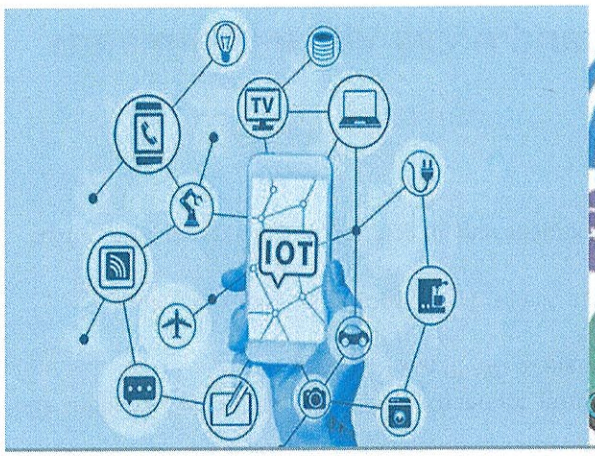


IoT
Internet of Things
STEM
SCIENCE TECHNOLOGY
ENGINEERING MATHEMATICS

OHIO UNIVERSITY
Voinovich School of Leadership and Public Affairs

AMERICAN ELECTRIC POWER FOUNDATION

PORTSFUTURE
IMAGINING THE OPPORTUNITIES, GATHERING YOUR IDEAS
THE FACILITY AT PIKETON, OHIO

Internet of Things Student STEM Project
Jackson High School

Lesson 4 – Smart Weather Lamp and Alexa Voice Commands

Time to complete Lesson

60-minute class period

Learning objectives

- Students learn how to control smart devices using Alexa and an Echo Dot while also sending sensor data to the cloud via AWS-IOT
- Students learn about different sensors and how to use IoT to produce meaningful data and observations
- Students learn about the variety of sensors available to use in their design
- Students learn how to quickly create Alexa to control connected devices for the home, like lights and thermostats, from the cloud.
- Students learn how to move on and connect your Arduino to any web-based resource

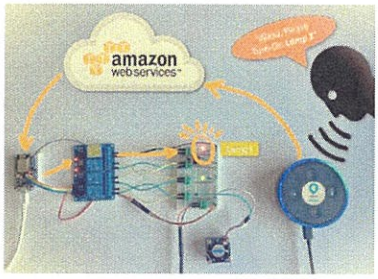
21st century technical skills gained through this activity

- Electronic Circuit Design
- Computer Programming
- Connect Arduino to web-based or Cloud services

Credits!

Code and tutorial based on the great open source lib & example code at:
http://www.instructables.com/id/ESP8266-Weather-Widget/
https://bitbucket.org/xoseperez/fauxmoesp

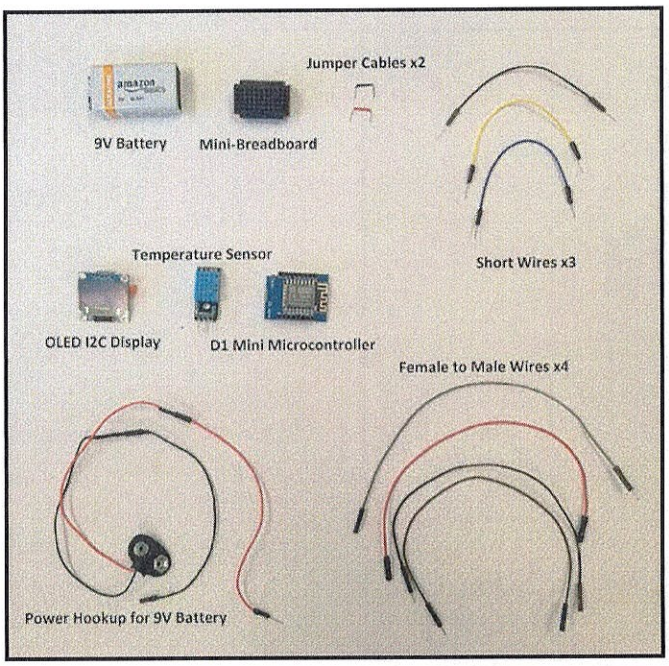
Introduction

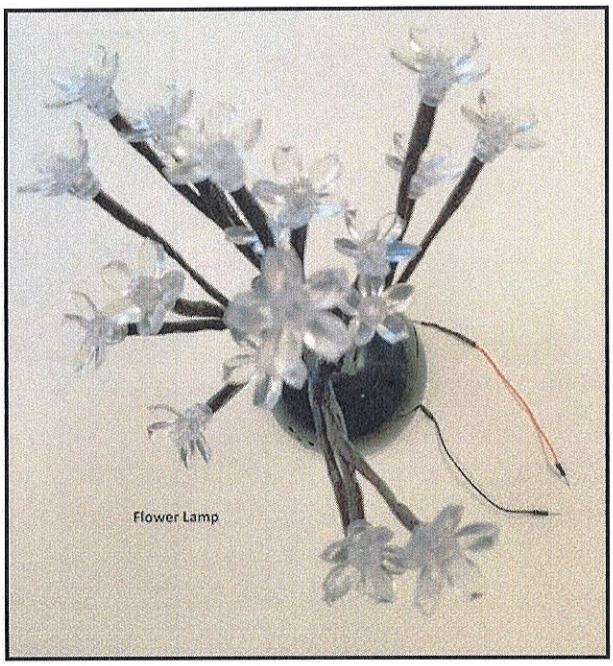


Smart assistants like Amazon Alexa are quickly becoming more useful and central to a smart home atmosphere. While there are currently many Alexa-enabled devices available to control your lights, appliances, and electronics, there is no easy way to design your own devices without using microcontrollers. AWS IoT is a managed cloud platform that lets connected devices easily and securely interact with cloud applications and other devices. In this lesson, we will go over how to utilize an ESP wireless microcontroller to control various smart home functions through Alexa!

Internet of Things (IoT) is a concept that aims to expand the benefits of connected internet connectivity continuously -the ability to share data, remote control, and etc, as well as on objects in the real world. With the adaptation for the ESP8266 modules of the Arduino libraries WiFiClient, it is very easy to exchange data with a home automation server or an online service over TCP/IP protocol. In this lesson, you can learn how to build an ESP8266 based Weather Display unit that retrieve localized weather information from http://www.wunderground.com/ by WLAN and display it on a 128x64 OLED Display. The Weather Station real-time data obtained from the Weather Underground (http://www.wunderground.com) website.

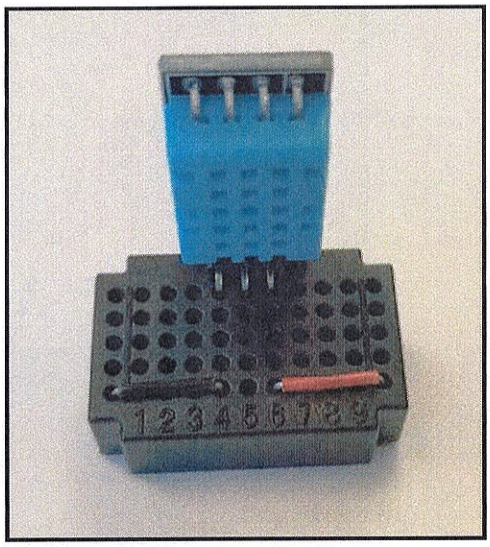
Required Components



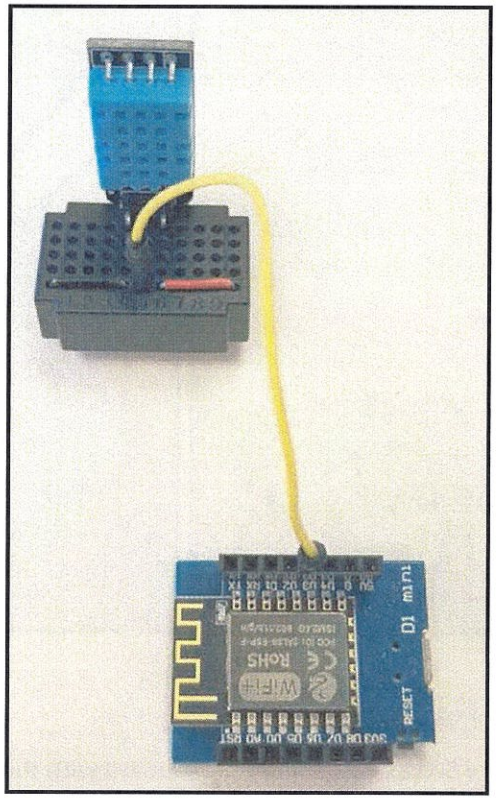


How to Build the Project

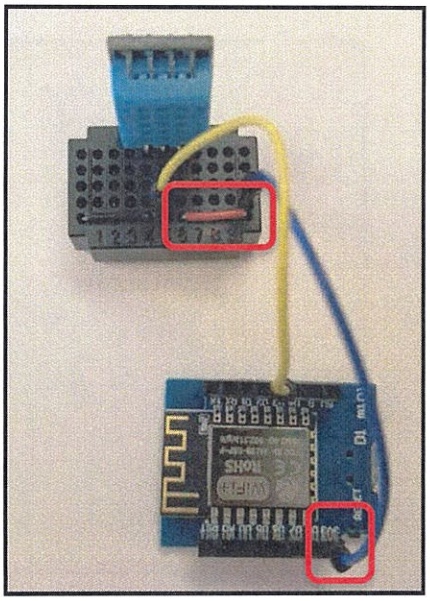
1. Add the Temperature Sensor to the mini-breadboard with the pins in columns 4, 5, and 6.
    a. Use the **BLACK** small Jumper Wire to connect column 4 to the left-most column on the board.
    b. Use the **RED** small Jumper Wire to connect column 6 to the right-most column of the board.



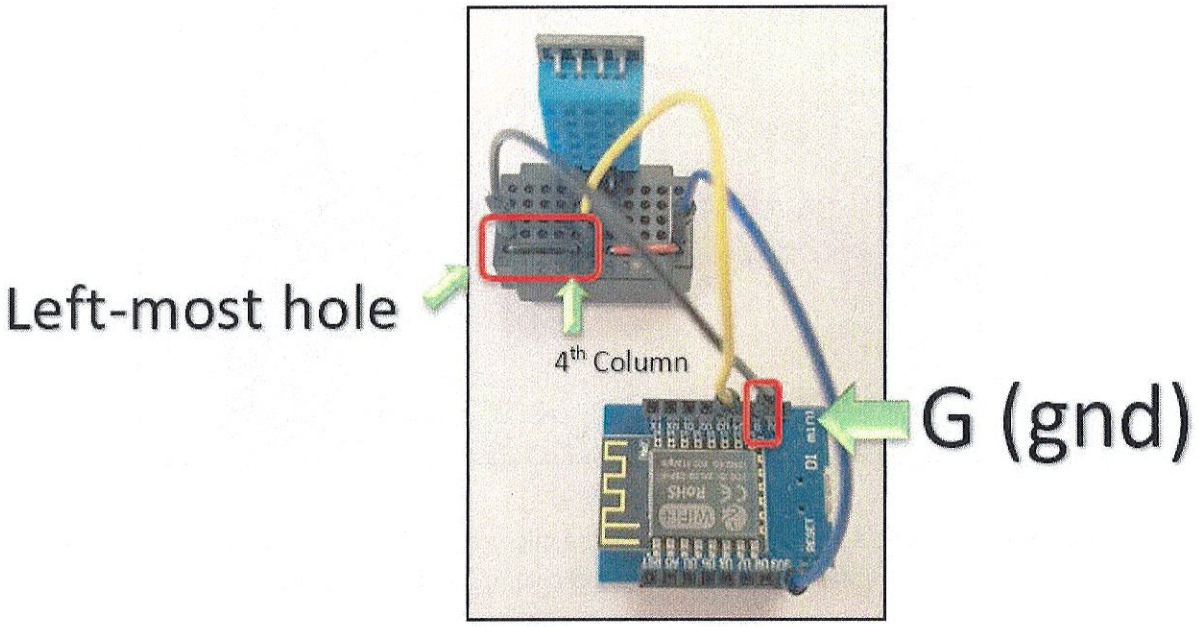
2. Use a wire to connect column 5, which is where the temperature sensor's data pin is, to the microcontroller's D3 pin.



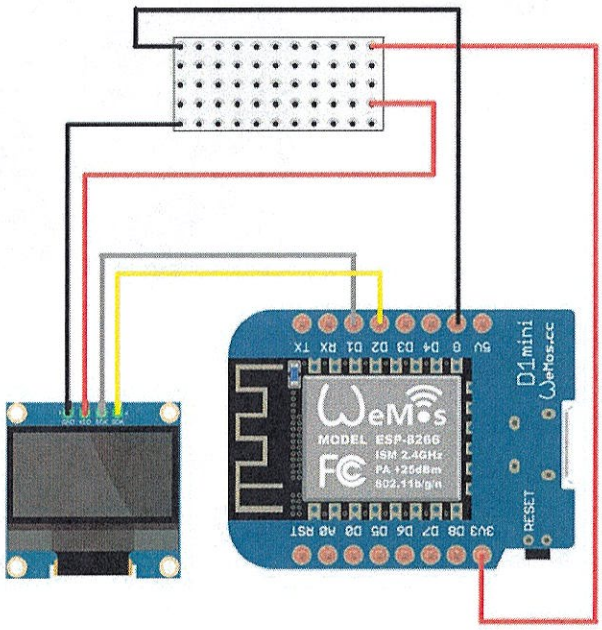
3. Connect the 3.3V power output pin on the microcontroller to the rightmost column on the breadboard.

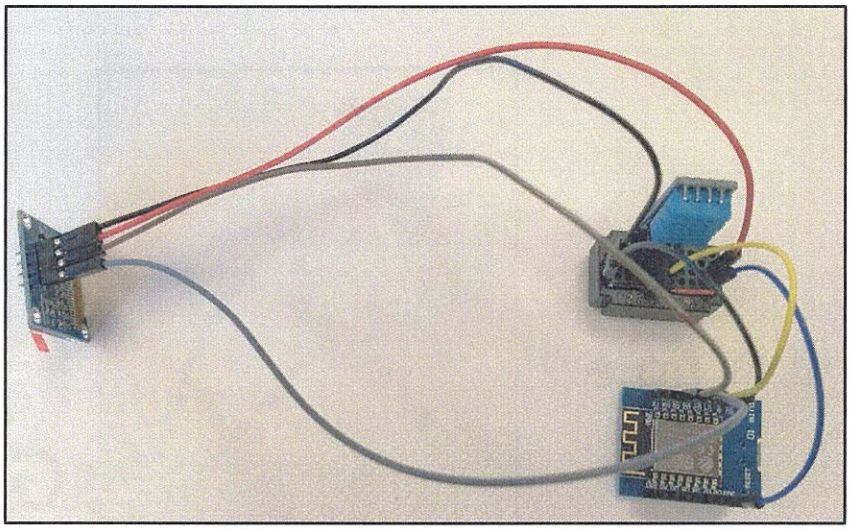


4. Connect the microcontroller's ground pin to the leftmost column of the breadboard.

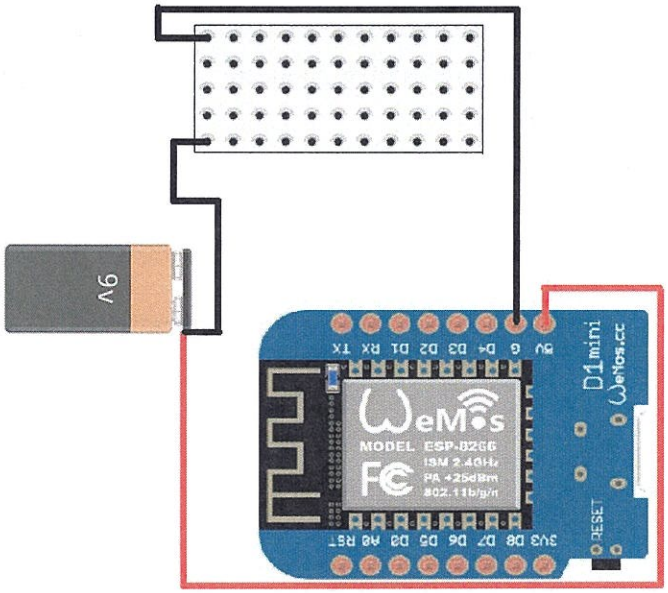


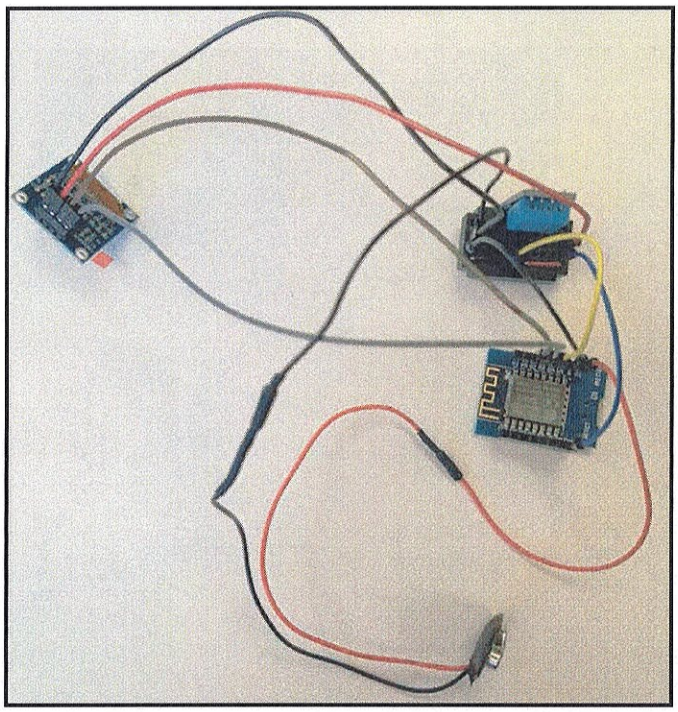
5. Connect the 'Female to Male' wires to the pins on the OLED Display.
    a. The pin labeled **GND** needs to be connected to the **ground** column on the leftmost part of the breadboard.
    b. The **VDD** pin. connects to the **3.3V** column on the right side of the breadboard.
    c. The **SCK** pin connects to the microcontroller's **D1** pin.
    d. The remaining pin labeled **SDA** is connected to the **D2** pin on the microcontroller



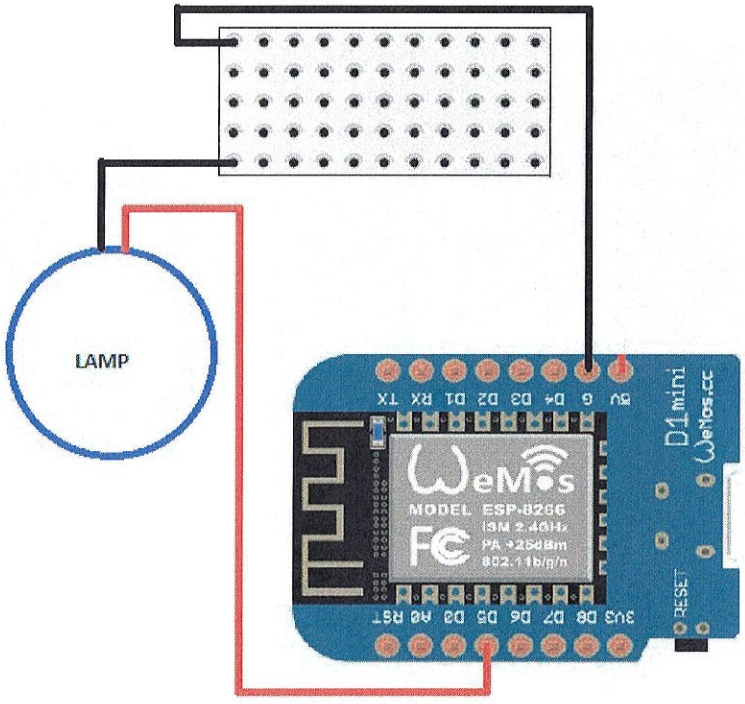


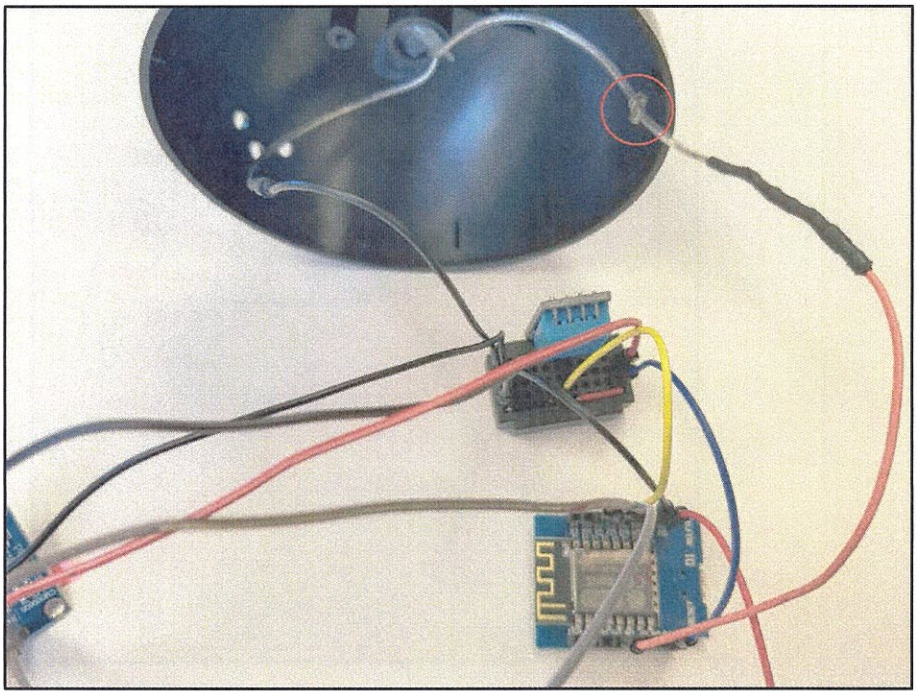
6. Add the Battery Clip's positive (**red**) wire to the microcontroller's **5V pin** and the clip's negative (**black**) wire to the **ground** column on the breadboard.





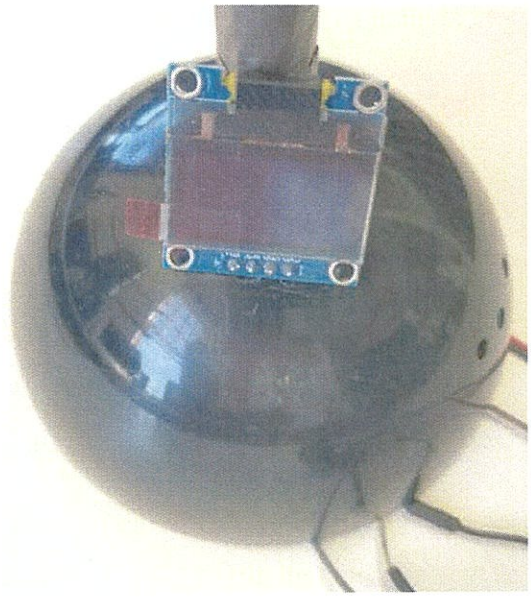
7. Wire the Lamp's positive wire (the one with the knot in it) to the D5 pin on the microcontroller. The other wire in the lamp gets connected to the ground column on the breadboard.



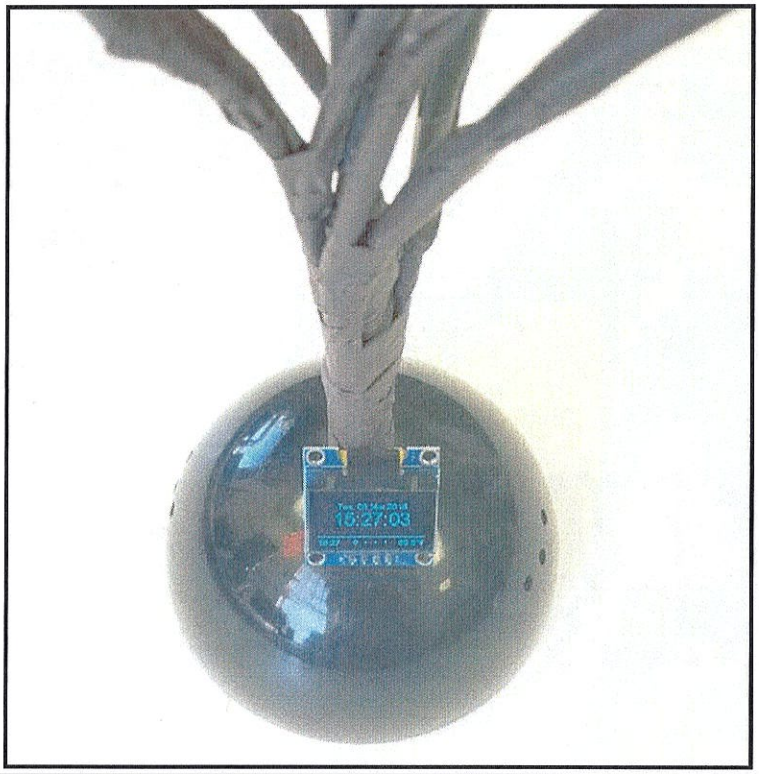


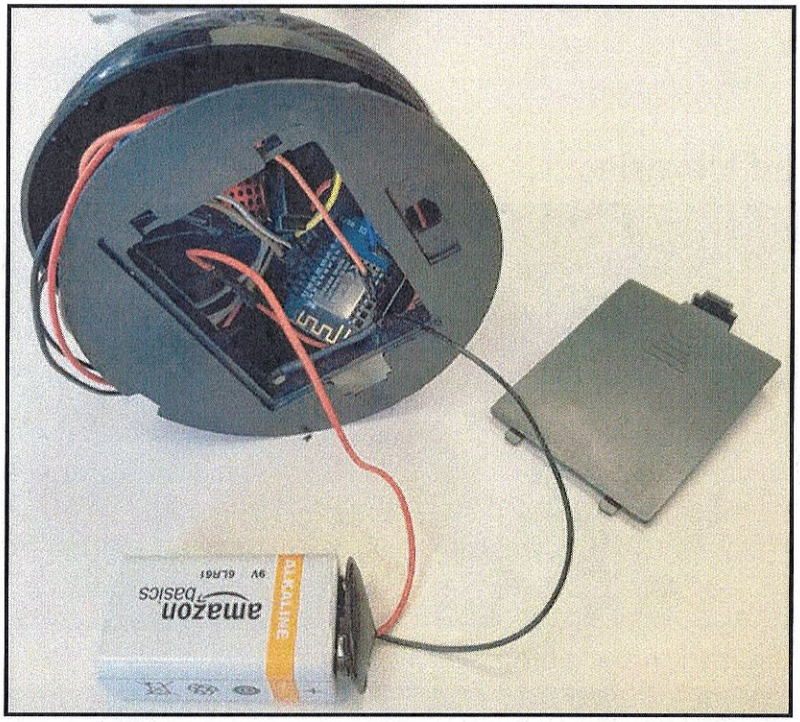
8. The lamp has an opening on its base for the OLED Display to be fitted on the outside. Place the pins for the display through the hole and rewire its pins.

e

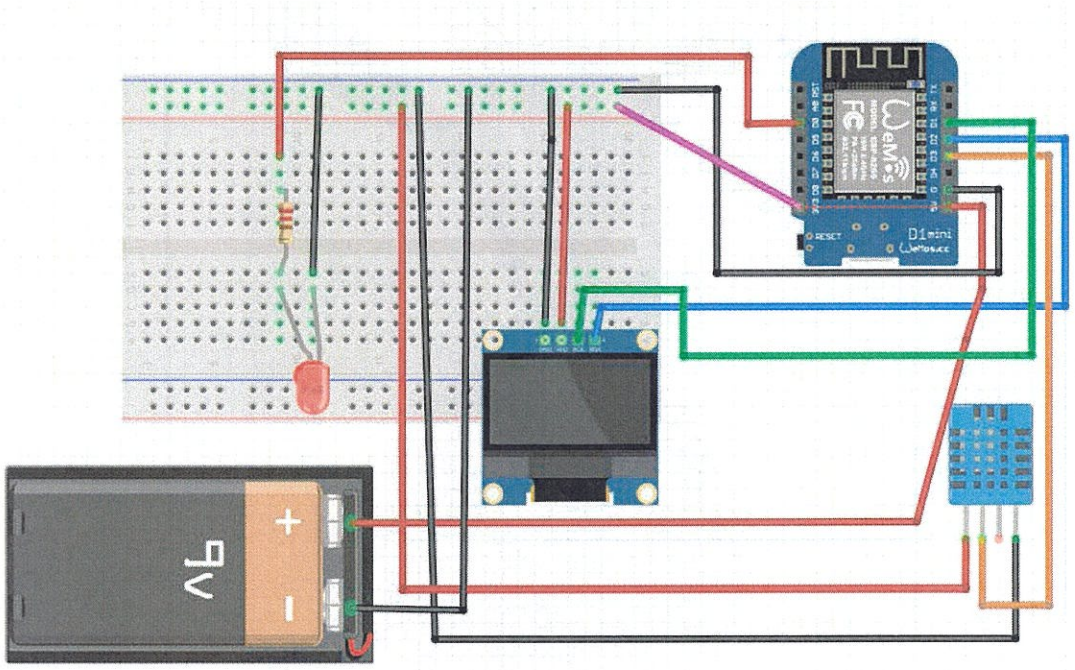


Assembly Complete





Circuit Sketch



The Code

Included Libraries

```
#include <ESP8266WiFi.h>
#include <Ticker.h>
#include <JsonListener.h>
#include "SSD1306Wire.h"
#include "OLEDDisplayUi.h"
#include "Wire.h"
#include "WundergroundClient.h"
#include "WeatherStationFonts.h"
#include "WeatherStationImages.h"
#include "TimeClient.h"
#include "ThingspeakClient.h"
```

Wifi Configuration Variables

```
// WIFI
const char* WIFI_SSID = "TP-LINK_AE7C";
const char* WIFI_PWD = "57988221";
```

DHT Setup

```
// DHT
#include "DHT.h"
#define DHTPIN D3         // what digital pin we're connected to
#define DHTTYPE DHT11     // DHT 11
DHT dht(DHTPIN, DHTTYPE);
```

Setup for Update Interval, Display, and TimeClient

```
// Setup
const int UPDATE_INTERVAL_SECS = 10 * 60; // Update every 10 minutes

// Display Settings
const int I2C_DISPLAY_ADDRESS = 0x3c;
const int SDA_PIN = D2;
const int SDC_PIN = D1;

// TimeClient settings
const float UTC_OFFSET = -5;
```

Wunderground API Settings

```
// Wunderground Settings
const boolean IS_METRIC = false;
const String WUNDERGRROUND_API_KEY = "536f741d66bde1bb";
const String WUNDERGRROUND_LANGUAGE = "EN";
const String WUNDERGROUND_COUNTRY = "OH";
const String WUNDERGROUND_CITY = "ATHENS";
```

OLED Display Setup

```
// Initialize the oled display for address 0x3c
// sda-pin=14 and sdc-pin=12
SSD1306Wire     display(I2C_DISPLAY_ADDRESS, SDA_PIN, SDC_PIN);
OLEDDisplayUi   ui( &display );
```

Global Variables and Object Instances

```
TimeClient timeClient(UTC_OFFSET);

// Set to false, if you prefere imperial/inches, Fahrenheit
WundergroundClient wunderground(IS_METRIC);

ThingspeakClient thingspeak;

// flag changed in the ticker function every 10 minutes
bool readyForWeatherUpdate = false;

String lastUpdate = "--";

Ticker ticker;
```

Forward Declarations for Project Functions

```
//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 drawThingspeak(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();
```

Frame and Overlay Settings

```
// Add frames
// this array keeps function pointers to all frames
// frames are the single views that slide from right to left
FrameCallback frames[] = { drawDateTime, drawCurrentWeather, drawForecast, drawThingspeak };
int numberOfFrames = 4;

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

Setup Function

```
120  void setup() {
121    Serial.begin(115200);
122    Serial.println();
123    Serial.println();
124
125    // initialize dispaly
126    display.init();
127    display.clear();
128    display.display();
129
130    display.flipScreenVertically();
131    display.setFont(ArialMT_Plain_10);
132    display.setTextAlignment(TEXT_ALIGN_CENTER);
133    display.setContrast(255);
134
135    WiFi.begin(WIFI_SSID, WIFI_PWD);
136    dht.begin();
137
138    int counter = 0;
139    while (WiFi.status() != WL_CONNECTED) {
140      delay(500);
141      Serial.print(".");
142      display.clear();
143      display.drawString(64, 10, "Connecting to WiFi");
144      display.drawXbm(46, 30, 8, 8, counter % 3 == 0 ? activeSymbole : inactiveSymbole);
145      display.drawXbm(60, 30, 8, 8, counter % 3 == 1 ? activeSymbole : inactiveSymbole);
146      display.drawXbm(74, 30, 8, 8, counter % 3 == 2 ? activeSymbole : inactiveSymbole);
147      display.display();
148
149      counter++;
150    }
151
152    ui.setTargetFPS(30);
```

```
154     ui.setActiveSymbol(activeSymbole);
155     ui.setInactiveSymbol(inactiveSymbole);
156
157     // You can change this to
158     // TOP, LEFT, BOTTOM, RIGHT
159     ui.setIndicatorPosition(BOTTOM);
160
161     // Defines where the first frame is located in the bar.
162     ui.setIndicatorDirection(LEFT_RIGHT);
163
164     // You can change the transition that is used
165     // SLIDE_LEFT, SLIDE_RIGHT, SLIDE_TOP, SLIDE_DOWN
166     ui.setFrameAnimation(SLIDE_LEFT);
167
168     ui.setFrames(frames, numberOfFrames);
169
170     ui.setOverlays(overlays, numberOfOverlays);
171
172     // Inital UI takes care of initalising the display too.
173     ui.init();
174
175     Serial.println("");
176
177     updateData(&display);
178
179     ticker.attach(UPDATE_INTERVAL_SECS, setReadyForWeatherUpdate);
180
181 }
```

Loop Function

```
183  void loop() {
184
185    // Reading temperature or humidity takes about 250 milliseconds!
186    // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
187    float h = dht.readHumidity();
188    // Read temperature as Celsius (the default)
189    float t = dht.readTemperature();
190    // Read temperature as Fahrenheit (isFahrenheit = true)
191    float f = dht.readTemperature(true);
192
193    // Check if any reads failed and exit early (to try again).
194    if (isnan(h) || isnan(t) || isnan(f)) {
195      Serial.println("Failed to read from DHT sensor!");
196      return;
197    }
198
199
200
201    if (readyForWeatherUpdate && ui.getUiState()->frameState == FIXED) {
202      updateData(&display);
203    }
204
205    int remainingTimeBudget = ui.update();
206
207    if (remainingTimeBudget > 0) {
208      // You can do some work here
209      // Don't do stuff if you are below your
210      // time budget.
211      delay(remainingTimeBudget);
212    }
213
214
215  }
```

Draw Progress Function

```
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();
}
```

Update Data Function

```
void updateData(OLEDDisplay *display) {
  drawProgress(display, 10, "Updating time...");
  timeClient.updateTime();
  drawProgress(display, 30, "Updating conditions...");
  wunderground.updateConditions(WUNDERGRROUND_API_KEY, WUNDERGRROUND_LANGUAGE, WUNDERGROUND_COUNTRY, WUNDERGROUND_CITY);
  drawProgress(display, 50, "Updating forecasts...");
  wunderground.updateForecast(WUNDERGRROUND_API_KEY, WUNDERGRROUND_LANGUAGE, WUNDERGROUND_COUNTRY, WUNDERGROUND_CITY);
  drawProgress(display, 80, "Updating thingspeak...");
  thingspeak.getLastChannelItem(THINGSPEAK_CHANNEL_ID, THINGSPEAK_API_READ_KEY);
  lastUpdate = timeClient.getFormattedTime();
  readyForWeatherUpdate = false;
  drawProgress(display, 100, "Done...");
  delay(1000);
}
```

Draw Date Time Function

```
void drawDateTime(OLEDDisplay *display, OLEDDisplayUiState* state, int16_t x, int16_t y) {
  display->setTextAlignment(TEXT_ALIGN_CENTER);
  display->setFont(ArialMT_Plain_10);
  String date = wunderground.getDate();
  int textWidth = display->getStringWidth(date);
  display->drawString(64 + x, 5 + y, date);
  display->setFont(ArialMT_Plain_24);
  String time = timeClient.getFormattedTime();
  textWidth = display->getStringWidth(time);
  display->drawString(64 + x, 15 + y, time);
  display->setTextAlignment(TEXT_ALIGN_LEFT);
}
```

Draw Current Weather Function

```
void drawCurrentWeather(OLEDDisplay *display, OLEDDisplayUiState* state, int16_t x, int16_t y) {
  display->setFont(ArialMT_Plain_10);
  display->setTextAlignment(TEXT_ALIGN_LEFT);
  display->drawString(60 + x, 5 + y, wunderground.getWeatherText());

  display->setFont(ArialMT_Plain_24);
  String temp = wunderground.getCurrentTemp() + "°F";
  display->drawString(60 + x, 15 + y, temp);
  int tempWidth = display->getStringWidth(temp);

  display->setFont(Meteocons_Plain_42);
  String weatherIcon = wunderground.getTodayIcon();
  int weatherIconWidth = display->getStringWidth(weatherIcon);
  display->drawString(32 + x - weatherIconWidth / 2, 05 + y, weatherIcon);
}
```

Draw Forecast Function

```
void drawForecast(OLEDDisplay *display, OLEDDisplayUiState* state, int16_t x, int16_t y) {
  drawForecastDetails(display, x, y, 0);
  drawForecastDetails(display, x + 44, y, 2);
  drawForecastDetails(display, x + 88, y, 4);
}
```

Draw Thingspeak Function

```
void drawThingspeak(OLEDDisplay *display, OLEDDisplayUiState* state, int16_t x, int16_t y) {
  // Reading temperature or humidity takes about 250 milliseconds!
  // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
  float h = dht.readHumidity();
  // Read temperature as Celsius (the default)
  float t = dht.readTemperature();
  // Read temperature as Fahrenheit (isFahrenheit = true)
  float f = dht.readTemperature(true);

  String tempf;
  tempf = String(f);

  String humid;
  humid = String(h);

  display->setTextAlignment(TEXT_ALIGN_CENTER);
  display->setFont(ArialMT_Plain_10);
  display->drawString(64 + x, 0 + y, "Indoor DHT11 Sensor");
  display->setFont(ArialMT_Plain_16);
  display->drawString(64 + x, 10 + y, tempf + "°F");
  display->drawString(64 + x, 30 + y, humid + "%");
}
```

Draw Forecast Details Function

```
void drawForecastDetails(OLEDDisplay *display, int x, int y, int dayIndex) {
  display->setTextAlignment(TEXT_ALIGN_CENTER);
  display->setFont(ArialMT_Plain_10);
  String day = wunderground.getForecastTitle(dayIndex).substring(0, 3);
  day.toUpperCase();
  display->drawString(x + 20, y, day);

  display->setFont(Meteocons_Plain_21);
  display->drawString(x + 20, y + 12, wunderground.getForecastIcon(dayIndex));

  display->setFont(ArialMT_Plain_10);
  display->drawString(x + 20, y + 34, wunderground.getForecastLowTemp(dayIndex) + "|" + wunderground.getForecastHighTemp(dayIndex));
  display->setTextAlignment(TEXT_ALIGN_LEFT);
}
```

Draw Header Overlay Function

```
void drawHeaderOverlay(OLEDDisplay *display, OLEDDisplayUiState* state) {
  display->setColor(WHITE);
  display->setFont(ArialMT_Plain_10);
  String time = timeClient.getFormattedTime().substring(0, 5);
  display->setTextAlignment(TEXT_ALIGN_LEFT);
  display->drawString(0, 54, time);
  display->setTextAlignment(TEXT_ALIGN_RIGHT);
  String temp = wunderground.getCurrentTemp() + "°F";
  display->drawString(128, 54, temp);
  display->drawHorizontalLine(0, 52, 128);
}
```

Set Ready For Weather Update Function

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

Additional Resources

https://wiki.wemos.cc/products:d1:d1_mini