Truong Hong Kha

Temperature and humidity monitor with ESP8266

Thesis 2019
Abstract

Truong Hong Kha
Temperature and humidity monitor with ESP8266, 32 pages, 2 appendices
Saimaa University of Applied Sciences
Technology, Lappeenranta
Bachelor of Engineering
Mechanical Engineering and Production Technology
Thesis 2019
Instructors: Lecturer Jukka Nisonen, Saimaa University of Applied Sciences

The main objective of this report is to design and provide implementation in details of IoT based ESP8266 to monitor environmental parameters. It also provides knowledge in IoT term and other solutions for this specific project. Besides using the final prototype, this report also listed other approaches and analyzed each of them.

This study used ESP8266 for both roles: Wi-Fi module and microcontroller. In the server side, it is hosted in https://000webhostapp.com

The proposed system can be used to track eight parameters only. To have a larger system with higher scalability, a design requiring more complexity in the server side can be considered.

Keywords: IoT, Internet of Things, ESP8266, NodeMCU
Table of contents

1 Introduction ........................................................................................................................................ 5
2 Components and modules used........................................................................................................ 6
   2.1 Software: Arduino IDE............................................................................................................... 6
      2.1.1 Introduction....................................................................................................................... 6
      2.1.2 The properties of Arduino IDE ......................................................................................... 6
      2.1.3 Arduino programming language ....................................................................................... 7
   2.2 Hardware ................................................................................................................................... 7
      2.2.1 NodeMCU ESP8266............................................................................................................ 7
      2.2.2 DHT11 ............................................................................................................................... 14
3 System model in details ................................................................................................................. 16
   3.1 Architecture............................................................................................................................. 16
      3.1.1 HTTP protocol .................................................................................................................. 18
      3.1.2 Model explanation ............................................................................................................ 22
   3.2 The whole circuit setup ............................................................................................................ 29
Reference........................................................................................................................................... 31
Appendix One – Setup Arduino IDE environment for ESP8266 ......................................................... 33
   Preparing Arduino IDE environment for ESP8266 ....................................................................... 33
   Installing Arduino IDE .................................................................................................................. 33
   Installing driver ............................................................................................................................. 33
   Installing the ESP8266 Board within Arduino IDE ......................................................................... 34
Appendix Two – Source code ............................................................................................................ 38
   Showdht11.php .............................................................................................................................. 38
   Update.php .................................................................................................................................... 40
   DHT11SentToServer.ino ................................................................................................................. 41
List of figures

Figure 1. NodeMCU v3 [4] .................................................................................................................. 8
Figure 2. ESP with Antenna ............................................................................................................. 8
Figure 3. NodeMCU with power supply ......................................................................................... 9
Figure 4. Station Mode [5] ............................................................................................................. 10
Figure 5. Soft Access Pont Mode [5] ............................................................................................ 11
Figure 6. NodeMCU Pinout ........................................................................................................... 13
Figure 7. DHT11 comparing with a penny ...................................................................................... 14
Figure 8. DHT 11 sensor pinout ..................................................................................................... 15
Figure 9. Typical wiring .................................................................................................................. 16
Figure 10. System diagram ........................................................................................................... 17
Figure 11. From user perspective .................................................................................................. 17
Figure 12. Data flow ....................................................................................................................... 17
Figure 13. GET request to showdht11.php .................................................................................... 24
Figure 14. GET request to data.json .............................................................................................. 26
Figure 15. Server response content ............................................................................................... 27
Figure 16. Server response value .................................................................................................. 28
Figure 17. Server respond value corresponding code .................................................................... 29
Figure 18. The whole circuit ......................................................................................................... 29
Figure 19. The main page to download the Arduino IDE ............................................................... 33
Figure 20. CP2102 driver missing .................................................................................................. 34
Figure 21. Adding ESP8266 URL into Arduino IDE ...................................................................... 35
Figure 22. Installing ESP8266 board ............................................................................................. 36
Figure 23. Selecting ESP8266 board ............................................................................................. 37
1 Introduction

For recent years, collecting and monitoring data have been developed rapidly. Together with the improvement in both device performance and cost, this demand is increasing and satisfying day by day. These devices can vary in many aspects, but they all follow a principle: they measure physical parameters and translate it into values.

The “IoT” term is often mentioned recently. Having devices connected has never been that easy. These sensors used to be interconnected by wired connections. With the development of wireless technology, more and more devices are connected via TCP/IP protocol or act as a station in a Wi-Fi network. Before such a device as ESP8266 was developed, to have this network connecting to Internet wirelessly, we need to have at least two devices: a processor (like Arduino Mega or Nano) and a Wi-Fi module. Thanks to Espressif Systems (a Shanghai-based Chinese manufacturer), these two functionalities are now integrated in simply one module, which will ease the work of developers.

But not only the devices and client side, the work in server side is also easier. There is no need to build a whole server to have a prototype version of an IoT system. There are several providers out there can help us to work with the server, one of them is https://000webhostapp.com.

This thesis uses DHT11 sensor to measure temperature and humidity values, ESP8266 to transmit data and 000webhostapp to host the server and visualize data. Not only the practical work, it also presents a basic knowhow of an IoT system and how to quickly setup a prototype, different approaches of processor and data acquisition.
2 Components and modules used

2.1 Software: Arduino IDE

2.1.1 Introduction

AVR controllers and PIC microcontrollers are increasingly common and more complete but we can say the appearance of Arduino in 2005 in Italy opened a new direction for microcontrollers. The appearance of Arduino has eased the work of people in programming and design. According to the main website of Arduino, Arduino is using the electronics platform as an open-source base to help user easy to use hardware and software. Arduino is an IDE with the built-in editor, compiler, programmer and it comes with firmware with the boot loader, built-in library kits and easy integration. It means whenever you have an idea and want to build something, you can just focus on designing and programming and not concern totally about the hardware and electronics stuff.

The language used is C / C++. All are open source and contributed by the community. There are several parts as Arduino Boards, Arduino programming language (based on Wiring) and Arduino Software IDE (based on Processing).

2.1.2 The properties of Arduino IDE

- Arduino hides the complexity of electronics with simple concepts. Setting up an output for a MCU by setting the register is so complicated that a professional even has to open a data sheet. With Arduino, a function is called and everything is solved.
- Because of its common, the user only needs to focus on product features rather than protocol, datasheet. It means anyone can make great products without much knowledge in electronics field.
- Debug by console.
- The IDE is well designed that it can integrate many types of compilers easily and a variety of hardware without losing performance.
2.1.3 Arduino programming language

The Arduino programming language has compiled on the Arduino IDE. It can be downloaded from https://www.arduino.cc/.

This programming language is based on a simple hardware programming language called Processing, which is very similar to C language [1].

2.2 Hardware

2.2.1 NodeMCU ESP8266

2.2.1.1 Introduction about ESP8266

To introduce about ESP8266, an assumption is made that you already know about Android. ESP8266 is used to be used as an Android extension to get access to the WiFi network. This is also the main functionality of its design – to provide WiFi connection for multi electronic products. [2]

But in fact, it is a System on Chip (SoC) manufactured by Espressif. Not only providing WiFi connection, it also comes up with a microcontroller and a lot of spare memory, so ESP8266 is barely used as a standalone solution. This means that it can be programmed like any normal microcontroller. [3]

ESP8266 is normally sold on the development boards such as NodeMCU. This means once it is under used, there is no programming hardware required. It comes with a USB-to-Serial converter, 3.3V converter. The only thing user has to care about is to plug the power, connect the board with another device and learn how to use it.

There are some common applications that it can be used in IoT domain, such as connecting sensor to the Internet, hosting a web server with real web pages, letting the smartphone connect to it.
2.2.1.2 NodeMCU

NodeMCU is chosen in this paper to ease the work of setup and installing accessories, since it has all kinds of features on-board: a USB-to-Serial converter for programming, a 3.3V regulator for power, on-board LEDs for debugging, a voltage divider to scale the analog input.

Figure 1. NodeMCU v3 [4]

Figure 2. ESP with Antenna
The ESP8266 integrates WiFi transceiver, so it not only connects to a WiFi network and interacts with the Internet through that network, but also sets up a network on its own. This topic will be discussed further in Operating mode section.

Figure 3. NodeMCU with power supply

The board can be powered via the on-board Micro USB connector. Otherwise, the VIN Pin can come in handy by directing supply a 5V voltage source

2.2.1.3 Operating mode

In this project, ESP8266 will play a role of a Web Client. However, thanks to its WiFi transceiver, it can also play a role of Web Server by setting up a network itself and allowing other devices to connect to it and access web page.

To describe it in detail, we need to understand three modes of ESP8266: Station mode, Soft Access Point mode, and both modes at once.

2.2.1.3.1 Station Mode

Any device that connects to a network (in this case, this is a WiFi network established by a wireless router) is called a Station.
2.2.1.3.2 Soft Access Point Mode

First, we should understand what Access Point is. Any device that creates its own network and acts as a hub (such as WiFi router) is called an Access Point. So why is there “Soft” in this term? Since this NodeMCU interface is not available for a wired network, such an operation is so called Soft Access Point. One more thing, the limit number of stations under this network is limited to five devices.

Figure 4. Station Mode [5]
Figure 5. Soft Access Pont Mode [5]

2.2.1.4 Specifications

<table>
<thead>
<tr>
<th>Specification of ESP8266</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Chip</strong></td>
</tr>
<tr>
<td>Frequency</td>
</tr>
<tr>
<td>RAM</td>
</tr>
<tr>
<td>Flash memory</td>
</tr>
<tr>
<td><strong>Power</strong></td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Multiplexed I/Os</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Serial communication</strong></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
2.2.1.5 Pin diagram

![NodeMCU Pinout](image)

Figure 6. NodeMCU Pinout

This MCU has total 30 pins that connect to the world.

2.2.1.6 Development platform

There are many platforms that can be used to program ESP8266. Some names in the list are Espruino – which is JavaScript SDK, Mongoose OS – an OS for IoT devices. Other pure programing languages are Processing, MicroPython.

This thesis will be concentrating mainly on programming with Arduino IDE. In Appendix One - Setup Arduino IDE environment for ESP8266, there is a guide on how to establish an Arduino IDE environment for ESP8266.
2.2.2 DHT11

2.2.2.1 Introduction

Figure 7. DHT11 comparing with a penny


Processor: 8-bit single-chip.

Small size and low consumption together with long transmission distance (up to 20 m) enable DHT11 to be well suited to operate in harsh environments.

2.2.2.2 Technical Specifications

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temperature Range</td>
<td>0 – 50°C / ± 2°C</td>
</tr>
<tr>
<td>Humidity Range</td>
<td>20 – 80% / ± 5%</td>
</tr>
<tr>
<td>Sampling Rate</td>
<td>1Hz</td>
</tr>
<tr>
<td>Body Size</td>
<td>15.5mm x 12mm x 5.5mm</td>
</tr>
<tr>
<td>Operating Voltage</td>
<td>3 – 5V</td>
</tr>
</tbody>
</table>
Max Current During Measuring | 2.5mA

Electrical characteristics

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Conditions</th>
<th>Min</th>
<th>Typical</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Supply</td>
<td>DC</td>
<td>3V</td>
<td>5V</td>
<td>5.5V</td>
</tr>
<tr>
<td>Current Supply</td>
<td>Measuring</td>
<td>0.5mA</td>
<td></td>
<td>2.5mA</td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>0.2mA</td>
<td></td>
<td>1mA</td>
</tr>
<tr>
<td></td>
<td>Standby</td>
<td>100µA</td>
<td></td>
<td>150µA</td>
</tr>
<tr>
<td>Sampling period</td>
<td></td>
<td>1s</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2.2.2.3 Pin diagram

![DHT 11 sensor pinout](image)

Figure 8. DHT 11 sensor pinout
Figure 9. Typical wiring

Pin configuration:

<table>
<thead>
<tr>
<th>Pin</th>
<th>Label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VCC</td>
<td>Power supply (from 3.5V to 5.5V)</td>
</tr>
<tr>
<td>2</td>
<td>Data</td>
<td>Temperature and humidity value</td>
</tr>
<tr>
<td>3</td>
<td>Ground</td>
<td>Connected to the ground</td>
</tr>
</tbody>
</table>

3 System model in details

3.1 Architecture

Scope of system: even it is an IoT project but we limit the scope of work to monitor the live data only. There will be no analysis, BI report or machine learning phase in the project.

URL to see chart: [https://khamec14thesis.000webhostapp.com/showdht11.php](https://khamec14thesis.000webhostapp.com/showdht11.php)
Figure 10. System diagram

Figure 11. From user perspective

Figure 12. Data flow
3.1.1 HTTP protocol

3.1.1.1 At a glance
Before going to HTTP protocol, there are terms “Web server” and “Client” needed to be explained. Web server is a place which stores, processes and delivers web pages to Web clients. Web client is nothing but a web browser on our laptops and smartphones. The communication between client and server takes place using a special protocol called Hypertext Transfer Protocol (HTTP) [5].

By its nature, HTTP is stateless. “Stateless” means that all requests are separated, so each of them must include enough information to fulfill the request, and each message has no idea what happened with previous messages [7].

3.1.1.2 In details
HTTP is a request-response client-server protocol as illustrated. An HTTP client sends a request message to an HTTP server. The server, in turn, returns a response message.

Interaction between client and server: whenever user accesses an URL from web browser
(e.g. type "https://github.com/" in URL bar), the browser turns the URL into a request message and sends that message to HTTP Server. The HTTP Server interprets (translate) the message, and responds that request with an appropriate message (can be a HTML page or error code or a service) [8].

![HTTP Request Diagram](image)

### 3.1.1.3 HTTP request

To start the communication, the client establishes an HTTP session by opening a connection to HTTP Server, then sending a request to that server. There are several ways to make a request: directly by entering a URL, or indirectly as a video is embedded in a webpage, anytime user request to a webpage, he pulls a request to that video as well.

<table>
<thead>
<tr>
<th>Request Header</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Request-Line)</td>
<td>GET / HTTP/1.1</td>
</tr>
<tr>
<td>Host</td>
<td><a href="http://www.stdio.vn">www.stdio.vn</a></td>
</tr>
<tr>
<td>User-Agent</td>
<td>Mozilla/5.0 (Windows NT 6.3; WOW64; rv:36.0) Gecko/20100101 Firefox/36.0</td>
</tr>
<tr>
<td>Accept</td>
<td>text/html,application/xhtml+xml,application/xml;q=0.9,<em>/</em>;q=0.8</td>
</tr>
<tr>
<td>Accept-Language</td>
<td>vi-VN,vt;q=0.8,en-US;q=0.5,en;q=0.3</td>
</tr>
<tr>
<td>Accept-Encoding</td>
<td>gzip, deflate</td>
</tr>
<tr>
<td>Cookie</td>
<td>_ga=GA1.2.525740874.1425574792; PHPSESSID=0e61l2eb0mff46hej7r01fms2; _gat=1; ci_session=a%3...</td>
</tr>
<tr>
<td>Connection</td>
<td>keep-alive</td>
</tr>
</tbody>
</table>

Starting of an HTTP request is the Request-Line with three data:

- **Method**: is the method in used. Usually they are GET, POST. There are HEAD, PUT, DELETE, OPTION, CONNECT also.
- **URI**: is the resource requested, can be blank (as the example above).
• HTTP version: two versions are currently in use: HTTP/1.0 and HTTP/1.1.

Some examples:

• GET /test.html HTTP/1.1
  HEAD /query.html HTTP/1.0
• POST /index.html HTTP/1.1

The Request-Line is followed by Request Header fields, allowing user to send extra data about the HTTP request or the client himself. Some common fields:

• `Accept`: content type accepted from respond message.
• `Accept-Encoding`: compressed type accepted.
• `Connection`: custom control for the current connection.
• `Cookie`: HTTP Cookie information from the server.

**HTTP response**

<table>
<thead>
<tr>
<th>Response Header</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Status-Line)</td>
<td>HTTP/1.1 304 Not Modified</td>
</tr>
<tr>
<td>Date</td>
<td>Tue, 17 Mar 2015 14:15:52 GMT</td>
</tr>
<tr>
<td>Server</td>
<td>Apache/2</td>
</tr>
<tr>
<td>Connection</td>
<td>Keep-Alive</td>
</tr>
<tr>
<td>Keep-Alive</td>
<td>timeout=1, max=100</td>
</tr>
<tr>
<td>Etag</td>
<td>&quot;b8c1e-2275-5115f173d1e40&quot;</td>
</tr>
<tr>
<td>Vary</td>
<td>Accept-Encoding, User-Agent</td>
</tr>
</tbody>
</table>

HTTP response structure is similar to HTTP request. The main difference is the Request-Line: HTTP response has the Status-Line instead. There are three parts in Status-Line also:

• `HTTP-version`: the highest HTTP version that server supports.
• `Status-Code`: the code of respond message.
• `Reason-Phrase`: description of Status-Code.
3.1.1.4 Methods
HTTP protocol defines a set of methods to have client to use these methods in request message in communicating with server. Most common methods are GET and POST, however there are more of them.

GET — used to request data from a specified resource where data is not modified it in any way as GET requests do not change the state of resource.

POST — used to send data to a server to create a resource.

3.1.1.5 HTTPS
Secure version of HTTP is HyperText Transfer Protocol Secure (HTTPS). HTTPS provides encrypted communication between a client and the server.

Here is how HTTPS secures the communication: it uses a public key and a private key to encrypt a message. Anyone can use the public key to encrypt a message. But the private key is secret, so only the intended receiver can decrypt the message. [7]

The huge benefit of HTTPS is encrypting user’s information, like credit card number or sensitive information.
3.1.2 Model explanation

3.1.2.1 Transfer data

3.1.2.1.1 Establish WIFI connection

After all devices are connected, we can move to compile code for ESP8266 to connect it to the Internet.

At first the ESP8266 needs to connect to the router (local network) and set up itself as a station mode (STA). To do so, replace the network SSID and password in source code to match what we have. When it is connected successfully (and of course the router should connect to the Internet), ESP8266 acts as a client and starts communicating with server (will be explained later).

```cpp
#include <ESP8266WiFi.h>
const char* ssid = "ssid";
const char* password = "password";

void setup() {
    Serial.begin(115200);
    Serial.println("DHTxx test!");
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        Serial.println("..");
        delay(500);
    }
}
```

3.1.2.1.2 Collect data by DHT11

Connecting to the Internet successfully does not mean that the data is sent to the server automatically. It needs to go through some steps before the server receives data.

First, the data must be available. It means that the DHT sensor should be able to transfer info from real world into data as below:

```cpp
// declare library
#include "DHT.h"
#define DHTPIN 2
#define DHTTYPE DHT11  // DHT 11
DHT dht(DHTPIN, DHTTYPE);

void setup() {
```
dht.begin();
}
void loop() {
  delay(2000);
  float h = dht.readHumidity();
  float t = dht.readTemperature();
  float f = dht.readTemperature(true);
  if (isnan(h) || isnan(t) || isnan(f)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }
}

### 3.1.2.1.3 ESP8266 sends data to server by GET request

After DHT11 collects data, here GET request comes in handy. In a nutshell, ESP8266 sends a URL with parameters (which are temperature and humidity) so the server can update data accordingly:

```c
void loop()
{
                  + String(t) + ";humi=" + String(h);
  HttpRequest(request);
}
void HttpRequest(String request) {
  if (WiFi.status() == WL_CONNECTED) {
    std::unique_ptr<BearSSL::WiFiClientSecure>client(new BearSSL::WiFiClientSecure);
    client->setInsecure();
    HTTPClient https;
    if (https.begin(*client, request)) {
      int httpsCODE = https.GET();
      Serial.println();
      Serial.println(request);
      Serial.println("https code :" + String(httpsCODE));
      if (httpsCODE > 0) {
        String payload = https.getString();
        Serial.println(payload);
      } else {
        Serial.printf("HTTPS GET failed, ERRORS: %s\n",
                      https.errorToString(httpsCODE).c_str());
      }
    }
  }
}
```
To make things easier, we can go through the snippet code above to understand what ESP8266 does:

1. Create a URL including humidity and temperature value, which updates every 2s
2. Open HTTP, define the destination where ESP8266 will do things
3. Send GET request method
4. Check if the request is failed or not. If it fails, print out the https code, otherwise print out the payload

3.1.2.2 Render chart from browser

3.1.2.2.1 Browser sends GET request to server

To see the chart, the user has to access to this URL


As we can see, in the very beginning when the user accesses the URL above, the browser sends a GET request to showdht11.php.

Figure 13. GET request to showdht11.php
This page is to render the chart (or to “draw” the chart). However, to have the data updated accordingly, there is one thing underneath the hood.

```javascript
load: function() {
    var series_temp = this.series[0],
        series_humi = this.series[1],
        chart = this;
    setInterval(function() {
        var xmlhttp = new XMLHttpRequest();
        var url = "data.json";
        xmlhttp.overrideMimeType("application/json");
        xmlhttp.onreadystatechange = function() {
            if (this.readyState == 4 && this.status == 200) {
                var myArr = JSON.parse(this.responseText);
                json_temp = myArr['temp']
                json_humi = myArr['humi']
                console.log("Temp:", json_temp);
                console.log("Humi:", json_humi);
            }
        }
        xmlhttp.open("GET", url, true);
        xmlhttp.send();
    }
}
```

In the code of `showdht11.php`, there is a part that is called `data.json`. By checking the network tab from client browser, we can see actually it makes a GET request to `data.json` also, not directly but through `showdht11.php`. 
Figure 14. GET request to data.json
3.1.2.2.2 Server responds

As the figure above shows the server responds to the request with a json string. For more details, we can look at Console tab.

---

**Figure 15. Server response content**

---
Figure 16. Server response value

So this is the value of the message after parsing. In the code, they are corresponding to the line 49, 50, 59, 60, 61
```javascript
39  cnart = this;
40  setInterval(function () {
41      var xhr = new XMLHttpRequest();
42      var url = "data.json";
43      xhr.overrideMimeType("application/json");
44      xhr.onreadystatechange = function () {
45          if (xhr.readyState == 4 && xhr.status == 200) {
46              var myArr = JSON.parse(xhr.responseText);
47              json_temp = myArr['temp']
48              json_humi = myArr['humi']
49                console.log("Temp:", json_temp);
50                console.log("Humid:", json_humi);
51          }
52      }
53      xhr.open("GET", url, true);
54      xhr.send();
55  }
56  var x = (new Date()).getTime(),
57  y_temp = Number(json_temp),
58  y_humi = Number(json_humi);
59  console.log(typeof(y_temp));
60  console.log("YT:", y_temp);
61  console.log("YH", y_humi);
62  series_temp.addPoint([x, y_temp], true, true);
63  series_humi.addPoint([x, y_humi], true, true);
64  activeLastPointTooltip(chart);
```

Figure 17. Server respond value corresponding code

### 3.2 The whole circuit setup

![The whole circuit](image)

Figure 18. The whole circuit
The whole circuit is only: a NodeMCU board, power supply and DHT11 sensor. To verify the connection from the beginning, circuit separation is needed to be done. First, we connect only the power supply with the NodeMCU board and check if the WiFi connection is connected properly by printing out SSID and IP address in Serial Monitor Log. Then another sketch is written to check if the NodeMCU can receive data from DHT11 before sending it to server. After all, these works can be merged to have the full sketch as attached in this thesis.
Reference


https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html. [Haettu 16 October 2019].
Appendix One – Setup Arduino IDE environment for ESP8266

Preparing Arduino IDE environment for ESP8266

Making sure that Arduino IDE installed in the computer is the latest version before starting to the installation procedure. ESP8266 is being integrated with the Arduino IDE which allows to program ESP8266 using the Arduino IDE and its programming language.

Installing Arduino IDE

Go to the main page of Arduino [https://www.arduino.cc](https://www.arduino.cc)

(This setup assumes that user is using Windows Operating System)

Software > Downloads > Choose the correct version for Windows. Download and install the IDE

Download the Arduino IDE

![Arduino IDE Download](image)

Figure 19. The main page to download the Arduino IDE

Installing driver

The computer does not know how to communicate with ESP8266 when we first connecting them. This can be seen in Device Manager
Figure 20. CP2102 driver missing

To resolve this issue, a CP2102 USB-to-UART driver needs to be installed. Follow this link [https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers](https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers) to download and install USB Driver CP2102

**Installing the ESP8266 Board within Arduino IDE**

To help the IDE recognize ESP8266, we need to update the board manager with a custom URL as follow:

1. Open the Arduino IDE
2. Go to File > Preferences
Figure 21. Adding ESP8266 URL into Arduino IDE
4. Go to Tools > Board > Board Manager and search for ‘esp8266’. Select the latest version, click install.

Figure 22. Installing ESP8266 board

5. Select the ESP8266 board in programming sketch
Figure 23. Selecting ESP8266 board
Appendix Two – Source code

Showdht11.php

```html
<!DOCTYPE html>
<html>
<head>
<script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
<script src="https://code.highcharts.com/highcharts.js"></script>
<script src="https://code.highcharts.com/modules/exporting.js"></script>
</head>
<body>
<div style="height:410px;min-height:100px;margin:0 auto;" id="main">
<p>hello</p>
<script type="text/javascript"/>
Highcharts.setOptions({
  global: {
    useUTC: false
  }
});

function activeLastPointToolip(chart) {
  var points = chart.series[0].points;
  chart.tooltip.refresh(points[points.length - 1]);
}

var temp_1 = 10.21;
// alert(json_temp);
$('#main').highcharts({
  chart: {
    type: 'spline',
    animation: Highcharts.svg,
    marginRight: 10,
    events: {
      load: function() {
        var series_temp = this.series[0],
            series_humi = this.series[1],
            chart = this;
        setInterval(function() {
          var xmlhttp = new XMLHttpRequest();
          var url = "data.json";
          xmlhttp.overrideMimeType("application/json");
          xmlhttp.onreadystatechange = function() {
            if (this.readyState == 4 && this.status == 200) {
              var myArr = JSON.parse(this.responseText);
              json_temp = myArr['temp']
              json_humi = myArr['humi']
              console.log("Temp:", json_temp);
              console.log("Humi:", json_humi);
            }
          }
          xmlhttp.open("GET", url, true);
          xmlhttp.send();
        }
      }
    }
  }
});
```

```
xmlhttp.open("GET", url, true);
xmlhttp.send();

var x = new Date().getTime(),
y_temp = Number(json_temp),
y_humi = Number(json_humi);
console.log(typeof y_temp);
console.log("YT:", y_temp);
console.log("YH", y_humi);

series_temp.addPoint([x, y_temp], true, true);
series_humi.addPoint([x, y_humi], true, true);
activeLastPointTooltip(chart);
}
},

{title: {
text: 'Temperature & Humidity'
},
credits: {
enabled: false
},
xAxis: {
type: 'datetime',
tickPixelInterval: 150
},
yAxis: {
title: {
text: 'data sensor'
},
plotLines: [{
value: 0,
width: 1,
color: '#808080'
}]
},
tooltip: {
formatter: function() {
return '<b>'+this.series.name+'</b><br/>'+
Highcharts.dateFormat('%Y-%m-%d %H:%M:%S', this.x) + '<br/>'+
Highcharts.numberFormat(this.y, 2);
}
},
legend: {
enabled: false
},
exporting: {
enabled: false
},
series: [{
name: 'temperature',
data: (function() {
// generate an array of random data
var data = [];

})
}];
time = (new Date()).getTime(),
i;
for (i = -19; i <= 0; i += 1) {
data.push({
  x: time + i * 1000,
  y: Math.random()
});
}
return data;
}()
},
{name: 'humidity',
data: (function() {
  // generate an array of random data
  var data = [],
      time = (new Date()).getTime(),
i;
for (i = -19; i <= 0; i += 1) {
data.push({
  x: time + i * 1000,
  y: Math.random()
});
}
return data;
}()}
], function(c) {
  activeLastPointTooltip(c)
});
</script>
</body>
</html>

Update.php

```php
<?php
if(isset($_GET['temp']) && isset($_GET['humi'])){
    $temp = $_GET['temp'];
    $humi = $_GET['humi'];
    $data = array(
        'temp'=>$temp,
        'humi'=>$humi
    );
    $filedata = fopen("data.json", "w");
    if( $filedata == false )
    {
        echo "error make file ";
        exit();
```
```php
$data = json_encode($b);
fwrite($filedata, $data);
fclose($filedata);
echo($data);
}
else{
    echo "no data";
}
?>

DHT11SentToServer.ino

#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <WiFiClientSecureBearSSL.h>
#include "DHT.h"
#define DHTPIN 2
int checksensor;

const char* ssid = "Khanh";
const char* password = "chanvitchanga";

#define DHTTYPE DHT11 // DHT 11
DHT dht(DHTPIN, DHTTYPE);

void setup() {
    Serial.begin(115200);
    Serial.println("DHTxx test!");
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        Serial.println("..");
        delay(500);
    }
    dht.begin();
}

void loop() {
    delay(2000);

    float h = dht.readHumidity();
    float t = dht.readTemperature();
    float f = dht.readTemperature(true);

    if (isnan(h) || isnan(t) || isnan(f)) {
        Serial.println("Failed to read from DHT sensor!");
        return;
    }

    String request = "https://khamecl4thesis.000webhostapp.com/update.php?temp="
                    + String(t) + "&humi=" + String(h);
    HttpRequest(request);
```
Serial.print("Humidity: ");
Serial.print(h);
Serial.print(" \\
\t");
Serial.print("Temperature: ");
Serial.print(t);
Serial.print(" °C ");

void HttpRequest(String request) {
    if (WiFi.status() == WL_CONNECTED) {
        std::unique_ptr<BearSSL::WiFiClientSecure>client(new
            BearSSL::WiFiClientSecure);
        client->setInsecure();
        HTTPClient https;
        if (https.begin(*client, request)) {
            int httpsCODE = https.GET();
            Serial.println();
            Serial.println(request);
            Serial.println("https code :") + String(httpsCODE);
            if (httpsCODE > 0) {
                String payload = https.getString();
                Serial.println(payload);
            } else {
                Serial.printf("HTTPS GET failed, ERRORS: %s\n",
                    https.errorToString(httpsCODE).c_str());
            }
            https.end();
        }
    }
}