IoT Coursework - Indoor Plant Monitoring Device

UP879148

Table of Contents

1.0 Introduction

1.1 What is IoT?

The Internet of Things (IoT) is a rapidly rising area of technology in which there is a system of devices that are able to collect data from the environment and transfer that data between each other over a wireless network. (H Kopetz, 2011)

1.2 Development of IoT

The concept of connecting devices together is not new. However, the term IoT was initially used by Kevin Ashton in 1999 in the context of using IoT devices to improve supply change management. However, since then the term IoT has become much more wide-ranging, referring to any connected systems that can sense data without human involvement. (J Gubbi et al, 2013)

In 2009 there were an estimated 0.9 billion IoT devices. However, by 2020 the number has increased to an estimated 26 billion IoT devices being used. (I Lee & K Lee, 2015) The primary driver behind this huge increase is advancements in technology such as in the availability and affordability of wireless chips.

The impact of IoT is very large and wide-ranging. IoT devices are being used by both individuals and corporations alike. For example, in homes IoT devices can be used for climate control, cooking and automatically lighting rooms. Whereas in corporations IoT devices are being used in fields such as agriculture for monitoring crops and soil known as “Agrotech” or in medicine for monitoring the conditions of ICU patients and instantly alerting doctors if there is an issue. (P Baruah et al, 2019)

1.3 Proposed IoT device and motivation

This project aims to create an IoT device that can be used in indoor at-home gardening to monitor the status of the environment that the plants are in. The variables that the IoT device aims to measure are atmospheric humidity, temperature, light levels, and soil moisture levels.

The motivation behind creating this IoT device is that often one of the hardest parts of indoor home gardening is ensuring that the plants are kept in the right conditions. Each variable that the device will monitor is essential for plant growth:

- **Temperature** - Impacts photosynthesis, transpiration, respiration, germination and flowering. Highlighting how important it is to get the correct temperature for a given plant.
- **Moisture & Humidity** - A primary component in photosynthesis and acts as a solvent for minerals. Without the right level, plants cannot grow effectively.
- **Light** - The amount of light a plant receives determines its capacity for producing food through photosynthesis. Therefore the factor is crucial for plant growth.

(Oregon State University, 2019)

2.0 Hardware and software tools used

This section will detail the hardware and software tools used to create the IoT device. It will describe their purpose, why the component is a good fit for this project and how they can be implemented. This section will also compare the component being used to other potential solutions.

Before the IoT device can be designed it is important to identify the constraints by which the device will be under. The reason for this is because unless the constraints and limitations of the device are identified, it is impossible to best determine what components are best for this device.

For this project, there are two types of constraint:

- **Technical constraints** - These are technical design decisions that impose a limit or restriction upon the system. For example, the amount of power a device needs.
- **Non-technical constraints** - These are non-technical design decisions that impose a limit or restriction upon the system. For example, cost.

Technical Constraints	Non-technical constraints
Device power - Most sensors work at or around 3.3V so the power supply needs to be appropriate for this.	Project scope - The project aims to create a device for indoor personal use. So anything that would not aid this is outside the scope of the project.
Device range - The range of the connection needs to be enough for a home network. Anything less isn't useful and anything more is wasteful.	Project cost - The device needs to be cost-effective. This means that expensive components won't be used. Generally, the cost of this project shouldn't exceed £30.
Device accuracy - For the device to be of any use the readings need to be fairly accurate. Generally to within a few units.	

2.1 NodeMCU/ESP8266

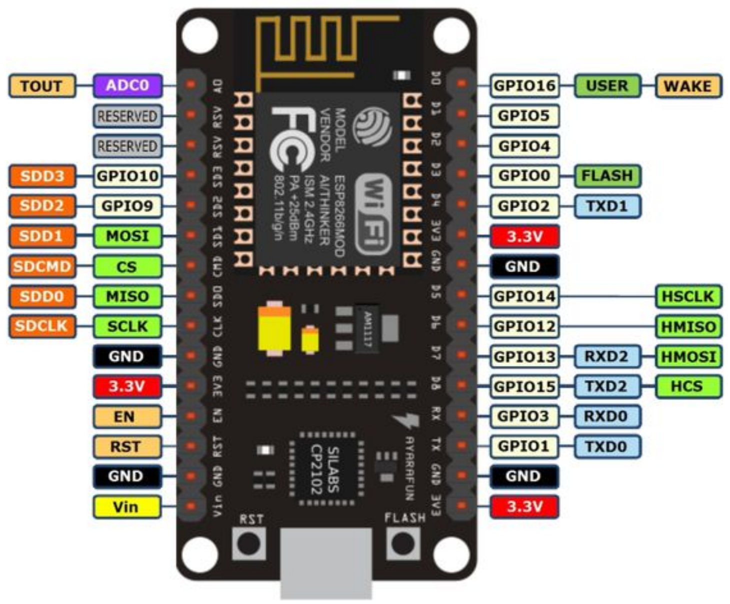


Figure 1 - NodeMCU/ESP8266 pin layout (ESP8266 NodeMcu Pinout, 2018)

This project will be using a NodeMCU/ESP8266 as the microcontroller for the IoT device. The NodeMCU/ESP8266 is a low-cost open-source microcontroller that also contains an ESP8266 integrated Wi-Fi chip. (Saputra, L. K., & Lukito, Y, 2017) The NodeMCU/ESP8266 also contains several pins, shown in figure 1. Pins such as GPIO can be used for data transfer between the microcontroller and other components. Whereas the 3.3V pin can be used to power the components. (NodeMCU Documentation, 2021)

Factors that make the NodeMCU/ESP8266 is ideal for this project:

- Low-cost (£3.61).
- Wifi-compatible.
- Supports Lua scripts.
- A large number of diverse pins.
- 3v3 power supply.
- Lots of modules.

Another approach that was considered was to use an Arduino Uno Rev3 microcontroller. This microcontroller is similar to the NodeMCU in that it provides a similar pin layout and uses an ESP8266 wifi chip for wireless connectivity. While good, the NodeMCU is superior for this project because it is cheaper, has more processing power (32-bit compared to 8-bit), more flash memory (4MB compared to 32KB) and more RAM (128KB compared to 2KB). (Arduino, Arduino Uno Rev3, 2021)

2.2 Breadboard and Jumper wires

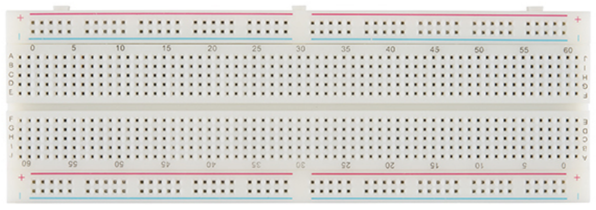


Figure 2 - Breadboard

Figure 3 - Jumper wires

A breadboard and jumper wires will be used to connect the components of the IoT device. The breadboard is useful because it enables the components to be connected by simply plugging in the jumper wires into it, removing the need to solder. While using this method would not be appropriate for a real IoT device it will allow for the creation of a prototype device that can be tested as a proof of concept which is sufficient for this project.

Factors that make the breadboard and jumper wires ideal for this project:

- Low-cost (£1.80).
- Easy to make changes.
- No soldering.

The only alternative to this method was to solder. While I do have a soldering iron it is designed for larger projects rather than the more precise pin soldering which would be required by this project. Overall, soldering is just unnecessary for creating a proof-of-concept device as using a breadboard is less prone to error, easy to make changes and cheaper. So this method was not used.

2.3 Temperature and humidity sensor

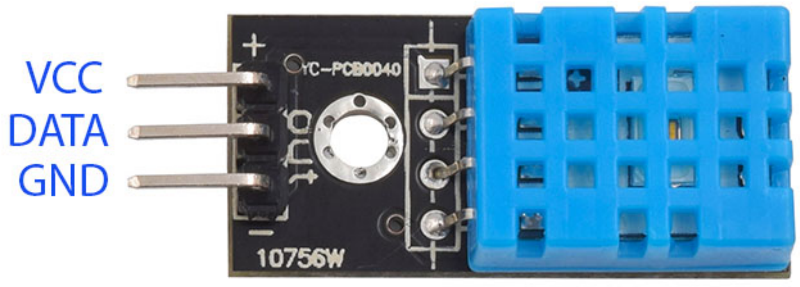


Figure 3 - DHT11 (DHT11 humidity and temperature sensor module, 2021)

To measure the temperature and humidity of the atmosphere the IoT device will be using a DHT11 temperature and humidity sensor, shown in figure 4. The DHT11 is a 3 pin sensor that can send real-time data to the microprocessor. The DHT11 has a temperature range of between 0°C to 50°C accurate to +-1°C and a humidity range of 20%-90% at 25°C accurate to +-4%. (*DHT11 datasheet*) The context of this project is an amateur at-home plant monitoring these measurement and accuracy ranges are sufficient. It is unlikely that the room the plant will be in will ever be below 0°C or above 50°C and the +-1°C is not likely to matter. Similarly, it's unlikely that the humidity will be out of the 20%-90% range and the +-4% is unlikely to matter.

DHT11 pin configuration:
1. VCC power supply pin.
2. Data pin - connected to GPIO pin for data transfer.
3. GND ground pin.

To read the data from the DHT11 the IoT device will be using the DHT module for the NodeMCU. This module allows for easy reading of a DHT11 sensor. To read the sensor, there is a function called dht.read(pin) which takes the value of the data pin connected to the DHT11. This function returns the status, temperature and humidity values. (Dht - NodeMCU Documentation, 2021) When implemented with a timer this allows us to get a constant reading of temperature and humidity information.

Factors that make the DHT11 is ideal for this project:
- Low-cost (£2.99).
- Simple pin layout.
- Provides accurate enough temperature and humidity information.
- Existing module support.

The project could also be implemented with a DHT22 which is an improved version of the DHT11 however, it is more expensive and has an unnecessary capture range for this project of -40°C to 80°C. So, due primarily to availability the DHT11 is more suitable.

2.4 BH1750 Light Intensity Sensor

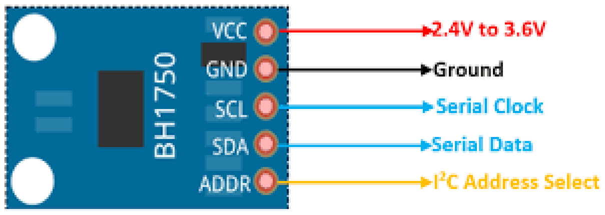


Figure 4 - BH1750 light intensity sensor. (Components 101, 2021)

This project will use the BH1750 digital ambient light sensor to measure the light levels in the environment. The BH1750 uses 16 bits and thus is capable of providing light levels of between 1 and 65535 Lux. To transmit the data the BH1750 uses I2C communication to transmit sensory information to the microcontroller. (Gao, J et al. 2017) This is ideal as the NodeMCU firmware installed on our microcontroller already contains an I2C module.

BH1750 pin configuration:

1. VCC power supply pin.
2. GND ground pin.
3. SCL (serial clock line) - provides the clock pulse for the I2C communication.
4. SDA (serial data address) pin - provides data transfer for I2C communication.
5. ADDR (device address) pin - selects address when multiple modules are connected.

(*BH1750 datasheet*)

To read data from the BH1750 sensor the device will be using the BH1750 Lua module. This module provides useful functions which when given an SDA and SCL pin value can read the BH1750's output. (BH1750, 2021, NodeMCU documentation)

Factors that make the BH1750 ideal for this project:

- Low cost (£4.99).
- Allows for digital data transmission.
- Easy to implement with existing modules (BH1750 Lua and I2C C modules).
- Wide data capture range (1 -> 65535 Lux).

An alternative method of measuring the light level that was considered involved using a variable light resistor to measure. This method would change the resistance based on the light level, allowing us to get a reading. However, a variable light resistor uses analogue input. This is a problem as the soil moisture sensor also uses analog so it resulted in conflict meaning the method was not suitable for this device.

2.5 Soil Moisture Sensor

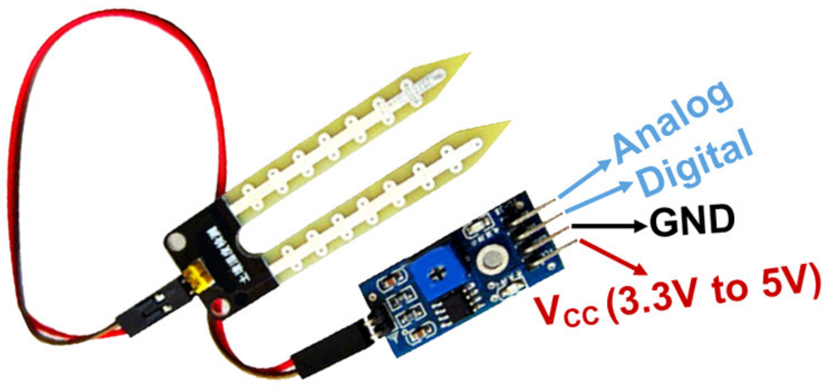


Figure 5 - Soil moisture sensor. (Components 101, 2021)

To measure the moisture levels in the soil the IoT device will be using a soil moisture sensor that can be connected to the NodeMCU. The sensor that will be used has four pins, there are two standard ground and power pins. However, the other two are sensor output pins and provide different approaches which can be used to gather data. The first sensor pin (A0) is an analog output pin that outputs an analog voltage which represents the soil moisture level. The second sensor pin (D0) provides a digital output signal which measures the soil moisture level through a variable resistor. (Soil Moisture Sensor Datasheet, 2020)

Soil moisture sensor pin configuration:
1. VCC power supply pin.
2. GND ground pin.
3. A0 analog output pin.
4. D0 digital output signal.

For this device, the soil moisture levels will be measured using the analog pin (A0). This will provide data on the moisture percentage in the soil. The reason for choosing analog is because it is more simple and provides all the necessary data.

Factors that make the soil moisture sensor ideal for this project:
- Low-cost (£1.79).
- Easy to configure - uses analog.
- Provides accurate soil information.

Alternative solutions were researched however, there is really only one type of soil-moisture sensor which is cheap, available and suitable for this project.

2.6 Light-emitting diodes

Figure 6 - 2-pin LEDs. (75x3mm Red Green Yellow Assorted Color LED Light Emitting Diodes AD, 2021)

2-pin LEDs will be used on the IoT device to provide visual feedback about the environmental data to the user. These LEDs are easy to implement with a NodeMCU. The device will contain a red and green LED. The green LED will be lit if the data received by the sensors is within the threshold set by the user and the red LED will be lit if it is not.

The LEDs in this project can be controlled using the PWM (pulse width modulation) module installed on the NodeMCU. PWM allows us to control the amount of power sent to a LED by using the on-off signal. These PWM signals can be used to adjust the intensity of the LED by controlling the duty-cycle, the higher the duty cycle percentage, the higher the intensity of the LED. (PWM Module - NodeMCU Documentation, 2021)

Factors that make the 2-pin LED ideal for this project:

- Low-cost.
- Energy-efficient.
- Provides bright light.
- Easy to program and implement (existing module support).
- Good for visual feedback.

Including LEDs is not a necessary part of the device. However, as the components were available I decided to include them as a good way of providing visual feedback to the user on the status of the IoT device.

2.7 Resistors

The IoT device will be using a 10k resistor connected to the power supply of each component. The resistor is used to cap the current flow in the circuit, this is necessary to prevent a short-circuit of the device or to prevent one of the LEDs from blowing up. (REF)

2.8 ESPlorer

The IDE that will be used to program the LUA script that will run on the IoT device is ESPlorer. ESPlorer is an IDE built-in java that is specifically designed to create LUA scripts that can run on the NodeMCU/ESP8266. ESPlorer also allows for uploading of the script directly to the NodeMCU which allows for easy testing and implementation. (*ESPlorer documentation,* 2016)

2.9 MQTT Protocol

MQTT (Message Queuing Telemetry Transport) is a protocol developed to facilitate data transfer over wireless sensor networks (WSN). The protocol aims to overcome the difficulties of transferring data on a WSN meaning it needs to be bandwidth and energy-efficient as well as able to run on devices with limited resources. (Thangavel D et al, 2014)

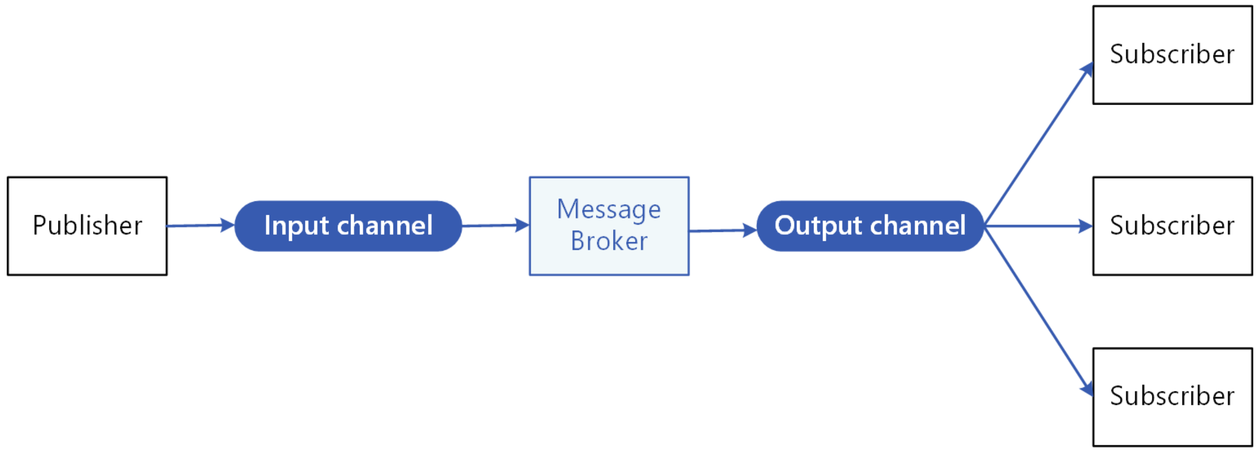


Figure 7 - Publish-subscribe architecture diagram. (Microsoft. 2018. Publisher-subscribe pattern)

MQTT uses a publish-subscribe architecture. The publish-subscribe architecture allows for devices to publish information to the broker. Devices can also subscribe to events that are outputted by the broker allowing the subscriber to be notified of any subsequent updates to that event. (Eugster P et al, 2014)

MQTT is ideal for this project because:

- TCP transmission.
- Provides publish/subscribe model.
- Easy to understand.
- Supports long connection.
- Existing module support.

(Naik, N. 2017)

CoAP (Constrained Application Protocol) was also considered and provides some benefits compared to MQTT. Such as, requiring less power which would be beneficial in this project. Furthermore, there is a CoAP module for the NodeMCU which would have made implementation easier. However, for this project, it was decided that it was not as suitable as an MQTT. This is because:

- The server-client model is more complex.
- A long connection requires a packet to be sent continuously.
- Less security than MQTT.
- Lower message size.

(Naik, N. 2017)

2.10 Adafruit I/O

Adafruit is a cloud-based message broker that will be used for the IoT device. Adafruit allows for the easy creation of a dashboard that can display real-time data from IoT devices. Additionally, Adafruit is also able to send data to IoT devices that are subscribed to it. In the context of this project, Adafruit will allow for the environmental data collected by the IoT device to be displayed on mobile or desktop devices as well as give the user the ability to provide input to the IoT device from their device. For example, allowing the user to turn off the device from their phone or enabling the user to set parameters. (Adafruit IO, 2021)

Another considered approach was to create my own GUI components using HTML/CSS and have the broker be a local server. However, while providing more flexibility, this approach would take considerably longer to implement. Consequently, it was decided that for this project using a pre-existing broker such as Adafruit is the best option.

2.11 Firmware

The tools found at (https://nodemcu-build.com/) were used to build the firmware for the device. This website was useful because it provides a selection of modules that can be included in the firmware. Once the modules are selected it will email you the firmware file to download. This is ideal because it means that no unnecessary modules are installed on the microcontroller. Meaning the limited space available is not being wasted.

Modules on the firmware:

- **DHT -** Needed to use the DHT11 sensor.
- **GPIO -** Needed to use the GPIO pins on the NodeMCU.
- **I2c -** Needed to facilitate data transfer for the soil moisture sensor.
- **PWM -** Needed to allow for changing LED brightness.
- **TMR -** Needed to use timers for polling data and connections.
- **Wifi -** Needed to allow the device to connect to the Wi-Fi.

(C Modules - NodeMCU Documentation, 2021)

2.12 Cost estimation

Table 1 - Project cost estimate

Device	**Cost**
NodeMCU/ESP8266	£ 3.61
Breadboard + Jumper wires	£ 1.80
DHT11	£ 2.99
BH1750	£ 4.99
LED kit	£ 5.79
Soil moisture sensor	£ 1.79
Total	**£ 20.97**

The price indicated in Table 1 shows the total cost spent on this project. However, an important point to consider is that the actual cost per IoT device will be significantly lower than this value. That is because components such as the LED kit and Jumper wires can only be purchased in bulk. The subsequent 20 or so IoT devices would not have these associated costs. The total average cost of these IoT devices after initial purchase is around £10 to £12 per device. This is, in my opinion, a good cost for this IoT device.

3.0 Project design

3.1 Architecture

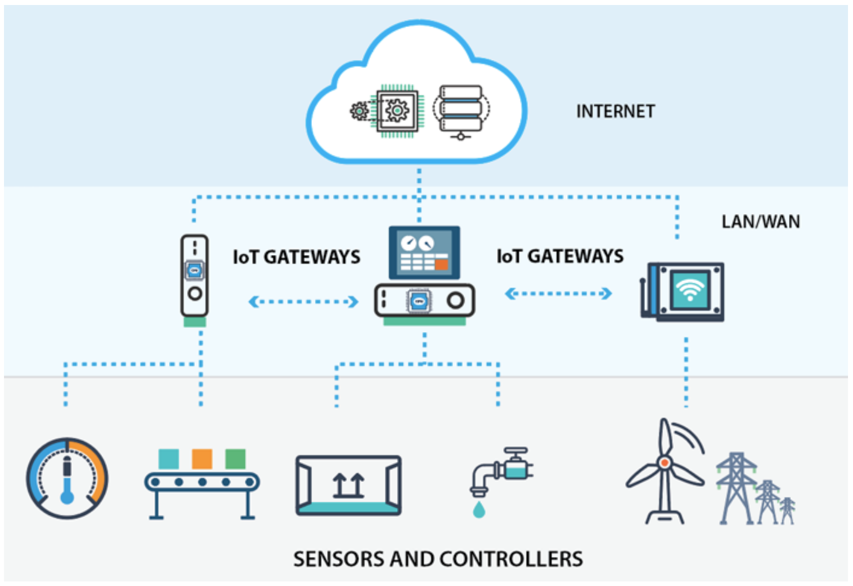


Figure 6 - Three-layered IoT architecture model (IoT architecture: 3 layers, 4 stages explained. 2020, https://www.zibtek.com/blog/iot-architecture)

The architecture model that will be used for the IoT model will be three-layered, shown in figure 6. The top layer is the application layer which is the internet or cloud services that are communicating with the IoT network (Adafruit). The middle layer is the network layer and contains the networking devices such as routers or switches. This allows IoT devices to communicate with each other and with the wider internet (local LAN). The bottom layer is the perception layer and contains the IoT devices which gather data from the environment via sensors (the IoT device). (Banu, N., & Sujatha, C. 2017. *IoT Architecture a comparative Study*)

The three-layered model is ideal for this project, because of the small scope and scale of the project and due to the relative experience I have in IoT. Other models were researched such as a five-layered model which contains additional business and processing layers. However, this model is more suitable for large scale operations within corporations and thus is not suitable for this project. (Sethi, P., & Sarangi, S. 2017)

3.2 Flow Diagram

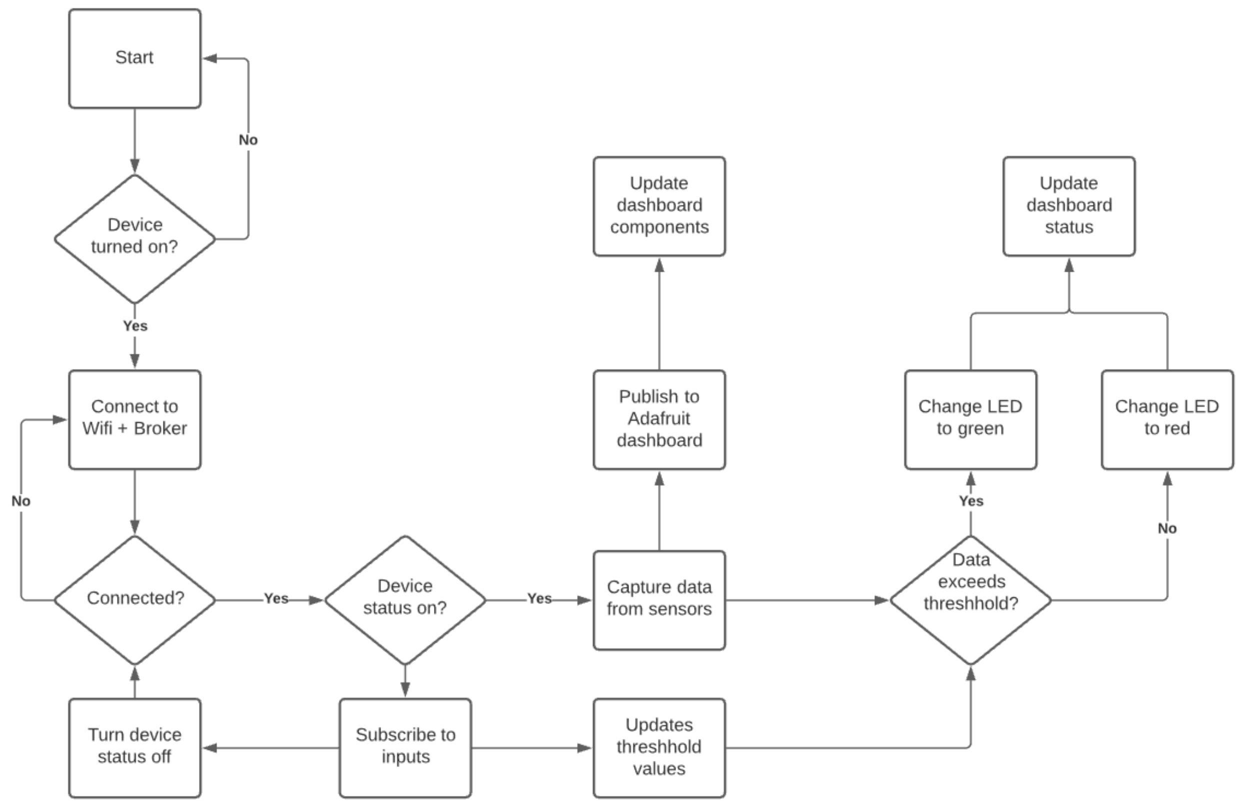


Figure 7 - Device flow diagram

Figure 7 shows the flow diagram created for the IoT device. The purpose of the flow diagram is to visualise the logical processes that are present in the device. This makes it easier to view the system and identify any areas that might need changing in the design before implementation.

Key information:

- First, the device checks if the condition is true. By default, it always will be when the device starts but if there is an error it might not be.
- There is a loop that will attempt to connect to the wifi and broker. If it is not successful then it will try again until a connection is established.
- The device then checks if the broker is set to “on”. If it is then it will subscribe to the inputs and will begin capturing data. If not it will loop until the broker is set to “on”.
- When the data is being captured it will be published to the relevant Adafruit dashboard components. Each time new data is collected it will be checked if it exceeds the user-set threshold. The LED will and dashboard status symbol will then be updated accordingly.
- The system can also receive inputs from the user. The user can turn the device capture off in which case the system will loop waiting for the device to be turned back on. Additionally, the user can set thresholds. When a threshold is changed the device will check it against existing data.

3.3 Pseudocode

```
Initialize connectionTimer
Initialize connected to False
Initialize threshold
Initialize on = false

on (start connectionTimer)
    if connection equals nil then
            print "Attempting to connect to WiFi"
    else
            print "Connected to: " + IP
            connect to MQTT client
            on = true
            stop connectionTimer
    end if
end timer

mqtt on "message"
            if message = "on" then
              on = true
    waitForConnection()
            else if message = "off" then
              on = false
            end if
end

function checkData (temperature, humidity, light, moisture)
    if temperature > threshold
            set redLED to ON
            set greenLED to OFF
            mqtt publish "Data not OK"
    else
            set redLED to OFF
            set greenLED to ON
            mqtt publish "Data OK"
    end if
end function

function collectData ()
    capture temperature + humidity
    print "Temperature: " + temperature + " Humidity: " + humidity
    capture light
    print "Light level: " + light
    capture moisture
    print "Soil moisture" + moisture
    mqtt publish temperature, humidity, light, moisture
    checkData(temperature, humidity, light, moisture)
end function


Initialize waitForConnection

on (start waitForConnection)
            if connected AND on = true then
            collectData()
            stop waitForConnection
    end if
end timer
```

3.4 Circuit Diagram

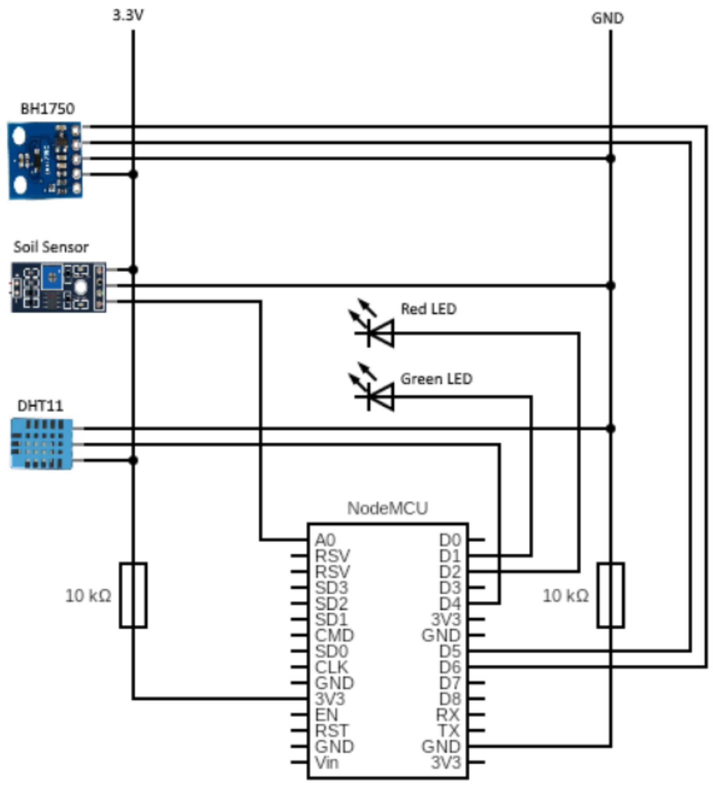


Figure 9 - Device circuit diagram

Figure 9 shows the circuit diagram for the IoT device. The reason for creating this circuit diagram is to help visualise the structure of the device and how all the components are connected to each other.

Key information:

- The 3v3 power supply pin is connected to each sensor device. This is done by connecting the power supply to the power rails on the side of the breadboard, allowing for efficient connection to the devices. There is also a 10k resistor to prevent short-circuiting.
- The GND pin is connected to the breadboard's power rails on the other side of the board from the 3v3 power supply. Similarly, this allows for the efficient connection of the GND pin to the sensors and to the LEDs. There is also a 10k resistor to prevent the LEDs from short-circuiting.
- The green LED and red LED are connected to the D1 and D2 GPIO pins respectively. Allowing for controlling the duty cycle using these pins.
- The BH1750 is connected to the GPIO pins D5 and D6 to enable SCL and SDA data transmission respectively through these pins.
- The soil sensor is connected to A0 to enable analog transmission through this pin.

4.0 Project implementation

4.1 IoT device

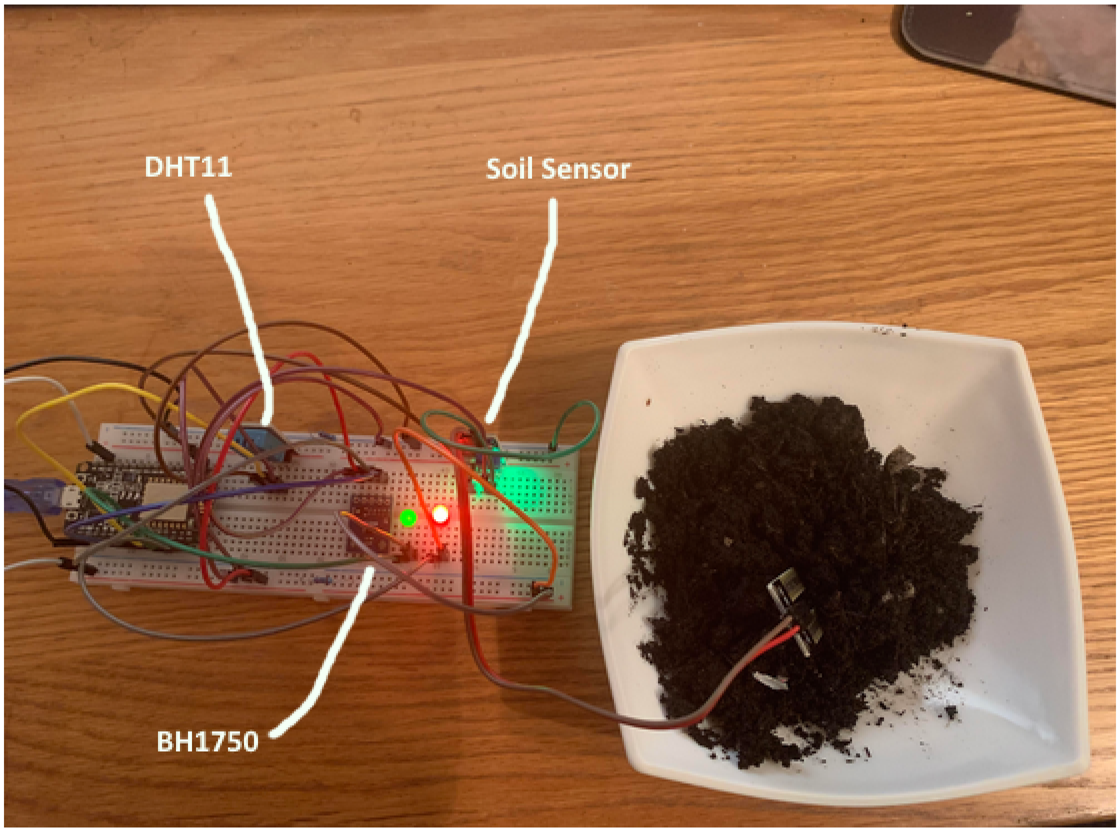
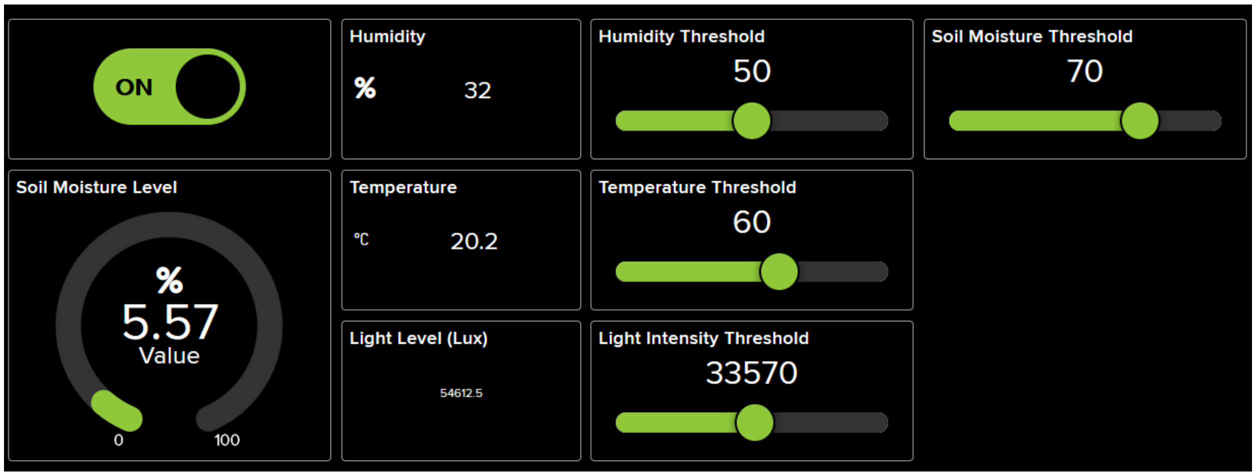


Figure 10 - IoT device

4.2 Adafruit dashboard


ON

Humidity
% 32

Humidity Threshold
50

Soil Moisture Threshold
70

Soil Moisture Level
%
5.57
Value
0 100

Temperature
℃ 20.2

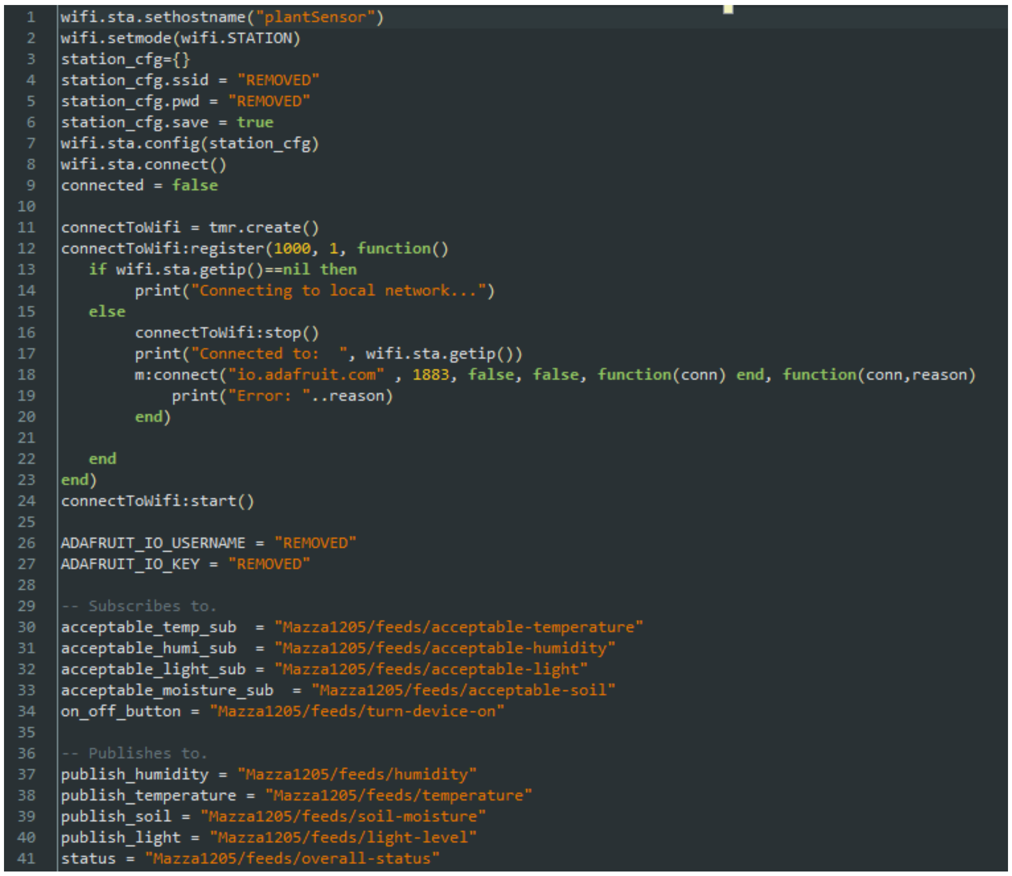
Temperature Threshold
60

Light Level (Lux)
54612.5

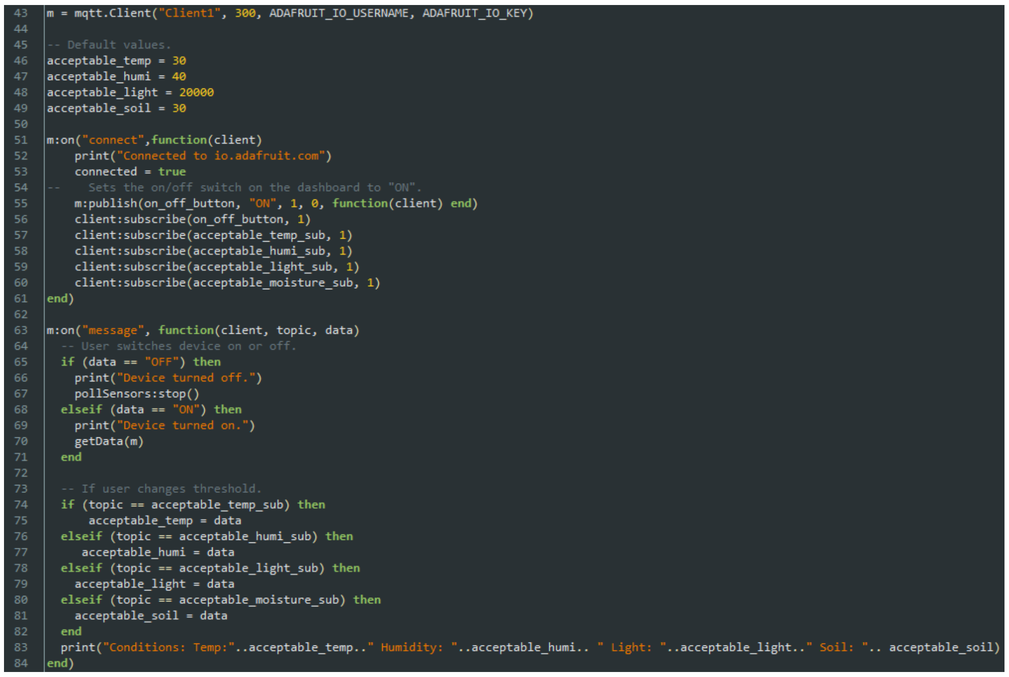
Light Intensity Threshold
33570

Figure 11 - Adafruit Dashboard

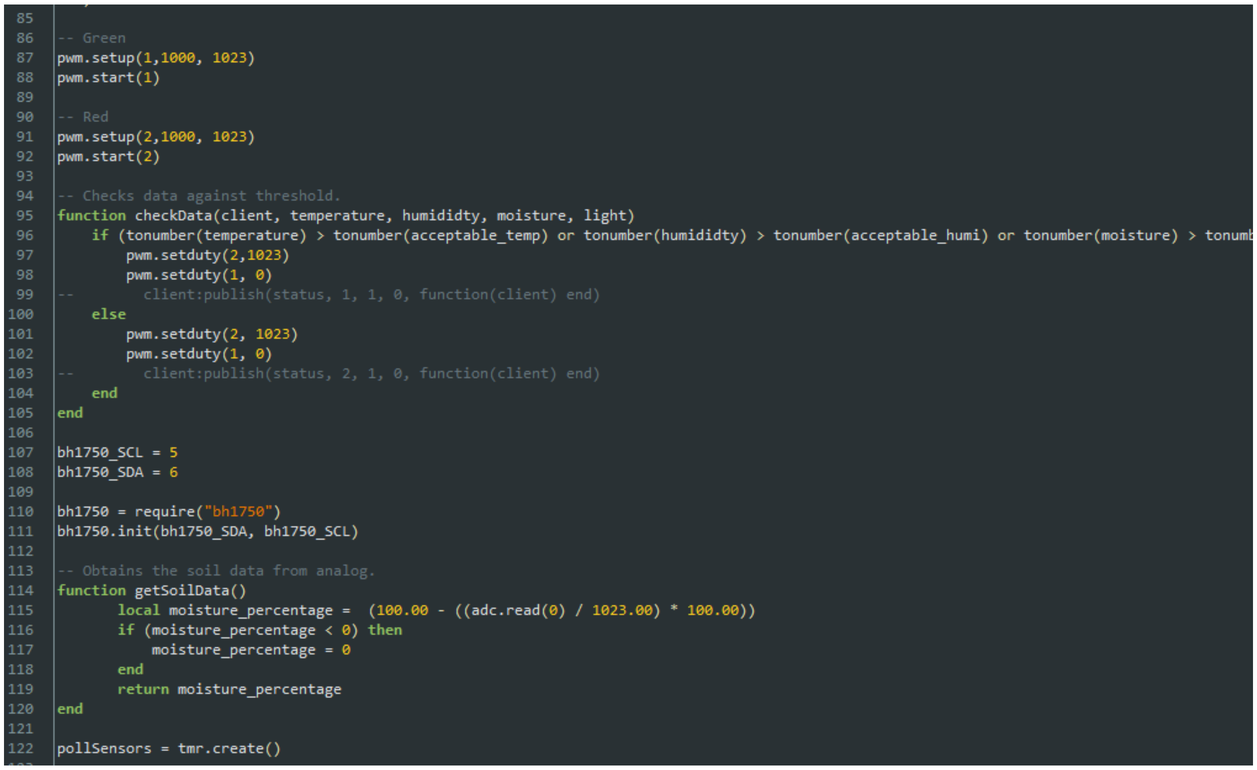
4.3 Device code



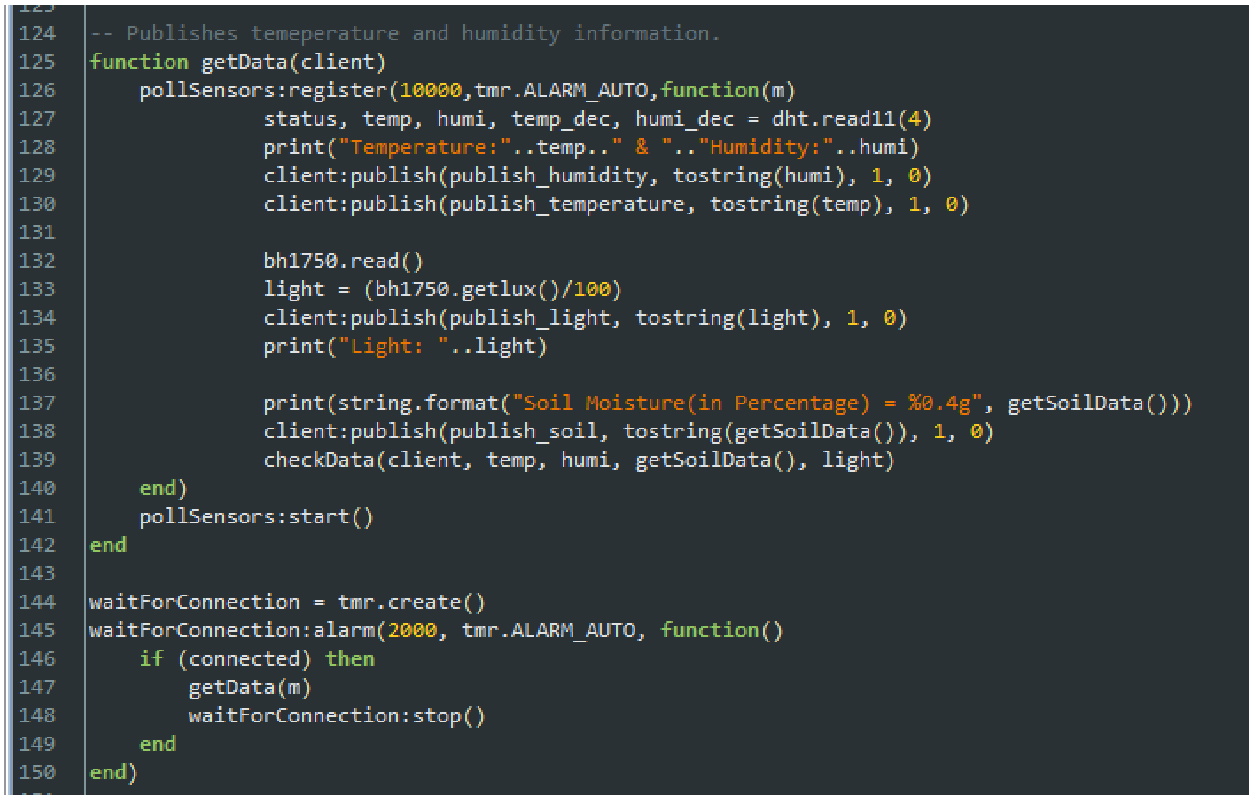
```lua
wifi.sta.sethostname("plantSensor")
wifi.setmode(wifi.STATION)
station_cfg={}
station_cfg.ssid = "REMOVED"
station_cfg.pwd = "REMOVED"
station_cfg.save = true
wifi.sta.config(station_cfg)
wifi.sta.connect()
connected = false

connectToWifi = tmr.create()
connectToWifi:register(1000, 1, function()
   if wifi.sta.getip()==nil then
        print("Connecting to local network...")
   else
        connectToWifi:stop()
        print("Connected to:  ", wifi.sta.getip())
        m:connect("io.adafruit.com" , 1883, false, false, function(conn) end, function(conn,reason)
            print("Error: "..reason)
        end)

   end
end)
connectToWifi:start()

ADAFRUIT_IO_USERNAME = "REMOVED"
ADAFRUIT_IO_KEY = "REMOVED"

-- Subscribes to.
acceptable_temp_sub  = "Mazza1205/feeds/acceptable-temperature"
acceptable_humi_sub  = "Mazza1205/feeds/acceptable-humidity"
acceptable_light_sub = "Mazza1205/feeds/acceptable-light"
acceptable_moisture_sub  = "Mazza1205/feeds/acceptable-soil"
on_off_button = "Mazza1205/feeds/turn-device-on"

-- Publishes to.
publish_humidity = "Mazza1205/feeds/humidity"
publish_temperature = "Mazza1205/feeds/temperature"
publish_soil = "Mazza1205/feeds/soil-moisture"
publish_light = "Mazza1205/feeds/light-level"
status = "Mazza1205/feeds/overall-status"
```



```lua
m = mqtt.Client("Client1", 300, ADAFRUIT_IO_USERNAME, ADAFRUIT_IO_KEY)

-- Default values.
acceptable_temp = 30
acceptable_humi = 40
acceptable_light = 20000
acceptable_soil = 30

m:on("connect",function(client)
    print("Connected to io.adafruit.com")
    connected = true
--    Sets the on/off switch on the dashboard to "ON".
    m:publish(on_off_button, "ON", 1, 0, function(client) end)
    client:subscribe(on_off_button, 1)
    client:subscribe(acceptable_temp_sub, 1)
    client:subscribe(acceptable_humi_sub, 1)
    client:subscribe(acceptable_light_sub, 1)
    client:subscribe(acceptable_moisture_sub, 1)
end)

m:on("message", function(client, topic, data)
  -- User switches device on or off.
  if (data == "OFF") then
    print("Device turned off.")
    pollSensors:stop()
  elseif (data == "ON") then
    print("Device turned on.")
    getData(m)
  end

  -- If user changes threshold.
  if (topic == acceptable_temp_sub) then
      acceptable_temp = data
  elseif (topic == acceptable_humi_sub) then
     acceptable_humi = data
  elseif (topic == acceptable_light_sub) then
    acceptable_light = data
  elseif (topic == acceptable_moisture_sub) then
    acceptable_soil = data
  end
  print("Conditions: Temp:"..acceptable_temp.." Humidity: "..acceptable_humi.. " Light: "..acceptable_light.." Soil: ".. acceptable_soil)
end)
```



```lua

-- Green
pwm.setup(1,1000, 1023)
pwm.start(1)

-- Red
pwm.setup(2,1000, 1023)
pwm.start(2)

-- Checks data against threshold.
function checkData(client, temperature, humididty, moisture, light)
    if (tonumber(temperature) > tonumber(acceptable_temp) or tonumber(humididty) > tonumber(acceptable_humi) or tonumber(moisture) > tonumb
        pwm.setduty(2,1023)
        pwm.setduty(1, 0)
--          client:publish(status, 1, 1, 0, function(client) end)
    else
        pwm.setduty(2, 1023)
        pwm.setduty(1, 0)
--          client:publish(status, 2, 1, 0, function(client) end)
    end
end

bh1750_SCL = 5
bh1750_SDA = 6

bh1750 = require("bh1750")
bh1750.init(bh1750_SDA, bh1750_SCL)

-- Obtains the soil data from analog.
function getSoilData()
       local moisture_percentage =  (100.00 - ((adc.read(0) / 1023.00) * 100.00))
       if (moisture_percentage < 0) then
           moisture_percentage = 0
       end
       return moisture_percentage
end

pollSensors = tmr.create()
```



```lua
-- Publishes temeperature and humidity information.
function getData(client)
    pollSensors:register(10000,tmr.ALARM_AUTO,function(m)
              status, temp, humi, temp_dec, humi_dec = dht.read11(4)
              print("Temperature:"..temp.." & ".."Humidity:"..humi)
              client:publish(publish_humidity, tostring(humi), 1, 0)
              client:publish(publish_temperature, tostring(temp), 1, 0)

              bh1750.read()
              light = (bh1750.getlux()/100)
              client:publish(publish_light, tostring(light), 1, 0)
              print("Light: "..light)

              print(string.format("Soil Moisture(in Percentage) = %0.4g", getSoilData()))
              client:publish(publish_soil, tostring(getSoilData()), 1, 0)
              checkData(client, temp, humi, getSoilData(), light)
    end)
    pollSensors:start()
end

waitForConnection = tmr.create()
waitForConnection:alarm(2000, tmr.ALARM_AUTO, function()
    if (connected) then
        getData(m)
        waitForConnection:stop()
    end
end)
```

5.0 Project discussion

5.1 Design Summary & Conclusion

The original purpose of this device was to provide a solution to monitor the environmental conditions of plants within an indoor home environment. This is important as one of the biggest challenges facing home gardening is ensuring that the plant is in the right conditions.

To achieve this the device has three environmental sensors. A DHT11 sensor to measure temperature and humidity. A BH1750 sensor to measure light intensity and a sensor that can measure soil moisture levels. These conditions were selected to be monitored as they are all key factors in determining plant growth.

Each sensor was chosen because of its low cost, accurate data capture, ease of implementation and existing support.

Overall, the implementation of the IoT device can be considered a success as the device meets all of the design goals outlined. Each variable can be effectively measured by the device, solving the problem of monitoring plants in an indoor home setting.

5.2 Pitfalls encountered

5.2.1 Memory

An issue that was encountered was the microcontroller running out of memory to run the IoT device. The error message stated: “Lua error: not enough memory”. After some research, I figured out that the issue was caused by my script being too long. This is also exacerbated by the additional Lua modules and firmware that were loaded onto the NodeMCU for this project.

This issue was resolved by reducing the overall size of the script. The efficiency of the script was improved by eliminating unnecessary lines of code and improving logic so that it took up less space.

However, the issue does mean the scope of the project is somewhat limited by the size issue. However, the device still achieves all desired aims so I do not really think that this is an important concern.

5.2.2 Firmware and modules

An issue that was encountered was a problem with not having installed the correct firmware and modules. The firmware that was initially installed on the did not contain the I2C module on it. This meant that when I tried to use the soil moisture sensor I was getting error messages. While it was a simple issue to resolve in hindsight, simply installing the correct firmware. At the time, due to my lack of relative experience in making IoT devices, it took a bit of time to figure out.

5.2.3 Data throttling

An issue that was encountered was data throttling on the Adafruit broker that was being used in the project. When too much data is being sent to Adafruit it will temporarily ban the device from sending anymore. The current maximum rate is 30 requests per minute. (Changes to Throttling in Adafruit IO, 2017) Initially, the device did more than this and kept getting throttled. To resolve this I increased the publish timer from 5 seconds to 10 seconds so less data was being sent every minute.

5.3 Future work and improvements

Future work for this project should aim around moving to a self-made broker rather than using a public broker such as Adafruit. This would provide several advantages to the device. Firstly, it would enable the creation of custom GUI components using HTML. This would allow for greater flexibility in what functions the GUI can perform as well as provide the ability to make the GUI more efficient.

Furthermore, future work could aim to expand the scope of the project. The initial scope aimed at indoor amateur plant growing. However, the scope could be changed to also monitor outdoor plants. The functionality of the device would not have to be changed to facilitate this. However, protection would have to be added to protect the components of the device from the elements such as wind and rain.

Finally, the project could be expanded to capture more variables thus providing a more complete picture of the environment to the user. One potential route for expansion would be to capture the soil nutrient levels. This could be implemented by using a Soil NPK Meter. This device can be connected to a microcontroller and can measure the nitrogen, phosphorus and potassium levels of the soil. All of which are important for plants. (Ramane, D. 2015)

References

Banu, N., & Sujatha, C. (2017). IoT Architecture a comparative Study. https://doi.org/10.12732/ijpam.v117i8.10

Baruah, P. D., Dhir, S., & Hooda, M. (2019). Impact of IoT in the current era. 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon). https://doi.org/10.1109/comitcon.2019.8862187

BH1750 Light sensor Pinout, features & datasheet. (n.d.). Components101. https://components101.com/sensors/bh1750-ambient-light-sensor

Bh1750. (2021). Overview - NodeMCU Documentation. https://nodemcu.readthedocs.io/en/release/lua-modules/bh1750/

Dht. (2021). Overview - NodeMCU Documentation. https://nodemcu.readthedocs.io/en/release/modules/dht/

DHT11 humidity and temperature sensor module. (2021). Smart Prototyping. https://www.smart-prototyping.com/DHT11-Humidity-and-Temperature-Sensor-Module

Environmental factors affecting plant growth. (2019, July 15). OSU Extension Service. https://extension.oregonstate.edu/gardening/techniques/environmental-factors-affecting-plant-growth

ESP8266 NodeMcu Pinout. (2018, June 15). ESP8266 Shop. https://esp8266-shop.com/esp8266-guide/esp8266-nodemcu-pinout/

ESPlorer – esp8266. (2016, August 15). esp8266. https://esp8266.ru/esplorer/

Eugster, P. T., Felber, P. A., Guerraoui, R., & Kermarrec, A. (2003). The many faces of publish/subscribe. ACM Computing Surveys, 35(2), 114-131. https://doi.org/10.1145/857076.857078

Gao, J., Luo, J., Xu, A., & Yu, J. (2017). Light intensity intelligent control system research and design based on automobile sun visor of BH1750. 2017 29th Chinese Control And Decision Conference (CCDC). https://doi.org/10.1109/ccdc.2017.7979192

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems, 29(7), 1645-1660. https://doi.org/10.1016/j.future.2013.01.010

IoT architecture: 3 layers, 4 stages explained. (2020, August 7). Custom Software Development Insights | Zibtek Blog. https://www.zibtek.com/blog/iot-architecture

Kopetz H. (2011) Internet of Things. In: Real-Time Systems. Real-Time Systems Series. Springer, Boston, MA. https://doi.org/10.1007/978-1-4419-8237-7_13

Lee, I., & Lee, K. (2015). The Internet of things (IoT): Applications, investments, and challenges for enterprises. Business Horizons, 58(4), 431-440. https://doi.org/10.1016/j.bushor.2015.03.008

Mortensen, L. (1986). Effect of relative humidity on growth and flowering of some greenhouse plants. Scientia Horticulturae, 29(4), 301-307. https://doi.org/10.1016/0304-4238(86)90013-0

Naik, N. (2017). Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. 2017 IEEE International Systems Engineering Symposium (ISSE). https://doi.org/10.1109/syseng.2017.8088251

Ramane, D., Patil, S., & Shaligram, A. (2015). Detection of NPK nutrients of soil using Fiber Optic Sensor. https://www.researchgate.net/profile/J_Tarafdar/post/Does_any_one_have_any_idea_about_NPK_Measurement_Sensor_which_are_available_in_the_market2/attachment/59d6515b79197b80779a9e54/AS%3A506974598111233%401497883571255/download/ACGT18.pdf

Saputra, L. K., & Lukito, Y. (2017). Implementation of air conditioning control system using REST protocol based on NodeMCU ESP8266. 2017 International Conference on Smart Cities, Automation & Intelligent Computing Systems (ICON-SONICS). https://doi.org/10.1109/icon-sonics.2017.8267834

Sethi, P., & Sarangi, S. (2017). Internet of Things: Architectures, Protocols, and Applications. https://doi.org/10.1155/2017/9324035

Thangavel, D., Ma, X., Valera, A., Tan, H., & Tan, C. K. (2014). Performance evaluation of MQTT and CoAP via a common middleware. 2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP). https://doi.org/10.1109/issnip.2014.6827678