Technical article
Arduino SIM900 SIM800 IoT End-to-End Solution

Contents

1 Introduction

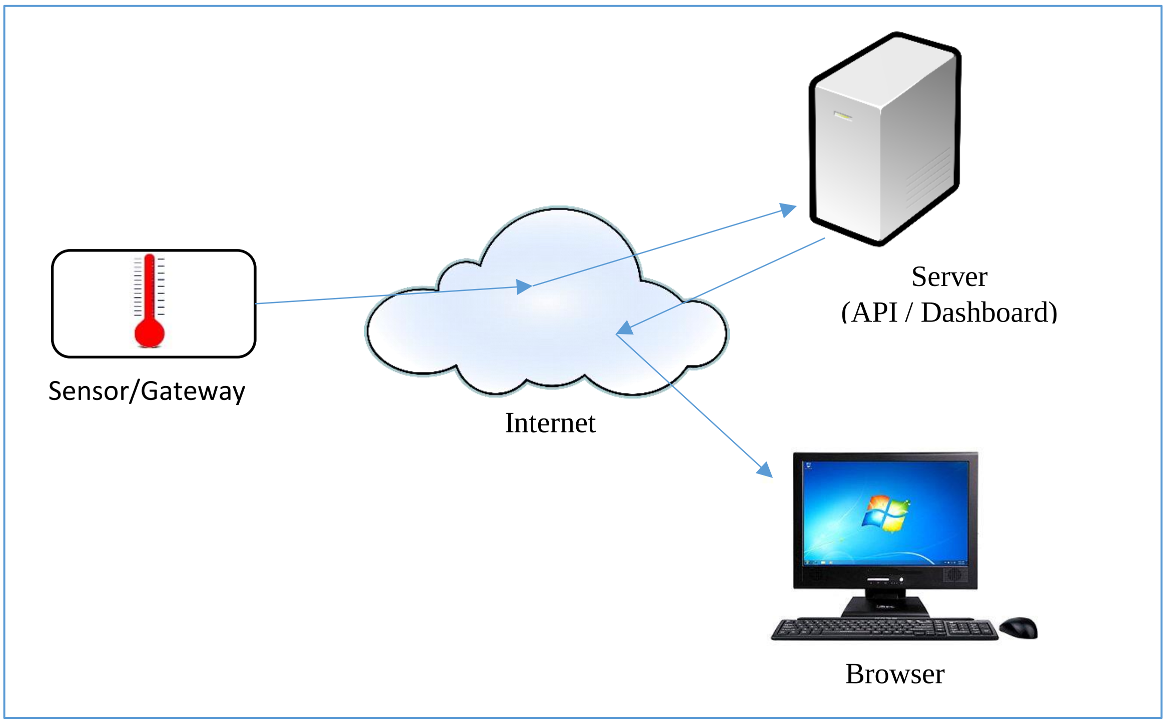
This article will show you how to roll-your-own simple end-to-end internet-of-things (IoT) solution without the need for IoT platforms such as thingspeak or cloudMQTT at virtually no cost. If that appeals to you then read on.

For simplicity we'll assume you have a hosting account free or otherwise supporting php and MySql and has some form of control panel to aid setting up the database and uploading and modifying files.

The three main aspects of the end-to-end solution are:

- Sensor and gateway for sending data to the remote server (Device)
- Server side application end point (API)
- Server side user interface (Dashboard)

The architecture of the solution is illustrated in the figure below.



What you will need:

1. Arduino UNO.
2. Arduino GSM SIM900 shield.
3. SIM card with known access point name (APN) and password.
4. Hosting account supporting MySql and PHP. There are completely free hosting accounts that can be used for doing this. See the appendix for more information.

2 Pre-requisites

You will need to have, or obtain, skills in:

- Arduino hardware and Arduino shields
- Arduino programming
- Arduino sketch (basic knowledge)
- Some PHP understanding
- Uploading files to a hosting account
- Setting up MySql database

3 Server

We'll start at the server end and work back to the sensor/gateway (device). First assuming you have a hosting account with some form of control panel. This could also be done with something like XAMPP on your desktop PC but this can get a little more involved with setting up when dealing with routers and firewalls. So we will just concentrate on an externally hosting account. The one used for preparing this tutorial was 'www.freewebhostingarea.com', there are others such as 'www.000webhost.com', no doubt there are many more with varying degrees in quality of service and features.

3.1 Configure MySql

3.1.1 Create data base

What we need to do here is create a data base and data base table. To do this log on to your control panel and create a MySql data base. The way this is done varies from control panel to control panel so be sure to following the instructions given by your host on how to do this. Usually this is a matter of filling in and submitting an online form.

There are four important parameters you need to make a note of when creating the data base, these are:

1. Server name (this is usually 'localhost').
2. The database user name.
3. The database password.
4. The name of the database.

If the data base has already been created you will need to obtain the above parameters.

3.1.2 Create Data base table

Next is to create the data base table.

There are two ways of doing this:

1. Manually create the table through the database management tool such as PhpMyAdmin and set up the required columns. These should be as per the script below.
2. The second, and by far the simplest, is to create the table by executing the script. This is done by copying the script below and pasting it into the SQL execution area on the data base management tool and executing. I've shown an example of this for PhpMyAdmin in the appendix at the end of this article.

```
CREATE TABLE device_type_1 (
    log_time INT(8) UNSIGNED,
    device_id VARCHAR(30),
    field1 VARCHAR(30)
)
```

3.2 Application Programmer Interface (API)

The application programmer interface or API, is the end point for our device to send data to. This has been kept as simple as possible for illustration purposes and thus is not suitable for production due a number of security issues. More on this later.

3.2.1 API Code

The API code is shown below in ‘api.php’. When executed either using a standard HTML form or sent from a device, will open the database, read the posted data and insert it in to our database table created above. You will need to modify the code below (highlighted in red) with parameters for your database, these are localhost, username, password and data_base_name to the ones you are using for your database. Once done, create a file on you server’s public folder and name it ‘api.php’. Open the file and past in the code below and make the required changes.

api.php

```
<?php

    // Open data base.
    //-----------------
    $servername = "localhost";
    $username = "username";
    $password = "password";
    $dbname = "data_base_name";
    $conn = new mysqli($servername, $username, $password, $dbname);

    // Collect posted data and get the current time.
    //-----------------------------------------------
    $device_id =  $_POST['device_id'];
    $field1 =  $_POST['field1'];
    $log_time = time();

    // Create and execute SQL query string
    // to put data into table.
    //---------------------------------------
    $sql = "INSERT INTO device_type_1 VALUES ($log_time, \"$device_id\", $field1 )";
    $conn->query($sql);

    // Respond to client with the text 'OK'.
    echo "OK<br>";

?>
```

3.2.2 API Test

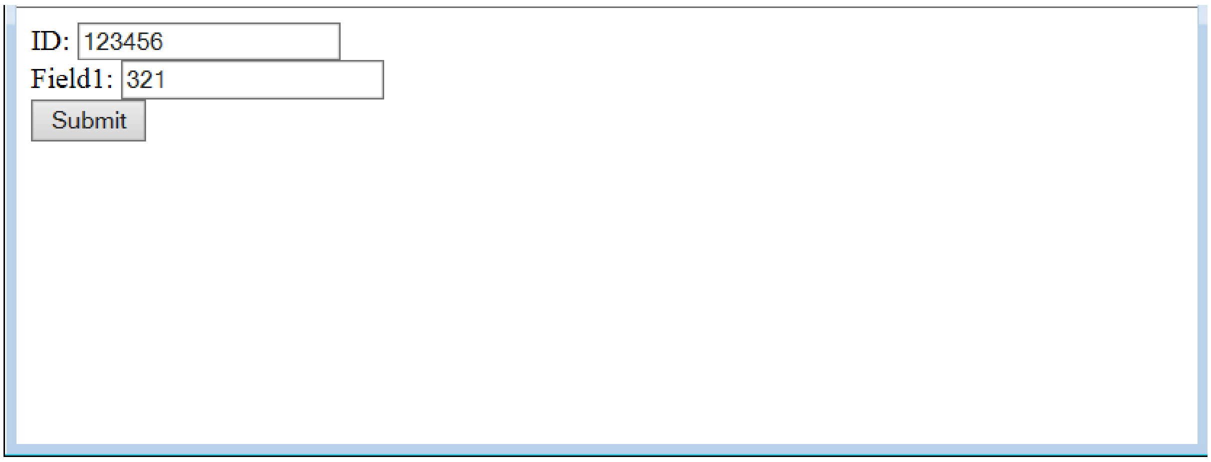
Great, now we’re ready to test the API. But before we start playing around with the device we’re going to do a simple test using an HTML form. The code for this is below. Create a file on your server’s public folder and name it ‘apitest.html’. Edit the file and copy and paste the code below. Change ‘your_host_url’ to your URL. Do not include www in the URL or the post won’t work properly.

apitest.html

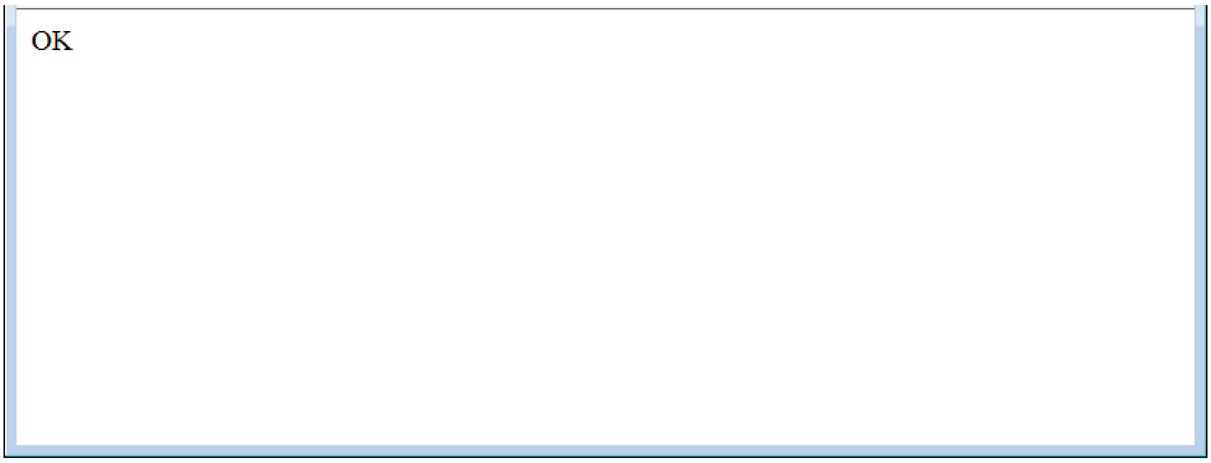
```
<!DOCTYPE html>
```

```
<form action="http://your_host_url/api.php" method="post" target="_blank">
  ID: <input type="text" name="device_id" value ="123456"><br>
  Field1: <input type="text" name="field1" value ="321"><br>
  <input type="submit" value="Submit">
</form>

</html>
```

On your browser, enter 'www.your_host_url/apitest.html' and the following form should be displayed:


ID: 123456
Field1: 321

Click submit and the following should be displayed:


OK

If you get any other response, check carefully that 'api.php' and 'apitest.html' have the correct data base settings and are in the root directory.

Next, open up your data base admin tool (e.g. PhpMyAdmin) and navigate to the device_type_1 table and browse the data. A row of data should have been inserted with having values of data for 'device_id' and 'field1' of 123456 and 321 respectfully. If you have got this far without any hitches you're doing well.

3.3 The dashboard

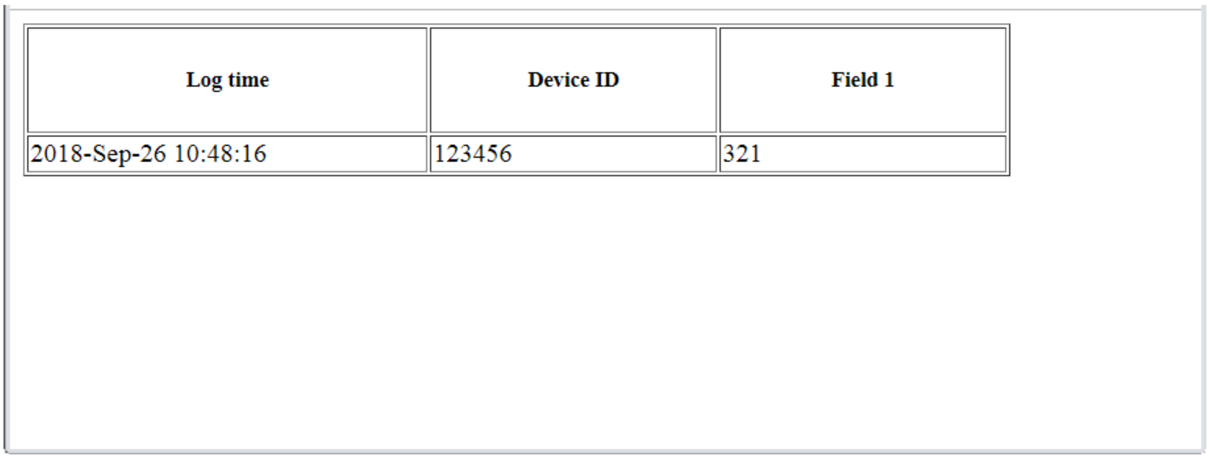
OK, now we've got data in the table it's time to show it on our ultra-simple dashboard. The script below creates an html table and populates it with up to 25 rows of data from the database table. Also to save having to manually refresh the page to see if new data has arrived, this is done automatically once every 10 seconds which will be useful later when real data is coming in.

Similarly, create a file on your server's public folder and name it 'dashboard.php'. Edit it and cut and past the code below into the file making changes to the code (marked in red) as before. Remember to save before exiting the editor.

dashboard.php

```
<!DOCTYPE html>
<html>
  <head>
    <title>Simple Dash board</title>
    <meta http-equiv="refresh" content="10" >
  </head>
  <body>

    <!-- Dispaly a simple table containing most recent 25 rows of data in time
    decending order. -->

    <!-- Table headers -->
    <table width='600' border = "1" >
    <tr>
    <th width="35%"><h5>Log time</h5></th>
    <th width="25%"><h5>Device ID</h5></th>
    <th width="25%"><h5>Field 1</h5></th>
    </tr>


    <!-- Table rows -->
    <?php
    // Open data base.
    //-----------------
    $servername = "localhost";
    $username = "username";
    $password = "password";
    $dbname = "data_base_name";
    $conn = new mysqli($servername, $username, $password, $dbname);


    // Get data from table for device_type_1 .
    //----------------------------------------
    $sql = "SELECT log_time, device_id, field1, field2 FROM device_type_1
        ORDER BY log_time DESC LIMIT 25";
    $result = $conn->query($sql);


    // Output each row of table data.
    //-------------------------------
    while($row = $result->fetch_assoc())
    {
        // Convert log time to human readable time.
        $ut = $row['log_time'];
        $time = date('Y-M-d H:i:s', $ut);

        // Output html table code with table values.
        echo "<tr>";
        echo "<td>" . $time . " </td>";
        echo "<td>" . $row['device_id'] . " </td>";
        echo "<td>" . $row['field1'] . " </td>";
        echo "</tr>";
    }
    ?>

    <!-- End of table -->
    </table>
  </body>
</html>
```

Now let's test this by entering 'www.your_host_url/dashboard.php' and if all is well the page below will be shown with a table containing data from the database table, in this case just one row. Try entering different values in 'apitest.html', resubmit and watch them appear on the dashboard.



Log time	Device ID	Field 1
2018-Sep-26 10:48:16	123456	321

3.4 RAW HTTP test

This step is not strictly necessary but does help gain insight into forming an HTTP post message which we'll be doing with the device firmware later. It also serves as a debug tool when there are problems with the device sending data to the server.

What we're going to do here is create a raw HTTP message and send it to the server via a utility such as Tera Term. This closely mimics what our device is going to do.

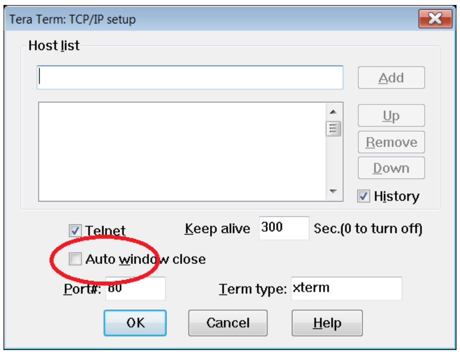
So, let's get started by sending a raw HTTP post message like the one prepared below:

Raw HTTP POST request

```
POST /api.php HTTP/1.1
Host: your_host_url
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 27

device_id=123456&field1=321
```

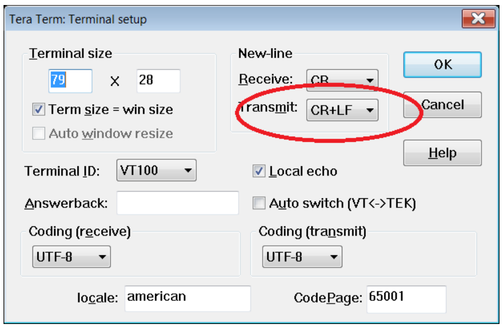
I'll show you how to do this with Tera Term but other terminal emulators could be used.

1. Start Tera Term and turn off auto window close from 'Setup > TCP/IP…'. Make sure 'Auto window close' is unchecked. This ensures Tera Term will remain open after the server closes saving us having to restart it every time a message is sent.



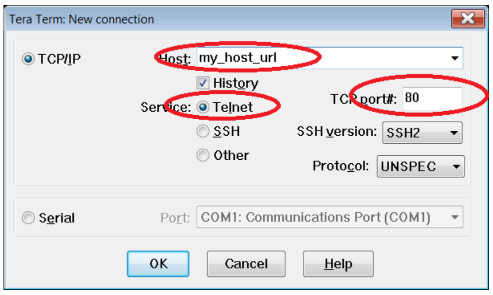
☑ History
☑ Telnet
Keep alive 300 Sec.(0 to turn off)
☐ Auto window close
Port#: 80
Term type: xterm

2. Set the Tera Term to transmit CR+LF from ‘Setup > Terminal…’. The server needs the line feed as well as the carriage return.



Terminal size: 79 x 28
☑ Term size = win size
☐ Auto window resize
New-line
Receive: CR
Transmit: CR+LF
Terminal ID: VT100
☑ Local echo
Answerback:
☐ Auto switch (VT<->TEK)
Coding (receive): UTF-8
Coding (transmit): UTF-8
locale: american
CodePage: 65001

3. Connect to your server from ‘File > New connection…’. Make sure the URL is correct, Telnet selected and port is set to 80. Click OK.



◉ TCP/IP
Host: my_host_url
☑ History
Service: ◉ Telnet
○ SSH
○ Other
TCP port#: 80
SSH version: SSH2
Protocol: UNSPEC
○ Serial
Port: COM1: Communications Port (COM1)

4. A connection should now be made with the server which is waiting for some input. If we do nothing or are not fast enough responding, the server will time out (after a minute or so) and a new connection will need making. What we have to do now is enter raw HTTP. Rather than type this in by hand which will lead to errors, we’re simply going to copy the raw HTTP to the clip board and past it into Tera Term by right clicking. Make sure the correct URL has been set in the raw HTTP.
5. When this is done the server will respond with something similar to the server response below.

```
HTTP/1.1 200 OK
Date: Wed, 26 Sep 2018 08:43:23 GMT
Server: Apache/2.4.34
X-Powered-By: PHP/7.1.22
Upgrade: h2,h2c
Connection: Upgrade, close
Transfer-Encoding: chunked
Content-Type: text/html; charset=UTF-8

8
OK<br>
0
```

If this has all worked we should now be able to observe some new updates on our dashboard. Succeeding at this will give us a high level of confidence that sending the same raw HTTP from our device will work without a hitch.

4 Device (Sensor / Gateway)

The sensor, in this case an Arduino UNO measuring a potentiometer (pot) is attached to the gateway in the form of an Arduino GSM shield. The pot is measured and its value sent to the server via the gateway where it is recorded in a database and viewed on the dashboard using a browser. The firmware can drive and has been tested on SIM900 and SIM800 GSM shields albeit with limitations.

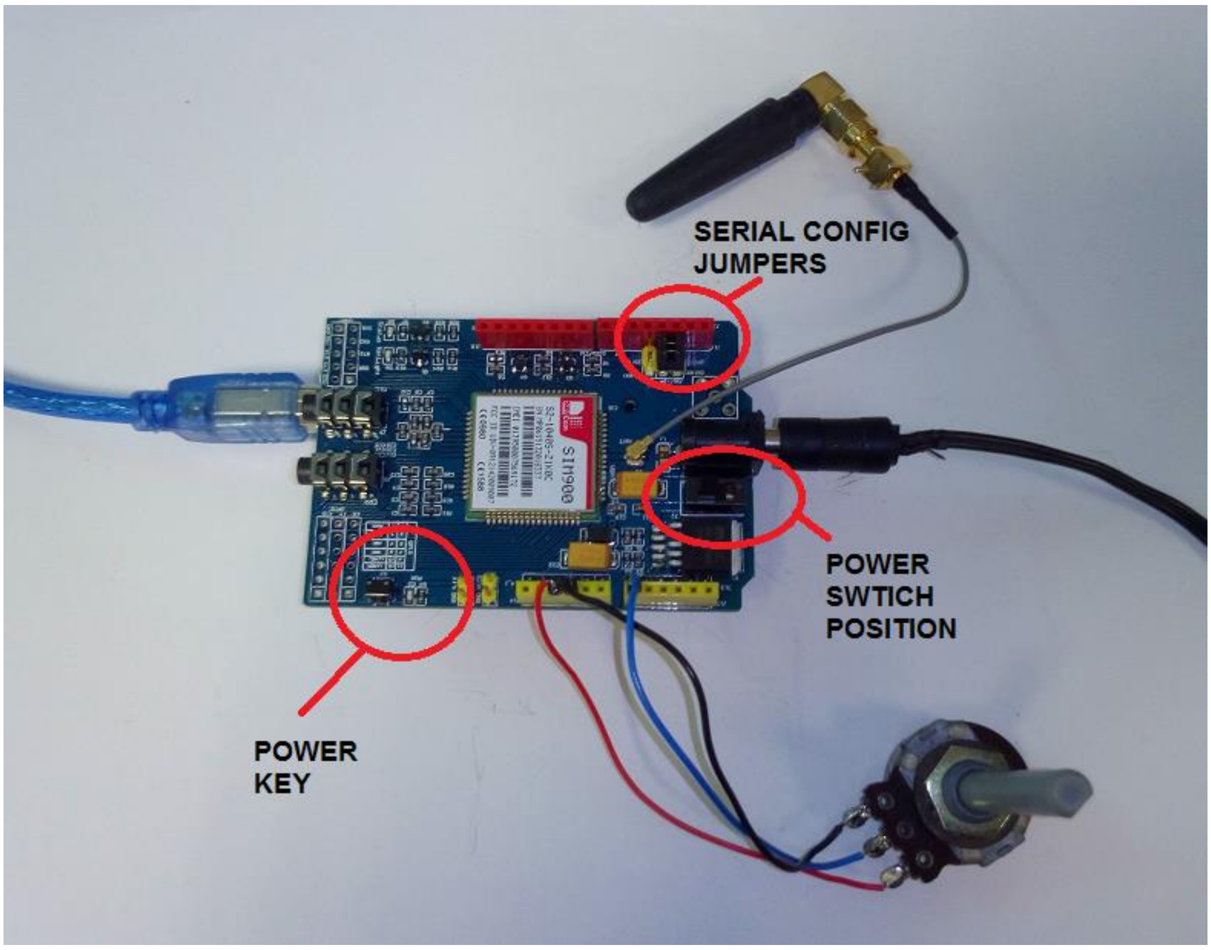
4.1 Set up

<u>ARDUINIO UNO / SIM900 Setup</u>

Pictured below is the SIM900 shield mounted on top of the Arduino UNO (hidden from view).

Note:

1. The required jumper settings to allow software serial to be use IO pins 8 & 7.
2. External power is needed from a 9V to 12V 2A (or better) power supply. Make sure the power switch is the position shown.
3. Always have an antenna connected when powered, not doing so could cause damage to the RF power stage of the modem. The antenna connector has been removed from the board as shown due to short comings in the board layout, not doing so may result in unpredictable operation.
4. Once power is applied the GSM modem will need turning on by pressing the power key.
5. Not visible on the underside is the SIM socket.

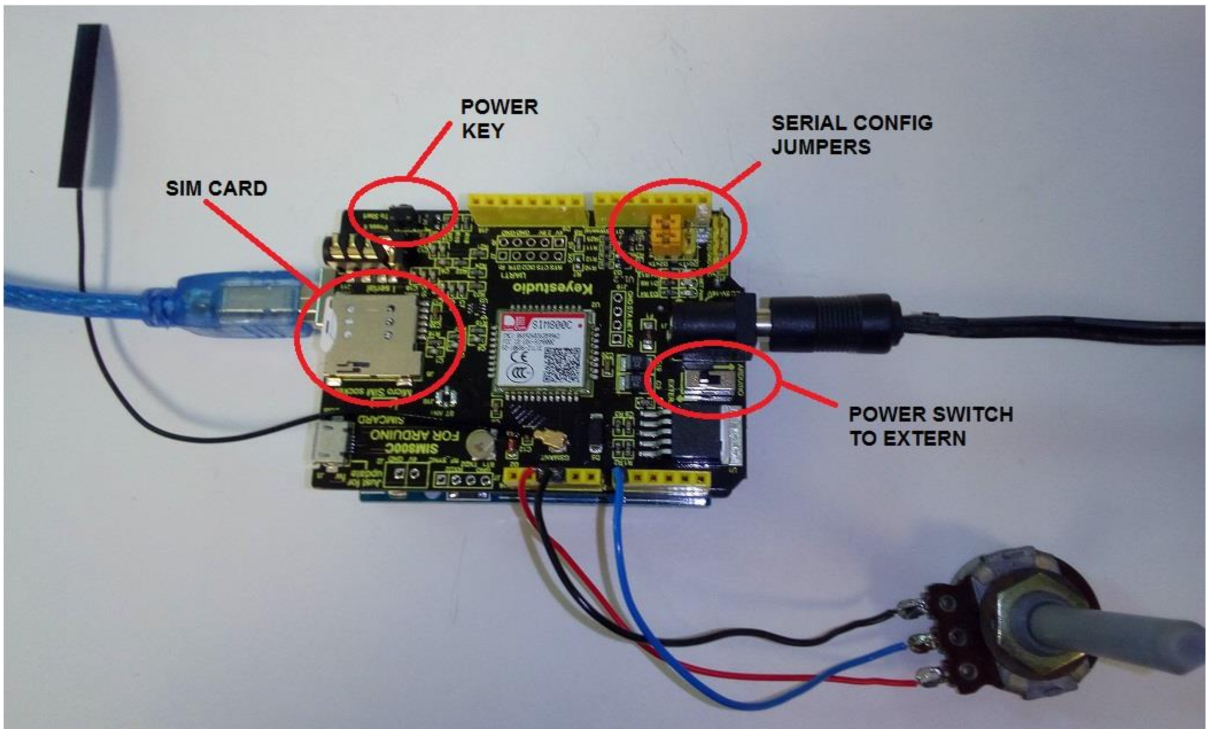


<u>ARDUINIO UNO / SIM800 Setup</u>

Similarly, pictured below is the SIM800 shield mounted on top of the Arduino UNO.

Note:

1. The required jumper settings to allow software serial to be used on IO pin 8 & 7 (these differ to the SIM900).
2. External power is needed from a 9V to 12V 2.5A capable power supply. Make sure the power switch is set to extern. Although the switch can be set to Arduino, this simply won't work as there is not enough current available from the USB to drive the modem.
3. Always have an antenna connected when powered, not doing so could cause damage to the RF power stage of the modem.
4. Once power is applied the GSM modem will need turning on by pressing the power key.



4.2 Source code

The source code is shown below (gsmiot_rev1.ino). All the source code specific to this project has been put into the main sketch rather the complicate matters by splitting it into libraries or modules. The only library used is 'SoftwareSerial' which comes with the normal installation of Arduino IDE.

The down side is there are quite a few lines of code, but to give some clarity these have been split into the following sections:

1. TIMER METHODS
   This section contains useful middleware timer functions for blinking the LED, deciding when to send data and support for GSM communications.
2. GSM METHODS
   A variety of methods for driving the modem with the required AT commands enabling us to send data.
3. APPLICATION CODE
   The likes of setup() and loop(). The code here initialises middleware and performs the function of the device, namely read a sensor (in this case a pot) and sending its value to the server.

The sketch, after configuration, should work out of the box by simply copying and pasting into a new sketch and then compiling/uploading.

You will notice a liberal amount of 'Serial.print()' statements in the code. These output message to the monitor to help trouble shoot any problems that may be encountered.

gsmiot_rev1.ino

```
/* Arduino UNO IoT Device
 * ---------------------
 *
 * Internet of things (IoT) demonstration using Arduino Uno with SIM900 or SIM800
 * shield.
 *
 * Reads a potentionmeter and sends the value to proprietary server.
 *
 *
 *
 * 03-10-2018 Rev 1.0
 * 1. Initial release.
 *
 * ==========================================================================
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * For a copy of the GNU General Public License see:
 * https://www.gnu.org/licenses/.
 * ==========================================================================
 */
#include <SoftwareSerial.h>

//------------//
// CONFIGURE
//------------//
// Set your server URL and API here.
#define      _SRV_URL       "your_host_url"
#define      _SRV_API       "/api.php"

// Set your SIM provider Access point name here.
#define      _APN_NAME      "giffgaff.com"
#define      _APN_USER      "giffgaff"
#define      _APN_PASSWORD  ""

// Device details:
#define      _DEV_ID        "123456"


////////////////////////////////////////////////////////////////////////////////
// TIMER METHODS                                                              //
////////////////////////////////////////////////////////////////////////////////

// Interrupt driver timer counter. Incremented once per millisecond.
static signed long _msTimerCounter;

// Defines the timer variable type.
#define TTimer long

// Define macro to turn on or off timer1 interrupts.
#define _TM1_INTERRUTS_ON TIMSK1 |= (1 << TOIE1)
#define _TM1_INTERRUTS_OFF TIMSK1 &= ~(1 << TOIE1)


/**
 * Initialise timer. Uses hardware timer1.
 *
 */
void timerInit()
{
  // Initialize timer1
  noInterrupts();           // Disable all interrupts
  TCCR1A = 0;
  TCCR1B = 0;

  TCNT1 = 65475;            // Preload timer 65536-(16MHz/256/1000Hz).
  TCCR1B |= (1 << CS12);    // 256 prescaler.
  TIMSK1 |= (1 << TOIE1);   // Enable timer overflow interrupt.
  interrupts();              // Enable all interrupts.
}


/**
 * Timer1 overflow interrupt.
 */
ISR(TIMER1_OVF_vect)
{
  TCNT1 = 65475;             // Preload timer to time out after ~1ms.
  _msTimerCounter++;         // Increment the 1ms timer counter.
}

/**
 * Get safe value of 1ms timer.
 *
 */
long timerGetTimerCounter()
{
  long t1, t2;

  do  {
    t1 = _msTimerCounter;
    t2 = _msTimerCounter;
  } while (t1 != t2);

  // Return value of timer.
  return t1;
}

/**
 * Sets up timer to time out after a number of milliseconds.
 * Use with timedOut()
 *
 * @param timer Pointer to timer variable.
 * @param value Value in milliseconds to time out.
 */
void timeOutAfter(long *timer, long value)
{
  *timer = timerGetTimerCounter() + value;
}


/**
 * Test timer for time out.
 *
 * @return 0:Not timed out, 1:timed out.
 */
int timedOut(long timer)
{
  long diff;

  diff = timer - timerGetTimerCounter();

  if (diff < 0)
    // Timed out.
    return !0;
  else
    // Not timed out.
    return 0;
}



////////////////////////////////////////////////////////////////////////////////
// GSM METHODS  (for SIM900 shield)                                           //
////////////////////////////////////////////////////////////////////////////////

// GSM uses software serial on io pin 7 & 8. Ensure jumpers
// are set correctly on GSM SIM900 or SIM800 shield.
```

```
const byte    GSM_RX_PIN = 7;
const byte    GSM_TX_PIN = 8;

// General purpose GSM line buffer. This is in global space
// to save stack space.
char         _gsmLine[80];

// Define software serial object for GSM.
SoftwareSerial _gsmSer (GSM_RX_PIN, GSM_TX_PIN);

/**
 * Send string to GSM.
 *
 * @param s String to send.
 */
void gsmSend(char *s)
{
  // Out put to monitor.
  Serial.print(F("[GSM TX]:"));
  Serial.println(s);

  // Send to GSM.
  _gsmSer.print(s);
}


/**
 * Send string to GSM and append line feed carriage return.
 *
 * @param s String to send.
 */
void gsmSendLn(char *s)
{
  // Output to monitor.
  Serial.print(F("[GSM TX]:"));
  Serial.print(s);
  Serial.println(F("\\r\\n"));

  // Send the string to GSM.
  _gsmSer.println(s);
}

/**
 * Sends command string to GSM and wait for OK response or timeout.
 *
 * @return 0:No, or wrong response. 1:Success got OK response.
 */
int gsmSendLnWaitOK(char *cmd)
{

  // Send command.
  gsmSendLn(cmd);

  // Get the reponse.
  gsmGetLnWait( _gsmLine, sizeof( _gsmLine), 1000);

  if (strstr(_gsmLine, "OK"))
    return 1;
  else
    return 0;

}

/**
 * Wait for response from GSM. Waits upto msTimeout milliseconds.
 *
 * @param line Line buffer where response will be stored.
 * @param n Size of line buffer.
 * @param msTimeout Max time to wait for response.
 * @return 0:timeout waiting for response. 1: success.
 */
int gsmGetLnWait(char *line, int n, long msTimeout)
{
  char ch;
  char *psz;
  char *pszEnd;
  TTimer msTimer;

  // Initialise.
  psz = line;
  *psz = 0;
  pszEnd = line + n-1;
  timeOutAfter(&msTimer, msTimeout);

  // Collect response line from GSM or timeout.
  while ( 1 ) {
    // Return error if timed out.
    if (timedOut(msTimer)) {
      *psz = 0; // Terminate with null.
      _TM1_INTERRUTS_OFF;
      Serial.print(F("[GSM RX]: "));
      Serial.println(line);
      _TM1_INTERRUTS_ON;
      // Uncomment to reveal time outs.
      //Serial.println("<timed out>");
      return 0;
    }

    // Collect one character.
    if (_gsmSer.available())
      ch = _gsmSer.read();
    else
      // Nothing to collect, go round again.
    continue;

    // Parse collected character.
    switch(ch) {

      // Line feed //
    case '\n':
      // Got a line feed. That means we've received a complete line.
      *psz = 0; // Terminate with null.

      // Ignore blank lines.
      if (line[0] == 0)
        // Try for another.
        continue;

      // Got a non-blank line. Output to monitor and return.
      _TM1_INTERRUTS_OFF;
      Serial.print(F("[GSM RX]:"));
      Serial.println(line );
      _TM1_INTERRUTS_ON;
      return 1; // Success got line.
      break;

      // Carriage return //
    case '\r':
      break;  // Ignore carriage return;

      // All other characters //
    default:
      if ( psz < pszEnd) {
        // Insert char into buffer.
        *psz = ch;
        psz++;
      }
      else {
        // Maxed out buffer.
        *psz = 0;
      }
      break;

    } // end switch
  } // end while ( gsmRxHit() )
}

/**
 * Get connection status.
 */
int gsmGetConnectionStatus(char *line, int n)
{
  // Request connection status.
  gsmSendLn("AT+CIPSTATUS");

  // Get 1st line response.
  gsmGetLnWait(line, n, 1000);     // OK.

  if (gsmGetLnWait(line, n, 1000))
    return 1;
  else
    return 0;
}


/**
 * Get the registration status. Refer to AT command
 * for registration number meanings.
 *
 * @return -1: Error, n:reg status.
 */
int gsmGetRegistrationStatus()
{
    //char line[16];
    int reg;

    // Send the registration command.
    gsmSendLn("AT+CREG?");

    // Get response.
    // Expect "+CREG: 0,1"
    //                  ^- Reg. status.
    if (!gsmGetLnWait(_gsmLine, sizeof(_gsmLine), 500)) {
      // Error, could not get response.
      return -1;
    }
    reg = _gsmLine[9] - '0';

    // Expect OK.
    gsmGetLnWait(_gsmLine, sizeof(_gsmLine), 500);

    // Return registration satus.
    return reg;
}

/**
 * Initiate start of sending data.
 *
 * @return 0:Error, 1:success.
 */
int gsmDataBegin()
{
  //char line[8];

  // Data begin.
  Serial.println(F("Begin data send:..."));
  gsmSendLn("AT+CIPSEND");
  gsmGetLnWait(_gsmLine, sizeof(_gsmLine), 2000);
  if (!strstr(_gsmLine, "> ")) {
    // Error.
    Serial.println(F("Begin data send:fail."));
    return 0;
  }

  // Success.
  Serial.println(F("Begin data send:sucess."));
  return 1;
}

/**
 * End data send.
 *
 * @return 0:Error, 1:Success.
 */
int gsmDataEnd()
{
  //char line[16];

  // Send a 0x1A to invoke sending data thats in the GSM transmit
  // buffer. This can take serveral seconds.
  Serial.println(F("End data send:..."));
  _gsmLine[0] = 0x1A;
  _gsmLine[1] = 0;
  gsmSend(_gsmLine);

  // Expect SEND OK response from GSM. Allow a good half minute for this.
  gsmGetLnWait(_gsmLine, sizeof(_gsmLine), 30000);
  if (!strstr(_gsmLine, "SEND OK")) {
      // Error did not get expected response.
      Serial.println(F("End data send:fail"));
      return 0;
  }

  // Success.
  Serial.println(F("End data send:sucess."));
  return 1;
}


/**
 * Close socket.
 *
 */
int gsmSocketDisconnect()
{
  //char line[10];

  Serial.println(F("Disconnect:..."));
  gsmSendLn("AT+CIPSHUT");
  gsmGetLnWait(_gsmLine, sizeof(_gsmLine), 10000);
  if (!strstr(_gsmLine, "SHUT OK")) {
    // Failed to disconnect.
    Serial.println(F("Disconnect: fail."));
    return 0;
  }

  // Success.
  Serial.println(F("Disconnect success."));
  return 1;
}


/**
 * Connect to socket.
 *
 * @return  0: Fail. 1:success.
 */
int gsmSocketConnect()
{
  //char line[32];
  int i, reg;


  // Wait for registration.
  Serial.println(F("Registration:..."));
  i = 6;
  while(1) {

    reg = gsmGetRegistrationStatus();
    if (reg==1 || reg==5)
      // Great, registered with network. Break from loop.
      break;

    // Another attempt.
    if (i-- == 0) {
      // Too manu attempts. Return error.
      Serial.println(F("Registration: not registered."));
      return 0;
    }

    // Check in 5 seconds time.
    delay(5000);
  }
  Serial.println(F("Registration: registered."));


  // Shut CIP.
  Serial.println(F("Shut:..."));
  gsmSendLn("AT+CIPSHUT");
  if (!gsmGetLnWait(_gsmLine, sizeof(_gsmLine), 500)) {
      Serial.println(F("Shut: fail."));
      return 0;
  }

  if (!strstr(_gsmLine, "SHUT OK")) {
      Serial.println(F("Shut: fail."));
      return 0;
  }

  // Attach.
  Serial.println("Attach:...");
  if (!gsmSendLnWaitOK("AT+CGATT=1")) {
      Serial.println(F("Attach: fail."));
      return 0;
  }
  Serial.println(F("Attach: success."));
  gsmGetConnectionStatus(_gsmLine, sizeof(_gsmLine));


  // Define PDP context.
  Serial.println(F("Set PDP:..."));
  sprintf(_gsmLine, "AT+CGDCONT=1,\"IP\",\"%s\"", _APN_NAME );
    if (!gsmSendLnWaitOK(_gsmLine)) {
    Serial.println(F("Set PDP fail."));
    return 0;
  }
  Serial.println(F("Set PDP: success."));
  gsmGetConnectionStatus(_gsmLine, sizeof(_gsmLine));

  // Set APN.
  Serial.println(F("Set APN:..."));
  sprintf(_gsmLine, "AT+CSTT=\"%s\",\"%s\",\"%s\"", _APN_NAME, _APN_USER, _APN_PASSWORD);
  if (!gsmSendLnWaitOK(_gsmLine)) {
    Serial.println(F("Set APN fail."));
    return 0;
  }
  Serial.println(F("Set APN: success."));
  gsmGetConnectionStatus(_gsmLine, sizeof(_gsmLine));

  // Bring up wirless. This can take many seconds.
  Serial.println(F("Bring up wireless:..."));
  gsmSendLn("AT+CIICR");
  gsmGetLnWait(_gsmLine, sizeof(_gsmLine), 30000);
  if (!strstr(_gsmLine, "OK")) {
    Serial.println(F("Bring up wireless: fail."));
    return 0;
  }
  Serial.println(F("Bring up wireless: success."));

  // Get local IP address.
  gsmSendLn("AT+CIFSR");
  gsmGetLnWait(_gsmLine, sizeof(_gsmLine), 500);
  gsmGetConnectionStatus(_gsmLine, sizeof(_gsmLine));


  // Start TCP connection.
  Serial.println(F("Start TCP connect:..."));
  gsmSend("AT+CIPSTART=\"TCP\",\"");
  gsmSend(_SRV_URL);                          // _SRV_URL
  gsmSendLn("\",\"80\"");                // Port.

  // Expect OK.
  gsmGetLnWait(_gsmLine, sizeof(_gsmLine), 30000);
  if (!strstr(_gsmLine, "OK")) {
    Serial.println(F("Start TCP connect: fail."));
    return 0;
  }
  // Expect CONNECT OK.
  gsmGetLnWait(_gsmLine, sizeof(_gsmLine), 30000);
  if (!strstr(_gsmLine, "CONNECT OK")) {
    Serial.println(F("Start TCP connect: fail."));
    return 0;
  }
  Serial.println(F("Start TCP connect: success."));

  // Success.
  return 1;
}


/**
 * Initialise GSM.
 *
 * @return 0:Initialise error, 1:Success.
 */
int gsmInit()
{
  int tries = 0;

  // Open serial communications to GSM and initialise.
  _gsmSer.begin(9600);

  // Loop until got AT response.
  while (1) {
    if (tries++ > 5) {
      Serial.println(F("Is the GSM powered up?"));
      tries = 0;
    }

    // Send AT command.
    Serial.println(F("Initialise GSM:..."));
    _gsmSer.flush();
    gsmSendLn("AT");

    // Expect echoed AT or OK response.
    if (!gsmGetLnWait(_gsmLine, sizeof(_gsmLine), 1000) != 0)
      // Timed out waiting for response. Try again.
      continue;

    if (strstr(_gsmLine, "OK")) {
      // Great! Echo is turned off, all done.
      break;
    }

    if (strstr(_gsmLine, "AT")) {
      // Echo not off so lets turn it off.
      _gsmSer.flush();
      gsmSendLn("ATE0");
    }
  }

  // Text mode.
  gsmSendLnWaitOK("AT+CMGF=1");

  // Sucess.
  Serial.println(F("Initialise GSM: success"));
  return 1;
}


////////////////////////////////////////////////////////////////////////////////
// APPLICATION CODE                                                           //
////////////////////////////////////////////////////////////////////////////////

// Timer for LED blink one second blink.
TTimer        _ledTimer;

// Timer for server update rate.
TTimer        _serverUpdateTimer;

// Define pin for LED.
#define       _LED_PIN 13

// Define pin used for reading analogue potentionmeter.
#define       _POT_PIN 0

/**
 * Send sensor data to server.
 *
 * @param field1 Field 1 sensor value to send to server.
 * @return 0:failed to send, 1: successful send.
 */
int sendSensorDataToServer(int field1)
{
  char line[40];
  int len;
  char lenStr[4];


  //-------------------
  // Connect to socket.
  //--------------------
  _gsmSer.flush();
  if (!gsmSocketConnect())
    return 0;

  //-----------
  // Send data.
  //-----------

  // Begin data send.
  if (!gsmDataBegin()) {
    // Error occured.
    return 0;
  }

  // Assemble HTTP POST data string.
  sprintf(line, "device_id=%s&field1=%d", _DEV_ID, field1);
  len = strlen(line);
  sprintf(lenStr, "%d", len);
  // Send the HTTP POST header.
  gsmSend("POST "); gsmSend(_SRV_API); gsmSendLn(" HTTP/1.1");
  gsmSend("Host: "); gsmSendLn(_SRV_URL);
  gsmSendLn("Connection: close");
  gsmSendLn("Content-Type: application/x-www-form-urlencoded");
  gsmSend("Content-Length: "); gsmSendLn(lenStr);
  gsmSendLn("");

  // Send HTTP data.
  gsmSendLn(line);


  if (!gsmDataEnd()) {
    // Error ending data send.
    return 0;
  }

  //-------------------------------------
  // Collect response from server, if any.
  //-------------------------------------
  Serial.println(F("\r\nServer response:..."));
  while (1) {

    if (!gsmGetLnWait(line, sizeof(line), 1000)) {
      // Timed out. No response or no more lines sent by server.
      break;
    }
    // This is where to process the received line of data if needed.

  }
  Serial.println(F("Server response: end\r\n"));


  //--------------------
  // Disconnect socket.
  //--------------------
  if (!gsmSocketDisconnect())
    return 0;
  else
    return 1;
}

/**
 * Setup. Execution starts here.
 */
void setup()
{
  // Set the LED pin and timer.
  pinMode(_LED_PIN, OUTPUT);
  timeOutAfter(&_ledTimer, 5000);

  // Initialise the timer. General purpose timer.
  timerInit();

  // Initialise the monitor.
  Serial.begin(9600);
  while(!Serial);

  // Initialise GSM.
  gsmInit();

  // Set initial server update to take place in 5 seconds.
  timeOutAfter(&_serverUpdateTimer, 5000);
}

/**
 * Main program loop.
 */
void loop()
```

```
{
  int potVal;


  // Every second.
  if (timedOut(_ledTimer) ) {
    // Reload timer counter.
    timeOutAfter(&_ledTimer, 1000);

    // Toggle LED.
    digitalWrite(_LED_PIN, digitalRead(_LED_PIN) ^ 1);
  }


  // Send sensor data to server every minute.
  if (timedOut(_serverUpdateTimer) ) {
    // Set to time out in one minute.
    timeOutAfter(&_serverUpdateTimer, 60000);

    // Read the value of the potentiometer (sensor) and send it to the
    // server.
    potVal = analogRead(_POT_PIN);
    sendSensorDataToServer( potVal);
  }

}
```

4.3 Configure sketch

Below, found at the top of sketch are the configuration settings for the server and SIM access point name (APN). You will need to set these for your own server and APN. An example is shown for GiffGaff. The device ID can also be set here with '_DEV_ID' which appears in the data base every time data is send. The 'your_host_url' is the url you are using with the 'www.' bit omitted, otherwise it won't work.

Snippet from 'gsmiot_rev1.ino'

```
//------------//
// CONFIGURE
//------------//
// Set your server URL and API here.
#define      _SRV_URL       "your_host_url"
#define      _SRV_API       "/api.php"

// Set your SIM provider Access point name here.
#define      _APN_NAME      "giffgaff.com"
#define      _APN_USER      "giffgaff"
#define      _APN_PASSWORD  ""

// Device details:
#define      _DEV_ID        "123456"
```

4.4 Making it work

By now you should:

1. Have a host and uploaded you scripts to the server and tested as per section 2.
2. Setup your Arduino UNO with a SIM900/800 shield plugged in with the correct jumper setting.
3. Known working SIM card in the shield.
4. Have the sketch configured correctly.

5. Have the shield using an external power supply capable of delivering at least 9V at 2A.

Next do the following:

1. Open up a browser and enter the web address of your dashboard. This will be something like 'www.your_host_url/dashboard.php'. This page will automatically update every 10 seconds.
2. Power up the Arduino and GSM shield.
3. Start the Arduino IDE and upload the sketch.
4. Open the monitor on the Arduino IDE from 'Tools > Serial Monitor'. The baud rate should be set to 9600.
5. Press the power key on the shield. On both the SIM900 and 800 a power LED will come on. Another LED shows the state of the SIM registration which should settle down to one flash every three or four seconds.
6. Observe the monitor and confirm the message 'Initialise GSM: success' has appeared.
7. After a few seconds to a minute the device will send the latest potentiometer (or pot) reading to the server. There are a great many message output to the monitor. The last message should say 'Disconnect success'. Then the whole thing repeats again in another minute.
8. Observe the dashboard on your browser, it should now be updated with the latest pot value.

4.5 Trouble shooting

It is rare, if ever, this will work without a hitch. There are a great many things that can go wrong. Below is a list of problems and possible solutions.

Problem	**Possible reason**	**Solution**
Monitor message: 'Is the GSM powered up?' keeps repeating.	Jumpers incorrectly set on shield.	Check jumpers.
	Shield not powered up.	Check external power connected. Press power key on shield.
	Shield power switch in wrong position.	Move power switch to correct position.
Monitor message: 'Registration: not registered'.	SIM card not plugged in.	Insert SIM card.
	SIM card has no credit.	Test SIM card in phone.
	Antenna not connected.	Connect antenna
	Modem RF power amplifier damage due to lack of antenna.	Replace shield.
	Power supply inadequate.	Ensure power supply can deliver 9V at 2A.
Monitor message: 'Set APN fail'.	APN configuration incorrect.	Check APN configuration correctly set in sketch. See #defines _APN_NAME etc.
Monitor message: 'Bring up wireless: fail'.	Power supply in adequate.	Ensure power supply can deliver 9V at 2A.
Monitor message: 'Start TCP connect: fail'.	Incorrectly set server URL.	Check server configuration in sketch. See #define _SRV_URL etc.

Server doesn't update.	Server script not correctly installed.	Refer to API test section 3.2.2.
	SIM card out of credit.	Check SIM card in working phone ant that it has credit.
Server response looks garbled when using SIM800 shield.	Arduino 'SoftwareSerial' limited buffer space/not full duplex.	Potential solution is to upgrade to Arduino Mega and use hardware serial ports throughout.
Just doesn't seem to connect to network reliably.	Poor signal quality.	Test SIM in phone in similar position to device. Check signal strength is good.
	Network problems.	Try later. Cycle power on UNO/GSM.

5 Conclusion

We have shown here how to implement a complete end-to-end IoT solution using 'free' servers, some Arduino hardware and a bit of code. We did this without having to use a proprietary IoT platform free or otherwise at very little cost.

However, there are some limitations.

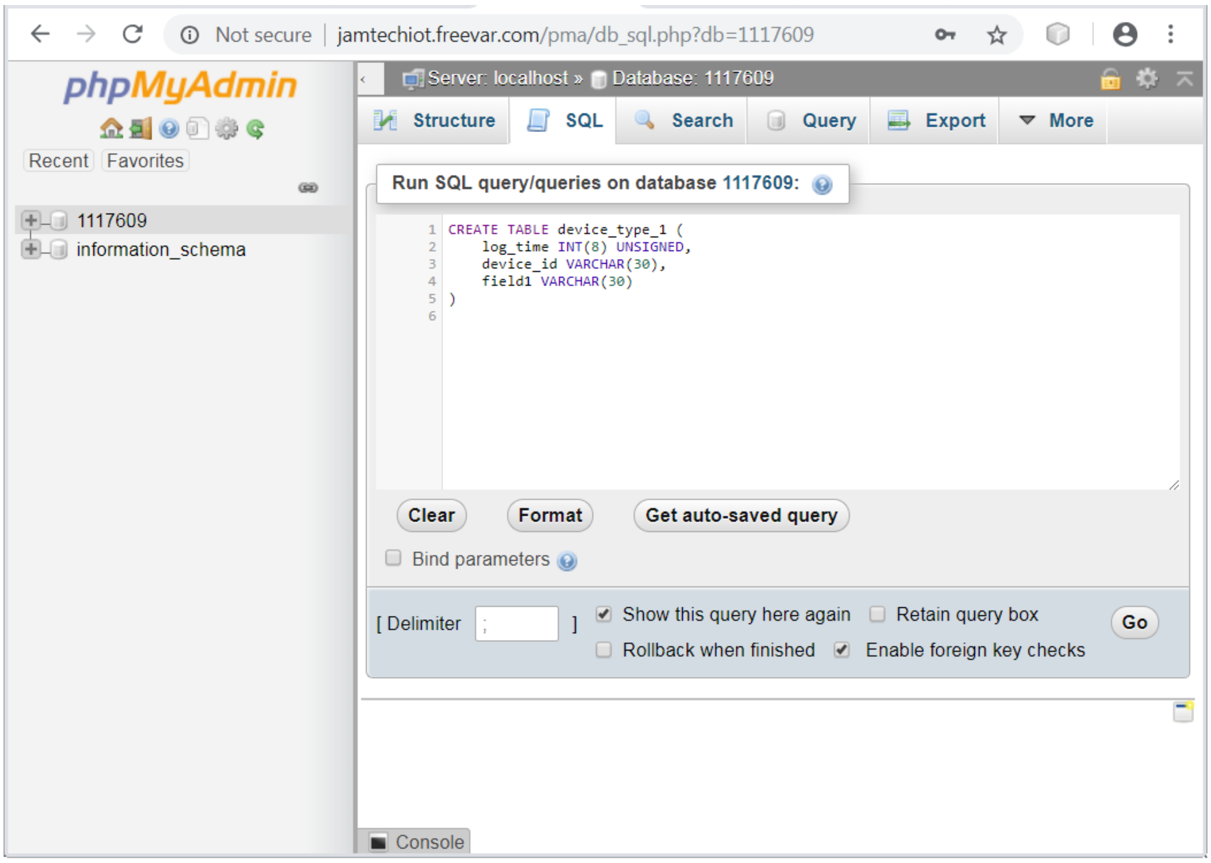
1. The API script ('api.php') is anything but secure and is vulnerable, not least to SQL injection attack. There are ways around this by writing a better script, examples of mitigating SQL injection and the tools for validation can be found on the net. The script does not vet the device which could lead to an attacker spoofing the device.
2. The data we send is plain text and can therefore be viewed by a 'man in the middle'. This can be solved by encryption at the device and decryption at the server. The UNO is unlikely to have the resources to do this adequately.
3. The UNO / Shield setup is not stand alone. You can't just plug it in turn it on, kind of. When the power is cycled, the power key on the shield must be pressed to power it up. This is not much use in the field if the power cycles and no one is there to 'press the power key'. This can be overcome by bringing out the power key line to an IO pin on the UNO and get the software to effectively 'press the key' and sense if the GSM is responding or not by talking to it.
4. Uses a lot of power, relatively. Not much good on a battery if the server is updated once per day, most of the time the modem is on when it needn't be. This can be solved with some form of power control. Plus the UNO would ideally need to go to sleep, if this is possible.
5. Not a biggie, but this set up won't work without the USB connected to a PC. This can be resolved by applying power externally to the UNO and removing all the 'Serial.print' statements from the code.
6. Server response is garbled when using the SIM800 shield. This either is or isn't a problem. If you want to process the server response then you may well have problems even when turning of the 'Serial.print' statements. It's not the SIM800, as the server response is correct whensniffed with Tera Term. The problem is the UNO soft serial not keeping up with the output of the 800. I suspect the 900 has more transmission delay between characters allowing it work alright. I did try to solve the problem by turning off and on timer interrupts here and there which did improve things a little. The solution really is to upgrade the UNO to a Mega which is faster, has more memory and extra hardware serial ports.

Well, I hope you have enjoyed this project. It is certainly rewarding to do a lot with so little and dispel some of the mystery around IoT.

6 APPENDIX

6.1 phpMyAdmin 'SQL' tab

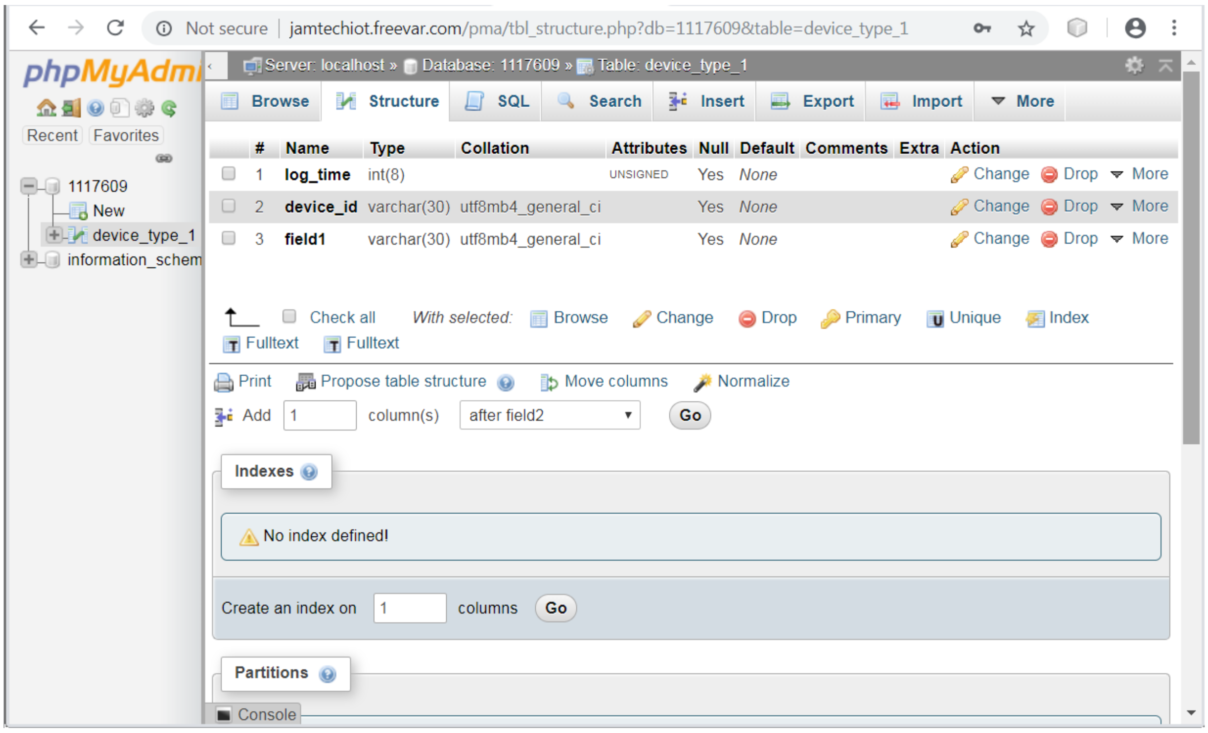
Screen shot below of phpMyAdmin showing SQL tab with table creation script pasted on. Click go to execute script.



```
CREATE TABLE device_type_1 (
    log_time INT(8) UNSIGNED,
    device_id VARCHAR(30),
    field1 VARCHAR(30)
)
```

6.2 phpMyAdmin 'Structure' tab

Screen shot below of table structure created by script.


Not secure | jamtechiot.freevar.com/pma/tbl_structure.php?db=1117609&table=device_type_1
phpMyAdmin
Recent Favorites
1117609
New
device_type_1
information_schem
Server: localhost » Database: 1117609 » Table: device_type_1
Browse Structure SQL Search Insert Export Import More
Check all With selected: Browse Change Drop Primary Unique Index Fulltext Fulltext
Print Propose table structure Move columns Normalize
Add 1 column(s) after field2 Go
Indexes
No index defined!
Create an index on 1 columns Go
Partitions
Console

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	log_time	int(8)		UNSIGNED	Yes	None			Change Drop More
2	device_id	varchar(30)	utf8mb4_general_ci		Yes	None			Change Drop More
3	field1	varchar(30)	utf8mb4_general_ci		Yes	None			Change Drop More

Disclaimer:

Although every care has been taken to ensure the accuracy of the information in this article, such information is provided "as-is" and without any warranty or implied fitness for any purpose. In acting on any information provided by this article you accept that this is entirely at your own risk and indemnify the author against any loss or damages, actual or consequential that may occur.