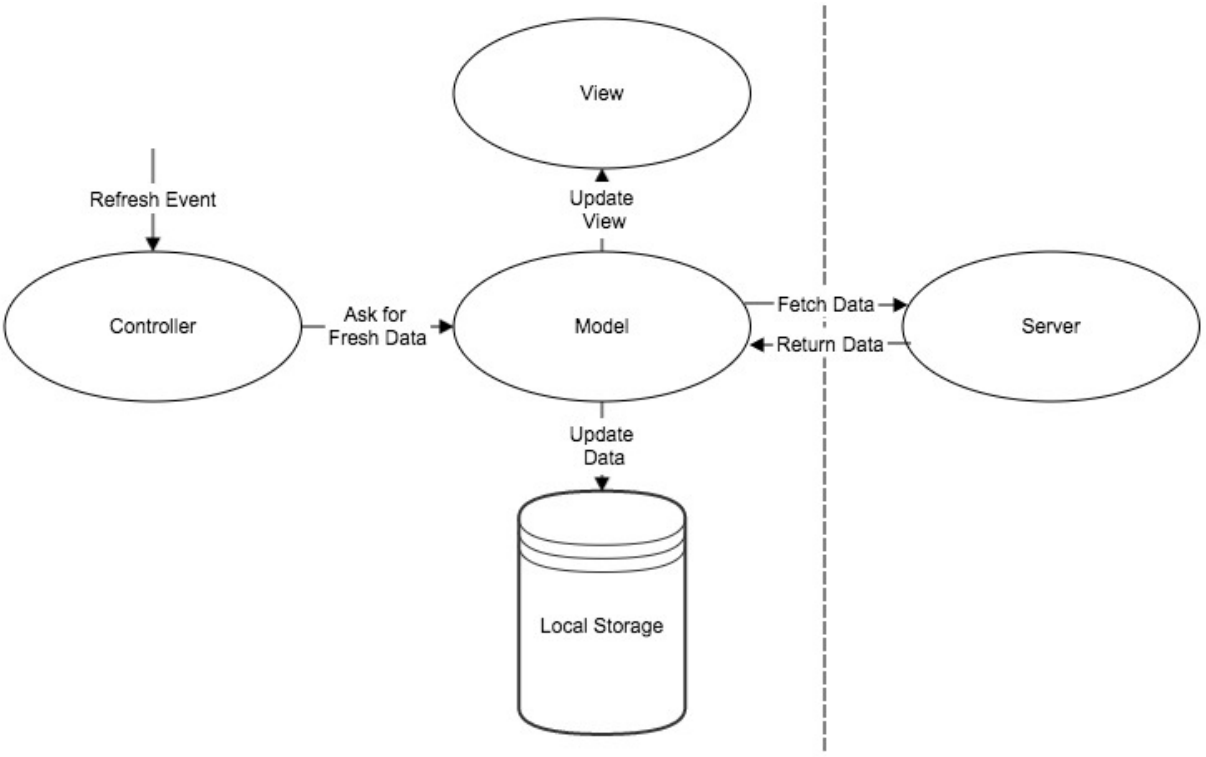


Figure 18: A diagram of the refresh event

3.4.5. MVC

Angular lends itself to the MVC [49] architecture. This means that the different elements of an application are divided into 3 distinct sections. The Model translates to application data, the view is what the user sees and the controller is what the user interacts with.
A visualisation of this can be seen below in Figure 19:

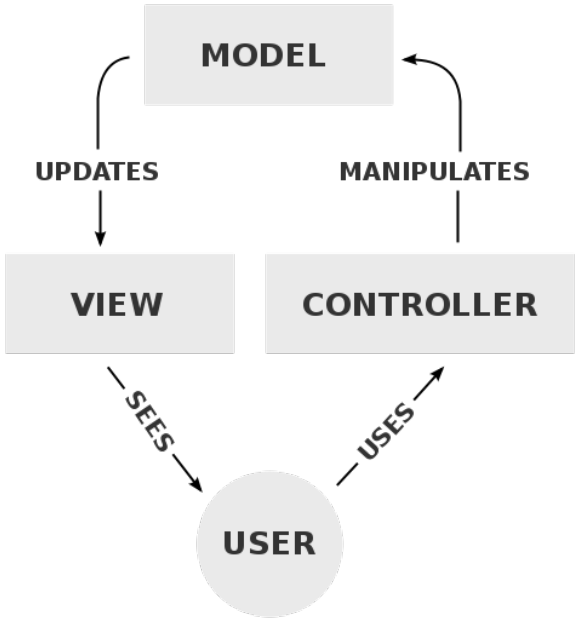


Figure 19: The model view controller architecture. [47]

3.4.5.1 Folder Structure

A benefit of MVC is a tidy folder structure with a high level of compartmentalisation for the code base as seen in Figure 20:

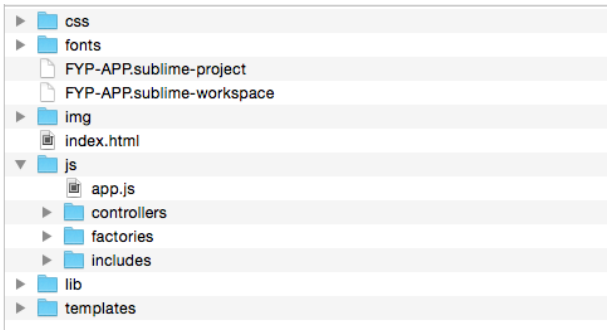


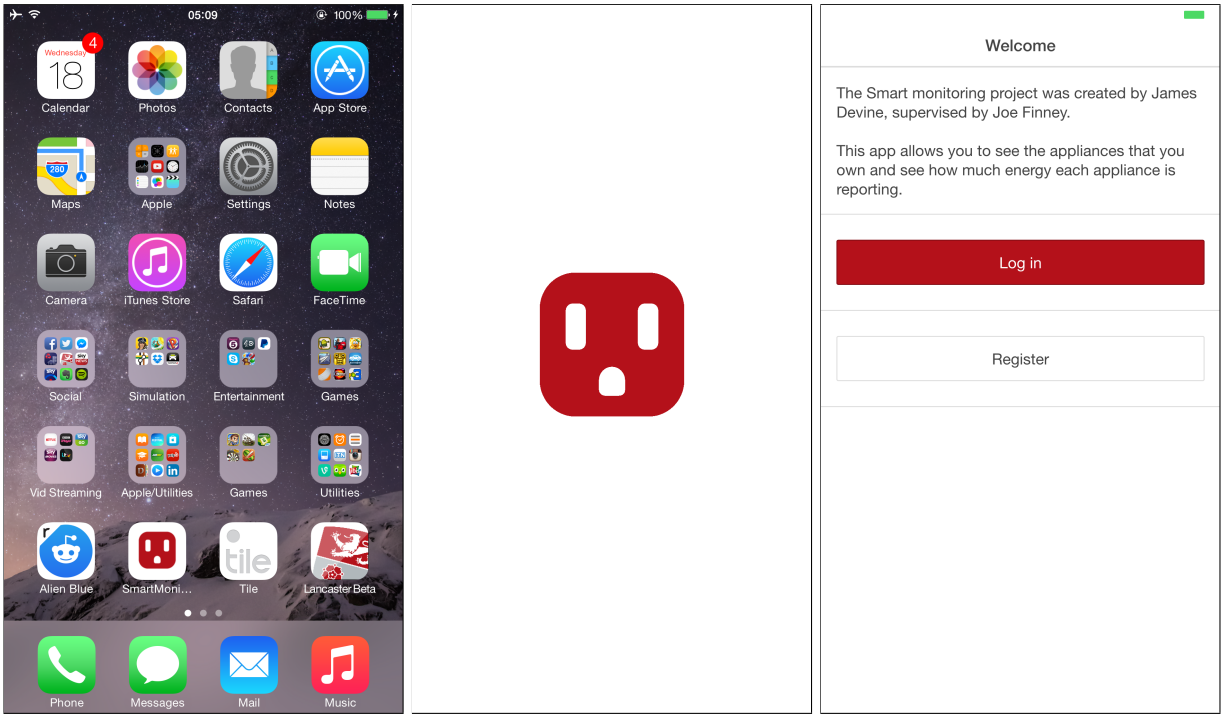
Figure 20: The folder structure for the application.

The views are contained in templates, the controllers are in the controllers folder and the models are contained within the factories folder.

3.4.6. User Interface

In this section, each view will be described. The thought behind each views' design and the UI features will also be discussed

3.4.6.1 Icon, splash and login

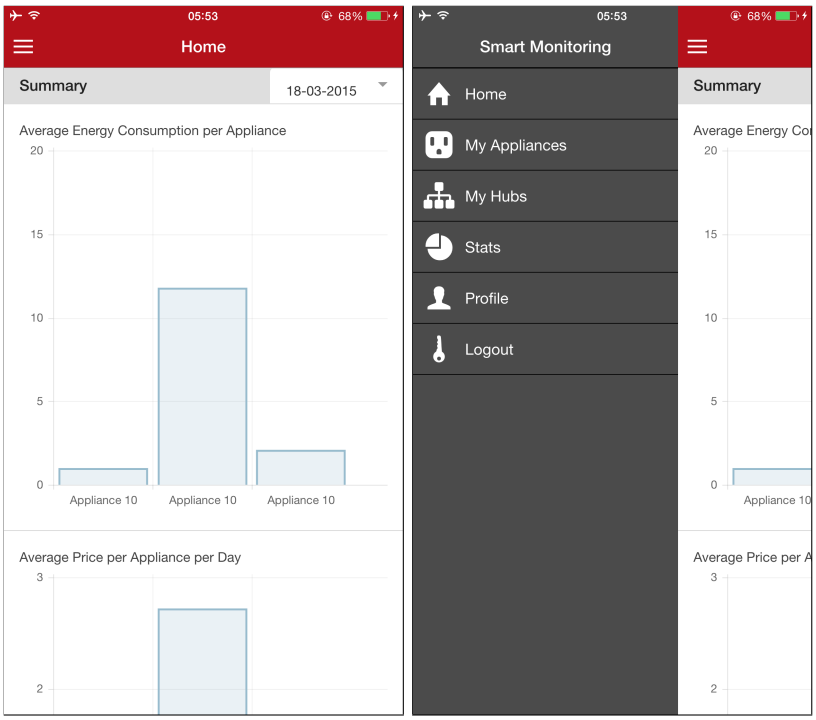


(a) The application icon (b) The application splash view (c) The application login view.

Figure 21: The application icon, splash and login views

In figure 21.a, the application icon is shown in situ on an iOS device. The icon is on the second to last row, second in from the left.
The icon uses the university colour scheme which persists throughout the entire application. The icon is also reused as a menu option in the application menu.
Figure 21.b shows the splash view which is presented every time the application is freshly opened, giving time for resources to be loaded in the background.
The final figure 21.c shows the view that is presented when the user has no local session stored on the device. If a user has logged into the application at a previous point, their user profile is stored locally and this view is skipped, allowing the seamless resuming of a previous session.
If the login button is clicked, a login form is presented, similarly, if the register option is pressed a registration form is presented to the user.

3.4.6.2 Home and Menu

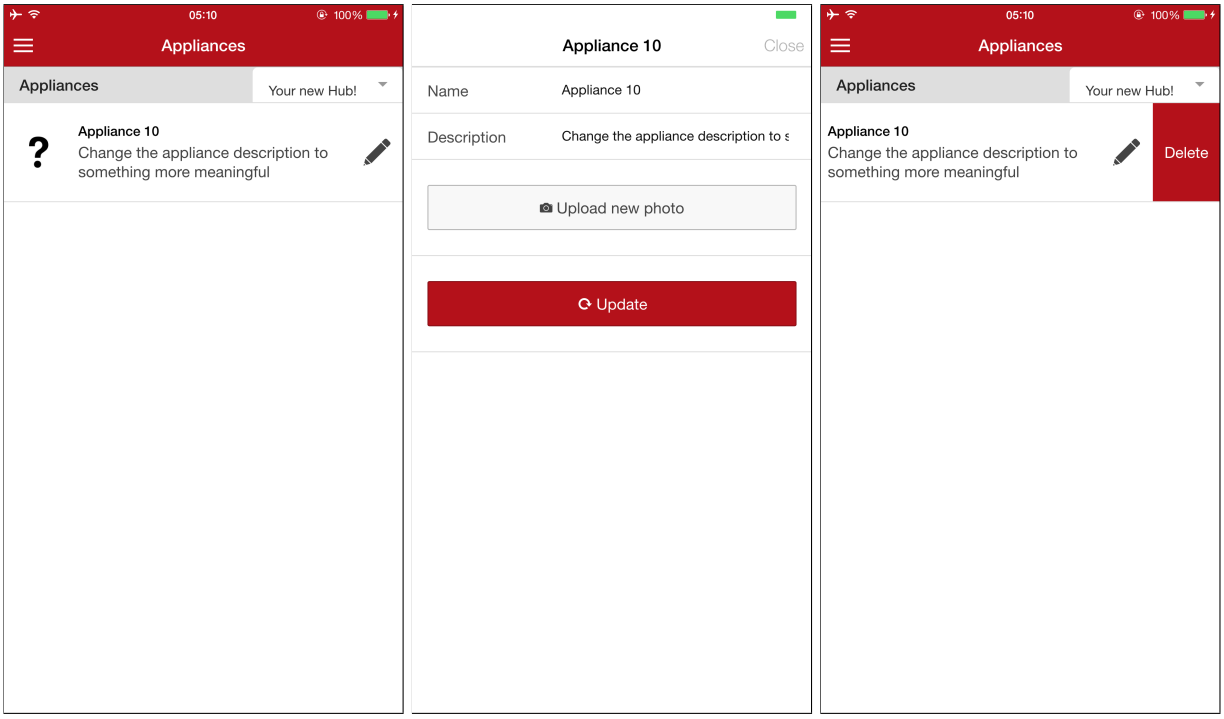


(a) The application home view. (b) The application menu view.

Figure 22: The home and menu views.

In figure 22.a, the home page is shown. The home page surmises core data from the server and displays statistics for the four most used Appliance sensors. At the time of this screen shot, data for three separate appliances across 3 different hubs was available.
A key point to mention is the use of the drop down menu in the upper right hand corner of the screen to change the date, and ultimately the data. This is a common way of changing data sets used at many points in the application.
Figure 22.b shows the application main menu, accessed via the three lines at the top left of every view (asside from login). This menu is common place in many popular applications on both Android and iOS platforms, adhering to the heuristic of consistency and standards [36]. The menu is the main method of navigating the main features of the application. To make navigation easier, icons have been used based on the rule of recognition rather than recall proposed by Neilsen [36].
The “My Appliances” option has the same icon as that used in the application icon and splash screen in 21.a and 21.b.
The “My Hubs” icon is symbolic of that of a centralised system where all of the Appliance sensors report to a single point.
The remainder of the icons used are common place in the industry, and matches the users’ understanding between the system and the real world [36].

3.4.6.3 Appliance views



(a) The list appliances view. (b) The edit appliance view. (c) Removing an appliance.

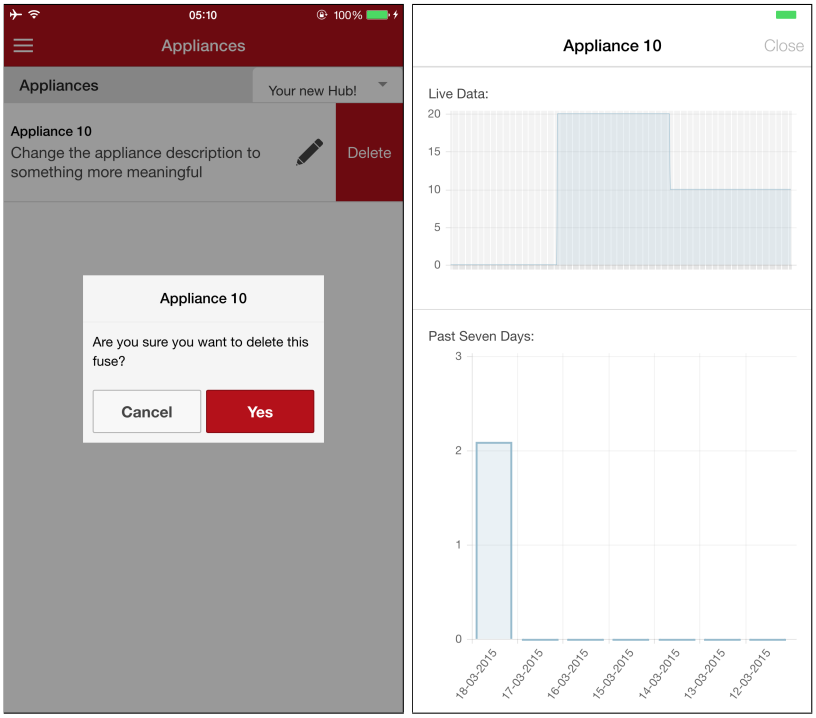
Figure 23: The appliance views.

The listing of the appliances in figure 23.a allows the user to perform a number of actions:

- A tap on the individual appliance will display the live appliance view seen in figure 24.b
- A tap on the pen will launch the edit view seen in figure 23.b
- A swipe on the list item will reveal the delete option seen in figure 23.c

All of the above actions are familiar throughout applications on both iOS and Android. The tap to view an item behaviour is seen on every single application that implements a list view, the tap the pen to edit motif is also present in many applications, and finally the swipe to delete behaviour is seen in the messages application on both platforms.
As mentioned in the section discussing the home view, there is the recurring drop down list in the upper right hand corner adhering to the heuristic of consistency and standards [36]. In this view, it allows a user to switch between various hubs that may be linked to the user account.
Figure 23.b allows a user to change the name, the description and upload an image for the selected appliance. By default the appliance name is set to "Appliance {ID}", and the description is set to "Change the appliance description to something more meaningful", with

the default image set to a question mark to indicate to the user they should change the data. The idea behind having a picture for an Appliance was to allow a user to quickly identify specific appliances connected to the Appliance sensor. For instance, if an Appliance sensor was connected to a monitor, the user would take a picture of the aforementioned monitor. The picture, combined with the name and description fields would speed up appliance identification ten fold.
Figure 23.c shows the method of deleting an appliance, swipe to the left to reveal the delete action. When the delete button is pressed, a confirmation box appears to ensure that the user wants to carry out the delete action, this can be seen in figure 24.a. This feature is used to prevent user errors in accordance with Nielsens Heuristics [36].



(a) UI prompt to ensure that the user wants to complete the delete action.

(b) The live view and the summary of the past seven days for the selected appliance.

Figure 24: The 'Are you sure you want to delete view' and the live view of the data being reported by the selected Appliance sensor.

Figure 24.b shows the live data view and the seven day summary for the Appliance sensor titled 'Appliance 10'. As a hub reports data for the selected Appliance sensor, the application will be notified, and the data will appear in the live view.
The seven day summary displays the power utilisation over the past seven days for the selected Appliance sensor.

3.4.6.4 Statistics views

(a) The carbon level statistics view.

(b) The power statistics view.

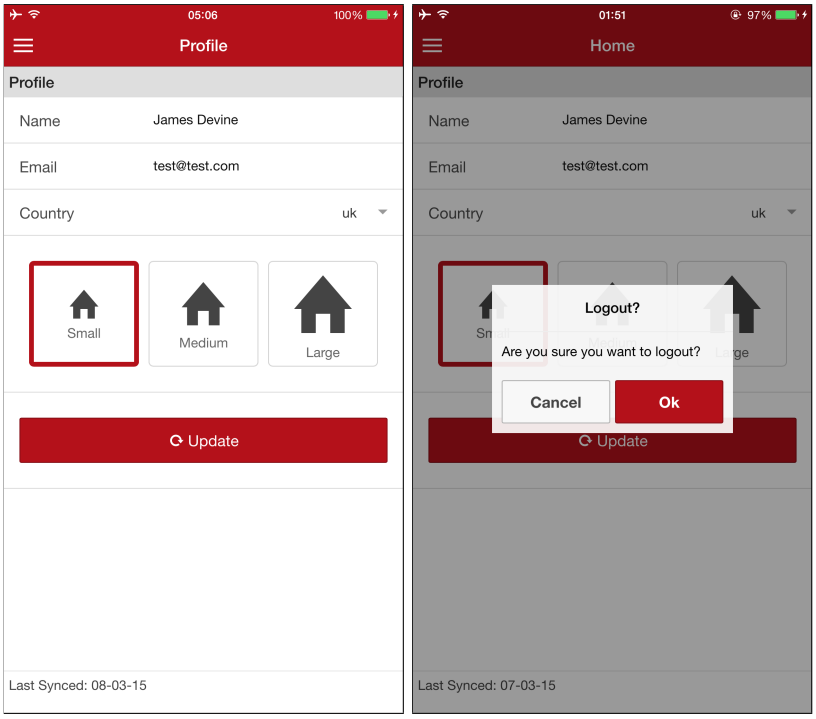
(c) The price statistics view.

Figure 25: The statistics views.

The statistics views allow a user to gain various insights into the current and historic carbon, power and price data. In the all three views date selection is available so that the user can look at information for previous days.
The aim behind this collection of views is to allow the user to evaluate the times at which they turn their big appliances on and off.
For the environmentally conscious, the carbon view (25.a) allows the user to see the live estimated level of carbon in the UK.
Throughout various points of the day, the level of carbon fluctuates.
If there is a high level of carbon, a red unhappy face is displayed, conveying to the user that the level of carbon is quite high and the use of big appliances will make a large impact to the level of carbon.
If there is a medium level of carbon, an orange indifferent face is displayed, indicating that the level of carbon is moderate so the use of big appliances will make a small impact to the overall level of carbon.
If there is a low level of carbon, a green happy face is displayed as seen in figure 25.a, this advises the user that the use of big appliances will make no significant difference to the overall carbon level.

The hope behind the availability of this statistic is to level out the carbon emissions throughout the day and change user behaviour so that the use of appliances is convenient to the planet, but not neccessarily to the user.
For users who are interested in where the sources of energy for the UK originate, the power view is available, seen in figure 25.b. This view shows real time and historic power generation statistics for the UK in percent for each source of generation.
In the doughnut charts, there is a relation between the colour and each source of generation. Coal fueled generation is represented by a black ring, nuclear power generation is represented by a luminescent green and so forth.
The final statistic shown in figure 25.c is the price data. This view shows the historic prices for the past 24 hours as well as the current price.
From the graph, really interesting insights can be gleaned. Over time there are various points where there is a spike in energy prices, this is due to the load increasing on the UK's electricity grid. Common spikes are around 10/11 am and 6/7pm where activities like boiling kettles and cooking take place.
There is a lot of research currently being placed into this field in an effort to reduce these spikes, thereby reducing price, and the load on the energy grid. A lot of energy is also wasted during these spikes, as the grid uses a demand and response system to predict energy usage, increasing or decreasing energy generation accordingly, if not all of the energy is utilised, it will be burnt off as heat.
In 2010, a study of demand and response approaches to European energy generation occurred [45]. The paper concludes that outdated metering technologies mean that the average home owner is not aware of the real time energy prices, and are not billed accordingly. Through the use of the data gathered by the project, it could allow companies to more accurately bill the home owner, as well as providing valuable insights into energy consumption. If a home owner was also aware of the impact that their appliances have on the grid, this may prompt a change in user behaviour, or encourage people to buy less energy intensive appliances. For example, a slow boiling kettle would do wonders for the grid, as the intensity of the energy consumed would be slower, the only cost being time consumed waiting for the kettle to boil.

3.4.6.5 Profile and logout



(a) The application profile view.

(b) The application logout view.

Figure 26: The profile and logout views.

The final two views of the application are the profile and logout views. In figure 26.a the profile view can be seen. This allows a user to change their profile information stored on the server. They can change their name, their email address, their country and house size. The reason a users' country and house size are stored is because it acts as another metric for the energy suppliers to analyse.
The logout view in figure 26.b is accessible through the main menu. The user will be asked if they would like to like out in order to prevent user error.
If a user clicks ok, all of the user information will be removed from local storage, the user will then be redirected to the login screen. If the user cancels the action, nothing happens.

3.5. Server

The Server consists of three core elements: the Database, the Web Site and the API.
This section will breakdown these core elements and describe how the implementation of each came about, and the design of each.

3.5.1. Requirements

- Scalable - The solution must scale to a number of servers, allowing for a distributed architecture.
- Fast - The solution needs to fast to minimise client hang time.
- Development Time - The speed of development for the chosen solution should be rapid to meet the tight time constraints of the project.
- RESTful - The application should be RESTful, using traditional HTTP methods and status codes to provide feedback to the application.

The additional requirements mentioned above are specific to the server. All of these requirements have been matched in the implementation as you will read.

3.5.2. Database

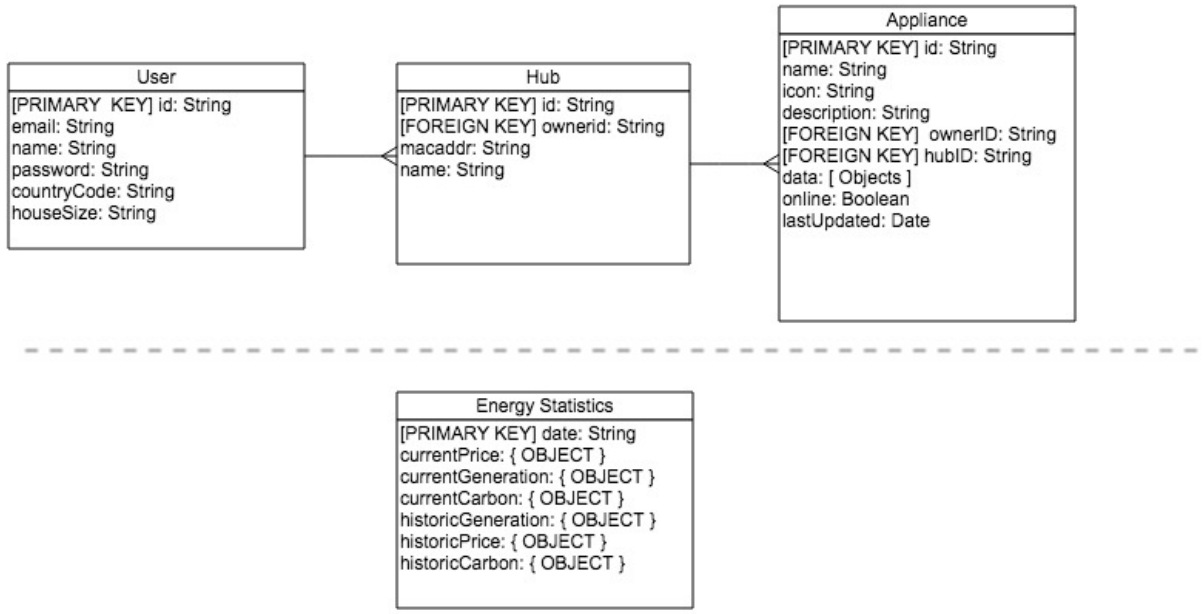


Figure 27: Entity Relationship Diagram

The collection (database) contains four documents (tables) core to the operation of the platform.

The User document stores all details for the users registered with the platform, the Hub document stores all of the information that represents a Hub object and the Appliance document stores all of the details pertaining to an Appliance sensor including the data transmitted from the Hubs.
The mappings between the documents are as follows:

- A User has a unique identifier.
- A Hub has a unique identifier and also has an ownerid that maps onto the User document through the use of the UserID.
- An Appliance has a unique identifier. An Appliance also contains an ownerID and a hubID which maps onto both the User document and the Hub document.

The documents were constructed in this manner because an Appliance sensor can have an ID between 0 and 4095 (2^{12}). If a user had multiple installs in different locations, thereby requiring the use of more than one hub, without the Hub ID there would be duplicate Appliance IDs. A User also wouldn't want to see all of the Appliance sensors for every single location.
Using the HubID allows for the easy segmentation between locations and removes the possible collision of Appliance sensor IDs.

The relationships between the documents are thusly:

"A User can have many Hubs and a Hub can have many Appliances"

The Energy Statistics document is used to store data scraped from various APIs, which will be explained further on in this document.
Energy Statistics has a primary key of date, and bears no relation to any other documents.

3.5.3. Web Site

Aside from the cross platform application, the server contains some additional front end user interfaces with the aims of detailing more information about the project, as well as allowing others to quickly register and join the project.

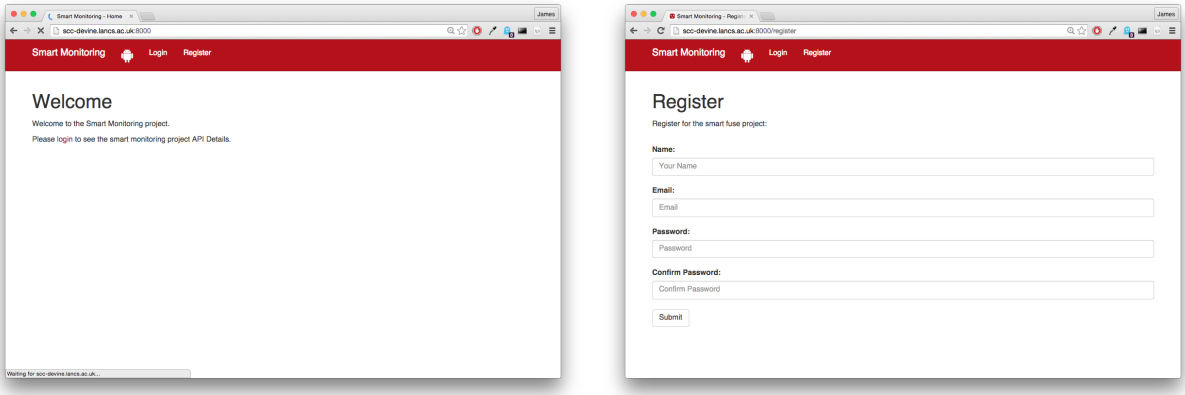


Figure 28: The home and registration views of the site.

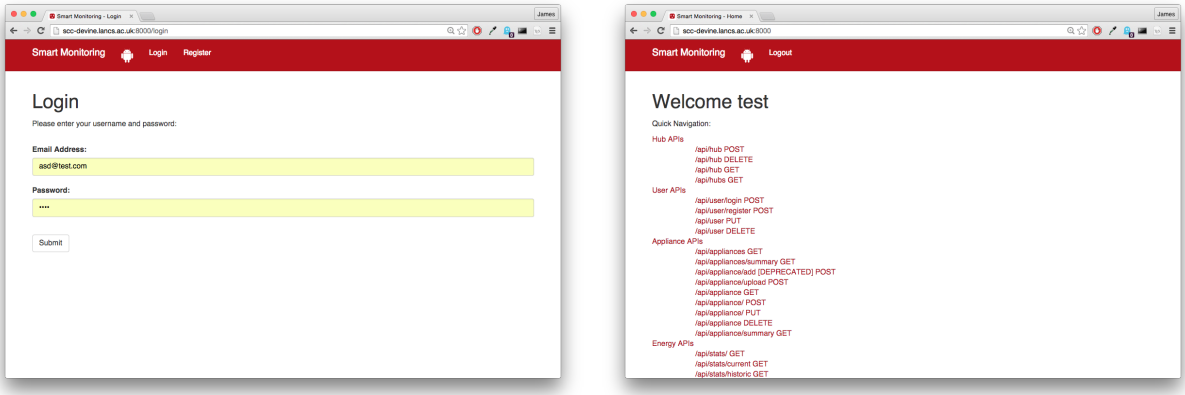


Figure 29: The login and home page after login.

The site is built using the Jade [32] template engine and a customised bootstrap [37] theme, which means that the site is also responsive.
Users are able to download the most recent build of the android application from this site, unfortunately due to the restrictions imposed by Apple, the same cannot be said for the iOS version of the application.
Once a user logs in, the API documentation is then made available to them. A User can use this information to create their own applications, or recommend additional functionality. Users can also fork the project on Github, to collaborate and promote open source practices.

3.5.4. Additional Features

3.5.4.1 Socket.io

Socket.io [42] is a javascript library compatible with both angular and node js. It allows bidirectional event-based communication between client and server.
Socket.io is used to update clients with real time data when a data point is added to an

Appliance by a hub, and the application is running.
Every time a user opens the app, they join a socket indexed by their user ID. This allows real time information to be push to clients even when the same user is logged in on different platforms. When a data point is received, a look up is performed using the user ID provided in the /api/appliance POST request. The event is then broadcast down the socket if it exists, and is then digested by the client.

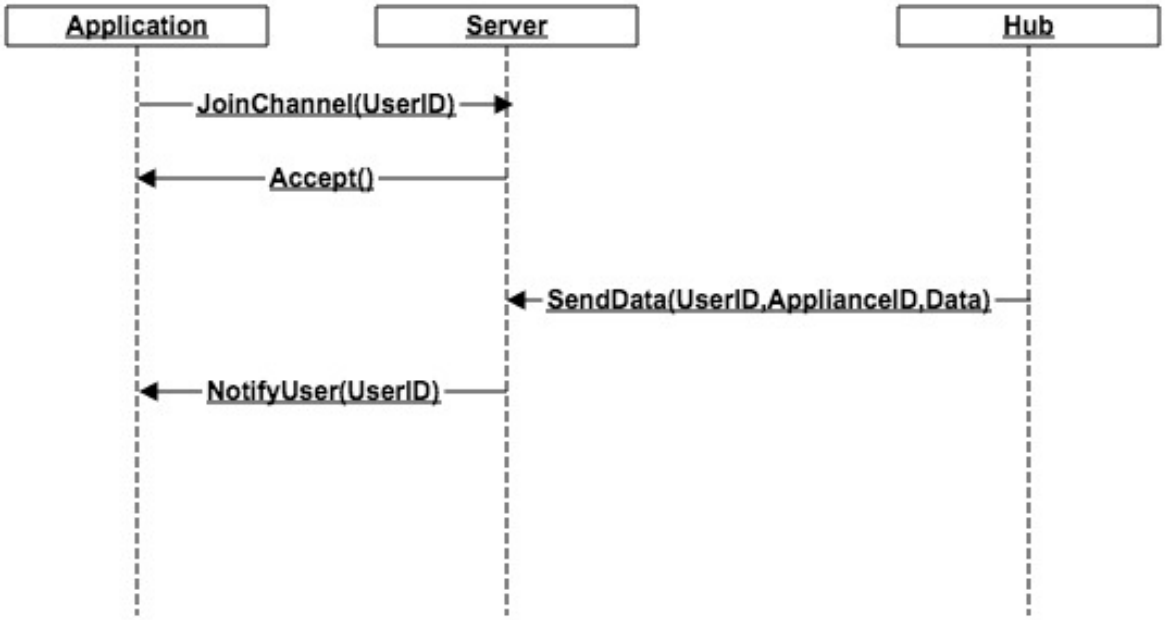


Figure 30: The sequence diagram of live data reporting using Socket.io

Socket.io uses the well known method of sockets and falls back to long polling to keep connections open if the use of sockets aren't permitted by the configuration. It also enables the detection of disconnected clients through the use of a keep-alive pulse every x seconds.

3.5.4.2 Data Scraping

In order to produce the statistics seen in the application, the server is required to fetch data from a number of publicly available APIs.
If the current minute is: 0, 15, 30 or 45, the server harvests data from the following URLs as discussed in the Existing APIs section:

- `http://www.earth.org.uk/_gridCarbonIntensityGB.xml`
- `http://www.nordpoolspot.com/api/marketdata/page/1639`
- `http://www.bmreports.com/bsp/additional/soapfunctions.php?element=generationbyfuel`

- `http://www.nordpoolspot.com/api/marketdata/page/4708`

All of the APIs return data in XML format. The server transposes them into easy to digest JSON objects, and stores them in the database to be retrieved by clients at a later date.

3.5.4.3 Adding Appliance Data Points

When a Hub reports a data point for an Appliance, the server optimises the packet to decrease the number of data points stored server side. The main reason for implementing the system in this manner entirely rests on the fact that keeping every single data point would result in an unnecessarily over precise and bloated system.
This optimisation is a summed average of the data points over a configurable period of time and is as follows:

$$\text{power utilisation } = \frac{\text{sum of the reported voltages}}{\text{number of samples}}$$

The configurable period variable dictates the number of available samples for a given day. To calculate this value, the following will need to be evaluated:

$$\text{available samples } = \frac{\text{seconds in a day}}{\text{configurable period in seconds}}$$

In a full day there would be a total of 10800 samples for each appliance if each appliance sensor transmitted data every 8 seconds.
Using the configurable period variable, the number of samples can be reduced significantly. Currently, the configurable period is set to 10 minutes. Evaluating the previous calculation will result in the number of available samples to be determined:

$$\frac{86400}{600} = 144 \text{ samples}$$

Each sample would then contain the average of 75 data points reported by the Hub if the Appliance sensors were reporting once in an 8 second interval.
The downside to this approach is that the platform will not immediately react to sharp changes in voltages reported by the Appliance sensors.
An interesting point to mention, is that when a new data point is reported by a hub, the real time energy price is fetched and used to calculate the costing of that data point. This feature enables the Users to see accurately the price of the energy they are consuming.

3.5.5. Architecture Diagram

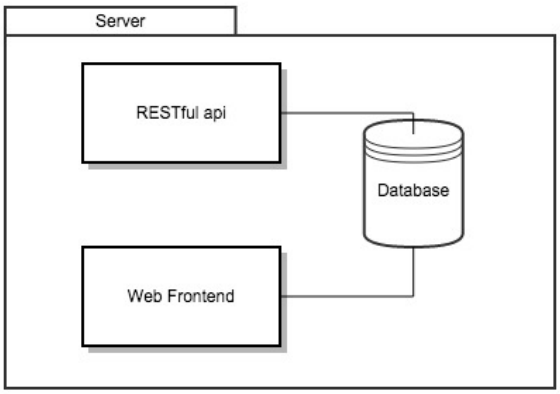


Figure 31: Server Architecture Diagram

3.5.6. RESTful API

REST stands for Representation State Transfer and outlines a software architecture style for creating scalable web services.
REST is stateless, meaning that the clients' state isn't maintained by the server and is instead maintained client side.
True REST uses HTTP status codes and request types. The following request types and status codes are prominent throughout the API:

Request Types:

- GET - Read an item.
- POST - Create an item.
- PUT - Edit an existing item.
- DELETE - Remove an item.

Each request type above is used as specified by w3.org [46]. The four request types map onto the essential CRUD operations that are performed in most data-oriented systems.

Status Codes:

- 200 - Ok.
- 201 - Created.
- 400 - Bad request.
- 403 - Forbidden.

- 404 - Not found.
- 500 - Internal server error.

The usage of the above status codes is in compliance with those specified by w3.org [46]. They are used to provide primitive error detection capabilities to the client.
To ensure that all requests are valid, parameters are checked before carrying out every request, if one or more parameters are missing, the HTTP status code 400 Bad Request is returned.
Some requests like /api/appliances GET will return an empty array if there are no appliances available for a user instead of displaying an error code. This is intentional because the request has been successfully carried out, the result being that the user doesn't have any appliances.
Mostly when a transaction has been carried out successfully, the status code returned will be 200 Ok. There is an exception to this rule, like when performing a POST request. If the POST request is valid, a 201 Created status code will be returned indicating that the object has been created and is ready to use.
500 Internal Server Error is only returned when something unexpected has happened, and the server cannot fulfill the request.
Every time an API is called, the server provides additional feedback on the operation that just occurred on top of the HTTP status codes.
If an operation has been completed successfully the following JSON will be returned along with any additional data:

```
1 {success: "SUCCESS MESSAGE",....}
```

If an operation fails to complete, the following JSON will be returned along with any additional data:

```
1 {error: "ERROR MESSAGE",....}
```

3.5.6.1 Hub API's

/api/hub

Type: **POST**

Description: Links a users' account with a hub using the userID and the hubID. The hub can then be viewed in the application.

Parameters:

- userID: A string containing the id of the user
- hubID: A string containing the hub the user is trying to link

Expected Response:

Status Code: **201**

```
{success: "Hub added"}
```

Error Response:

Status Code: **400**

```
{error: "Hub couldn't be added"}
```

/api/hub

Type: **DELETE**

Description: Removes a hub from the database, effectively unlinking a user account from the hub itself. The hub can still report data and will be aware that it is no longer linked to a user.
Any data prior to this request will be purged from the database.

Parameters:

- userID: A string containing the id of the user. This is used to ensure that owners can't remove each others' hubs.
- hubID: A string containing the id of the hub the user is trying to remove.

Expected Response:

Status Code: **200**

```
{success: "Hub removed"}
```

Error Response:

Status Code: **400**

```
{error: "Hub couldn't be removed"}
```

/api/hub

Type: **GET**

Description: Retrieves a hub object using the mac address of the hub. This API is used by the hub to effectively register and obtain its credentials such as the ownerID, and its hubID. This API call creates a hub object if this hub is new, otherwise it will retrieve the object from the database

Parameters:

- macaddr: The MAC address of the hub trying to fetch its details!

Expected Response:

Status Code: **200**

```
{success:"Hub retrieved!", hub:{HUB OBJECT}}
```

Error Response:

Status Code: **400**

```
{error:"Hub not linked!", hub:{HUB OBJECT}}
```

/api/hub

Type: **PUT**

Description: Updates a hubs' details in the database. The only editable feature is the name of the hub.
The name is used in the front end to allow the user to identify specific hubs rather than hunting for a specific UID.

Parameters:

- userID: A string containing the id of the user. This is used to ensure that owners can't remove each others' hubs.
- hubID: A string containing the id of the hub the user is trying to update.
- name: A string containing the new name of the hub.

Expected Response:

Status Code: **200**

```
{success: "Hub updated"}
```

Error Response:

Status Code: **500**

```
{error: "Hub not updated"}
```

/api/hubs

Type: **GET**

Description: Retrieves a list of hubs owned by the user

Parameters:

- userID: A string containing the id of the user

Expected Response:

Status Code: **200**

```
{success:"Hubs retrieved!", hubs:[HUB OBJECTs]}
```

Error Response:

Status Code: **200**

```
{success:"Hubs retrieved!", hubs:[]}
```

3.5.6.2 User API's

/api/user/

Type: **GET**

Description: Logs in a user and returns the user object for use with subsequent requests

Parameters:

- email: A string containing the email address of the user.
- password: A string containing the password for the user, that was previously defined in the registration view.

Expected Response:

Status Code: **200**

```
{success: "User logged in!",user: {USER OBJECT} }
```

Error Response:

Status Code: **403**

```
{error: "User credentials incorrect"}
```

/api/user/

Type: **POST**

Description: Registers a user with the Smart Monitoring Project.

Parameters:

- email: A string containing the email address of the user.
- password: A string containing the password for the user.
- name: A string containing the name of the user.

Expected Response:

Status Code: **201**

```
{success: "User registered!",user: {USER OBJECT} }
```

Error Response:

Status Code: **400**

```
{error: "User already registered!"}
```

/api/user

Type: **PUT**

Description: Updates the user object stored in the database.

Parameters:

- userID: A string containing the id of the user.
- name: A string containing the name of the user.
- email: A string containing the email address of the user.
- countryCode: A string containing the countryCode of the user.
- houseSize: A string containing the houseSize of a user.

Expected Response:

Status Code: **200**

```
{success:"User updated!"}
```

Error Response:

Status Code: **500**

```
{error:"User could not be updated"}
```

/api/user

Type: **DELETE**

Description: Deletes the user object stored in the database for the given userID.

Parameters:

- userID: A string containing the id of the user.

Expected Response:

Status Code: **200**

```
{success:"User removed!"}
```

Error Response:

Status Code: **500**

```
{error: "User couldn't be removed"}
```

3.5.6.3 Appliance API's

/api/appliances

Type: **GET**

Description: Gets the appliances based on a supplied userID.

Parameters:

- userID: A string containing the id of the user - retrieved from "/api/user/login"

Expected Response:

Status Code: **200**

```
{success: "appliances retrieved",appliances: [appliance OBJECTS]}
```

Error Response:

Status Code: **200**

```
{success: "appliances retrieved",appliances: []}
```

/api/appliances/summary

Type: **GET**

Description: Gets the users' top four appliance summaries for the given date.

Parameters:

- userID: A string containing the id of the user - retrieved from "/api/user/login".
- date: A string containing the desired date for summary in the format DD-MM-YYYY.

Expected Response:

Status Code: **200**

```
{success: "appliance summary retrieved",summary: {summary object}}
```

Error Response:

Status Code: **200**

```
{success: "appliance summary retrieved",summary: {}}
```

/api/appliance/upload

Type: **POST**

Description: Uploads an image in base 64 format to the server. This feature is used to help users identify the appliance that is connected to each appliance.

Parameters:

- userID: A string containing the id of the user - retrieved from "/api/user/login"
- applianceID: The ID of the appliance to upload an icon for.
- hubID: The ID of the hub which the appliance is reporting from.
- image: The base 64 string of the image.

Expected Response:

Status Code: **201**

```
{success: "Image uploaded"}
```

Error Response:

Status Code: **400**

```
{error: "Image upload failed"}
```

/api/appliance

Type: **GET**

Description: Gets a singular appliance based on a user ID and an appliance ID

Parameters:

- userID: A string containing the id of the user - retrieved from "/api/user/login"
- applianceID: The ID of the appliance to fetch.
- hubID: The ID of the hub which the appliance is reporting from.

Expected Response:

Status Code: **200**

```
{success: "appliance retrieved",appliance: [appliance OBJECT]}
```

Error Response:

Status Code: **404**

```
{error: "appliance not found"}
```

/api/appliance/

Type: **POST**

Description: If the appliance exists it adds an item to the data array - otherwise it creates an appliance object. This API call also notifies any users that are subscribed to the appliance and have the app open.
This API is called by the hub.

Parameters:

- userID: A string containing the id of the user - retrieved from "/api/user/login"
- applianceID: The ID of the appliance to add.
- applianceVal: The data to be added
- hubID: The ID of the hub which the appliance is reporting from.

Expected Response:

Status Code: **201**

```
{success: "appliance data added"}
```

Error Response:

Status Code: **404**

```
{error: "appliance not found"}
```

/api/appliance/

Type: **PUT**

Description: Updates the details of the appliance stored in the database.

Parameters:

- userID: A string containing the id of the user - retrieved from "/api/user/login".
- applianceID: The ID of the appliance to upload an icon for.
- hubID: The ID of the hub which the appliance is reporting from.
- applianceName: The new name of the appliance.
- applianceDescription: The new description of the appliance.

Expected Response:

Status Code: **200**

```
{success: "appliance edited"}
```

Error Response:

Status Code: **403**

```
{error: "appliance couldn't be edited"}
```

/api/appliance

Type: **DELETE**

Description: Removes an appliance from the Smart Monitoring Project

Parameters:

- userID: A string containing the id of the user - retrieved from "/api/user/login"
- applianceID: The ID of the appliance to remove.
- hubID: The ID of the hub which the appliance is reporting from.

Expected Response:

Status Code: **200**

```
{success: "appliance removed"}
```

Error Response:

Status Code: **403**

```
{error: "appliance couldn't be removed"}
```

/api/appliance/summary

Type: **GET**

Description: Fetches the summary for the past seven days for the given appliance.

Parameters:

- userID: A string containing the id of the user - retrieved from "/api/user/login"
- applianceID: The ID of the appliance to fetch the summary for.
- hubID: The ID of the hub which the appliance is reporting from.

Expected Response:

Status Code: **200**

```
{success: "appliance summary retrieved",summary:{SUMMARY OBJECT}}
```

Error Response:

Status Code: **404**

```
{error: "appliance not found"}
```

3.5.6.4 Energy API's

/api/stats/

Type: **GET**

Description: Retrieves all available stats held for the date passed.

Parameters:

- date: A string in the format DD-MM-YYYY

Expected Response:

Status Code: **200**

```
{success: "Stats retrieved!",stats:{ STATS OBJECT } }
```

Error Response:

Status Code: **404**

```
{error: "Stats are not available for this date.", stats: {} }
```

/api/stats/current

Type: **GET**

Description: Retrieves all available stats with field names beginning with current, for the date passed.

Parameters:

- date: A string in the format DD-MM-YYYY

Expected Response:

Status Code: **200**

```
{success: "Stats retrieved!",stats:{ STATS OBJECT } }
```

Error Response:

Status Code: **404**

```
{error: "Stats are not available for this date.", stats: {} }
```

/api/stats/historic

Type: **GET**

Description: Retrieves all available stats with field names beginning with historic, for the date passed.

Parameters:

- date: A string in the format DD-MM-YYYY

Expected Response:

Status Code: **200**

```
{success: "Stats retrieved!",stats:{ STATS OBJECT } }
```

Error Response:

Status Code: **404**

```
{error: "Stats are not available for this date.", stats: {} }
```

4. Testing and Evaluation

In this section, testing of each part of the final platform will be described. The results from each test will be analysed and evaluated.

4.1. Appliance Sensor

4.1.1. Test Outline

Important tests were carried out on the Appliance Sensor to determine the maximum Baud rates rates that could be achieved through the identified hardware, as well as considering how the length of transmission cable (Earth line) would affect error rate and reception by the Hub.

A number of different assembly files were created to test different baud rates (changing clock speed if required). The assembly files generated packets at a fixed timing to reliably determine reception on the Hub-side. The selected baud rates were: 9600, 19200 and 57600.

To test the reliability of transmissions over varying lengths of cable, different lengths were created and used in the experiment, these lengths being: 0m, 1m, 5m, 10m, 15m, 19m, 100m. Combining these different lengths generated lengths other than those specified. The final selected cable lengths were: 0m, 1m, 5m, 10m, 20m, 30m, 50m, 100m, 150m.

The test devised combined baud rates with the selected cable lengths. For each cable length, 1000 packets were sent at a rate of 100 packets per second.

On the Hub, a testing script was created based on the final version of the software implementation. This script provided enhanced logging which allowed the generation of various data sets in CSV format.

Google Sheets was used to process the data into summaries which were used to generate the following charts.

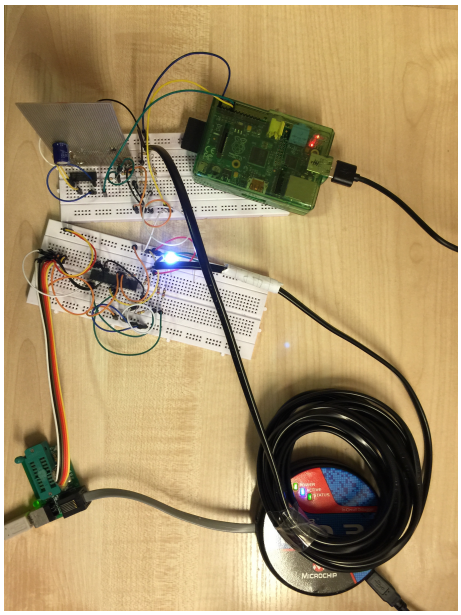


Figure 32: The test rig for evaluating the capabilities of the Appliance Sensor (5m).

4.1.2. Results

In the coming graphs, the following terminology is used:

- Discarded Packet - A discarded packet is where the Hub is in the middle of processing a packet, and sees no more activity for a period of 1500 microseconds. It will then time out and wait for the next packet to begin. In other words, a framing error.
- Corrected Packet - A corrected packet is when the Hub captures a packet, and is sent to the Decoder. The Decoder processes a packet, and identifies an incorrect bit, and corrects it.
- Uncorrectable Packet - An Uncorrectable packet is where a packet is fully captured and sent to the Decoder. The Decoder then determines that there are 2 or more bit errors, and will therefore cease processing of said packet.

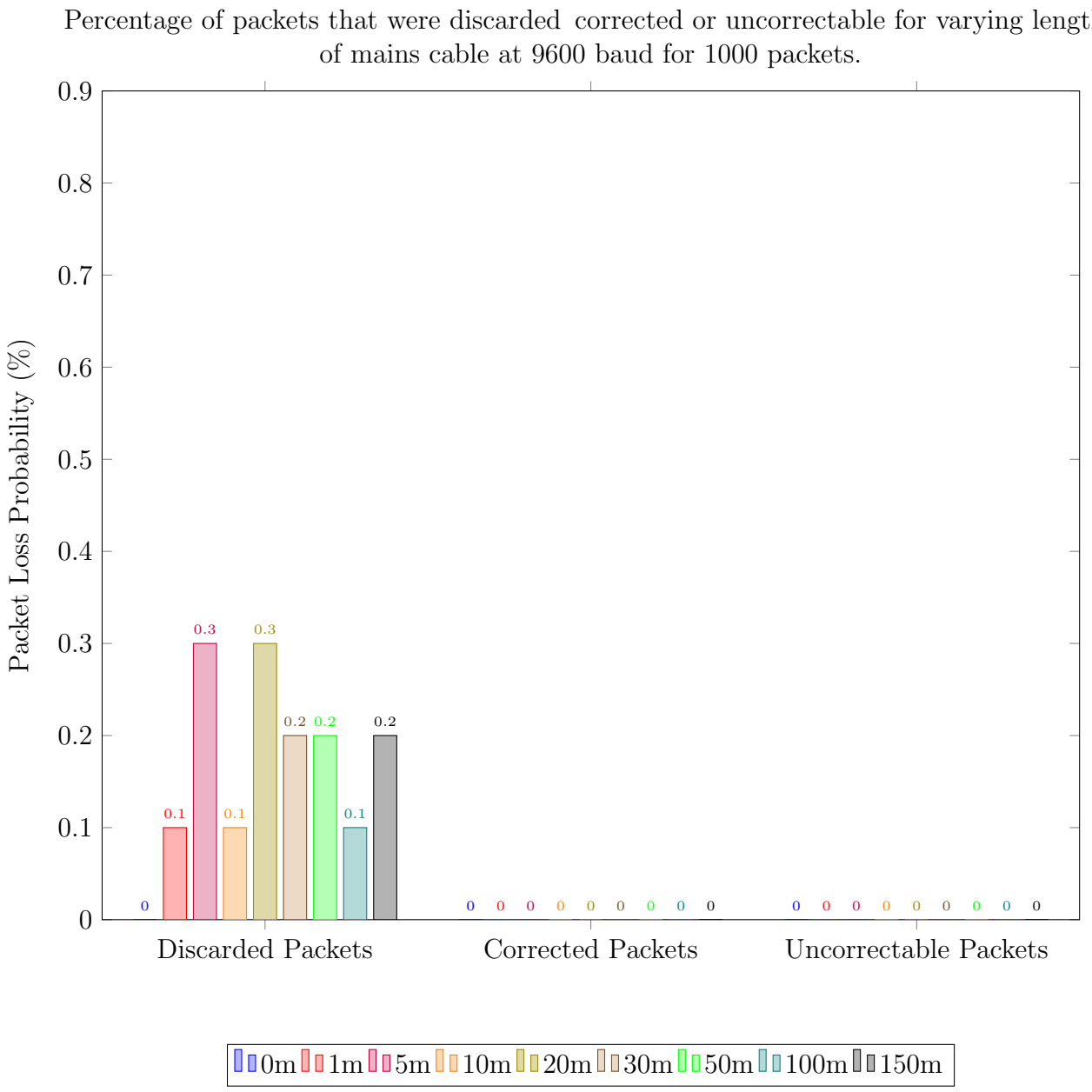


Figure 33: The results from the testing of the Appliance Sensor.

Figure 33 shows the average number of discarded packets, the average number of corrected packets and the average number of uncorrectable packets.
The graph shows that there isn't any correlation between distance and the average number of corrected, and uncorrectable packets. It also shows that there isn't a distinct correlation between discarded packets and cable length.
The experiment presented an unknown issue. The hardware cannot cope with a baud rate higher than 9600, as shown by the singular graph above. The same experiment was repeated on the previously identified baud rates, and the hub was unable to identify any clean packets.

The reason for this was due to the OpAmp which has a slew rate of 0.5 volts per microsecond. It deformed the packets received hub side, meaning that that could not be recognised by the Hub. The effect of the slew rate can be seen in the following image:

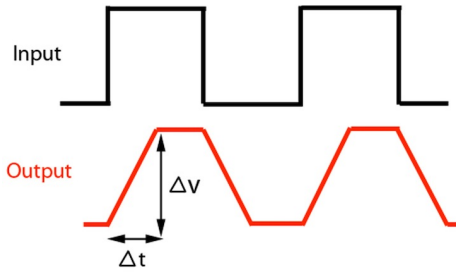


Figure 34: A graphical representation of how the slew rate affects square waves [6]

4.1.3. Conclusions

There is only one conclusion that can be drawn from the evaluation of the Appliance Sensor: The baud rate is low, and the selected OpAmp means that a faster baud rate cannot be used.
A different OpAmp with a lower slew rate could be used to improve the data rate in future so to ensure that the hub can correctly receive packets.

4.2. Hub

4.2.1. Test Outline

The next test to be performed concerned only the hub. This test was designed to assess the overall effectiveness of error detection and correction Hub-side.
The testing apparatus consisted of two Raspberry Pi's. One Pi would act as the Appliance Sensor, and send a stream of packets to the Hub.
A whole new program was devised that generated packets, recorded each packet and it's type, and the time it was sent from the 'Appliance Sensor' Pi.
The packets sent from the 'Appliance Sensor' Pi consisted of some percentage of each type of packet: a correct packet, a correctable packet, an uncorrectable packet and a packet with an incorrect checksum. These packets were randomly ordered and streamed to the Pi.
The script used for testing the Appliance Sensor was used again to record the received packets, and each type the received packet had for the Hub.
The two data sets generated from the Hub and the 'Appliance Sensor' Pi could then be correlated and cross examined to produce statistics about the capture rate and the different detection rates of each type of packet.
Google Sheets was once again used to collate and analyse the data into a short summary that could be used to generate graphs.
Both programs logged the data in CSV format making it extremely easy to transform into a spreadsheet.

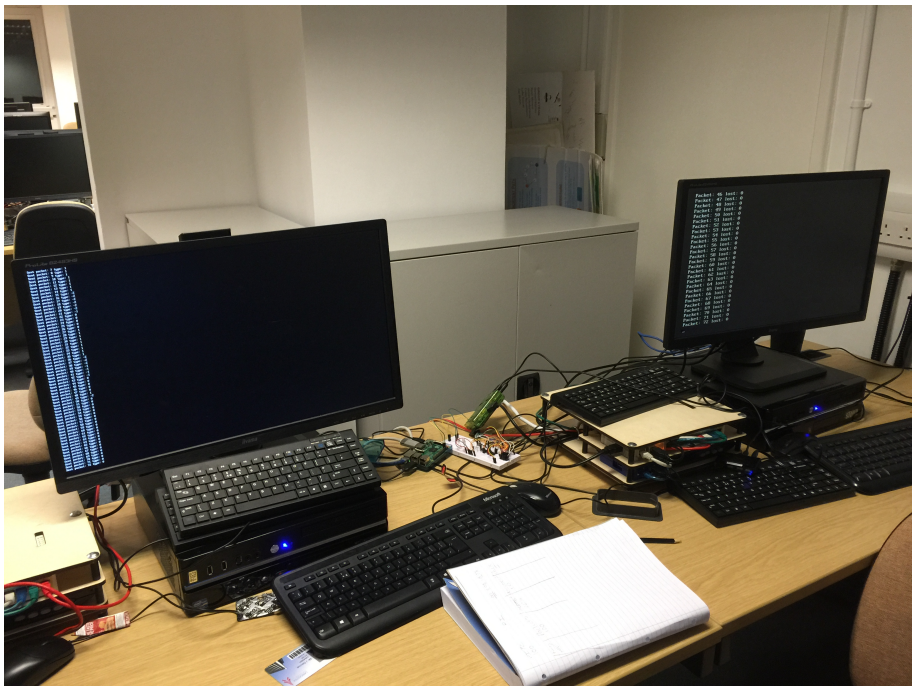


Figure 35: The test rig for detecting the error correction and detection rates for the Hub.

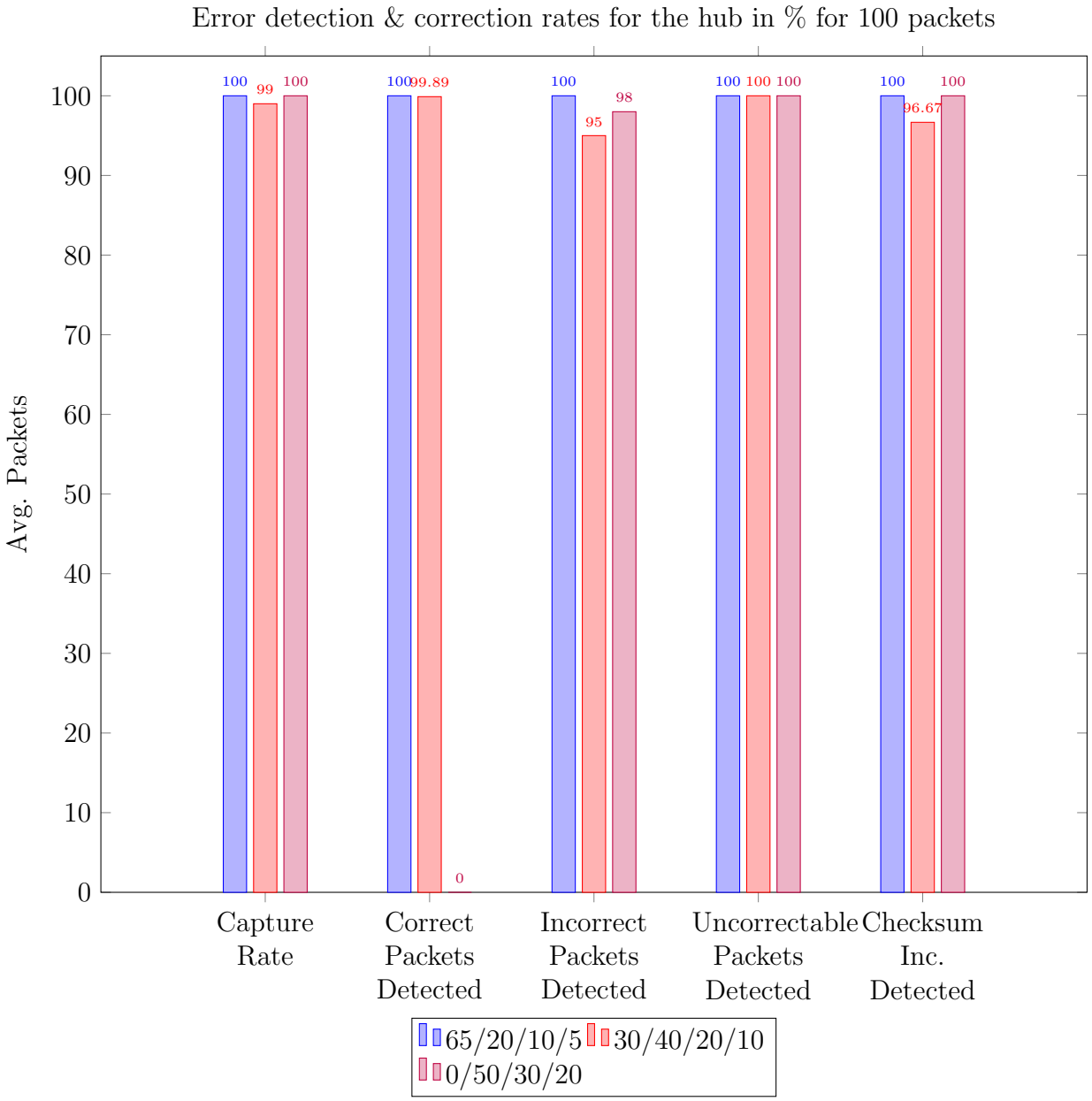
4.2.2. Results

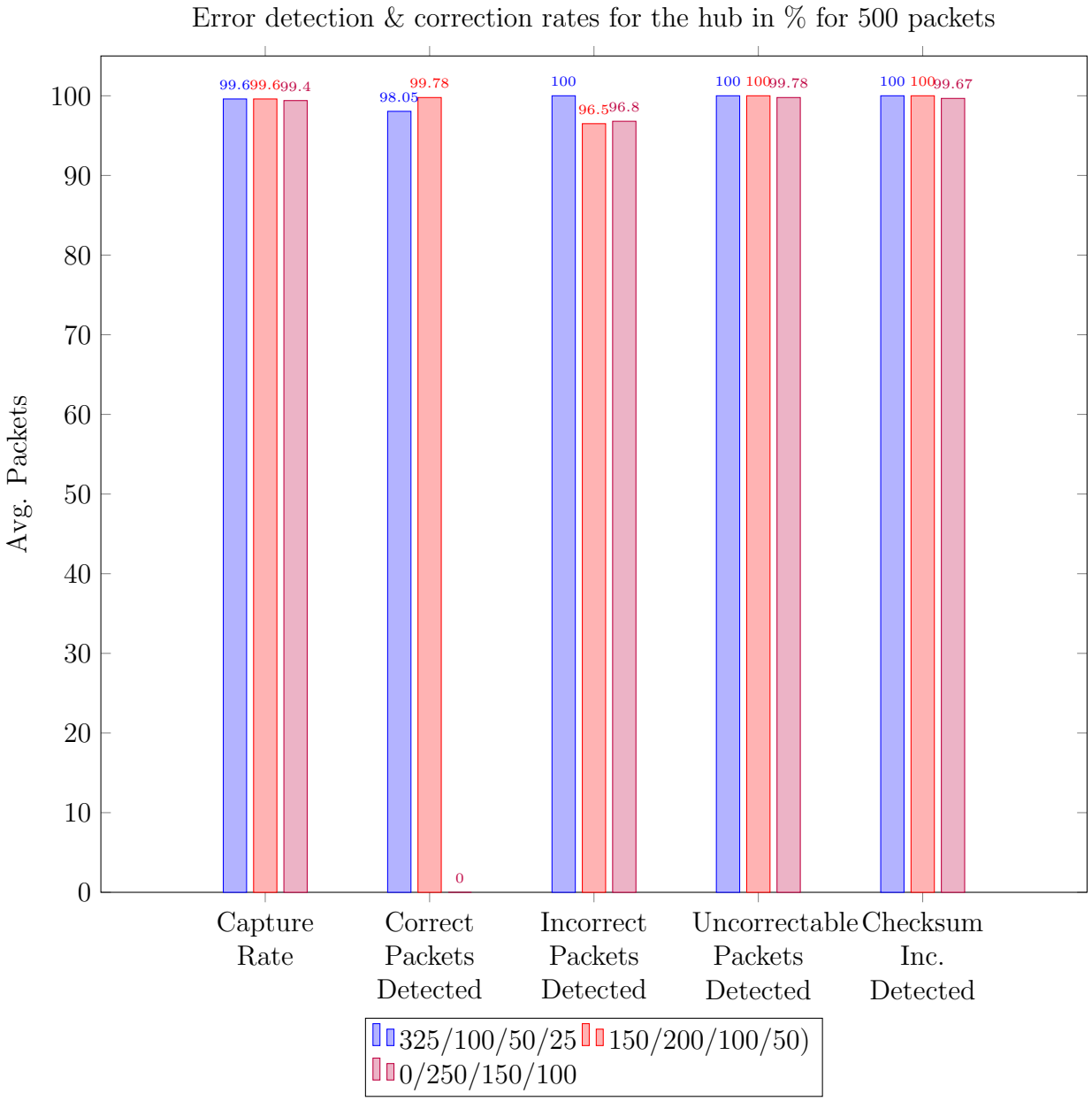
Each graph shows the results generated from the test outlined in the previous section.
Each series label represents the number of each type of packet used in that test, it is formatted as follows:

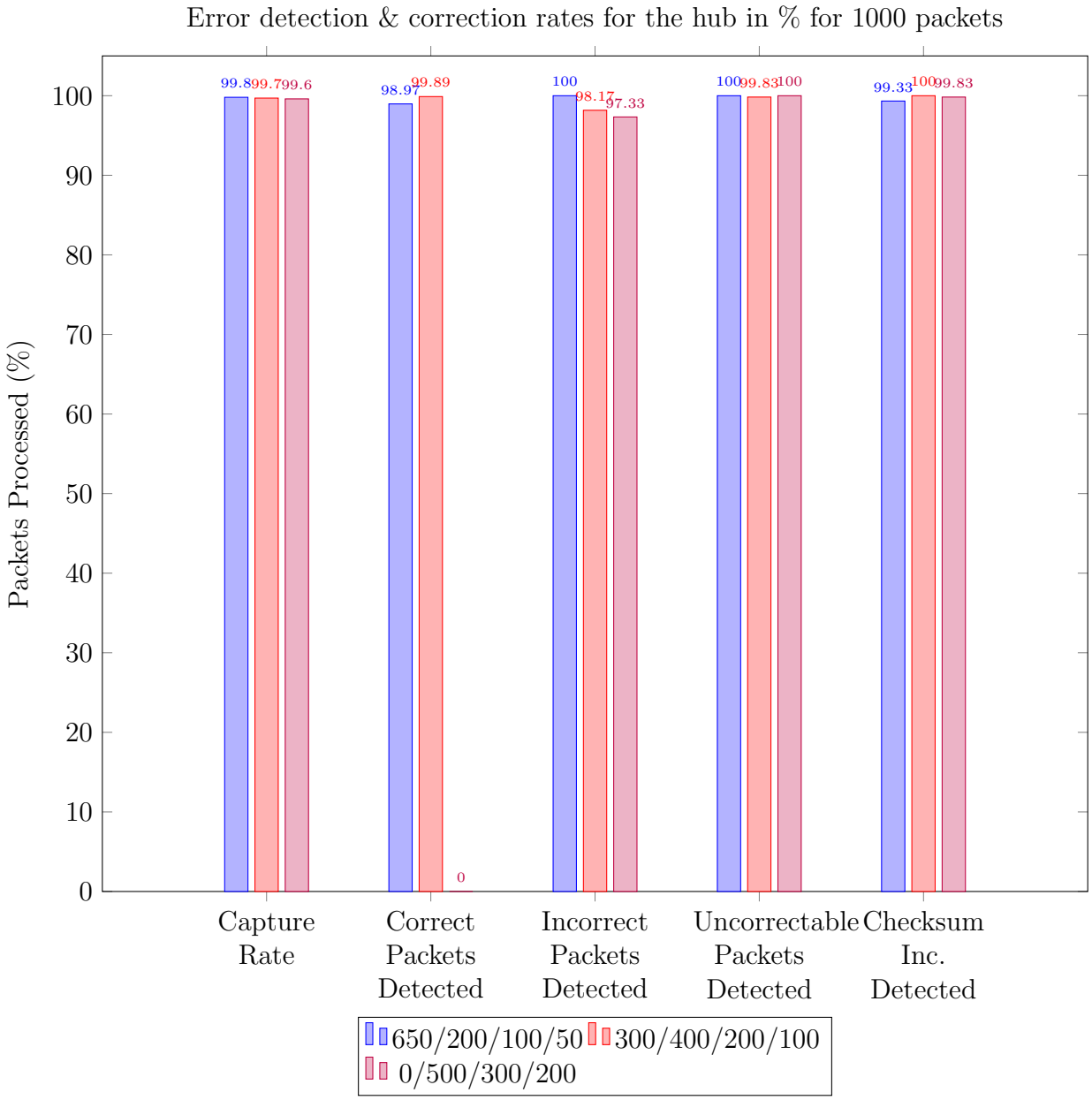
Correct packets / Incorrect packets / Uncorrectable packets / Incorrect checksum packets

For each graph the following terminology is used:

- Capture Rate - This term is used to identify the percentage of packets that were captured out of the specified total.
- Correct Packet - An correct packet means that no corrections were made to the packet during the Decoding stage.
- Incorrect Packet - An incorrect packet means that a 1 bit error was identified and corrected by the Hub.
- Uncorrectable Packet - An uncorrectable packet is a packet that contained 2 or more bit errors, couldn't be corrected, and was subsequently discarded.
- Checksum Incorrect Packet - This term describes a packet where the checksum is incorrect and had to be discarded.


Error detection & correction rates for the hub in % for 100 packets
Avg. Packets
0
10
20
30
40
50
60
70
80
90
100
100
99
100
100
99.89
0
100
95
98
100
100
100
100
96.67
100
Capture Rate
Correct Packets Detected
Incorrect Packets Detected
Uncorrectable Packets Detected
Checksum Inc. Detected
65/20/10/5
30/40/20/10
0/50/30/20


Error detection & correction rates for the hub in % for 500 packets
Avg. Packets
0
10
20
30
40
50
60
70
80
90
100
99.6
99.6
99.4
98.05
99.78
0
100
96.5
96.8
100
100
99.78
100
100
99.67
Capture Rate
Correct Packets Detected
Incorrect Packets Detected
Uncorrectable Packets Detected
Checksum Inc. Detected
325/100/50/25
150/200/100/50)
0/250/150/100



The graphs show that the overall error correction & error detection rates for the hub are almost perfect.
In a few of the tests packets were discarded by the Hub. When a packet is discarded on the hub, there is a cool off period before resuming listening for the beginning of the next packet to the value of 1.5 milliseconds. Therefore, if one packet is lost, at the rate the 'Appliance Sensor' Pi was transmitting, 2 packets would be lost which is why there is a mismatch between the number of sent packets and received packets.

4.2.3. Conclusions

Overall, the evaluation of the error correction and detection capabilities of the Hub has proved to be very successful. The Hub appears to be pretty impervious to errors, and it detected 100% of each type of packet that it received, if the packet made it through the capture process.

Once point to note however for possible future improvement, is the cool off period between discarding a packet and resuming the next packet. Some fine tuning on this aspect could improve the capture rate of the hub.

4.3. App

4.3.1. Test Cases

The following tables are the test cases of the application for the project. Each test has a purpose, a distinct number of steps and a clear result.

Case:	**Register Functionality**		
Purpose	To determine whether the in-app registration feature works correctly		
Criteria	**Test Step**		**Expected Results**
	No.	Description	Description
	1	The user enters their name, their email address and their desired password and the user taps the Register button.	The details are sent to the server, and returned to the application, the user is then subsequently logged in providing details are correct.
Actual Result	**SUCCESS** - The new user was automatically logged in and redirected to the home screen.		

Case:	**Login Functionality**		
Purpose	To determine whether an existing using can log in to the application		
Criteria	**Test Step**		**Expected Results**
	No.	Description	Description
	1	The user enters their user name and password and taps the Login button.	The user should be redirected to the home screen upon a successful login with correct credentials.
Actual Result	**SUCCESS** - The existing user was logged in, and redirected to the home screen		

Case:	**Updating An Appliance Details Functionality**		
Purpose	To determine whether a user can edit an appliance, and update the details stored on the server.		
Criteria	**Test Step**		**Expected Results**
	No.	Description	Description
	1	The user taps on the desired appliances' edit button, amends the appliances' details as required and taps the Update button.	The updated appliance details are sent to the server, and synced back with the client, the modal then closes closes.
Actual Result	**SUCCESS** - The user was able to adjust the appliance details, and those changes were synced with the server.		

Case:	**Appliance Image Upload Functionality**		
Purpose	To determine whether an existing user can update an appliance with a new profile picture.		
Criteria	**Test Step**		**Expected Results**
	No.	Description	Description
	1	The user taps on the desired appliances edit button, taps the 'Upload new photo' button, takes a picture and hits ok.	The new image is then uploaded to the server, and is synced with the client.
Actual Result	**FAIL** - Using a large phone (iPhone 6+) to take the photo caused the application to crash. This was tested again on the iPhone 5, and it uploaded without a hitch.		

Case:	**Delete an appliance Functionality**		
Purpose	To determine whether an existing user can remove an appliance from their accout.		
Criteria	**Test Step**		**Expected Results**
	No.	Description	Description
	1	The user swipes left on the appliance they would like to delete and taps the delete button.	The appliance is removed from the list of appliances and isn't shown again until the appliance sensor next reports.
Actual Result	**SUCCESS** - The appliance was removed from the account, and then at the next time the appliance sensor reported data, the appliance was added back to the list.		

Case:	**Live Data Functionality**		
Purpose	To determine if a user can view live data reported by an Appliance Sensor		
Criteria	**Test Step**		**Expected Results**
	No.	Description	Description
	1	The user taps on an appliance and is presented with the live view.	The user is able to see live data being reported by the appliance sensor.
Actual Result	**SUCCESS** - The user could see live data being reported by the appliance sensor		

Case:	**Add a Hub Functionality**		
Purpose	To determine if a user can link their account with a hub		
Criteria	**Test Step**		**Expected Results**
	No.	Description	Description
	1	The user taps the plus button and types in the Hub ID.	The user is successfully able to link their hub and see data being reported by the appliance sensors connected to that hub.
Actual Result	**SUCCESS** - After the user entered the new hub ID and linked it with their account, the appliances screen was updated, and allowed the viewing of the additional hub.		

Case:	**Remove a Hub Functionality**		
Purpose	To determine if a user can unlink their user account with a hub.		
Criteria	**Test Step**		**Expected Results**
	No.	Description	Description
	1	The user swipes left on the desired hub, and taps the delete button	The hub is unlinked from the user account.
Actual Result	**SUCCESS** - After the user tapped the delete button, the hub was then unlinked from the user account, and no appliances were displayed.		

Case:	**Carbon Statistics Functionality**		
Purpose	To determine if a user can view the current Carbon statistics for the UK.		
Criteria	**Test Step**		**Expected Results**
	No.	Description	Description
	1	The user taps on the Carbon statistics tab	The Carbon statistics should be displayed.
Actual Result	**SUCCESS** - After the user tapped the Carbon statistics tab, the statistics were displayed.		

Case:	**Power Statistics Functionality**		
Purpose	To determine if a user can view the current Power statistics for the UK.		
Criteria	**Test Step**		**Expected Results**
	No.	Description	Description
	1	The user taps on the Power statistics tab	The Power statistics should be displayed.
Actual Result	**SUCCESS** - After the user tapped the Power statistics tab, the statistics were displayed.		

Case:	**Power Statistics Functionality**		
Purpose	To determine if a user can view the current Price statistics for the UK.		
Criteria	**Test Step**		**Expected Results**
	No.	Description	Description
	1	The user taps on the Price statistics tab	The Price statistics should be displayed.
Actual Result	**SUCCESS** - After the user tapped the Price statistics tab, the statistics were displayed.		

Case:	**Edit Profile Functionality**		
Purpose	To determine if a user view and edit their profile.		
Criteria	**Test Step**		**Expected Results**
	No.	Description	Description
	1	The user taps on profile item, changes some information, and taps the update button.	The Users' profile information should be synced with the server, and then updated on the client.
Actual Result	**SUCCESS** - After the user tapped update, the user details were synced with the server and updated client side.		

4.3.2. Proposed Usability Study

Given more time, a full Usability study would be conducted with the aim of evaluating the usefulness of the application, and a study into how it affects user behaviour.
At the end of the usability study, each participant should answer the following questions:

- On a scale of 1-10, how easy is it to login or register using the application?
- On a scale of 1-10, how easy is the application to Navigate?
- On a scale of 1-10, how useful was the Home screen for the application?
- On a scale of 1-10, how useful was the live data offered by the application?
- On a scale of 1-10, how insightful were the live statistics offered by the application?
- On a scale of 1-10, how would you rate the applications usefulness?
- Do you feel that the application helped to lower the energy expenditure of your household? If so, how?
- Do you feel that the application affected the times at which you used your big appliances? If not, why?
- Any additional comments?

The results obtained from the usability study, would help narrow the scope of future work.

4.3.3. Conclusions

The majority of test cases resulted in successes, the one exception being a fault with the PhoneGap Cordova Camera plugin, when testing Image upload. The iPhone 6+ is a relatively new device, with a high number of pixels per inch, meaning that the image produced is too big for the Camera plugin to handle, causing it to crash.

A possible fix for this is to reduce the resolution of the image, or decrease the image size specifically for the iPhone 6+.

Conducting the proposed Usability Study would provide a number of insights into the project, that could help direct the future work for this project.

4.4. Server

4.4.1. Test Outline

A test was devised to perform various stress tests and see how the server reacts under various loads.
The server specification is as follows: a 2.3Ghz CPU and 4 gigabytes of RAM.
The stress tests were conducted using a piece of software called JMeter [1]. This piece of software runs locally on a personal machine, and imitates a user specified number of clients. JMeter can generate a number of statistics and graphs, the main statistic this test was interested in was Concurrent Users Vs. Response Time.
To test the server, a single URL was called 100 times by X number of concurrent users, with a ramp up period of 10 seconds.
The maximum number of users that JMeter can support is 2200, therefore there were three test steps: 100, 1000 and 2000.
The selected URL was the Login url (`scc-devine.lancs.ac.uk:8000/api/user`) which returns quite a large JSON object (316 bytes), which has to be retrieved from the database. The tests used valid credentials which meant that the server had to fetch the data on every request.

4.4.2. Results

In the following graphs, Users are referred to as Threads, because in JMeter a thread acts as a user.

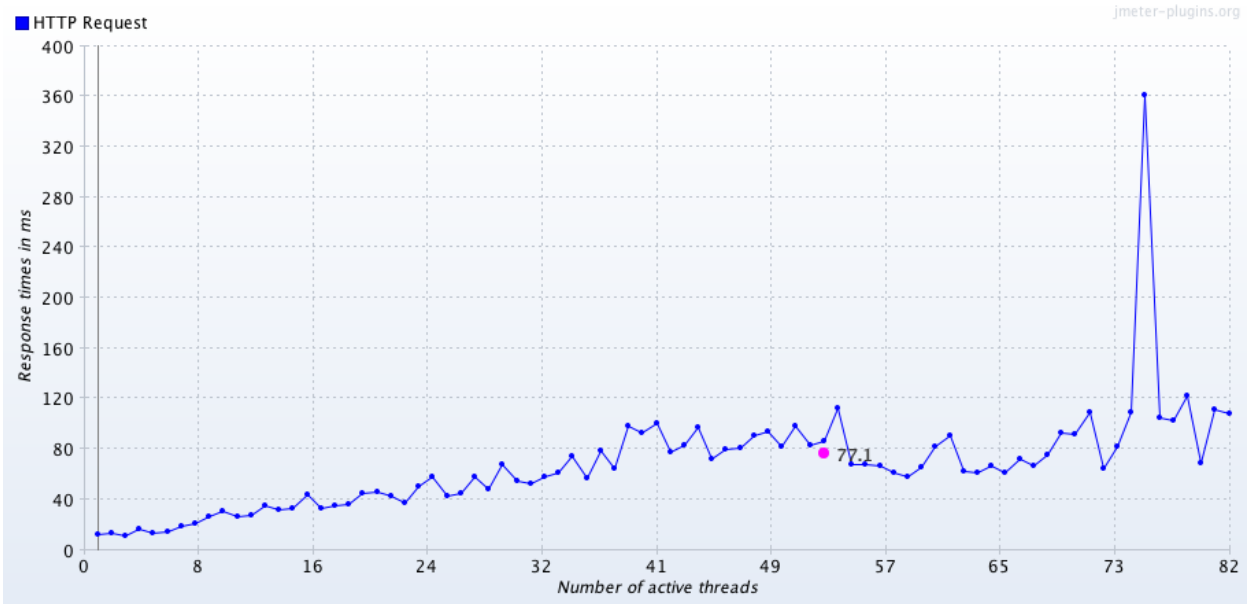


Figure 36: A stress test of 100 concurrent users performing 100 requests

Figure 36 shows the results of a test of 100 concurrent users. The test shows that the server handles 100 concurrent clients pretty well with one bad spike where a user had to wait 400

ms for a response, which immediately scales back down for the next request.

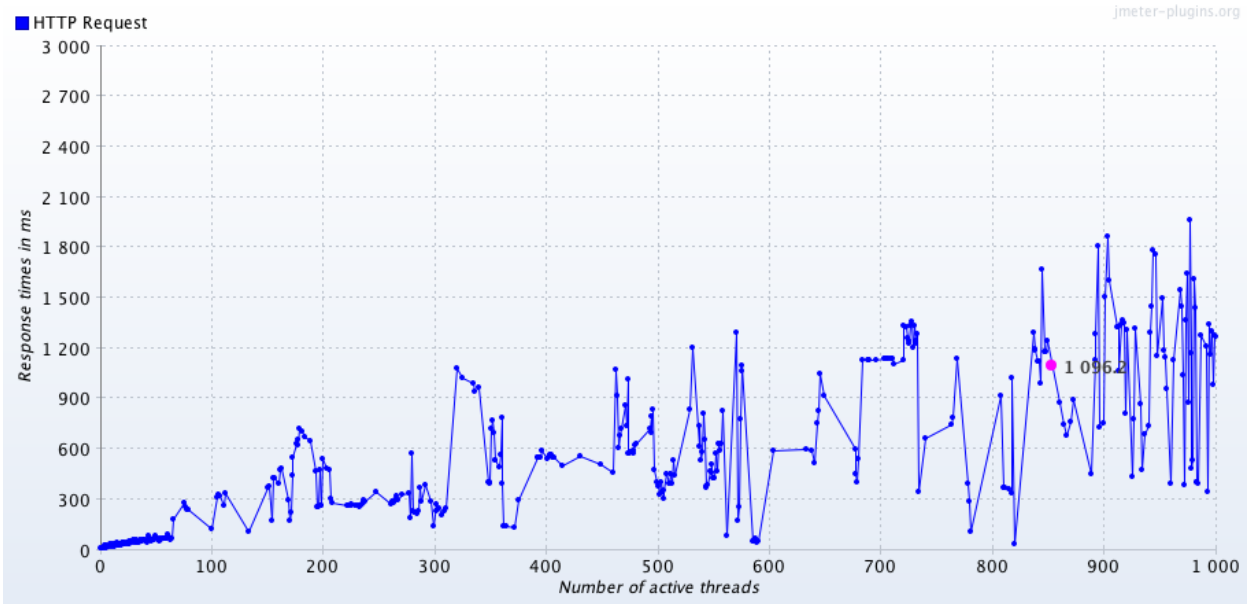


Figure 37: A stress test of 1000 concurrent users performing 100 requests

Figure 37 shows the results of a test of 1000 concurrent users. The Server starts well, with a steady increase of response time. As the number of users increases to 500 the difference in response times is huge, with a lot of random spikes. At 1000, the spikes are even worse with a peak of around 2000 ms response time.

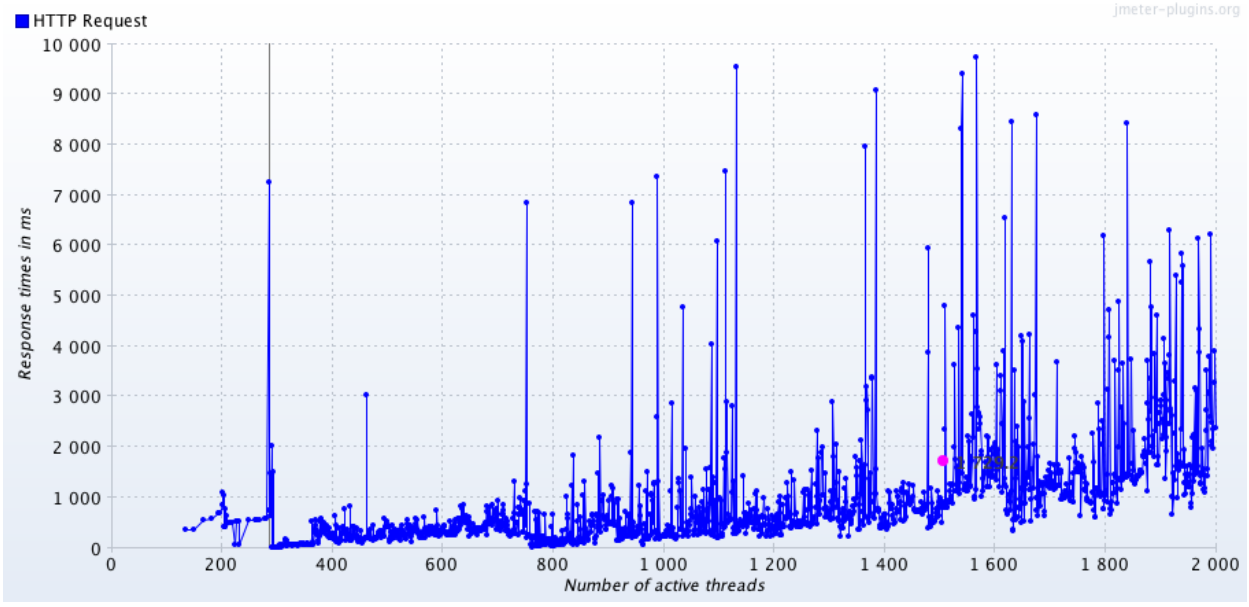


Figure 38: A stress test of 2000 concurrent users performing 100 requests

Figure 37 shows the results of a test of 2000 concurrent users. The server copes well to begin with, and at around 300 active users, there is a huge spike in response time, which immediately settles. Later, the huge variance in response time returns and becomes more sporadic the more concurrent users there are. The longest response time in this figure is around 9700 ms.

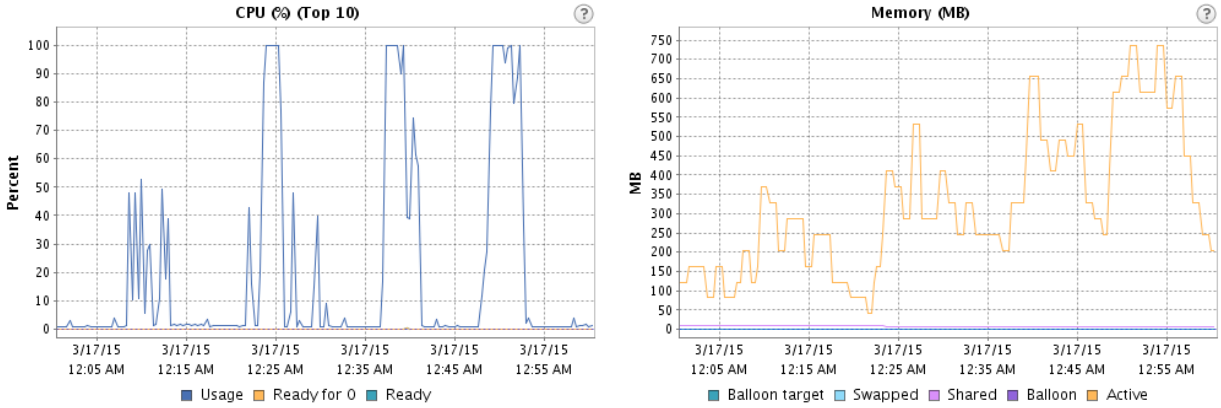


Figure 39: The performance of the server as indicated by VMWare performance tools.

Figure 39 shows the VMWare statistics. There are four clear spikes, the first being 100 concurrent users, the second 1000 concurrent users, the third 2000 concurrent users and the last one is an attempt at 10,000 users which was omitted.
Spikes can be seen in both the CPU graph and the Memory usage graph, and they clearly match onto one another. Interestingly, even though the CPU usage sky rockets, the memory usage remains pretty low in comparison.

4.4.3. Conclusions

It is clear from the stress tests that a single CPU server with a small amount of RAM is not up for the task of a hugely utilised system. However, the server-side software has been designed with this in mind, and is scalable across server clusters, allowing for a more robust, distributed system.

5. Conclusion & Future Work

5.1. Conclusion

At the beginning of the project, a number of requirements were specified:

- Small - The device needs to be small and unobtrusive and if possible, it should be hidden.
- Minimalist- The devices design should be simple and not overly complex.
- Low power - The device shouldn't consume a lot of energy, as that would defeat the aim of the device.
- Low cost - The device should be inexpensive, so that the technology is easily accessible to all.
- Simple to install and use - The device should work "out of the box" and require minimal setup and installation.
- Accessible data - The data produced should be instantly accessible through either a web interface or smartphone app.
- Real time - The data produced and displayed should be real time and up to date.
- Breaks down data per appliance - The data produced should be tied to a particular appliance so that the user can visually identify each device.
- Reliable - The device should be reliable and require minimal maintenance.

Size was a requirement, and it specified that the final system must be small. Each Appliance Sensor is small in comparison to existing products, and components can be minified further to provide a more compact final solution. Similarly, with the Hub, the Raspberry Pi has a small form factor, and is very easy to hide. The additional hardware accompanying the Hub can also be minified and integrated into the Hub, reducing the form factor further.
Minimalism was a key requirement, and the solution produced matches this requirement. The minimalist aspects in the Appliance Sensor, which has a very simple circuit: 1 PIC, 5 resistors and 2 LEDS.
Low Power was also specified as a requirement. For an installation of 1 hub and 40 Appliance Sensors, the system will consume 700 mA for the hub, and 40 x 8 µA, giving a total of 0.700 32 A, which works out at around 3.5016 W.
The total cost of a system consisting of 1 Hub and 40 Appliance Sensors is around £98. The Raspberry Pi including required accessories and additional hardware prices up at around £50 and the Appliance Sensor costs around £1.20 each, which adds the additional £48 for 40 sensors. In comparison to the existing products section, the price offered by this system is low cost for the accuracy received, adhering to the requirement.

The installation process required to setup an Appliance Sensor and a Hub is not very simple and is quite laborious in it's current implementation. Some further research will be required in order to speed up the installation process.
The data produced by the system is readily accessible, and the use of an application makes the data instantly available to users. The application is also cross platform making it accessible from a wide variety of devices. This further enforicing that the requirement of accessible data has been met.
The application supports real time data that a User can view at their disposal, which adheres to the requirement of the system being real time.
Each appliance is tied to it's own data set, and the use data ranges allows a User to navigate through the history of the desired appliance.
As proven in the Evaluation section, the produced system is quite reliable. The Appliance Sensor has an extremely high range and has so far experienced no downtime, the error correction and detection capabilities of the Hub indicate that the system is impervious to errors. The server however needs to be scaled and distributed if the system were to go into production.
The overall results obtained from this project, shows that a low cost, low infrastructure approach to household energy consumption monitoring that utilises the Earth line as a method of communication is viable.

5.2. Future Work

There are a few weak areas to the project that could be improved the first of which is the security of the platform.
Although security was not specified as a requirement, it is an important consideration in any system. To implement security in the project, all requests to the server would use the Hyper-text Transfer Protocol Secure to remove the interception and readability of transmitted packets from the Hub and the Application. As well as this method of security, a module for Node called Passport.js [4] would be used server side to implement OAuth authentication.
Another security risk is the transmission of packets down the Earth line. If someone were to plug a similar listener into the Earthed line, information could be obtained about the energy consumption of different appliance, which could be deemed sensitive information by Users of the platform.
For the application, the conduction of a User study and a Usability study would help direct improvements to the application, and analyse the overall usefulness of the application. As well as this, field checking could be added to the application to ensure that the User enters correct data each time they want to carry out an action. Additionally, the Axis labels for the graphs are non-existant in the application. This would affect user understanding of each graph, and would obviously need to be improved before the system was widely utilised.
Further work could be placed into improving the graphing capabilities, so that users can obtain key information at a glance. Push notifications could also be used to inform users that the carbon levels are high, or that an appliance is using an unusually large amount of

power.
The Appliance Sensor could be improved by adding control features from the application, in other words bi-directional communication between the hub and an Appliance Sensor. This could enable the ability to turn appliances on and off remotely adding another capability to the system.
Another feature that would improve the Appliance Sensor would be the addition of CDMA (Code Division Multiple Access) as outlined in the proposal. This would mean that collisions between packets would no longer be an issue, meaning that more samples could be transmitted more regularly.
For the Hub, as mentioned in the evaluation section, the fine tuning of the cool down period would be beneficial and would improve the capture rate.
As mentioned in the conclusion, the installation of a Hub and an Appliance Sensor is currently laborious. Further work will need to be performed in order to simplify this process and make installation simple.
Evaluating the system on a live mains line would also be beneficial to see how the system reacts and handles the noise experienced on the Earth line.
Pitching the system to Energy companies could also benefit the project massively. A system that allows comprehensive breakdowns of per appliance energy consumption is attractive to Energy companies who are actively looking to reduce the load on the grid. The platform could also provide insights into many different appliances and this information could be fed back to manufacturers as a prompt to start a conversation about increased energy conservation.

[1] Apache jmeter - apache jmeter. http://jmeter.apache.org/. (Visited on 03/17/2015).
[2] n e t a. http://www.bmreports.com/bwx_home.htm. (Visited on 03/18/2015).
[3] Nord pool spot. http://nordpoolspot.com/About-us/. (Visited on 03/18/2015).
[4] Passport - simple, unobtrusive authentication for node.js. http://passportjs.org/. (Visited on 03/17/2015).
[5] Pdc live - hamming code. http://pdc.ro.nu/hamming.html. (Visited on 03/08/2015).
[6] Slew rate image (720514). http://www.audioholics.com/audio-amplifier/amplifier-slew-rate/image. (Visited on 03/17/2015).
[7] Ying-Wen Bai and Chi-Huang Hung. Remote power on/off control and current measurement for home electric outlets based on a low-power embedded board and zigbee communication. In *Consumer Electronics, 2008. ISCE 2008. IEEE International Symposium on*, pages 1–4. IEEE, 2008.
[8] Chovy. What is express.js? http://stackoverflow.com/questions/12616153/what-is-express-js, 2012. [Online; accessed 8/02/2015].
[9] Citrusbyte. Redis. http://redis.io/, 2015. [Online; accessed 22/02/2015].
[10] Citrusbyte. Redis faq. http://redis.io/topics/faq, 2015. [Online; accessed 22/02/2015].
[11] CNET. Energy aware neurio: Home intelligence preview - cnet. http://www.cnet.com/uk/products/energy-aware-neurio-home-intelligence/. (Visited on 03/10/2015).
[12] CNET. Neurio detection rates. http://www.cnet.com/products/energy-aware-neurio-home-intelligence/, 2013. [Online; accessed 7/02/2015].
[13] P3 International Corporation. Kill a watt. http://www.p3international.com/products/p4400.html, 2014. [Online; accessed 26/10/2014].
[14] Enrico Costanza, Sarvapali D Ramchurn, and Nicholas R Jennings. Understanding domestic energy consumption through interactive visualisation: a field study. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pages 216–225. ACM, 2012.
[15] Debian. Python 3 vs php. http://benchmarksgame.alioth.debian.org/u32/benchmark.php?test=all&lang=python3&lang2=php, 2014. [Online; accessed 8/02/2015].
[16] Debian. Ruby vs php. http://benchmarksgame.alioth.debian.org/u32/compare.php?lang=yarv&lang2=php, 2014. [Online; accessed 8/02/2015].
[17] Express. Expressjs. http://expressjs.com/, 2013. [Online; accessed 8/02/2015].
[18] Apache Software Foundation. Cordova. http://cordova.apache.org/, 2013. [Online; accessed 9/02/2015].
[19] Apache Software Foundation. Couchdb. http://couchdb.apache.org/, 2015. [Online; accessed 22/02/2015].
[20] MongoDB Foundation. Mongodb. http://www.mongodb.org/, 2015. [Online; accessed 22/02/2015].
[21] Raspberry Pi Foundation. Raspberry pi. http://beagleboard.org/bone, 2014. [Online; accessed 24/02/2015].
[22] Raspberry Pi Foundation. Raspberry pi. http://www.raspberrypi.org, 2015. [Online; accessed 24/02/2015].
[23] Thomas Hunter. Php and apache stack vs node.js (development). https://thomashunter.name/blog/php-vs-nodejs/, 2012. [Online; accessed 8/02/2015].
[24] Energy Aware Technology Inc. Neurio. http://neur.io/, 2013. [Online; accessed 26/10/2014].
[25] Famous Industries. Famo.us. http://famo.us/, 2013. [Online; accessed 9/02/2015].
[26] Jskv. Php and apache stack vs node.js (performance). https://code.google.com/p/node-js-vs-apache-php-benchmark/wiki/Tests, 2012. [Online; accessed 8/02/2015].
[27] Kristof Kovacs. Cassandra vs mongodb vs couchdb vs redis. http://kkovacs.eu/cassandra-vs-mongodb-vs-couchdb-vs-redis, 2015. [Online; accessed 22/02/2015].
[28] Christopher Laughman, Kwangduk Lee, Robert Cox, Steven Shaw, Steven Leeb, Les Norford, and Peter Armstrong. Power signature analysis. *Power and Energy Magazine, IEEE*, 1(2):56–63, 2003.
[29] Jian Liang, Simon KK Ng, Gail Kendall, and John WM Cheng. Load signature studypart ii: Disaggregation framework, simulation, and applications. *Power Delivery, IEEE Transactions on*, 25(2):561–569, 2010.

[30] Chia-Hung Lien, Hsien-Chung Chen, Ying-Wen Bai, and Ming-Bo Lin. Power monitoring and control for electric home appliances based on power line communication. In *Instrumentation and Measurement Technology Conference Proceedings, 2008. IMTC 2008. IEEE*, pages 2179–2184. IEEE, 2008.
[31] Chia-Hung Lien, Chi-Hsiung Lin, Ying-Wen Bai, Ming-Fong Liu, and Ming-Bo Lin. Remotely controllable outlet system for home power management. In *Consumer Electronics, 2006. ISCE'06. 2006 IEEE Tenth International Symposium on*, pages 1–6. IEEE, 2006.
[32] Forbes Lindesay. Jade - template engine. `http://jade-lang.com/`, 2015. (Visited on 03/06/2015).
[33] Mean. Mean. `http://mean.io/#!/`, 2015. [Online; accessed 8/02/2015].
[34] Microchip. Pic16f1788. `http://www.microchip.com/wwwproducts/Devices.aspx?product=PIC16F1788`, 2013. [Online; accessed 24/02/2015].
[35] Microchip. Pic10f322. `http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en552977`, 2014. [Online; accessed 24/02/2015].
[36] Jakob Nielsen. Ten usability heuristics. 2005.
[37] Mark Otto. Bootstrap the world's most popular mobile-first and responsive front-end framework. `http://getbootstrap.com/`, 2015. (Visited on 03/06/2015).
[38] PhoneGap. Phonegap. `http://phonegap.com/about/`, 2015. [Online; accessed 9/02/2015].
[39] Blaine A Price, Janet van der Linden, Jacky Bourgeois, and Gerd Kortuem. When looking out of the window is not enough: informing the design of in-home technologies for domestic energy microgeneration. *Proc. ICT4S*, pages 73–80, 2013.
[40] Solid Run. Hummingboard. `http://www.solid-run.com/products/hummingboard/`, 2014. [Online; accessed 24/02/2015].
[41] Connor Sears. Ratchet. `http://goratchet.com/`, 2014. [Online; accessed 9/02/2015].
[42] Einar Otto Stangvik. Socket.io. `http://socket.io/`, 2015. (Visited on 03/06/2015).
[43] Ionic Team. Ionic. `http://ionicframework.com/`, 2014. [Online; accessed 9/02/2015].
[44] Sticknfind Technologies. Meter plug. `http://meterplug.com/`, 2013. [Online; accessed 25/10/2014].
[45] Jacopo Torriti, Mohamed G Hassan, and Matthew Leach. Demand response experience in europe: Policies, programmes and implementation. *Energy*, 35(4):1575–1583, 2010.
[46] w3.org. Status code definitions. `http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html`. (Visited on 03/06/2015).
[47] Wikipedia. 2000px-mvc-process.svg.png (20002200). `http://upload.wikimedia.org/wikipedia/commons/thumb/a/a0/MVC-Process.svg/2000px-MVC-Process.svg.png`, 2015. (Visited on 03/07/2015).
[48] Wikipedia. Django (web framework). `http://en.wikipedia.org/wiki/Django_%28web_framework%29`, 2015. [Online; accessed 8/02/2015].
[49] Wikipedia. Modelviewcontroller. `http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller`, 2015. (Visited on 03/06/2015).
[50] Wikipedia. Node.js. `http://en.wikipedia.org/wiki/Node.js`, 2015. [Online; accessed 8/02/2015].
[51] Wikipedia. Ruby on rails. `http://en.wikipedia.org/wiki/Ruby_on_Rails`, 2015. [Online; accessed 8/02/2015].
[52] Wikipedia. V8 javascript engine. `http://en.wikipedia.org/wiki/V8_%28JavaScript_engine%29`, 2015. [Online; accessed 8/02/2015].

Appendices

.1. Proposal

Smart Fuse For Appliances

James Devine

Abstract

Currently, some home users have the ability to view the power consumed by their household, but nothing to inform them of the power consumed by individual devices. An innovative and creative solution is specified in this proposal, as well as a detailed approach to the resolution of this problem.

/*The introduction contains a background on the problem and some of the already existing solutions as well as what is required in order to satisfy the aims of this project. The proposed project section outlines potential solutions, which will be developed into the final system. The 'Programme of Work' section specifies the key areas of development and displays a Gantt chart to show how my time will be distributed throughout the project. In the 'Resources Required' section I explain any hardware or software I deem are essential to the project and my justifications as to why they are essential. The last section is 'References' and here I provide links to resources I have used to produce this report.*/

1. Introduction

In today's modern society it is extremely important from a social point of view to keep track of your energy consumption and the damage you are ultimately doing to the environment, it's even more important from an economical point of view to save money in today's disastrous economy.

The problem that exists today is that users are able to see the overall power consumed by their household, but not how much energy is consumed by individual devices – this can be compared to a bank account where you can't see individual transactions, it is not very useful.

However, there are products that allow the user to see individual energy consumption, but their limitation is size, and how obtrusive they are into the users' environment. They are also limited in the way they communicate, some devices require manual readings, and others still produce a single figure for the entire household, even though they have readings for each individual device.

To the right, is an existing product that allows individual appliances to be monitored through a device that the user plugs the appliance into, and details are displayed above the socket on a LCD. This approach is limited as you have to manually read figures off of the socket display and you have to buy a socket per appliance – or group appliances together using an adapter.

Another product measures the overall output of a household by attaching a sensor to the mains cable leading into the house. The limitations of this approach are clear. You can only see the consumption of energy as a whole, not individually. However, these devices do allow an accompanying web app and LCD to visualize the energy consumption, which the previous solution lacked.

Finally, the last product has a number of plugs which you can plug appliances into, and they are then fed back to a central hub using the X10 power line protocol. The limitations of this approach are the cost per plug and the central hub price. Also the plugs are very big which is obtrusive in any environment, which is problematic for your average homeowner.

From the three existing products above, there is nothing that is neat, inexpensive and allows the monitoring of individual devices.

The device that should exist, is one that is both discrete in it's implementation, and allows readings per device, with data transmitted to a central storage device, that allows the user to visually breakdown power consumption per device.

2. Generating a System from High Level Requirements

There are a few high level requirements that can be defined at this early stage:

- The device should be able to communicate the energy consumed by an appliance to a central hub.
- The Central Hub should display the data held about each fuse in a graphical format
- The device should communicate readings at a regular interval.
- The device should be unobtrusive in its environment

The first step is to determine where this device is to be embedded. In each electronic appliance, there is only one shared entity, the fuse. Every single appliance has a fuse that breaks when the appliance's circuitry draws too much power. This is an ideal place to insert an embedded system as from this location you can see how much energy the appliance is currently drawing. From now on, this embedded system will be referred to as the Fuse.

The next step is to determine how the Fuse will communicate with the central hub. Wifi and Bluetooth seem like the obvious choices, however, they add bulk to the device, which would reject the requirement for the device to be unobtrusive. Also, Wifi and Bluetooth can be heavily distorted in uncontrolled environments, not unlike the environments in which this device will be deployed, implying that these means of communication aren't suited to this project.

One means of communication that is still heavily utilized today is power line communication. This allows devices to use the infrastructure provided by the

power lines of a building as a method of communication. There are existing communication technologies such as X10 and PLC (Power Line Communication) which have defined protocols to allow communication over the power line. Despite having defined protocols, the hardware that is introduced with these technologies is quite bulky as you can see in the third example of existing products. Using these technologies would go against a high level requirement, and would make the end device obtrusive in its environment.

This implies that an existing technology/protocol isn't appropriate for this project, and a new protocol will have to be created that is appropriate for power line communication.

The last a final step concerns the notion of a centralized storage point for all of the data to be sent to, the idea of a Hub. The central hub needs to have some sort of communication with the power line in order to receive information transmitted by the fuse, as well as this the device needs to be capable of storing the received information in a database, that can then subsequently be displayed on a webpage.

3. Derivation of Functional Requirements

From the high level requirements, the components have been generated that will form the end product of this research. Below is a further breakdown of each individual component, and the functional requirements attributed to that component.

3.1 Functional Requirements - The Fuse

- The Fuse should be universally compatible.
- The Fuse should be small enough to fit into a plug.
- The Fuse should be energy efficient and not consume a large amount of electricity.
- The Fuse should be cost effective.
- The Fuse should be scalable to all types of environments.
- The Fuse should sample the energy consumption at a regular interval, and then report the average reading back to the Central Hub.
- The Fuse should be bound to a Central Hub to prevent others from hijacking data.
- Each Fuse should emit a regular pulse that uniquely identifies itself.
- The Fuse should be able to communicate at the same time as other fuses on the power line, collisions should be resolved.
- The Fuse should be able to function as a normal Fuse, and cut off when the circuit draws too much current.
- The Fuse should be easy to install, just like changing a normal fuse.
- The Fuse should have a high tolerance to variability of readings from the appliance.

3.2 Functional Requirements – The Central Hub

- The Central Hub should always be checking for a reading from the power line.
- The Central Hub should be portable and easy to install.
- The Central Hub should store received data in a database.
- The Central Hub should present data in an easy to understand format for the user via a web interface.
- The Central Hub web interface should only be viewable by those authorised to view it.
- The Central Hub should be able to detect and differentiate between each fuse.
- The Central Hub should be cheap and inexpensive.
- The Central Hub should have the ability to be accessed over the internet or through a smartphone application.
- The Central Hub should be able to communicate with other hubs if required, to produce a scalable solution.
- The Central Hub should be device independent.

3.3 Functional Requirements – Communication Protocol

- The Protocol used should be simple and require a small amount of energy.
- The Protocol should automatically detect collisions as they occur and resolve them.
- The Protocol should be able to support a large amount of fuses.
- The Protocol should be able to adapt to large buildings.
- The Protocol should have a distinct format that specifies the packet structure.
- The Protocol should have a simple checksum feature that allows the central hub to determine if all data has been received correctly.
- The Protocol should not exceed the space limitations imposed by the fuse.

4. The Proposed Project

The project is an experiment to test whether an embedded system would work in a place such as a plug and whether power line communication is a viable choice; the end goal is not to have a finished product, but a set of results, which can be used to develop the end product further. The three main areas of focus for this project are to design a fuse that communicates over power line, to gather data sent over the power line using a new protocol created through the conception of this project and to receive those results on a central hub. The final test would be to implement the system in a real environment to see if it

promotes energy awareness among the test subjects, and if it is a useful tool to help reduce bill sizes.

4.1 The Fuse

The design of the Fuse should align to the requirements derived earlier in the proposal.

From previous discussions in this proposal, it is implied that the Fuse will consist of a microcontroller and a simple fuse wire that will overheat if too much current is drawn by the appliance. The microcontroller contains software that will drive the sampling of the power usage of the appliance, and the transmission of data over the power line using a brand new protocol.

When developing the fuse, safety is of paramount importance, which means development will need to occur using a low voltage testing harness, ideally 16v.

An ideal sampling rate needs to be determined during development, so that accurate readings are transmitted to the central hub. Also a tolerance will need to be determined during development, so that an offset can be calculated to normalize outliers when sampling the appliance voltage.

Then the issue arises of how to actually sample the appliance. There are two key types of current, Alternating Current (AC) and Direct Current (DC). DC is the ideal situation and requires a simple equation called Ohms Law:

$P=I^2\times R$ $\quad$ *where* P *is the Wattage,* I *is the current and* R *is the resistance.*

This would be complex to implement, however, AC is far more complex:

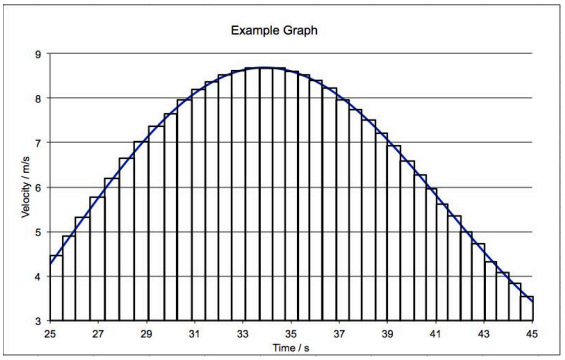
$$P_{avg} = V\times I\times \cos \emptyset$$

The formula above requires the constant sampling of a complete sine wave produced in the AC circuit, once a complete cycle has been sampled, the power can be calculated. Both types of current have their own complexities, but to begin with, embarking on the DC type would be ideal, as it is simpler than it's AC equivalent. Once the DC implementation is complete, development of the AC approach will begin.

One important aspect of the Fuse that can't be overlooked is its ability to detect which type of current is flowing and perform subsequent calculations appropriately.

Another element that needs to be investigated is how an AC wave is sampled.

The image to the right shows an example method of determining the area under a curve. Integration is used at various sample points across the waveform, from the values returned by the integral, you can then calculate an approximation of the area underneath the curve. Of course, an AC waveform alternates



between positive and negative values, an absolute value would have to be returned for those points sampled in the negative region of the waveform.

4.2 Communication

As determined earlier in this proposal, there are no suitable communication protocols for this project, so there is a need to invent a new protocol.

The protocol will need to align with the requirements specified earlier in the document.

During the development phase, the best approach would be to transmit pulses at a regular interval, and use a wave oscillator to detect differences in the waveform to see if the device transmissions are detectable.

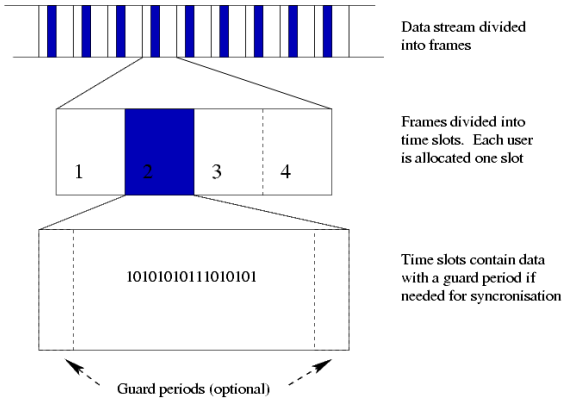
Each fuse will need to be able to be uniquely identified; this can be done at manufacture by assigning each fuse a unique id a similar concept to MAC addressing.

Existing systems work by modulating a carrier signal onto the power line. The power line typically transmits at frequencies between 50Hz and 60Hz, the devices built for detecting transmissions on the power line are aware of this fact and look towards detecting transmissions at much higher frequencies between 100Hz and 200Hz.

It is entirely plausible for a microcontroller to modulate frequencies up to this range; the next task will be to determine how to handle collisions on the power line.

The two methods of collision detection and resolution are Time Division Multiple Access (TDMA), and Code Division Multiple Access (CDMA).

TDMA divides transmissions over time using slots defined slots assigned by a host. The Fuse system will have unidirectional communications to begin with; so assigning slots to each Fuse is not possible. A possible solution to the problem is to introduce a back off, which prevents each fuse from transmitting for a random amount of time if a collision is detected.



Alternatively, CDMA could be used instead. An analogy for CDMA is a room (channel) where people want to talk to each other simultaneously, if people talk in different languages (codes), you can isolate each language, and translate what they are trying to say. If each Fuse is required to speak a different language, a table of codes could be stored on the microcontroller and depending on the mac address of each fuse, the respective channel code is selected, then subsequently used in its implementation.

The CDMA version implemented would have to be synchronous which would require orthogonal coding in the implementation. Orthogonality in this case is determined by the dot product of two simultaneous signals which must resolve to 0 i.e. they do not interfere with each other. *Maximum number defined by welsh code is 64 orthogonal signals…?*

Packet Definition:

0	1	2								3															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Type	Status	Checksum								Payload															

The entire packet uses 26 bits

Type - used to indicate what content is contained in the payload whether the reading is an Average (AC) or an Actual reading (DC). Consumes one bit of the packet.

Status – used to indicate if the appliance is currently on or off. 1 for on, 0 for off. Allows the immediate evaluation of a packet, without looking at its payload. Consumes one bit of the packet.

Checksum – used to check if the packet has been corrupted during transmission. Consumes 8 bits of the packet.

Payload – the content of the transmission. Consumes 16 bits of the packet.

4.3 The Central Hub

As discussed in the requirements section, the central hub will require a database to store data received over the power line. In terms of databases there are 2 distinct solutions that present themselves, MySQL and SQLite. MySQL is suitable for large scale and production developments, which is a requirement not matched by the proposed system. On the other hand SQLite is great for testing, prototyping and embedded systems, it's lightweight, simple to deploy, and requires little or no administration – an ideal solution.

The functional requirements state that the central hub to be cheap and inexpensive. For the required level of functionality, a simple inexpensive device would be a Raspberry Pi, which draws relatively low amounts of power, is small and unobtrusive, has the ability to communicate with other devices via mediums such as Bluetooth and Wifi and allows for a large multitude of scripting and programming languages.

Obviously the Central Hub must understand the protocol outlined in 4.2 and be able to interpret data sent over the power line.

4.4 Additional Scope For this Project

Bidirectional communications would add a number of features and benefits to the system. The first immediate benefit it would bring is the ability to

communicate with fuse, which would thereby allow some sort of appliance control, turning the appliance on and off for example.

The next benefit that this would bring is the dynamic allocation of unique addresses. This means that no two Fuses will have the same address, and the Fuse would operate in a plug and play fashion. With the present MAC addressing solution, there is still a small chance that two devices could end up on the same system, which would cause errors. Along with this dynamic allocation of addresses, channel codes for CDMA could be allocated at the same time. In essence the Fuse would communicate using TDMA initially, but would then begin to use CDMA once a unique address has been assigned and a channel code received.

One final benefit that bidirectional communication would bring is security. Security could be enforced through the binding of fuses to hubs. Each hub would have its own unique table of channel codes that it can assign, thereby only allowing messages to be understood by hubs with the channel codes assigned to them.

A major addition to the SmartFuse would be a Polymeric Positive Temperature Coefficient (PPTC) device. The PPTC device would act as a fuse, the main benefit being its reusability. PPTC devices match current with resistance as the current is raised, meaning that if a circuit began to draw too much electricity, the PPTC device would match this with resistance, preventing circuit load. Once the huge amount of current is removed, the resistance is reduced to a conductive state once more.

5. Programme of work

6. Resources Required

7. References

http://www.ethicalsuperstore.com/products/efergy/efergy-energy-monitoring-socket/

http://www.theenergydetective.com/prohome

http://www.britishgas.co.uk/products-and-services/gas-and-electricity/energysmart.html

http://masterslic.tripod.com/olcalculator.html

http://hyperphysics.phy-astr.gsu.edu/hbase/electric/powerac.html

http://en.wikipedia.org/wiki/Power-line_communication

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

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

http://www.l4labs.soton.ac.uk/tutorials/excel/images/curve12.jpg

http://stackoverflow.com/questions/3630/sqlite-vs-mysql

http://www.sqlite.org/whentouse.html

.2. Original Gantt Chart

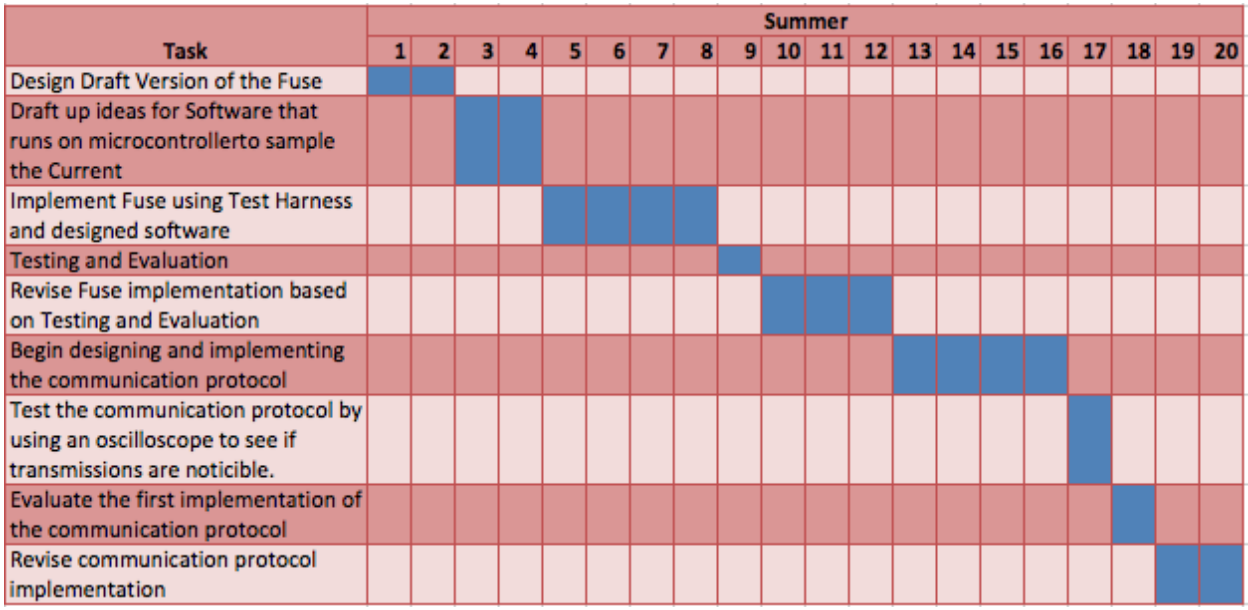


Figure 40: The Gantt Chart for the Summer break

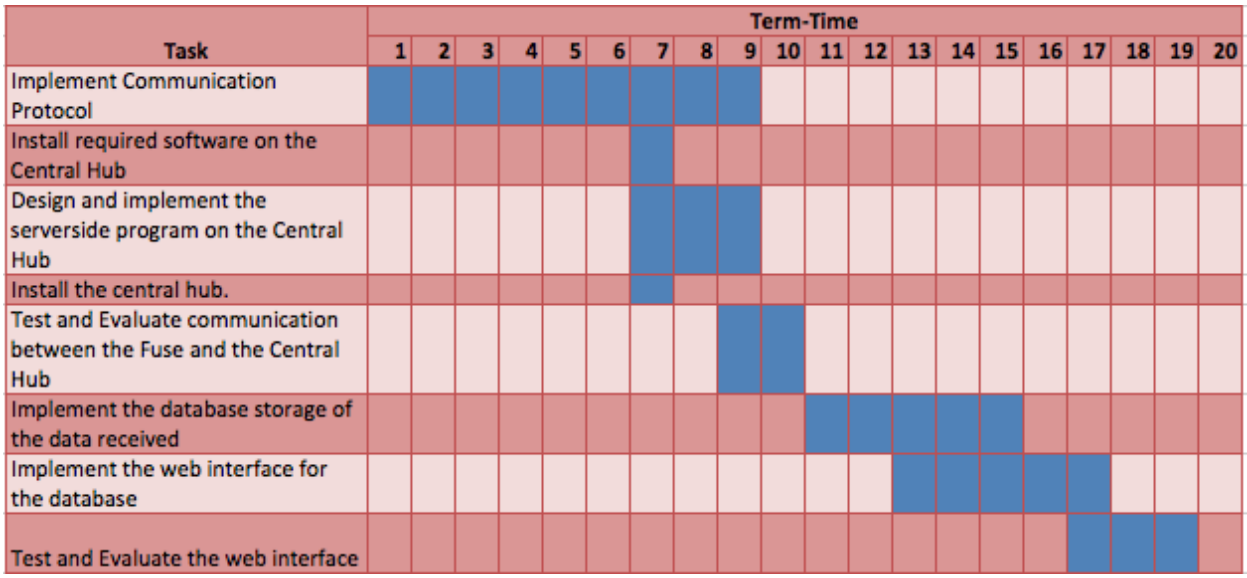


Figure 41: The Gantt Chart for Term Time

.3. Deviations from the Proposal

There were a number of deviations from the original proposal:

- Operation as a Fuse - The proposal indicated that the Project would act as a replacement for the traditional fuse. Growing requirements and hardware restrictions meant that the project shifted away from this idea.
- CDMA - Code Division Multiple Access was suggested as a means for communication and collision detection and resolution. The scope for this idea was shifted due to time constraints of the project and was replaced by Extended Hamming Code as a means for collision detection and resolution.
- Earth Line as opposed to power line - The original proposal indicated that the Power line would be used. When conducting research, the realisation was made that existing solutions had heavily investigated the use of the Power line. Instead, the Earth line was proposed as the communication medium.
- Application as well as Web Interface - The proposal indicated that purely a web interface would be used to display the data. Instead this shifted over the course of the project, and an application was created in conjunction with a small Web Interface, which can be seen as an addition.
- Intercommunication of hubs - The proposal suggested that the intercommunication of hubs may be required. The current implementation means that only one Hub would be required per installation, and so the intercommunication of hubs was no longer a requirement.
- Ease of installation - The project set a requirement that the system should be easy to install. Currently the system is quite difficult to install due to it being an initial prototype.

The above amendments were approved by Joe Finney, supervisor for this project.

.3.1. Ammended Gantt Chart

Submitted alongside the Proposal was the original Gantt chart which can be seen in Figures 40 & 41.

Over the course of the project, the original Gantt Chart changed dramatically, the actual Gantt charts can be seen in the following two figures:

Task	Summer																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Design Draft Version of the Fuse																				
Draft up ideas for Software that runs on microcontroller to sample the Current of an appliance																				

Task	Term-Time																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Implement Fuse using Test Harness and designed software																				
Testing and Evaluation																				
Revise Fuse implementation based on Testing and Evaluation																				
Begin designing and implementing the communication protocol																				
Test the communication protocol by using an oscilloscope to see if transmissions are noticible.																				
Evaluate the first implementation of the communication protocol																				
Revise communication protocol implementation																				
Implement Communication Protocol																				
Install required software on the Central Hub																				
Design and implement the software that runs on the central hub																				
Install the central hub.																				
Test and Evaluate communication between the Fuse and the Central Hub																				
Implement the database storage of the data received																				
Implement the web api for the database																				
Test and Evaluate the web api																				
Begin designing the application that integrates with the created API																				
Implement the application																				
Test and evaluate the entire platform																				
Project Writeup																				