Sheffield Hallam University
Department of Engineering
BEng (Hons) Electrical and Electronic Engineering

Activity ID	Activity Title	Laboratory Room No.	Level
Lab 106	Developing IoT applications with MQTT	4302	6

Semester	Duration [hrs]	Group Size	Max Total Students	Date of approval/review	Lead Academic
1	8	2	25	09-22	Alex Shenfield

Equipment (per student/group)

Number	Item
1	PC
1	Arduino kit

Learning Outcomes

	Learning Outcome
2	Demonstrate an understanding of the various tools, technologies and protocols used in the development of embedded systems with network functionality
3	Design, implement and test embedded networked devices

Developing IoT applications with MQTT

Introduction

Computing is about more than the PC on your desktop! Embedded devices are everywhere – from wireless telecommunications infrastructure points to electronic point of sale terminals. One definition of an embedded system is:

> "An embedded system is a computer system designed to perform one or a few dedicated functions often with real-time computing constraints."
>
> (http://en.wikipedia.org/wiki/Embedded_system)

In this series of labs you are going to be introduced to the open source Arduino platform – a cheap, simple to program, well documented prototyping platform for designing electronic systems. Although the Arduino platform started out with the Arduino Uno (based on the commonly used ATMega328 chip), other, more capable, boards have since been released. In this series of labs you will be using the Arduino MKR WiFi 1010 board – a Cortex M0+ based Arduino board with built in WiFi module.

The purpose of this lab session is to build on the network functionality explored during the last lab session to develop Internet-of-Things applications using the popular MQTT protocol.

Bibliography

There are no essential bibliographic resources for this laboratory session aside from this tutorial sheet. However the following websites and tutorials may be of help (especially if you haven't done much electronics previously or your digital logic and/or programming is a bit rusty):

- http://www.arduino.cc/
- http://www.ladyada.net/learn/arduino/index.html
- http://tronixstuff.wordpress.com/tutorials/

Methodology

Check that you have all the necessary equipment (see Figure 1)!

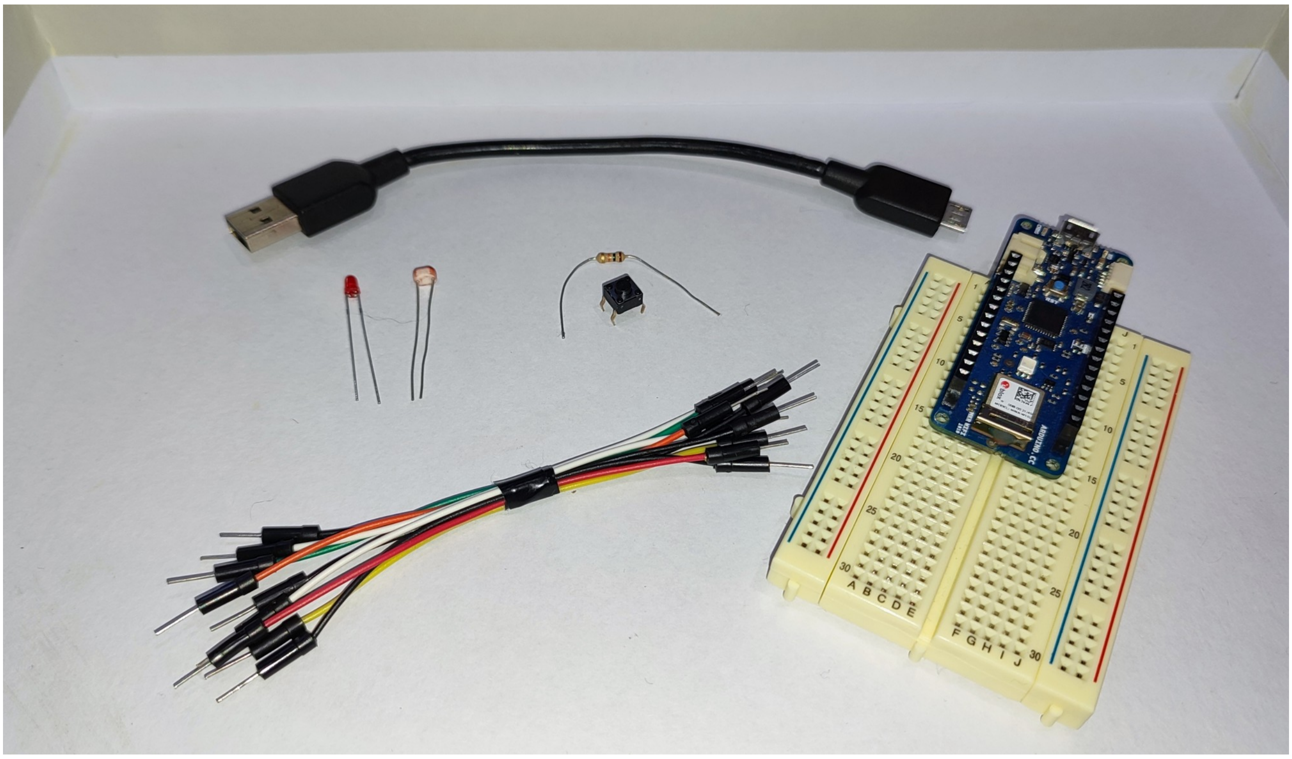

Figure 1 – The necessary equipment for this lab

Task 1

Last week we looked at using the Arduino MKR WiFi 1010 with the HTTP protocol – creating both a simple web server and RESTful webservices using HTTP's **request – response** paradigm. However, even the relatively light-weight REST web service architecture has limitations when it comes to embedded networked devices in the Internet-of-Things. For this reason, many Internet-of-Things devices (particularly those with strict resource constraints such as small micro-controllers) avoid HTTP and use alternative protocols such as CoAP and MQTT.

The official MQTT 3.1.1 specification[1] says:

> "MQTT is a Client Server publish/subscribe messaging transport protocol. It is light weight, open, simple, and designed so as to be easy to implement. These characteristics make it ideal for use in many situations, including constrained environments such as for communication in Machine to Machine (M2M) and Internet of Things (IoT) contexts where a small code footprint is required and/or network bandwidth is at a premium."

One of the key benefits in comparison to HTTP is that MQTT uses 10s of bytes in protocol headers, whereas HTTP can use 100s – 1000s of bytes for headers. Stanford-Clark and Nipper (1999) specified the following goals for MQTT:

- Simple to implement
- Provide a Quality of Service Data Delivery
- Lightweight and Bandwidth Efficient
- Data Agnostic
- Continuous Session Awareness

Another benefit of MQTT is the **publish – subscribe** model which reduces network traffic by allowing a server to push updates to subscribed clients when there is new information of interest available. Figure 2 illustrates this interaction.

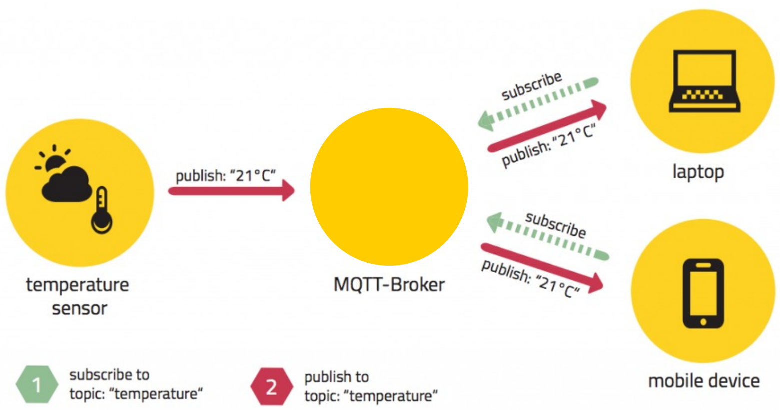

Figure 2 – MQTT's publish – subscribe model

1 http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

In this task we are going to write a simple Arduino application capable of posting a status message to a remote MQTT broker and receiving status updates back. We are going to use the Adafruit IO data analysis platform to send and receive our data – it is a user friendly way of developing web interfaces to visualise the data coming from Internet-of-Things devices.

First you will need to head over to https://io.adafruit.com/ and sign up for an Adafruit account. You can then join Adafruit IO and log in with your account. After logging in you will see a tab containing a list of dashboards (as in Figure 3).

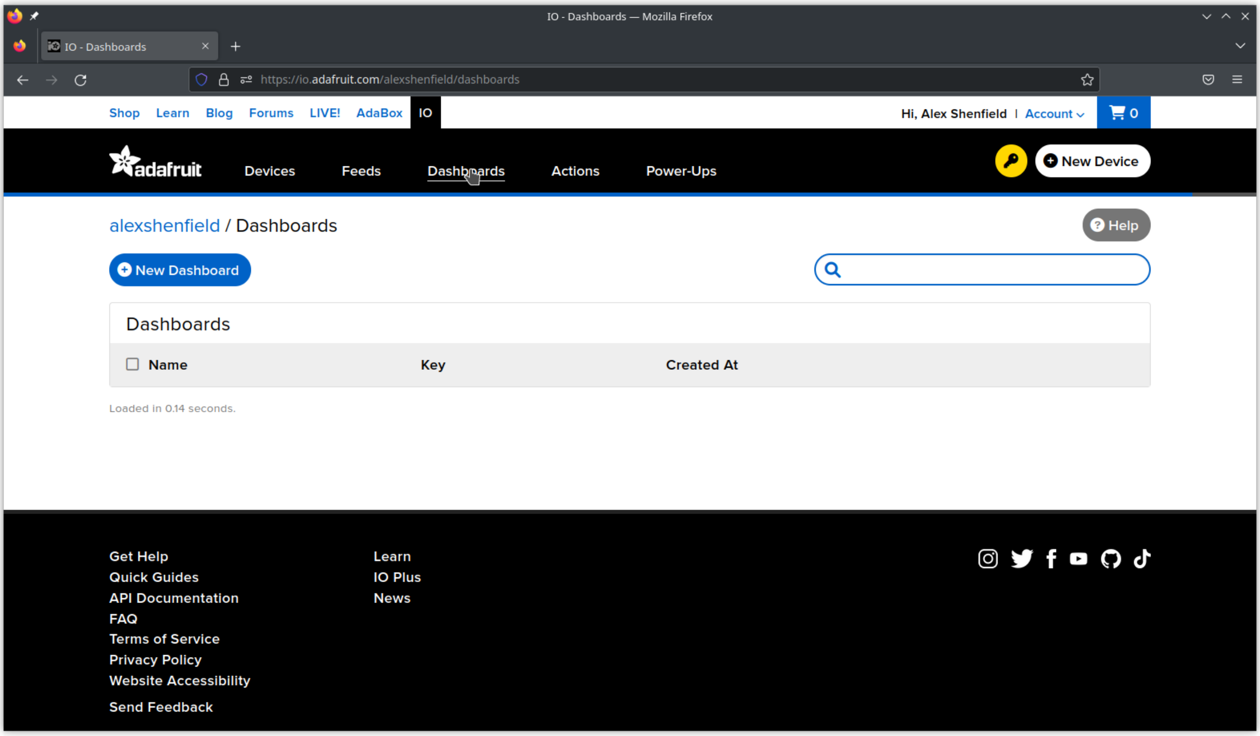

Figure 3 – The Adafruit IO dashboard list

For this lab session we will create a new dashboard and some new feeds for receiving the data via our embedded system. For more details on creating feeds and dashboards, see the getting started guide available at https://learn.adafruit.com/welcome-to-adafruit-io.

Note, the free version of Adafruit.io has some significant limitations in terms of the number of dashboards and feeds (i.e. MQTT topics) we can use. One particular limitation to be aware of is the rate limit (see details on page 9) – if you exceed this **you will get banned**. These limitations should be fine for the work we are doing this year – though it is not very hard to create your own MQTT broker and dashboard if needed[2].

2 In previous years we have used the free public brokers offered eclipse, mosquito, and hivemq – however, these are also not without issue!

From the "Dashboards" list, create a new dashboard (see Figure 4).

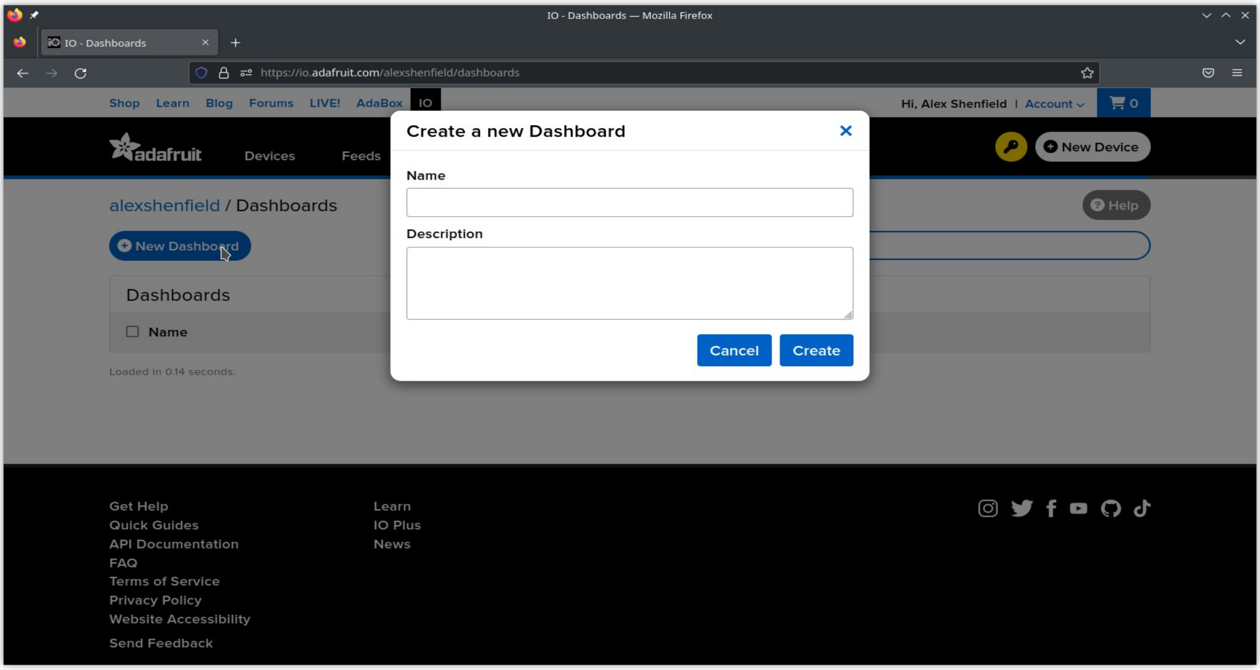

Figure 4 – Creating a new dashboard

Give the new dashboard a sensible name (I have chosen to call mine "My first IoT application") and then add a new block to it from the "Dashboard Edit Controls" button (see Figure 5).

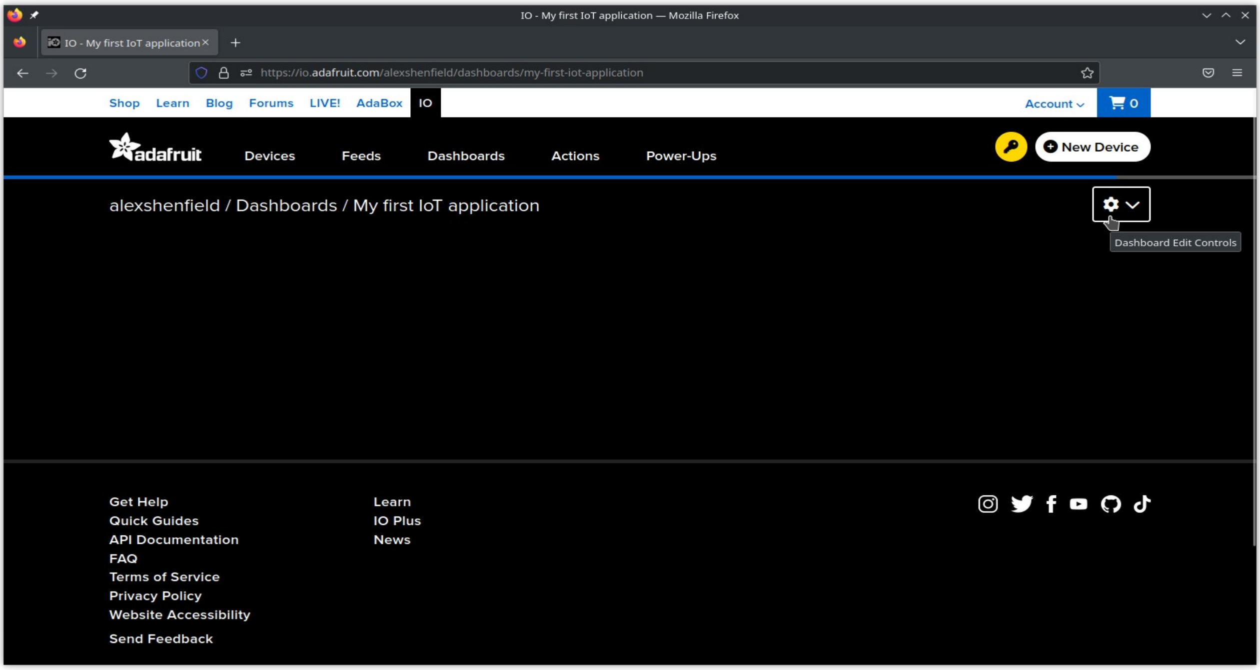

Figure 5 – A blank dashboard

We want to create a new stream block that we can use for displaying text sent to a feed (or multiple feeds). This will allow us to display our status messages. See Figure 6.

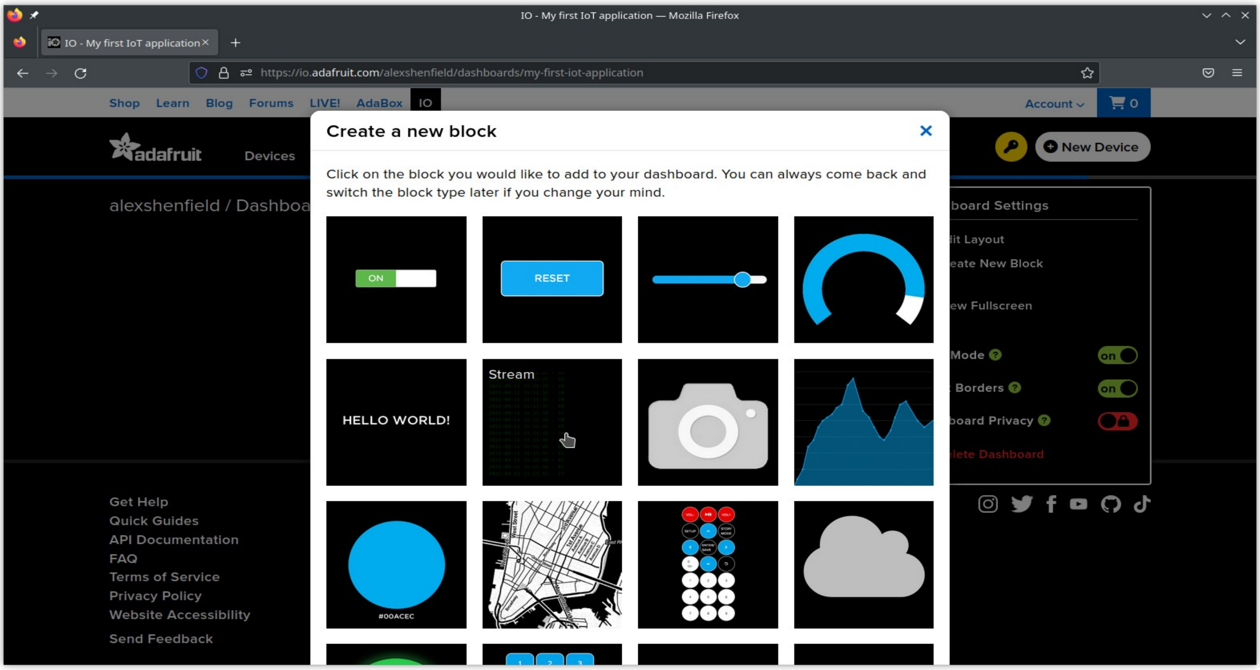

Figure 6 – Adding elements to our dashboard

In the wizard that follows, we want to set the stream block up to be attached to a "status-messages" feed. To do this you will need to type the feed name into the appropriate box and click the "Create" button. You can then add the feed to the stream block as in Figure 7.

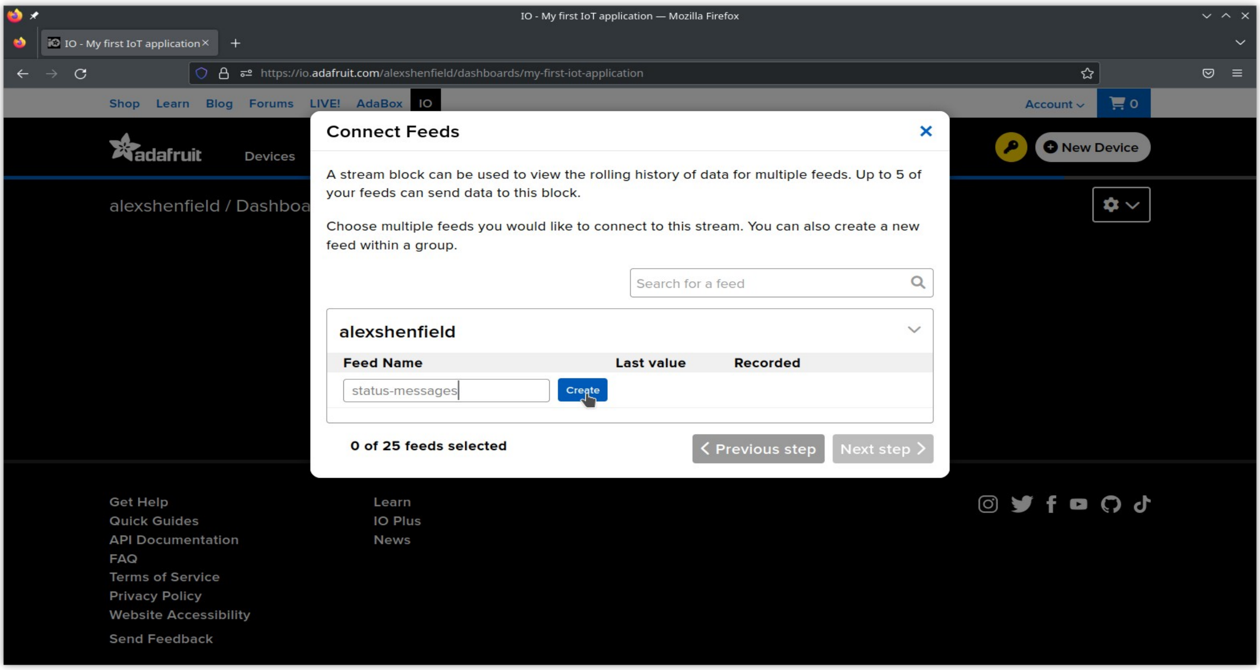

Figure 7 – Attaching a feed / topic to our stream block

Once you have done this you can alter the settings for the stream block – I like the defaults (who doesn't like green text on a black background?!) but I also want the feed to show real times data (so I set “Hours of History” to 0). Once you have created the block you can resize it appropriately from the “Edit Layout” option – see Figure 8.

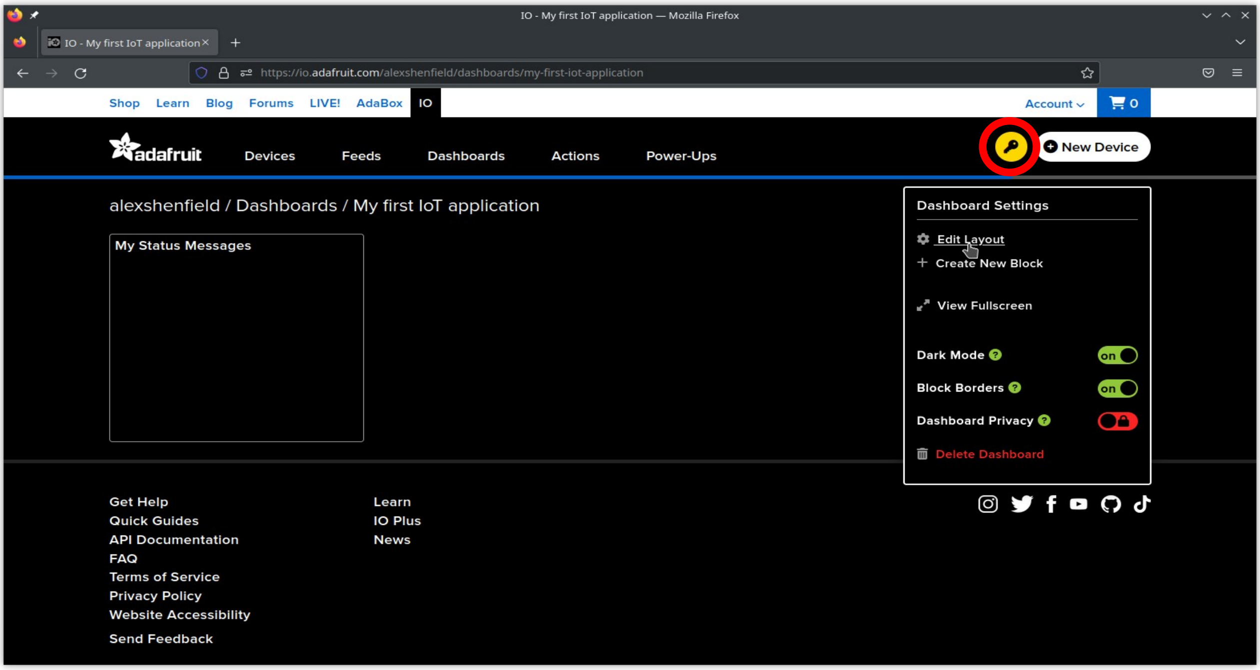

Figure 8 – Our simple dashboard

The last thing we need to do is to get the Adafruit IO key that serves as a password to access your feeds. To do this, click on the key icon (shown highlighted in red in Figure 7, above) and copy the resulting key as shown in Figure 9.

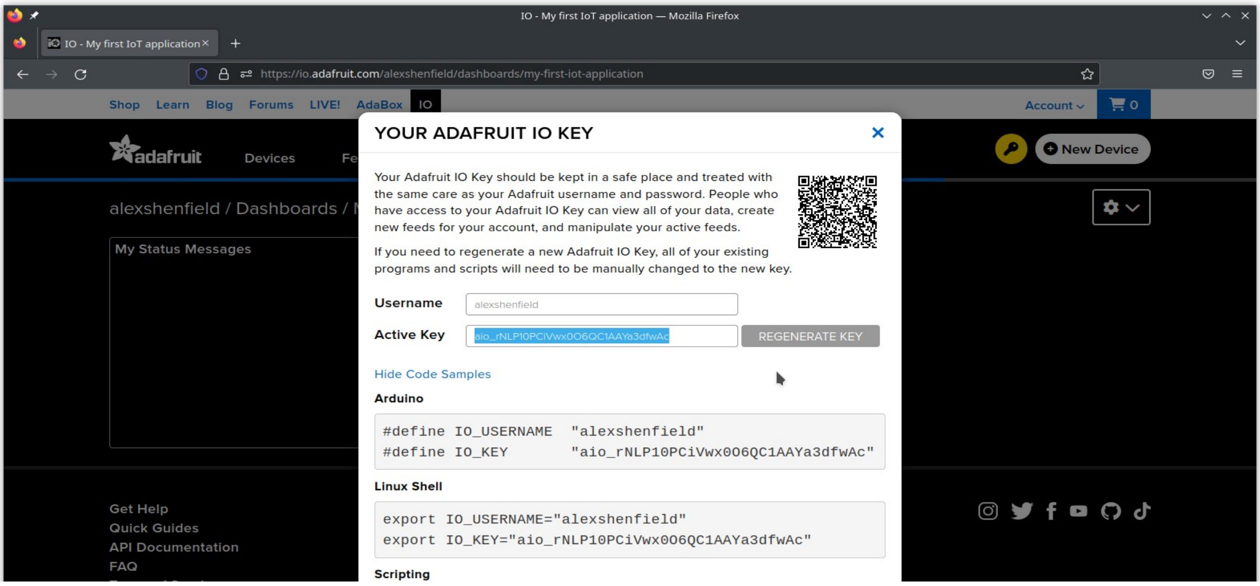

Figure 9 – Retrieving the AIO key

So we have now set up a simple dashboard in Adafruit IO!

However, before we write our Arduino application we will need to install the PubSubClient library for Arduino. We can do this via the library manager in the Arduino IDE (see Figure 10). Just search for "PubSubClient" by Nick O'Leary.

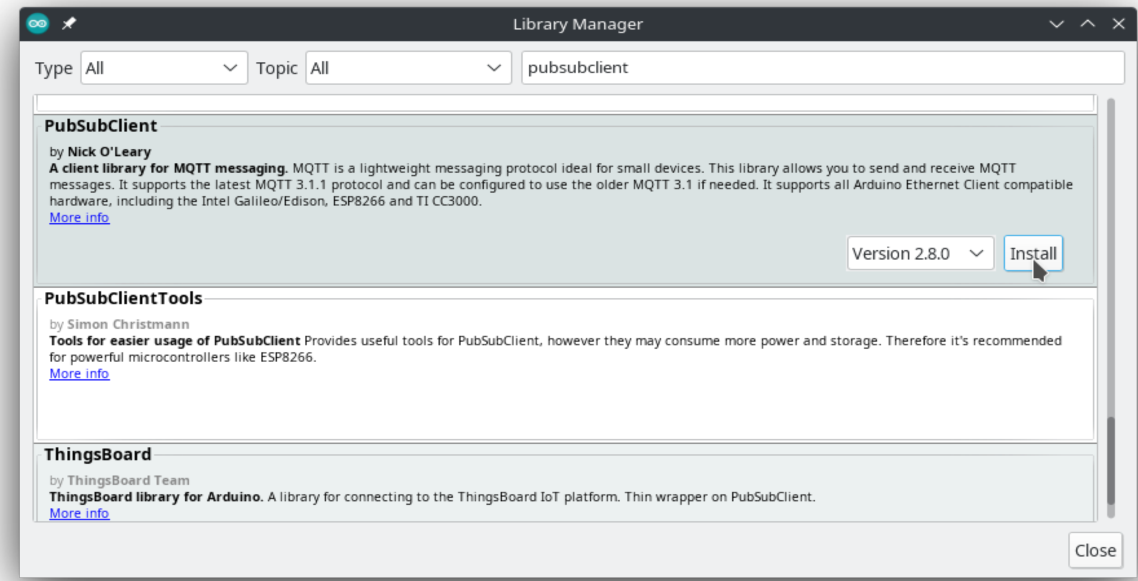

Figure 10 – Installing the PubSubClient library

We can now proceed to write the code for our Arduino application! The point of this application is to make a connection to our MQTT broker, post a status message to the remote MQTT broker, and receive status updates back. If the connection is broken for any reason then the Arduino MQTT client will keep trying to reconnect and then send a status message when it does. We have structured this code in much the same way as the programs in lab 4 – with a “MyCredentials.h” file (shown in code listing 1) and the main application (shown in code listing 2). Note that the “MyCredentials.h” file now also includes your Adafruit IO API credentials and a client id for your IoT device[3].

This code is also available on GitHub.

Note – Adafruit.io limits the amount of messages you can send to **30** per minute. If you try to send more than this you will be throttled initially and repeated offenders will potentially be blocked from using the service! It is therefore very important that the timing code works properly! Pay particular attention to the highlighted code in the main loop.

As an early warning of this – you can see that we are subscribing to the user’s “throttle” topic. This should let us know if we are exceeding the rate limit for messages in adafruit.io.

Code listing 1:

```
// my wifi credentials
const char my_ssid[] = "<your ssid>";
const char my_pass[] = "<your password>";

// my adafruit io credentials
#define CLIENT_ID "<an arbitrary client id>"
#define USER_ID   "<your adafruit io username>"
#define API_KEY   "<your adafruit io api key>"
```

3 This can be anything you want – but try and make it unique to this device (conflicting client ids can sometimes lead to strange connection errors).

Code listing 2:

```
/**
 * 1_simple_mqtt.ino
 *
 * basic mqtt sketch to send a simple status message to an adafruit.io feed and
 * then subscribe to all the feeds. it then periodically sends status update
 * messages (once every 10 seconds) containing the current uptime of the system.
 *
 * author:  alex shenfield
 * date:    03/11/2022
 */

// LIBRARY IMPORTS

// include the necessary libraries for wifi functionality
#include <SPI.h>
#include <WiFiNINA.h>

/// include the pubsub client
#include <PubSubClient.h>

// my wifi credentials are included as a seperate header file
#include "MyCredentials.h"

// WIFI INITIALISATION

// set the initial wifi status (to idle)
int status = WL_IDLE_STATUS;

// MQTT DECLARATIONS

// mqtt server details
const char server[] = "io.adafruit.com";
const long port     = 1883;

// get a wifi client object and pass it to the mqtt client
WiFiClient wifi_client;
PubSubClient client(wifi_client);

// timing variables - note: adafruit.io allows a maximum of 30 messages
// per minute - any more than this and your account will be throttled
// and then blocked!
long previous_time = 0;
long connection_interval = 10000;
```

```
// CODE

// this method runs once (when the sketch starts)
void setup()
{
  // set up serial comms for debugging and display of DHCP allocated
  // IP address
  Serial.begin(115200);
  while (!Serial);
  Serial.println("starting mqtt client on arduino ...");

  // mqtt server and subscription callback
  client.setServer(server, port);
  client.setCallback(callback);

  // attempt to connect to the wifi network
  while (status != WL_CONNECTED)
  {
    Serial.print("attempting to connect to network: ");
    Serial.println(my_ssid);

    // connect to wifi network
    status = WiFi.begin(my_ssid, my_pass);

    // wait for 5 seconds for wifi to come up
    delay(5000);
  }
  Serial.println("connected to WiFi network");

  // print the IP address to the serial monitor
  IPAddress my_ip = WiFi.localIP();
  Serial.print("my ip address: ");
  Serial.println(my_ip);

  // print the received signal strength
  long rssi = WiFi.RSSI();
  Serial.print("my signal strength (rssi):");
  Serial.print(rssi);
  Serial.println(" dBm");
}
```

```
// this methods loops continuously
void loop()
{
  // if the client isn't connected then try to reconnect
  if (!client.connected())
  {
    reconnect();
    previous_time = millis();
  }
  else
  {
    // handle subscriptions to topics (i.e. incoming messages)
    client.loop();

    // periodically publish a message to a feed (note, this uses the
    // same non blocking timing mechanism as in blink_without_delay.ino
    // from lab 1)
    unsigned long current_time = millis();
    if(current_time - previous_time > connection_interval)
    {
      previous_time = current_time;
      send_message();
    }
  }
}
```

```
// MQTT FUNCTIONS

// reconnect to adafruit io mqtt server
void reconnect()
{
  // loop until we're reconnected
  while (!client.connected())
  {
    Serial.println("attempting mqtt connection...");

    // try to connect to adafruit.io
    if (client.connect(CLIENT_ID, USER_ID, API_KEY))
    {
      Serial.println("... connected");

      // once connected, subscribe to all feeds (also subscribe to the
      // "throttle" and "errors" topics which will let us know if we are
      // bumping up against the adafruit.io rate limit)
      client.subscribe((USER_ID "/feeds/#"));
      client.subscribe((USER_ID "/throttle"));
      client.subscribe((USER_ID "/errors"));

      // ... and publish an announcement
      client.publish((USER_ID "/feeds/status-messages"), "we are alive!");
    }
    else
    {
      // print some error status
      Serial.print("connection failed, rc = ");
      Serial.print(client.state());
      Serial.println();
      Serial.println("we will try again in 5 seconds");

      // wait 5 seconds before retrying
      delay(5000);
    }
  }
}

// mqtt message received callback (triggered every time we get a
// message)
void callback(char* topic, byte* payload, unsigned int length)
{
  // print the feed the message comes from
  Serial.print("message arrived [");
  Serial.print(topic);
  Serial.print("] ");

  // print the message
  for (int i = 0; i < length; i++)
  {
    Serial.print((char)payload[i]);
  }
  Serial.println();
}
```

```
// APPLICATION FUNCTIONS

// convert the current uptime to a string and publish to the status-messages
// feed
void send_message()
{
  char time_str[40];
  sprintf(time_str, "up for %lu s", (millis() / 1000));
  client.publish((USER_ID "/feeds/status-messages"), time_str);
}
```

Enter this code and upload it to your Arduino. Don't forget to use your own user name and API key in "MyCredentials.h"!

Once you have uploaded this code, you should see the following output in the serial monitor window (see Figure 11).

Figure 11 – Serial monitor output

And this should update the stream view on Adafruit IO (as in Figure 12).

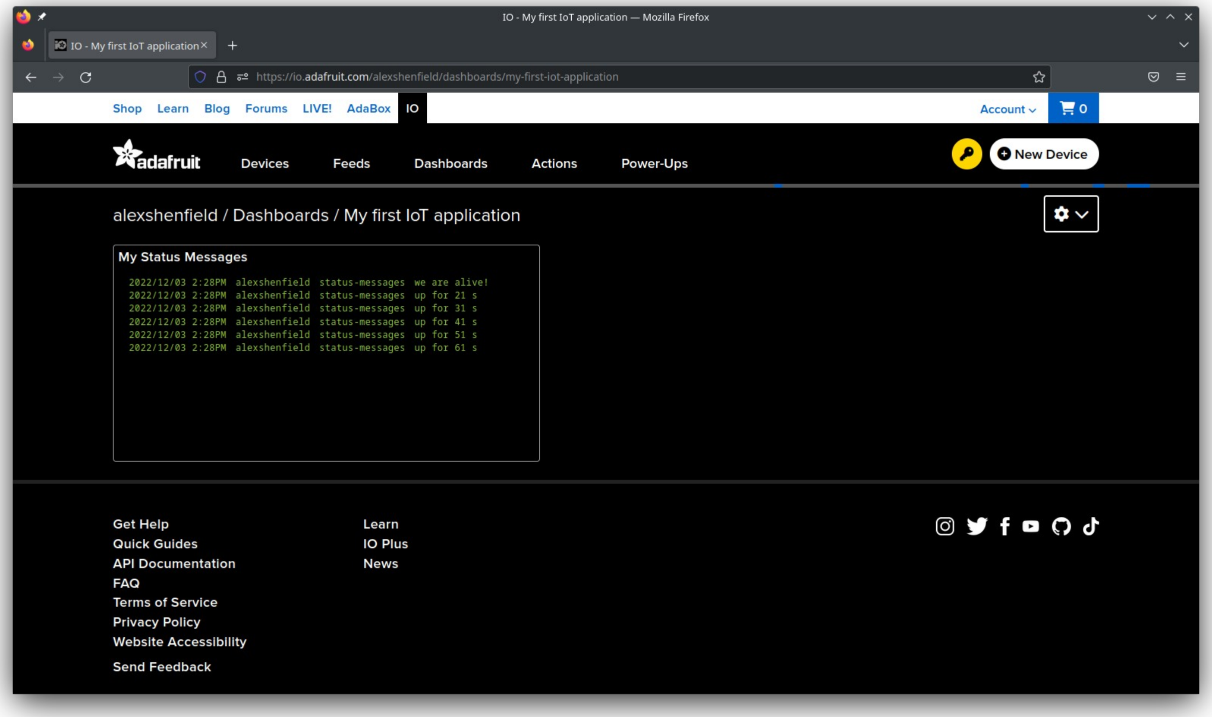

Figure 12 – MQTT messages sent to Adafruit IO

Exercises

Now you are going to make some modifications to the program to add functionality:

1. At the moment you are subscribed to every topic in your feed (which is why you see a lot of repetition when a status message is updated). Alter your program to only subscribe to the status-message topic and the throttle topic.

2. Add some additional controls (e.g. sliders and / or buttons) to your Adafruit IO dashboard and subscribe your Arduino application to those feeds / topics.

3. Adapt the method that periodically publishes status messages to send random integer values to the Adafruit IO service. Publish these to a specific topic – for example:

   <your_aio_username>/feeds/random_numbers

Task 2

We have now introduced the basics of publishing and subscribing to feeds / topics using MQTT on the Arduino. In this task we are going to create a system that monitors ambient light levels within a room and reports them back to the MQTT broker. We will then visualise these light levels using the graph block functionality of the Adafruit IO platform.

Build the circuit shown in Figure 13.

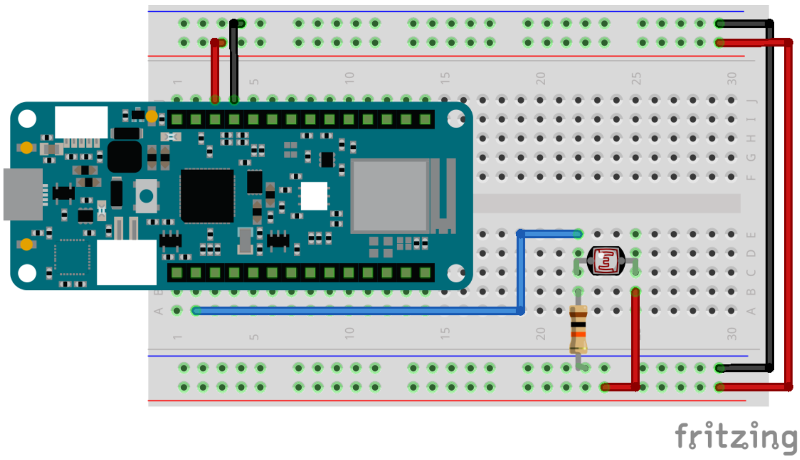

Figure 13 – Light sensor circuit

Code listing 3 provides the full code for this MQTT based light monitoring system. This program reuses a lot of the code from the previous task in setting up the MQTT connection to Adafruit IO and handling message received callbacks.

Don't forget to use your own user name and API key in "MyCredentials.h" and your own username in the feeds!

Code listing 3:

```
/**
* 2_iot_light_levels.ino
*
* mqtt sketch to send ambient light levels to the adafruit io data
* analytics platform for visualisation
*
* author:  alex shenfield
* date:    03/11/2022
*/

// LIBRARY IMPORTS

// include the necessary libraries for wifi functionality
#include <SPI.h>
#include <WiFiNINA.h>

/// include the pubsub client
#include <PubSubClient.h>

// my wifi credentials are included as a seperate header file
#include "MyCredentials.h"

// WIFI INITIALISATION

// set the initial wifi status (to idle)
int status = WL_IDLE_STATUS;

// MQTT DECLARATIONS

// mqtt server details
const char server[] = "io.adafruit.com";
const long port     = 1883;

// get a wifi client object and pass it to the mqtt client
WiFiClient wifi_client;
PubSubClient client(wifi_client);

// timing variables - note: adafruit.io allows a maximum of 30 messages
// per minute - any more than this and your account will be throttled
// and then blocked!
long previous_time = 0;
long connection_interval = 10000;
```

```
// CODE

// this method runs once (when the sketch starts)
void setup()
{
  // set up serial comms for debugging and display of DHCP allocated
  // IP address
  Serial.begin(115200);
  while (!Serial);
  Serial.println("starting mqtt client on arduino ...");

  // mqtt server and subscription callback
  client.setServer(server, port);
  client.setCallback(callback);

  // attempt to connect to the wifi network
  while (status != WL_CONNECTED)
  {
    Serial.print("attempting to connect to network: ");
    Serial.println(my_ssid);

    // connect to wifi network
    status = WiFi.begin(my_ssid, my_pass);

    // wait for 5 seconds for wifi to come up
    delay(5000);
  }
  Serial.println("connected to WiFi network");

  // print the IP address to the serial monitor
  IPAddress my_ip = WiFi.localIP();
  Serial.print("my ip address: ");
  Serial.println(my_ip);

  // print the received signal strength
  long rssi = WiFi.RSSI();
  Serial.print("my signal strength (rssi):");
  Serial.print(rssi);
  Serial.println(" dBm");
}
```

```
// this methods loops continuously
void loop()
{
  // if the client isn't connected then try to reconnect
  if (!client.connected())
  {
    reconnect();
    previous_time = millis();
  }
  else
  {
    // handle subscriptions to topics (i.e. incoming messages)
    client.loop();

    // periodically publish a message to a feed (note, this uses the
    // same non blocking timing mechanism as in blink_without_delay.ino
    // from lab 1)
    unsigned long current_time = millis();
    if(current_time - previous_time > connection_interval)
    {
      previous_time = current_time;
      send_light();
    }
  }
}
```

```
// MQTT FUNCTIONS

// reconnect to adafruit io mqtt server
void reconnect()
{
  // loop until we're reconnected
  while (!client.connected())
  {
    Serial.println("attempting mqtt connection...");

    // try to connect to adafruit.io
    if (client.connect(CLIENT_ID, USER_ID, API_KEY))
    {
      Serial.println("... connected");

      // once connected, subscribe to relevant feeds
      client.subscribe((USER_ID "/feeds/status-messages"));
      client.subscribe((USER_ID "/feeds/light-level"));
      client.subscribe((USER_ID "/throttle"));
      client.subscribe((USER_ID "/errors"));

      // ... and publish an announcement
      client.publish((USER_ID "/feeds/status-messages"), "we are alive!");
    }
    else
    {
      // print some error status
      Serial.print("connection failed, rc = ");
      Serial.print(client.state());
      Serial.println();
      Serial.println("we will try again in 5 seconds");

      // wait 5 seconds before retrying
      delay(5000);
    }
  }
}

// mqtt message received callback (triggered every time we get a
// message)
void callback(char* topic, byte* payload, unsigned int length)
{
  // print the feed the message comes from
  Serial.print("message arrived [");
  Serial.print(topic);
  Serial.print("] ");

  // print the message
  for (int i = 0; i < length; i++)
  {
    Serial.print((char)payload[i]);
  }
  Serial.println();
}
```

```
// APPLICATION FUNCTIONS

// send the light level value
void send_light()
{
  // if the client is already connected then send data
  if (client.connected())
  {
    // get the light level
    int light = analogRead(A0);

    // convert the temperature to a string of characters
    char light_level[5];
    sprintf(light_level, "%i", light);
    client.publish((USER_ID "/feeds/light-level"), light_level);
  }
}
```

Enter this code, but **before you upload it** you should configure your Adafruit IO dashboard to get this data and process it!

To do this we need to add a graph block to our Adafruit IO dashboard and link it to the light level topic. I have also edited the stream block so that the light level messages appear there too (this helps us with debugging and making sure messages are getting through, etc.).

We can then upload the code to our Arduino and start receiving data on our dashboard!

My dashboard now looks like Figure 14.

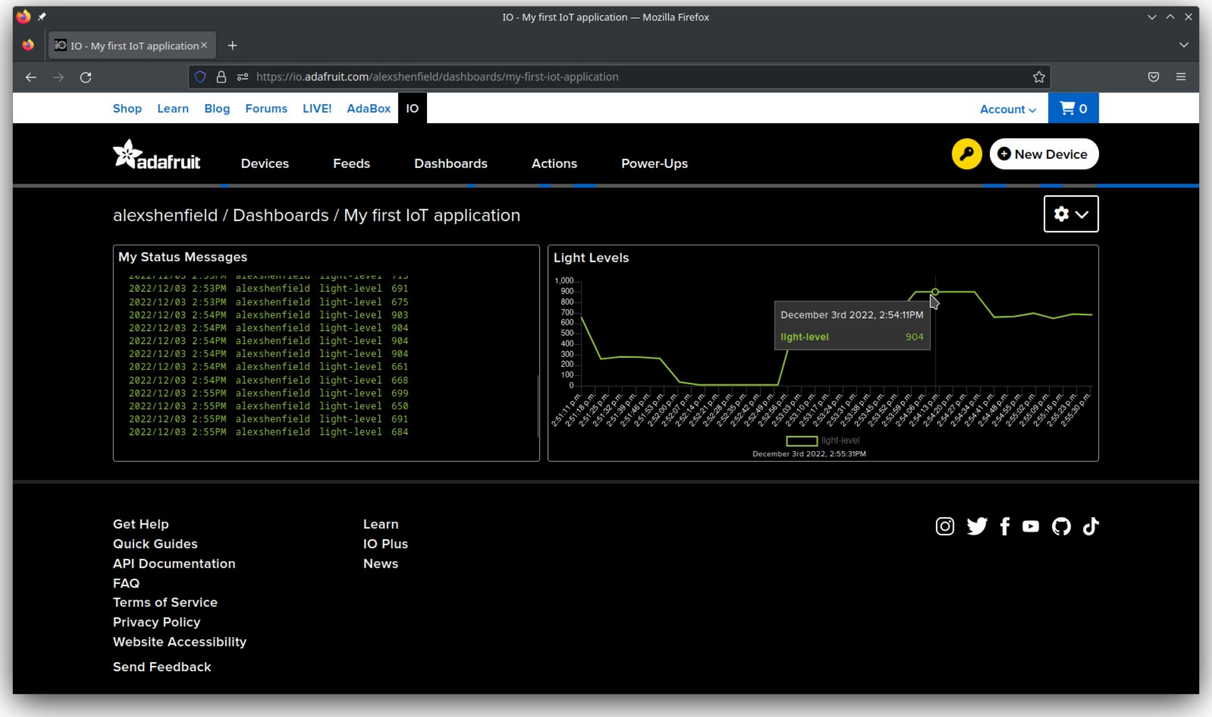

Figure 14 – My Adafruit IO dashboard for light level logging

Exercises

Now you are going to make some modifications to the program to add functionality:

1. Provide a calibration routine for the light sensor during the setup() function that means that the light value is remapped as a percentage before sending to Adafruit IO. You can use the same techniques as in the extension exercises in the previous lab.

2. Add a potentiometer to your system. Read the data from that and send to the Adafruit IO platform for visualisation on the same graph as the light readings.

3. Add an automatic monitoring routine in your Arduino code that turns on an LED when the light level goes below a certain set point.

Task 3

We are now going to build our first complete Internet-of-Things application – an IoT enabled light switch! This application will have 3 parts to it: the hardware, the Arduino application, and the web interface. The hardware is very simple – it is just a button and an LED. This circuit is shown in Figure 15, below.

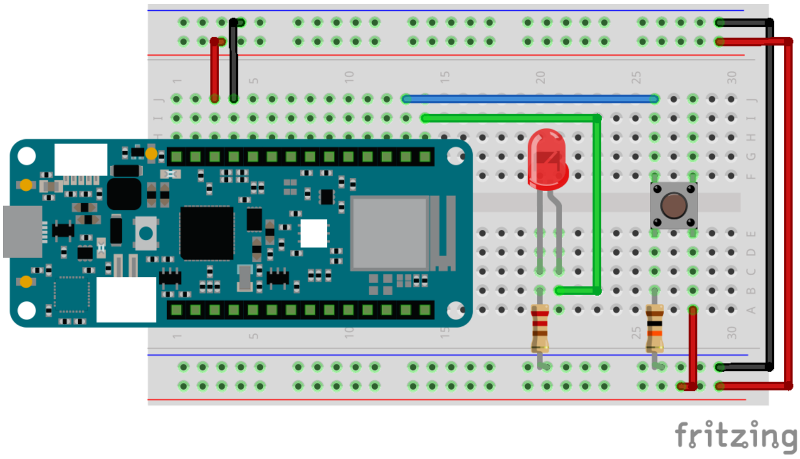

Figure 15 – A light switch

The code for the Arduino application is the most complicated bit of this system – so we will develop it in stages.

The first stage is to develop and test the debouncing code for toggling our led when the button is pressed. Remember, when a button is pressed, the mechanical contacts in the switch may "bounce" around – potentially triggering several button presses in quick succession. In previous labs we have developed our own debouncing code based on timings to avoid this – however, in this lab we will use the EasyButton library by Evert Arias[4]. As with the installation of the other libraries in these lab sessions, we can install this via the library manager in the Arduino IDE.

This library allows us to not only reduce the amount of extra code we need to write to support button debouncing, but also provides a whole host of extra features such as:

1. Simplified support for multiple buttons
2. Support for interrupts
3. Support for double click and long click events

Example debouncing code using this library is shown in code listing 4, below.

4 https://github.com/evert-arias/EasyButton

Code listing 4:

```
/**
* 3a_lazy_button_handling.ino
*
* a simple example of using the EasyButton library to handle button presses
* without a tonne of extra code
*
* author:  alex shenfield
* date:    03/11/2022
*/

// LIBRARY IMPORTS

// include our lazy button library
#include <EasyButton.h>

// BUTTON AND LED DECLARATIONS

// light bulb settings
const int light_bulb = 6;
int light_state = LOW;

// light switch (on pin 7)
EasyButton light_switch(7);

// CODE

// this method runs once (when the sketch starts)
void setup()
{
  // set up serial comms for debugging
  Serial.begin(115200);
  while (!Serial);
  Serial.println("starting lazy button handler ...");

  // set up the light bulb and light switch (including the button pressed
  // callback function)
  pinMode(light_bulb, OUTPUT);
  light_switch.begin();
  light_switch.onPressed(light_toggle);
}

// this methods loops continuously
void loop()
{
  // read the button status
  light_switch.read();
}
```

```
// APPLICATION FUNCTIONS

// callback function attached to the light switch button
void light_toggle()
{
  // print a status message
  Serial.println("button pressed!");

  // toggle the state of the light
  light_state = !light_state;
  digitalWrite(light_bulb, light_state);
}
```

Build the circuit in Figure 16, enter the code above, and test that everything works!

As with the previous applications, much of the code that sets up the MQTT connection and connects and subscribes to topics will remain the same. However, as we will now be reading some data from the remote MQTT broker we will have to work on handling data in our MQTT callback – to do this we will have to extract the data from the message payload and then process it. This means that we need to know what data we are sending from our web interface!

The next step is to design our Adafruit IO dashboard for this application. My dashboard is shown in Figure 16 – it simply has a stream for status messages and a toggle switch for the light switch.

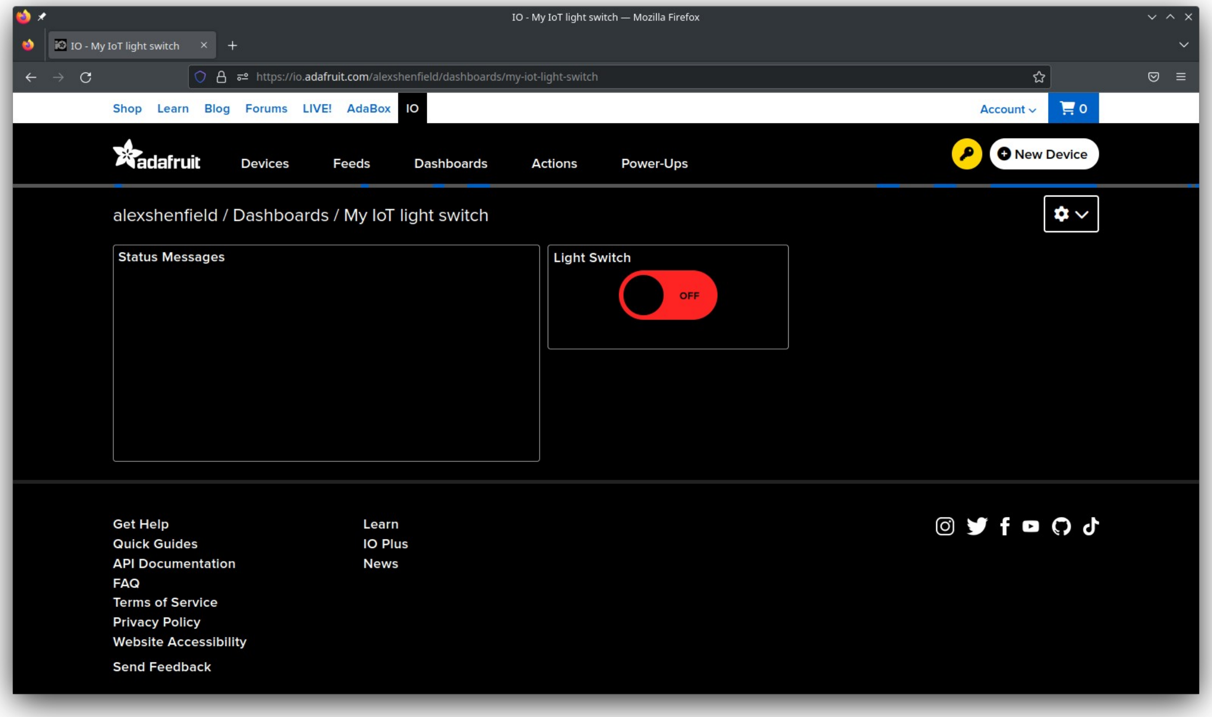

Figure 16 – My IoT light switch dashboard

We can develop a simple MQTT Arduino application to connect to these feeds and display the data on the serial monitor so that we can understand exactly what data is being sent from our web interface. This code is given in code listing 5, below.

Code listing 5:

```
/**
* 3b_iot_subscriptions.ino
*
* a simple mqtt application to parse mqtt messages on subscribed
* topics. in this sketch we have enhanced the message received
* callback to extract the data from the payload into a complete
* string so we can process it (rather than just pinting each
* character
*
* author:  alex shenfield
* date:    03/11/2022
*/

// LIBRARY IMPORTS

// include the necessary libraries for wifi functionality
#include <SPI.h>
#include <WiFiNINA.h>

/// include the pubsub client
#include <PubSubClient.h>

// my wifi credentials are included as a seperate header file
#include "MyCredentials.h"

// WIFI INITIALISATION

// set the initial wifi status (to idle)
int status = WL_IDLE_STATUS;

// MQTT DECLARATIONS

// mqtt server details
const char server[] = "io.adafruit.com";
const long port     = 1883;

// get a wifi client object and pass it to the mqtt client
WiFiClient wifi_client;
PubSubClient client(wifi_client);

// timing variables - note: adafruit.io allows a maximum of 30 messages
// per minute - any more than this and your account will be throttled
// and then blocked!
long previous_time = 0;
long connection_interval = 10000;
```

```
// CODE

// this method runs once (when the sketch starts)
void setup()
{
  // set up serial comms for debugging and display of DHCP allocated
  // IP address
  Serial.begin(115200);
  while (!Serial);
  Serial.println("starting mqtt client on arduino ...");

  // mqtt server and subscription callback
  client.setServer(server, port);
  client.setCallback(callback);

  // attempt to connect to the wifi network
  while (status != WL_CONNECTED)
  {
    Serial.print("attempting to connect to network: ");
    Serial.println(my_ssid);

    // connect to wifi network
    status = WiFi.begin(my_ssid, my_pass);

    // wait for 5 seconds for wifi to come up
    delay(5000);
  }
  Serial.println("connected to WiFi network");

  // print the IP address to the serial monitor
  IPAddress my_ip = WiFi.localIP();
  Serial.print("my ip address: ");
  Serial.println(my_ip);

  // print the received signal strength
  long rssi = WiFi.RSSI();
  Serial.print("my signal strength (rssi):");
  Serial.print(rssi);
  Serial.println(" dBm");
}

// this methods loops continuously
void loop()
{
  // if the client isn't connected then try to reconnect
  if (!client.connected())
  {
    reconnect();
  }
  else
  {
    // handle subscriptions to topics (i.e. incoming messages)
    client.loop();

    // do nothing else here (as we are just testing how message
    // subscriptions work)
    // ...
  }
}
```

```
// MQTT FUNCTIONS

// reconnect to adafruit io mqtt server
void reconnect()
{
  // loop until we're reconnected
  while (!client.connected())
  {
    Serial.println("attempting mqtt connection...");

    // try to connect to adafruit.io
    if (client.connect(CLIENT_ID, USER_ID, API_KEY))
    {
      Serial.println("... connected");

      // once connected, subscribe to relevant feeds feeds (also subscribe
      // to the "throttle" and "errors" topics which will let us know if
      // we are bumping up against the adafruit.io rate limit)
      client.subscribe((USER_ID "/feeds/status-messages"));
      client.subscribe((USER_ID "/feeds/light-switch"));
      client.subscribe((USER_ID "/throttle"));
      client.subscribe((USER_ID "/errors"));

      // ... and publish an announcement
      client.publish((USER_ID "/feeds/status-messages"), "we are alive!");
    }
    else
    {
      // print some error status
      Serial.print("connection failed, rc = ");
      Serial.print(client.state());
      Serial.println();
      Serial.println("we will try again in 5 seconds");

      // wait 5 seconds before retrying
      delay(5000);
    }
  }
}

// enhanced mqtt message received callback (triggered every time
// we get a message)
void callback(char* topic, byte* payload, unsigned int length)
{
  // get the topic
  String t = String(topic);

  // get the value of the message payload
  char data[length + 1];
  for (int i = 0; i < length; i++)
  {
    data[i] = payload[i];
  }
  data[length] = '\0';

  // print the message to the serial window
  Serial.print("message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  Serial.println(data);
}
```

Entering and running this code shows the following output in our serial window when we toggle the switch on our web interface (see Figure 17).

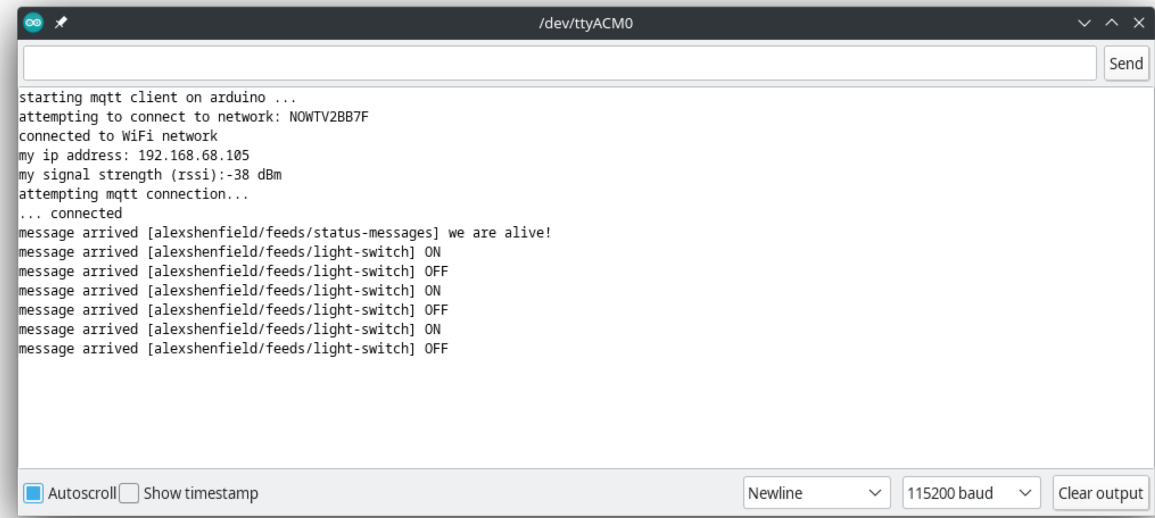

Figure 17 – Serial window output from toggling a switch on my dashboard

So this means that we can interact with the toggle switch by sending and receiving "ON" and "OFF". We can use the **strcmp** function to check for these values on the light-switch topic / feed in our MQTT subscription callback.

The code for the complete application is shown in code listing 6.

Code listing 6:

```
/**
* 3c_iot_light_switch.ino
*
* an internet of things light switch that integrates with adafruit io
* to allow remote control of lighting.
*
* because we are (i am!) lazy, we will use a library for button debouncing.
* this is the "EasyButton" library which can be installed via the library
* manager.
*
* WARNING: if you toggle the light switch too fast, you are likely to
* hit the rate limits of adafruit.io - pay attention to the throttle
* topic in the serial window.
*
* author:  alex shenfield
* date:    03/11/2022
*/

// LIBRARY IMPORTS

// include the necessary libraries for wifi functionality
#include <SPI.h>
#include <WiFiNINA.h>

/// include the pubsub client
#include <PubSubClient.h>

// include our lazy button library
#include <EasyButton.h>

// my wifi credentials are included as a seperate header file
#include "MyCredentials.h"

// WIFI INITIALISATION

// set the initial wifi status (to idle)
int status = WL_IDLE_STATUS;
```

```
// MQTT DECLARATIONS

// mqtt server details
const char server[] = "io.adafruit.com";
const long port     = 1883;

// get a wifi client object and pass it to the mqtt client
WiFiClient wifi_client;
PubSubClient client(wifi_client);

// timing variables - note: adafruit.io allows a maximum of 30 messages
// per minute - any more than this and your account will be throttled
// and then blocked!
long previous_time = 0;
long connection_interval = 10000;

// BUTTON AND LED DECLARATIONS

// light bulb settings
const int light_bulb = 6;
int light_state = LOW;

// light switch (on pin 7)
EasyButton light_switch(7);
```

```
// CODE

// this method runs once (when the sketch starts)
void setup()
{
  // set up serial comms for debugging and display of DHCP allocated
  // IP address
  Serial.begin(115200);
  while (!Serial);
  Serial.println("starting mqtt client on arduino ...");

  // mqtt server and subscription callback
  client.setServer(server, port);
  client.setCallback(callback);

  // attempt to connect to the wifi network
  while (status != WL_CONNECTED)
  {
    Serial.print("attempting to connect to network: ");
    Serial.println(my_ssid);

    // connect to wifi network
    status = WiFi.begin(my_ssid, my_pass);

    // wait for 5 seconds for wifi to come up
    delay(5000);
  }
  Serial.println("connected to WiFi network");

  // print the IP address to the serial monitor
  IPAddress my_ip = WiFi.localIP();
  Serial.print("my ip address: ");
  Serial.println(my_ip);

  // print the received signal strength
  long rssi = WiFi.RSSI();
  Serial.print("my signal strength (rssi):");
  Serial.print(rssi);
  Serial.println(" dBm");

  // set up the light bulb and light switch (including the button pressed
  // callback function)
  pinMode(light_bulb, OUTPUT);
  light_switch.begin();
  light_switch.onPressed(light_toggle);
}
```

```
// this methods loops continuously
void loop()
{
  // if the client isn't connected then try to reconnect
  if (!client.connected())
  {
    reconnect();
  }
  else
  {
    // handle the light switch (but only bother if the state has changed)
    if (digitalRead(light_bulb) != light_state)
    {
      digitalWrite(light_bulb, light_state);
    }

    // handle mqtt functions (e.g. subscriptions to topics)
    client.loop();
  }

  // read the button status
  light_switch.read();
}
```

```
// MQTT FUNCTIONS

// reconnect to adafruit io mqtt server
void reconnect()
{
  // loop until we're reconnected
  while (!client.connected())
  {
    Serial.println("attempting mqtt connection...");

    // try to connect to adafruit.io
    if (client.connect(CLIENT_ID, USER_ID, API_KEY))
    {
      Serial.println("... connected");

      // once connected, subscribe to relevant feeds
      client.subscribe((USER_ID "/feeds/status-messages"));
      client.subscribe((USER_ID "/feeds/light-switch"));
      client.subscribe((USER_ID "/throttle"));
      client.subscribe((USER_ID "/errors"));

      // ... and publish an announcement
      client.publish((USER_ID "/feeds/status-messages"), "we are alive!");
    }
    else
    {
      // print some error status
      Serial.print("connection failed, rc = ");
      Serial.print(client.state());
      Serial.println();
      Serial.println("we will try again in 5 seconds");

      // wait 5 seconds before retrying
      delay(5000);
    }
  }
}
```

```
// mqtt message received callback
void callback(char* topic, byte* payload, unsigned int length)
{
  // get the topic
  String t = String(topic);

  // get the value of the message
  char data[length + 1];
  for (int i = 0; i < length; i++)
  {
    data[i] = payload[i];
  }
  data[length] = '\0';

  // print the message to the serial window
  Serial.print("message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  Serial.println(data);

  // parse the light switch topic to work out what the light bulb
  // should be doing
  if (t.indexOf("light-switch") > 0)
  {
    if (strcmp(data, "ON") == 0)
    {
      light_state = HIGH;
    }
    else if (strcmp(data, "OFF") == 0)
    {
      light_state = LOW;
    }
  }
}

// APPLICATION FUNCTIONS

// callback function attached to the light switch button
void light_toggle()
{
  // toggle the state of the light
  light_state = !light_state;

  // if the client is connected then send a data update
  if (client.connected())
  {
    // send "ON" if the led is on, and "OFF" if the led is off
    if (light_state == HIGH)
    {
      client.publish((USER_ID /f/light-switch"), "ON");
    }
    else
    {
      client.publish((USER_ID /f/light-switch"), "OFF");
    }
  }
}
```

You can see from the above code that we have restructured the MQTT subscription callback so that we extract the data from the MQTT message payload and use that to decide what to do about the state of the light bulb.

We also make sure that when we operate the physical button to turn the light on, that we notify the web interface that the light switch has been turned on or off. This is so the web interface remains consistent with the rest of our system.

Enter this code and test it to make sure everything works.

You should see the led turning on and off with both the physical button and the toggle switch on the web interface. My completed system is shown in Figure 18.

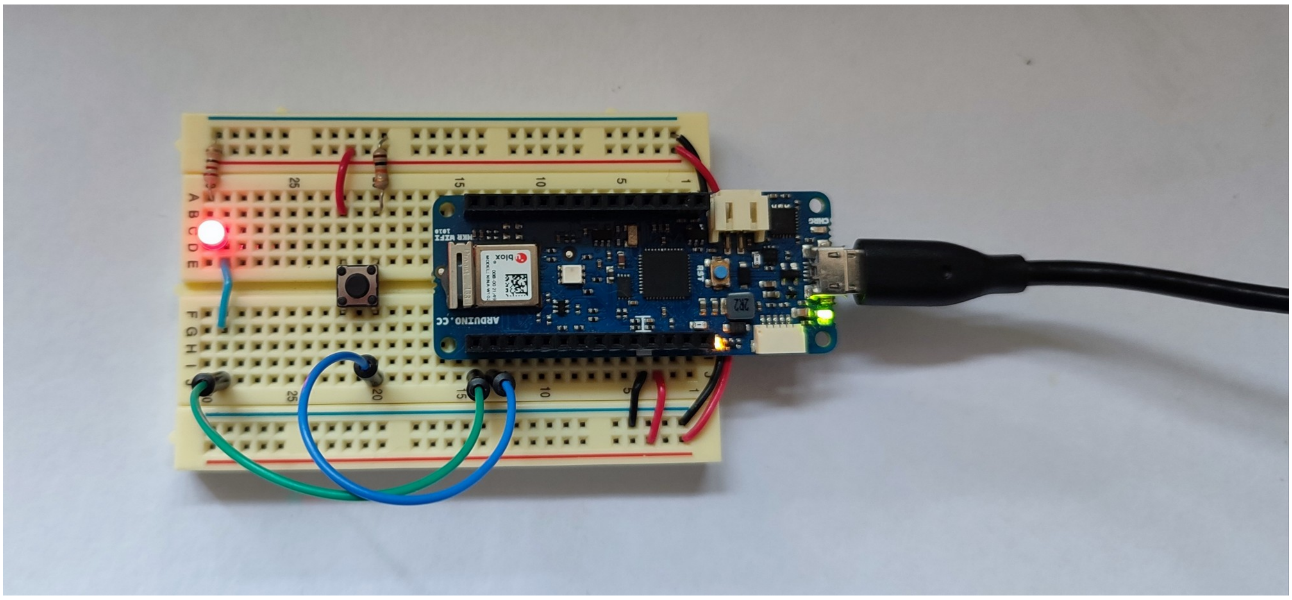

Figure 18 – My IoT light switch!

Exercises

Now you are going to make some modifications to the program to add functionality:

1. Add a calibrated light sensor (as in the exercises in task 2) to send the ambient light levels to the Adafruit IO platform and chart these on a graph block in your dashboard.

2. Add the ability to turn an automatic monitoring routine on or off from the web interface (e.g. using another toggle button). This automatic monitoring routine should turn the LED on when the light level goes below a certain set point and off when it goes above the set point.

3. Add the ability to change the lighting set point from the web interface (e.g. using a slider control).

Check list

Task	Complete
Task 1 – Program	
Task 1 – Exercise 1 (Topics)	
Task 1 – Exercise 2 (Controls)	
Task 1 – Exercise 3 (Additional feeds)	
Task 2 – Program	
Task 2 – Exercise 1 (Calibration)	
Task 2 – Exercise 2 (Potentiometer)	
Task 2 – Exercise 3 (Automatic monitoring routine)	
Task 3 – Program 1	
Task 3 – Program 2	
Task 3 – Program 3	
Task 3 – Exercise 1 (Calibrated light levels)	
Task 3 – Exercise 2 (Control of automatic monitoring)	
Task 3 – Exercise 3 (Lighting set-point adjustment)	

Feedback