

# SET09118 WORKBOOK

Brian Davison

EDINBURGH NAPIER UNIVERSITY 40278490

# House Sensing workbook

## Part 1: Outline

The aim of the project was to setup and demonstrate 2-way communication between a hardware sensor and a mobile device. For this task, I have chosen to make a mobile application that can control an LED strip. The hardware component will also have a temperature sensor attached that will take in temperature and humidity to be displayed on the mobile application.

The initial idea was to use a mobile application to control a Wi-Fi enabled ESP8266 chipset with an Arduino Uno, however during the process of the development, the Arduino was dropped as it wasn't needed to achieve the aim. To see the initial development plan, go to Appendix A.

## Part 2: Planning

### Background research

---

*What information do I want displayed on the mobile application?*

- Temperature;
- Humidity;
- Room Name; - To be named in program on hardware (Arduino / ESP8266)
- LED Strip's Colour; - RGB values

*What products are already available and what do they do?*

Home automation systems and home heating Smart-meters already exist and have been on the market for a few years. British Gas already has a home heating application called HIVE that works with smart-meters to produce statistics on heating and energy usage, allowing the control of a houses heating from a mobile phone or computer.

*Topics that require investigation*

What ESP8266 board will be used?

What Temperature and Humidity Sensors will be used?

What format will the messages sent between the Hardware Circuit and the Android Application?

How will the LED Strip be controlled?

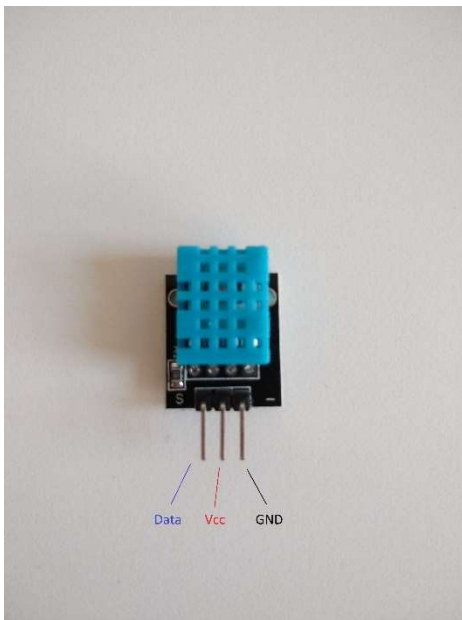
### *Results of investigation – strategic choices*

**The WeMos D1 mini** is an ESP8266 board that has a built in USB-to-UART bridge, so a separate board isn't required, and it can be programmed directly. The board also has inbuilt voltage regulators allowing the board to be powered using 5V rather than the standard 3.3V for ESP boards. Furthermore, the chip can be powered via its micro-USB port which is useful as the ESP8266 chip consumes a lot of power compared to other components and it can be plugged directly into the mains power via a standard USB power adapter (5 volts, 1 amp).



This requires a board manager (URL: [http://arduino.esp8266.com/stable/package\\_esp8266com\\_index.json](http://arduino.esp8266.com/stable/package_esp8266com_index.json)). When installing the board, the version of the driver is **2.4.2** and not the most recent version as there are issues with compiling and uploading to the board for certain libraries.

**The DHT11 Sensor** is a temperature and humidity sensor with three pins (DATA, VCC, GND). According to the product page on the Adafruit website (<https://www.adafruit.com/product/386>), the DHT11 temperature and humidity sensor can:



- Measure humidity between 20 and 80%, accurate to 5%;
- Measure temperature between 0 and 50°C, accurate to 2°C;
- Take readings once every second;

To access the sensor, the Arduino needs a couple of libraries. These libraries are:

- Adafruit's Unified Sensor Library ([https://github.com/adafruit/Adafruit\\_Sensor](https://github.com/adafruit/Adafruit_Sensor));
- Adafruit's DHT Sensor Library (<https://github.com/adafruit/DHT-sensor-library>);

Both libraries can be downloaded and installed via the ArduinoIDE's library manager.

**The FastLED LED library** allows the control over a variety of LED controller types, including the ones that will be used for this project (WS2801, <http://fastled.io/>). The library is easy to use and works on the ESP8266 architecture, unlike other LED control libraries.



Picture of one of the LED Strips that will be used. As can be seen in the picture, the strip has 4 wires: The power leads (Red and Black), the Clock wire (Yellow) and the Signal wire (Green).

**MQTT** is the type of web protocol that will be used for the messages between the hardware and the mobile application. A library must be installed for the ESP8266 to implement this (<https://github.com/256dpi/arduino-mqtt>). For processing, a version of the MQTT library must also be installed (<https://github.com/256dpi/processing-mqtt>).

**JSON** stands for **JavaScript Object Notation** and it is a convenient format for messages as it treats them like an object with multiple values. ArduinoJSON (<https://arduinojson.org/>) is a library that allows the use of JSONObjects within the Arduino Code. The version installed is **5.13.2** as I have other projects that are being developed that are reliant on that version, however the library should still work as intended.

*MoSCoW***M**

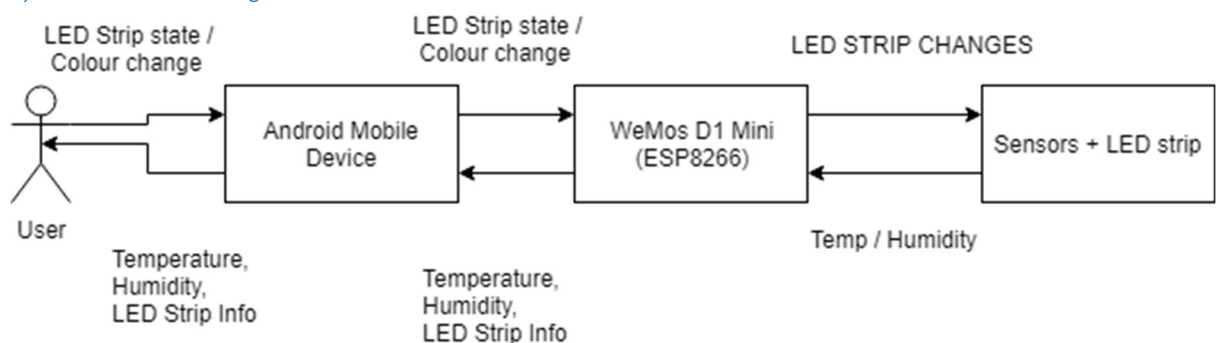
- Temperature displayed on ESP8266 serial monitor;
- Humidity displayed on ESP8266 serial monitor;
- Communication between Android application and ESP8266 Chip; - connecting to MQTT broker.
- Temperature and Humidity displayed on android application;
- LED strip powered state changed manually;
- LED strip colour changed manually
- LED strip powered state changed via Android application;

**S**

- LED strip colour changed wirelessly;
- Multiple sets of ESP8266 connected sensors communicating with Android application;
- Make application look nice visually;

**C**

- Fan (DC MOTOR) powered dependent on temperature;
- Fan (DC MOTOR) powered state displayed on Android application;
- Scan for LCD I2C addresses connected to ESP8266;
- Temp and Humidity displayed on LCD Screen **IF LCD AVAILABLE**;
- Wireless control of fan speed;
- House the hardware components in a project box or shell, making it visually appealing;
- Use Raspberry Pi as an MQTT broker instead of shiftr.io;

**W****Design***System overview diagram**List of system elements and their requirements***Main System Elements:**

- **Android Application** – Programmed using Android Studio;

INPUTS (From Circuit):

Temperature, Humidity, LED Strip info (ON/OFF/COLOUR);

INPUTS (From User):

LED Strip state/colour changes;

OUTPUTS (To ESP8266):

LED Strip state/colour changes;

OUTPUTS (To User):

Temperature, Humidity, LED Strip info (ON/OFF/COLOUR);

- **ESP8266 Circuit** – Programmed using the Arduino IDE;

INPUTS (From sensors):

Temperature, Humidity;

INPUTS (From Mobile):

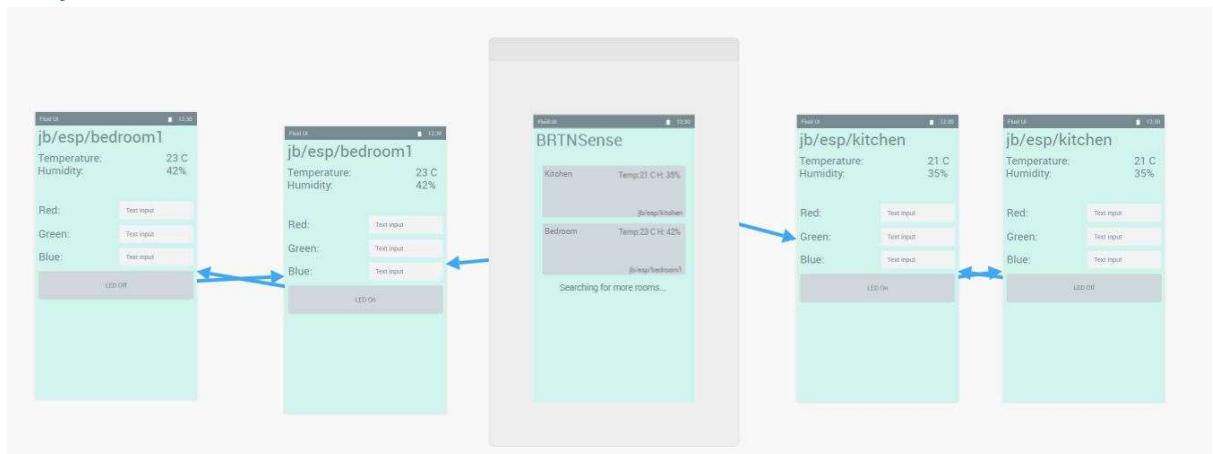
LED state/colour changes;

OUTPUTS (To LED strip):

LED state/colour changes;

OUTPUTS (To Mobile):

Temperature, Humidity, LED Strip Info (ON/OFF/COLOUR);

*Wireframes**Challenges expected*

The main difficulty will be in relearning the programming environment required to develop my application, along with applying newly learned concepts and protocols such as MQTT in the new environment (Android Studio). Additionally, applying JSON in both environments is an unfamiliar concept and will require a lot of research to complete. To see the challenges expected from the original project idea, see Appendix A.

## Project plan

---

### *Order of development*

1. Set up shiftr.io account and relevant Namespace. This will be the broker for the MQTT messages that are being sent between the Mobile Application and the ESP8266.
2. Develop a Hardware prototype that only deals with taking in temperature and humidity data and printing it to the Serial.
3. Develop a prototype app that can send and receive via the MQTT Broker
4. Develop code for the ESP8266 to receive messages from the MQTT Broker
5. Add LED strip to the circuit
6. Add WeMos D1 mini to the circuit programming it to send temperature data via the MQTT broker.
7. Add temperature and Humidity to the Mobile APP
8. Use this message to turn on the LED Strip
9. Add a second room with controls
10. Clean up app and hardware making it more presentable

### *Time schedule*

The time given for this project is approximately 4 weeks, with aims for main development to be completed by the end of week 3 and beautification occurring during week 4. I aim to spend a minimum of 40 hours developing and writing this project.

## Part 3: Implementation

### ESP8266

---

#### *Software specification*

There are multiple constraints with the development of the Hardware implementation, with memory being the main one. This is because the WeMos D1 mini has limited memory and storage which, in turn, limits the size and complexity of code that can be written. ArduinoIDE will be used to program the WeMos D1 mini board, with full commentary throughout the code. Additionally, a GitHub repository will be used for version control for both the files for the ESP8266 and the Mobile Application programs.

Libraries being implemented:

- ArduinoJSON (<https://arduinojson.org/>);
- Adafruit's Unified Sensor Library ([https://github.com/adafruit/Adafruit\\_Sensor](https://github.com/adafruit/Adafruit_Sensor));
- Adafruit's DHT Sensor Library (<https://github.com/adafruit/DHT-sensor-library>);
- Freetronic's FTRGBLED (<https://github.com/freetronics/FTRGBLED/>);
- MQTT (<https://github.com/256dpi/arduino-mqtt>);

A previous version of the ArduinoJSON library is being implemented due to its use in another project. The version used, as mentioned in my background research, is **5.13.2**, as opposed to the most recent version which is **6.10.0** at the time of writing this.

The ESP8266/WeMos D1 Mini board needs installed onto the ArduinoIDE by adding the URL to the ArduinoIDE's preferences (URL:

[http://arduino.esp8266.com/stable/package\\_esp8266com\\_index.json](http://arduino.esp8266.com/stable/package_esp8266com_index.json)). Version **2.4.2** has been installed as I have had problems when compiling code for the ESP8266 using newer versions.

## Android application

### Software specification

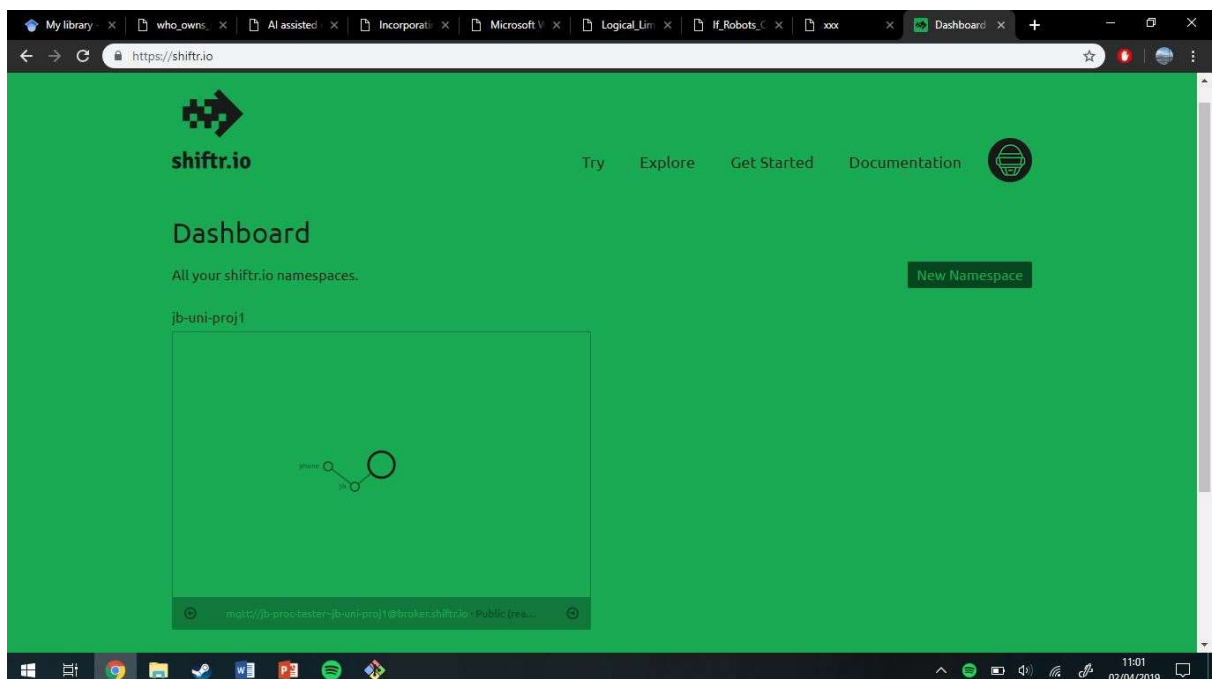
For the mobile application, I have used Android Studio. This is similar to Processing 3+ that I am familiar with as both are written using Java, however, Android Studio allows the development of more complicated applications in quicker time as there is a visual XML design editor that doesn't require the hard coding of basic features such as buttons. The MQTT protocol that allows the communication between the mobile application and the ESP8266 circuit, will require an additional library and as such I have used Eclipse Paho's MQTT Library.

To view the previous Mobile Application specification see Appendix A.

## Integration Log

### Shiftr.io

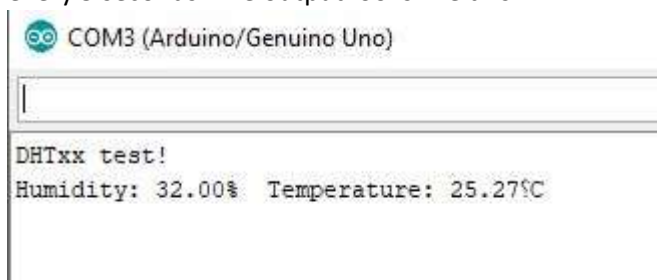
Fortunately, I already have a Shiftr.io account set up, with a namespace, from a previous project requiring MQTT and an MQTT broker, that can be used without changing much.



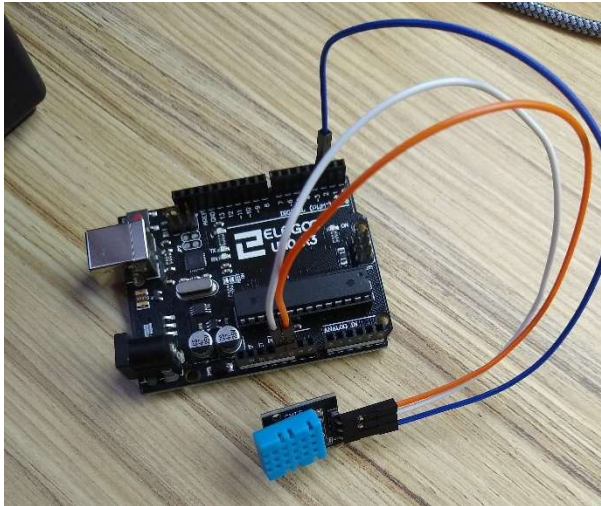
The shiftr.io dashboard can be seen in this image with the previous namespace visible with one branch currently able to be subscribed and published to.

### First Prototype: Temperature to Serial

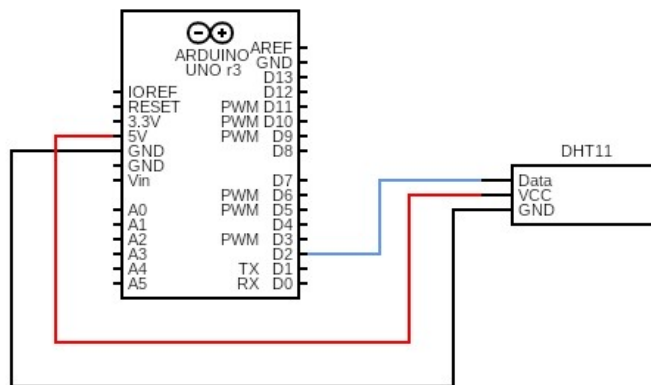
For the first prototype, an edited version of the DHT sensor library > DHTtester example sketch has been used. It displays only the temperature and the humidity and takes a measurement every 5 seconds. The output looks like this:







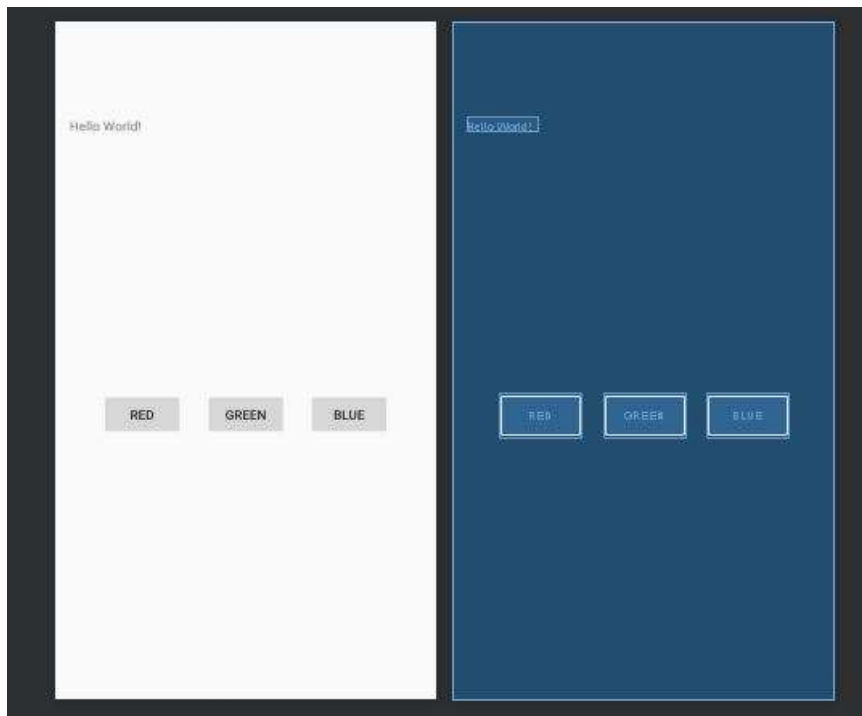
Here is a picture of the circuit that was used to get the previous picture's output. The next photo will be a circuit diagram of the above circuit.



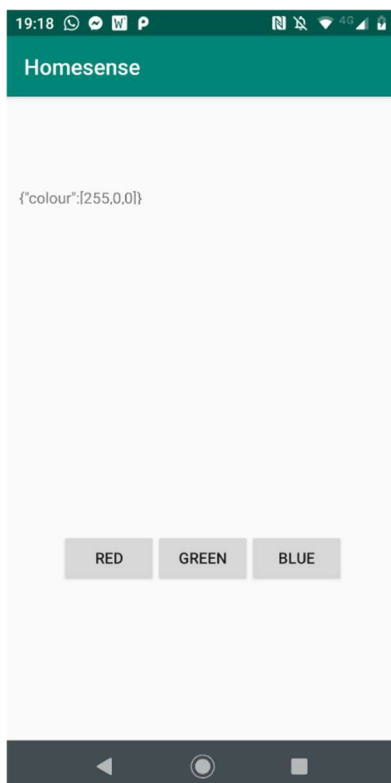
**POST DEVELOPMENT NOTE:** This was developed for the Arduino board as at this time in the project I had assumed that my project would use both the WeMos D1 Mini board AND an Arduino Uno, however the program was fully transferrable to the WeMos board without edits.

#### *First Mobile Application*

The processing application has been found to be a little limited when being used for Android development, therefore I have decided to use Android Studio with Eclipse Paho's MQTT library. Development of the application has been relatively easy until this stage as Android Studio does most of the work for setting up the initial base app, and there are plenty of guides online on how to use MQTT with Android. One such guide on how to set up MQTT with android is MQTT Android Client Tutorial by Wildan Maulana Syahidillah (<https://wildanmsyah.wordpress.com/2017/05/11/mqtt-android-client-tutorial/>). This was used to get the Mobile Application to subscribe to a topic, however its usefulness stopped there, and it was no longer used.



This layout is being used for the first mobile application prototype as it allows me to test three separate values being sent via the MQTT broker.

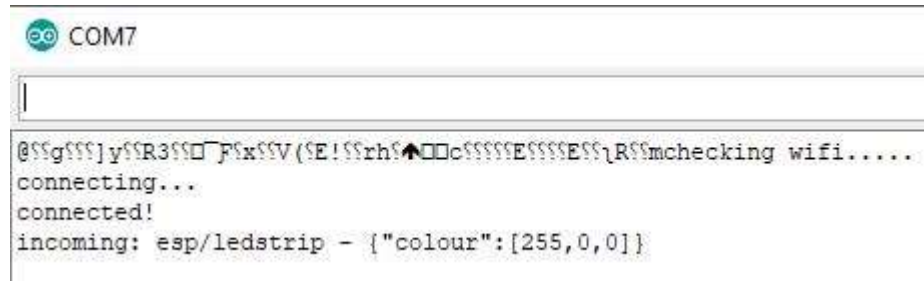


Here is a screenshot of the first prototype after the button, “red”, has been pressed. Where “Hello World” was on the previous picture (The wireframe from Android Studio) has been changed as the button was pressed to a JSON message that was sent via the MQTT Broker (shiftr.io). The JSON value contains the serialized RGB value for red, and when the other buttons are pressed, their respective RGB value is displayed as well. These buttons and values will be important for testing the LED Strip with multiple values.

**POST DEVELOPMENT NOTE:** Unit tests should have been developed for this however they were not implemented due to my unfamiliarity with Android Studio’s unit testing system and as such the only unit tests that have been developed test JSON serialization.

### Hardware Prototype 2a: Receiving MQTT

By adapting one of the example scripts from the MQTT library with our own values, a connection to the MQTT broker has been made. These values are the password and the username for the shiftr.io namespace that allows a connection to be made. To test this, the same button was pressed on the mobile app.



```
@[g[[]y[R3[]F[x[V(E![]rh[]Co[]E[]E[]R[]mchecking wifi.....
connecting...
connected!
incoming: esp/ledstrip - {"colour":[255,0,0]}
```

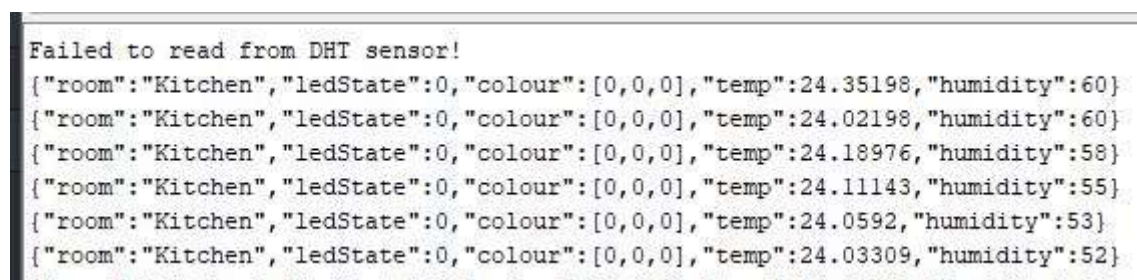
### Hardware prototype 2b: Revised System

The WeMos D1 mini functions as its own microcontroller, much like the Arduino, therefore, I have decided to cut the Arduino out completely and only use the WeMos D1 mini. This means that I need to change the LED strip library currently being used, FTRGBLED, to another library, FastLED. This is because the FTRGBLED library utilises the Arduino's avr architecture and won't compile when on the ESP8266 chip. Utilizing FastLED and the WeMos D1 mini, we can cut out the separate power supply meaning that the overall circuit will be smaller and easier to manage, and we can stop using the SoftwareSerial library as it will no longer be needed.

The new library has been easy to implement, and can be seen in this video (made and uploaded by me) <https://youtu.be/QRZyD9ZKtmI> showing an edited version of the blink library that blinks a green LED every half second. To see the research on the libraries no longer being used, see Appendix A.

### Hardware Prototype 3: DHT Sensor on WeMos D1 mini

The next iteration of the hardware prototype is the integration of the DHT sensor to the WeMos D1 mini and the JSON serialization of the results. The code is almost identical to the Arduino prototype that takes in and prints the temperature and humidity to the Serial monitor. The output should be a JSON string containing values for the temperature and humidity. The following image is the output from the ESP8266 program:



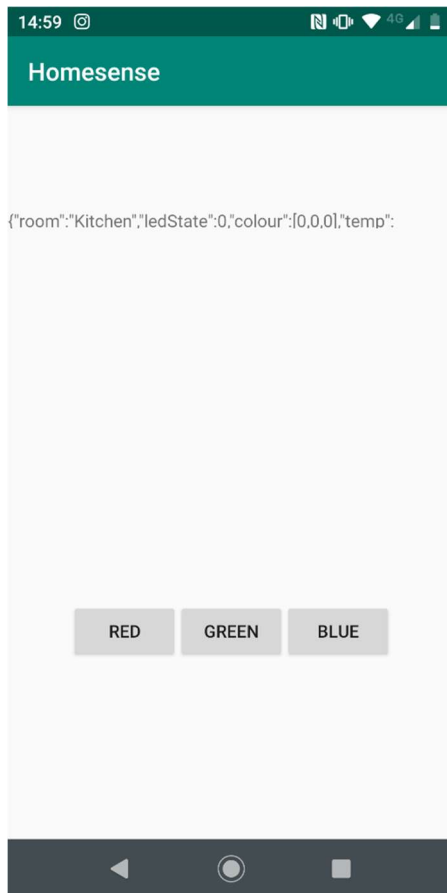
```
Failed to read from DHT sensor!
{"room":"Kitchen","ledState":0,"colour":[0,0,0],"temp":24.35198,"humidity":60}
{"room":"Kitchen","ledState":0,"colour":[0,0,0],"temp":24.02198,"humidity":60}
{"room":"Kitchen","ledState":0,"colour":[0,0,0],"temp":24.18976,"humidity":58}
{"room":"Kitchen","ledState":0,"colour":[0,0,0],"temp":24.11143,"humidity":55}
{"room":"Kitchen","ledState":0,"colour":[0,0,0],"temp":24.0592,"humidity":53}
{"room":"Kitchen","ledState":0,"colour":[0,0,0],"temp":24.03309,"humidity":52}
```

The JSON string contains all the information that will be sent to the mobile app in the next prototype.

### Hardware Prototype 4: One-way Communication

This prototype should be relatively easy as it only requires the addition of a single line of code (`client.publish("phone", payload);`). As the Mobile application can already accept incoming

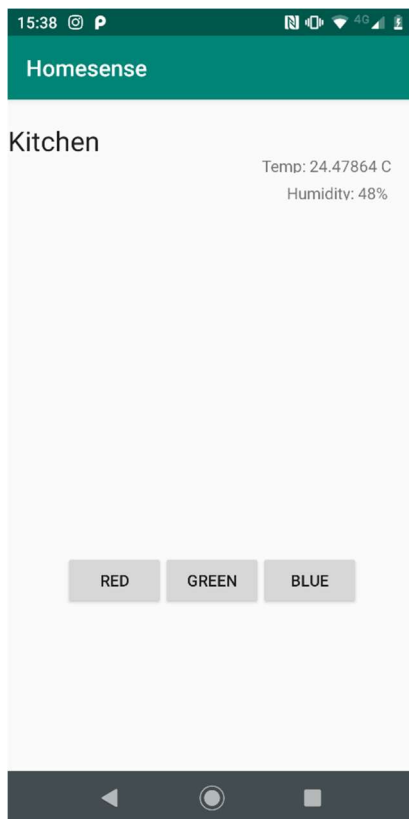
MQTT messages, the test for this is seeing whether the string from the previous prototype is displayed on the mobile application.



Unfortunately, the mobile application doesn't have enough space to display the full JSON string sent from the hardware prototype, however, this does show that one-way communication is taking place.

#### *Mobile Application Prototype 2: Processing a JSON input*

Processing the JSON message using Android Studio is relatively easy as Java has in-built JSON processing capabilities. Due to this I have separated the temperature, humidity and room name into their own TextViews. This is limited as it can only deal with one input, however I aim to fix that in one of the future prototypes.



Screenshot showing the processed JSON with the room, temperature and humidity given their own TextViews. This updates every time that the ESP8266 takes a reading meaning a live temperature and humidity are being displayed on the screen.

```
@Override
public void messageArrived(String topic, MqttMessage mqttMessage) throws Exception {
    Log.v( "tag: \"Debug\", mqttMessage.toString());
    String msgString = mqttMessage.toString();
    String var;
    try {
        JSONObject msgJSON = new JSONObject(msgString);
        var = "Temp: " + msgJSON.getString( name: "temp") + " C";
        temp.setText(var);
        var = "Humidity: " + msgJSON.getString( name: "humidity") + "%";
        humidity.setText(var);
        var = msgJSON.getString( name: "room");
        room.setText(var);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

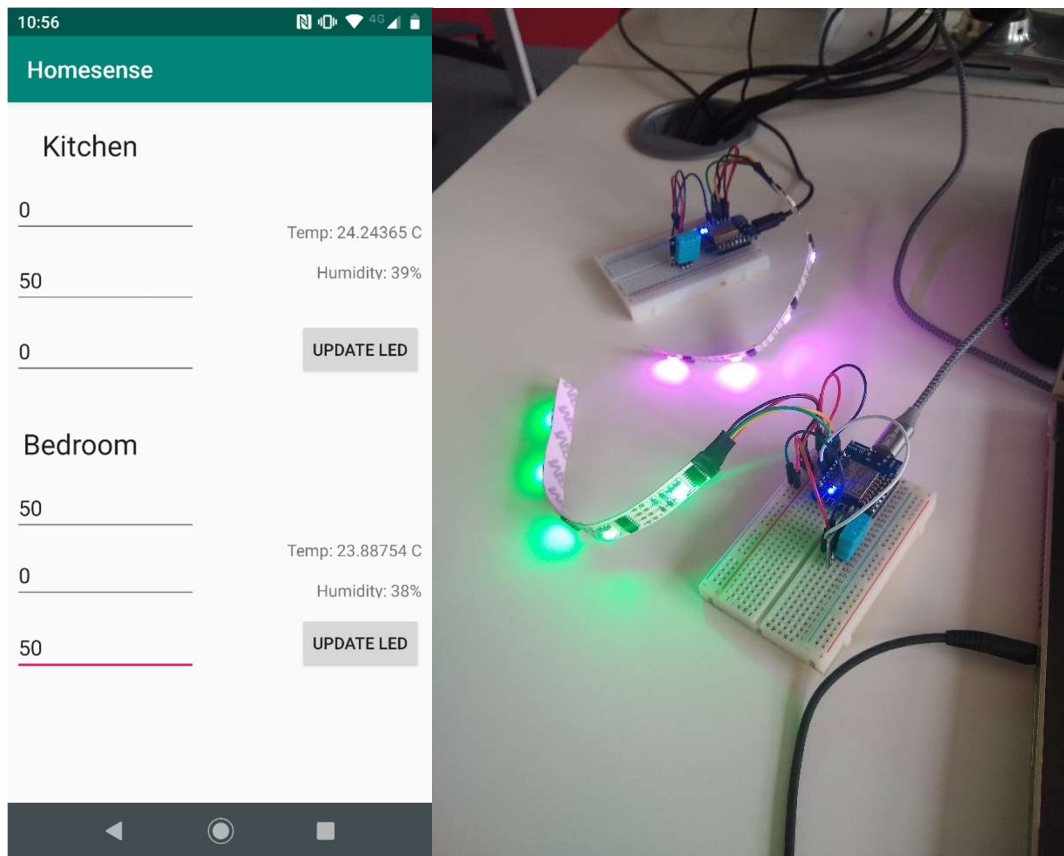
Screenshot of the code

that processes the incoming messages into their separate TextViews.

#### *Hardware Prototype 5/6/Final: Processing an incoming JSON message to turn on the LED strip*

This step turned into both prototypes 5 and 6 as step 6 was going to be adding multiple sensor 'agents', representing different. The two prototypes are together because they blended into each other and I am unsure where one of the prototypes ends and the next begins. This has allowed me to completely rearrange the layout of the application while the hardware has remained the same.

This version displays the information from 2 separate hardware nodes and allows for the control of both LED strips power state and colour in RGB values. It uses the free online shiftr.io MQTT broker to process the MQTT messages, connected through a Wi-Fi hotspot hosted from my laptop.



Images showing the information displayed on the mobile app, along with the resultant LED colours from the values sent.

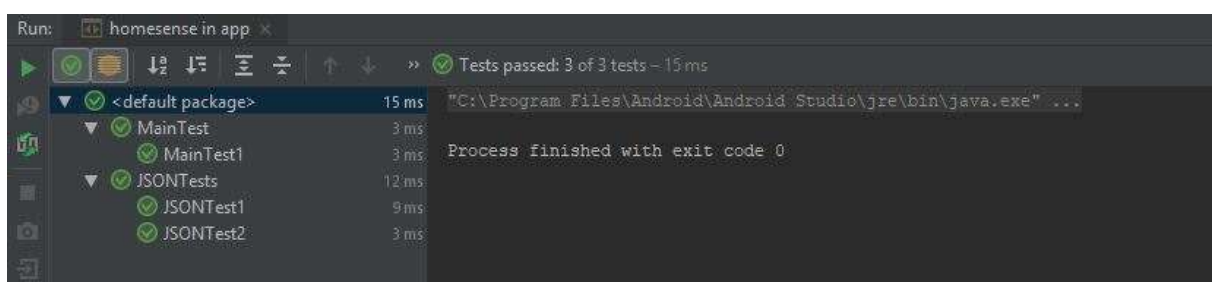
The version of both programs that was demonstrated is the final version. The mobile displays the temperature and humidity of 2 rooms and allows the user to control both LED strips separately. The code, along with this report, are held in the following GitHub Repository: <https://github.com/JBurton26/Temp-Monitor-LED.git>

## Part 6: Testing

### Test plan

For testing, a series of Unit Tests should be written, as well as visual testing as Arduino doesn't support unit testing. These Unit Tests should test various elements of the mobile applications functionality such as JSON and MQTT, however due to my unfamiliarity with Android Studio's unit testing and unit testing in general, only JSON serialization has been successfully unit tested.

The unit tests for testing the JSON serialization have all passed, as seen in the following image:



Additionally, the Android application features input validation for the various RGB values meaning that no false messages end up being sent to the circuits.

## Part 7: Problems Encountered

### LED Strip not working as expected

---

When using an example script provided by the study material, there was an error when labelling the clock and data wires for the LED strip meaning it no longer worked, despite it working in the weeks prior. This was solved by trying different configurations of wires.

### Compiling using newest version of ESP8266 board driver

---

There were many unsolvable issues when compiling code for the ESP8266 using a newer version of the library for another project. This issue was solved by backdating the library version to 2.4.2 wherein it compiled without issues.

### MQTT Message coming out as random characters

---

When sending an MQTT message via the broker, the message came out in a similar format to this: “{“colour”:[@ | 78c2f34 |]}. This was due to adding an entire array at once to a JSONArray in Android Studio and solved by adding an iterative function to add each array member.

### Connecting to the MQTT Broker

---

When connecting to the MQTT Broker, the processing sketch managed to connect less than a tenth of the time. This became too frustrating to deal with due to the lack of online documentation on MQTT usage within Processing 3+ and ended up being one of the main reasons behind swapping to using Android Studio.

### SoftwareSerial library not working as expected

---

During initial development, when both an Arduino and ESP8266 board were expected to be used, the SoftwareSerial library was used to transfer messages between the ESP8266 chip and the Arduino board. However, when implemented, many messages were lost or corrupted or did not react as expected. This was one of the factors that led to the removal of the Arduino board in favour of only using the ESP8266 board meaning that the SoftwareSerial Library no longer had to be used.

### When changing project to use only WeMos d1 mini

---

When the project was changed to only use the WeMos D1 mini, the example program for FTRGBLED would no longer compile. This was due to the ESP8266 having a different processor/microprocessor from the Arduino. Due to this, the LED controller library needed to be changed to a library that supports the ESP8266 architecture. The solution to this was in the library FastLED by FastLED (<https://github.com/FastLED/FastLED>) as it is compatible with the ESP8266 architecture.

## Part 8: Evaluation / Conclusion

### *Critical Evaluation of Project Deliverables*

Through development of the project, I have fulfilled all the M and S class items on the MoSCoW list. Due to this, I consider this project a success and I am happy with what has been developed.



I have implemented version control and the first part of a larger testing plan and tried to keep within high coding standards with documented code and regular commits to my GitHub repository for the project files. If I were to develop the project further, I would like to complete the C class items of the MoSCoW list as they mainly deal with how both parts of the project look rather than the functionality.

The main limitations of the prototype are:

- Only 2 rooms can be displayed and controlled by the application and these are hard-coded into the application due to my unfamiliarity with Android Studio. If I were to further develop this, I would like to add dynamic buttons that would appear for every room in the network and would only display the room name and temperature in the button text, but, when pressed would open a different screen with the ability to control the LED strip and displaying all of the relevant information about the selected room. This would accommodate a potentially unlimited number of rooms that can be displayed.
- The MQTT messages are reliant on an online MQTT broker that cannot always be relied on, for example if the broker is 'down' for maintenance etc., the entire system doesn't work. This could be solved by utilising a Raspberry Pi to act as a localised MQTT broker and changing the programs to connect via this broker. Additionally, the current prototype is reliant on the Wi-Fi hotspot hosted by my laptop and not a proper Wi-Fi network from a router, due to the difficulty of changing the SSID and Password values in the code to each Wi-Fi network compared to changing network on a laptop.

#### *Personal Evaluation*

I have enjoyed developing this project as I have had to learn about many new concepts and programming environments such as Android Studio. Even though I have spent upwards of 30 hours overall working on this project, I would like to have spent more time working on it, however my commitments to other university modules has limited my ability to do so. Due to both previous points, I am going to further develop this project in my own time, which will hopefully look good on my CV.



## References

### Libraries

---

1. Arduino's SoftwareSerial –
2. FastLED –
3. Freetronic's FTRGBLED –
4. Adafruit's Unified Sensor Library - [https://github.com/adafruit/Adafruit\\_Sensor](https://github.com/adafruit/Adafruit_Sensor)
5. Adafruit's DHT Sensor Library - <https://github.com/adafruit/DHT-sensor-library>
6. Eclipse's MQTT for Android - <https://github.com/eclipse/paho.mqtt.android>
7. Joel Gahwiler's MQTT for Arduino - <https://github.com/256dpi/arduino-mqtt>
8. ArduinoJSON - <https://arduinojson.org/>
9. ESP8266 Board - [http://arduino.esp8266.com/stable/package\\_esp8266com\\_index.json](http://arduino.esp8266.com/stable/package_esp8266com_index.json)

### Other Links

---

1. MQTT Android Client Tutorial by Wildan Maulana Syahidillah - <https://wildanmsyah.wordpress.com/2017/05/11/mqtt-android-client-tutorial/>
2. Adafruit DHT Sensors - <https://www.adafruit.com/product/386>

## Appendices

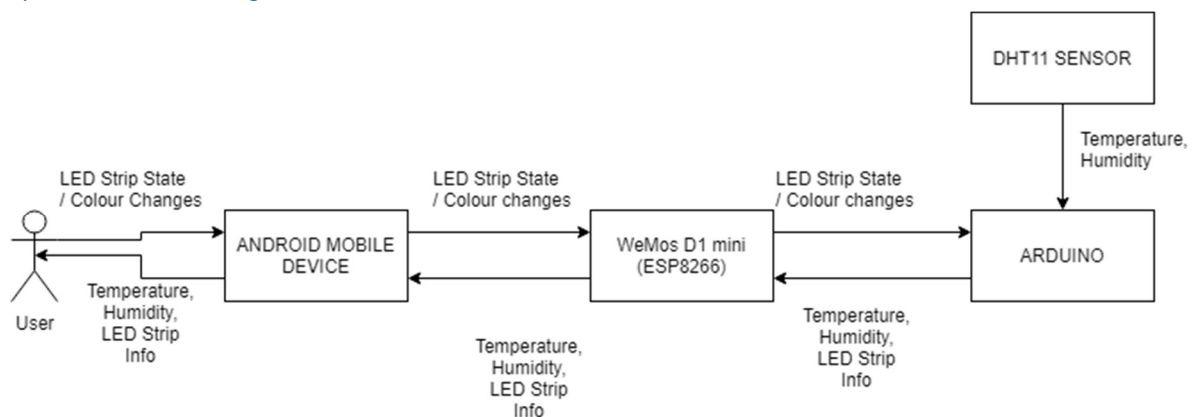
### Appendix A: Early development plan

#### Research

**To control the LED Strip** a separate library will be used to minimize the length of the code. The library FTRGBLED by freetronics (<https://github.com/freetronics/FTRGBLED/>) will be used to control the LED strips. This library was chosen as it deals with a variety of types of LED strips. A WS2801, the type being used of chip on the strip, makes each RGB LED addressable and individually controllable and is included in the types of LED strip controllers that can be controlled by the library.

**For testing purposes**, both the ESP8266 and the Arduino will remain plugged into the computer to monitor output, however when they are plugged in, there are issues with Sending messages via the Serial pins therefore a software solution is needed. This comes in the form of a pair of libraries both called SoftwareSerial. SoftwareSerial for the Arduino comes pre-installed on the ArduinoIDE, however a separate version of this must be downloaded for implementation on the ESP8266. The library must be downloaded online (<https://github.com/plerup/espsoftwareserial>) and installed manually. This library is implemented the same way as it would normally be implemented.

#### System overview diagram



#### List of system elements and their requirements

##### Main System Elements:

- Android Application** – Programmed using Processing 3+;
  - INPUTS (From Circuit): Temperature, Humidity, LED Strip info (ON/OFF/COLOUR), Fan info;
  - INPUTS (From User): LED Strip state/colour changes;
  - OUTPUTS (To ESP8266/ARDUINO): LED Strip state/colour changes;
  - OUTPUTS (To User): Temperature, Humidity, LED Strip info (ON/OFF/COLOUR), Fan info;
- ESP8266 Circuit** – Programmed using the Arduino IDE;
  - INPUTS (From Arduino): Temperature, Humidity, LED Strip Info (ON/OFF/COLOUR), Fan Info;

INPUTS (From Mobile): LED state/colour changes;

OUTPUTS (To Arduino): LED state/colour changes;

OUTPUTS (To Mobile): Temperature, Humidity, LED Strip Info (ON/OFF/COLOUR), Fan Info;

- **Arduino;**

INPUT (From Sensors): Temperature, Humidity;

INPUT (From ESP8266): LED state/colour changes;

OUTPUT (To ESP8266): Temperature, Humidity, LED Strip Info (ON/OFF/COLOUR), Fan Info;

The Arduino application will be written using Processing 3+ and will communicate via Wi-Fi to the ESP8266 and the ESP8266 will relay those messages to the Arduino via the Rx and Tx Serial wires.

#### *Challenges Expected*

The main difficulties will be within the Android Processing Application and this may result in the Android Studio IDE being used instead. This will require a large amount of new research into using Android Studio AND will likely mean that the final product application will be of a lower quality. However, the Android Processing application could be an early prototype on the Mobile Application side with a future version of the application being built on Android Studio.

Additionally, the format of the message sent by both sides of the wireless communication could prove an issue due to some corruption in the message as it is transferred due to loose wires and connections etc.

#### *Implementation*

Libraries implemented:

- ArduinoIDE's in-built SoftwareSerial Library and the ESP8266 SoftwareSerial Library addition(<https://github.com/plerup/espsoftwareserial>);

#### *Processing Android application:*

##### *Software specification*

For the mobile application, I am planning to use Processing 3+, however in the event that the app becomes more complicated than Processing can handle, then Android Studio will be used to develop the app as it is written using Java, the same language that Processing uses therefore should take the least time to familiarize myself with.

To communicate with the ESP8266, the application needs to use some form of MQTT protocol, with Processing using an adapted version of Eclipse Paho's MQTT library (<https://github.com/256dpi/processing-mqtt>).