

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



Activity ID		Activity Title			Laboratory Room No.	Level
Lab 105		Arduino and the Internet			4302	6
Semester	Duration [hrs]	Group Size	Max Total Students	Date of approval/review	Lead Academic	
1	4	1	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

## **Arduino and the Internet**

### **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](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 demonstrate some of the network functionality we can add to the Arduino platform using the built-in u-blox NINA W102 WiFi module, before showing how we can run RESTful web services on the Arduino.

### **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/>

## **Equipment**

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

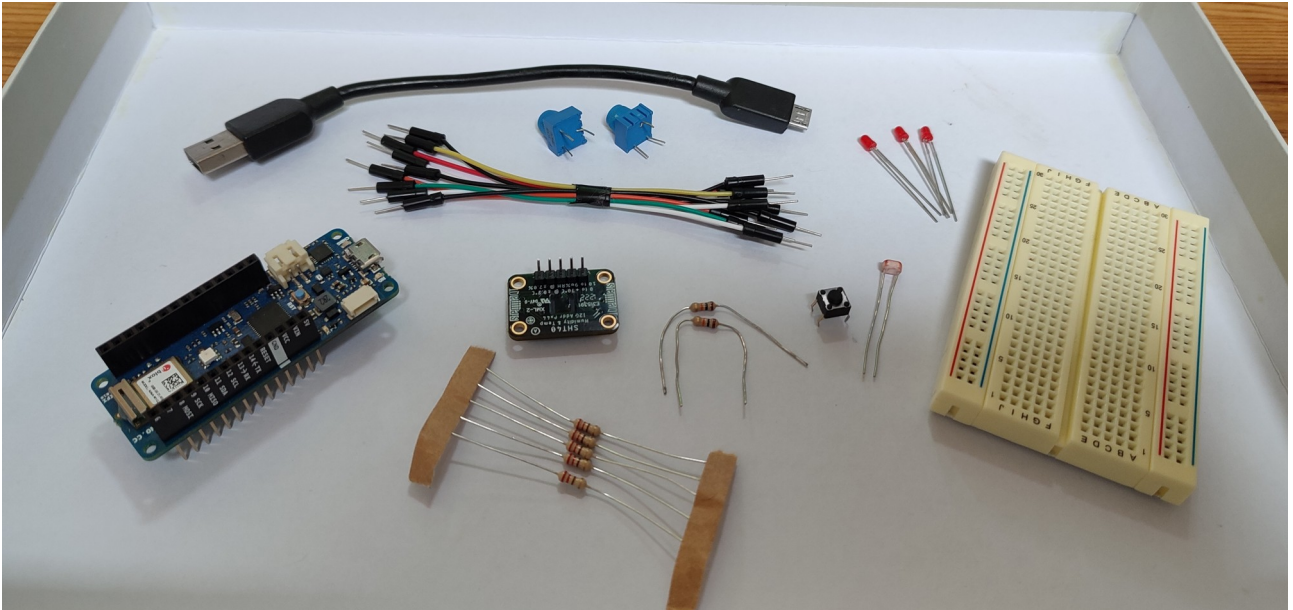


Figure 1 – The necessary equipment for this lab

## Task 1

The Arduino MKR WiFi 1010 includes a ublox NINA-W102 WiFi module – a stand-alone multi-radio MCU module that integrates a powerful microcontroller (an STM32L0 processor) and a radio for wireless communication. This module is capable of turning the Arduino into both a web client (like a browser) or a web server using commands sent to the module over SPI.

To manage this communication we will need to use the WiFinINA library – as used in lab 103 to control the RGB LED. If needed, you can install it from the Arduino Library Manager – press CTRL + Shift + I, and then search for “WiFinINA” (see Figure 2).

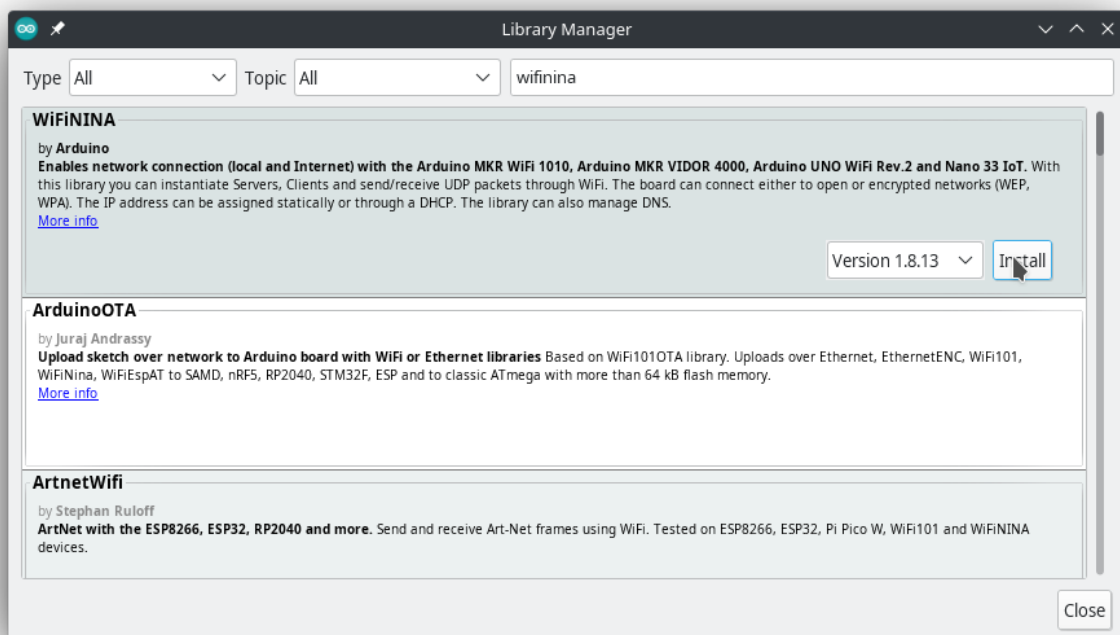


Figure 2 – Installing the WiFinINA library

We are now going to write a simple program to make the Arduino act like a web server. The code is provided in code listings 1 & 2 (below) and goes through the following steps:

1. Sets up a WiFi network connection to the Arduino
2. Listens for data being sent from a client
3. Displays this data on the serial monitor
4. Sends the standard HTTP response header and closes the connection when there is no more data being sent

Firstly we need to provide our wireless access point credentials. I like to do this in a separate header file to keep these apart from the main code. To add a separate header file you can click the down arrow on the right hand side of the Arduino IDE as shown in Figure 3a (or alternatively you can press CTRL + SHIFT + N). This will allow you to choose a name for the new file (as shown in Figure 3b) – I have chosen “MyCredentials.h”.

**Make sure to enter the SSID and password for the WiFi network you want to use!**<sup>1</sup>

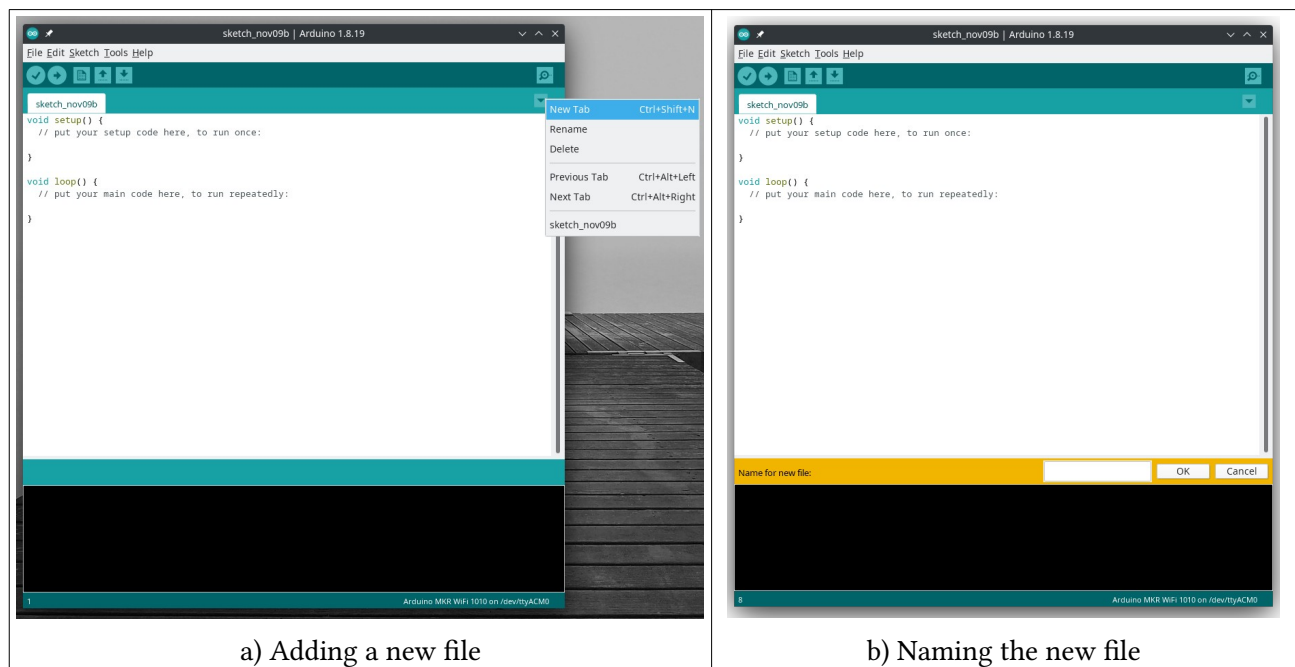


Figure 3 – Adding a new header file for WiFi credentials

Code listing 1: MyCredentials.h

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

1 You will not be able to connect to the SHU-USS WiFi network as it uses WPA-Enterprise security; however, if you are working in Sheaf 4302, there is a separate network you can use. The details are:  
 SSID = ProgramThings  
 Password = 0102030405

Code listing 2: A simple web server

```
/**
 * 1_simple_web_server.ino
 *
 * a simple web server for demonstrating arduino mkr wifi 1010
 * connectivity
 *
 * author: alex shenfield
 * date: 01/11/2022
 */

// LIBRARY IMPORTS

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

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

// WIFI INITIALISATION

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

// set up a server on port 80 (note: web browsers usually assume that
// the server is running on port 80 unless told otherwise)
WiFiServer server(80);
```

```
// 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("simple wifi web server running ...");

  // 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");

  // start the server
  server.begin();
}
```

```
// this methods loops continuously
void loop()
{
    // check for a client connection
    WiFiClient client = server.available();

    // if a remote client is connected
    if (client)
    {
        // get the ip address of the remote client
        IPAddress ip = client.remoteIP();
        Serial.print("new client at ");
        Serial.println(ip);

        // while the client is still connected
        while (client.connected())
        {
            // and has more data to send
            if (client.available() > 0)
            {
                // read bytes from the incoming client and write them to
                // the serial monitor
                Serial.print((char)client.read());
            }
            // when the client is done sending data
            else
            {
                // send standard http response header (to acknowledge the
                // data)
                client.println("HTTP/1.1 200 OK");
                client.println("Content-Type: text/html");
                client.println();

                // disconnect from client
                client.stop();
            }
        }
    }
}
```



We can then upload our code and open the Arduino IDE serial monitor window. This should show us the IP address of our networked Arduino as in Figure 4, below.



Figure 4 – The serial monitor display the Arduino web server's IP address

Now, using your favourite web browser, you can navigate to this address (see Figure 5). You should see the serial monitor display all the connection information (see Figure 6).

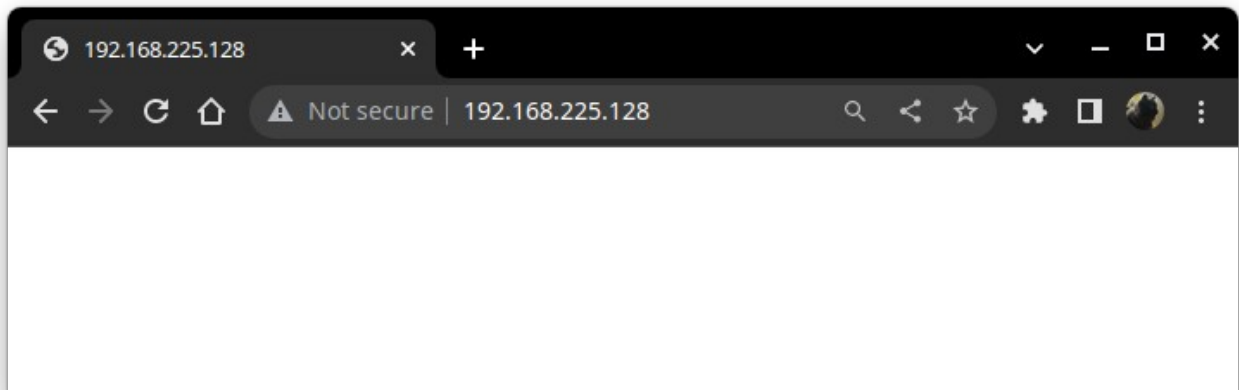


Figure 5 – Navigating to the Arduino web server

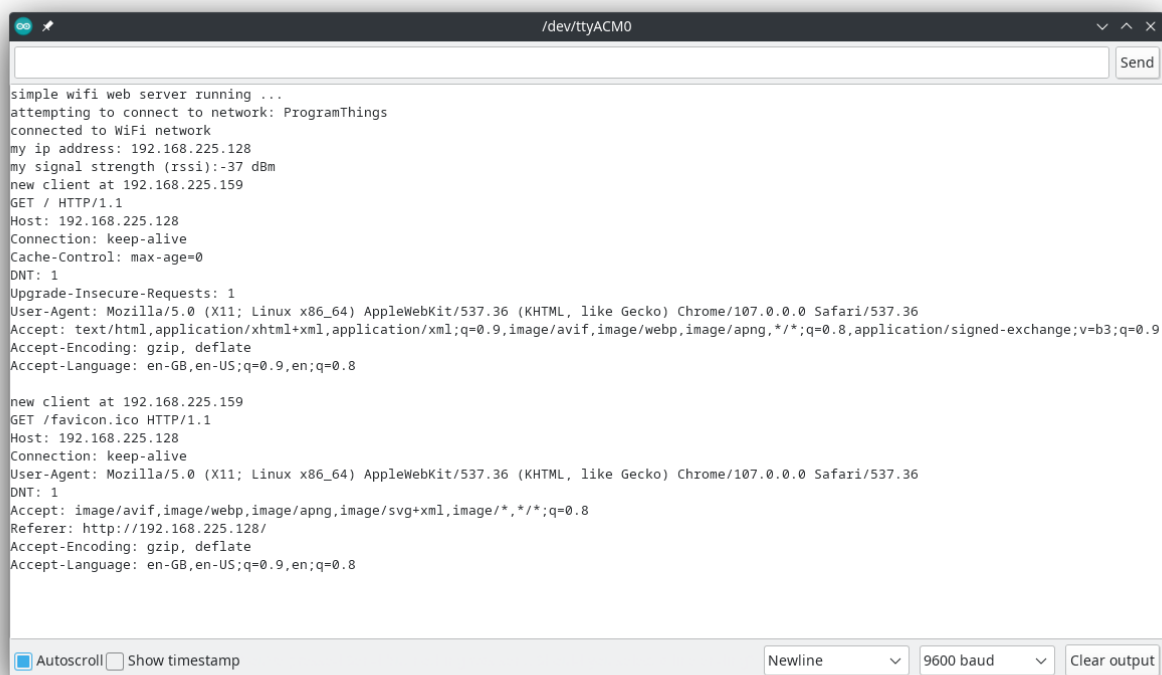


Figure 6 – Serial monitor output

Try entering a different URL and seeing what output you get on the serial monitor – e.g.

*192.168.225.128*/**testing**<sup>2</sup>

Q1 What is the key difference?<sup>3</sup>



<sup>2</sup> Don't forget to use the actual IP address of your board!

<sup>3</sup> Hint – look at the GET requests that are shown in the Serial Monitor.

## Exercise

Now alter your program to actually display some properly formatted HTML in the web page rather than just sending an empty HTTP response header. A good resource for some basic HTML commands is:

<http://www.w3schools.com/html/default.asp>

Some suggestions are:

1. displaying a simple text string
2. displaying a link to another web page
3. experimenting with the various HTML formatting options

Figure 7 shows a (purposely hideous) example web page served up by the Arduino. Hopefully you can do better!

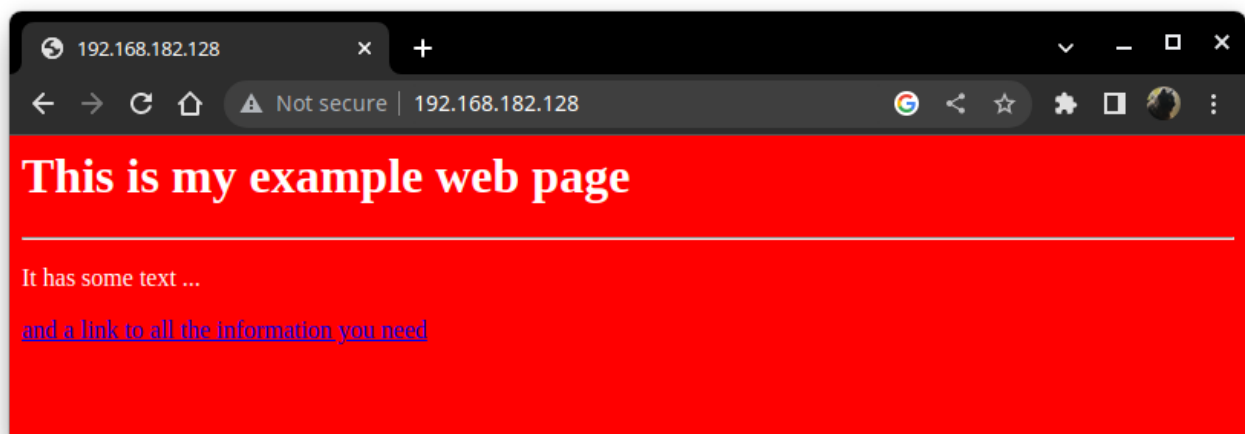


Figure 7 – “My eyes are bleeding!”

## Task 2

Now we have mastered the basics and learnt a bit about the structure of HTTP requests and responses, we can use our Arduino as more than just a web server. In fact, by network enabling our Arduino, we are well on our way to creating elements of the “Internet of Things”.

In this task we are going to use the Arduino MKR WiFi 1010 as a web service to control 4 LEDs attached to pins 2, 3, 4, and 5.

First build the circuit shown in Figure 8.

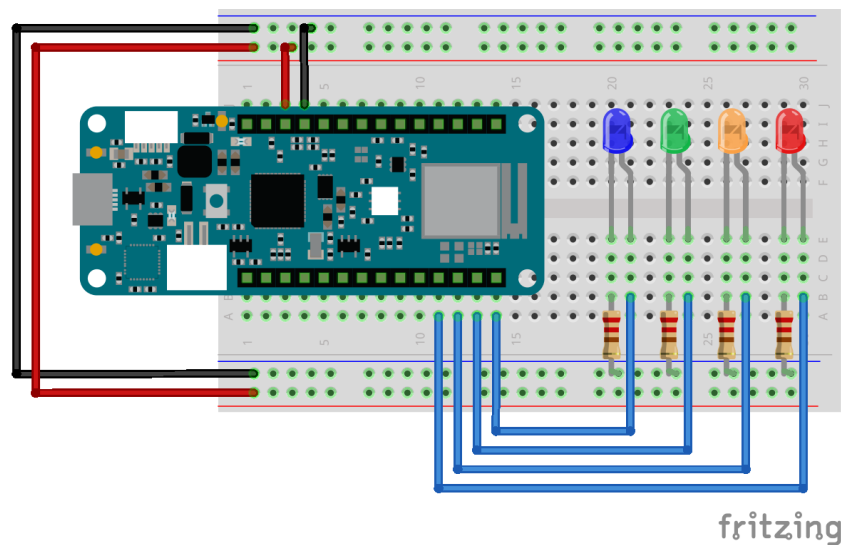


Figure 8 – WiFi controlled LEDs

We now need to write the code to control these LEDs over the network. We can break this down into 3 parts:

1. the initialisation part (where we set up the Arduino and the WiFi network connection) shown in code listing 3. Don't forget to add a header file with your WiFi credentials!
2. the main loop of the program (that simply checks for a connection to the server) and the functions that handle the web client's HTTP requests to the Arduino (shown in code listing 4)
3. the functions that are called to actually control the LEDs (shown in code listing 5)

You should recognise most of the code in code listing 3 from the simple web server we created in Task 1. The HTTP handler code (in code listing 4) basically just parses the information returned from the HTTP client (i.e. the web browser) looking for key markers (in this case a '?' character). Once it sees this marker, it knows the following data is either the Arduino pins to turn on or a command to turn the pins off.

Code listing 3: Setting up the web server

```
/**
 * 2_web_addressable_leds.ino
 *
 * a simple web service to control LEDs using the arduino mkr wifi 1010
 * connectivity
 *
 * author: alex shenfield
 * date: 01/11/2022
 */

// LIBRARY IMPORTS

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

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

// WIFI INITIALISATION

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

// set up a server on port 80 (note: web browsers usually assume that
// the server is running on port 80 unless told otherwise)
WiFiServer server(80);

// variable indicating when to start paying attention to data
boolean pay_attention = true;
```

```
// 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("simple wifi web server running ...");

    // 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");

    // start the server
    server.begin();

    // set up the pins as outputs
    pinMode(2, OUTPUT);
    pinMode(3, OUTPUT);
    pinMode(4, OUTPUT);
    pinMode(5, OUTPUT);
}

// this methods loops continuously
void loop()
{
    // constantly check for connections
    check_for_connections();
}
```

Code listing 4: HTTP request handler functions

```
// HTTP REQUEST HANDLER

// method to check for incoming connections and process them
void check_for_connections()
{
    // get any client that is connected to the server and
    // trying to send data
    WiFiClient client = server.available();

    // record whether we have already sent the standard http
    // response header
    boolean header_sent = false;

    // if a remote client is connected
    if (client)
    {
        // get the ip address of the remote client
        IPAddress ip = client.remoteIP();
        Serial.print("new client at ");
        Serial.println(ip);

        // while the client is connected ...
        while (client.connected())
        {
            // if we haven't already sent the http header, then send it
            if (!header_sent)
            {
                // send standard http response header (to acknowledge the
                // data)
                client.println("HTTP/1.1 200 OK");
                client.println("Content-Type: text/html");
                client.println();
                header_sent = true;
            }
        }
    }
}
```



```

// if the client has more data available ...
if (client.available() > 0)
{
    // read the next byte
    char c = client.read();

    // debugging ...
    Serial.print(c);

    // pay attention to all the data between the '?' character
    // and a space
    if (c == '?')
    {
        pay_attention = true;
    }
    else if (c == ' ')
    {
        pay_attention = false;
    }

    // if we are paying attention ...
    if (pay_attention)
    {
        // use a switch statement to decide what to do
        switch (c)
        {
            case '2':
                trigger_pin(2, client);
                break;
            case '3':
                trigger_pin(3, client);
                break;
            case '4':
                trigger_pin(4, client);
                break;
            case '5':
                trigger_pin(5, client);
                break;
            case 'x':
                clear_pins(client);
                break;
        }
    }
}
// when the client is done sending data
else
{
    // disconnect from client
    client.stop();
}
}
}
}

```

Code listing 5: The LED control functions

```
// UTILITY FUNCTIONS

// separate function for triggering pins - note we only need
// the wifi client so we can send data to it
void trigger_pin(int pin_number, WiFiClient client)
{
    // print a status message
    client.print("Turning on pin <b>");
    client.println(pin_number);
    client.println("</b><br>");

    // turn on appropriate pin
    digitalWrite(pin_number, HIGH);
}

// another function to clear all pins - again the wifi client
// is only needed to return data to the web page
void clear_pins(WiFiClient client)
{
    // print a status message
    client.println("Clearing all pins!<br>");

    // turn off pins
    digitalWrite(2, LOW);
    digitalWrite(3, LOW);
    digitalWrite(4, LOW);
    digitalWrite(5, LOW);
}
```

Enter this code, make sure you understand it, and then upload it to the Arduino. You should then be able to control the LEDs by **navigating to the IP address shown in the serial monitor.**

You can turn on the LEDs on pins 3 and 5 by going to the URL:

*192.168.182.128/?35*

and turn off all the LEDs by going to the URL:

*192.168.182.128/?x*

This is the first step in exposing some simple functionality of our Arduino as a RESTful web service! We can now turn on some of our digital pins (i.e. the LEDs) by sending simple HTTP GET commands to our Arduino web service.

### **Task 3**

So in the last task we exposed some basic functionality of our system as a RESTful web service, providing the ability to control some of the digital outputs of our Arduino via HTTP calls. However, in this case we have had to implement all the HTTP connection parsing code ourselves. For the simple example in the previous task, this is not too difficult – however, if we want to implement a more complex system (for example, monitoring various analog inputs and providing the ability to call more complex functions via HTTP), the code gets substantially more complex!

Fortunately there is no need to reinvent the wheel! One of the key advantages of the Arduino platform is that there is a vast ecosystem of existing libraries and projects to help us. In this case, Marco Schwartz has written the excellent aREST library for Arduino that enables us to write programs that register variables and methods with a REST API – without having to write all the low level HTTP handling code! See:

- <https://github.com/marcoschwartz/aREST>
- <https://arest.io/>

This allows us to focus on the application design rather than the REST API and web service details. As with the installation of the Wi-FiNINA library in task 1, we can install this via the library manager in the Arduino IDE.

In this task we are going to create a system that uses the aREST framework to allow us to control the brightness of an LED and monitor the ambient light levels within a room – all via HTTP calls to a RESTful web service running on our Arduino.

Code listing 6 shows a pseudo-code outline of this system.

Code listing 6: RESTful light controller pseudo code

```
Procedure LIGHTS:

  initialise_system()

  // register variables and functions with
  // the REST api
  register_variable_with_api(light_level)
  register_function_with_api(led_control)

LOOP:
  light_level = read_sensor()

  // if we have a client connection,
  // parse the HTTP commands, extract
  // the parameters, and service the
  // callbacks
  if client.connected()
    handle_connection()
  end if

end LOOP

end Procedure LIGHTS
```

Now build the circuit shown in Figure 9.

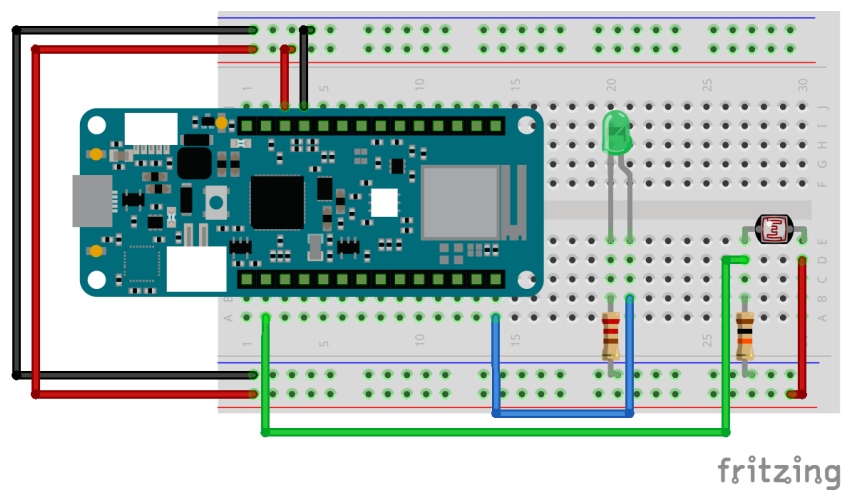


Figure 9 – The light controller circuit

Code listing 7 provides the full code for this RESTful light controller application.

Code listing 7: A RESTful lighting controller

```

/**
 * 3_restful_arduino.ino
 *
 * arduino restful api example using the aREST library - in this example we are
 * setting the brightness of an led on pin 5 and reading light values
 *
 * author: alex shenfield
 * date: 01/11/2022
 */

// LIBRARY IMPORTS

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

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

// include the aREST library (making sure that it knows we are using a WiFi
// client)
#define WiFi_h 1
#include <aREST.h>

// WIFI INITIALISATION

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

// set up a server on port 80 (note: web browsers usually assume that
// the server is running on port 80 unless told otherwise)
WiFiServer server(80);

// AREST DECLARATIONS

// create an aREST instance
aREST rest = aREST();

// variables to monitor with our webservice
int light_level = 0;

```

```
// CODE

// this method runs once (when the sketch starts)
void setup()
{
    // set up serial ...

    // set up serial comms for debugging and display of DHCP allocated
    // IP address
    Serial.begin(115200);
    while (!Serial);
    Serial.println("simple rest client running ...");

    // set up wifi ...

    // 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);

    // start the server
    server.begin();

    // set up rest functionality ...

    // set output pin mode
    pinMode(5, OUTPUT);

    // expose the light level variable to the REST api
    rest.variable("light_level", &light_level);

    // expose the led trigger function to the REST api
    rest.function("led", led_control);

    // set the name and id for this webservice node
    rest.set_id("001");
    rest.set_name("alexs_arduino");
}

// main loop
void loop()
{
    // update our light level each time round the loop
    light_level = analogRead(A0);

    // listen for incoming clients
    WiFiClient client = server.available();
    rest.handle(client);
}
```



```
// FUNCTIONS EXPOSED TO THE REST API

// led control function accessible by the API
int led_control(String command)
{
    // debugging information about the actual command string
    Serial.print("command is ");
    Serial.println(command);

    // get pwm signal from command
    int pwm = command.toInt();
    pwm = constrain(pwm, 0, 255);

    // send pwm signal to the led
    analogWrite(5, pwm);

    // return 1 (indicating success)
    return 1;
}
```

Enter this code, make sure you understand it, and then upload it to the Arduino. Now open the serial monitor - you should see something like that shown in Figure 10.

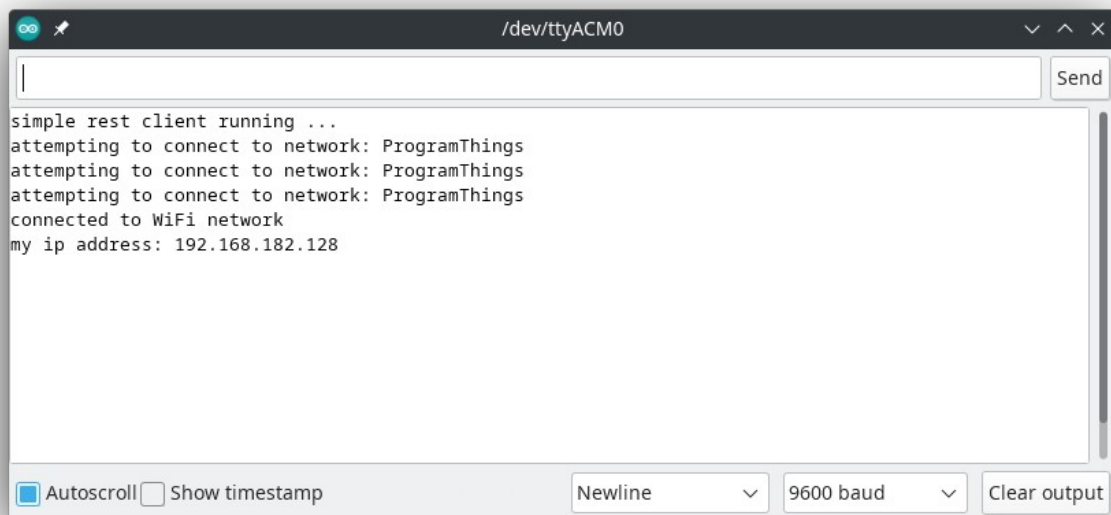


Figure 10 – Serial monitor window

To test the aREST api is working you can open your favourite web browser and enter the URL<sup>4</sup>:

`192.168.182.128/mode/5/o`

This sets digital pin 5 on the Arduino to be an output and should return the JSON output shown in Figure 11.



Figure 11 – JSON output for setting D3 as an output

Now if we go to the URL:

`192.168.182.128/digital/5/1`

We should see the LED connected to pin 5 light up and the following JSON output returned (see Figure 12).

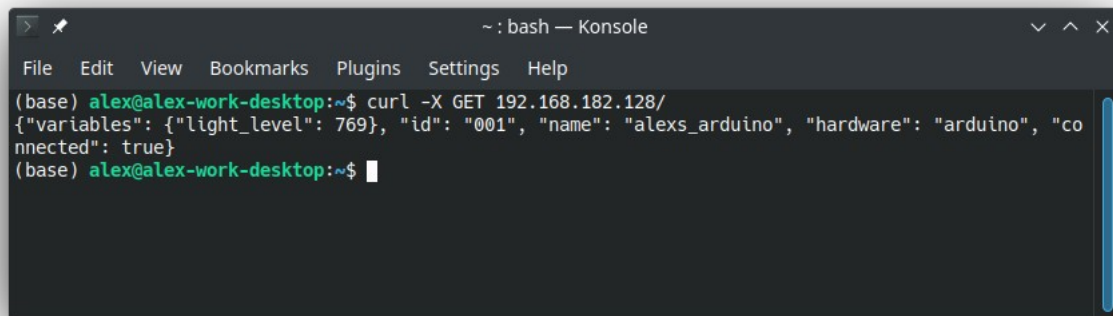


Figure 12 – JSON output for turning pin D3 on

<sup>4</sup> Don't forget to change this to the IP address of your IoT device!

Ultimately, when we enter a URL using our web browser, what we are really doing is sending HTTP GET commands to the web service. Whilst we can carry on doing this using a web browser, I am going to use **curl**<sup>5</sup> to send commands to the web service in the rest of these examples.

If we send an HTTP GET request for the web service root the aREST api provides us with some information about what variables are being monitored (see Figure 13).



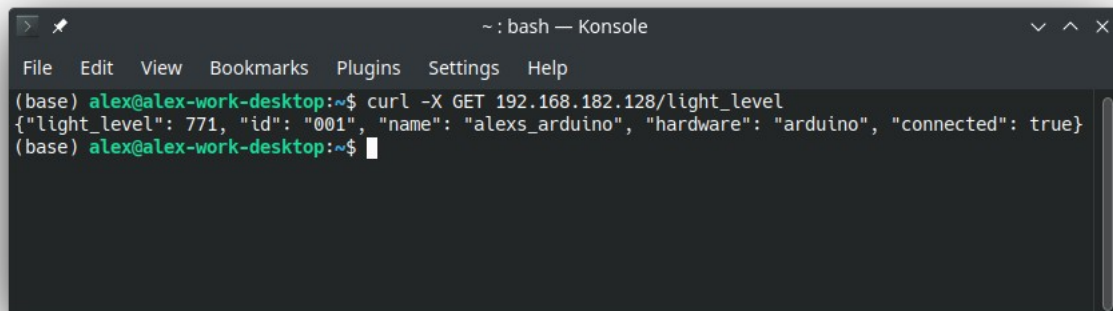
```

~ : bash — Konsole
File Edit View Bookmarks Plugins Settings Help
(base) alex@alex-work-desktop:~$ curl -X GET 192.168.182.128/
{"variables": {"light_level": 769}, "id": "001", "name": "alexs_arduino", "hardware": "arduino", "connected": true}
(base) alex@alex-work-desktop:~$

```

Figure 13 – HTTP GET of the web service root

We can see in Figure 15 that we are monitoring a variable called “light\_level” (if we were monitoring more than one variable, they would all show up here). We can then send an HTTP GET request for just that variable to find out what it's current value is (see Figure 14).



```

~ : bash — Konsole
File Edit View Bookmarks Plugins Settings Help
(base) alex@alex-work-desktop:~$ curl -X GET 192.168.182.128/light_level
{"light_level": 771, "id": "001", "name": "alexs_arduino", "hardware": "arduino", "connected": true}
(base) alex@alex-work-desktop:~$

```

Figure 14 – HTTP GET of the “light\_level” variable

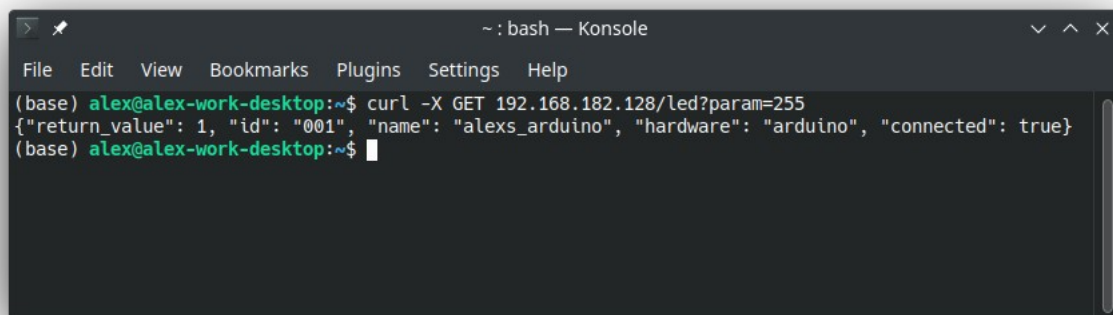
5 See <https://curl.haxx.se/> for more information. Curl is now available in Windows 10 from either powershell or the command window.

Now we are going to remotely call the **led\_control** function via the aREST api. To do this, we need to provide some parameters (i.e. the brightness of the LED as a PWM signal) to the method call.

Send an HTTP GET command with the following format (either via curl or your web browser):

`192.168.182.128/led?param=255`

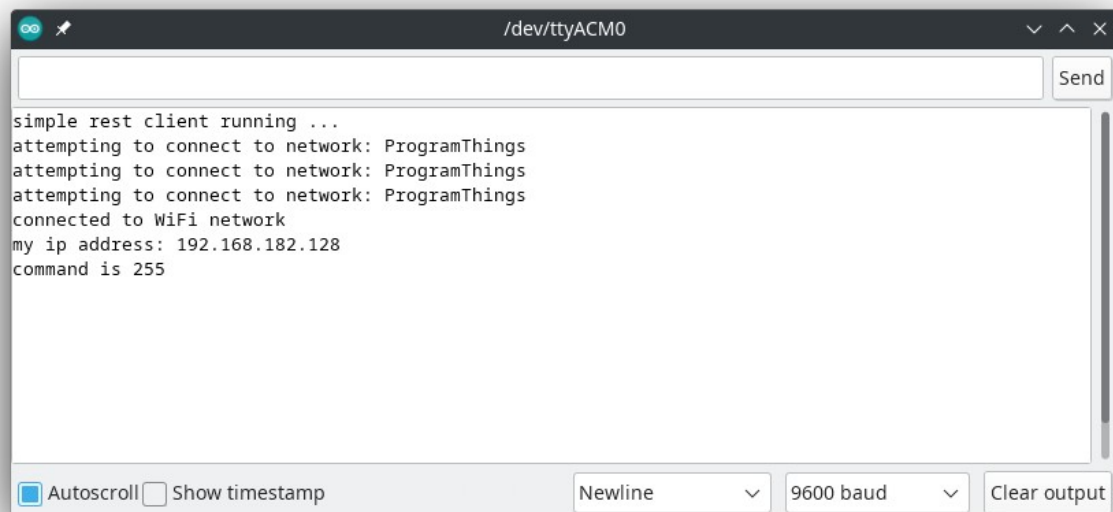
This will turn on the LED and set it to maximum brightness. Figure 15 shows the curl HTTP GET call and response, Figure 16 shows the debugging output in the serial monitor window, and Figure 19 shows the actual system.



```

~ : bash — Konsole
File Edit View Bookmarks Plugins Settings Help
(base) alex@alex-work-desktop:~$ curl -X GET 192.168.182.128/led?param=255
{"return_value": 1, "id": "001", "name": "alexs_arduino", "hardware": "arduino", "connected": true}
(base) alex@alex-work-desktop:~$
  
```

Figure 15 – HTTP GET command setting the brightness of the LED



```

/dev/ttyACM0
simple rest client running ...
attempting to connect to network: ProgramThings
attempting to connect to network: ProgramThings
attempting to connect to network: ProgramThings
connected to WiFi network
my ip address: 192.168.182.128
command is 255
  
```

Figure 16 – Serial monitor window showing the debugging output for the application (i.e. exactly what parameter has been passed to the **led\_control** method)

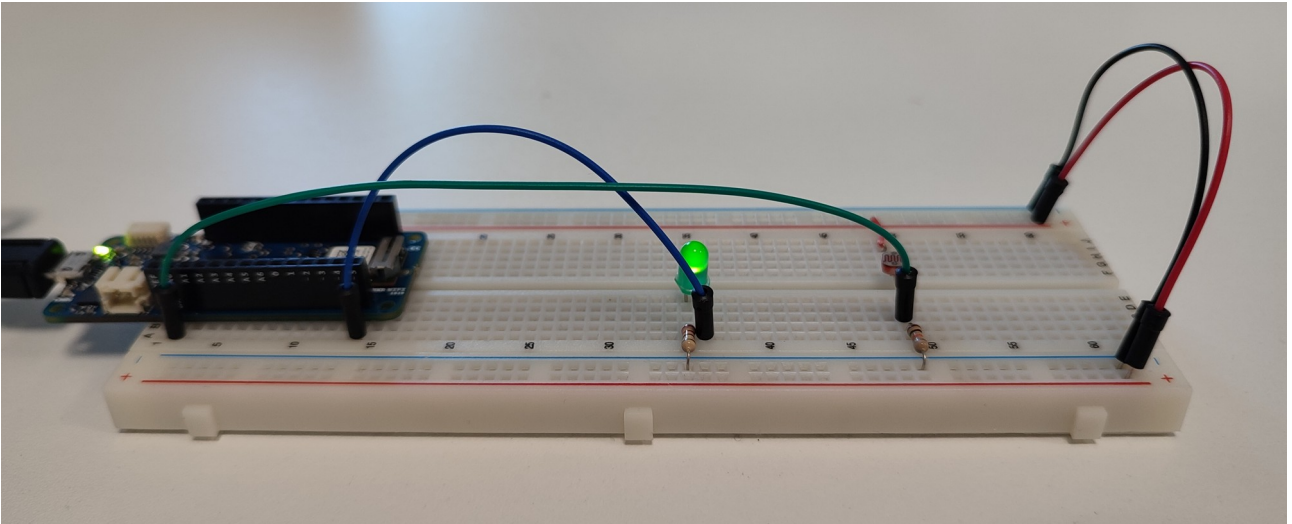


Figure 17 – The embedded web service with the LED turned on at maximum brightness

Now try modifying the HTTP GET request to set the brightness at a different level (e.g. try a value of 50 or 100).

## **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.
2. Use the aREST API to control the in-built RGB LED.
3. Connect up a SHT40 sensor and adapt the code above (in code listing 7) to monitor temperature and humidity too.
4. Add more LEDs to your system and write an additional method that cycles through them every time it is called. Make sure this is hooked in to the web service api.

### **Check list**

Task 1 – Program	
Task 1 – Web development exercises	
Task 2 – Program	
Task 3 – Program	
Task 3 – Exercise 1 (Calibration)	
Task 3 – Exercise 2 (RGB LED)	
Task 3 – Exercise 3 (Other sensors)	
Task 3 – Exercise 4 (More LEDs)	

### **Feedback**

--