A report submitted in partial fulfilment of the regulations governing the award of the Degree of BSc (Honours) Computer Science at the University of Northumbria at Newcastle

Project Report

Internet of Things (IoT)

Wei Guang Heng

2016/2017

Software Engineering Project

Acknowledgement

Northumbria University for using their resources.

David Kendall for being my project supervisor.

Jungong Han for being my second marker.

DECLARATIONS

I declare the following:

(1) that the material contained in this dissertation is the end result of my own work and that due acknowledgement has been given in the bibliography and references to **ALL** sources be they printed, electronic or personal.

(2) the Word Count of this Dissertation is 8761

(3) that unless this dissertation has been confirmed as confidential, I agree to an entire electronic copy or sections of the dissertation to being placed on the eLearning Portal (Blackboard), if deemed appropriate, to allow future students the opportunity to see examples of past dissertations. I understand that if displayed on eLearning Portal it would be made available for no longer than five years and that students would be able to print off copies or download.

(4) I agree to my dissertation being submitted to a plagiarism detection service, where it will be stored in a database and compared against work submitted from this or any other School or from other institutions using the service.

In the event of the service detecting a high degree of similarity between content within the service this will be reported back to my supervisor and second marker, who may decide to undertake further investigation that may ultimately lead to disciplinary actions, should instances of plagiarism be detected.

(5) I have read the Northumbria University/Engineering and Environment Policy Statement on Ethics in Research and Consultancy and I confirm that ethical issues have been considered, evaluated and appropriately addressed in this research.

SIGNED:

Abstract

The project is to develop and implement an end to end solution using the mbed IoT platform documenting the features and challenges face during the process.

A client application is developed to run on an ARM mbed enabled board and its features are to push sensors values to the mbed device connector and to process any request that may be coming from the it as well.

A web application consisting of a backend server and a front end client is also developed to pull and send requests to the mbed device connector. The back end of the web application will act as a middle man in between of the device connector and the front end client. It will be responsible for pulling data that have been published to the device connector and pushing the new data over to the front end client and at the same time listen for any requests that may be made from the front end client. The front end client will mainly be displaying data that have been pushed from the back end and sends as request made by the user to the back end.

The analysis chapter researched about the mbed IoT platform which influenced how the synthesis chapter will be structured and discuss about the process of the application development. And the evaluation will access the development issues and challenges.

Overall this produce a project that has been researched, developed and tested to help other developers to have a better understanding of the platform.

Table of contents

1. Introduction

The project is to explore and implement features that are offered by the mbed IoT platform from the hardware to software and cloud solutions. At the end of this project there should be a better understanding on how the whole platforms integrate together.

1.1. Project aims

- To build a device that runs on the mbed OS and able to communicate through the internet via ARM clouds services from the web or mobile device.

1.2. Project objectives

- To investigate the use of features that are usually provided by a RTOS and how mbed OS implements them.
- To build a backend service on the ARM cloud platform to remotely retrieve or control the sensors and actuators.
- To build a web or mobile client that communicates with the ARM cloud services.

The report will include a literature review on the area of Internet of Things and look into the different platforms that are available currently and how they provide a platform that developers can be used to develop their ideas. The report will also include a requirement specifications in which the project will achieve and define the scope of the project.

1.3. Project main features

The main features of this project is to have an end to end solution base on the mbed IoT platform and to document the features and the process of developing on the platform. As the popularity of IoT continue to rise, there are a few prototyping platforms available to developers. All this development platforms aim to have a easy and lost cost way for business to develop their hardware solutions without compromising the security threats that is currently being expose to such devices at this point. At the end of this project there will be a web application that a user can use to control the LEDs on the board and to read its sensor values over the internet.

1.4. Tools and techniques

The project will utilise the C programming language to build the client application, a widely use programming language for embedded system. The back end of the web application will be built using the NodeJS framework and the front end will be built using the standard web technologies HTML, CSS and javascript. Most of the work will mainly be done using the mbed online compiler and a web text editor like Atom to write the web application.

2. Analysis

2.1. Literature Review

This literature review will discuss the concepts of Internet of Things, the applications and issues with them and possible solutions that the industry is proposing.

2.1.1. The rise of Internet of Things (IoT)

As the internet is becoming more widely available and more devices are equipped with Wi-Fi and cellular radios, it is getting more than easy to get connected to the internet. This has created a new category of devices as we know now as the Internet of Things (IoT). The term Internet of Things have been used differently but according to Morgan (2014) Internet of Things is a concept of connecting any device with an on and off switch to the Internet. These devices can exists in any form and there are many reasons fuelling the growth of such devices. Currently one of the most important feature of IoT devices is data collection and it have been widely used to collect informations on the weather, medical informations and animal tracking. Such information are useful in predicting weather patterns, helping doctors make better diagnoses and studying on animals migration patterns. IoT has also contribute to the growth of smart appliances at home. This ranges from coffee machine, refrigerator, web cams and thermostats. This has allow home owners to monitor the situation at home through their smart phones, from seeing who is at home to switching off the light if the owner left it on when leaving home in a hurry.

Other than waiting for hardware manufactures to release products like an internet connected light which can be controlled remotely, the Raspberry Pi, a low-cost general purpose computer made by the Raspberry Pi Foundation has encouraged many people to start building their own kind of devices that are able to interact with the physical world. The Raspberry Pi has created a new community of developers and hobbyist alike and many of them have been working on their own version of an internet connected home. To date The Raspberry Pi Foundation has sold 12.5 millions Raspberry Pi in five years and there does not seems to be a trend to slow down. (The MagPi Magazine, 2017) and according to Forbes (2016) the number of IoT devices will continue to grow to 30.7 billion in 2020.

2.1.2. Security concerns

However a recent attack in October 2016 cause some major websites like Twitter, Spotify and Reddit to be taken offline. Security analysts believes that the attack was made by exploiting the loopholes found in IoT devices from web connected homes to launch the attack. (BBC News, 2016). One of the reason why such attack is possible is due to weak implementation of the software running those devices and there is no mechanism to patch the exploit. This has also lead to huge privacy concerns. A hacker will also be able to monitor or listen in to a house through an insecure camera. In some cases the hacker will also be able to lock the owners of the house in and demand ransom for release.

2.1.3. Possible solutions

As discussed in previous section, there can be serious consequences when security is taken lightly when designing an internet connected device. However building and designing a secure device is tedious and difficult. Canonical (2017) published a white paper detailing the current situation of the issues and offers their solution. Canonical released an operating system call the Ubuntu Core and they believe they are able provide a better and more secure operating system base on the following eight concepts.

- The OS is designed from the ground up for security.
- The OS is kept simple while maintaining functionality.
- The OS is able to update itself from a centralise mechanism.
- The OS should be able to rollback to the last working state if an update fails.
- Files should only be read only and can only be replaced with a digital authentication.
- Applications should be self contained and sandboxed.
- The OS should feature familiar architecture and known coding methods.
- Security should not restrict the open and innovative nature of the OS.

Another big player offering an IoT platform is Google with Android Things. It is currently in the preview status but Google aims to provide a platform base on their existing products. Android Things is build on top of Google's mobile operating system and Google is committed to keep the operating system up to date and fix any exploits that might be found in the future. Apart from providing an IoT OS for developers, Google has a suite of cloud base solution that allows the IoT device to connect to. Developers will be able to take advantage of Google's cloud infrastructure and scale their applications to demand.

One of the biggest reason that lead to having unsecured devices in the market currently is due to the cost involved to implement the features needed to provide a secure operating system and the infrastructure to maintain them. As discussed, Canonical and Google are providing some kind of solutions to help developers reduce the cost. The next few chapters will discuss and detail the features and process of developing an end to end solution on the ARM mbed IoT platform.

2.2. mbed IoT Platform

The mbed IoT Platform is a platform by ARM that provides an embedded operating system, transport security and cloud services to help create connected embedded solutions. It's aim is to make transporting data from sensor to the server simple and secure.

According to ARM (2017) documentation the platform is made up of two key sets of products, the device software and the cloud base device management system. Figure 1 shows the overall architecture and the communication protocol implemented on the platform. The end to end solution will be made up of three parts. The device software will consists of the mbed OS and the mbed Client library. The mbed Client library helps to provision the ARM mbed enabled board to the mbed connecter and makes the sensors or actuators resource available online. ARM does not provide the infrastructure needed to host the web app, therefore it will be the responsibility of developer to develop a web app and access the board resources through the mbed connector APIs.

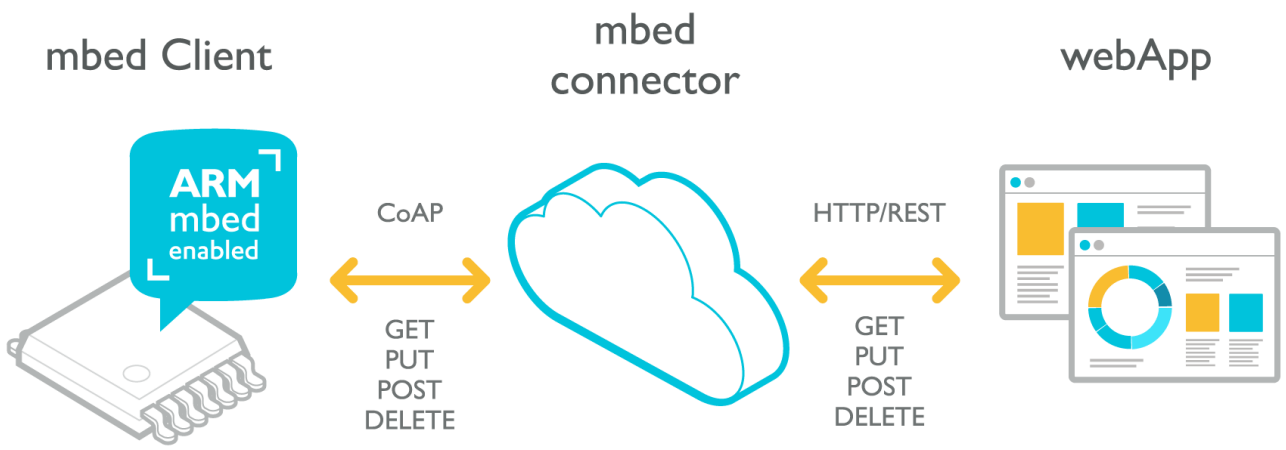


Figure 1 - Overall architecture of the platform (ARM, 2017)

2.3. Tools

ARM provides two set of development tools that are available on their website. They provide a command line tool and a online compiler. Both tools can be used to compile the codes and generate the binary file which can be install on the mbed board. Whenever a mbed board is connected, it will appear on the computer just like how a storage drive would appear. This allows the binary to be easily loaded on to the board using the drag and drop method. Projects can also be exported to supported Integrated Development Environment (IDEs) if the user prefers. This section will briefly introduce the tools available.

2.3.1. mbed CLI

The mbed CLI is a python base tool specifically for mbed OS 5. It enables the full mbed workflow which includes version control, maintaining and updating repositories. A toolchain for the mbed board will be needed to be installed to compile projects and generate the binary which will be loaded on to the board. The tool can also be used to export the program for debugging in supported third party IDEs.

2.3.2. mbed Online compiler

The mbed online compiler available at https://developer.mbed.org/compiler is the easiest and fastest tool to get your code up and running on your mbed board. Repositories can be managed online and the program binary can be generated without installing any tools on your local computer. Internet connection will be needed to be able to develop your program. Figure 2 shows the interface of the mbed online complier. The panel on the left shows all the project that the developer has in the account. The workspace is on the right and the bottom panel is where the compiler messages, search results and notifications are shown. On the toolbar, the user will be able to create a new project, import a sample project or library and work with version control.

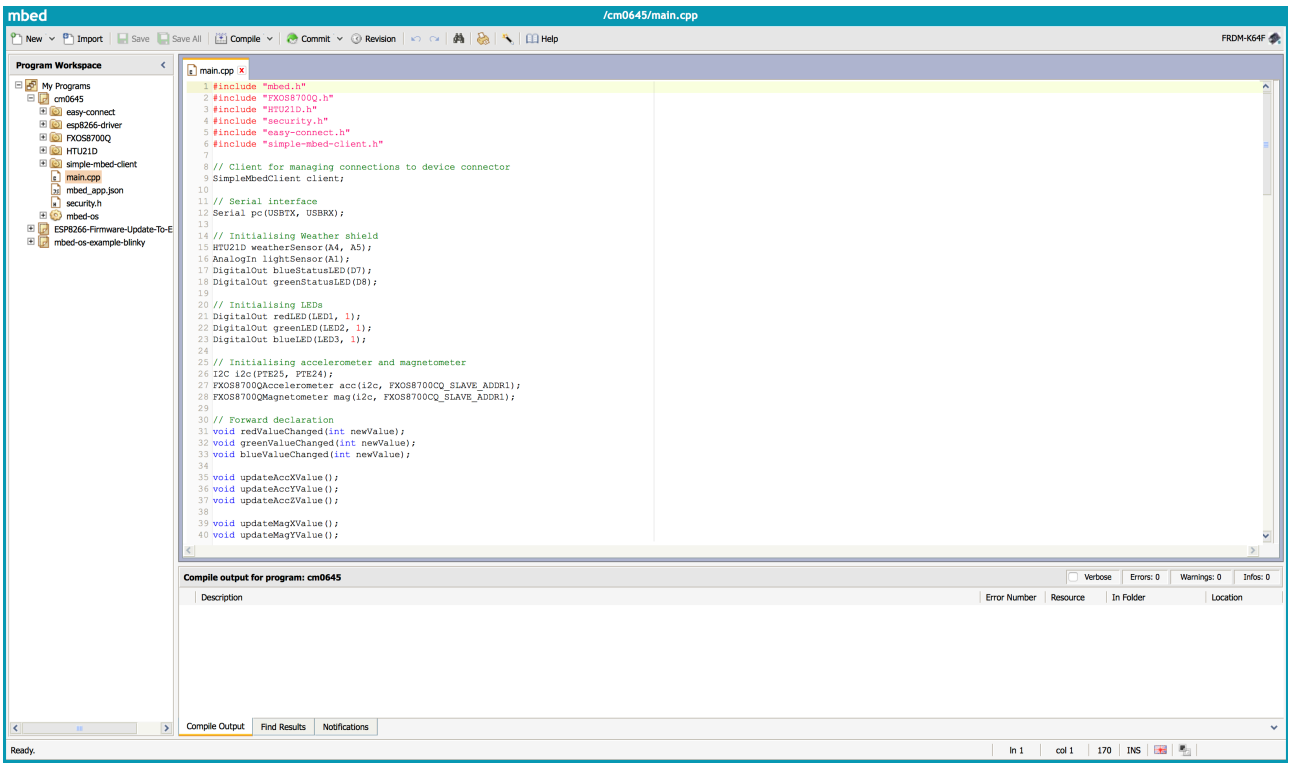


Figure 2 - mbed online compiler interface

2.3.3. Third Party IDEs

Both mbed CLI and mbed online compiler have the option to export the program for debugging in third party IDEs. Using a third party IDE will allow more freedom in how a project is built and optimised but ARM does not guarantee the same consistency as when using the mbed online compiler.

2.4. mbed OS

mbed OS is the main operating system that will be used to develop the client program and it is based on the open-source CMSIS-RTOS RTX. The operating system supports deterministic, multithreaded real time software execution and supports a wide range of ARM Cortex-M based devices. ARM mbed enable boards are designed to allow simple USB drag and drop programming to allow rapid prototyping and ease of use. In terms of hardware and software security, mbed OS uses a supervisory kernel uVisor thats creates an isolated security domains to restrict access to memory and peripherals, this is important as the operating system should not allow the application to have more access than it needs. Having unrestricted access may allow hackers to enter and exploit the hardware features in it or developers may write certain codes which may cause unintended effects. To ensure that the device communicates over the internet securely, its communication APIs uses the SSL and TLS protocol. These protocols ensure that any information that is sent will be encrypted and limit the possibility of a middle in man attack. The next few section will discuss some of the Application Programming Interfaces (APIs) included and their features.

2.4.1. Task management

Task management APIs provided by mbed OS handles the creation and destructions of threads in the system and provides a mechanism for a safe inter thread communication. This section will discuss the different ways of managing them and the classes associated with them.

2.4.1.1. Threads

A thread is a set of instructions that the processor needs to execute in a program and is usually managed by the scheduler of an operating system. Referring to ARM (2017) documentation, the Thread class in mbed OS allows the defining, creating and controlling thread functions in the program. A thread in the system can be running, ready, waiting and inactive. Figure 3 shows how a thread can switch from one state to another. When a thread is in the running state, it is the thread that is being process by the processor at that moment and will be the only thread to be in this state. A ready thread with the highest priority will be the next running thread whenever the running thread before it is terminated or switches to the waiting state. A waiting thread is a thread that is waiting for an event or interrupt to occur before switching back to the ready state. Inactive thread are threads that are not created or terminated, they typically does not consume any system resources.

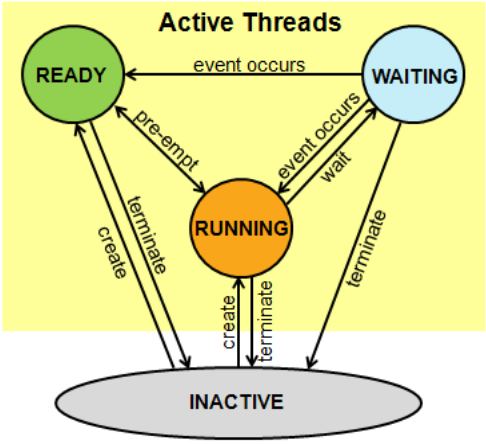


Figure 3 - Threads and their states (ARM, 2017)

The function main is a special thread function that the system initialise and it will have an initial priority of osPriorityNormal. This thread will be the first thread that the RTOS schedules. The code shown in Figure 4 shows how two separated threads are created to blink two different LEDs. The first thread will be created automatically and executes the main function and a second thread will be created explicitly in the main function.

main.c Raw

```c
#include "mbed.h"
// Initialsing the resources
DigitalOut led1(LED1);
DigitalOut led2(LED2);
Thread thread;

// Function that will be excuted on the second thread
void led2_thread() {
  while (true) {
    // Toggle led2 every second
    led2 = !led2;
    wait(1);
  }
}

// The first thread will be created automatically and main() will be executed
int main() {
  // Start executing the second thread
  thread.start(led2_thread);
  // Toggle led1 every 0.5 second
  while (true) {
    led1 = !led1;
    wait(0.5);
  }
}
```

Figure 4 - Blinking two LEDs on a separate thread example

2.4.1.2. Event loop

An event loop is a mechanism provided by mbed OS that allows us to defer the execution of code to a different context. The reason of having such mechanism is because there are certain C functions that runs in an interrupt context which is not safe. The event loop offers to make those functions safer by deferring those codes from the interrupt context to the user context. In addition the event loop can be used anywhere instead of just in an interrupt handler. Consider the following code shown in figure 5. (ARM, 2017)

main.c Raw

```c
#include "mbed.h"
#include "mbed_events.h"

// Declaring LED
DigitalOut led1(LED1);
// Declaring push button
InterruptIn sw(SW2);
// Creating an EventQueue with a size of 32
EventQueue queue(32 * EVENTS_EVENT_SIZE);
Thread t;

void rise_handler(void) {
    printf("rise_handler in context %p\r\n", Thread::gettid());
    // Toggle LED
    led1 = !led1;
}

void fall_handler(void) {
    printf("fall_handler in context %p\r\n", Thread::gettid());
    // Toggle LED
    led1 = !led1;
}

int main() {
    // Start the event queue
    t.start(callback(&queue, &EventQueue::dispatch_forever));
    printf("Starting in context %p\r\n", Thread::gettid());
    // The 'rise' handler will execute in IRQ context
    sw.rise(rise_handler);
    // The 'fall' handler will execute in the context of thread 't'
    sw.fall(queue.event(fall_handler));
}
```

Figure 5 - Thread example

As shown in figure 5, the rise_handler function will run in an interrupt context when the button is push and the fall_handler will run in a user context when the button is released. However there is a problem with this implementation. The printf in rise_handler is potentially unsafe when called on the interrupt context. As mentioned in the previous paragraph, the event queue can be used in this case to move the handler to run in the user context instead of the interrupt context.

ARM (2017) recommends the following changes to the code.

Replace:
sw.rise(rise_handler);

With:
sw.rise(queue.event(rise_handler));

The printf in rise_handler is now safe as its running in a user context. But this implementation introduces another problem. The rise_handler function is now queued and no longer runs immediately. In cases where the code has to execute immediately, we need another solution. The solution documented by ARM (2017) is to split the rise_handler function into two parts, a time critical portion which will run in the interrupt context and a non critical portion will run in the user context. Figure 6 shows how the function can be divided into critical and non critical sections.

main.c Raw

```c
void rise_handler_user_context(void) {
    // Non critical and unsafe code will run here
    printf("rise_handler_user_context in context %p\r\n", Thread::gettid());
}

void rise_handler(void) {
    // Execute critical portion here
    led1 = !led1; // Example toggle LED immediately

    // Use the 'queue.call' function to add an event (the call to 'rise_handler_user_context') to the queue.
    queue.call(rise_handler_user_context);
}
```

Figure 6 - Separating into critical and non critical parts

Codes that need to be executed immediately will now be in rise_handler function on the interrupt context and those codes which are not critical and is unsafe to run on the interrupt context will now be added into the rise_handler_user_context function. This design pattern of having a time critical and non time critical portion can be easily implemented with event queue. Other than queuing non interrupt safe code, any other codes can also be used with an event queue to queue and defer for later execution.

2.4.2. Input and output

Being able to read information from sensors and responding through actuators are important in an IoT application. For example, a water pump should stop pumping water when a water level sensor detects that the water tank is full. mbed OS provides a few APIs that is able to read or write analog and digital signals. An analog signal can be a value that is coming off from a potentiometer, the AnalogIn API can be used in this case to measure the voltage. The DigitalOut API can be used to switch on or off a LED just by sending a digital value of 1 or 0. Figure 7 demonstrate how we can use a potentiometer to turn on or off an LED. (ARM, 2017)

main.c Raw

```c
#include "mbed.h"

// Initialising an analog input e.g potentiometer
AnalogIn ain(A0);
// Initialising a digital ouput e.g LED
DigitalOut dout(LED1);

int main(void) {
    while (1) {
        // Read the voltage on the potentiometer
        // If the value is greater than 0.3 * VCC turn on the LED
        // Else turn off the LED
        if(ain > 0.3f) {
            dout = 1;
        } else {
            dout = 0;
        }
        wait(0.5);
    }
}
```

Figure 7 - Turning on or off and LED base on potentiometer value

2.4.3. Digital interfaces

In a more complex system, a sensor may not just sends its value in a form of voltage like the potentiometer. One of such example is having multiple sensors connected to a single module and the module will transmit the data it receives in a single chunk of data using a particular protocol. mbed OS supports four different protocols and they are serial, SPI, I2C and CAN. These protocols allows a larger and more complex information to be send between different devices. The decision as to which protocol to use will depend on the physical design of the communication hardware, if the transfer type is synchronous or asynchronous and the number of peripherals needed to support. Figure 8 shows the basic attribute of the serial protocol on Google (2017) Android Things webpage. These serial protocol requires a minimum of two wires, one for transferring (TX) and one for receiving (RX) data.

Protocol	Transfer Type	# of Wires	# of Peripherals	Transfer Speed
I2C	Synchronous	2	Up to 127	Low
SPI	Synchronous	4+	Unlimited	High
UART	Asynchronous	2 or 4	1	Medium

Figure 8 - Basic attribute of different serial protocol (Google, 2017)

2.4.4. Communications

An internet connection will be needed to enable the board to be connected to the mbed Device Connector. There are quite a few communication interfaces that mbed OS supports. The Ethernet and Wi-Fi interfaces are the most commonly used. The nature of this project requires the flexibility to collect data at any location and using the Wi-Fi interface would be the most appropriate. The WiFiInterface API provides abstraction needed to connect to any Wi-Fi hotspot. The API can be extended to supported different Wi-Fi module that are available. The use of this API will be further discussed in the development section found in the synthesis.

2.5. mbed client

mbed client is a library included in mbed OS to connect devices to the embedded devices to the Device Connector Service. A Linux version is also available for IoT devices running on Linux operating system. The library can be used to manage devices on the mbed Device Server, communicate with internet services securely and controlling the endpoint and application logic. The development section of the synthesis will discuss the implementation and uses of this library on mbed OS.

2.6. mbed Device Connector

The mbed Device Connector is a cloud service provided by ARM available at https://connector.mbed.com allows developers to connect their IoT devices without having to worry about building any infrastructure. The service is built with security, simplicity and the ability to scale in mind reducing the barrier to adoption. The service uses the Constrained Application Protocol

(CoAP). CoAP is a specialised industry standard mainly to be used in devices with constrained nodes and networks. Figure 9 shows the architecture of the device connector and Figure 10 shows the dashboard that the user have access to.

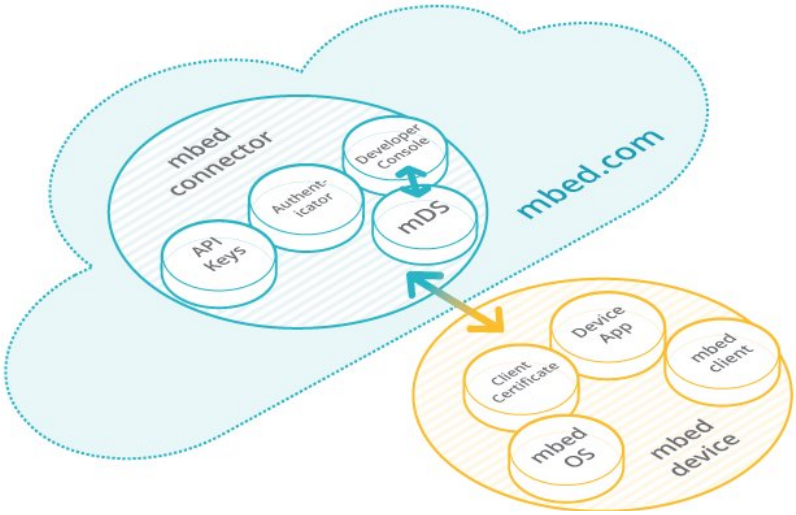


Figure 9 - Different components on mbed.com and the mbed device (ARM, 2017)

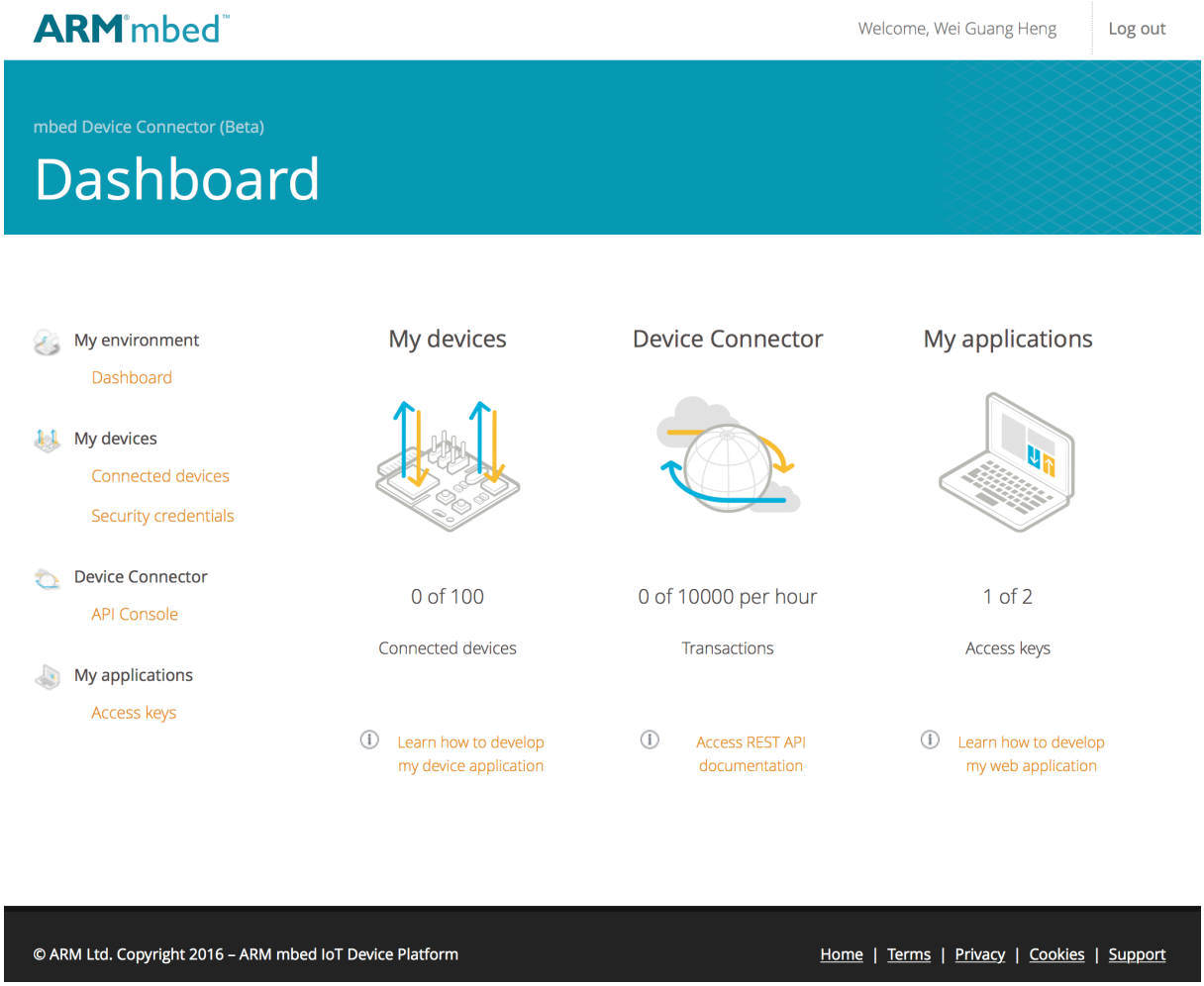


Figure 10 - Device cloud connector dashboard

2.7. Requirements specifications

With an overview of the whole mbed IoT platform, there is a clearer idea to what can be achieved and implemented and this section will show the functional and non functional requirement that the project will try to achieve.

2.7.1. Functional requirements

1. Create an application uses the mbed OS APIs that runs on a mbed enabled board.
2. The mbed enabled board should has some input and output peripherals of which their values can be uploaded or controlled remotely from the device connector.
3. Create a web app that is able to connect to the device connector and retrieve the mbed board sensor informations or control any output peripherals available.

2.7.2. Non functional requirements

1. Client application should be responsive.
2. Web application should be simple and user friendly.

2.8. Test plan

The following list shows the plan to ensure that the product will be able to meet its aims and objectives which will be evaluated at the final stage of the report.

2.8.1. mbed client

1. Must able to connect to the internet using the chosen network interface.
2. Must be able to authenticate itself to device connector and publish some data to it.
3. A request can be made on the device connector dashboard to get data from a particular sensor or to control a LED that is on the board.

2.8.2. Web application

1. Must be able to connect to the device connector and to check for any devices that is connected.
2. Must be able to retrieve data from the connected device.
3. Must be able to subscribe to changes on the connected device and display them.
4. Must be able to send a request to the connected device to control the LED.

3. Synthesis

The synthesis will explain the hardware and software components involved and the process of developing the end to end solution using the mbed IoT platform. The implementation is mostly base on the example “Building an internet connected lighting system” provided on ARM (2017) website.

3.1. Freedom-K64F

The Freedom-K64F is an ARM mbed enabled board developed by NXP in collaboration with mbed for prototyping purposes. (ARM, 2017) The board comes with a 6-axis accelerometer and magnetometer, 2 user push buttons and a RGD LED. In additional to the sensors, buttons and LED the board also comes with an ethernet port and its functionality can be further expanded through it’s header pins. The outer row pins are compatible with the Arduino R3 standard which is very useful as there are many R3 shields available in the Arduino ecosystem. Figure 11 shows the header configuration of the Freedom-K64F board.

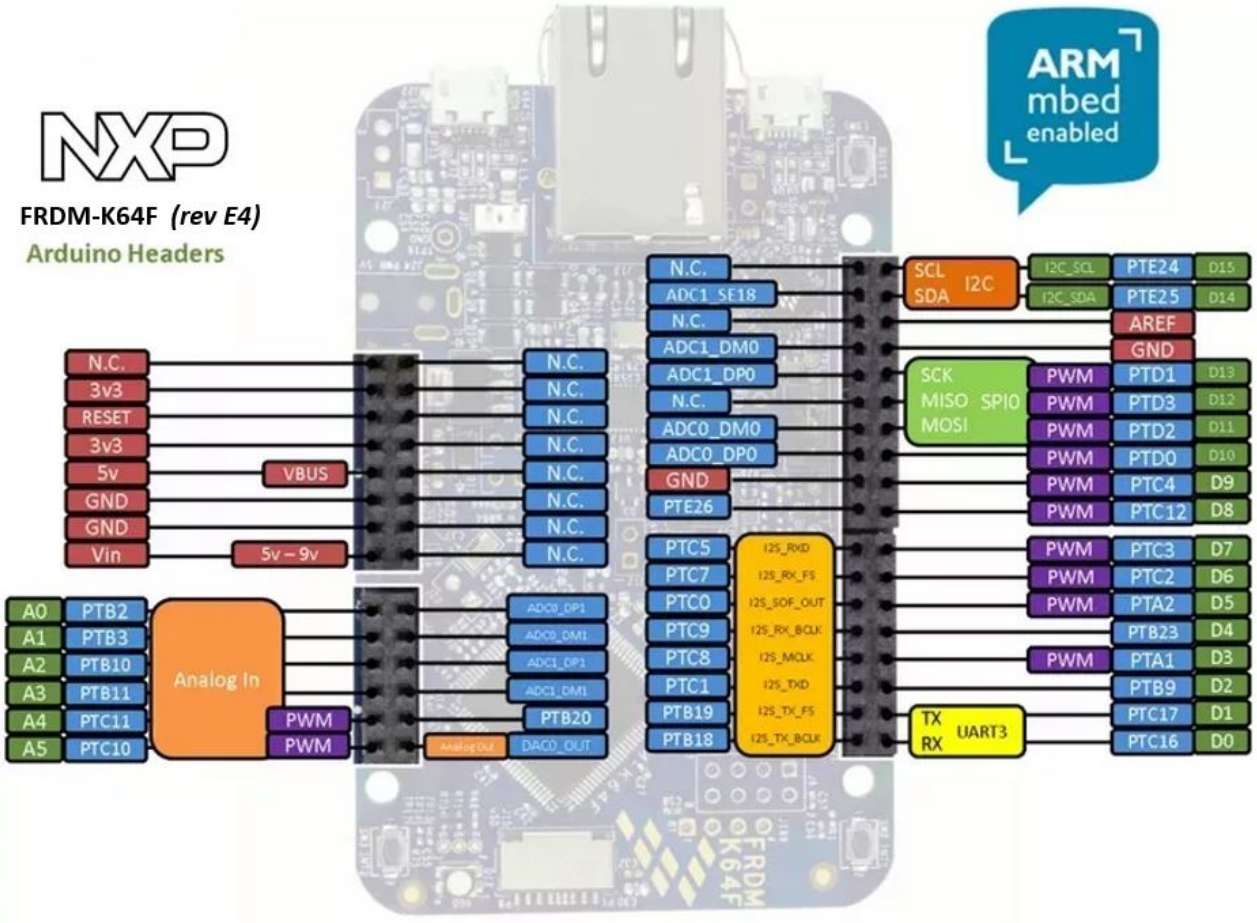


Figure 11 - Freedom-K64F board header pin configurations (ARM, 2017)

3.2. ESP8266

The ESP8266 WiFi module is a self contained SOC with integrated TCP/IP protocol stack, it is capable of hosting an application independently or to be used as a WiFi interface to an existing micro controller (Sparkfun, 2017). More information on the module pins and commands examples are available in the appendices. There are a number of firmwares available and the user is able to flash a new firmware according to their needs. For the purpose of this project, the mbed team (2016) recommends the Espressif firmware and they provide a easy to use tool to flash the firmware onto the module.

3.2.1. Updating firmware to Espressif

A step by step video to update the firmware is available on ARM website. However the guide on ARM's website uses the Seeed Grove Serial WiFi module, a different variant of the ESP8266 module which has the peripherals to connect to the Grove Shield. The Grove shield is a Arduino R3 shield which will then be stacked onto the Freedom-K64F board. The ESP8266 module that was used in this project is the simpler module without the additional peripherals and therefore there is a need to connect it to the Freedom-K64F board manually. According to the Espressif GitHub, Angus Gratton (n.d.) the maintainer of the esptool documented that the GPIO0 pin needs to be grounded for the ESP8266 module to enter serial boot loader for its firmware to be updated. To start flashing our ESP8266 module to the Espressif firmware we would have to wire up the module to the Freedom board differently, Table 1 shows the connections that have to be made.

Table 1

Freedom-K64F	ESP8266
3.3V	4 (Chip enable)
3.3V	8 (3.3V)
GND	1 (GND)
D1 (TX)	7 (RX)
D0 (RX)	2 (TX)
GND	5 (GPIO0)

After the connection is made, flash the program provided by ARM to the Freedom-K64F board and connect the it to a serial terminal. In the program, ensures that the pins for the TX and RX values matches the pins that the module is connected to and the value of the switch that is to be used matches the board button identifier. In this case the D1 is used as the TX pin and D0 as the RX pin. SW2 is the button that is used to trigger the update. Once the Freedom-K64F board boots up the serial terminal will provide the instructions on screen to go through the update process. The process involves pushing a physical button and waiting for the different files to be transferred on to the module.

3.3. Sparkfun weather shield

Sparkfun weather shield is an Arduino shield base on the R3 standard and it utilise HTU21D, MPL3115A2 and ALS-PT19 sensors to provide humidity, barometric pressure and light informations. The values of HTU21D and MPL3115A2 sensors are available through the I2C protocol and the ALS-PT19 value can be read by measuring the voltage that is coming out from its output pin. The schematics of the shield is available in the appendices section.

3.4. Software and libraries

Developing the application from scratch is a very difficult and a tedious process and use of libraries will provide the abstraction of not working directly on the hardware easier. Table 2 shows the libraries that are used and their uses in the application.

Table 2

Libraries	Functions
easy-connect	Provides an easy way to connect to the internet
esp8266-driver	Driver for the ESP8266 wifi module
FXOS8700Q	Driver for the motion sensors on the Freedom-K64 board
HTU21D	Driver for the sensors on the Sparkfun weather board
simple-mbed-client	Provides an abstraction from managing device connector APIs directly
mbed-os	The operating system itself for managing resources on the mbed board

3.5. Development

This section will explain the process of setting up the three hardware components together, developing the client application and including steps on importing the libraries mentioned in the software and libraries section. When the client application is tested and working, the next step is to add in the codes needed to connect the board to the mbed device connector service. And lastly when the data is accessible from the device connector dashboard, the final stage will be to develop a web application that polls data from the device connector and display them in a browser.

3.5.1. Setting up the hardware

As previously discuss, the development board that will be used in this project is the Freedom-K64 with a Sparkfun shield and the ESP8266 wifi module. In addition a few female to male jumper wires and male to male jumper wires are needed. Figure 12 shows the hardware components.

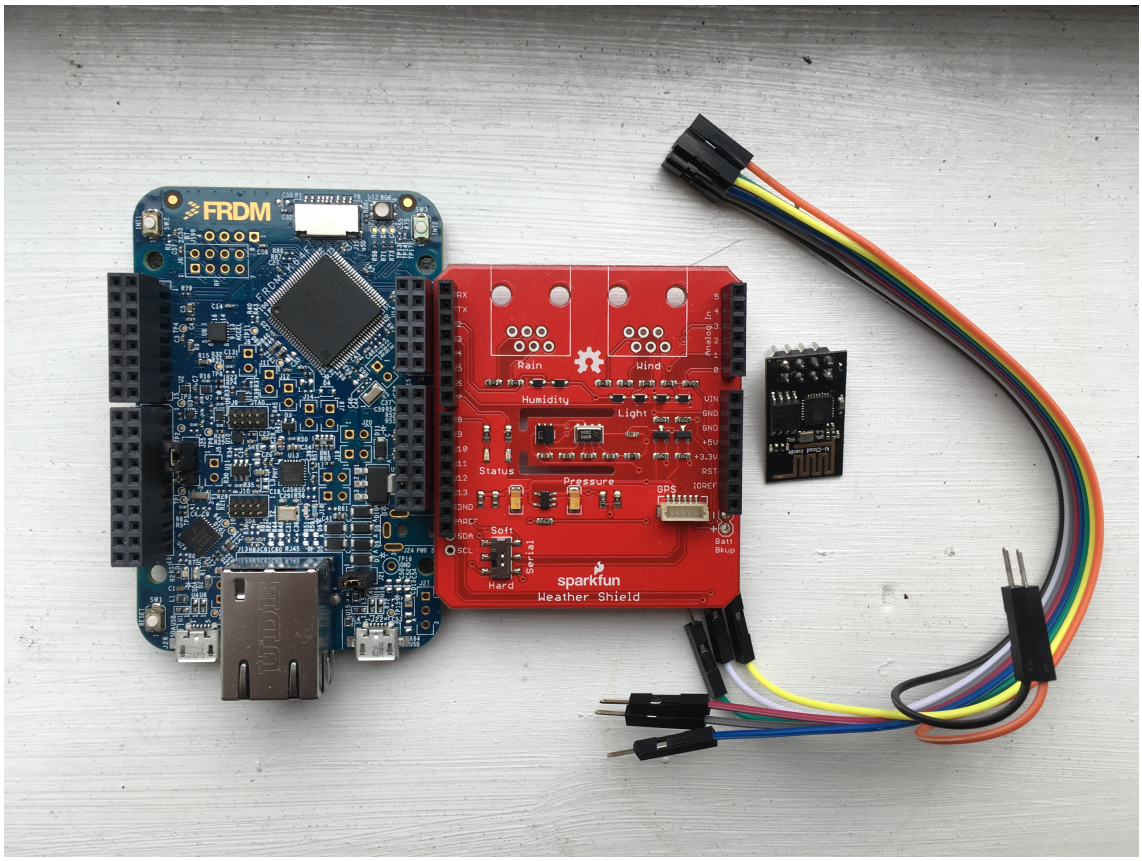


Figure 12 - Components to be used in this project, from left FRDM-K64, Sparkfun weather shield, ESP8266 Wifi module and jumper wires.

Connecting the Sparkfun weather shield to the development board is as easy as stacking it on top to it, as the header pins are compatible to the Arduino R3 standard. However the following connections have to be made from the ESP8266 module to the Sparkfun weather shield as shown in table 3 using the jumper wires.

Table 3

Sparkfun weather shield	ESP8266
3.3V	4 (Chip enable)
3.3V	8 (3.3V)
GND	GND
TX	7 (RX)
RX	2 (TX)

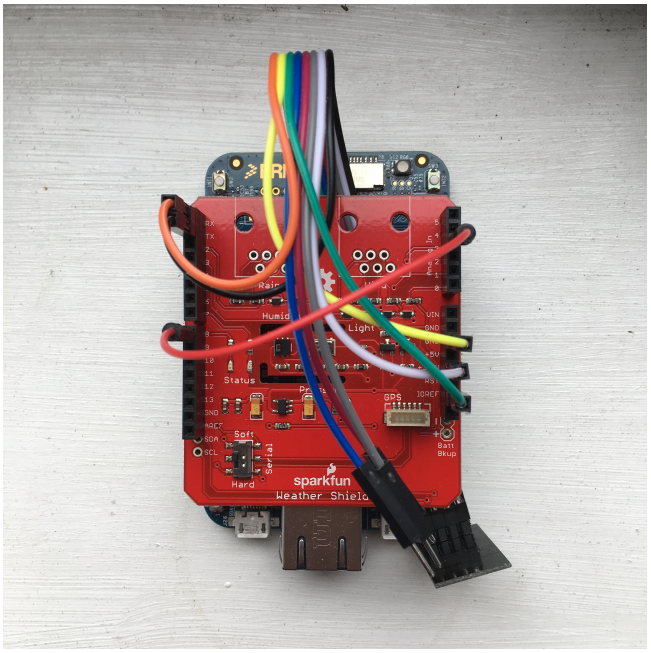


Figure 13 - All components connected together

Figure 13 show how the board will look like with the different components connected. However as shown in figure 13, there is an additional connection from pin 3 to pin 8 on the weather shield. Referring to the schematics of the weather shield there is a need to provide a 3.3V to the light sensor. As other 3.3V pins are connected to the wifi module, there is a need to find another pin to provide that 3.3V. As the result pin D8 which can be set to high is used to provide that 3.3V for the light sensor.

3.5.2. Setting up the client project

Now that the hardware is all wired up, it time to start building the application for the board. However to access the mbed online compiler, an mbed account will be needed and can be created on mbed developer website. Once the account is created, login into the mbed online compiler and the next step is to setup the workspace for the application. To get started create a new project from the top left hand corner. A new pop out should appear, in this case select FRDM-K64 as the platform and empty program for the template and finally give the program a name.

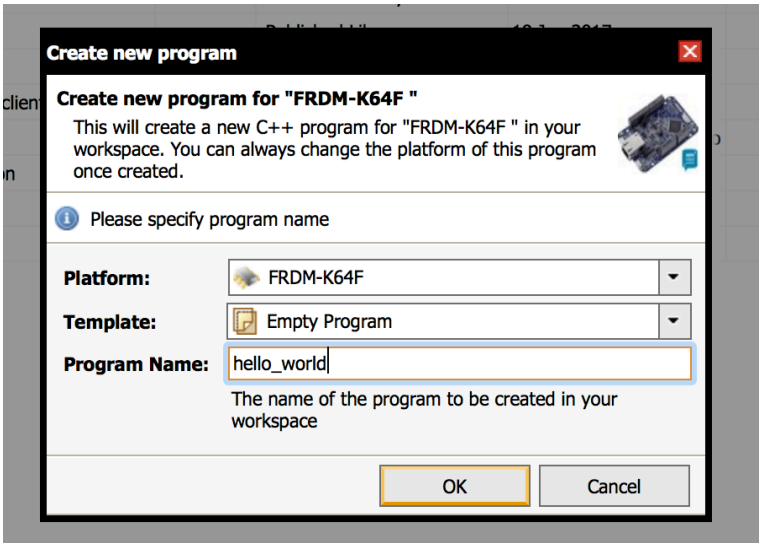


Figure 14 - Creating a new project

The next step is to import the libraries mentioned in the software and libraries section using the import button and click on the “Click here to import from URL” option shown in figure 15. Figure 16 shows the options available for importing the library. Enter all the url from table 4 and ensure that the target path is the project that was created earlier. Ensure that the import as library option is selected and update all sub-libraries is unchecked. Additional information about the library are also available on the url provided. After importing the libraries create the following three files, main.cpp, mbed_app.json and security.h in the program.

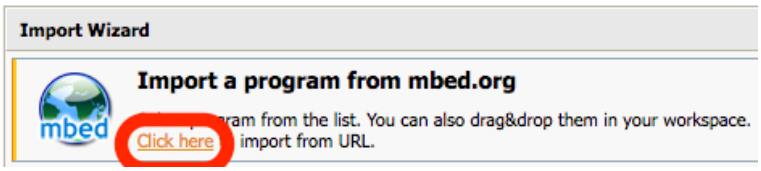


Figure 15 - Import from URL option (ARM, 2017)

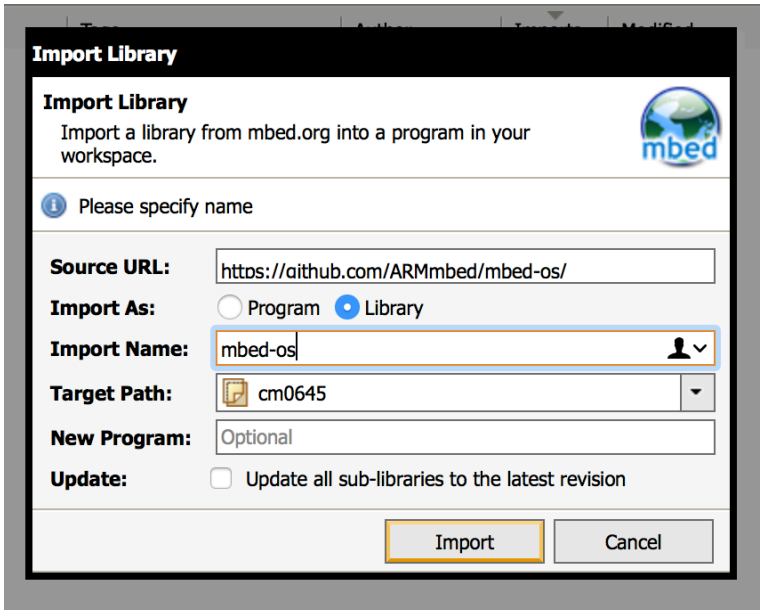


Figure 16 - Import library dialog

Table 4

Libraries	URL
easy-connect	https://github.com/ARMmbed/easy-connect/
esp8266-driver	http://developer.mbed.org/teams/ESP8266/code/esp8266-driver/
FXOS8700Q	http://developer.mbed.org/teams/NXP/code/FXOS8700Q/
HTU21D	http://developer.mbed.org/users/hwing91/code/HTU21D/
simple-mbed-client	http://developer.mbed.org/teams/sandbox/code/simple-mbed-client/
mbed-os	https://github.com/ARMmbed/mbed-os/

At this point the workspace should look like figure 17.

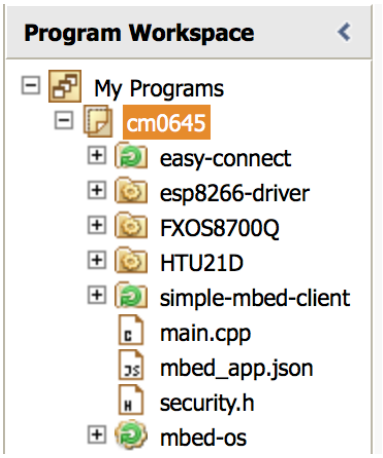


Figure 17 - Project with libraries set up and the default files required.

3.5.3. Building the client application

3.5.3.1. Connecting to the internet

With the project set up, its finally the time to add some application code. Lets start off by connecting the board to a wifi hotspot with an internet connection. Firstly, declare the wifi informations in the mbed_app.json file, this file is used to separate configuration and application code. A template is provide on ARM (2017) documentation and is shown in Figure 18. The next step is to add the codes shown in figure 19 to main.cpp to build a simple program that connects to the wifi declared in the configuration file. The programs uses the easy_connect library to help initialise the wifi module and connects to the internet.

mbed_app.json

```
{
    "config": {
        "network-interface":{
            "help": "options are ETHERNET,WIFI_ESP8266,MESH_LOWPAN_ND,MESH_THREAD",
            "value": "WIFI_ESP8266"
        },
        "esp8266-tx": {
            "help": "Pin used as TX (connects to ESP8266 RX)",
            "value": "D1"
        },
        "esp8266-rx": {
            "help": "Pin used as RX (connects to ESP8266 TX)",
            "value": "D0"
        },
        "esp8266-ssid": {
            "value": "\"Enter your Wifi SSID here\""
        },
        "esp8266-password": {
            "value": "\"Enter you Wifi password here\""
        },
        "esp8266-debug": {
            "value": false
        }
    },
    "target_overrides": {
        "*": {
            "target.features_add": ["IPV4"]
        }
    }
}
```

Figure 18 - Declaring the wifi connection information

main.cpp

```
#include "mbed.h"
#include "easy-connect.h"

// Serial interface
Serial pc(USBTX, USBRX);

int main() {
    // Connecting to wifi network
    NetworkInterface *network = easy_connect(true);
    if (!network) {
        pc.printf("Unable to connect to the internet.\n");
        return 1;
    } else {
        pc.printf("Successfully connected to the internet.\n");
    }
}
```

Figure 19 - Simple program to connect to the internet

3.5.3.2. Flashing the program

Once the codes are added, click on the compile button on the top of the online compiler and the binary of the program will be download on to the local computer. Plugged in the Freedom-K64 board in to a USB port on a computer and it should appear like a normal usb storage drive. Drag the binary onto the storage and the new program will be flashed on to the board. When done, open up a serial terminal and the status of the program should be shown depending on the status of the network. It should prints the success message when the wifi module manages to connect to the internet or an error message if something goes wrong. In a case where the wifi connection cannot be made, the esp8266-debug value in the mbed_app.json can be set to true to allow a more verbose debugging.

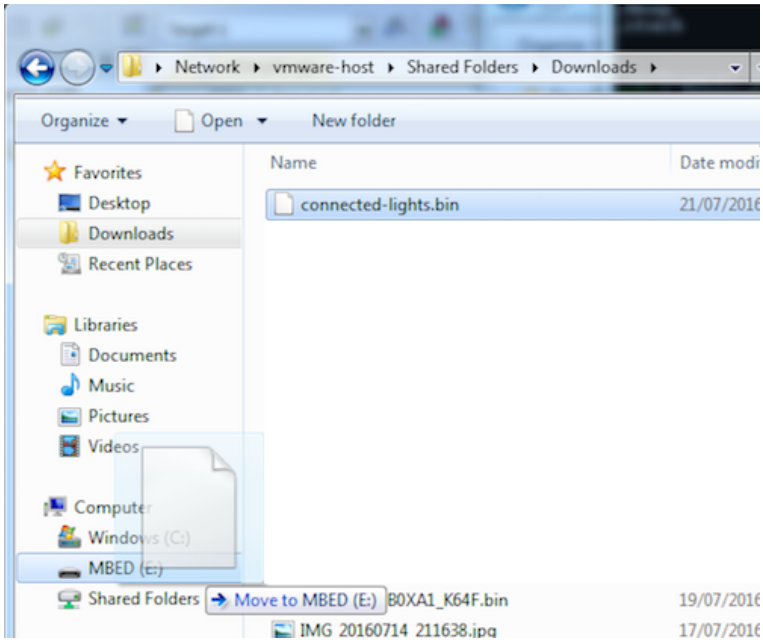


Figure 20 - Dragging the binary on to the board (ARM, 2017)

3.5.3.3. Connecting to device connector

Now that the board is able to connect to the internet, it time to connect the board to the device connector and make the sensors on the board accessible. All the data that flows between the board and the mbed device connector are encrypted and therefore before the board can be connected to the device connector a security certificate needs to be added into the application. The security certificate is available from the device connector webpage https://connector.mbed.com/ Login using the same mbed developer account that was created earlier and navigate to the *My Device > Security Credentials.* Click on *Get my security credentials* and the content inside the grey box shown in figure 21 will be needed to be copied into the security.h file that was created when the project is being set up.

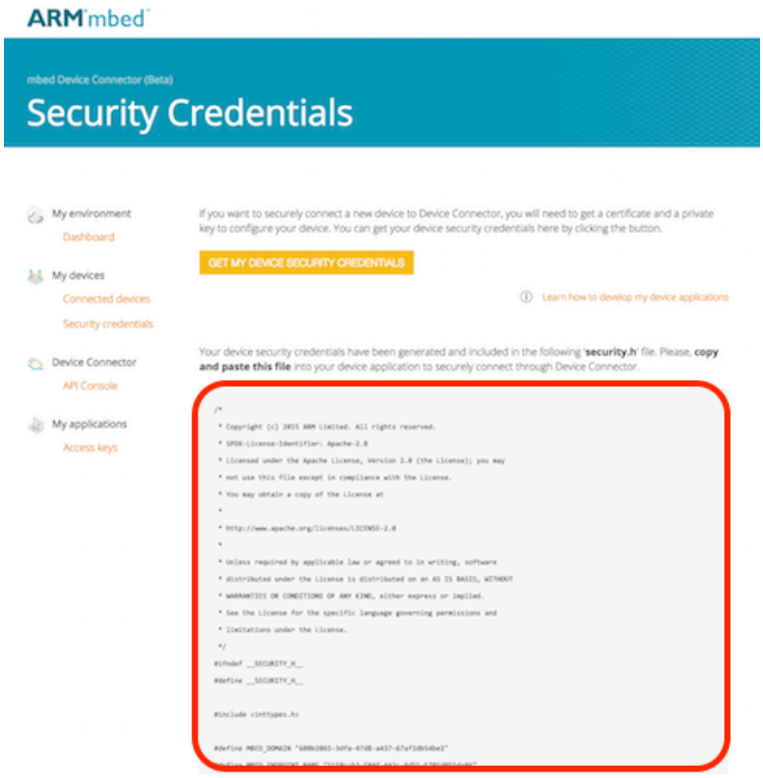


Figure 21 - mbed Device Connector security credentials page with security certificate generated (ARM, 2017)

With the security certificate in place its time to connect the board with the network connection that was created earlier. To connect to the mbed device connector, the simple-mbed-client library will be used. Figure 22 shows how the simple-mbed-client library can be used with the NetworkInterface to initiate the client which will be used later to manage the sensors and LEDs on the board.

main.cpp Raw

```cpp
#include "mbed.h"
#include "easy-connect.h"
#include "security.h"
#include "simple-mbed-client.h"

// Client for managing connections to device connector
SimpleMbedClient client;

// Serial interface
Serial pc(USBTX, USBRX);

int main() {
    // Connecting to wifi network
    NetworkInterface *network = easy_connect(true);
    if (!network) {
        pc.printf("Unable to connect to the internet.\n");
        return 1;
    }

    // Initialising mbed client and connecting to mbed device connector
    struct MbedClientOptions options = client.get_default_options();
    options.DeviceType = "mbed-program";
    if (!client.setup(options, network)) {
        pc.printf("Unable to set up mbed client.\n");
        return 1;
    }
}
```

Figure 22 - Updated program with the simple-mbed-client initialised

At this point the program is now able to connect to the internet and register itself to the mbed device connector. If all this goes well, the unique identifier that was generated earlier from the security credentials will appear in the mbed device connector dashboard shown in figure 23.

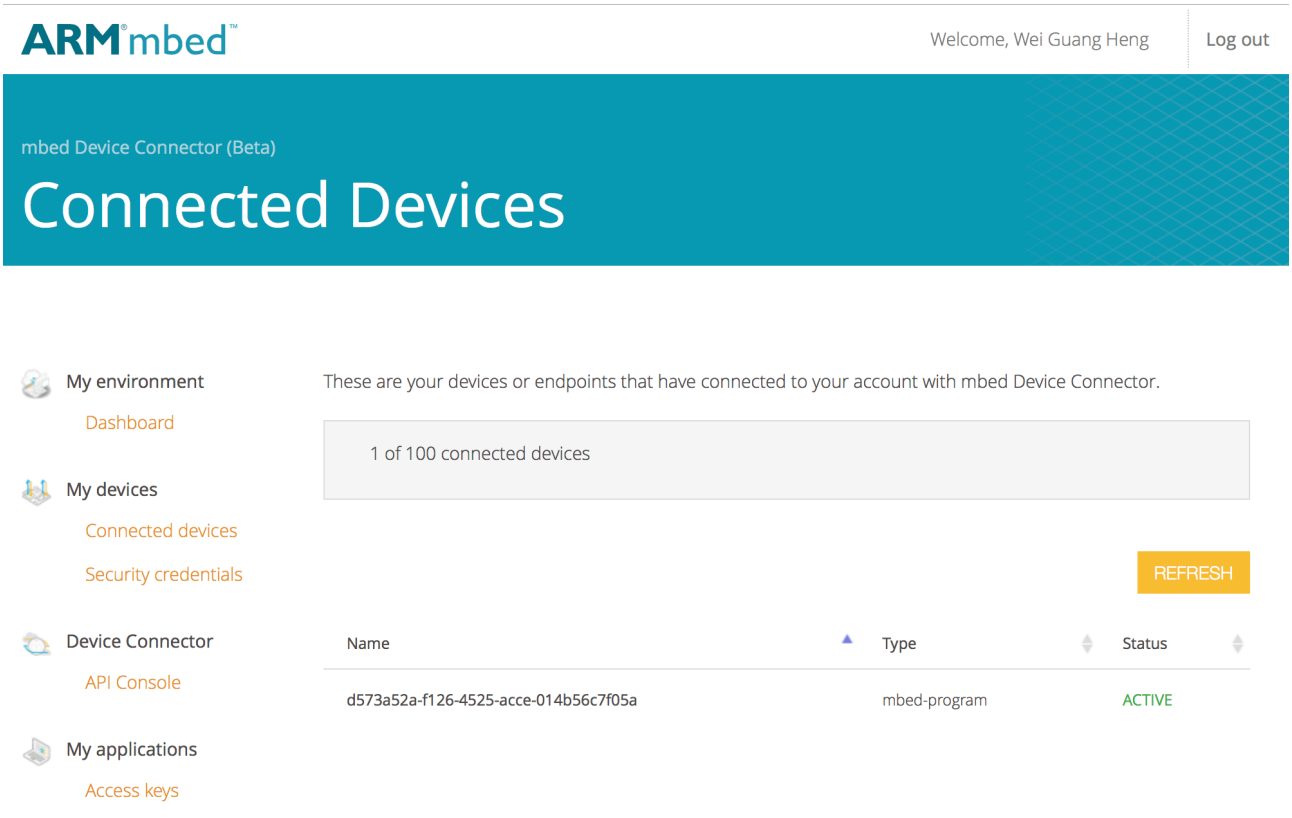


Figure 23 - mbed Device Connector showing the connected device

3.5.3.4. Making the sensors and LEDs accessible online

At this point the board should be able to connect to the internet and register itself on the mbed device connector. The last stage of the client development is to send the sensor values to the device connector and to allow the LEDs to be turned on or off online. Just a recap, there are two sensors on the Freedom-K64 board, the accelerometer and the magnetometer, three LEDs, red blue and green. In addition, there are two LEDs and three sensors on the Sparkfun weather shield, light, temperature and humidity. The LEDs on the weather shield will be used to indicate the status of the board. With this in mind, consider all these sensors and LEDs as resources and thats how they will be declared. The LEDs and sensors will be initialised normally shown in figure 24 and they will be declared as resources to the device connector shown in figure 25.

main.cpp Raw

```
// Initialising Weather shield
HTU21D weatherSensor(A4, A5);
AnalogIn lightSensor(A1);
DigitalOut blueStatusLED(D7);
DigitalOut greenStatusLED(D8);

// Initialising LEDs
DigitalOut redLED(LED1, 1);
DigitalOut greenLED(LED2, 1);
DigitalOut blueLED(LED3, 1);

// Initialising accelerometer and magnetometer
I2C i2c(PTE25, PTE24);
FXOS8700QAccelerometer acc(i2c, FXOS8700CQ_SLAVE_ADDR1);
FXOS8700QMagnetometer mag(i2c, FXOS8700CQ_SLAVE_ADDR1);
```

Figure 24 - Initialising the LEDs and sensors

main.cpp Raw

```
// Device connector resources:
SimpleResourceInt redValue = client.define_resource("led/0/red", 1, &redValueChanged);
SimpleResourceInt greenValue = client.define_resource("led/0/green", 1, &greenValueChanged);
SimpleResourceInt blueValue = client.define_resource("led/0/blue", 1, &blueValueChanged);

SimpleResourceInt accXValue = client.define_resource("acc/0/x", 0, M2MBase::GET_ALLOWED);
SimpleResourceInt accYValue = client.define_resource("acc/0/y", 0, M2MBase::GET_ALLOWED);
SimpleResourceInt accZValue = client.define_resource("acc/0/z", 0, M2MBase::GET_ALLOWED);

SimpleResourceInt magXValue = client.define_resource("mag/0/x", 0, M2MBase::GET_ALLOWED);
SimpleResourceInt magYValue = client.define_resource("mag/0/y", 0, M2MBase::GET_ALLOWED);
SimpleResourceInt magZValue = client.define_resource("mag/0/z", 0, M2MBase::GET_ALLOWED);

SimpleResourceInt lightValue = client.define_resource("weather/0/light", 0, M2MBase::GET_ALLOWED);
SimpleResourceInt humidityValue = client.define_resource("weather/0/humidity", 0, M2MBase::GET_ALLOWED);
SimpleResourceInt temperatureValue = client.define_resource("weather/0/temperature", 0, M2MBase::GET_ALLOWED);
```

Figure 25 - Declaring resources that will be available on device connector

As shown in figure 24, the LEDs are initialised using the mbed DigitalOut API. This will allow the LED to be turned on or off just by assigning a 0 or 1. The other sensors are then initialised with their respective libraries with the exception of the light sensor as it only provide its values as a voltage, therefore the AnalogIn API is used.

In figure 25, each resources are declared as a simple integer. Notice that the client that was previously declared is now used to define the resource that should show up in the device

connector dashboard which the board successfully connects itself. The first parameter of the define_resource function takes in a string that is formatted as the mbed Device Connector data model. The mbed Device Connector data model follows the following format.

ObjectID/ObjectInstance/ResourceId

The ObjectID is used to group things that are similar to each other. For example the first three resources related to the LEDs and therefore has the ObjectID of led and subsequently acc for accelerometer, mag for magnetometer and weather for the sensors on the weather shield.

In the current case there is only one instance of each individual items and therefore all ObjectInstance are declared as 0.

And finally the ResourceID is used to uniquely identify the individual resource on an object. For example there are three different colours of LED so therefore they are declared as red, green and blue but they are all under the same ObjectID.

The second parameter of the function is to define the default value of that resource. For the LEDs the default values are 1 which will ensure the LEDs will be off by default and the other resource will start with a default value of 0. The last parameter for the LED is an address of the function that will be called when the value of that resources is changed online. So for example, from the device connector the value of the LED resource can be changed from 1 to 0, when this happens the function redValueChanged will be called and depending on the new value that is sent from the device connector the LED can be turned on or off.

Now that the sensors and LEDs are initialised and declared as resources the final step is to upload the sensor values to the device connector which can be easily achieved just by updating the SimpleResourceInt value. Figure 26 shows an updated main function with the code to connect the device connector and upload the sensor readings to device connector.

main.cpp Raw

```
int main() {
    // Connecting to wifi network
    NetworkInterface *network = easy_connect(true);
    if (!network) {
        pc.printf("Unable to connect to the internet.\n");
        return 1;
    }

    // Initialising mbed client and connecting to mbed device connector
    struct MbedClientOptions options = client.get_default_options();
    options.DeviceType = "mbed-program";
    if (!client.setup(options, network)) {
        pc.printf("Unable to set up mbed client.\n");
        return 1;
    }
    // Updating the values
    while (1) {
      // Assigning the weather shield data to the weather resource
      lightValue = lightSensor.read_u16();
      humidityValue = weatherSensor.sample_humid();
      temperatureValue = weatherSensor.sample_ctemp();

      // Assigning the accelerometer data to the acc resource
      int16_t raX, raY, raZ;
      acc.getX(raX);
      accXValue = raX;
      acc.getY(raY);
      accYValue = raY;
      acc.getZ(raZ);
      accZValue = raZ;

      // Assigning the magnetometer data to the mag resource
      int16_t rmX, rmY, rmZ;
      mag.getX(rmX);
      magXValue = rmX;
      mag.getY(rmY);
      magYValue = rmY;
      mag.getZ(rmZ);
      magZValue = rmZ;
    }
}
```

Figure 26 - Uploading sensor values to the device connector

The sensor values are now uploaded to the device connector and can be accessed through the mbed device connector dashboard as shown in figure 27. Figure 28 shows the response of the API when a GET request is made on the temperature resource. The Base64 encoded payload indicates the current temperature around the board at the point of request is 23 degrees celcius.

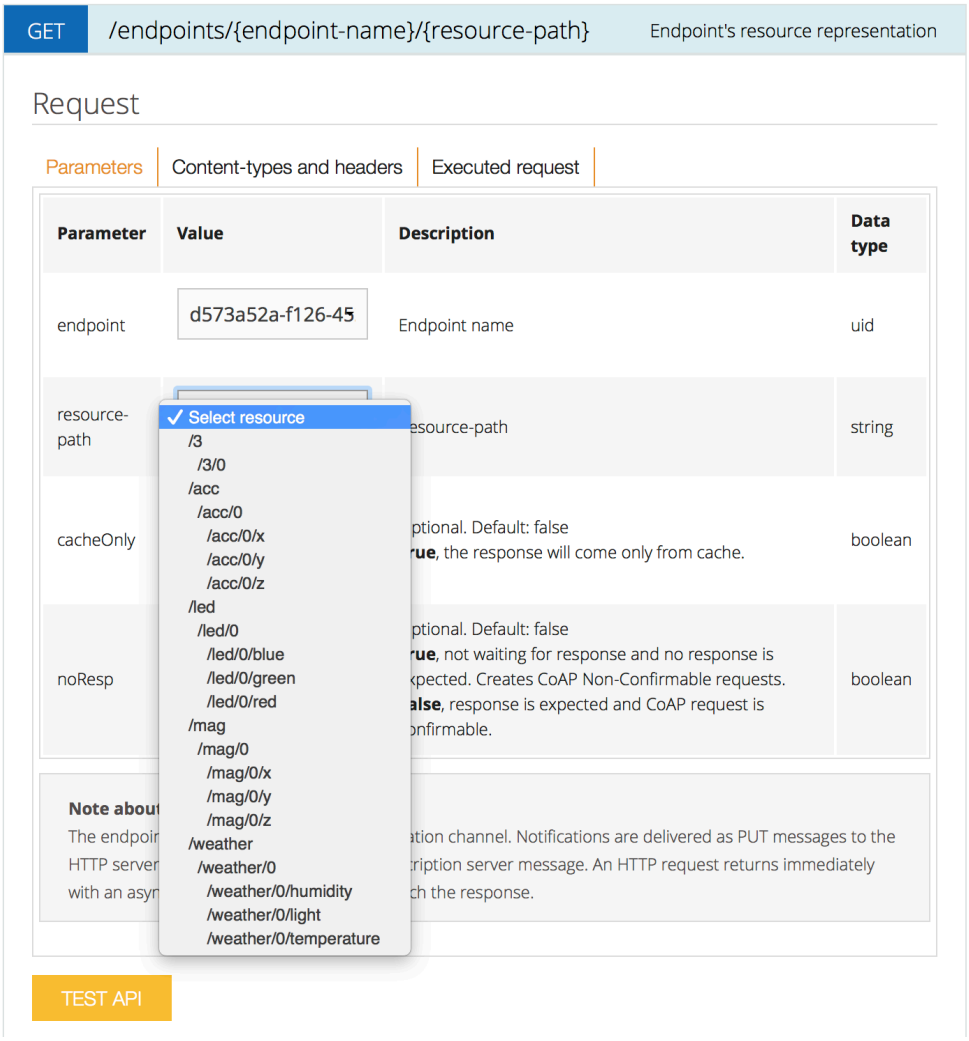


Figure 27 - mbed Device Connector dashboard showing the resources available

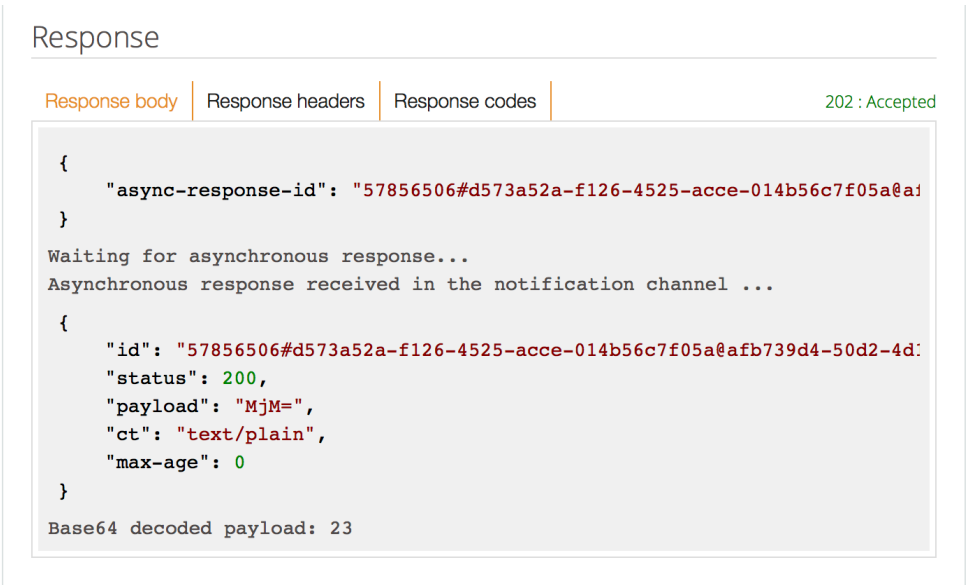


Figure 28 - Response from the REST API on the temperature resource

At this point, the board is publishing its sensor values to the device connector and the user is able to retrieve and manipulate the resources values. However while testing the program with the dashboard, the API seems to be unresponsive or slow. Upon reviewing the code, there is an issue with the way the resource value is updated. To update the values to the device connector, time is needed to collect the data from the sensor and to transmit them over the internet. The issue with the codes in figure 26 is that all this retrieving of values and updating are done on the same thread which has resulted in blocking and unexpected behaviour. As previously discussed, mbed OS provides an API EventQueue which helps to execute different functions in a different context. The simple-mbed-client library comes with an EventQueue object which can help to queue the updating value tasks in to different context to ensure the function are being executed efficiently. To do that, a function is created for every resources that needs to be updated and the codes that were initially in the main function in broken up in to individual function and will be passed in to the client event queue object as shown in figure 29 with the function being executed every one second.

main.cpp Raw

```
client.eventQueue()->call_every(1000, &updateAccXValue);
client.eventQueue()->call_every(1000, &updateAccYValue);
client.eventQueue()->call_every(1000, &updateAccZValue);
client.eventQueue()->call_every(1000, &updateMagXValue);
client.eventQueue()->call_every(1000, &updateMagYValue);
client.eventQueue()->call_every(1000, &updateMagZValue);
client.eventQueue()->call_every(1000, &updateLightValue);
client.eventQueue()->call_every(1000, &updateHumidityValue);
client.eventQueue()->call_every(1000, &updateTemperatureValue);
```

Figure 29 - Using the mbed client event queue to update the resources values

Now the program should be more responsive when requesting the value from the dashboard as all calls to the network is now queued and executed in a different context. A copy of the main.cpp file is available in the appendixes which will show the whole program.

3.5.4. Building the web application

Now that the board is publishing its resources and are accessible online, the final product of the solution is the web application. The web application will communicate with the device connector through the RESTful APIs and will extend the functionality of the dashboard, for example with the data from the device connector a graph can be plot to show how the values of the sensors changes over time. This section will explain the use of the mbed connector node API and build a NodeJS application to display the data from the board in a more intuitive way.

3.5.4.1. Getting the application access key

As usual, before any application can talk to the device connector a key is required this is to ensure that access to the mbed client board is restricted to the user's account. The access key can be easily generated from the device connector dashboard access keys section shown in figure 30.

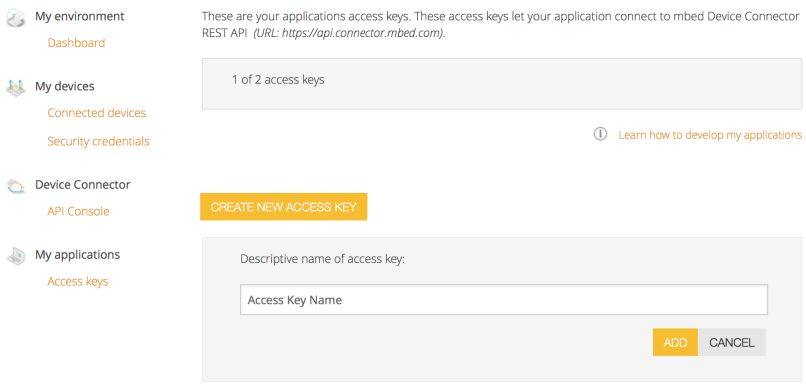


Figure 30 - Generating application access key

3.5.4.2. Setting up the NodeJS application

The NodeJS application that will be developed is base on the express web framework and will be using the mbed connector api node library to help manage the connections to the device connector. Before creating a new NodeJS project ensure that NodeJS is installed on the development computer. Additional libraries will be added to add more features like sockets for sending message between the back end server and front end client, handlebar templating engine for rendering html files and smoothie for plotting graphs. Once NodeJS is installed create a directory with the following structure shown in figure 31.

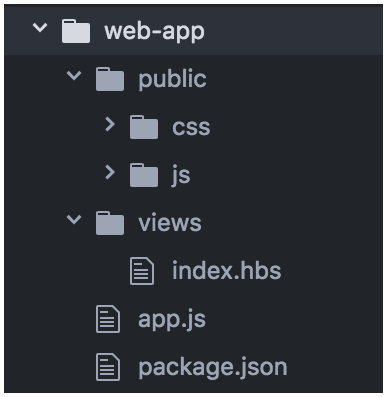


Figure 31 - Web application project structure

The css and js folder will contain the front end library files. This project uses the Skeleton boilerplate for easy creating a responsive site and Smoothie Charts for plotting the graph. Download the libraries from http://getskeleton.com and http://smoothiecharts.org respectively and add those files into this folder.

Inside the views folder there is a index.hbs file. This will be the template file that the server will render to the client when the user visit the front page. The file will contain all the html code and some javascript to layout all the different elements on the page. The app.js file will contain all the server side logic including the mbed connector api and the package.json file will describe the dependencies and provide some information about the application. Figure 32 shows all the dependencies that are needed on the server side.

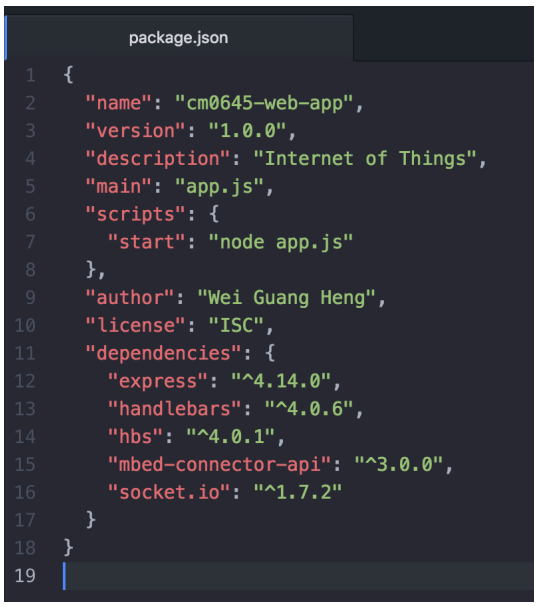


Figure 32 - package.json

3.5.4.3. Building the web server

With the dependencies declared in the package.json file run the command *npm install* from the project directory in the bash terminal and the package manager will download the files in to a new folder node_modules. After the libraries are installed declare all the modules that will be used in the app.js shown in figure 33.

Figure 33 - Importing modules

Next declare the access keys and the endpoints of the mbed client resources.

```
8  // Access key to mbed Device connector
9  var accessKey = process.env.ACCESS_KEY || "Put your access key here";
10
11 // Server port
12 var port = process.env.PORT || 8080;
13
14 // Device endpoint
15 var frdmK64Endpoint = 'Put your device endpoint here';
16
17 // Resource endpoints
18 var ledRedResourceURI = '/led/0/red';
19 var ledGreenResourceURI = '/led/0/green';
20 var ledBlueResourceURI = '/led/0/blue';
21
22 var accXResourceURI = '/acc/0/x';
23 var accYResourceURI = '/acc/0/y';
24 var accZResourceURI = '/acc/0/z';
25
26 var magXResourceURI = '/mag/0/x';
27 var magYResourceURI = '/mag/0/y';
28 var magZResourceURI = '/mag/0/z';
29
30 var temperatureResourceURI = '/weather/0/temperature';
31 var humidityResourceURI = '/weather/0/humidity';
32 var lightResourceURI = '/weather/0/light';
```

Figure 34 - Declaring access key and resource url

For the last part of building the server app, an instance of the express server object will be created. The express server will be responsible of serving the front page, manage the sockets between itself and the clients and connecting to the mbed connector to poll for new data. For the polling to work, the server will have to subscribe to the resource that it is interested in. In this case all the resources declared earlier will be subscribed. Once subscribed the notification call back function will be called every time the server received a new value on the resource. Sockets are used to allow the web client to communicate to the server and vice versa. When the notification callback is triggered or when the user toggles the switch on the front page, a message will be send via the sockets to either update the values on the webpage or to make a request to toggle the LEDs one the board. The message that were send will be in a key value dictionary form. The next few figures shows the remaining part of the server codes that sets up the server, creates the sockets, resource subscriptions and notification callbacks.

app.js

```
33
34  // Instantiate an mbed Device Connector object
35  var mbedConnectorApi = new MbedConnectorApi({
36    accessKey: accessKey
37  });
38
39  // Setting up the app
40  var app = express();
41
42  // Setting up the view directory
43  app.set('views', path.join(__dirname, 'views'));
44  app.set('view engine', 'hbs');
45
46  // Setting up the public directory
47  app.use(express.static(path.join(__dirname, 'public')));
48
49  // Set the port
50  app.set('port', port);
51
52  // Render the index page
53  app.get('/', function(request, response) {
54    response.render('index');
55  });
56
57  var server = http.Server(app);
58  var io = ioLib(server);
59
60  // Setup sockets for updating web UI
61  io.on('connection', function (socket) {
62    // Toggling the checkbox
63    socket.on('toggle-switch', function(data) {
64      console.log('Toogle switch: ', data);
65      mbedConnectorApi.putResourceValue(frdmK64Endpoint, data.id, data.value, function(error) {
66        if (error) console.error(error);
67      });
68    });
69  });
```

Figure 35 - Setting up the web server

app.js

```
71  // Notification callback
72  mbedConnectorApi.on('notification', function(notification) {
73    var path = notification.path;
74    var payload = notification.payload;
75
76    io.emit('notifications', {
77      id: path,
78      value: payload
79    });
80  });
```

Figure 36 - Notification callback

```
// Start the app
server.listen(port, function() {
  // Set up the notification channel, app will poll for notications from mbed
  mbedConnectorApi.startLongPolling(function(error) {
    if (error) console.error(error);

    // Check if device is connected to cloud connector
    mbedConnectorApi.getEndpoints(function(error, devices) {
      if (error) throw error;

      console.log('Found', devices.length, 'devices', devices);

      // Subscribe to notifications
      devices.forEach(function(device) {
        // Subscribe to accelerometer X value
        mbedConnectorApi.putResourceSubscription(frdmK64Endpoint, accXResourceURI, function(error) {
          if (error) console.error(error);
        });

        // Subscribe to accelerometer Y value
        mbedConnectorApi.putResourceSubscription(frdmK64Endpoint, accYResourceURI, function(error) {
          if (error) console.error(error);
        });

        // Subscribe to accelerometer Z value
        mbedConnectorApi.putResourceSubscription(frdmK64Endpoint, accZResourceURI, function(error) {
          if (error) console.error(error);
        });
      });
    }, { parameters: { type: 'mbed-program' } });

    console.log('App is now listening on port %s', port);
  })
});
```

Figure 37 - Start server and subscribing to notifications

3.5.4.4. Building the web client

The front page of the web application (Figure 38) should be displayed when the user visits the root url of the server, example http://localhost:8080 when the server is being access from the development machine. All the user interface logic are in the index.hbs file in the views folder. The css and js files will be loaded from the public folder. In the javascript section of the codes shown in figure 39 is where the sockets are initialised and handles the communication between itself and the server. The html portion will display the switches for the LED, values of the weather sensors and plot a graph for the motion sensors.

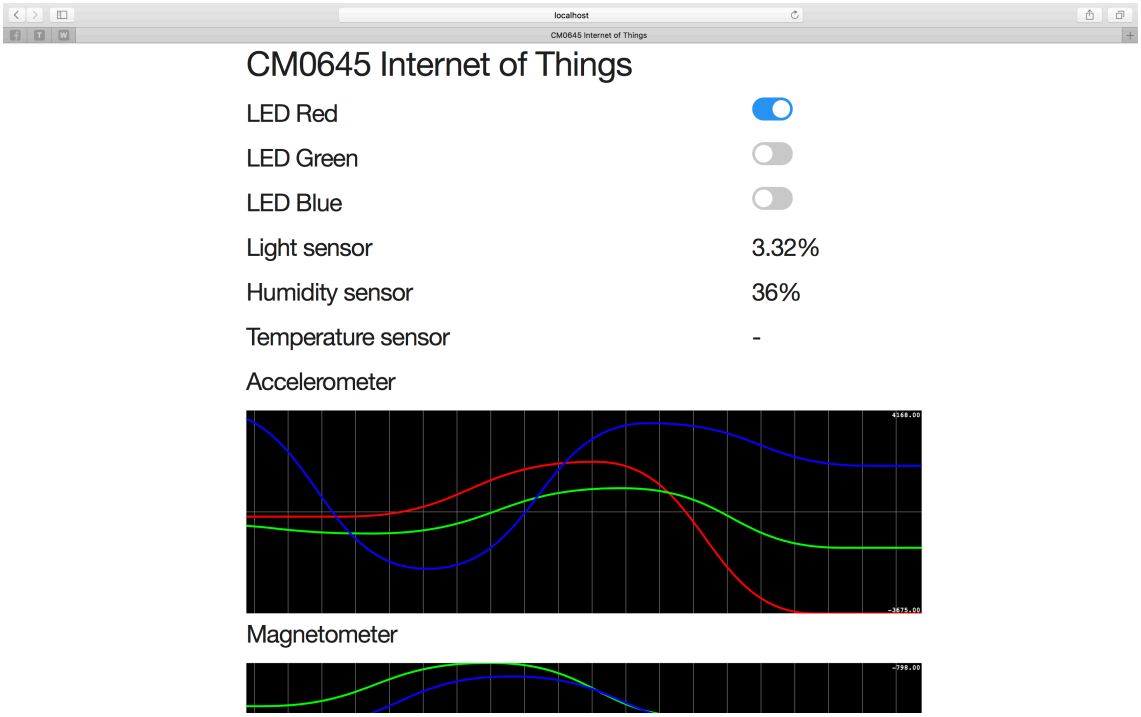


Figure 38 - Front page of the front end client

index.hbs

```
<script type="text/javascript" src="js/jquery-2.1.4.min.js"></script>
<script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/1.7.3/socket.io.js"></script>
<script type="text/javascript" src="js/smoothie.js"></script>
<script type="text/javascript">
  $(function() {
    var socket = io();

    // Handling toggle button events
    document.getElementById('red-toggle-switch').onclick = function() {
      var value = 1;
      if (this.checked) {
        value = 0;
      }
      socket.emit('toggle-switch', {
        id: '/led/0/red',
        value: value
      });
    };

    // Initialising acclerometer graph diagram
    var accelerometersmoothie = new SmoothieChart();
    accelerometersmoothie.streamTo(document.getElementById("accelerometercanvas"));

    // Data
    var accXLine = new TimeSeries();
    var accYLine = new TimeSeries();
    var accZLine = new TimeSeries();

    // Add to SmoothieChart
    accelerometersmoothie.addTimeSeries(accXLine, { strokeStyle:'rgb(255, 0, 0)', fillStyle:'rgba(255, 0, 0, 0.0)', lineWidth:3 });
    accelerometersmoothie.addTimeSeries(accYLine, { strokeStyle:'rgb(0, 255, 0)', fillStyle:'rgba(0, 255, 0, 0.0)', lineWidth:3 });
    accelerometersmoothie.addTimeSeries(accZLine, { strokeStyle:'rgb(0, 0, 255)', fillStyle:'rgba(0, 0, 255, 0.0)', lineWidth:3 });

    // Listen to server response
    socket.on('notifications', function (data) {
      console.log('Receieved data: ', data);
      if (data.id == "/acc/0/x") { // Accelerometer
        accXLine.append(new Date().getTime(), data.value);
      } else if (data.id == "/acc/0/y") {
        accYLine.append(new Date().getTime(), data.value);
      } else if (data.id == "/acc/0/z") {
        accZLine.append(new Date().getTime(), data.value);
      }
    });
  });
</script>
```

Figure 39 - Handling communications between server and client

4. Evaluation

This chapter will be divided into three different sections. The first section will discuss about the challenges and issues, the second section will evaluate the product and lastly the last section will discuss on the project process.

4.1. Challenges and issues

This section discuss about the challenges and issues mainly faced on the development side of the project and how each of these challenges and issues are being approached.

4.1.1. Updating the ESP8266 firmware

Updating the ESP8266 wifi module prove to be tricky initially as there are many different variants of it and are configured differently. There were also different tools available that can be used to update its firmware but they were confusing and difficult to use as its not easy to get a feedback to ensure the firmware is flashed correctly. Fortunately, Gratton maintainer of the esptool.py published the details of the ESP8266 that was use in this project and detailed the connection needed to set it to the flash firmware mode. The tool that was provide on the GitHub repository does not work after a few attempts and eventually the firmware was updated using the tool that mbed provides.

4.1.2. Unreliable networking

Even though the mbed board manages to connected to a wifi hotspot with the wifi module, it seems to have difficulties keeping itself connected over a long period of time. It was difficult to find the root cause of the issue as the driver does not really provide a way to show the signal strength of the hotspot that the wifi module is connected to. Eventually the last attempt hoping that the wifi module stay connected is to place it as close to the hotspot as possible but it still did not manage to keep itself connected.

4.2. Product evaluation

This product evaluation section will evaluate the product against the requirements specification and test plan initially set and discuss on the amount of success that was achieve for each one of them.

4.2.1. mbed Client application evaluation

As stated out in the requirements specification the client application is written and designed to run on ARM mbed enabled boards and uses APIs provided by mbed OS. The client is able to connect and publish information to the device connector and at the same time receives and execute request from the device connector. As previously discussed in the synthesis chapter of uploading sensor data to the device connector on the responsiveness of the API, steps were taken to ensure that the data could be uploaded and retrieved as quickly as possible using the EventQueue. However as there were nine sensors and three LEDs there is a noticeable delay between the

request received to turn on the LED to having the LED actually turning on. It is understood that the EventQueue will execute functions one by one and thats highly the reason why there is such delay.

4.2.2. Web application evaluation

The web application has also fulfilled the requirements of being able to present the data coming from the device connector and pushing request like switching on or off the LEDs. However, the web application is currently built to poll the device connector for new data, and that is not really an efficient way of handling new informations. Ideally ARM recommends to use its interrupt handler API which will only triggers the notification callback when there is new data from the connected devices. This approach is not feasible as it requires a public facing ip address that the device connector can connect to deliver the data. In the current scope of this project, the web application is running on a machine behind a firewall and does not have a public facing address.

4.3. Process evaluation

The project process consist of reviewing similar platforms and literature, sourcing for hardware components, producing requirements and test plans, developing and testing the product and writing the report. This section will focus on the project process, project management and personal learning curves.

4.3.1. Project management

The initial stage of coming out with the requirements prove to be tricky as there were quite a few uncertainties on what the platform is able to provide and the type of product that can be developed. Much effort have been put in to decide on the selection of the development board that is to be used. An ideal board for this project is to be able to have features that allows it to have enough sensors data to upload to the device connector and some LEDs to demonstrate that the board can be controlled remotely. The project plan did not consider the shipping time that may be needed for the development boards to be delivered and troubleshooting any possible hardware issues.

The literature review has only briefly discussed about the security and similar platforms. This projects requires more topics on the communication protocols and their design to be use in a network constraint environment. In additional there is no discussion on how such platforms actually helps developers to bring their prototype products to production devices.

4.3.2. Learning curves

This project requires various programming skills, knowledge about real time operating system and digital serial protocols in order to develop the product. During the development stage the following skills and knowledge have been gained or improved.

- C++
  The client application is fully developed in C++ language and it has enhance the understanding of pointers, classes and structs.

- NodeJS
  The web application is developed using the NodeJS framework based on javascript. There was no prior experience with this framework and through the development of this application, there was knowledge gained on managing dependencies using a package manager and some basic web hosting concepts.

- Operating system concepts
  During the analysis and development of the client application, more characteristic of an operating system have been introduced. Use of threads, queues and interrupt handlers are crucial to the performance of the application.

- Digital serial protocols
  Communication protocols like serial, I2C and SPI was introduced when sourcing for a sensor board. Previous knowledge in this area was initially limited to ethernet and CAN networks.

5. Conclusion and Recommendations

Even though the aims and objectives are met for this project, as discussed in the evaluation there are areas in the the application can be improved. The following are some recommendation that can be look into to improved the application and the literature review.

5.1. mbed Client

- mbed OS supports a range of network interfaces. Other connection method like mesh network or ethernet also be used.
- Look in to alternative ways to manage the resources on the mbed board to improve responsiveness.
- Explore the mbed OS files API as the Freedom-K64 comes with a SD card slot.

5.2. Web application

- Deploy the web application to a web hosting site with a public facing ip address so that it would be possible to use the interrupt handler method instead of the polling method.
- Connect to a database to store information coming from the board.
- As the connection between the user and the front end client is not encrypted consider adding SSL feature to the web application to only handle requests through SSL for more enhance security.

5.3. Literature review

- A more in depth discuss on the communication protocols or concepts like CoAP and M2M communications.

6. Appendices

Appendix - Terms of references

Background to project

As the world gets more connected to each other via the internet, gone are the days when an internet connected device is a desktop computer. People now are able to get online easily through their mobile phones. Mobile phones has changed the way we work and communicate. We are now able to respond to emails and get informations anywhere we are with an internet connection. Messaging services like WhatsApp and Telegram allows us to be able to message friends across the globe without worrying about the expensive text message fees. Video call services like Skype, Apple FaceTime and Google Hangouts has also enabled people to chat and see each other even though they are miles away from each other. This are just some of the examples of what the internet can do for us.

As technology advances, a new category of devices starts to emerge. These devices are what we know now as the Internet of Things (IoT). They are devices that able to receive or transmit information over the internet. One of such device can be a camera where a user will be able to remotely retrieve the image taken by it. Another such device can be a sensor device that measures temperature and humidity which uploads the measurements to a server where users can remotely monitor the environment that the device is in.

In the consumer market, one of the more common application of IoT is home automation. A home owner now is able to install these “smart devices” which they can control through an app on their Android or iOS devices. Some of these devices are able to connect to the internet and therefore allow the user to be able to control the devices even when they are not at home.

As the interest within the consumer market on home automation grew, companies like Amazon, Google and Apple are pushing out hardware and software that help to connect all these devices through a centralised app or product.

Amazon currently offers a consumer product call the Amazon Echo. It is a voice command device that allows the user to use their voice to issue commands to control those home devices. It also has a virtual assistant called Alexa which can help the user gets the weather forecast of just searching wikipedia for information.

Google has also announced a new product called Google Home. It is similar to Amazon Echo which also allows the user to control their home devices with their voice and comes with Google Assistant built into the device. Google Assistant is Google’s virtual assistant

that can help a person to manage their daily schedule and help user to search for information.

Apple has not announce any products similar to the Amazon Echo or Google Home but they have released a protocol known as HomeKit accessory protocol (HAP) for hardware markers to conform to when developing a smart device for the Apple ecosystem. These smart devices that conform to the protocol will be able to integrate with the Home app that Apple has introduce in iOS 10. Once the user has go through the initial setup of adding the devices into the app, the user will then be able to also control the home device and create schedules to automate their house scenes. Apple has also started partnering home builders to boast the rate of adoption for the platform.

Internet of Things are definitely gaining interest and popularity, developers and hobbyist are now getting into developing their own version of smart devices with single-board computers like the Raspberry Pi. The development community for these devices is growing and people are getting creative about the kinds of devices they can come out with.

Proposed work

ARM currently offers an integrated platform mbed which consists of a RTOS and cloud services that allows a developer to build a device and connecting it to a backend service which can be accessed remotely.

During the course of the project I will be investigating on the features that is provided by the mbed OS and implementing them through some sensors and actuators. After which I will be investigating on the features that ARM provides on their cloud services. Throughout this process I will be documenting on my findings and experience of developing on the platform that they provide.

Aims

- To build a device that runs on the mbed OS and able to communicate through the internet via ARM clouds services from the web or mobile device.

Objectives

- To investigate the use of features that are usually provided by a RTOS and how mbed OS implements them.
- To build a backend service on the ARM cloud platform to remotely retrieve or control the sensors and actuators.
- To build a web or mobile client that communicates with the ARM cloud services.

Skills

Writing a program in C	From modules operating systems and embedded systems.
Writing REST APIs on mbed platform	Documentations provided at www.mbed.com
Building a web client	From module web technology.
Building an iOS app	To be learn via online tutorial at www.raywenderlich.com
Building an Android app	From module mobile application development.

Resources - hardware / software

Software

- IDE Keil MDK
  - IDE for developing the micro controller board.
- Android studio
  - IDE for developing Android apps.
- Xcode
  - IDE for developing iOS apps

Hardware

- FRDM-K64F
- An Android device
- An iOS device

* All hardware will be provided by student.

Structure & contents of project report

I. Abstract
II. Introduction
III. Analysis
IV. Synthesis
V. Evaluation
VI. Conclusion and Recommendations
VII. Appendices
    A. Terms of references
    B. Source code

Marking scheme

Report: 40%

Abstract & Introduction	5%
Analysis	30%
Synthesis	30%
Evaluation & Conclusions	30%
Presentation	5%

Product: 50%

Fitness for Purpose	40%
Build Quality	60%

Viva 10%

[Complete after approval]

Red	☐
Amber	☐
Green	☑

northumbria UNIVERSITY NEWCASTLE

Department of Computer Science and Digital Technologies

UNDERGRADUATE COMPUTING PROJECTS: ETHICS REGISTRATION AND APPROVAL FORM

Section One: Registration *[To be completed by student]*

Title of research project/dissertation	IoT
Researcher's name	Wei Guang Heng
Programme of study	Computer Science
Academic Year	2016/2017
Module code	CM0645
Supervisor's name	David Kendall
Second marker's name	Jungong Han
Start Date of Project	12/9/16

Brief outline of research topic:

To study and to build an IoT solution based on the mbed platform.

Short description of proposed research methods including identification of participants:

Ethical considerations in the research project	YES	NO
1. Does your research involve an external organisation or partner?	☐	☑
2. Does your research involve human participants?	☐	☑
3. If yes to Q.2, will you inform the participants about the research?	☐	☑
4. Will you obtain their consent using the standard consent form?	☐	☑
5. Is any deception involved?	☐	☑
6. Do any participants constitute a 'vulnerable group'? (refer to definition of Vulnerable People)	☐	☑
7. Will the research involve the following information?		
Commercially sensitive	☐	☑
Personally sensitive	☐	☑
Politically sensitive	☐	☑
Legally sensitive	☐	☑
8. Is the research likely to have any significant environmental impacts?	☐	☑
9. Are there likely to be any risks for the participants in your research?	☐	☑
10. Are there likely to be any risks for you in conducting the research?	☐	☑
11. If yes [to 5, 6, 7, 8, 9 or 10 above] have you identified steps to address the issues and mitigate any risks to participants, yourself or the environment?	☐	☐ N/A

Statement to explain how any issues identified above will be addressed and what steps will be taken to mitigate such risks or adverse impacts

Section Two: Approval

Supervisor/Module Tutor's name confirming ethical risk status	DAVID KENDALL

Ethical approval *[Please tick as appropriate]*

Green - Ethical approval is given without conditions (supervisor may approve)	☑
Amber – (to be approved by one independent reviewer) Ethical approval is given with the following conditions: • Information to be provided to all participants • Participant consent to be obtained using the standard Research Participant Consent Form or otherwise in accordance with Faculty procedures • Data to be stored and destroyed securely in accordance with University guidelines • Adherence to Data Protection Act • Anonymity to be offered to participants • Commercial confidentiality to be provided to organisations(s) • Software vulnerabilities, exploits, hazardous software etc. not to be published without prior 'responsible disclosure' to the affected supplier. • Other (please state):	☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐
Red - Project is referred to FREC for approval.by two independent reviewers *Please e-mail the submission to PGR Faculty Support*	☐

Name & role of reviewer 1:	.....
Signature	.....
Date	.....

Name & role of reviewer 2:	.....
Signature	.....
Date	.....

Outcome of Review

Ethical category of research project
Based on the above Ethical Considerations and with reference to the University's Ethical Scrutiny Risk Assessment tool identify the Ethical category of your research project (refer to http://www.northumbria.ac.uk/static/5007/respdf/riskassesmenttool for further guidance):

[Please tick as appropriate]

Red	☐	vulnerable participants; human tissue; sensitive data; risks to participants & researchers etc.
Amber	☐	human participants requiring informed consent; commercially sensitive information etc.
Green	☑	no participants involved; secondary data only; no sensitive data

I have read the University and the Faculty Ethics Policy and Procedures and confirm that the answers I have given above are correct. Where issues arise under items 5, 6, 7, 8, 9 or 10 [above] I have described in writing how I intend to approach these issues in the research.

Researcher's signature _____

Date 21/10/16

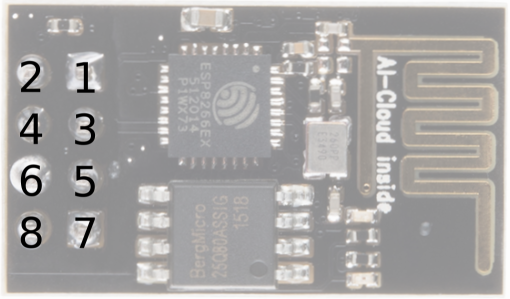
Section 1 Ethics Registration to be submitted to Principal Supervisor or Module Tutor and allocated to a reviewer as follows:

Green risk - may be approved by Supervisor
Amber risk - to be submitted for approval by one independent reviewer (second marker)
Red risk - to be submitted for approval by two independent members of Faculty Research Ethics Committee

ESP8266 Module (WRL-13678)

Name | Power
Ground | Control
Serial | GPIO
Arduino

D7	GPIO1	TX	2- TXO
		Chip Enable	4- CHPD
		Reset	6- RST
		3.3V	8- 3V
		GND	1- GND
	D2/SDA	GPIO2	3- GPIO2
	D0	GPIO0	5- GPIO0
D8	GPIO3	RX	7- RXI



PCB Antenna

Power
VCC-3.0-3.6V
Standby ~ 0.9uA
Running ~60-215mA,
Average ~ 80mA

Wifi Features
802.11 b/g/n
2.4GHz
WPA/WPA2
Wifi Direct

+20dBm output power (802.11b)

I/O Features
Integrated TCP/IP
Integrated TR switch, LNA, balun

Memory/Speed Features
80MHz
64KB instruction RAM
96KB data RAM
64K boot ROM
1MB* Flash Memory

Basic Connection
VCC - 3.3V
GND - GND
TX - RX on Arduino or FTDI
RX - TX on ARduino or FTDI
Chip Enable - 3.3V

Default Baud Rate
11520* 8N1

LEDs
Red: Power
Blue: TX

*milage may vary on different version of the board

AT Command Usage

Commands are case sensitive and should end with /r/n

Commands may use 1 or more of these types
Set = AT+<x>=<...> - Sets the value
Inquiry = AT+<x>? - See what the value is set at
Test = AT+<x>=? - See the possible options
Execute = AT+<x> - Execute a command

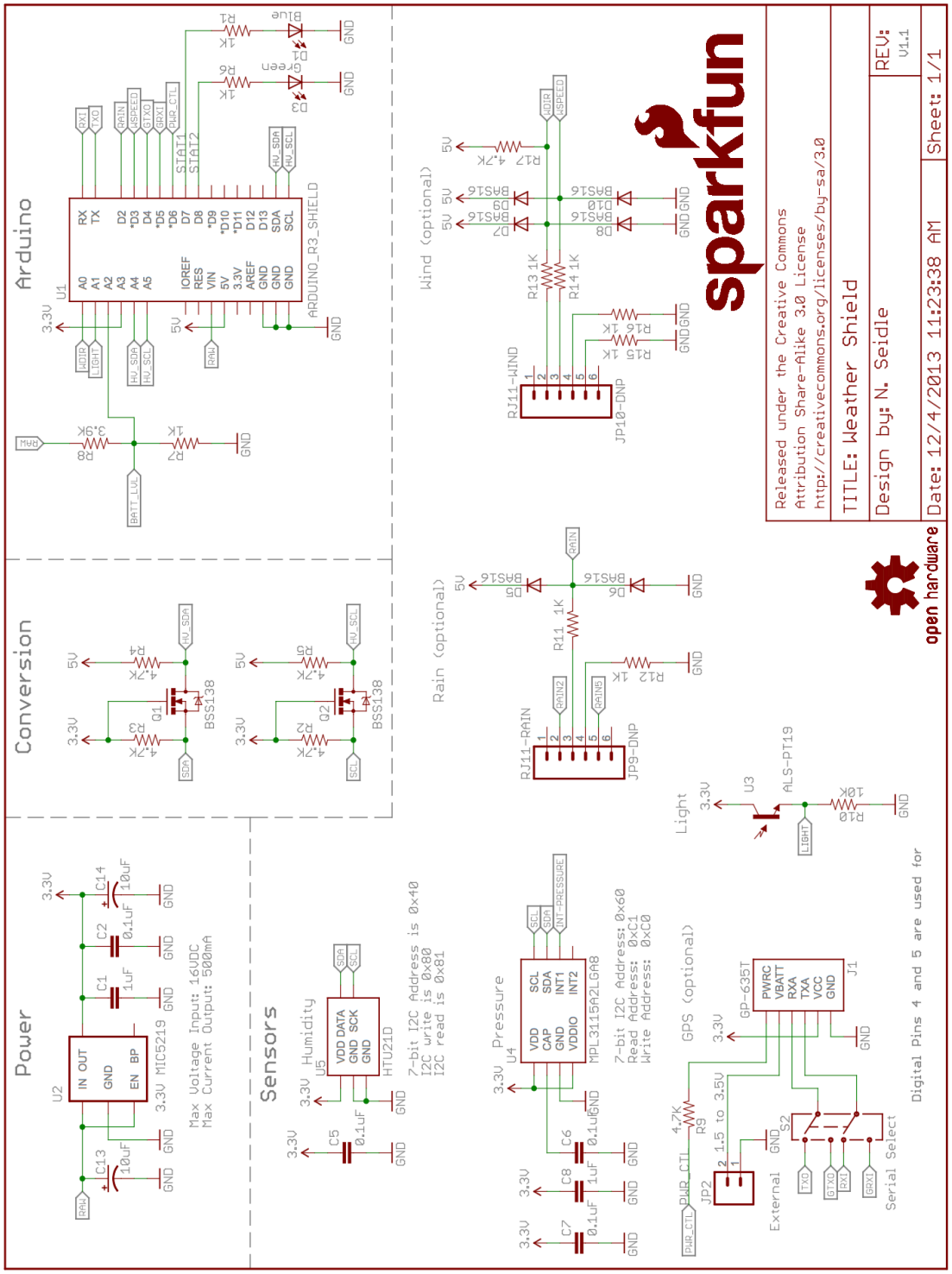
Commands with * have been depreciated in favor of COMMAND_CUR and COMMAND_DEF. CUR will not write the value to flash, DEF will write the valueto flash and be usedas the default in the future.

AT Command List

AT - Attention
AT+RST - Reset the board
AT+GMR - Firmware version
AT+CWMODE* - Operating Mode
 1. Client
 2. Access Point
 3. Client and Access Point
AT+CWJAP*=<ssid>,<pwd> - Join network
AT+CWLAP - View available networks
AT+CWQAP - Disconnect from network
AT+CWSAP*=<ssid>,<pwd><chl><ecn> - Set up access point
 0. Open. No security
 1. WEP
 2. WPA_PSK
 3. WPA2_PSK
 4. WPA_WPA2_PSK
AT+CWLIF - Show assigned IP addresses as access point
AT+CIPSTATUS - Show current status as socket client or server
AT+CIPSTART=<type>,<addr>,<port> - Connect to socket server
 IP is fixed at 192.168.4.1, mask is fixed at 255.255.255.0
 if CIPMUX is set to multichanel add <id> to beginning of string
AT+CIPCLOSE - Close socket connection
AT+CIFSR - Show assigned IP address when connected to network
AT+CIPMUX=<mode> - Set connection
 0. Single Connection
 1. Multi-Channel Connection
AT+CIPSERVER=<mode>[,<port>](AT+CIPMUX=1) - Default port is 333
 0. Close the Socket Server
 1. Open the Socket Server
AT+CIPMODE=<mode> - Set transpartent mode
 Data received will be sent to serial port as
 0. +IPD,<connection channel>,<length>format (AT+CIPMUX=[0,1])
 1. Data stream (AT+CIPMUX=0)
AT+CIPSTO=<time> - Set auto socket client disconnect timout from 1-28800s

Example commands
 AT+CWMODE=? //View options for mode (test)
 AT+CWMODE=3 //Set mode to client and access modes (set)
 AT+CWLAP //View available networks (execute)
 AT+CWJAP = "ssid","password" //Join network (set)
 AT+CWJAP? //View the current network (inquiry)
 AT+CIFSR //Show IP address (execute)
 AT+CWQAP //Disconnect from network (execute)
 AT+CWSAP="apoint","pass",11,0//Setup an open access point (set)
 AT+CWLIF //Show devices connected to access point

Appendix 2 - Sparkfun weather shield schematics



Appendix 3 - mbed client main.cpp

```
#include "mbed.h"
#include "FXOS8700Q.h"
#include "HTU21D.h"
#include "security.h"
#include "easy-connect.h"
#include “simple-mbed-client.h"

// Client for managing connections to device connector
SimpleMbedClient client;

// Serial interface
Serial pc(USBTX, USBRX);

// Initialising Weather shield
HTU21D weatherSensor(A4, A5);
AnalogIn lightSensor(A1);
DigitalOut blueStatusLED(D7);
DigitalOut greenStatusLED(D8);

// Initialising LEDs
DigitalOut redLED(LED1, 1);
DigitalOut greenLED(LED2, 1);
DigitalOut blueLED(LED3, 1);

// Initialising accelerometer and magnetometer
I2C i2c(PTE25, PTE24);
FXOS8700QAccelerometer acc(i2c, FXOS8700CQ_SLAVE_ADDR1);
FXOS8700QMagnetometer mag(i2c, FXOS8700CQ_SLAVE_ADDR1);

// Forward declaration
void redValueChanged(int newValue);
void greenValueChanged(int newValue);
void blueValueChanged(int newValue);

void updateAccXValue();
void updateAccYValue();
void updateAccZValue();

void updateMagXValue();
void updateMagYValue();
void updateMagZValue();

void updateLightValue();
void updateHumidityValue();
void updateTemperatureValue();

void registered();

// Device connector resources:
SimpleResourceInt redValue = client.define_resource("led/0/red", 1,
&redValueChanged);
SimpleResourceInt greenValue = client.define_resource("led/0/green", 1,
&greenValueChanged);
SimpleResourceInt blueValue = client.define_resource("led/0/blue", 1,
&blueValueChanged);

SimpleResourceInt accXValue = client.define_resource("acc/0/x", 0,
M2MBase::GET_ALLOWED);
SimpleResourceInt accYValue = client.define_resource("acc/0/y", 0,
M2MBase::GET_ALLOWED);
SimpleResourceInt accZValue = client.define_resource("acc/0/z", 0,
M2MBase::GET_ALLOWED);
```

```
SimpleResourceInt magXValue = client.define_resource("mag/0/x", 0,
M2MBase::GET_ALLOWED);
SimpleResourceInt magYValue = client.define_resource("mag/0/y", 0,
M2MBase::GET_ALLOWED);
SimpleResourceInt magZValue = client.define_resource("mag/0/z", 0,
M2MBase::GET_ALLOWED);

SimpleResourceInt lightValue = client.define_resource("weather/0/light", 0,
M2MBase::GET_ALLOWED);
SimpleResourceInt humidityValue = client.define_resource("weather/0/humidity",
0, M2MBase::GET_ALLOWED);
SimpleResourceInt temperatureValue = client.define_resource("weather/0/
temperature", 0, M2MBase::GET_ALLOWED);

// Colour updated from the cloud,
void redValueChanged(int newValue) {
    redLED = newValue;
}

void greenValueChanged(int newValue) {
    greenLED = newValue;
}

void blueValueChanged(int newValue) {
    blueLED = newValue;
}

// Upload results to device connector
void updateAccXValue() {
    int16_t raX;
    acc.getX(raX);
    accXValue = raX;
}

void updateAccYValue() {
    int16_t raY;
    acc.getY(raY);
    accYValue = raY;
}

void updateAccZValue() {
    int16_t raZ;
    acc.getZ(raZ);
    accZValue = raZ;
}

void updateMagXValue() {
    int16_t rmX;
    mag.getX(rmX);
    magXValue = rmX;
}

void updateMagYValue() {
    int16_t rmY;
    mag.getY(rmY);
    magYValue = rmY;
}

void updateMagZValue() {
    int16_t rmZ;
    mag.getZ(rmZ);
    magZValue = rmZ;
}

void updateLightValue() {
```

```
    lightValue = lightSensor.read_u16();
}

void updateHumidityValue() {
    humidityValue = weatherSensor.sample_humid();
}

void updateTemperatureValue() {
    temperatureValue = weatherSensor.sample_ctemp();
}

// Called when we registered with mbed Device Connector
void registered() {
    // Turn on blue LED to indicate client is now connected
    blueStatusLED = 1;

    // Enable the accelerometer
    acc.enable();

    // Start updating values when connected
    client.eventQueue()->call_every(1000, &updateAccXValue);
    client.eventQueue()->call_every(1000, &updateAccYValue);
    client.eventQueue()->call_every(1000, &updateAccZValue);
    client.eventQueue()->call_every(1000, &updateMagXValue);
    client.eventQueue()->call_every(1000, &updateMagYValue);
    client.eventQueue()->call_every(1000, &updateMagZValue);
    client.eventQueue()->call_every(1000, &updateLightValue);
    client.eventQueue()->call_every(1000, &updateHumidityValue);
    client.eventQueue()->call_every(1000, &updateTemperatureValue);
}

int main() {
    // Turn on green LED
    greenStatusLED = 1;

    // Ensure the blue LED is turned off first
    blueStatusLED = 0;

    // Connecting to wifi network
    NetworkInterface *network = easy_connect(true);
    if (!network) {
        pc.printf("Unable to connect to the internet.\n");
        return 1;
    }

    // Initialising mbed client and connecting to mbed device connector
    struct MbedClientOptions options = client.get_default_options();
    options.DeviceType = "mbed-program";
    if (!client.setup(options, network)) {
        pc.printf("Unable to set up mbed client.\n");
        return 1;
    }
    client.on_registered(client.eventQueue()->event(&registered));
}
```

Appendix 4 - Web server app.js

```
// Importing modules
var MbedConnectorApi = require('mbed-connector-api');
var express = require('express');
var path = require('path');
var ioLib = require('socket.io');
var http = require('http');

// Access key to mbed Device connector
var accessKey = process.env.ACCESS_KEY ||
"SwgkE7Q8UD9j5W4PP6lUw1PbUVe66EUDq2Ur3OfDqb6OoXsH0P3U0o1wBnOTv1oFXuAvAmAJofZHGgn
47E7Lv1zVWI5yHjLAI4Su";

// Server port
var port = process.env.PORT || 8080;

// Device endpoint
var frdmK64Endpoint = 'd573a52a-f126-4525-acce-014b56c7f05a';

// Resource endpoints
var ledRedResourceURI = '/led/0/red';
var ledGreenResourceURI = '/led/0/green';
var ledBlueResourceURI = '/led/0/blue';

var accXResourceURI = '/acc/0/x';
var accYResourceURI = '/acc/0/y';
var accZResourceURI = '/acc/0/z';

var magXResourceURI = '/mag/0/x';
var magYResourceURI = '/mag/0/y';
var magZResourceURI = '/mag/0/z';

var temperatureResourceURI = '/weather/0/temperature';
var humidityResourceURI = '/weather/0/humidity';
var lightResourceURI = '/weather/0/light';

// Instantiate an mbed Device Connector object
var mbedConnectorApi = new MbedConnectorApi({
  accessKey: accessKey
});

// Setting up the app
var app = express();

// Setting up the view directory
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'hbs');

// Setting up the public directory
app.use(express.static(path.join(__dirname, 'public')));

// Set the port
app.set('port', port);

// Render the index page
app.get('/', function(request, response) {
  response.render('index');
});

var server = http.Server(app);
var io = ioLib(server);
```

```
// Setup sockets for updating web UI
io.on('connection', function (socket) {
  // Toggling the checkbox
  socket.on('toggle-switch', function(data) {
    console.log('Toogle switch: ', data);

    mbedConnectorApi.putResourceValue(frdmK64Endpoint, data.id, data.value,
function(error) {
      if (error) console.error(error);
    });
  });
});

// Notification callback
mbedConnectorApi.on('notification', function(notification) {
  var path = notification.path;
  var payload = notification.payload;

  io.emit('notifications', {
    id: path,
    value: payload
  });
});

// Start the app
server.listen(port, function() {
  // Set up the notification channel, app will poll for notications from mbed
  mbedConnectorApi.startLongPolling(function(error) {
    if (error) console.error(error);

    // Check if device is connected to cloud connector
    mbedConnectorApi.getEndpoints(function(error, devices) {
      if (error) throw error;

      console.log('Found', devices.length, 'devices', devices);

      // Subscribe to notifications
      if (devices.length > 0) {
        // Subscribe to accelerometer X value
        mbedConnectorApi.putResourceSubscription(frdmK64Endpoint,
accXResourceURI, function(error) {
          if (error) console.error(error);
        });

        // Subscribe to accelerometer Y value
        mbedConnectorApi.putResourceSubscription(frdmK64Endpoint,
accYResourceURI, function(error) {
          if (error) console.error(error);
        });

        // Subscribe to accelerometer Z value
        mbedConnectorApi.putResourceSubscription(frdmK64Endpoint,
accZResourceURI, function(error) {
          if (error) console.error(error);
        });

        // Subscribe to magnetometer X value
        mbedConnectorApi.putResourceSubscription(frdmK64Endpoint,
magXResourceURI, function(error) {
          if (error) console.error(error);
        });

        // Subscribe to magnetometer Y value
        mbedConnectorApi.putResourceSubscription(frdmK64Endpoint,
magYResourceURI, function(error) {
```

```
          if (error) console.error(error);
        });

        // Subscribe to magnetometer Z value
        mbedConnectorApi.putResourceSubscription(frdmK64Endpoint,
magZResourceURI, function(error) {
          if (error) console.error(error);
        });

        // Subscribe to light sensor value
        mbedConnectorApi.putResourceSubscription(frdmK64Endpoint,
lightResourceURI, function(error) {
          if (error) console.error(error);
        });

        // Subscribe to humidity sensor value
        mbedConnectorApi.putResourceSubscription(frdmK64Endpoint,
humidityResourceURI, function(error) {
          if (error) console.error(error);
        });

        // Subscribe to temperature sensor value
        mbedConnectorApi.putResourceSubscription(frdmK64Endpoint,
temperatureResourceURI, function(error) {
          if (error) console.error(error);
        });
      }

    }, { parameters: { type: 'mbed-program' } });

    console.log('App is now listening on port %s', port);

  })

});
```

Appendix 5 - Web client index.hbs

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="css/normalize.css">
  <link rel="stylesheet" href="css/skeleton.css">
  <link rel="stylesheet" href="css/styles.css">
  <title>CM0645 Internet of Things</title>
</head>

<body>
  <div class="container">
    <h1>CM0645 Internet of Things</h1>

    <!-- Start of row -->
    <div class="row">
      <!-- 9 columns to display resource name -->
      <div class="nine columns">
        <h3>LED Red</h3>
      </div>
      <!-- 3 columns to display toggle button -->
      <div class="three columns">
        <label class="switch">
          <input id= "red-toggle-switch" type="checkbox">
          <div class="slider round"></div>
        </label>
      </div>
    </div>
    <!-- End of row -->

    <!-- Start of row -->
    <div class="row">
      <!-- 9 columns to display resource name -->
      <div class="nine columns">
        <h3>LED Green</h3>
      </div>
      <!-- 3 columns to display toggle button -->
      <div class="three columns">
        <label class="switch">
          <input id= "green-toggle-switch" type="checkbox">
          <div class="slider round"></div>
        </label>
      </div>
    </div>
    <!-- End of row -->

    <!-- Start of row -->
    <div class="row">
      <!-- 9 columns to display resource name -->
      <div class="nine columns">
        <h3>LED Blue</h3>
      </div>
      <!-- 3 columns to display toggle button -->
      <div class="three columns">
        <label class="switch">
          <input id= "blue-toggle-switch" type="checkbox">
          <div class="slider round"></div>
        </label>
      </div>
    </div>
    <!-- End of row -->
```

```
    <!-- Start of row -->
    <div class="row">
      <!-- 9 columns to display resource name -->
      <div class="nine columns">
        <h3>Light sensor</h3>
      </div>
      <!-- 3 columns to display value -->
      <div class="three columns">
        <h3 id="light-sensor">-</h3>
      </div>
    </div>
    <!-- End of row -->

    <!-- Start of row -->
    <div class="row">
      <!-- 9 columns to display resource name -->
      <div class="nine columns">
        <h3>Humidity sensor</h3>
      </div>

      <!-- 3 columns to display value -->
      <div class="three columns">
        <h3 id="humidity-sensor">-</h3>
      </div>
    </div>
    <!-- End of row -->

    <!-- Start of row -->
    <div class="row">
      <!-- 9 columns to display resource name -->
      <div class="nine columns">
        <h3>Temperature sensor</h3>
      </div>
      <!-- 3 columns to display value -->
      <div class="three columns">
        <h3 id="temperature-sensor">-</h3>
      </div>
    </div>
    <!-- End of row -->

    <h3>Accelerometer</h3>
    <canvas id="accelerometercanvas" width= "1000" height="300"></canvas>

    <h3>Magnetometer</h3>
    <canvas id="magnetometercanvas" width="1000" height="300"></canvas>
  </div>

  <script type="text/javascript" src="js/jquery-2.1.4.min.js"></script>
  <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/
socket.io/1.7.3/socket.io.js"></script>
  <script type="text/javascript" src="js/smoothie.js"></script>
  <script type="text/javascript">
    $(function() {

      var socket = io();

      // Handling toggle button events
      document.getElementById('red-toggle-switch').onclick = function() {
        var value = 1;
        if (this.checked) {
          value = 0;
        }
        socket.emit('toggle-switch', {
          id: '/led/0/red',
          value: value
        });
```

```
      };

      document.getElementById('green-toggle-switch').onclick = function() {
        var value = 1;
        if (this.checked) {
          value = 0;
        }
        socket.emit('toggle-switch', {
          id: '/led/0/green',
          value: value
        });
      };

      document.getElementById('blue-toggle-switch').onclick = function() {
        var value = 1;
        if (this.checked) {
          value = 0;
        }
        socket.emit('toggle-switch', {
          id: '/led/0/blue',
          value: value
        });
      };

      // Initialising acclerometer graph diagram
      var accelerometersmoothie = new SmoothieChart();

accelerometersmoothie.streamTo(document.getElementById("accelerometercanvas"));

      // Data
      var accXLine = new TimeSeries();
      var accYLine = new TimeSeries();
      var accZLine = new TimeSeries();

      // Add to SmoothieChart
      accelerometersmoothie.addTimeSeries(accXLine, { strokeStyle:'rgb(255, 0,
0)', fillStyle:'rgba(255, 0, 0, 0.0)', lineWidth:3 });
      accelerometersmoothie.addTimeSeries(accYLine, { strokeStyle:'rgb(0, 255,
0)', fillStyle:'rgba(0, 255, 0, 0.0)', lineWidth:3 });
      accelerometersmoothie.addTimeSeries(accZLine, { strokeStyle:'rgb(0, 0,
255)', fillStyle:'rgba(0, 0, 255, 0.0)', lineWidth:3 });

      // Initialising magnetometer graph diagram
      var magnetometersmoothie = new SmoothieChart();

magnetometersmoothie.streamTo(document.getElementById("magnetometercanvas"));

      // Data
      var magXLine = new TimeSeries();
      var magYLine = new TimeSeries();
      var magZLine = new TimeSeries();

      // Add to SmoothieChart
      magnetometersmoothie.addTimeSeries(magXLine, { strokeStyle:'rgb(255, 0,
0)', fillStyle:'rgba(255, 0, 0, 0.0)', lineWidth:3 });
      magnetometersmoothie.addTimeSeries(magYLine, { strokeStyle:'rgb(0, 255,
0)', fillStyle:'rgba(0, 255, 0, 0.0)', lineWidth:3 });
      magnetometersmoothie.addTimeSeries(magZLine, { strokeStyle:'rgb(0, 0,
255)', fillStyle:'rgba(0, 0, 255, 0.0)', lineWidth:3 });

      // Listen to server response
      socket.on('notifications', function (data) {
        console.log('Receieved data: ', data);
        if (data.id == "/acc/0/x") { // Accelerometer
          accXLine.append(new Date().getTime(), data.value);
```

```
        } else if (data.id == "/acc/0/y") {
          accYLine.append(new Date().getTime(), data.value);
        } else if (data.id == "/acc/0/z") {
          accZLine.append(new Date().getTime(), data.value);
        } else if (data.id == "/mag/0/x") { // Magnetometer
          magXLine.append(new Date().getTime(), data.value);
        } else if (data.id == "/mag/0/y") {
          magYLine.append(new Date().getTime(), data.value);
        } else if (data.id == "/mag/0/z") {
          magZLine.append(new Date().getTime(), data.value);
        } else if (data.id == "/weather/0/light") { // Weather shield
          var value = (data.value/65536) * 100;
          $("#light-sensor").html(value.toFixed(2) + "%");
        } else if (data.id == '/weather/0/temperature') {
          $("#temperature-sensor").html(data.value + "°C");
        } else if (data.id == '/weather/0/humidity') {
          $("#humidity-sensor").html(data.value + "%");
        }
      });
    });
  </script>
</body>
</html>
```

7. References

Android Developer. (2016). *Peripheral I/O.* [online] Available at: https://developer.android.com/things/sdk/pio/index.html [Accessed 16 Mar. 2017].

ARM mbed. (n.d.) *ESP8266.* [online] Available at: https://developer.mbed.org/teams/ESP8266/ [Accessed 3 Apr. 2017].

ARM mbed. (n.d.) *ESP8266-Firmware-Update-To-Expressif* [online] Available at: https://developer.mbed.org/teams/ESP8266/code/ESP8266-Firmware-Update-To-Espressif/ [Accessed 3 Apr. 2017].

ARM mbed. (n.d). *FRDM-K64F.* [online] Available at: https://developer.mbed.org/platforms/frdm-k64f/ [Accessed 29 Mar. 2017].

ARM mbed. (n.d.). *Introduction to the mbed OS API.* [online] Available at: https://docs.mbed.com/docs/mbed-os-api-reference/en/latest/ [Accessed 28 Mar. 2017].

ARM mbed. (n.d.). *About the mbed OS event loop.* [online] Available at: https://docs.mbed.com/docs/mbed-os-api-reference/en/latest/APIs/tasks/events/ [Accessed 28 Mar. 2017].

ARM mbed. (n.d.). *Building an internet connected lighting system.* [online] Available at: https://docs.mbed.com/docs/building-an-internet-connected-lighting-system/en/latest/ [Accessed 28 Mar. 2017].

ARM mbed. (n.d.). *mbed-connector-api-node.* [online] Available at: https://github.com/ARMmbed/mbed-connector-api-node [Accessed 28 Mar. 2017].

BBC News. (2016). *'Smart' home devices used as weapons in website attack.* [online] Available at: http://www.bbc.co.uk/news/technology-37738823 [Accessed 16 Mar. 2017].

Canonical. (2017). *Taking charge of the IoT's securities vulnerabilities.* [online] Available at: https://pages.ubuntu.com/IoT-Security-whitepaper.html [Accessed 5 Apr. 2017].

GSMA. (n.d.). *Automotive & Intelligent Transport Systems.* [online] Available at: http://www.gsma.com/connectedliving/automotive/ [Accessed 22 Mar. 2017].

Raspberry Pi Foundation. (n.d.). *A security update for Raspbian Pixel.* [online] Available at: https://www.raspberrypi.org/blog/a-security-update-for-raspbian-pixel/ [Accessed 22 Mar. 2017].

Resin.io. (n.d.). *Features | Resin.io.* [online] Available at: https://resin.io/features/ [Accessed 16 Mar. 2017].

Sparkfun. (n.d.). Sparkfun Weather Shield [online] Available at: https://www.sparkfun.com/products/13956 [Accessed 3 Apr. 2017].

Sparkfun. (n.d.). *WiFi Module - ESP8266* [online] Available at: https://www.sparkfun.com/products/13678 [Accessed 3 Apr. 2017].

The MagPi Magazine. (2017). *Sales soar: Raspberry Pi British board beats Commodore 64, world's third best-selling computer.* [online] Available at: https://www.raspberrypi.org/magpi/raspberry-pi-sales/ [Accessed 16 Mar. 2017].

Amadeo, R. (2016). Google's new "Android Things" OS hopes to solve awful IoT security. [online] Ars Technica. Available at: https://arstechnica.com/gadgets/2016/12/google-brillo-rebrands-as-android-things-googles-internet-of-things-os/ [Accessed 16 Mar. 2017].

Brisbourne, A (n.d.). Tesla's Over-the-Air Fix: Best Example Yet of the Internet of Things?. [online] WIRED. Available at: https://www.wired.com/insights/2014/02/teslas-air-fix-best-example-yet-internet-things/ [Accessed 16 Mar. 2017].

Columbus, L. (2016). Roundup Of Internet Of Things Forecasts And Market Estimates, 2016. [online] Forbes. Available at: https://www.forbes.com/sites/louiscolumbus/2016/11/27/roundup-of-internet-of-things-forecasts-and-market-estimates-2016/#1808c080292d [Accessed 16 Mar. 2017].

Evan, D. (2011). The Internet of Things. How the Next Evolution of the Internet Is Changing Everything. [online] Cisco. Available at: http://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf [Accessed 5 Apr. 2017].

Gratton, A (n.d.). ESP8266 Boot Mode Selection. [online] GitHub. Available at: https://github.com/espressif/esptool/wiki/ESP8266-Boot-Mode-Selection [Accessed 4 Apr. 2017].

Morgan, J. (2014). A Simple Explanation Of 'The Internet Of Things'. [online] Forbes. Available at: https://www.forbes.com/sites/jacobmorgan/2014/05/13/simple-explanation-internet-things-that-anyone-can-understand/#5070f2b31d09 [Accessed 16 Mar. 2017].

Ouimet, M. (2017). Securing the Industrial Internet of Things. [online] CA Technologies. Available at: https://www.ca.com/us/rewrite/articles/security/securing-the-industrial-internet-of-things.html [Accessed 16 Mar. 2017].

Piekarski, W (2016). Announcing updates to Google's Internet of Things platform: Android Things and Weave. [online] Android Developers Blog. Available at: https://android-developers.googleblog.com/2016/12/announcing-googles-new-internet-of-things-platform-with-weave-and-android-things.html [Accessed 5 Apr. 2017].

Risteska Stojkoska, B. and Trivodaliev, K. (2017). A review of Internet of Things for smart home: Challenges and solutions. *Journal of Cleaner Production*, 140, pp.1454-1464.

Sarukkai, S (2016). Ransomware and the Internet of Things: A Growing Threat. [online] eSecurityPlanet. Available at: esecurityplanet.com/network-security/ransomware-and-the-internet-of-things-a-growing-threat.html [Accessed 5 Apr. 2017].

Weber, R. (2010). Internet of Things – New security and privacy challenges. *Computer Law & Security Review*, 26(1), pp.23-30.

8. Figures