Implementation

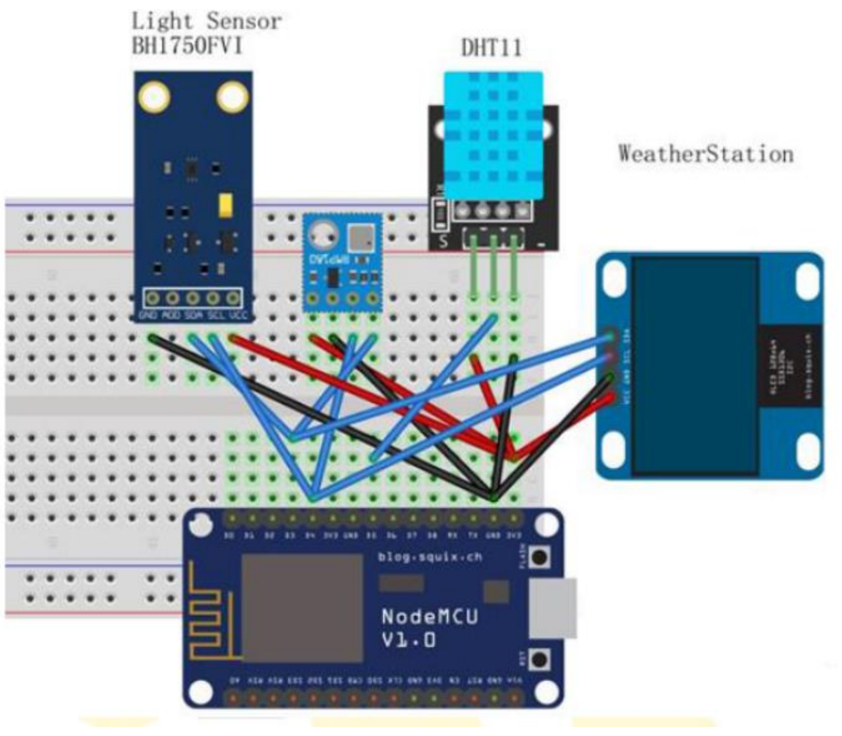
For the project we have used above and below mentioned sensors and following libraries

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
```

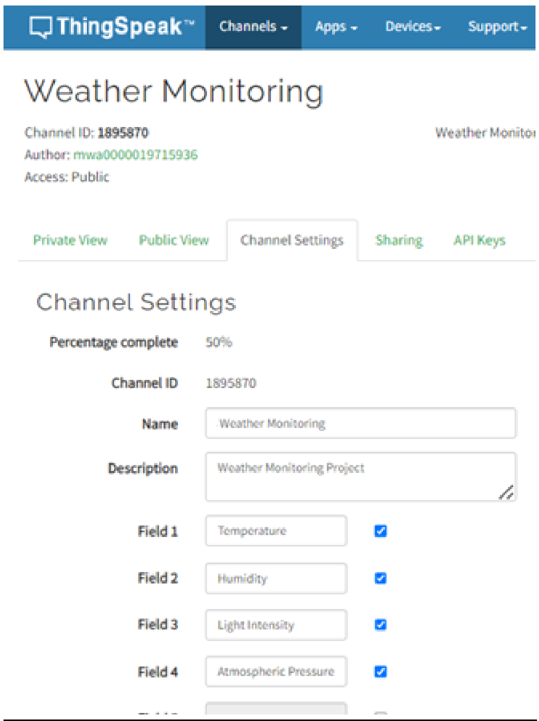
```
// Setup
const int UPDATE_INTERVAL_SECS = 20 * 60; // Update every 20 minutes
// Display Settings
const int I2C_DISPLAY_ADDRESS = 0x3c;
#if defined(ESP8266)

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:



Weather Monitoring

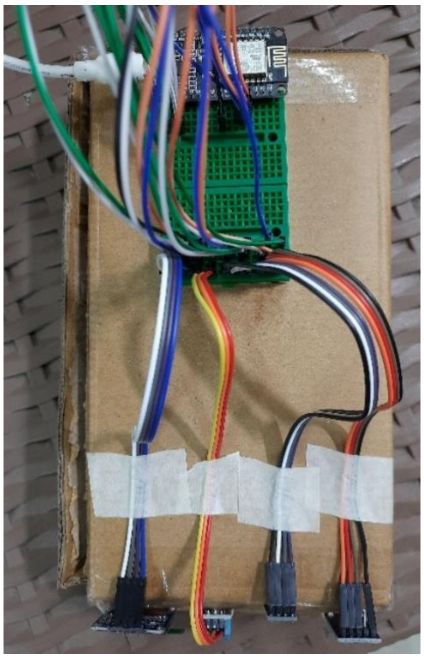
Channel ID: 1895870
Author: mwa0000019715936
Access: Public

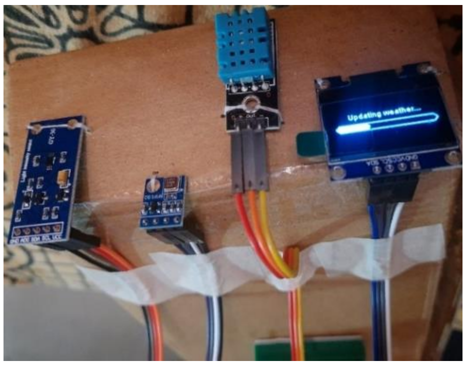
Weather Monitor

Channel Settings

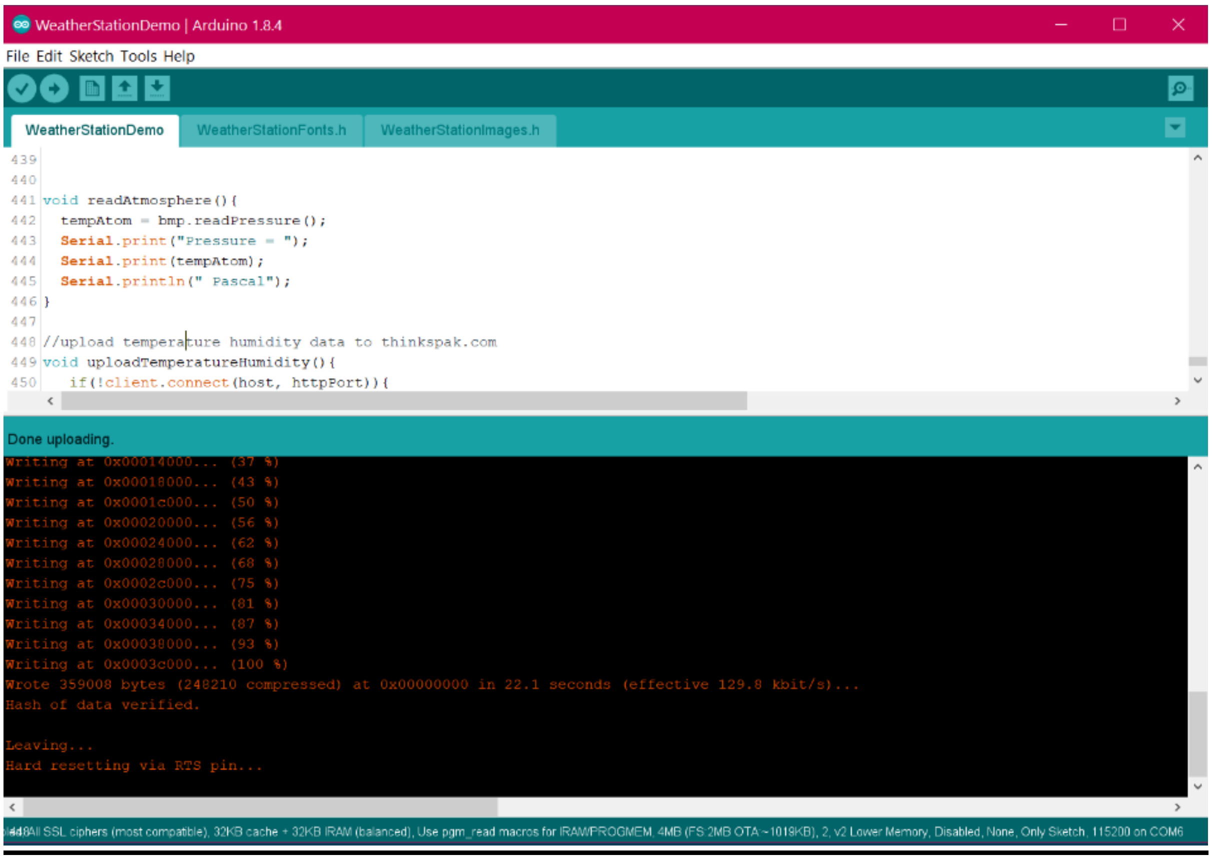
Percentage complete	50%	
Channel ID	1895870	
Name	Weather Monitoring	
Description	Weather Monitoring Project	
Field 1	Temperature	✓
Field 2	Humidity	✓
Field 3	Light Intensity	✓
Field 4	Atmospheric Pressure	✓

Circuit connections images:





Coding in Arduino IDE:



WeatherStationDemo | Arduino 1.8.4

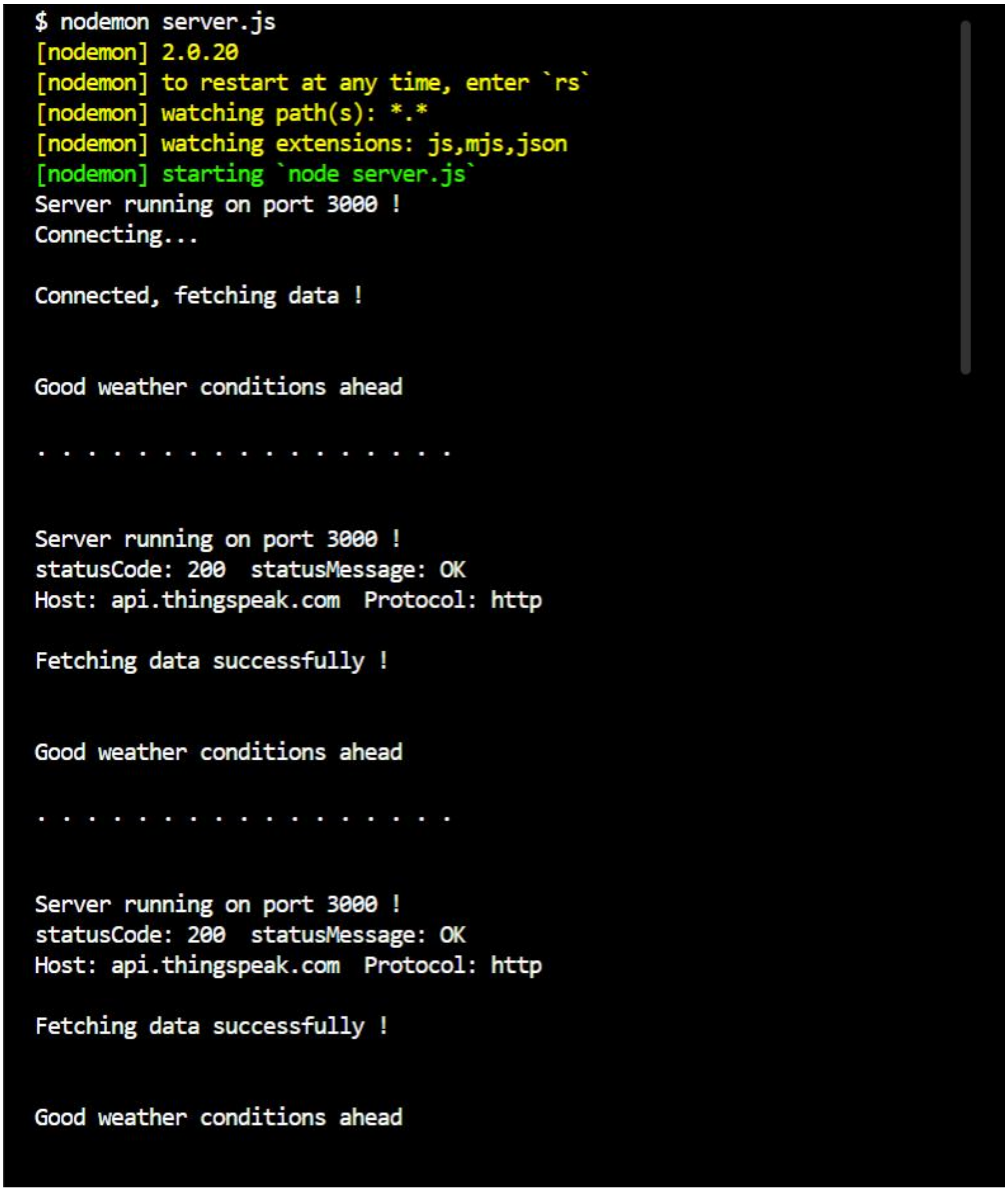
```

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)){
```

Done uploading.

```
Writing at 0x00014000... (37 %)
Writing at 0x00018000... (43 %)
Writing at 0x0001c000... (50 %)
Writing at 0x00020000... (56 %)
Writing at 0x00024000... (62 %)
Writing at 0x00028000... (68 %)
Writing at 0x0002c000... (75 %)
Writing at 0x00030000... (81 %)
Writing at 0x00034000... (87 %)
Writing at 0x00038000... (93 %)
Writing at 0x0003c000... (100 %)
Wrote 359008 bytes (248210 compressed) at 0x00000000 in 22.1 seconds (effective 129.8 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

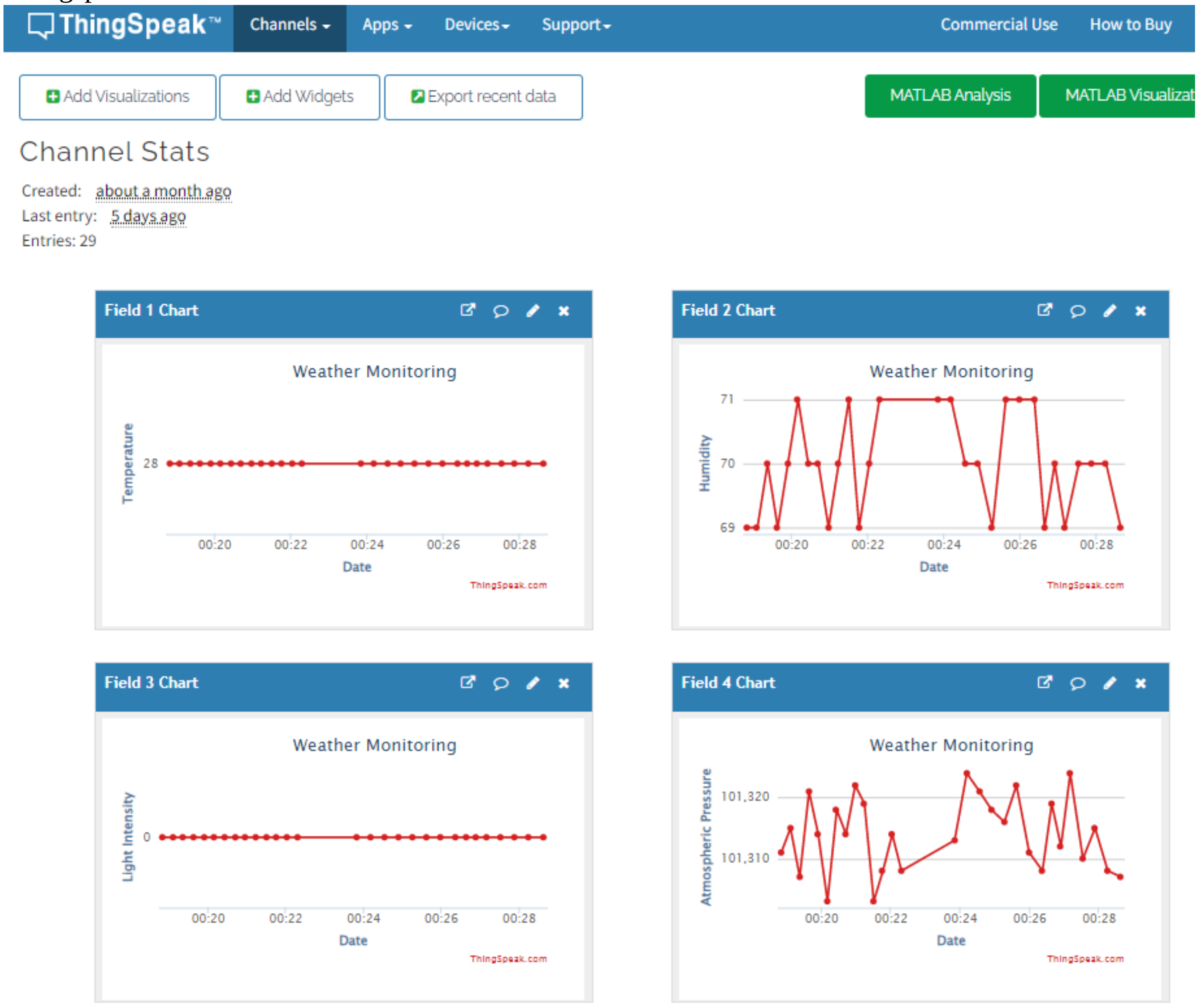
448All SSL ciphers (most compatible), 32KB cache + 32KB IRAM (balanced), Use pgm_read macros for IRAM/PROGMEM, 4MB (FS:2MB OTA:~1019KB), 2, v2 Lower Memory, Disabled, None, Only Sketch, 115200 on COM6

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:



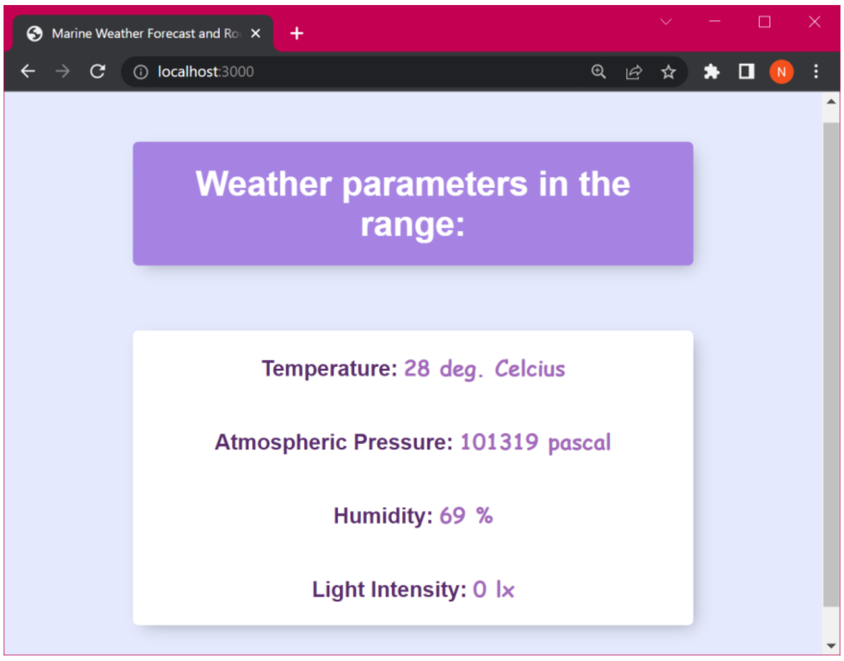
Channel Stats

Created: about a month ago

Last entry: 5 days ago

Entries: 29

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



Weather parameters in the range:

Temperature: 28 deg. Celcius

Atmospheric Pressure: 101319 pascal

Humidity: 69 %

Light Intensity: 0 lx

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