SOCKETS & SENSORS - TECHNICAL REPORT

National College of Ireland
Higher Diploma in Science in Computing
Internet of Things Online Stream Sep 2017

Name:	Richard Seaman
Student Number:	17119111
Email	x17119111@student.ncirl.ie
Supervisor	Josephine Andrews

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF TABLES

GLOSSARY OF TERMS

Term	Definition
IoT	Internet of Things
RPi	Raspberry Pi
HAT	Hardware Attached on Top
RF	Radio Frequency
SSH	Secure Shell
IEEE	The Institute of Electrical and Electronics Engineers
iOS	A mobile operating system created and developed by Apple Inc.
JSON	JavaScript Object Notation
Epoch Time	The Unix epoch is the number of seconds that have elapsed since January 1, 1970 (midnight UTC/GMT)
XCode	An integrated development environment for macOS containing a suite of software development tools developed by Apple for developing software for macOS, iOS, watchOS, and tvOS.
XBee	A radio module provided by Digi International.
DHT Sensor	A readily available sensor which senses temperature and humidity.
I/O	Input / Output
IDE	Integrated Development Environment
TX	Transmitter
RX	Receiver
XCTU	A software package provided by DIGI for configuration of their Xbee devices.

1. EXECUTIVE SUMMARY

This report outlines the final project of the Higher Diploma in Science in Computing, Internet of Things Stream. The project is called "*Sockets & Sensors*". The project addresses two broad topics related to building services engineering; monitoring internal environmental conditions and controlling systems. These topics are addressed in the context of a single domestic home.

The project has three main objectives. The first is to wirelessly monitor environmental conditions and record them over time. The second is to facilitate the remote control of a number of electrical wall sockets. The third and final objective is to provide a user interface in order to view the monitored data and to control the sockets.

A system is designed to achieve these objectives. The system consists of five main components; a back-end service, a base station, a wireless sensor node, radio-controlled wall sockets and a user interface in the form of an iOS application. The design and implementation of each of these components, and how they operate as a whole, is discussed in detail. A broad range of hardware devices are utilised within the system including a Raspberry Pi, an Arduino, Xbee modules, electronic components (LEDs, resistors, sensors etc.) and an iOS device. Similarly, multiple programming languages are utilised including Python, C/C++ and Swift.

Individual system components and the system as a whole are tested in turn. Each of the tests are successful and the system operates seamlessly as a whole. At the time of writing this report, the system had been gathering data and controlling wall sockets for approximately one month without issue.

The project achieved the three project objectives and provided a proof of concept for both monitoring environmental conditions and controlling systems. The resulting system can easily be scaled to cater for additional use cases.

2. INTRODUCTION

This document is the technical report submitted for the final project module of the Higher Diploma in Science in Computing, Internet of Things Stream. The title of this project is "*Sockets & Sensors*".

This report begins by introducing the project including its background and aims. The various functional and non-functional requirements are then outlined. The overall design and architecture of the system used for the project is described and a detailed outline of the implementation of each of the main system components is then given. The methods used to test the system are provided before a final evaluation and conclusion is drawn.

2.1. Background

As a building services engineer, I am very interested in energy use and internal environmental conditions within buildings. When analysing these two topics, it is vital to be able to monitor and control the various systems and environments within the building.

I decided to base my project on a system which would allow me to monitor and control various things within my own home.

2.2. Scope / Aims

With the limited time available, I was eager to contain the scope to something achievable. I decided to try and include a single monitoring component and a single control component. The logic being that if I could achieve this, I could easily add onto the system in the future.

The scope of this project is to develop a system that is capable of:

1. Wirelessly monitoring an environmental condition within my house
2. Controlling a number of wall sockets within my home (on/off)
3. Providing a user interface to allow me to view the monitored data and control the sockets.

I was also keen to move away from just using a single Raspberry Pi and I particularly didn't want to use the GrovePi HAT. I wanted to explore other IoT hardware and remove the constraints of the GrovePi system.

2.3. Technologies & Resources

A range of hardware, software and network resources were required for this project, including the following:

Hardware

- Raspberry Pi: for the base station (3 Model B V1.2 was used)
- Arduino: for the sensor node (an Uno was used)
- XBee x2: for communication between base station and sensor node (Series 1 modules were used)
- 433MHz Transmitter: for 433MHz communication between base station and wall sockets
- 433MHz Receiver: for "*sniffing*" the 433MHz RF codes from the remote controls that came with the sockets.
- DHT11 Sensor: to allow the sensor node sense it's environment (temperature and humidity)
- iPhone: for the user interface
- RF Wall Sockets: for RF control (on/off) of appliances connected to the sockets.
- Electronic Components: LEDs, resistors, jumper wires, breadboard

Software

- Libraries:
    - 433Utils: for communicating over 433Mhz RF (Ninja Blocks, 2018)
    - xbee-arduino: for using the XBees in an Arduino sketch (Rapp, 2016)
    - python-xbee: for using the Xbee devices in a python program (niolabs, 2018)
    - SwiftChart: for graphing data in the iOS app (gpbl, 2018)
- XCode: for developing the iOS application
- Arduino IDE: for developing and loading the sketch for the ardunio
- XCTU: for configuring the Xbee devices and network
- Firebase: for a back-end service

Network

- WiFi: A home wifi network, with internet access, is required to access the back-end services. It's also required to allow remote access into the base station (over SSH).

- ZigBee (2.4GHz): for communication between base station and sensor node (and future sensor nodes).
- 433MHz: for control of wall sockets.

2.4. Assumptions / Constraints / Standards

There are a number of assumptions in this project:

- I will be the only user
- I will have an iOS device available to load the app onto.
- There will be no need to publish the iOS to the AppStore. Instead, it will be loaded directly onto the device using XCode.

There are a number of constraints in this project:

- The base station must be powered up and running the relevant program in order for the project features and functionality to be available.
- The base station must have internet access at all times in order to check socket statuses and upload sensor readings
- The base station and wireless sensor node must not exceed the range of their respective XBee devices. For the average home, this will not be an issue.
- There is no caching/buffering at the wireless node. If the signal is lost, or not received by the base station, the data will be lost.

The IEEE 802.15.4 standard is utilised by the XBee devices used within this project.

3. REQUIREMENTS

There have been no changes to the project requirements since the initial submission.

3.1. Functional Requirements

I will be the sole user of this system. There will be no external users.
The end user requirements of this system align with the three items identified within the project scope:

1. An ability to wirelessly sense whatever condition I like, from anywhere in my home.
2. An ability to control (on/off) a number of wall sockets within my home
3. An ability to view the monitored data and control the sockets using my iPhone, from anywhere with internet access.

3.1.1. System Requirements (Use Case Diagram)

A use case diagram for the system is provided in Figure 1. Each of the use cases represent an end goal of the system and are in the format "*Verb-Noun*". The three actors identified are; Home Owner, Database Administrator and Installer / Developer. In practice, each of these roles will be undertaken by one person (myself).

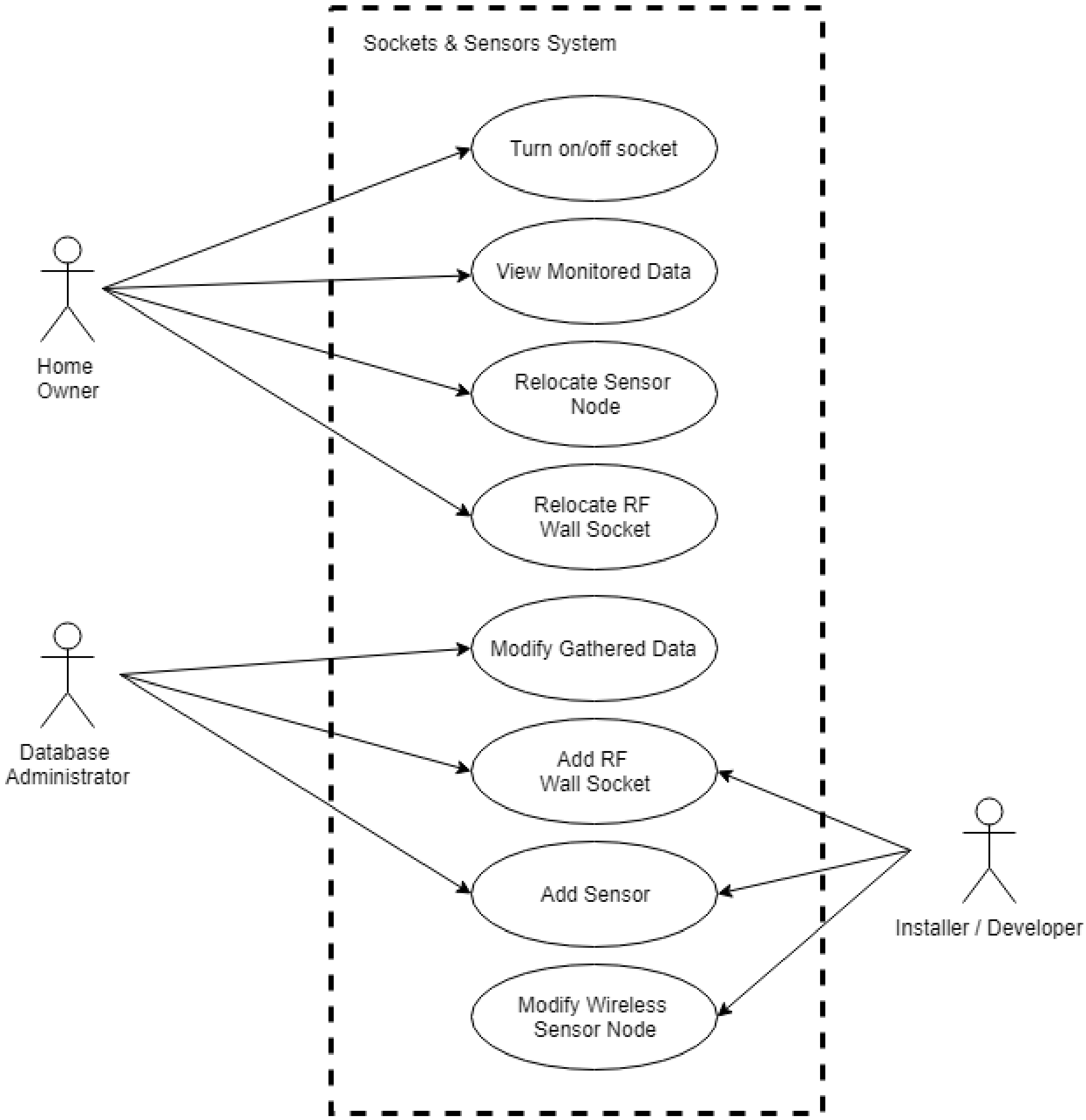

Figure 1 - Use Case Diagram

3.1.2. **Requirement – Turn On / Off Socket**

The use case “Turn on/off socket” is described in detail below.

Scope

The scope of this use case is for the home owner to turn on or off one of the RF controlled wall sockets.

Description

This use case describes the control of an RF controlled wall socket by the home owner.

Flow Description

Precondition

- The home owner has access to the iOS app and has been authenticated.
- The device with the iOS app installed on it has internet access.
- The wall sockets are plugged into various wall outlets around the house.

Activation

This use case starts when the home owner navigates to the “Sockets” tab within the iOS app.

Main flow

- The user navigates to the Sockets tab within the app.
- The RPi has recently synced with the back end (A1).
- The user decides to turn on/off a particular socket and switches the corresponding socket’s switch.
- The app updates the socket status on the back end.
- The base station detects a change on its next sync.
- The base station issues the RF signal with the corresponding code to turn on/off the socket.
- The wall socket turns on/off.

Alternate flow

- (A1 – RPi hasn’t synced recently)
- The user interface for the sockets tab is disabled, preventing the user from attempting to turn on/off a socket.

- The user is presented with an error message.
- The user is unable to turn on/off the socket.

Exceptional flow

-

Termination

The app’s user interface is updated accordingly and the system awaits further commands.

3.2. Non-Functional Requirements

The project includes a number of non-functional requirements, as detailed below.

3.2.1. Availability Requirement

There is an availability requirement for this project. The wireless sensor node and base station must be simultaneously available in order to successfully gather and record data. The base station must also be available for issuing on/off commands to the RF sockets.

If the base station or wireless sensor node fail at any time, the system will cease to function.

3.2.2. Reliability Requirement

There is a reliability requirement for this project. As with most IoT devices, they must be capable of operating for long periods of time with limited or no human interaction required.

Any hardware must therefore be durable and capable of withstanding the environments in which it will be installed. Software must be capable of handling errors while continuing to operate. For example, sensors may occasionally read strange readings which if unhandled, could cause the system to crash.

3.2.3. Security Requirement

There is a security requirement for this project. The monitored data is considered confidential. It should not be able to be viewed by anybody other than the intended user of the application.

The control of the sockets must also be limited to the intended user, otherwise anybody could switch on / off electrical items within the house.

This requirement was upheld by using Firebase Authentication. A single user is assigned permission to use this project (a specific Gmail address, signed in using Google's authentication).

3.2.4. Backup Requirement

There is a requirement to back up the data in the database. The database is the only place where the historic gathered data is recorded. If the database was to be deleted, all of the data would be lost. A more likely scenario, would be an entire node within the json tree being accidentally removed either within the Firebase console, or accidentally during the source code development.

Fortunately, Firebase automatically runs a daily back which includes real-time databases and users. In addition, the real-time database can be manually exported as a json file at any time (which was done periodically during the development process).

4. Design & Architecture

This project involves a range of devices and technologies. Please refer to Figure 2 which provides an overview of the project, broken down into five main components.

Back End

Firebase is used for the backend of the project. The Firebase services required for this project are the real-time databases and authentication, both of which are available within its "*Spark Plan*" (free tier). The real-time database is used to store the sensor data and the desired status of each wall outlet. The real-time aspect of the service means that as soon as the base station uploads a reading, it will be available for the user interface. Similarly, as soon the desired status of a wall socket (on/off) is changed on the user interface, the status will be available to the base station. The authentication service will ensure that access is kept (sufficiently) secure.

Base Station

A Raspberry Pi, Xbee device and 433MHs transmitter and receiver are used as the base station, which is responsible for:

- Listening for data from the sensor node
- Uploading sensor data to the back end
- Querying the back end for changes in the desired power outlet statuses.
- Transmitting on/off commands to the power outlets

A python script was created for the application logic and runs on an infinite loop on the RPi. The base station (Xbee) and sensor node utilise the IEEE 802.15.4 standard for communication (2.4GHz). The base station and wall sockets communicate using Radio Frequency in the 433MHz frequency band.

Sensor Node

An Arduino Uno, Xbee device and sensor circuit are used as the sensor node, which is responsible for:

- Taking sensor(s) readings at set intervals
- Transmitting the data to the base station

A sketch (Arduino program) was created for the application logic and runs on an infinite loop on the Arduino.

User Interface

An iOS application was developed for the user interface of the project and is responsible for:

- Viewing historic and real time data from the sensors

- Setting the status of the wall sockets.

RF Wall Sockets

Simple radio frequency-controlled wall sockets were used to switch the power supply on or off to a number of household appliances (when a transmission is received from the base station).

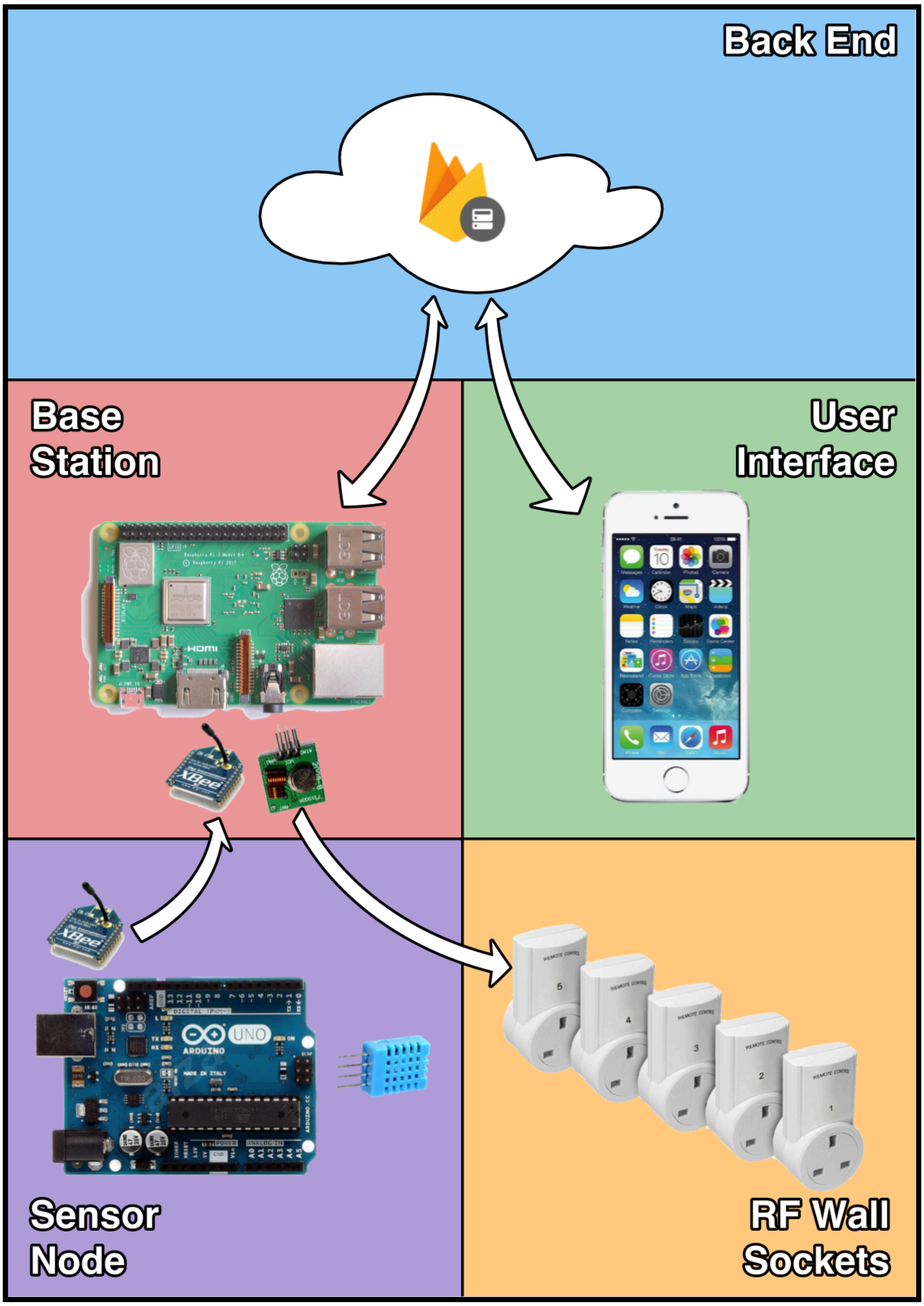

Figure 2 - Project Overview

5. Implementation

This section describes how the five main components of this project were implemented.

5.1. Backend (Database Design)

A Firebase real-time database is used in this project. A Firebase real-time database is structured as a single JSON tree. For this project, three root nodes are used; "*sensors*", "*sensorMeta"* and "*sockets*", as shown in Figure 3:

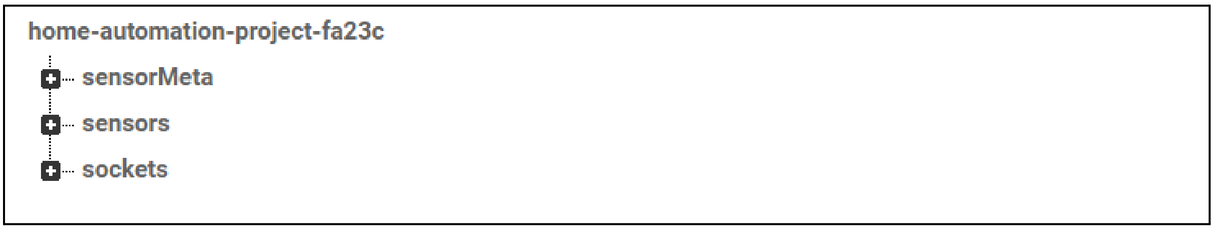

Figure 3 - Root Nodes

The *sensors* root node holds all of the data corresponding to the sensors, including any gathered data. Figure 4 highlights the structure of the *sensor* node.

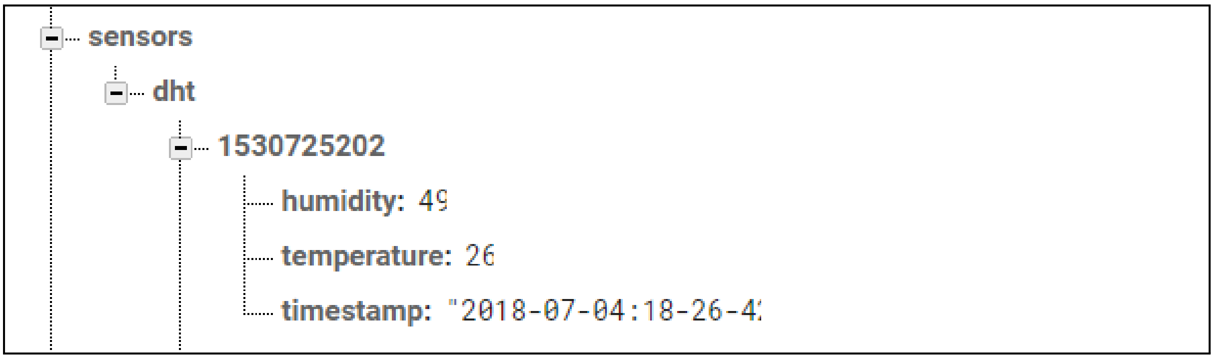

Figure 4 - Sensor Node

As there could potentially be multiple sensors included, the direct child nodes correspond to particular sensors. In the example provided, the "*dht*" node contains all of the data gathered from the DHT11 sensor on the wireless sensor node. If an additional sensor was included, its data would be contained in a separate child node.

As all of the gathered data is time series data, the key used for each entry is a timestamp (epoch time in seconds). For each timestamp, a human readable time (string) is provided alongside whatever data was gathered. In the example provided, both temperature and humidity readings were recorded.

The *sensorMeta* root node holds the meta data (non-time series data) for each of the sensors. Figure 5 highlights the structure of the *sensorMeta* node.

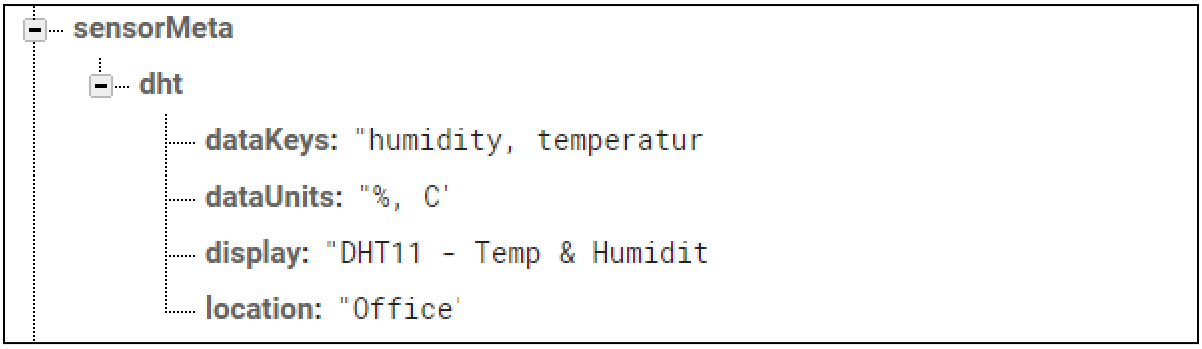

Figure 5 - Sensor Meta Data Node

This meta data is required for the iOS application to successfully parse the time series data and to initially display the list of available sensors. Each of the *dataKeys* are used when parsing the time series data (note that they correspond to the keys in Figure 4). The *dataUnits* are used when updating the y-axis units on the graphs. The *display* and *location* values are used on the sensor selection screen.

The sockets root node holds all of the data corresponding to the sockets, including the desired status (on/off) of each. Figure 6 highlights the structure of the sockets node.

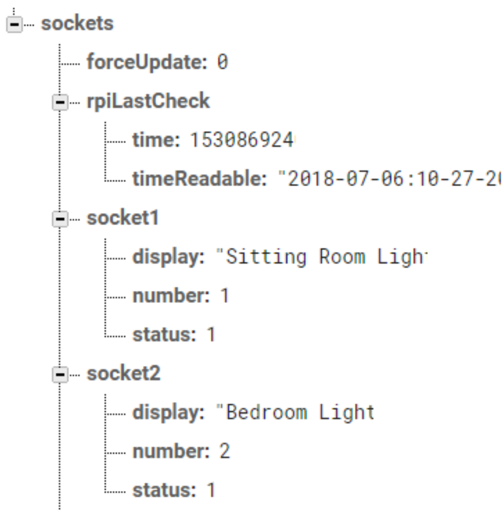

Figure 6 - Sockets Node

A child node is provided for each socket. Each of these nodes contain the display name (as it appears in the app), the socket number (the physical number on the device) and the desired status (1 = 0n, 0 = off) for their respective socket. An "*rpiLastCheck*" node is also provided. This node is where the last sync time of the base station is recorded. This enables the app to disable the UI if the base station hasn't synced recently.

A json export of the database was submitted along with this report.

5.2. **RF Wall Sockets**

The radio frequency-controlled wall sockets used in this project are readily available from various suppliers. A socket pack, consisting of five RF sockets and two remote controls was purchased for use in this project. The remote controls can be used to turn on or off each of the sockets.

Figure 7 - RF Wall Sockets & Remote Controls

5.2.1. **"*Sniffing*" the RF Codes**

The remote controls which come with the sockets are pre-programmed with the correct codes for turning on or off each socket. The remote controls have a 433MHz transmitter and each socket has a 433MHz receiver. When a button is pressed on the remote control, a 433MHz signal is transmitted with a code which corresponds to the socket number and command of the button pressed (e.g. socket 1, on). Each socket will receive the transmission, but only the socket whose on/off code matches the transmitted code will turn on/off its power.

In order to control the sockets from another 433MHz transmitter (i.e. one attached to the base station), it is necessary to determine what the codes are for each command. One method of doing this is to run a program which uses a 433MHz receiver to listen, or "*sniff*", for transmitted signals. Once the program is running, each button on the remote control can be pressed in turn and the corresponding code for each can be detected and recorded.

A program from a third-party library (Ninja Blocks, 2018) was used for sniffing the codes. The library was installed on the RPi and a 433MHz transmitter and receiver (complete with attached antennas) were connected to the RPi.

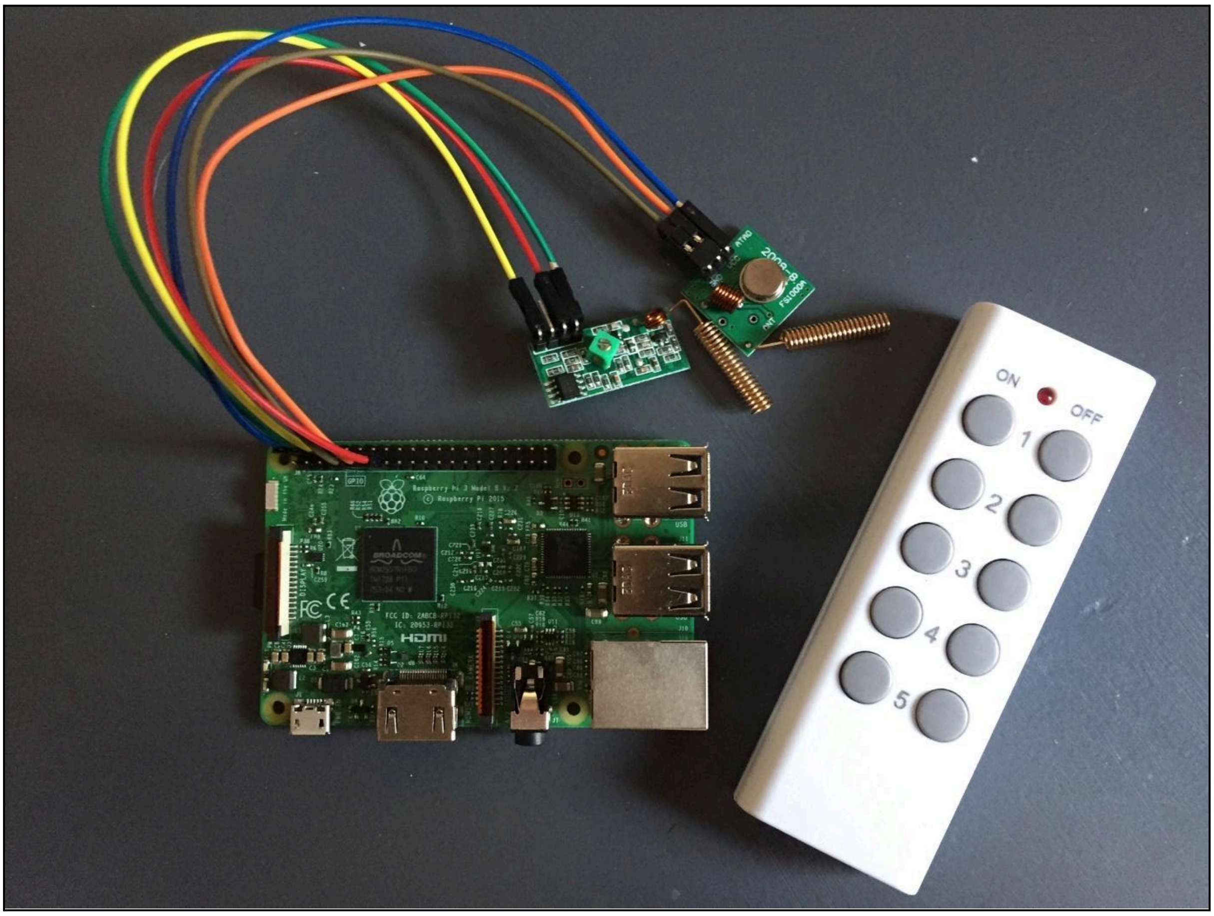

Figure 8 - RPi with 433MHz transmitter and receiver

The program was run from the command line and each of the buttons on the remote control were pressed in turn. When the program received a transmission, the code was printed out to the command line, as shown in Figure 9. These codes were recorded for later inclusion in the Base Station’s application logic.

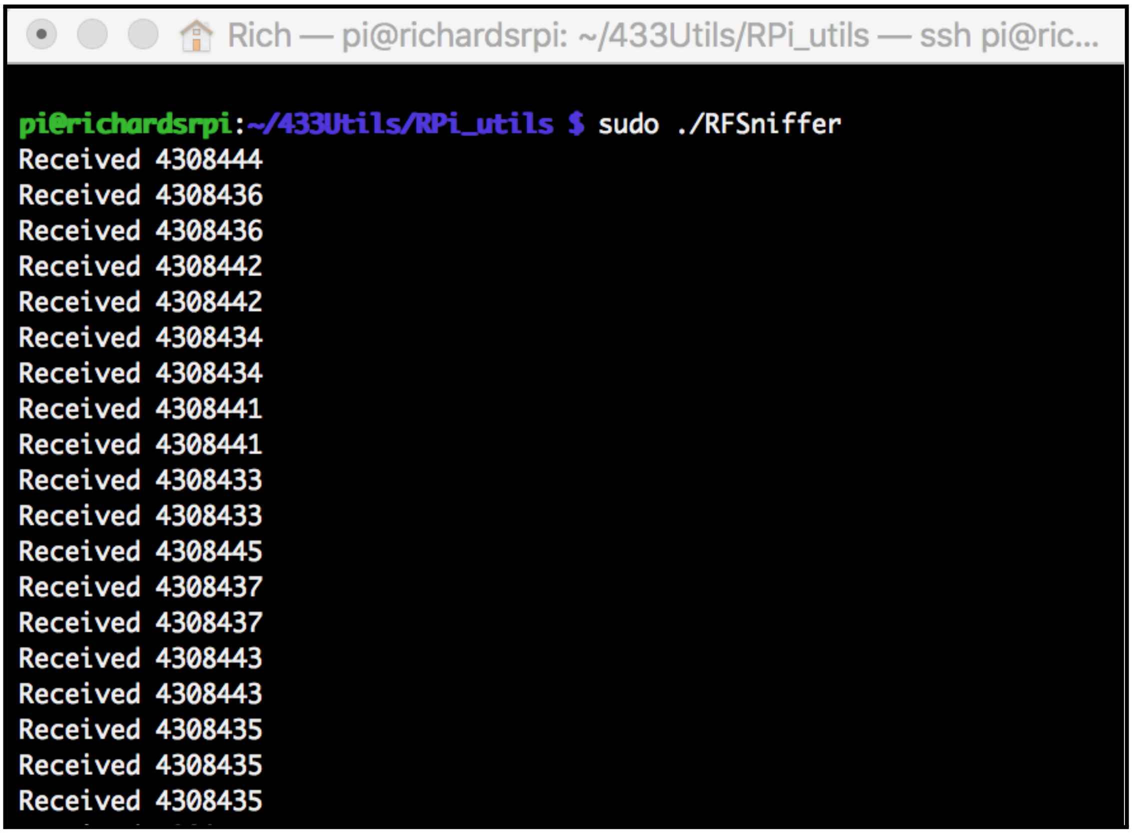

Figure 9 - "Sniffing" the Socket Codes

The codes discovered from this exercise are provided in Table 1.

Table 1 - Socket Codes

Socket	On Code	Off Code
1	4308444	4308436
2	4308442	4308434
3	4308441	4308433
4	4308445	4308437
5	4308443	4308435

5.2.2. **Positions of RF Wall Sockets**

The sockets can be used to control the power to any electrical item that plugs in to a standard wall socket. This provides great flexibility as the same RF wall sockets can be moved from one room to another depending on what appliance / item needs to be controlled.

Figure 10 shows the layout of my house. The current position of each socket is identified by the small purple icons with an adjacent number, which corresponds to the socket number.

Socket 1 and socket 5 in the Living Room control a plug-in light and the entertainment devices (TV, PS4, Sky Box) respectively. Multiple devices are controlled by socket 5 as they are all powered from a single extension lead. Socket 2 in the bedroom controls a plug-in light. Socket 4 in the office controls the power to multiple screens and laptop chargers (again through an extension lead). Socket 3 in the kitchen is currently not used to control anything.

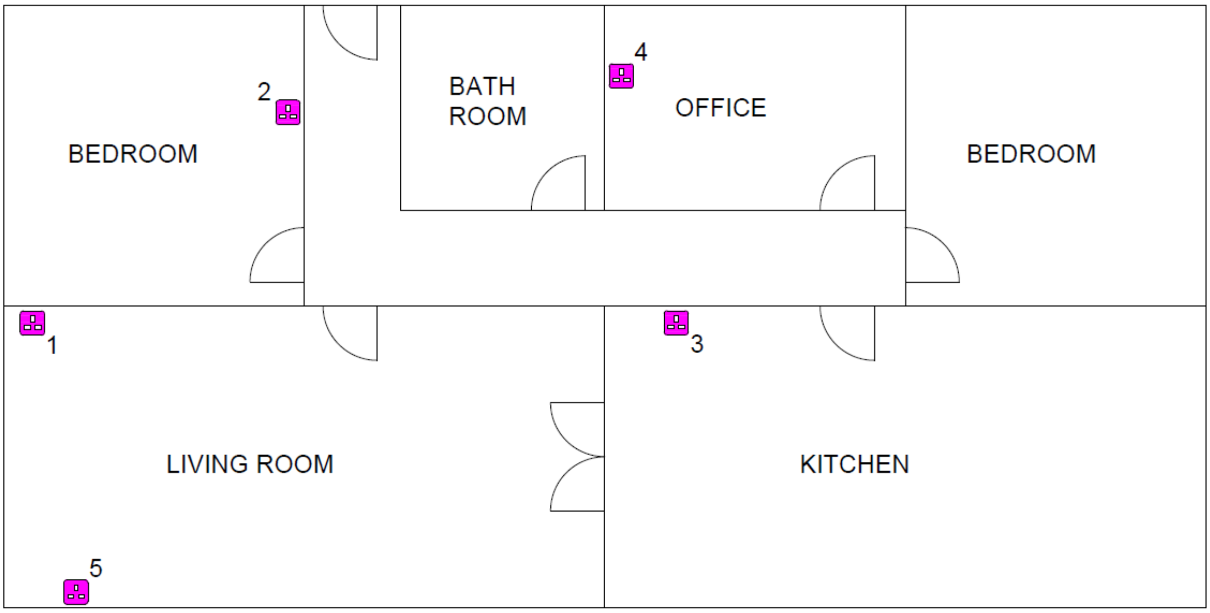

Figure 10 - Socket Positions

5.3. Wireless Sensor Node

The wireless sensor node (WSN) used in this project is responsible for gathering the environmental data and transmitting it back to the base station.

5.3.1. Architecture

The architecture of a sensor node is shown in Figure 11. Rather than attempting to build a sensor node from scratch, an Arduino Uno was used to provide the micro controller, analogue to digital converter and external memory components. In addition, an Arduino can easily be powered from a standard socket (or battery) and an integrated development environment (IDE) is available for developing and loading the "*sketches*" (programs) onto it. In this case, the power supply is via USB from a standard wall socket. Finally, the Xbee device acts as the transceiver, although it is only required for transmission in this application. The Xbee devices and network are discussed in greater detail in section 5.5.

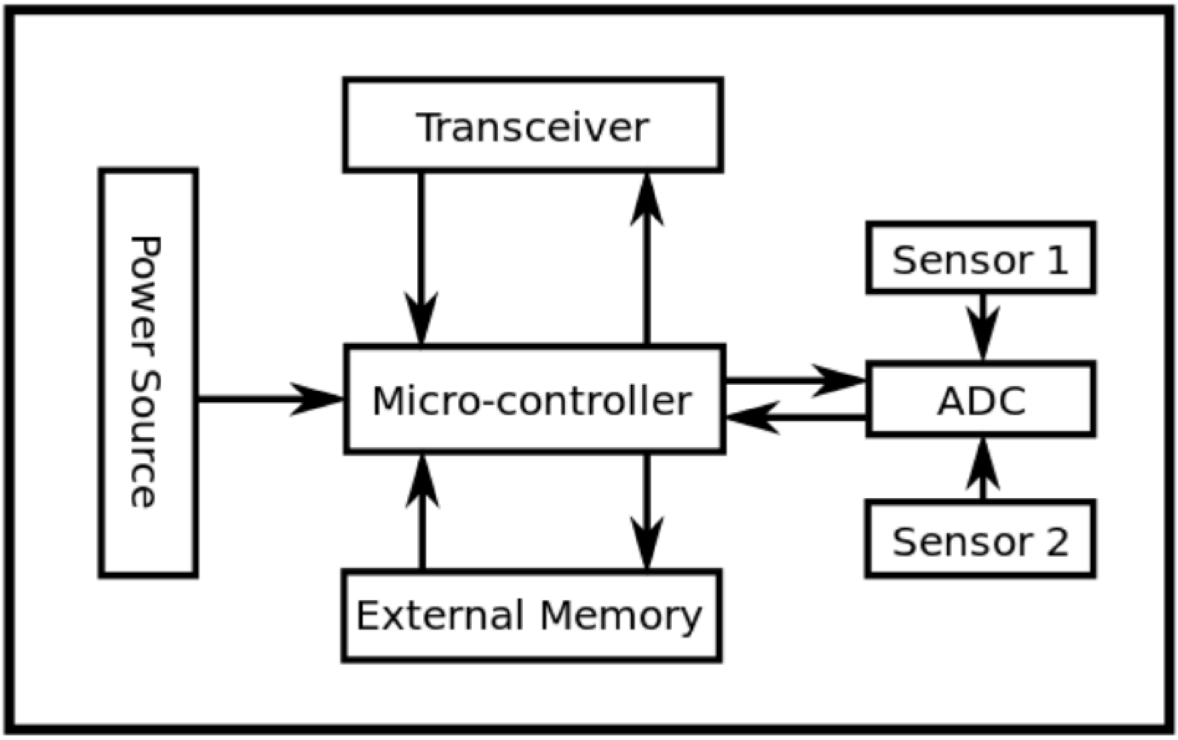

Figure 11 - Architecture of a Sensor Node (Carr & Zemouri, 2018)

5.3.2. Design & Implementation

The large number of I/O pins on the Arduino provided great flexibility and meant that multiple sensors and/or indicators could be included. The final design included a single DHT11 temperature and humidity sensor and three LEDs. A red LED was used to indicate an error, a green LED was used to indicate success and a blue LED was used to indicate activity. Figure 12 shows the wiring diagram between the Arduino and the sensor and LEDs.

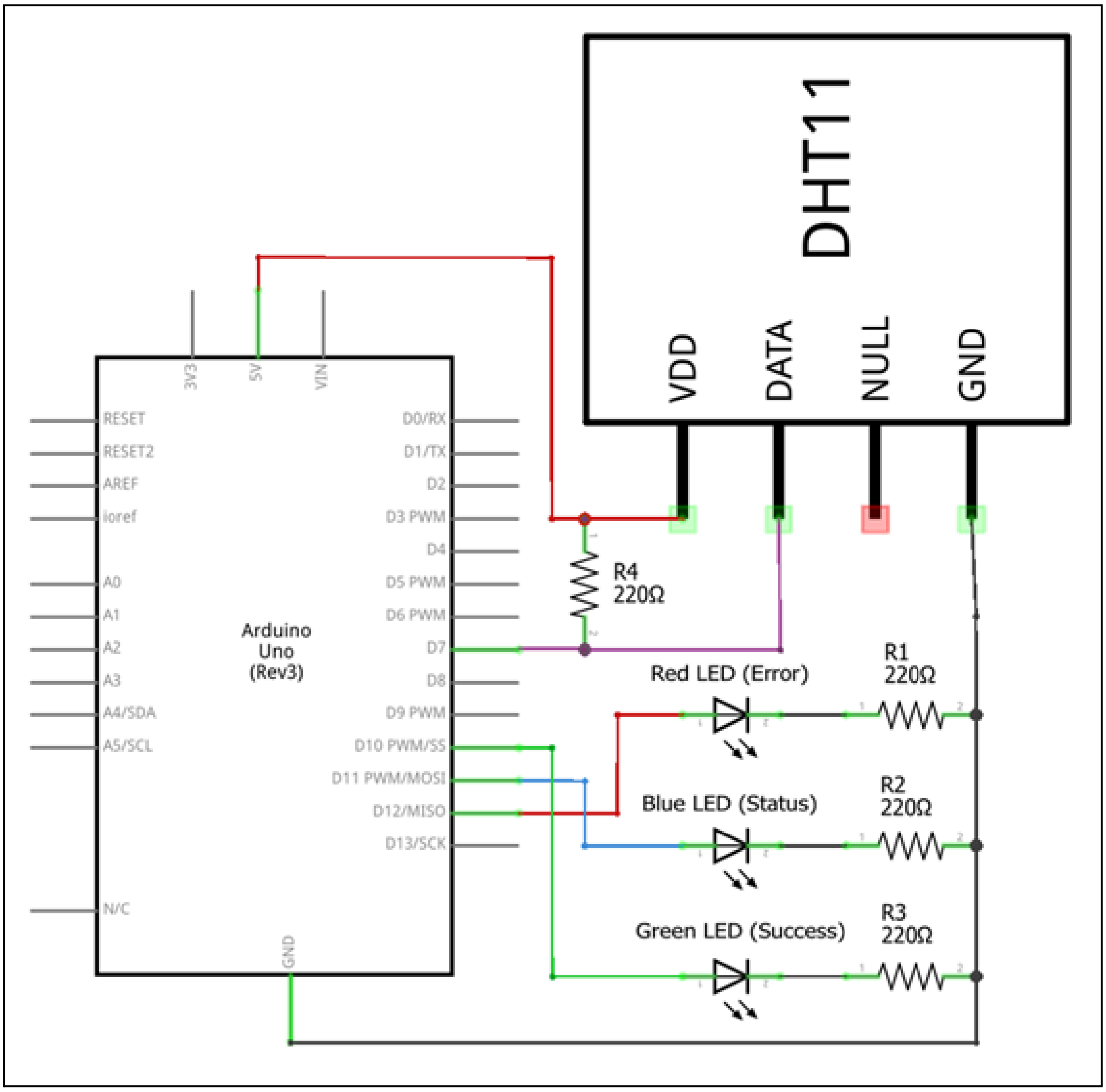

Figure 12 - Sensor Node Wiring Diagram

In order to interface the Xbee with the Arduino, a *shield* is used. A shield is simply a modular board which is plugged in on top of the Arduino in order to extend its capabilities. In this case, the shield provides a seat for the Xbee device, Xbee indicator LEDs (useful for troubleshooting / testing), a breakout space and replicates all of the I/O pins and the reset button from below.

Figure 12 and Figure 13 show the design of the WSN and the actual WSN respectively. Jumper wires were used for all connections.

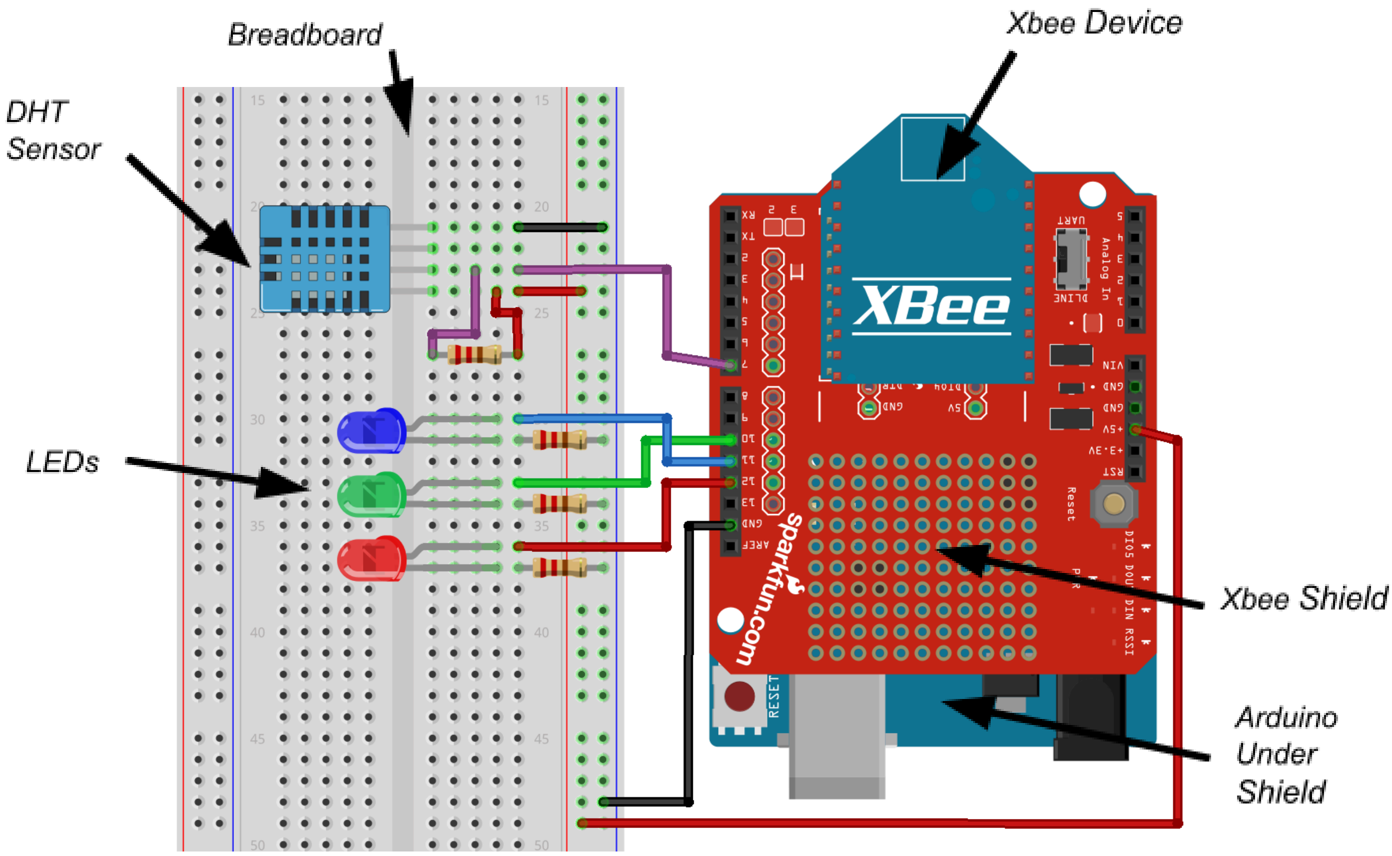

Figure 13 - WSN Design

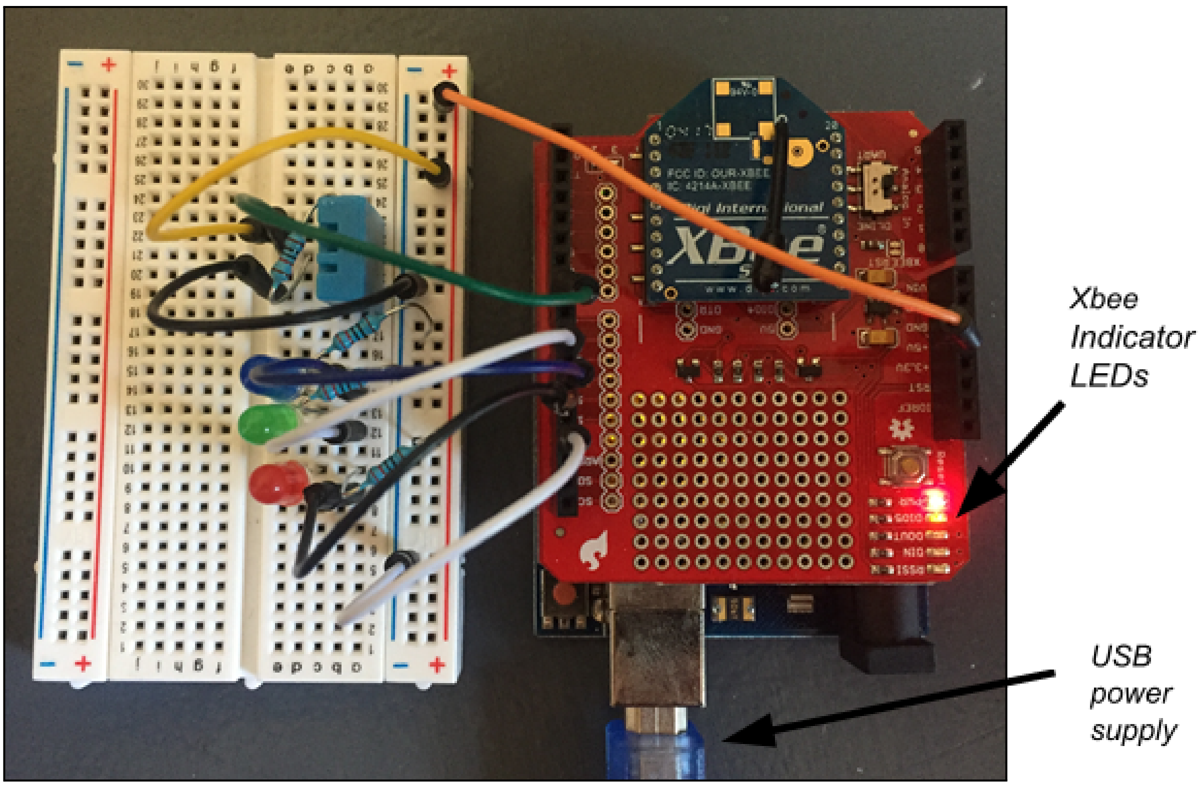

Figure 14 - WSN Actual

5.3.3. **Arduino Sketch**

A sketch is a block of code (program) that is uploaded to and run on an Arduino. The Arduino language used in a sketch is merely a set of C/C++ functions, which are passed to a C/C++ compiler.

Every sketch must contain two special functions*;* *setup()* and *loop().* The *setup()* function is called once, and as the name implies, is used for setup tasks and initialisation. The *loop()* function is called over and over again once the *setup()* function has run.

For this project, the *setup()* function is used to set the various pin modes, set the serial baud rate and initialise the Xbee object (Rapp, 2016). The *loop()* function then continuously takes readings from the sensor and attempts to transmit them to the base station. The various LEDs are used throughout the loop as indicators. An artificial delay is added at the end of the loop to limit the number of sensor readings taken.

The sketch developed for this project is available at the Github link below. It was also submitted together with this report.

https://github.com/Richard-Seaman/FinalProject-Arduino

5.4. **Base Station**

The base station (BS) used in this project is responsible for:

- receiving data from the wireless sensor node and uploading it to the back end
- checking the back end for socket status changes and transmitting the relevant on/off socket commands

5.4.1. **Architecture**

Figure 15 shows the architecture used for the base station. An internet connected computer is used as the main computational platform. A 433MHz transmitter (TX) and receiver (RX) are used to enable radio communication on the 433MHz frequency band. An Xbee is used to allow access to (and coordination of) the Xbee network.

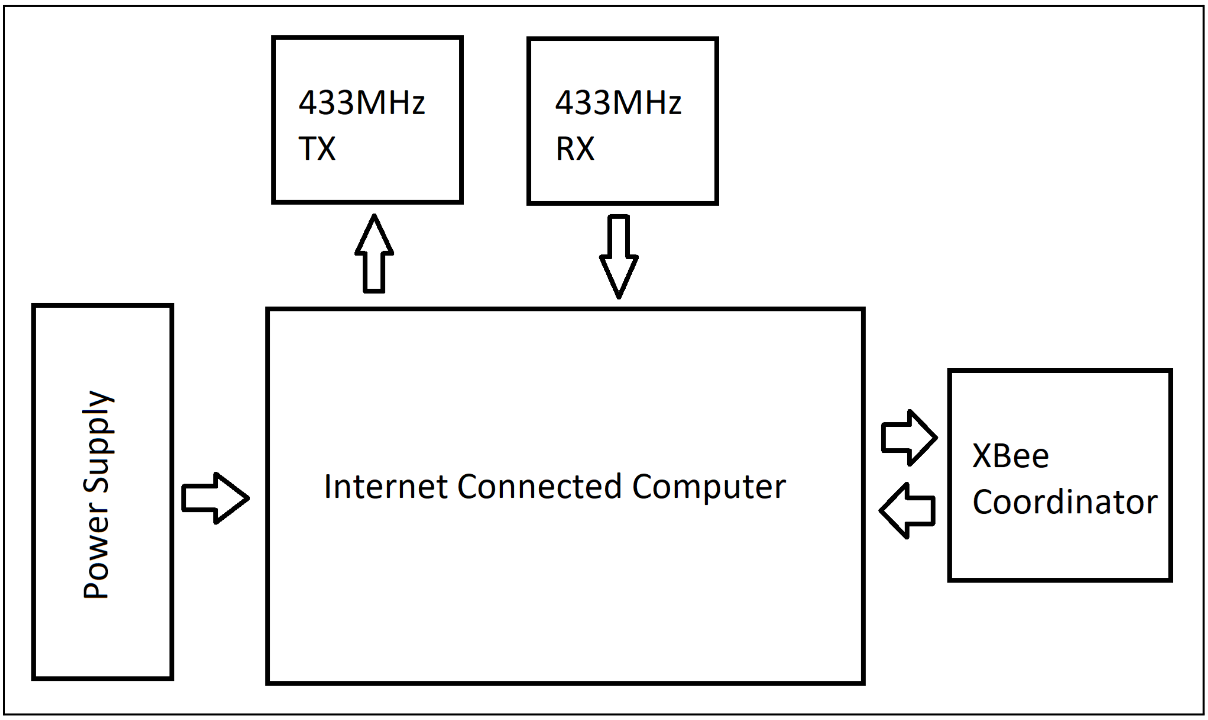

Figure 15 - Base Station Architecture

5.4.2. **Design & Implementation**

A Raspberry Pi (RPi) 3 Model B V1.2 is used as the base station's internet connected computer. A generic 433MHz transmitter and receiver are used for transmitting and receiving 433MHz radio signals. Antennas were soldered to both the TX and RX in order to improve performance. An Xbee Series 1 complete with explorer dongle (USB interface) is used as the Xbee coordinator. Figure 16 shows the wiring diagram for the RPi's I/O pins.

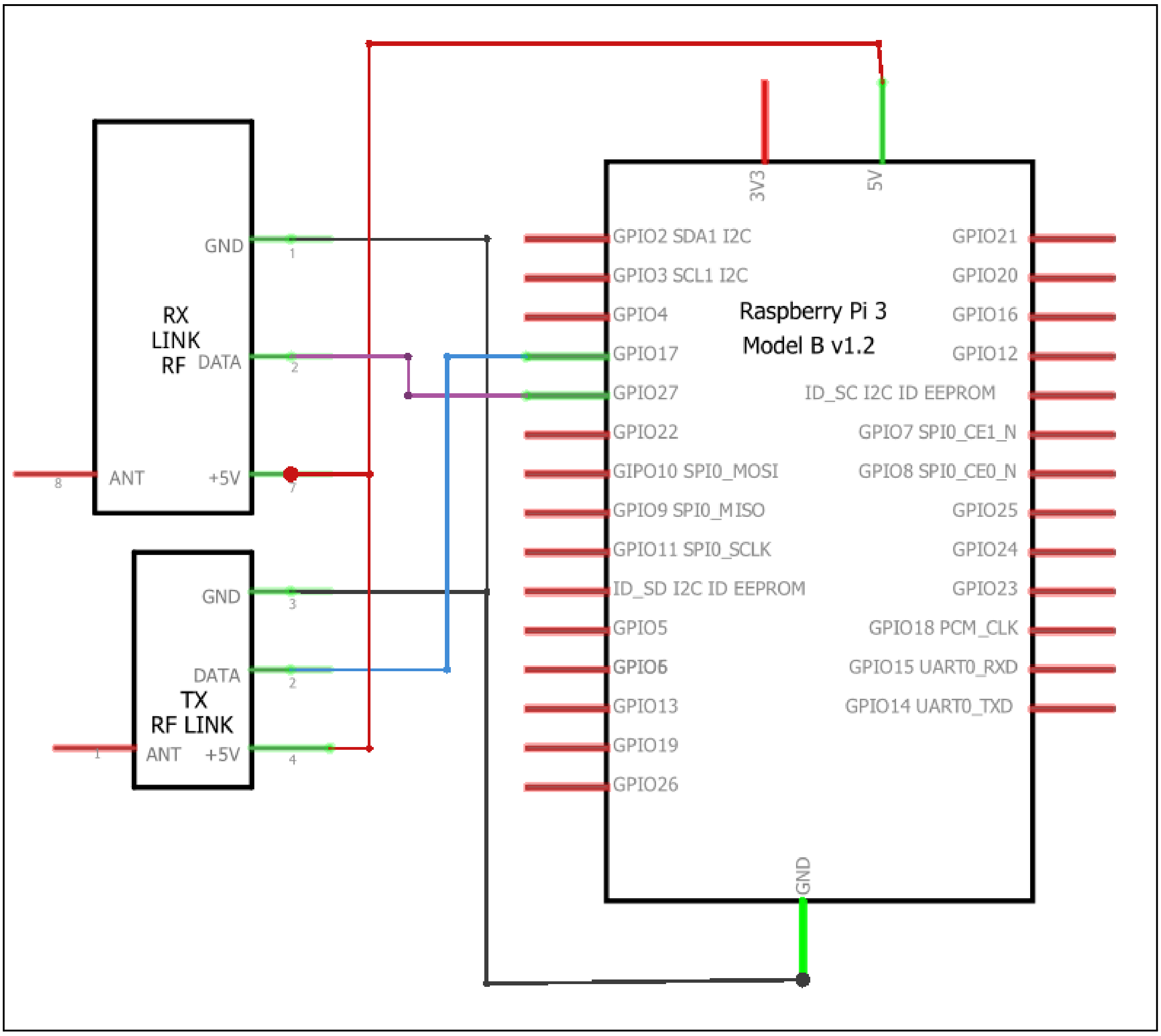

Figure 16 – RPi Wiring Diagram

Figure 17 and Figure 18 show the design of the base station and the actual base station respectively. Jumper wires were used for all connections.

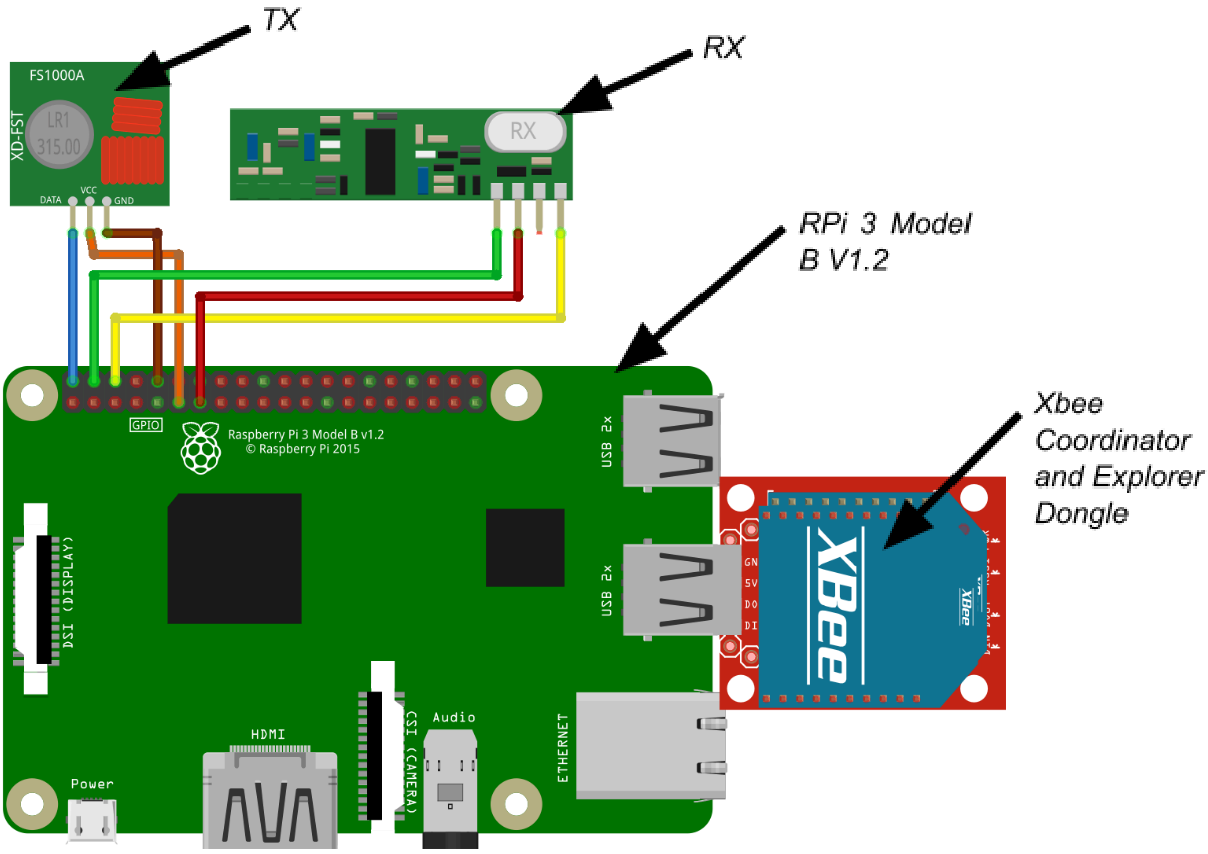

Figure 17 - Base Station Design

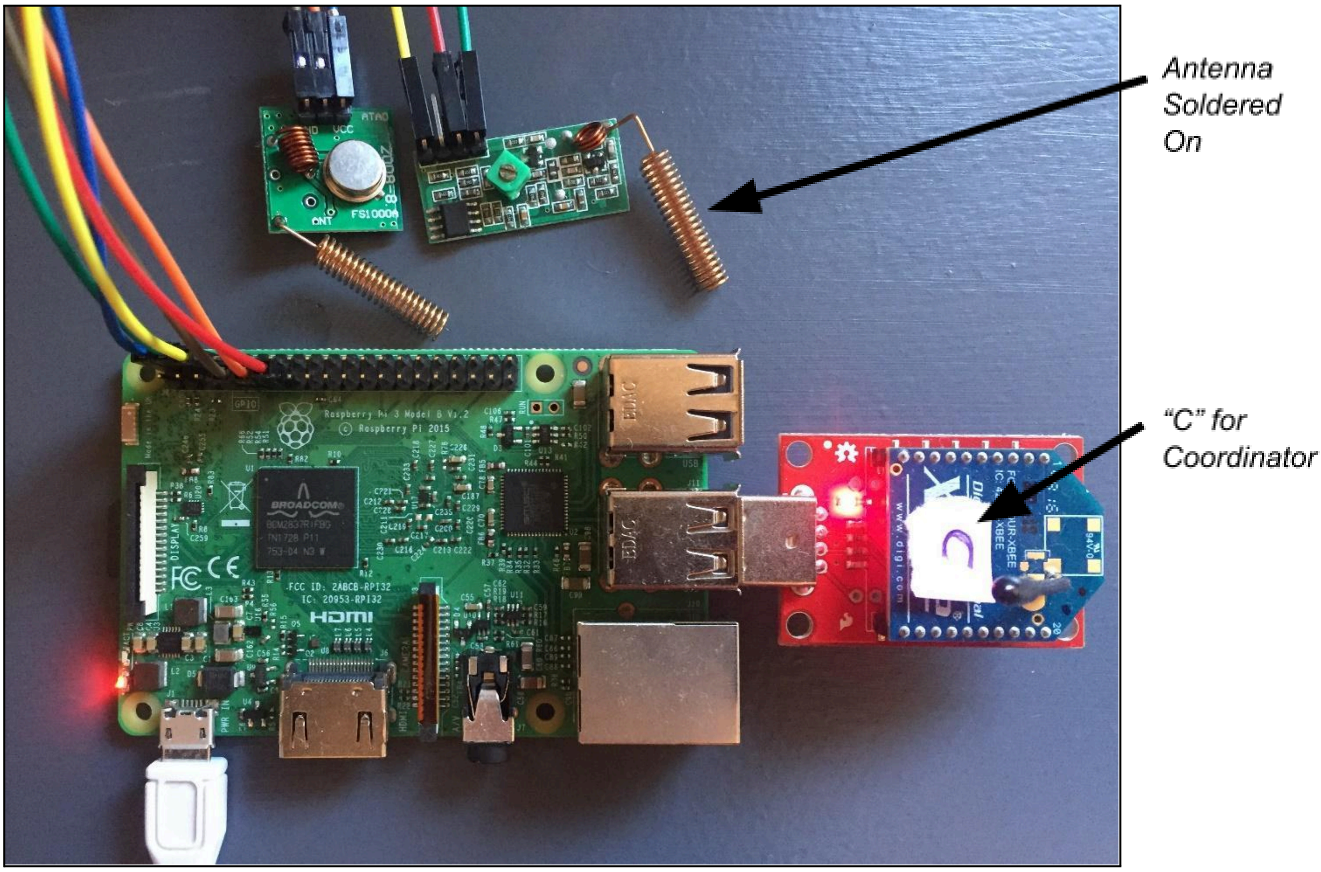

Figure 18 - Base Station Actual

5.4.3. “*Main.py*” Program

A single “Main.py” python program was written which contains all of the base station’s application logic.

One of the first things the program does, is create a log file. Any existing log files are timestamped and archived before a new log file is created. This log file allows important events or errors to be logged, which is invaluable during the development process. Having the log file as a separate file also allows it to be *tailed* while SSH’ed into the RPi. A number of these log files were included in this submission as examples.

The program authenticates itself using the hardcoded credentials provided and creates a persistent *reference* to the real-time Firebase database. This enables it to both read from and write to the database.

A serial port is opened to the Xbee device and an Xbee object is created (niolabs, 2018). This Xbee object provides an API for any communication to/from the Xbee network. For this project, the Xbee is only expected to receive transmissions from the WSN. When a transmission is received, the Xbee object calls a designated callback function on a background thread. In this case, the callback function handles the processing and uploading of the data received.

Each of the RF socket on/off commands are hardcoded into the program. The socket statuses are queried from the database and the corresponding commands are issued. A third-party library (Ninja Blocks, 2018) is utilised for issuing 433MHz RF signals. The program enters a continuous loop and periodically queries the database for changes to the socket statuses. If a change is found, the corresponding signal is issued.

Any signal interrupts are handled and the program gracefully exits.

The python program developed for this project is available at the Github link below. It was also submitted together with this report.

https://github.com/Richard-Seaman/FinalProject-Rpi

5.5. Xbee Devices & Network

The Xbee devices and network represent one of the most difficult aspects of this project. The devices and configuration software have a very steep learning curve and the supplier's documentation only cover generic use cases (DIGI, 2018). That being said, once the system is understood and implemented correctly, it becomes incredibly powerful with a vast range of possible applications.

At a very high level, Xbee devices operate in two main modes; transparent mode and API mode.

5.5.1. Transparent Mode

Transparent mode is the default operating mode. Xbee devices can actually communicate with each other straight out of the box, using transparent mode. The device acts as a serial line replacement when it is in this mode. Essentially, any RF data it receives, it transmits back out. The data passes through the device, which assumedly is where the name came from.

Figure 19 shows the initial testing of the Xbee devices in transparent mode. The window on the left is the Serial Monitor of the Arduino IDE (Xbee and shield connected to Arduino) and the window on the right shows the *"XCTU"* software provided by DIGI for configuring the devices (Xbee dongle connected to Macbook).

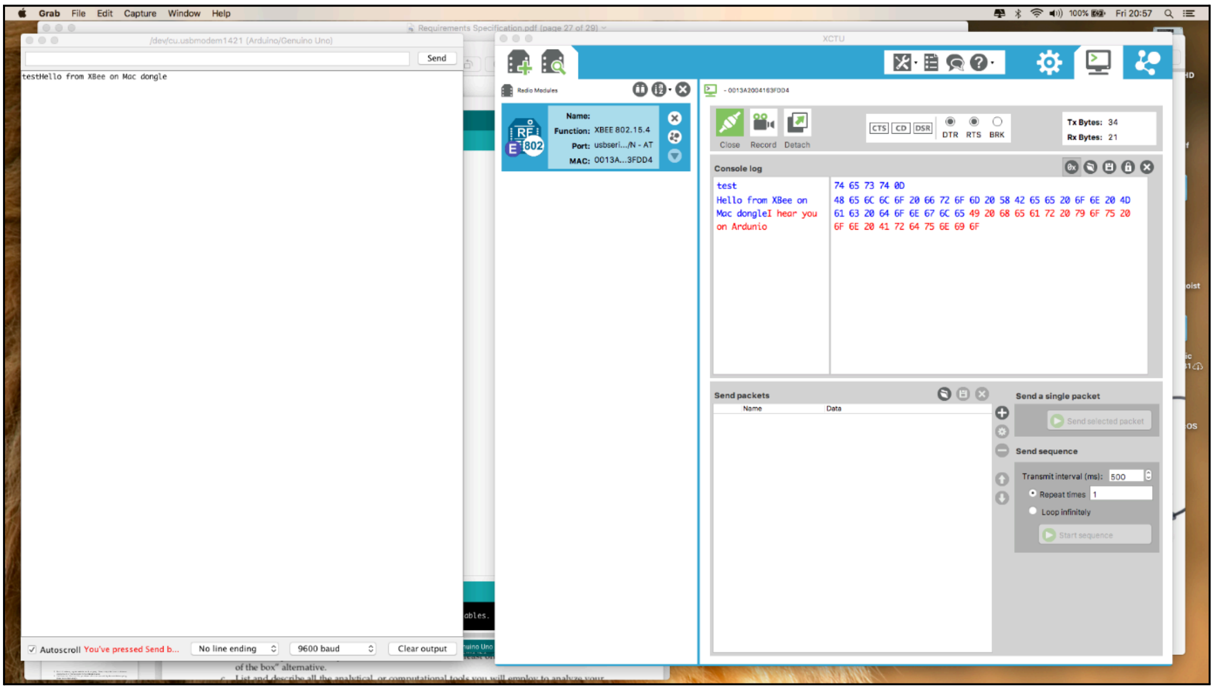

Figure 19 - Initial Testing of Serial Communication Between XBee Devices

5.5.2. **API Mode**

Application Programming Interface (API) operating mode is the alternative to transparent mode. It is much more powerful, but also much more complicated. When in API mode, all data that is received or transmitted by the device is contained within frames which define operations or events within the module. Frames can include data frames or command frames, for transmitting data payloads or remote configuration of devices. Frames also include signal strength, source addresses, response codes and much more, which is how API mode is capable of so much more than transparent mode.

Figure 20 shows the frames log within the XCTU software. In this case, a coordinator device has received a frame from an end device. The data payload (*RF data* in the bottom right of the figure) corresponds to temperature and humidity readings of 18C and 40% respectively.

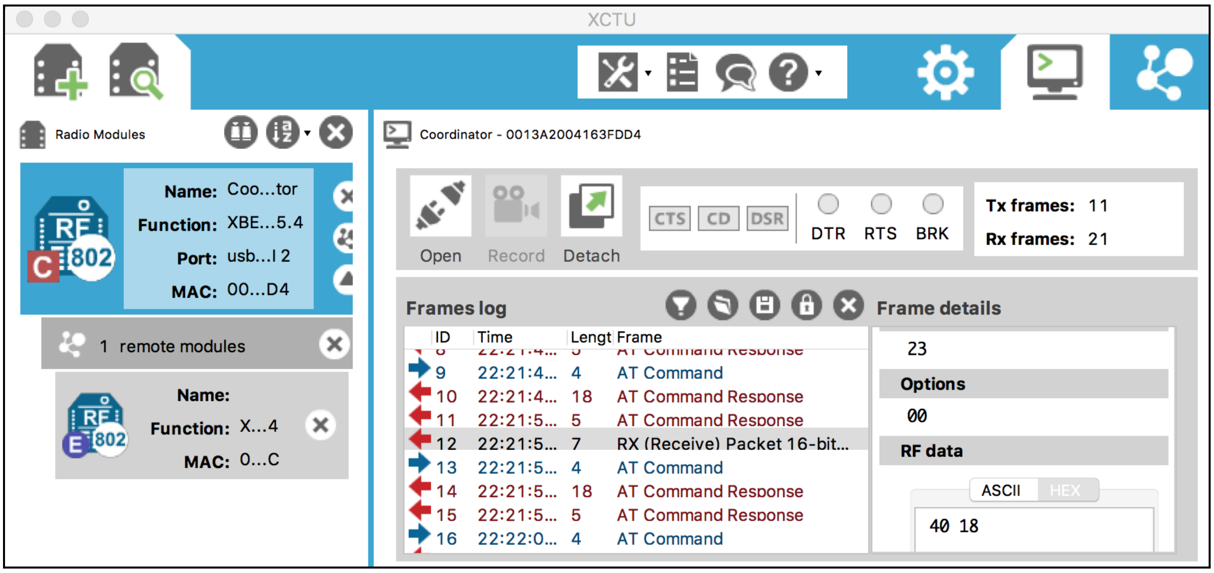

Figure 20 - API Mode: frames

5.5.3. **Device Configuration**

The devices were configured in API mode. The Xbee device for the base station was configured as the network coordinator. The Xbee device for the WSN was configured as an end device. Each device was configured to use the same channel and network ID and a 16-bit source address was designated for each device.

Table 2 provides the final configurations used for each of the devices. Any settings not included in this table were kept as the default values. Figure 21 shows the network configuration within the XCTU software, including a network diagram.

Once the devices were configured correctly, the various software libraries could be used within the base station / WSN programs.

Table 2 - XBee Device Configuration

Description	Abbrev.	Xbee 1 (BS)	Xbee 2 (WSN)
Node Identifier	NI	Coordinator	End Device
Channel	CH	C	C
Pan ID	ID	3385	3385
Dest. Addr. High	DH		0
Dest. Addr. Low	DL		0
16-bit Source Addr.	MY	1234	5678
Serial High	SH	0013A200	0013A200
Serial Low	SL	4163FDD4	4151306C
Coordinator Enable	CE	Coordinator [1]	End Device [0]
Interface Data Rate	BD	9600 [3]	9600 [3]
API Enable	AP	API enabled [2]	API enabled [2]

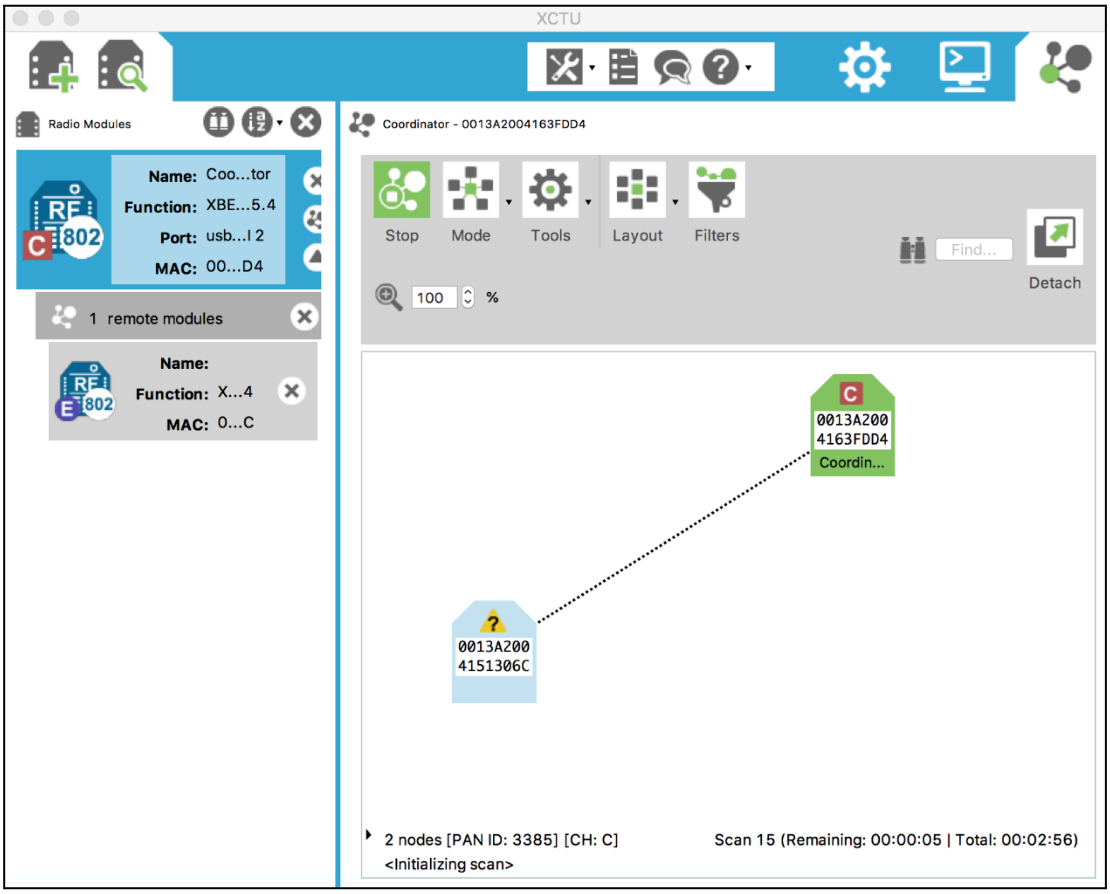

Figure 21 - Xbee Device Network Configuration

5.6. Graphical User Interface (iOS App)

The graphical user interface is an iOS application. The application was not published to Apple's AppStore as there is only one intended user. In fact, making the application available to multiple users would compromise the confidential information (monitored data) as well as the control of the wall sockets. For these reasons, the application must be loaded directly onto each iOS device, through XCode.

5.6.1. Architecture

A tab-based interface is used with two main tabs. The first tab is for controlling the RF Wall Sockets and the second tab is for viewing and analysing the monitored data. Using tabs in this manner provides a clean division between the Sockets and Sensors aspects of the application.

5.6.2. Sockets Tab

Figure 22 shows the socket tab of the iOS app, which provides an interface for controlling the sockets. There are two main sections included on this tab: "*RPI*" and "*Sockets*".

The RPI section provides information on when the base station last synced with the back end (Firebase). If the time shown is relatively recently, it means that the base station is available and running the required script for the control of the sockets to work. If the time shown isn't relatively recently, it means that the base station is unavailable (for whatever reason). In this case, a warning will be displayed over the sockets tab and the interface will be locked, preventing the user from attempting to control the sockets, as shown in the screenshot on the right of Figure 23.

The Sockets section provides information on each of the available sockets. Each socket has:

- The socket number (physical number on the socket)
- The display name for that socket, which is user adjustable. This allows the name to be changed if the same socket is moved from one location to another or has something new plugged into it. An example is provided in the screenshot on the left of Figure 23.
- The status switch, which allows the user to turn on or off the socket.

Note that the application parses all of the information from the back end and displays it accordingly. This means that if sockets are added or removed from the backend, the sockets shown within the app will automatically be updated.

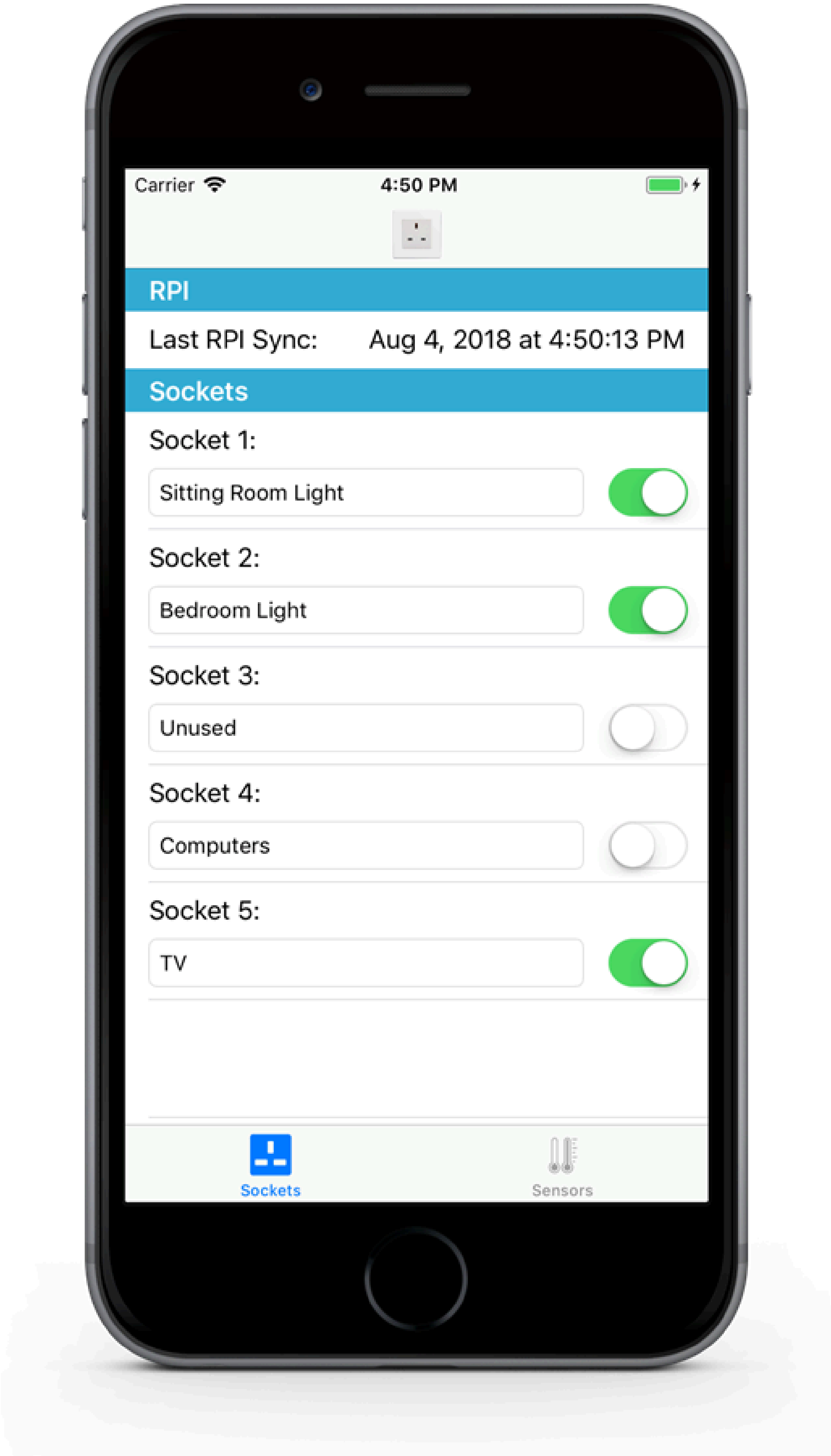

Figure 22 - Sockets Tab

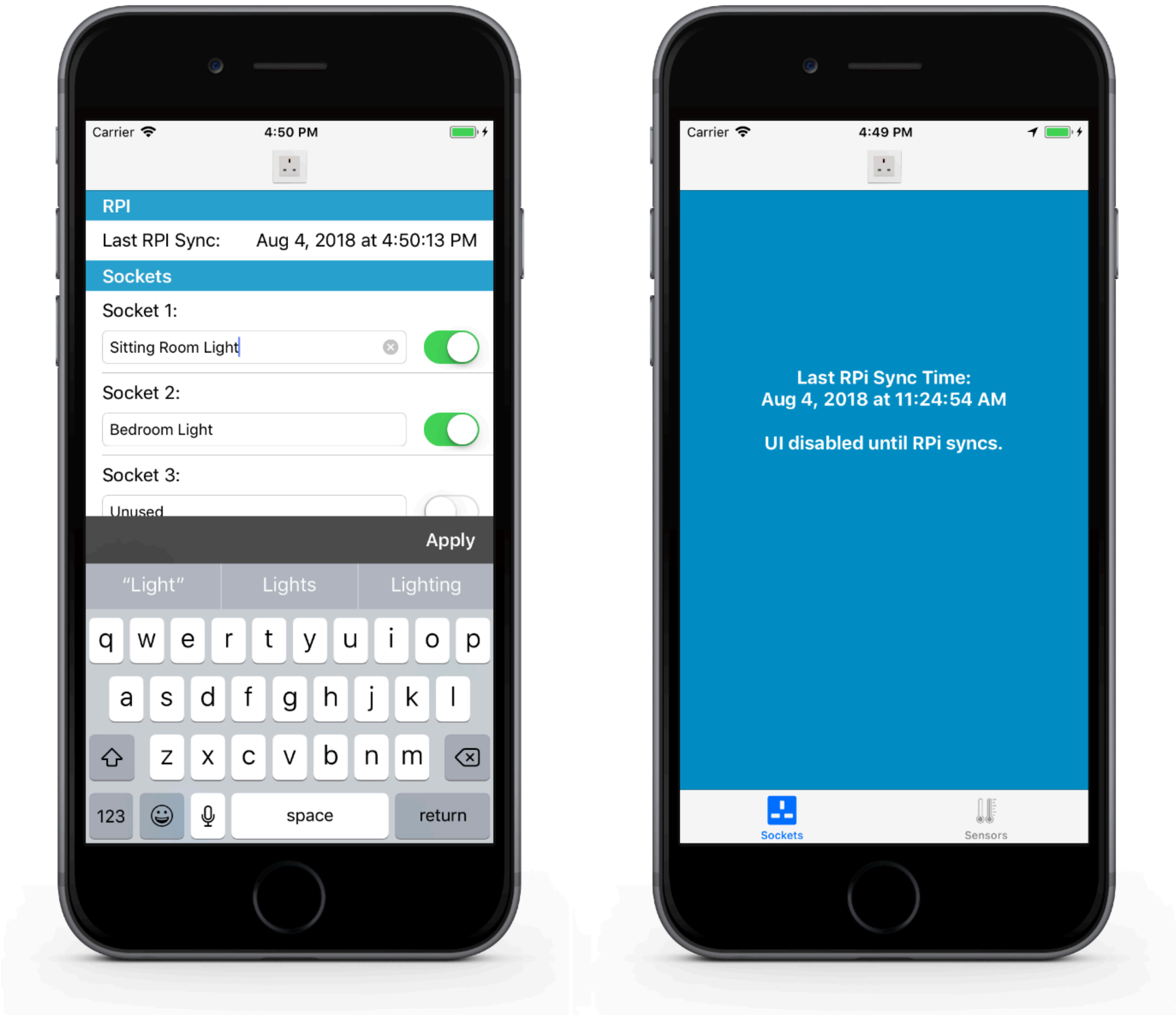

Figure 23 - Keyboard and RPi Sync Error

5.6.3. **Sensors Tab**

The first screen of the "*Sensors*" tab is shown in Figure 24. This "*selector*" screen allows the user to select a sensor from a list of available sensors. The list of available sensors is automatically parsed from the database. The sensor meta data (display and location) are displayed for each sensor. Similarly to the sockets tab, the location of the sensor is user adjustable.

When a sensor is selected, a new "*data*" screen is displayed which graphs the available data for that sensor against time. An example of the data screen is provided in Figure 25. Tapping "*Back*" on this screen will return to the selector screen.

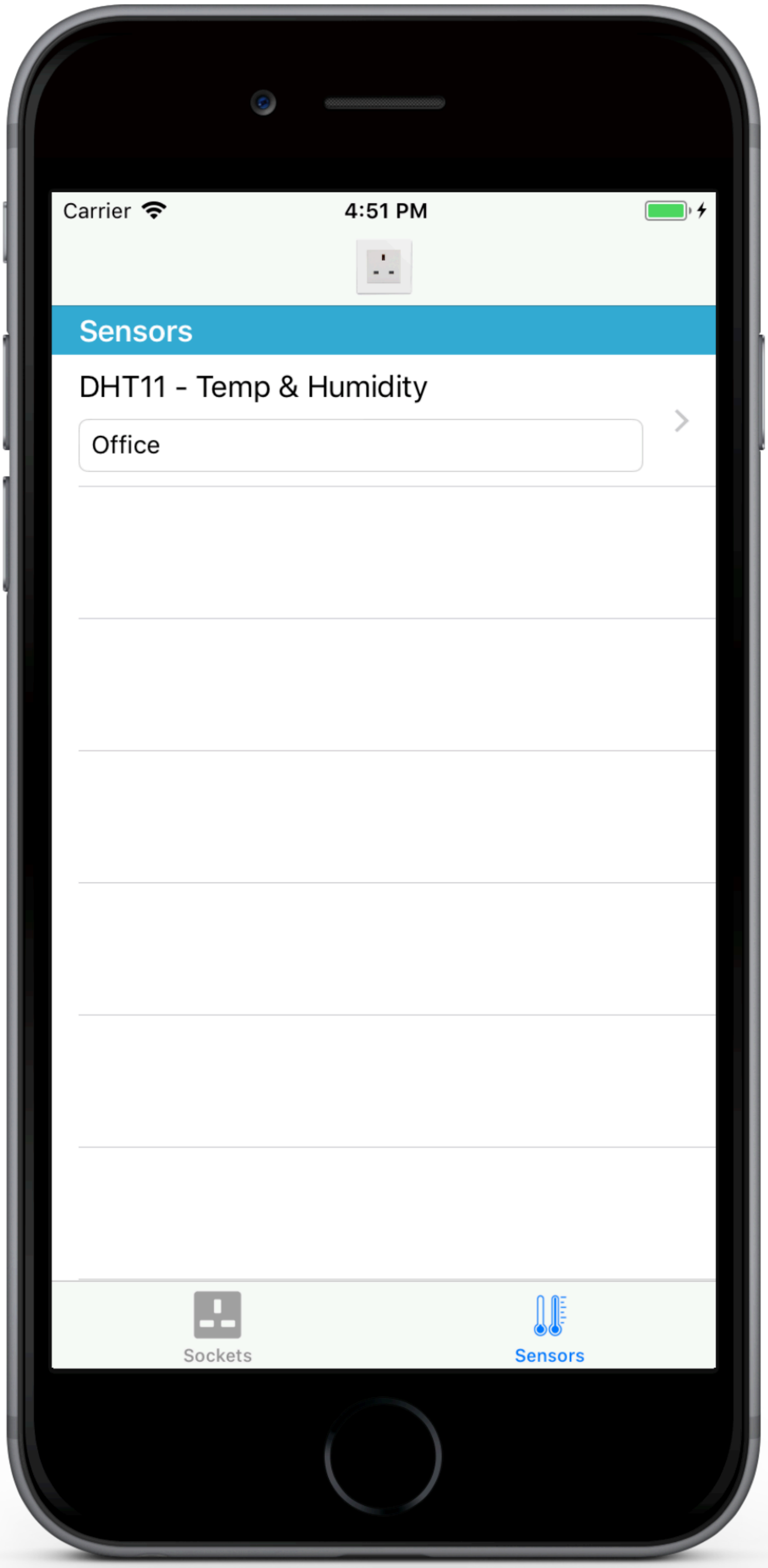

Figure 24 - Sensors Tab (Selector)

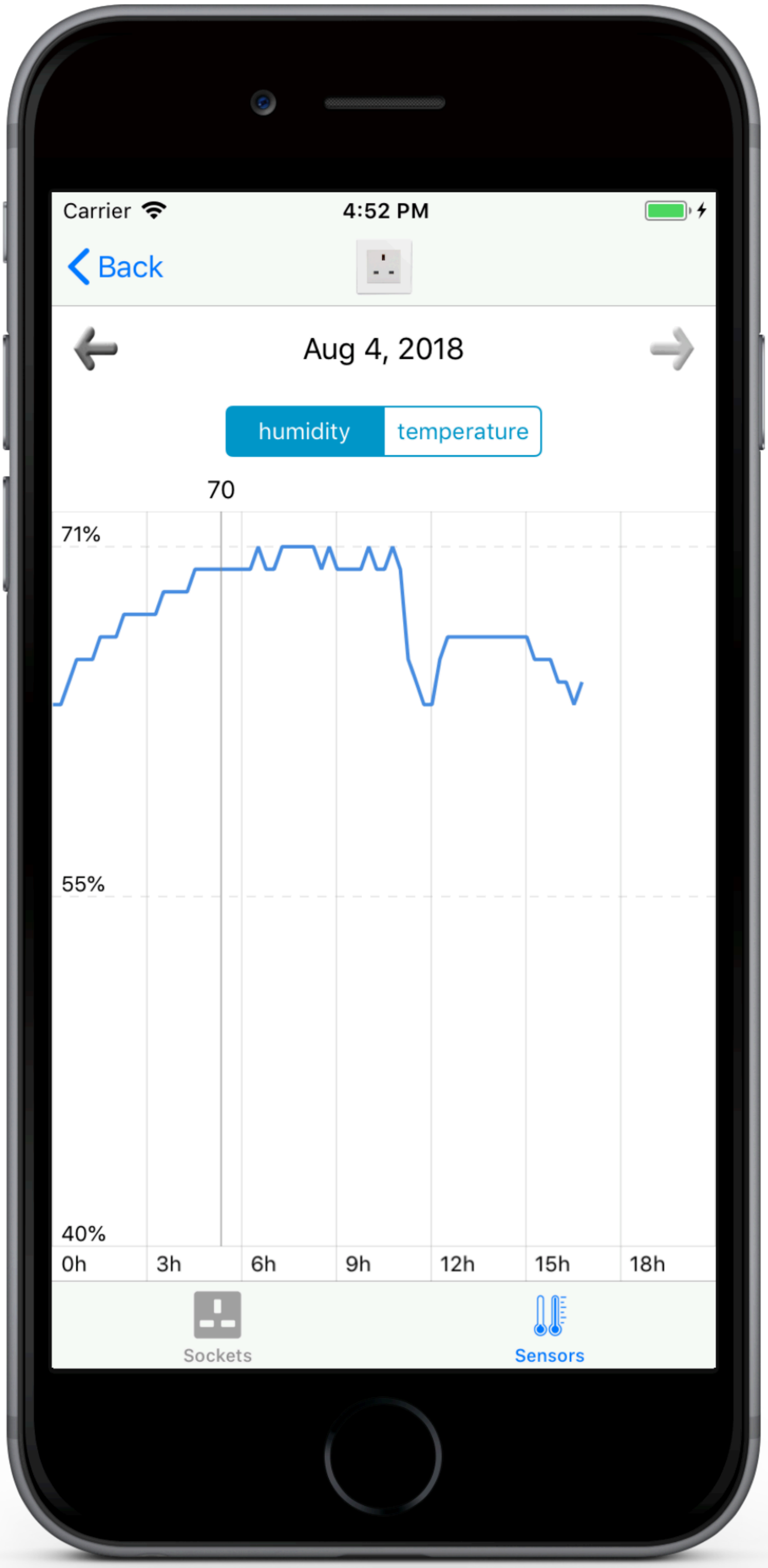

Figure 25 - Sensors Tab (Data)

The data screen graphs any available data for the selected day. The default date shown is the current day. It is possible to view historic data by using the arrows at the top left and right of the screen to move back and forward in increments of 1 day respectively. If the current date is selected, the next arrow (at the top right) is disabled.

If the selected sensor takes multiple readings (as is the case with the DHT11 sensor used), a reading selector will be displayed on top of the graph which can be used to swap between readings. If another reading is selected, the data and y-axis unit and scale will be updated accordingly. The sensor meta data from the database is used to achieve this functionality. The screenshot on the left of Figure 26 shows the alternative temperature readings.

If the value at a particular time is required, the user can tap anywhere on the graph to make a vertical line appear with the value at that time displayed above it.

As there is potentially a huge amount of data, a loading screen is displayed while the data is asynchronously fetched from the database. This loading screen is shown in the screenshot on the right of Figure 26.

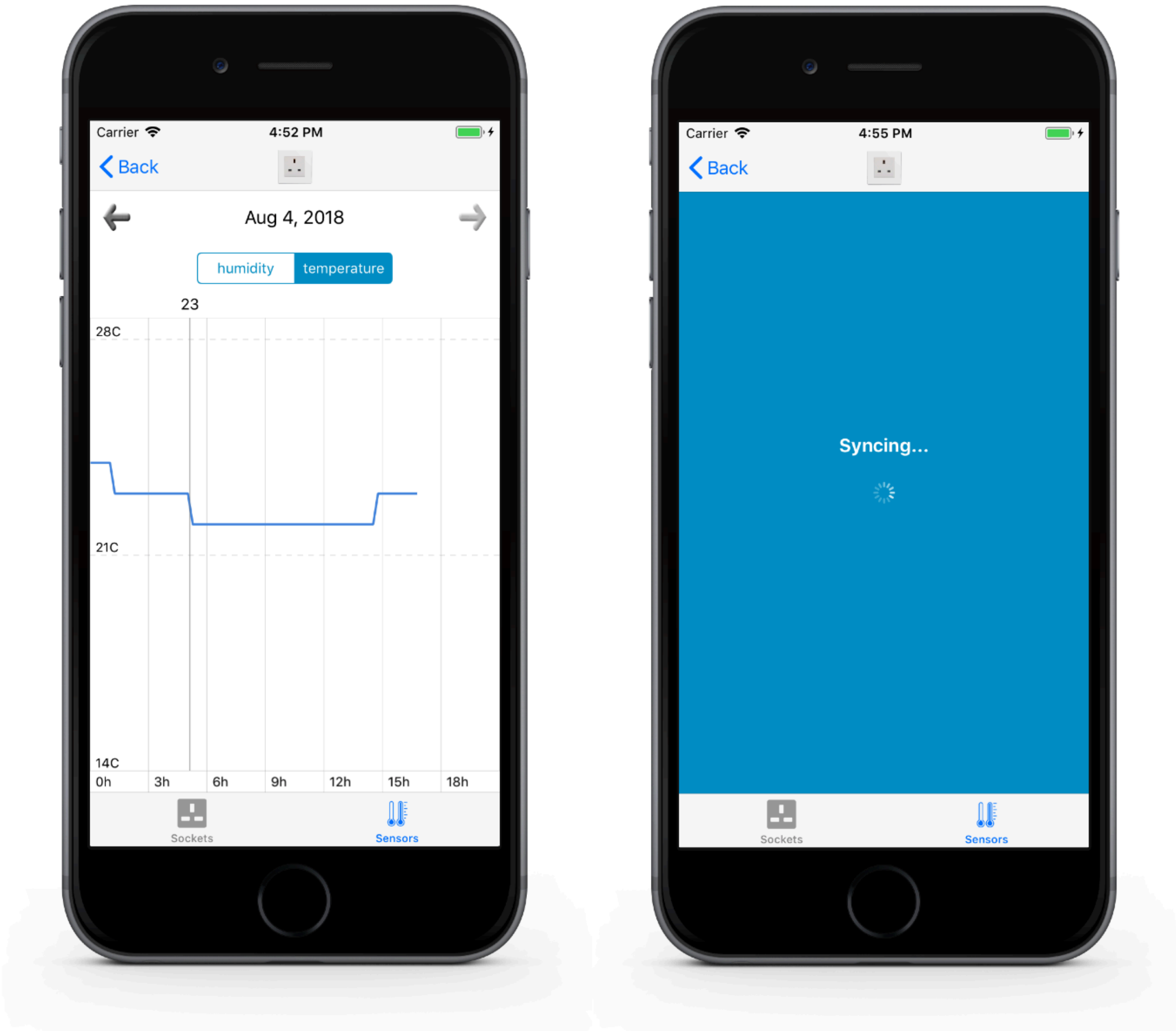

Figure 26 - Temperature Data & Loading Screen

5.6.4. **Source Code**

XCode was used to develop the iOS application for this project.

The source code for this application is available at the Github link below. It was also submitted together with this report.

https://github.com/Richard-Seaman/FinalProject-IOS

5.6.5.

6. Testing

Testing is incredibly important for any application. This is particularly true for this project, which consists of many different hardware, software and network components which must all work together to achieve the overall project objectives.

This section outlines the formal testing procedures and cases which were carried out to test this project. A number of white box and black box tests were undertaken. Each test is described in turn and the test procedure and result are provided in each case.

There was of course much informal testing carried out during the development process itself. However, these tests were not captured in any form of documentation. Nonetheless, these ongoing tests contributed to the overall quality of the project and the ultimate success of the black box test cases.

6.1. WHITE BOX TESTING

A number of white box tests were carried out to ensure the correct operation of the individual components within the system.

6.1.1. **DHT Sensor**

The sensor selected for the wireless sensor node was a DHT11 temperature and humidity sensor. The sensor was wired up to the Arduino and a preliminary sketch was developed to test the sensor readings and ensure the sensor responded to changes in its environment.

Test Case Description	DHT Sensor - Temperature
Tested By	Richard Seaman
Test Date	07/06/2018

Prerequisites

1	The sensor must be wired up to the Arduino (as per the final circuit design).
2	The WSN must be powered on.
3	The preliminary Arduino script must be used to print out the DHT sensor readings within the Serial Monitor of the Arduino IDE.
4	The preliminary script must be running on the Arduino.

Test Conditions

Step #	Step Details	Expected Results	Actual Results	Pass / Fail
1	Record the initial temperature reading from the Serial Monitor.	Approximately 20C (room temperature)	21C	Pass
2	Breathe over the sensor repeatedly for 10 seconds, then record the temperature.	The temperature reading should rise.	The temperature rose from 21C to 25C.	Pass
3	Stop breathing over the sensor and record the temperature after an additional 20 seconds.	The temperature should fall back to room temperature.	The temperature initially continued to rise (due to delay in sensor response) before falling back down to 21C.	Pass

Result PASS

Test Case Description	DHT Sensor - Humidity
Tested By	Richard Seaman
Test Date	07/06/2018

Prerequisites

1	The sensor must be wired up to the Arduino (as per the final circuit design).
2	The WSN must be powered on.
3	The preliminary Arduino script must be used to print out the DHT sensor readings within the Serial Monitor of the Arduino IDE.
4	The preliminary script must be running on the Arduino.

Test Conditions

Step #	Step Details	Expected Results	Actual Results	Pass / Fail
1	Record the initial humidity reading from the Serial Monitor.	Approximately 50% (room condition on a warm but not particularly humid day)	54%	Pass
2	Breathe over the sensor repeatedly for 10 seconds, then record the temperature.	The humidity reading should rise.	The temperature rose from 54% to 73%.	Pass
3	Stop breathing over the sensor and record the temperature after an additional 20 seconds.	The humidity should fall back to approximatley what it was in step 1.	The humidity fell back down to 55%.	Pass

Result PASS

6.1.2. **WSN to BS Communication**

The wireless sensor node (WSN) and base station (BS) communicate via their respective Xbee devices. There are various LEDs included within the WSN and the Xbee interfaces which were used as indicators for the communication process (see Table 3). The data received by the BS was also checked to ensure that it matched the data transmitted by the WSN.

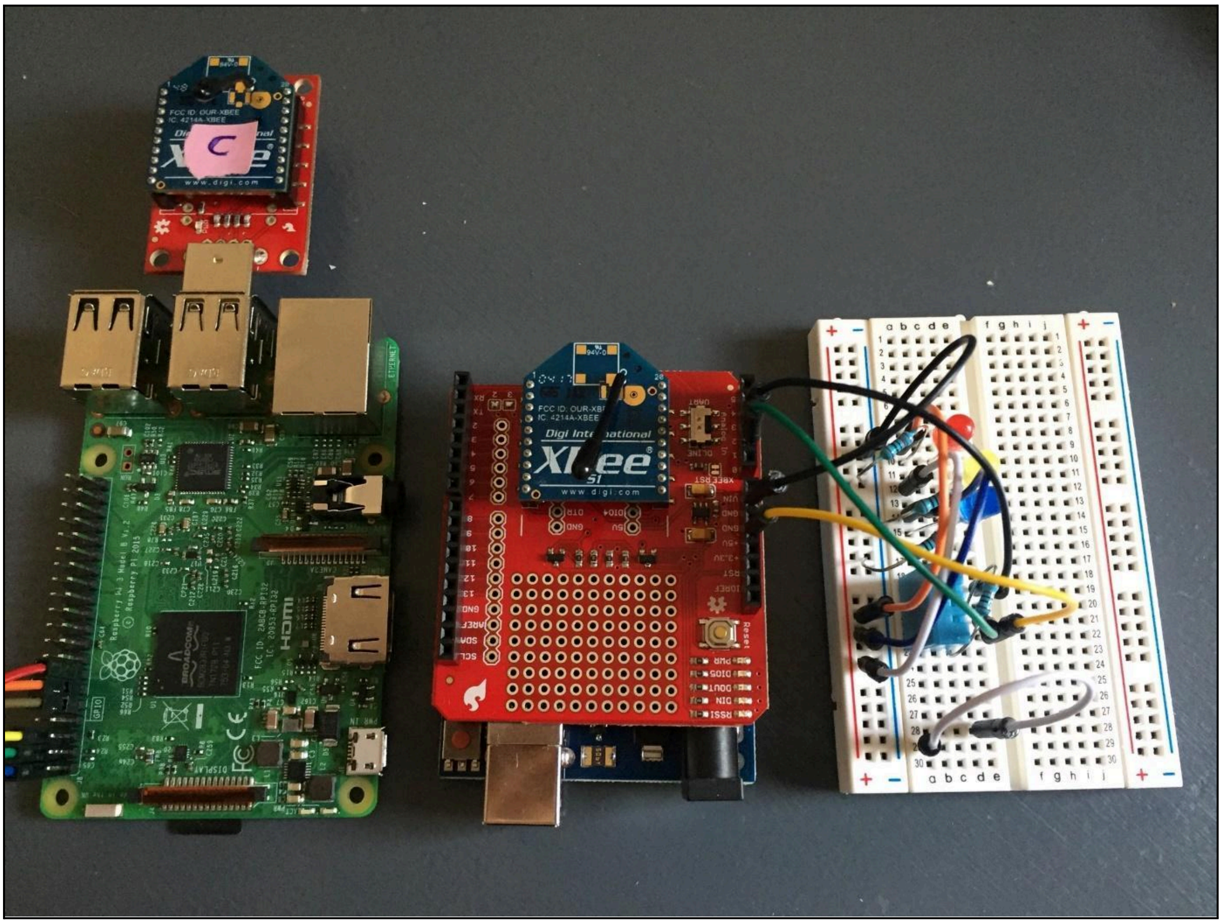

Figure 27 - Base Station and Wireless Sensor Node

Table 3 - XBee Shield Indicator LEDs (Sparkfun, 2018)

LED Label	LED Color	XBee Pin Connection	Default Operation Notes
PWR	Red	3.3V	Indicates power is present.
DIO5	Green	Associate/DIO5	Associated indicator -- blinks when the XBee is associated with another XBee.
DOUT	Red	DOUT	Indicates wireless data is being received.
DIN	Green	DIN	Indicates wireless data is being transmitted.
RSSI	Green	PWM0/RSSI	Indicates relative signal strength (RSSI) of last received transmission.

Test Case Description WSN to Base Station Communication.
Tested By Richard Seaman
Test Date 05/07/2018

Prerequisites

1	The WSN circuit must be complete with the final WSN sketch running on the Arduino.
2	The base station must be complete and the final python program must be running on the RPi.
3	The two Xbee devices must be configured accordingly within the same network.
4	The Serial Monitor of the Arduino IDE and the console log of the python program must be visible.

Test Conditions

Step #	Step Details	Expected Results	Actual Results	Pass / Fail
1	Check that the Xbee end device (WSN) is connected to the Xbee coordinator (base station)	The DIO5 LED on the WSN's Xbee shield should be blinking.	The DIO5 LED was blinking green.	Pass
2	Wait for a transmission and record the sensor readings being transmitted from the WSN (from the Serial Monitor).	Two integer values corresponding to the temperature and humidity readings from the DHT sensor.	23 and 57	Pass
3	Ensure the data was transmitted successfully from the WSN.	The DIN LED on the WSN's Xbee shield should light up. The blue status LED on the WSN should also light up.	The DIN LED and the blue status LED both lit up	Pass
4	Ensure the data was received successfully by the Base Station.	The RX LED should light up on the Base Station's Xbee explorer. The RSSI LED should light up strongly, indicating a good signal.	The RX LED lit up. The RSSI LED lit up strongly.	Pass
5	Ensure an acknowledgement was received by the WSN.	The TX LED should light on the Base Station's Xbee explorer. The DOUT and RSSI LEDs on the WSN's Xbee shield should light up. The green LED on the WSN should light up.	All of the expected LEDs lit up.	Pass
6	Ensure the transmission payload (sensor readings) were interpreted correctly by the Base Station.	The base station's interpretation of the transmission payload should be the same as the sensor readings recorded in step 2.	The base station interpreted the payload as 23 and 57.	Pass

Result PASS

6.1.3. **RPi Socket Control**

A third-party library is utilised for issuing the various socket on/off commands. The socket codes discovered during the sniffing exercise were tested to ensure each of the sockets responded as expected.

Figure 28 - Sockets Plugged in for Testing

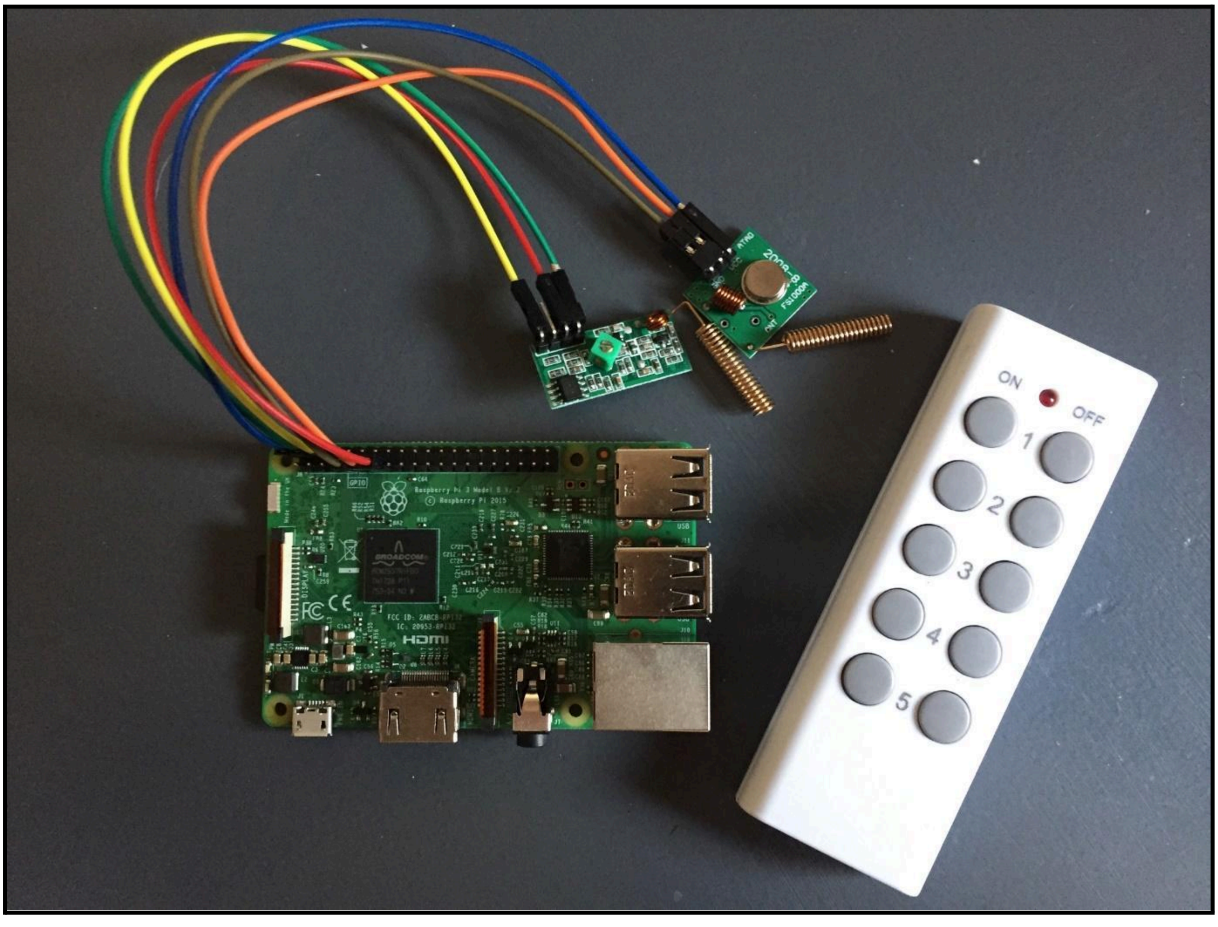

Figure 29 - RPi with 433MHz Transmitter & Receiver

```
pi@richardsrpi:~/433Utils/RPi_utils $ sudo ./codesend 4308437
sending code[4308437]
pi@richardsrpi:~/433Utils/RPi_utils $ sudo ./codesend 4308445
sending code[4308445]
pi@richardsrpi:~/433Utils/RPi_utils $ sudo ./codesend 4308436
sending code[4308436]
pi@richardsrpi:~/433Utils/RPi_utils $ sudo ./codesend 4308434
sending code[4308434]
pi@richardsrpi:~/433Utils/RPi_utils $
```

Figure 30 - Issuing Socket Commands Using 433 Utils Library

Test Case Description	RPi - Socket Control
Tested By	Richard Seaman
Test Date	14/06/2018

Prerequisites

1	Socket control codes must be known.
2	Sockets must be plugged in and visible.
3	433Utils library installed on RPi.
4	Terminal window open at the 433 Utils folder.

Test Conditions

Step #	Step Details	Expected Results	Actual Results	Pass / Fail
1	Manually switch socket 1 on	Socket 1 on	Socket 1 on	Pass
2	*sudo ./codesend 4308436*	Socket 1 off	Socket 1 off	Pass
3	*sudo ./codesend 4308444*	Socket 1 on	Socket 1 on	Pass
4	Manually switch socket 2 on	Socket 2 on	Socket 2 on	Pass
5	*sudo ./codesend 4308434*	Socket 2 off	Socket 2 off	Pass
6	*sudo ./codesend 4308442*	Socket 2 on	Socket 2 on	Pass
7	Manually switch socket 3 on	Socket 3 on	Socket 3 on	Pass
8	*sudo ./codesend 4308433*	Socket 3 off	Socket 3 off	Pass
9	*sudo ./codesend 4308441*	Socket 3 on	Socket 3 on	Pass
10	Manually switch socket 4 on	Socket 4 on	Socket 4 on	Pass
11	*sudo ./codesend 4308437*	Socket 4 off	Socket 4 off	Pass
12	*sudo ./codesend 4308445*	Socket 4 on	Socket 4 on	Pass
13	Manually switch socket 5 on	Socket 5 on	Socket 5 on	Pass
14	*sudo ./codesend 4308435*	Socket 5 off	Socket 5 off	Pass
15	*sudo ./codesend 4308443*	Socket 5 on	Socket 5 on	Pass

Result PASS

6.2. BLACK BOX TESTING

A number of black box tests were carried out to ensure that the system functioned as a whole and that the UI provided the following two features:

1. An ability to control (on/off) each of the wall sockets
2. An ability to view all monitored data.

6.2.1. UI - Socket Control

The iOS application was installed on an iPhone 6. Each of the sockets were tested in turn using the sliders provided on the "*Sockets*" tab in the application.

Test Case Description	UI - Socket Control
Tested By	Richard Seaman
Test Date	29/07/2018

Prerequisites

1	Sockets must be plugged in and visible.
2	Base Station must be powered on and running the required script.
3	App must be installed on device (iPhone).
4	Base Station and Device must be connected to the internet.

Test Conditions

Step #	Step Details	Expected Results	Actual Results	Pass / Fail
1	Open the app on the device and sign in using the following details: Email = rseamanrpi@gmail.com PW = RichardSeamanRPI321!	Sign in success.	Successfully signed in.	Pass
2	Navigate to the Sockets tab.	The sockets tab and information on each of the sockets and the last sync time should be displayed.	All information displayed as expected.	Pass
3	Change the status of socket 1 using the slider, then check to see if the actual physical socket changed staus (on/off)	The slider should change position on the UI. After a few moments, the physical socket should change to match the status of the slider (on/off).	The status of socket 1 was changed from on to off. After a brief moment, the actual socket 1 (bedroom) turned off, which turned off the light which was plugged into it.	Pass
4	Revert the status change made in the previous step.	The slider should change position on the UI. After a few moments, the physical socket should change to match the status of the slider (on/off).	The status of socket 1 was changed from off to on After a brief moment, the actual socket 1 (bedroom) turned on, which turned on the light which was plugged into it.	Pass
5	Repeat steps 3 & 4 for each of the remaining sockets	As above.	As above, for each of the remaining sockets.	Pass

Result PASS

6.2.2. **UI – Sensors**

The iOS application was installed on an iPhone 6. The "*Sensors*" tab was then tested. The correct sensor was selected and the data gathered from this sensor was viewed over the previous number of days. Precise times were checked to ensure that the correct values were given.

Figure 31 - Sensors Tab & Data

Test Case Description	UI - Sensors
Tested By	Richard Seaman
Test Date	29/07/2018

Prerequisites

1	App must be installed on device (iPhone).
2	Device must be connected to the internet.
3	There must be some historic sensor data in the Firebase database.
4	

Test Conditions

Step #	Step Details	Expected Results	Actual Results	Pass / Fail
1	Open the app on the device and sign in using the following details: Email = rseamanrpi@gmail.com PW = RichardSeamanRPI321!	Sign in success.	Successfully signed in.	Pass
2	Navigate to the Sensors tab.	A list of available sensors should be displayed. For this test, a single "DHT11" sensor is expected.	A list of the available sensors was displayed, with only one sensor (DHT11) included.	Pass
3	Select the DHT11 sensor.	A detailed screen should appear with the latest data for the selected sensor shown.	The data for the current date was shown.	Pass
4	Ensure both temperature and humidty are available for selection. Cycle between the two by tapping on them.	There should be two selectors at the top of the screen, one for humidity and one for temperature. Selecting one should update the data displayed and the y-axis units.	Cycled back and forth between humidity and temperature by tapping the selector. The UI updated as expected.	Pass
5	Tap anywhere on the graph to display the value for that time.	A vertical line with a label above it should appear at the selected time which shows the value at that time.	Multiple times selected. The vertical line and label followed the selection and updated the value accordingly.	Pass
6	Move back and forth in time using the arrows at the top of the screen.	The graph should update with data for the selected date, provided there is data in the database for that date. The next arrow should be disabled if the current date is selected.	Multiple dates selected. The graph updated to show data for the relevant date. Next arrow was disabled when the current date was selected. When a previous date with no data available was selected, the graph was empty.	Pass

Result PASS

7. Evaluation & CONCLUSION

As outlined in the introduction of this report, the project objectives were as follows:

1. Wirelessly monitoring an environmental condition within my house
2. Controlling a number of wall sockets within my home (on/off)
3. Providing a user interface to allow me to view the monitored data and control the sockets.

Each of these three objectives were successfully achieved.

The wireless sensor node is used to wirelessly monitor the temperature and humidity conditions within the office of my home. This sensor can easily be moved to another location or room to record the environmental conditions there.

The base station is successfully able to control a number of wall sockets within the home by issuing 433MHz RF signals with the corresponding on/off codes. Each of the five sockets included can be turned on or off by the base station.

The iOS app achieves the third and final objective of providing a user interface to view the monitored data and control the sockets. The app includes two tabs. The first tab is for controlling the sockets and the second tab is for viewing the monitored data.

Each of the tests that were carried out on the system passed. In addition, at the time of writing this report, the system had been live for approximately one month and had successfully gathered data without error. Not only is this evident when viewing historic data within the app, but it can also be seen by viewing all of the recorded sensor readings in the database json file included with this submission.

For the above reasons, one can conclude that the overall project was a success.

8. FURTHER DEVELOPMENT

Although the overall project was a success, there's certainly room for further development.

As mentioned within the Introduction of this document, the scope of this project was limited and was to establish proof of concepts for two broad objectives:

1. Monitoring
2. Control

Over time, the system may evolve to include additional monitored points or additional systems for control.

By using an Arduino within the wireless sensor node, the only limitation to what can be sensed is the number of pins available, of which there are plenty of both digital and analogue available. Additionally, entirely new wireless sensor nodes can easily be included due to the use of XBee devices which establish a ZigBee network. The XBee device on the base station acts as the coordinator over any routers or end devices included. It's just a case of configuring any additional XBee devices accordingly.

The exact same architecture could be used for a wireless node which controls something rather than senses something. By simply including an actuator or relay (or whatever device is required), instructions could be wirelessly transmitted from the base station for the Ardunio in the wireless node to implement. The implication of this would be that the wireless node would have to be awake and ready to receive instructions, so there would be a power consumption implication. The increased power requirements could be minimised however by various means (such as synchronising the transmission of instructions to particular times etc.).

Additional radio-controlled wall sockets could easily be included too.

Rather than hardcoding the socket on/off codes in the base station's program, it would be better to include this information within the *sensorMeta* node of the database. The program could then utilise arrays and automatically parse the information available in order to control any number of sockets. This would make adding or removing sockets much easier in the future.

9. Bibliography

Carr, D. & Zemouri, S., 2018. *Sensors.* Dublin: National College of Ireland.

DIGI, 2016. *Wireless Connectivity Kit Getting Started Guide.* [Online]
Available at:
https://www.digi.com/resources/documentation/digidocs/pdfs/90001456-13.pdf#page131
[Accessed 03 08 2018].

DIGI, 2018. *XBee/XBee-PRO S1 802.15.4 (Legacy).* [Online]
Available at:
https://www.digi.com/resources/documentation/digidocs/pdfs/90000982.pdf
[Accessed 03 08 2018].

gpbl, 2018. *SwiftChart.* [Online]
Available at: https://github.com/gpbl/SwiftChart
[Accessed 03 08 2018].

Ninja Blocks, 2018. *433 Utils.* [Online]
Available at: https://github.com/ninjablocks/433Utils
[Accessed 28 07 2018].

niolabs, 2018. *python-xbee.* [Online]
Available at: https://github.com/niolabs/python-xbee
[Accessed 03 08 2018].

Rapp, A., 2016. *xbee-arduino.* [Online]
Available at: https://github.com/andrewrapp/xbee-arduino
[Accessed 03 08 2018].

Sparkfun, 2018. *XBee Shield Hookup Guide.* [Online]
Available at: https://learn.sparkfun.com/tutorials/xbee-shield-hookup-guide
[Accessed 03 08 2018].

10. **APPENDIX**

The documents listed below were included with this report in the zipped project submission folder. The file names/folders are numbered as they appear in the following list:

#	File Name	Type	Description
1	*Project Report Proposal*	PDF	The initially submitted document
2	*Requirements Specification*	PDF	The initially submitted document
3	*Project Analysis & Design*	PDF	The initially submitted document
4	*Testing*	PDF	The initially submitted document
4a	*Testing*	XLSX	The excel workbook with each of the test cases.
5	*Project Planner*	PDF	The project management Gantt Chart complete until week 11.
6	*Project Log*	PDF	The initially submitted document.
7	*Github Clone – WSN*	Folder	A clone of the wireless sensor node's Github repository
8	*Github Clone – Base Station*	Folder	A clone of the base station's Github repository
9	*Github Clone – iOS App*	Folder	A clone of the iOS app's Github repository
10	*Database Export*	JSON	The Firebase real-time database exported as a JSON file. Note that this file includes all of the gathered data (at the time of export).
11	*Base Station Log Files*	Folder	Some of the archived log files produced by the *Main.py* program running on the base station.

11. GITHUB REPOSITORIES

The Github repository links for this project are provided below.

The sketch developed for the Wireless Sensor Node's Arduino is available at the Github repository below:

https://github.com/Richard-Seaman/FinalProject-Arduino

The python program developed for the Base Station is available at the Github repository below:

https://github.com/Richard-Seaman/FinalProject-Rpi

The source code for the iOS application is available at the Github repository below:

https://github.com/Richard-Seaman/FinalProject-IOS