SMART WEATHER STATION

A Project Report

submitted by

Name	Registration Number
Aarnav Gupta	20BCE2210
Nimesh Bhavsar	20BCE0542

submitted to

Prof. Dheeba J

Professor SCOPE - VIT Vellore

for subject

Internet of things

Acknowledgement

We would like to thank our faculty, **Professor Dheeba J** for giving us such a wonderful opportunity to work on an exciting and challenging project which not only helped us strengthen our foundation of ***internet of things***, but also enrich our skill-set for the real world. I am immensely grateful for her constantsupport, guidance and encouragement. We thank you for such great opportunity.

Table of Contents

Abstract

Generally, in the worldwide, the weather is very important for people life in numerous situations of rainy, hot, dry, dusty, and windy weather state. We must know these states before going to anywhere to take precautions. These precautions can be implemented by modern electronic and monitoring technologies. The objective is to create a low-cost, robust, and modular Internet of Things (IoT) weather station that can serve as a solution ease of weather monitoring. Our task will be to redesign the weather board to better suit our design needs.

Introduction

Our project introduces some of the following below comparison aspects and drawback current system has:

Older Technology

- Older Technology, could not display the dta in graphical format on the screen neither the data was available on internet
- Data updated every 10 min
- There were significantly more issues and limitations to availability and reliability

Our Technology

- Data is updated every 10 seconds
- This guarantees high speeds transfer that will significantly impact industry
- Can display the data in graphical format on the screen as well as the data will be available on internet

Proposed Work And working principals of sensors

1) DHT11- TEMPERATURE AND HUMIDITY SENSOR DHT11

It is a low-cost digital sensor for sensing temperature and humidity. DHT11 sensor consists of a capacitive humidity sensing element and a thermistor for sensing temperature. The humidity sensing capacitor has two electrodes with a moisture holding substrate as a dielectric between them. Change in the capacitance value occurs with the change in humidity levels. The IC measure, process this changed resistance values and change them into digital form. The temperature range of DHT11 is from 0 to 50 degree Celsius with a 2-degree accuracy. Humidity range of this sensor is from 20 to 80% with 5% accuracy. The sampling rate of this sensor is 1Hz .i.e. it gives one reading for every second. DHT11 is small in size with operating voltage from 3 to 5 volts. The maximum current used while measuring is 2.5mA

2) BH1750FVI LIGHT SENSOR BH1750FVI Is a Digital Light sensor ,

It is an digital Ambient Light Sensor IC for I2C bus interface. generates an output signal indicating the intensity of light by measuring the radiant energy that exists in a very narrow range of frequencies Features: 1) I2C bus Interface (f / s Mode Support) 2) Spectral responsibility is approximately human eye response 3) Illuminance to Digital Converter 4) Wide range and High resolution. (1 - 65535 lx) 5) Low Current by power down function 6) 1.8V Logic input interface 7) Light source dependency is little. 8) It is possible to select 2 type of I2C slave-address. 9) It is possible to detect min. 0.11 lx, max. 100000 lx by using this function

3) BMP180 BAROMETER The BMP180 sensor

It is mainly used to measure atmospheric pressure or biometric pressure. The working principle of the air pressure sensor is very simple, it works based on the weight of air. Because the air around us has a certain weight, and this weight has a specific pressure. Can measure temperature and altitude. Pressure range: 300 to 1100hPa High relative accuracy of ±0.12hPa Can work on low voltages 3.4Mhz I2C interface Low power consumption (3uA) Pressure conversion time: 5msec Potable size

4) OLED Display Module

is a precise small, White OLED module which can be interfaced with any microcontroller using SPI protocol. It is having a resolution of 128x64. The package includes display board, display, 6 pin male header. OLED (Organic Light-Emitting Diode) is a self light-emitting technology composed of a thin, multi-layered organic film placed between an anode and cathode. In contrast to LCD technology, OLED does not require a backlight. OLED possesses high application potential for virtually all types of displays and is regarded as the ultimate technology for the next generation of flat-panel displays

Literature Survey:

1. Weather Forecasting using Arduino Based Cube-Sat M. Rahaman Laskar, R. Bhattacharjee, M. SauGiri and P. Bhattacharya ICIMP - 2016

The authors have designed a small cube satellite system which has temperature & humidity sensor (DHT11), pressure sensor (BMP085) and Accelerometer (ADXL-335) connected to Arduino Uno. The system is then connected to transmitter and receiver module and RF module is used for wireless transmission data. A gas balloon is used to carry the cube satellite system. The system provides weather forecasting without using any internet network.

Connected to transmitter and receiver module, RF module is used for wireless transmission data. A gas balloon used to carry the cube satellite. Weather forecasting without using any internet network.

Advantages:

- The system is cost effective, portable, power efficient and reliable.
- It is easy to construct because of it's simple design.
- Data transmission has low cost as system doesn't use internet network.

Disadvantages:

- Without powerful transceivers section long distance communication is not possible by the device.
- There may be problem in recording data with the help of gas balloon at higher altitudes.
- The components of the system may get damaged because of rain or long use.

2. Efficient IoT based Weather Station Abu Saleh Bin Shahadat, S. Islam Ayon and M. R. Khatun EEE International WIE Conference on Electrical and Computer Engineering (WIECON-ECE) -2020

The authors have planned to develop a weather station consisting of a system including an Esp8266 Wi-Fi microchip as both data processor and TCP/IP stack,

DHT11 temperature and humidity sensor, BMP180 pressure sensor and SEN-00194 rain sensor and connecting cable. ESP8266 module processes sensor data and convert it into temperature, humidity, atmospheric pressure, rainfall data in a standard format.

The data is stored on a remote cloud server using Wi-Fi connection. Blynk IoT app gets the inputs using virtual terminals and show it on preferred settings. For the visualization of weather information web application interface is used.

It is small in size, inexpensive, reliable and can be used in different applications. The proposed system is fully automated, no human attention required. We can get prior alert of weather situation. Accuracy is high.

The authors claims that the proposed system can be used in following applications:

- In the field of agriculture, the weather forecasting system plays an important role.
- In the volcano and rain forests, it is also very efficient.
- At such places, where staying is difficult for a longer time, it is also helpful.
- The results produced are good in terms of accuracy and reliability.

3. IOT Based Rainfall Monitoring System Using WSN Enabled Architecture Mr. Onkar Amale, Dr. Rupali Patil IEEE – 2019

Old methods of monitoring the environmental parameters which as physically receiving information rainfall data from stations can be brutal and inhibiting monitoring required for careful imposition. WSN which offer the high quality rainfall monitoring at very cheap rate in terms of labor invested and capital. This paper includes the WSN-enabled architecture for rainfall monitoring system to transmit and collect real time data using GPRS (General Pocket Radio Service) via a cellular network. The data is sent from remote stations to the web server known as Weather Underground. Contribution work is an approach is bandwidth compressed waveform signal for increasing the number of connected devices Performance analysis using SVM machine learning classifier for prediction of rainfall.

The system uses a tipping bucket RG sensor, micro controller, heat sensor, GPRS module, air water sensors and rainfall information to a web module, also

a energy panels are provided for power backup in critical area for WSN. Also, different sensors like soil moisture, rainfall, temperature, humidity soil temperature etc. are used.

Advantages:

- Cost effective due to use of simple and efficient tipping bucket rain gauge system.

Disadvantages:

- Cost of routing path required is relatively high.

4. Development of a New Ship Adaptive Weather Routing Model Based on Seakeeping Analysis and Optimization Silvia Pennino, Salvatore Gaglione, Anna Innac, Vincenzo Piscopo and Antonio Scamardella Journal of Marine Science & Engineering - 2020

This paper provides a new adaptive weather routing model, based on the Dijkstra shortest path algorithm, aiming to select the optimal route that maximizes the ship performances in a seaway. The model is based on a set of ship motion-limiting criteria and on the weather forecast maps, providing the sea state conditions the ship is expected to encounter along the scheduled route. The new adaptive weather routing model is applied to optimize the scheduled route in the Northern Atlantic Ocean of the S175 containership, assumed as a reference vessel, based on the weather forecast data provided by the Global WAve Model (GWAM).

The model is based on the weather forecast maps, providing the sea weather conditions the ship is expected to encounter along the scheduled route.

Advantages:

- An increase of the ship performances in a seaway with a positive impact on the safety of navigation, without significantly affecting the voyage duration or the route length.
- The additional fuel savings, also result in low gas emissions in the atmosphere.

Disadvantages:

- The start and the endpoint are slightly far from the departure and arrival ports, mainly due to the marine traffic congestion near the coastline, that makes the application of adaptive routing models challenging, as additional restraints, mainly related to neighboring vessels, arise.

Implementation

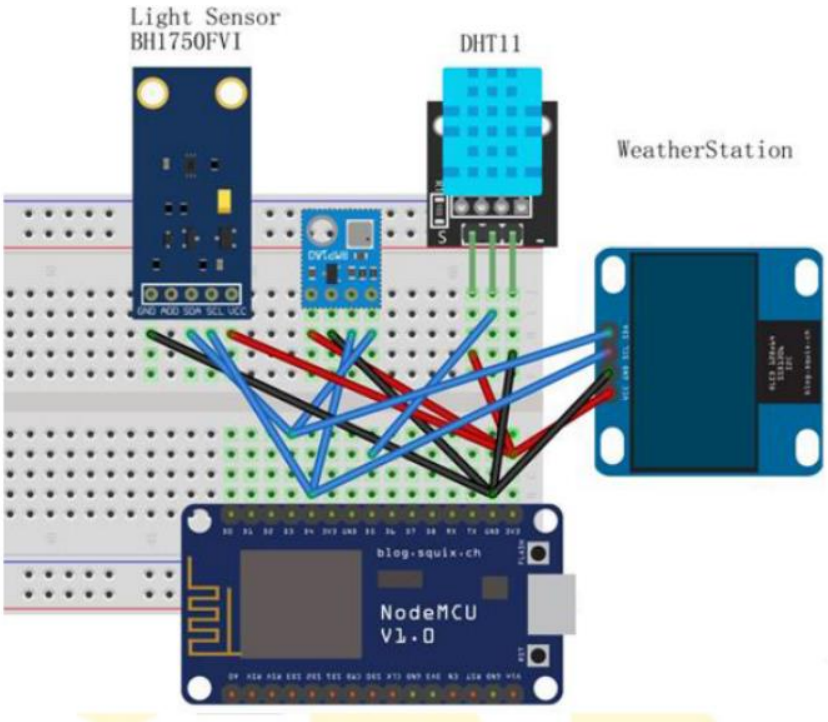
For the project we have used above and below mentioned sensors and following libraries

1. 1 x ESP8266 -12E Arduino
2. 1 x DHT11
3. 1 x BMP180
4. 1 x BH1750FVI
5. 1 x OLED Display
6. 1 x USB Cable
7. 2 x Mini Breadboard
8. 30 x Jumper wire DC Power

Major Libraries used:

1. NodeMCU 1.0 (ESP-12E Module)
2. Adafruit BMP085 ESP WiFi
3. OpenWeatherMap
4. ThinkSpeak API

Bread Board connection of Project:



Code and working:

Implementation (Code):

Arduino IDE Code:

```
#include <Adafruit_BMP085.h>

#include <ESPWiFi.h>
#include <ESPHTTPClient.h>
#include <JsonListener.h>

// time
#include <time.h>                       // time() ctime()
#include <sys/time.h>                   // struct timeval
#include <coredecls.h>                  // settimeofday_cb()

#include "SSD1306Wire.h"
#include "OLEDDisplayUi.h"
#include "Wire.h"
#include "OpenWeatherMapCurrent.h"
#include "OpenWeatherMapForecast.h"
#include "WeatherStationFonts.h"
#include "WeatherStationImages.h"
#include <ESP8266WiFi.h>

#include <Adafruit_BMP085.h>

/**************************
 * WIFI Settings
 **************************/
const char* WIFI_SSID = "N";
const char* WIFI_PWD = "harharmahadev";

/**************************
 * Begin DHT11 Settings
 **************************/
WiFiClient client;
const char *host = "api.thingspeak.com";                  //IP address of the
thingspeak server
const char *api_key ="LSIQ3M7BAQ7C372L";                  //Your own thingspeak
api_key
const int httpPort = 80;
#define pin 14       // ESP8266-12E  D5 read temperature and Humidity data
int temp = 0; //temperature
int humi = 0; //humidity
void readTemperatureHumidity();
void uploadTemperatureHumidity();
long readTime = 0;
long uploadTime = 0;

/**************************
 * Begin Atmosphere and Light Sensor Settings
 **************************/
void readLight();
void readAtmosphere();
Adafruit_BMP085 bmp;
const int Light_ADDR = 0b0100011;   // address:0x23
const int Atom_ADDR = 0b1110111;  // address:0x77
int tempLight = 0;
int tempAtom = 0;

/**************************
 * Begin Settings
 **************************/
#define TZ              2       // (utc+) TZ in hours
#define DST_MN 60      // use 60mn for summer time in some countries
```

```
// Setup
const int UPDATE_INTERVAL_SECS = 20 * 60; // Update every 20 minutes
// Display Settings
const int I2C_DISPLAY_ADDRESS = 0x3c;
#if defined(ESP8266)

const int SDA_PIN = D3;
const int SDC_PIN = D4;
#else

const int SDA_PIN = GPIO0;
const int SDC_PIN = GPIO2
#endif


// OpenWeatherMap Settings
const boolean IS_METRIC = true;
// Add our own thingpulse ID
String OPEN_WEATHER_MAP_APP_ID = "18b70ec058b900c924fd3aca0bbd7d36";
String OPEN_WEATHER_MAP_LOCATION = "Vellore,IN";


String OPEN_WEATHER_MAP_LANGUAGE = "en";
const uint8_t MAX_FORECASTS = 4;

// Adjust according to your language
const String WDAY_NAMES[] = {"SUN", "MON", "TUE", "WED", "THU", "FRI", "SAT"};
const String MONTH_NAMES[] = {"JAN", "FEB", "MAR", "APR", "MAY", "JUN", "JUL", "AUG",
"SEP", "OCT", "NOV", "DEC"};

/**************************
 * End Settings
 **************************/
 // Initialize the oled display for address 0x3c
 SSD1306Wire     display(I2C_DISPLAY_ADDRESS, SDA_PIN, SDC_PIN);
 OLEDDisplayUi   ui( &display );

OpenWeatherMapCurrentData currentWeather;
OpenWeatherMapCurrent currentWeatherClient;

OpenWeatherMapForecastData forecasts[MAX_FORECASTS];
OpenWeatherMapForecast forecastClient;

#define TZ_MN           ((TZ)*60)
#define TZ_SEC          ((TZ)*3600)
#define DST_SEC         ((DST_MN)*60)
time_t now;

// flag changed in the ticker function every 10 minutes
bool readyForWeatherUpdate = false;
String lastUpdate = "--";
long timeSinceLastWUpdate = 0;
//declaring prototypes
void drawProgress(OLEDDisplay *display, int percentage, String label);
void updateData(OLEDDisplay *display);
void drawDateTime(OLEDDisplay *display, OLEDDisplayUiState* state, int16_t x, int16_t
y);
void drawCurrentWeather(OLEDDisplay *display, OLEDDisplayUiState* state, int16_t x,
int16_t y);
void drawForecast(OLEDDisplay *display, OLEDDisplayUiState* state, int16_t x, int16_t
y);
void drawForecastDetails(OLEDDisplay *display, int x, int y, int dayIndex);
void drawHeaderOverlay(OLEDDisplay *display, OLEDDisplayUiState* state);
void setReadyForWeatherUpdate();
```

```
FrameCallback frames[] = { drawDateTime, drawCurrentWeather, drawForecast };
int numberOfFrames = 3;

OverlayCallback overlays[] = { drawHeaderOverlay };
int numberOfOverlays = 1;

void setup() {
  Serial.begin(115200);

  Wire.begin(0,2);

  Wire.beginTransmission(Atom_ADDR);
  //initialize Atmosphere sensor
  if (!bmp.begin()) {
    Serial.println("Could not find BMP180 or BMP085 sensor at 0x77");
  }else{
    Serial.println("Find BMP180 or BMP085 sensor at 0x77");
  }
  Wire.endTransmission();

  //initialize light sensor
  Wire.beginTransmission(Light_ADDR);
  Wire.write(0b00000001);
  Wire.endTransmission();

  // initialize dispaly
  display.init();
  display.clear();
  display.display();

  //display.flipScreenVertically();
  display.setFont(ArialMT_Plain_10);
  display.setTextAlignment(TEXT_ALIGN_CENTER);
  display.setContrast(255);

  WiFi.begin(WIFI_SSID, WIFI_PWD);

  int counter = 0;
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
    display.clear();
    display.drawString(64, 10, "Connecting to WiFi");
    display.drawXbm(46, 30, 8, 8, counter % 3 == 0 ? activeSymbole :
inactiveSymbole);
    display.drawXbm(60, 30, 8, 8, counter % 3 == 1 ? activeSymbole :
inactiveSymbole);
    display.drawXbm(74, 30, 8, 8, counter % 3 == 2 ? activeSymbole :
inactiveSymbole);
    display.display();

    counter++;
  }
  // Get time from network time service
  configTime(TZ_SEC, DST_SEC, "pool.ntp.org");
  ui.setTargetFPS(30);
  ui.setActiveSymbol(activeSymbole);
  ui.setInactiveSymbol(inactiveSymbole);

  ui.setIndicatorPosition(BOTTOM);
  // Defines where the first frame is located in the bar.
  ui.setIndicatorDirection(LEFT_RIGHT);

  ui.setFrameAnimation(SLIDE_LEFT);
  ui.setFrames(frames, numberOfFrames);
```

```
  ui.setOverlays(overlays, numberOfOverlays);
  ui.init();
  Serial.println("");
  updateData(&display);
  while (!client.connect(host, httpPort)) {
    Serial.println("Connection Failed");
  }

}

void loop() {
  //Read Temperature Humidity every 5 seconds
  if(millis() - readTime > 5000){
    readTemperatureHumidity();
    readLight();
    readAtmosphere();
    readTime = millis();
  }
  //Upload Temperature Humidity every 60 seconds
  if(millis() - uploadTime > 60000){
    uploadTemperatureHumidity();
    uploadTime = millis();
  }

  if (millis() - timeSinceLastWUpdate > (1000L*UPDATE_INTERVAL_SECS)) {
    setReadyForWeatherUpdate();
    timeSinceLastWUpdate = millis();
  }

  if (readyForWeatherUpdate && ui.getUiState()->frameState == FIXED) {
    updateData(&display);
  }

  int remainingTimeBudget = ui.update();

  if (remainingTimeBudget > 0) {

    delay(remainingTimeBudget);
  }
}

void drawProgress(OLEDDisplay *display, int percentage, String label) {
  display->clear();
  display->setTextAlignment(TEXT_ALIGN_CENTER);
  display->setFont(ArialMT_Plain_10);
  display->drawString(64, 10, label);
  display->drawProgressBar(2, 28, 124, 10, percentage);
  display->display();
}

void updateData(OLEDDisplay *display) {
  drawProgress(display, 10, "Updating time...");
  drawProgress(display, 30, "Updating weather...");
  currentWeatherClient.setMetric(IS_METRIC);
  currentWeatherClient.setLanguage(OPEN_WEATHER_MAP_LANGUAGE);
  currentWeatherClient.updateCurrent(&currentWeather, OPEN_WEATHER_MAP_APP_ID,
OPEN_WEATHER_MAP_LOCATION);
  drawProgress(display, 50, "Updating forecasts...");
  forecastClient.setMetric(IS_METRIC);
  forecastClient.setLanguage(OPEN_WEATHER_MAP_LANGUAGE);
  uint8_t allowedHours[] = {12};
  forecastClient.setAllowedHours(allowedHours, sizeof(allowedHours));
  forecastClient.updateForecasts(forecasts, OPEN_WEATHER_MAP_APP_ID,
OPEN_WEATHER_MAP_LOCATION, MAX_FORECASTS);
  readyForWeatherUpdate = false;
  drawProgress(display, 100, "Done...");
```

```
  delay(1000);
}


void drawDateTime(OLEDDisplay *display, OLEDDisplayUiState* state, int16_t x, int16_t
y) {
  now = time(nullptr);
  struct tm* timeInfo;
  timeInfo = localtime(&now);
  char buff[16];


  display->setTextAlignment(TEXT_ALIGN_CENTER);
  display->setFont(ArialMT_Plain_10);
  String date = WDAY_NAMES[timeInfo->tm_wday];

  sprintf_P(buff, PSTR("%s, %02d/%02d/%04d"), WDAY_NAMES[timeInfo->tm_wday].c_str(),
timeInfo->tm_mday, timeInfo->tm_mon+1, timeInfo->tm_year + 1900);
  display->drawString(64 + x, 5 + y, String(buff));
  display->setFont(ArialMT_Plain_24);

  sprintf_P(buff, PSTR("%02d:%02d:%02d"), timeInfo->tm_hour, timeInfo->tm_min,
timeInfo->tm_sec);
  display->drawString(64 + x, 15 + y, String(buff));
  display->setTextAlignment(TEXT_ALIGN_LEFT);
}

void drawCurrentWeather(OLEDDisplay *display, OLEDDisplayUiState* state, int16_t x,
int16_t y) {
  display->setFont(ArialMT_Plain_10);
  display->setTextAlignment(TEXT_ALIGN_CENTER);
  display->drawString(64 + x, 38 + y, currentWeather.description);

  display->setFont(ArialMT_Plain_24);
  display->setTextAlignment(TEXT_ALIGN_LEFT);
  String temp = String(currentWeather.temp, 1) + (IS_METRIC ? "°C" : "°F");
  display->drawString(60 + x, 5 + y, temp);

  display->setFont(Meteocons_Plain_36);
  display->setTextAlignment(TEXT_ALIGN_CENTER);
  display->drawString(32 + x, 0 + y, currentWeather.iconMeteoCon);
}


void drawForecast(OLEDDisplay *display, OLEDDisplayUiState* state, int16_t x, int16_t
y) {
  drawForecastDetails(display, x, y, 0);
  drawForecastDetails(display, x + 44, y, 1);
  drawForecastDetails(display, x + 88, y, 2);
}

void drawForecastDetails(OLEDDisplay *display, int x, int y, int dayIndex) {
  time_t observationTimestamp = forecasts[dayIndex].observationTime;
  struct tm* timeInfo;
  timeInfo = localtime(&observationTimestamp);
  display->setTextAlignment(TEXT_ALIGN_CENTER);
  display->setFont(ArialMT_Plain_10);
  display->drawString(x + 20, y, WDAY_NAMES[timeInfo->tm_wday]);

  display->setFont(Meteocons_Plain_21);
  display->drawString(x + 20, y + 12, forecasts[dayIndex].iconMeteoCon);
  String temp = String(forecasts[dayIndex].temp, 0) + (IS_METRIC ? "°C" : "°F");
  display->setFont(ArialMT_Plain_10);
  display->drawString(x + 20, y + 34, temp);
  display->setTextAlignment(TEXT_ALIGN_LEFT);
```

```
}

void drawHeaderOverlay(OLEDDisplay *display, OLEDDisplayUiState* state) {
  now = time(nullptr);
  struct tm* timeInfo;
  timeInfo = localtime(&now);
  char buff[14];
  sprintf_P(buff, PSTR("%02d:%02d"), timeInfo->tm_hour, timeInfo->tm_min);

  display->setColor(WHITE);
  display->setFont(ArialMT_Plain_10);
  display->setTextAlignment(TEXT_ALIGN_LEFT);
  display->drawString(0, 54, String(buff));
  display->setTextAlignment(TEXT_ALIGN_RIGHT);
  String temp = String(currentWeather.temp, 1) + (IS_METRIC ? "°C" : "°F");
  display->drawString(128, 54, temp);
  display->drawHorizontalLine(0, 52, 128);
}

void setReadyForWeatherUpdate() {
  Serial.println("Setting readyForUpdate to true");
  readyForWeatherUpdate = true;
}

//read temperature humidity data
void readTemperatureHumidity(){
  int j;
  unsigned int loopCnt;
  int chr[40] = {0};
  unsigned long time1;
bgn:
  delay(2000);
  //Set interface mode 2 to: output
  //Output low level 20ms (>18ms)
  //Output high level 40μs
  pinMode(pin, OUTPUT);
  digitalWrite(pin, LOW);
  delay(20);
  digitalWrite(pin, HIGH);
  delayMicroseconds(40);
  digitalWrite(pin, LOW);
  //Set interface mode 2: input
  pinMode(pin, INPUT);
  //High level response signal
  loopCnt = 10000;
  while (digitalRead(pin) != HIGH){
    if (loopCnt-- == 0){
      //If don't return to high level for a long time, output a prompt and start over
      Serial.println("HIGH");
      goto bgn;
    }
  }
  //Low level response signal
  loopCnt = 30000;
  while (digitalRead(pin) != LOW){
    if (loopCnt-- == 0){
      //If don't return low for a long time, output a prompt and start over
      Serial.println("LOW");
      goto bgn;
    }
  }
  //Start reading the value of bit1-40
  for (int i = 0; i < 40; i++){
    while (digitalRead(pin) == LOW){}
    //When the high level occurs, write down the time "time"
```

```
    time1 = micros();
    while (digitalRead(pin) == HIGH){}
    //When there is a low level, write down the time and subtract the time just saved
    //If the value obtained is greater than 50μs, it is '1', otherwise it is '0'
    //And save it in an array
    if (micros() - time1  > 50){
      chr[i] = 1;
    } else {
      chr[i] = 0;
    }
  }

  //Humidity, 8-bit bit, converted to a value
  humi = chr[0] * 128 + chr[1] * 64 + chr[2] * 32 + chr[3] * 16 + chr[4] * 8 + chr[5]
* 4 + chr[6] * 2 + chr[7];
  //Temperature, 8-bit bit, converted to a value
  temp = chr[16] * 128 + chr[17] * 64 + chr[18] * 32 + chr[19] * 16 + chr[20] * 8 +
chr[21] * 4 + chr[22] * 2 + chr[23];

    Serial.print("temp:");
    Serial.print(temp);
    Serial.print("    humi:");
    Serial.println(humi);

}
```

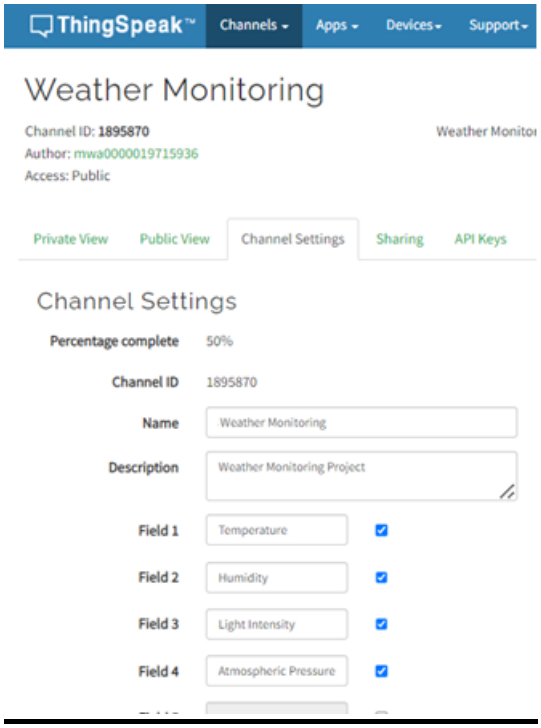
```
void readLight(){
  // reset
  Wire.beginTransmission(Light_ADDR);
  Wire.write(0b00000111);
  Wire.endTransmission();

  Wire.beginTransmission(Light_ADDR);
  Wire.write(0b00100000);
  Wire.endTransmission();
  // typical read delay 120ms
  delay(120);
  Wire.requestFrom(Light_ADDR, 2); // 2byte every time
  for (tempLight = 0; Wire.available() >= 1; ) {
    char c = Wire.read();
    tempLight = (tempLight << 8) + (c & 0xFF);
  }
  tempLight = tempLight / 1.2;
  Serial.print("light: ");
  Serial.println(tempLight);
}
```

```
void readAtmosphere(){
  tempAtom = bmp.readPressure();
  Serial.print("Pressure = ");
  Serial.print(tempAtom);
  Serial.println(" Pascal");
}

//upload temperature humidity data to thinkspak.com
void uploadTemperatureHumidity(){
   if(!client.connect(host, httpPort)){
    Serial.println("connection failed");
    return;
  }
  // Three values(field1 field2 field3 field4) have been set in thingspeak.com
```

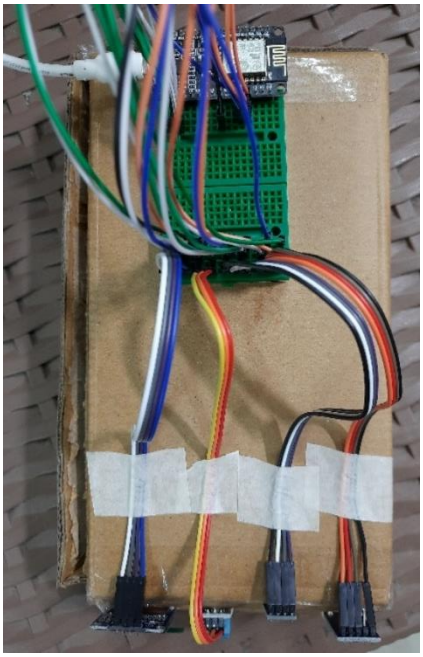
```
  client.print(String("GET ") +
"/update?api_key="+api_key+"&field1="+temp+"&field2="+humi +
"&field3="+tempLight+"&field4="+tempAtom+" HTTP/1.1\r\n" +"Host: " + host + "\r\n" +
"Connection: close\r\n\r\n");
  while(client.available()){
    String line = client.readStringUntil('\r');
    Serial.print(line);
  }
}
```

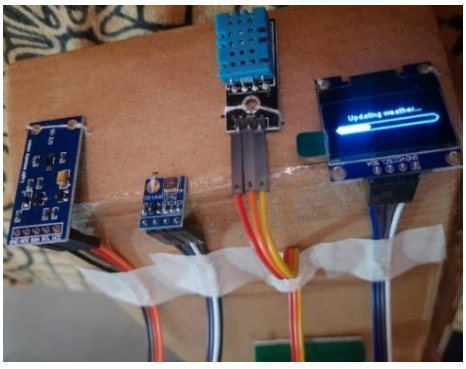
Results & Discussion:

Thingspeak server Configurations:

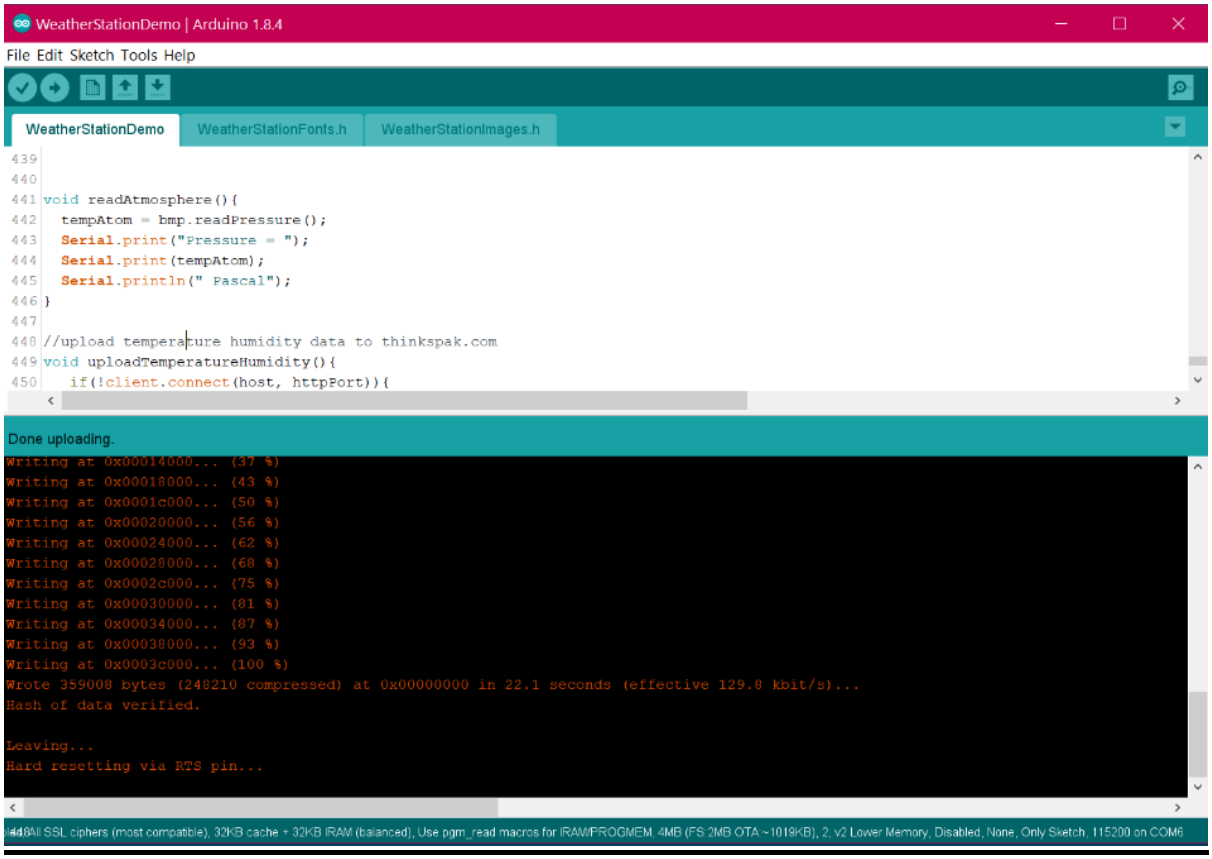


Circuit connections images:





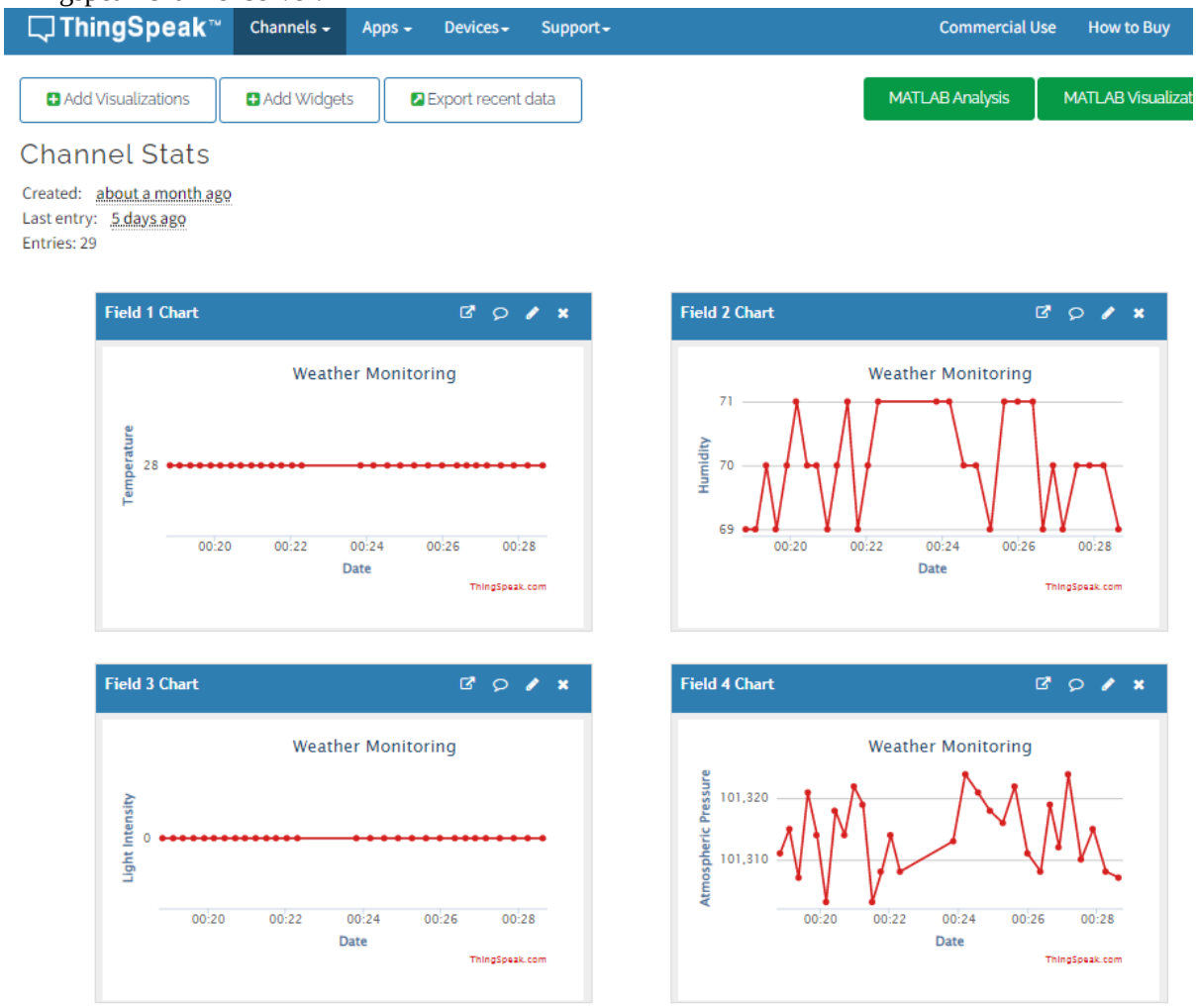
Coding in Arduino IDE:



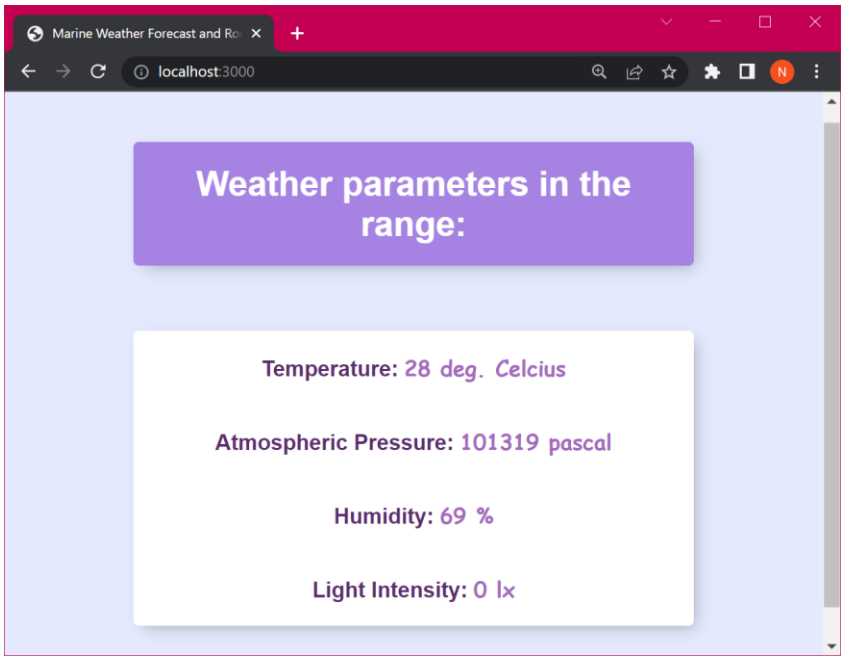
Output

```
$ nodemon server.js
[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
Server running on port 3000 !
Connecting...

Connected, fetching data !


Good weather conditions ahead

. . . . . .


Server running on port 3000 !
statusCode: 200  statusMessage: OK
Host: api.thingspeak.com  Protocol: http

Fetching data successfully !


Good weather conditions ahead

. . . . . .


Server running on port 3000 !
statusCode: 200  statusMessage: OK
Host: api.thingspeak.com  Protocol: http

Fetching data successfully !


Good weather conditions ahead
```

Thingspeak channel server:



Fetching weather data to local web-page:(data obtained from Thinkspeak server)



Conclusion

So, we have created a proper Weather Monitoring System using NodeMCU with WiFi Module ESP8266 -12E, Temperature and humidity sensor DHT11, Atmospheric pressure sensor BMP180, Digital Light Sensor BH1750FVI, Mini Breadboards, OLED Display, Jumper wires, USB Cable. It was able to correctly detect the values of the weather conditions present around, like – Temperature (deg. Celcius), Atmospheric Pressure (pascals), Light Intensity (Lux (lx)) and Humidity (%).

The detected parameter values were updated to a channel server created on thingspeak (a matlab tool) to visualize the graphs of the collected data. A local Web Page was created for the simulation view of the results, which fetched the data back from thingspeak server API.

References

- S. Bin Shahadat, S. Islam Ayon and M. R. Khatun, "Efficient IoT based Weather Station," 2020 IEEE International Women in Engineering (WIE) Conference on Electrical and Computer Engineering (WIECON-ECE), 2020, pp. 227-230, doi: 10.1109/WIECON-ECE52138.2020.9398041.

- O. Amale and R. Patil, "“IOT Based Rainfall Monitoring System Using WSN Enabled Architecture”," 2019 3rd International Conference on Computing Methodologies and Communication (ICCMC), 2019, pp. 789-791, doi: 10.1109/ICCMC.2019.8819721.

- M. Rahaman Laskar, R. Bhattacharjee, M. SauGiri and P. Bhattacharya, “Weather Forecasting using Arduino Based Cube-Sat”, Twelfth International Multi-Conference on Information Processing-2016 (IMCIP-2016), DOI: 10.1016/j.procs.2016.06.078

- Pennino, Silvia & Gaglione, Salvatore & Innac, Anna & Piscopo, Vincenzo & Scamardella, Antonio. (2020). Development of a New Ship Adaptive Weather Routing Model Based on Seakeeping Analysis and Optimization. Journal of Marine Science and Engineering. 8. 270. 10.3390/jmse8040270.

- https://in.mathworks.com/help/thingspeak/thingspeakwrite.html
- https://in.mathworks.com/help/thingspeak/examples.html?category=write-data&s_tid=CRUX_topnav
- https://create.arduino.cc/projecthub/Arduino_Genuino/getting-started-with-arduino-web-editor-on-various-platforms-4b3e4a