ESP32 LoRa Sensor Monitoring with Web Server (Long Range Communication)

In this project, you’ll build a sensor monitoring system using a TTGO LoRa32 SX1276 OLED board that sends temperature, humidity and pressure readings via LoRa radio to an ESP32 LoRa receiver. The receiver displays the latest sensor readings on a web server.



With this project you’ll learn how to:

* Send sensor readings via LoRa radio between two ESP32 boards;
* Add LoRa and Wi-Fi capabilities simultaneously to your projects (LoRa + Web Server on the same ESP32 board);
* Use the TTGO LoRa32 SX1276 OLED board or similar development boards for IoT projects.

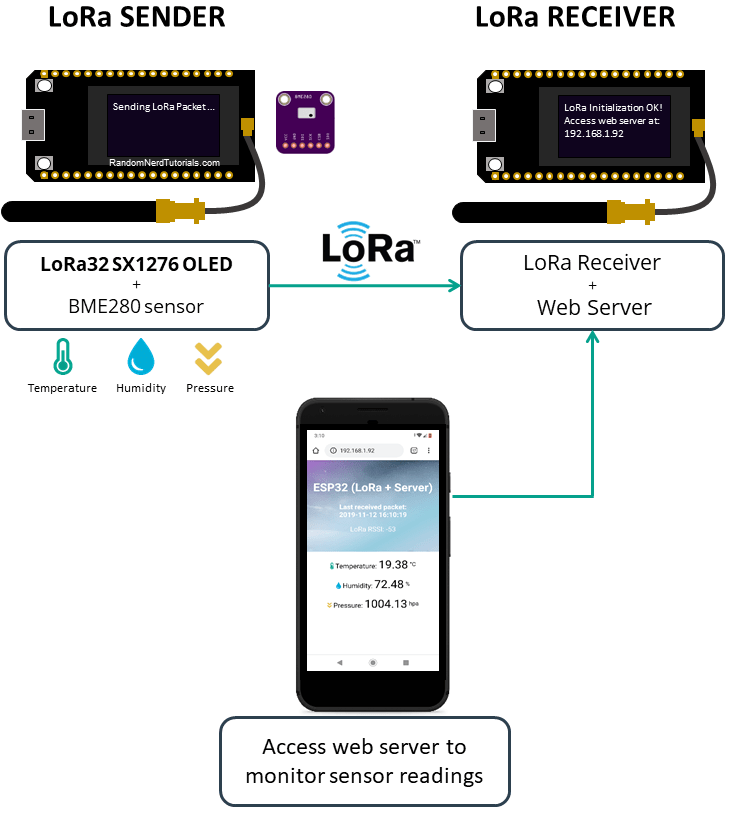
**Recommended reading:** [TTGO LoRa32 SX1276 OLED Board: Getting Started with Arduino IDE](https://randomnerdtutorials.com/ttgo-lora32-sx1276-arduino-ide/)

Watch the Video Demonstration

Watch the video demonstration to see what you’re going to build throughout this tutorial.

Project Overview

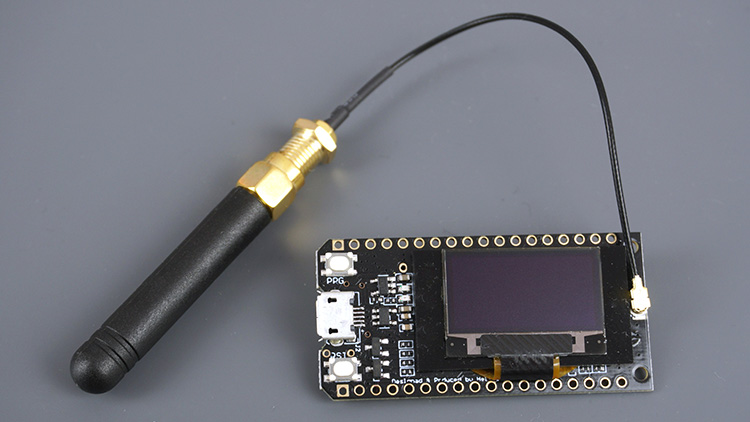
The following image shows a high-level overview of the project we’ll build throughout this tutorial.



* The LoRa sender sends BME280 sensor readings via LoRa radio every 10 seconds;
* The LoRa receiver gets the readings and displays them on a web server;
* You can monitor the sensor readings by accessing the web server;
* The LoRa sender and the Lora receiver can be several hundred meters apart depending on their location. So, you can use this project to monitor sensor readings from your fields or greenhouses if they are a bit apart from your house;
* The LoRa receiver is running an asynchronous web server and the web page files are saved on the [ESP32 filesystem (SPIFFS)](https://randomnerdtutorials.com/esp32-web-server-spiffs-spi-flash-file-system/);
* The LoRa receiver also shows the date and time the last readings were received. To get date and time, we use the [Network Time Protocol with the ESP32](https://randomnerdtutorials.com/esp32-ntp-client-date-time-arduino-ide/).

**For an introduction to LoRa communication:** what’s LoRa, LoRa frequencies, LoRa applications and more, read our [Getting Started ESP32 with LoRa using Arduino IDE](https://randomnerdtutorials.com/esp32-lora-rfm95-transceiver-arduino-ide/).

Parts Required

[](https://makeradvisor.com/tools/ttgo-lora32-sx1276-esp32-oled/)

For this project, we’ll use the following components:

* [**TTGO LoRa32 SX1276 OLED board**](https://makeradvisor.com/tools/ttgo-lora32-sx1276-esp32-oled/)**(2x):** this is an ESP32 development board with a LoRa chip and a built-in OLED. You can use similar boards, or you can use an [ESP32 + LoRa chip + OLED separately](https://randomnerdtutorials.com/esp32-lora-rfm95-transceiver-arduino-ide/).
* [BME280 temperature, humidity and pressure sensor](https://randomnerdtutorials.com/esp32-lora-rfm95-transceiver-arduino-ide/). You should be able to modify this project to use any other sensor.

You’ll also need some [jumper wires](https://makeradvisor.com/tools/jumper-wires-kit-120-pieces/) and a [breadboard](https://makeradvisor.com/tools/mb-102-solderless-breadboard-830-points/).

You can use the preceding links or go directly to [MakerAdvisor.com/tools](https://makeradvisor.com/tools/?utm_source=rnt&utm_medium=post&utm_campaign=post) to find all the parts for your projects at the best price!

[https://i1.wp.com/randomnerdtutorials.com/wp-content/uploads/2017/10/header-200.png?w=840&ssl=1](https://makeradvisor.com/tools/?utm_source=rnt&utm_medium=post&utm_campaign=post)

Preparing the Arduino IDE

To program the TTGO LoRa32 SX1276 OLED boards we’ll use [Arduino IDE](https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/). To upload files to the ESP32 filesystem, we’ll use the [ESP32 filesystem uploader plugin](https://randomnerdtutorials.com/install-esp32-filesystem-uploader-arduino-ide/).

**So, before proceeding, you need to**[**install the ESP32 package**](https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/)**and the**[**ESP32 filesystem uploader plugin**](https://randomnerdtutorials.com/install-esp32-filesystem-uploader-arduino-ide/) in your Arduino IDE.

Installing libraries

For this project you need to install several libraries.

LoRa, BME280 and OLED Libraries

The following libraries can be installed through the Arduino Library Manager. Go to **Sketch**> **Include Library**> **Manage Libraries** and search for the library name.

* LoRa library: [arduino-LoRa library by sandeep mistry](https://github.com/sandeepmistry/arduino-LoRa)
* OLED libraries: [Adafruit\_SSD1306 library](https://github.com/adafruit/Adafruit_SSD1306) and [Adafruit\_GFX library](https://github.com/adafruit/Adafruit-GFX-Library)
* BME280 libraries: [Adafruit\_BME280 library](https://github.com/adafruit/Adafruit_BME280_Library) and [Adafruit unified sensor library](https://github.com/adafruit/Adafruit_Sensor)

Asynchronous Web Server Libraries

To build the [asynchronous web server](https://randomnerdtutorials.com/esp32-dht11-dht22-temperature-humidity-web-server-arduino-ide/), you also need to install the following libraries:

These libraries are not available to install through the Library Manager. So, you need to unzip the libraries and move them to the Arduino IDE installation libraries folder.

Alternatively, you can go to **Sketch**> **Include Library** > **Add .ZIP library…** and select the libraries you’ve just downloaded.

NTPClient Library

Everytime the LoRa receiver picks up a new a LoRa message, it will request the date and time from an NTP server so that we know when the last packet was received.

For that we’ll be using the [NTPClient library forked by Taranais](https://github.com/taranais/NTPClient). Follow the next steps to install this library in your Arduino IDE:

1. [Click here to download the NTPClient library](https://github.com/taranais/NTPClient/archive/master.zip). You should have a .zip folder in your Downloads
2. Unzip the *.zip* folder and you should get **NTPClient-master**folder
3. Rename your folder from  to **NTPClient**
4. Move the **NTPClient**folder to your Arduino IDE installation **libraries**folder
5. Finally, re-open your Arduino IDE

Alternatively, you can go to **Sketch**> **Include Library**> **Add .ZIP library…** and select the library you’ve just downloaded.

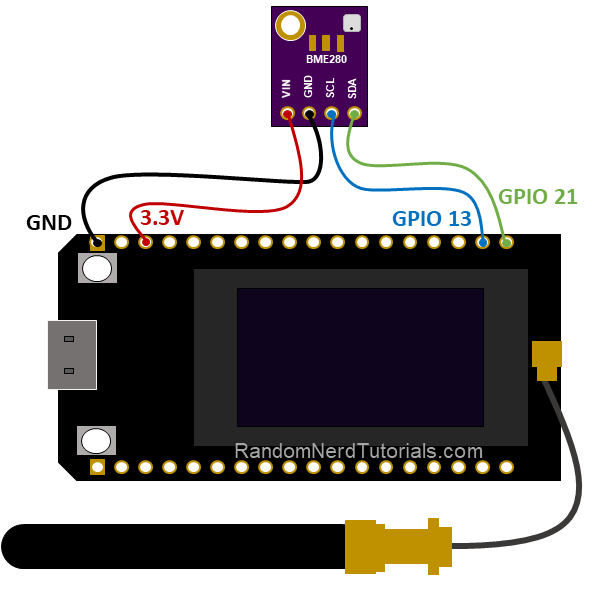
LoRa Sender

The LoRa Sender is connected to a [BME280 sensor](https://randomnerdtutorials.com/esp32-bme280-arduino-ide-pressure-temperature-humidity/) and sends temperature, humidity, and pressure readings every 10 seconds. You can change this period of time later in the code.

**Recommended reading:** [ESP32 with BME280 Sensor using Arduino IDE (Pressure, Temperature, Humidity)](https://randomnerdtutorials.com/esp32-bme280-arduino-ide-pressure-temperature-humidity/)

LoRa Sender Circuit

The BME280 we’re using communicates with the ESP32 using I2C communication protocol. Wire the sensor as shown in the next schematic diagram:



|  |  |
| --- | --- |
| **BME280** | **ESP32** |
| VIN | 3.3 V |
| GND | GND |
| SCL | GPIO 13 |
| SDA | GPIO 21 |

LoRa Sender Code

The following code reads temperature, humidity and pressure from the BME280 sensor and sends the readings via LoRa radio.

Copy the following code to your Arduino IDE.

/\*\*\*\*\*\*\*\*\*

Rui Santos

Complete project details at https://RandomNerdTutorials.com/esp32-lora-sensor-web-server/

Permission is hereby granted, free of charge, to any person obtaining a copy

of this software and associated documentation files.

The above copyright notice and this permission notice shall be included in all

copies or substantial portions of the Software.

\*\*\*\*\*\*\*\*\*/

//Libraries for LoRa

#include <SPI.h>

#include <LoRa.h>

//Libraries for OLED Display

#include <Wire.h>

#include <Adafruit\_GFX.h>

#include <Adafruit\_SSD1306.h>

//Libraries for BME280

#include <Adafruit\_Sensor.h>

#include <Adafruit\_BME280.h>

//define the pins used by the LoRa transceiver module

#define SCK 5

#define MISO 19

#define MOSI 27

#define SS 18

#define RST 14

#define DIO0 26

//433E6 for Asia

//866E6 for Europe

//915E6 for North America

#define BAND 866E6

//OLED pins

#define OLED\_SDA 4

#define OLED\_SCL 15

#define OLED\_RST 16

#define SCREEN\_WIDTH 128 // OLED display width, in pixels

#define SCREEN\_HEIGHT 64 // OLED display height, in pixels

//BME280 definition

#define SDA 21

#define SCL 13

TwoWire I2Cone = TwoWire(1);

Adafruit\_BME280 bme;

//packet counter

int readingID = 0;

int counter = 0;

String LoRaMessage = "";

float temperature = 0;

float humidity = 0;

float pressure = 0;

Adafruit\_SSD1306 display(SCREEN\_WIDTH, SCREEN\_HEIGHT, &Wire, OLED\_RST);

//Initialize OLED display

void startOLED(){

//reset OLED display via software

pinMode(OLED\_RST, OUTPUT);

digitalWrite(OLED\_RST, LOW);

delay(20);

digitalWrite(OLED\_RST, HIGH);

//initialize OLED

Wire.begin(OLED\_SDA, OLED\_SCL);

if(!display.begin(SSD1306\_SWITCHCAPVCC, 0x3c, false, false)) { // Address 0x3C for 128x32

Serial.println(F("SSD1306 allocation failed"));

for(;;); // Don't proceed, loop forever

}

display.clearDisplay();

display.setTextColor(WHITE);

display.setTextSize(1);

display.setCursor(0,0);

display.print("LORA SENDER");

}

//Initialize LoRa module

void startLoRA(){

//SPI LoRa pins

SPI.begin(SCK, MISO, MOSI, SS);

//setup LoRa transceiver module

LoRa.setPins(SS, RST, DIO0);

while (!LoRa.begin(BAND) && counter < 10) {

Serial.print(".");

counter++;

delay(500);

}

if (counter == 10) {

// Increment readingID on every new reading

readingID++;

Serial.println("Starting LoRa failed!");

}

Serial.println("LoRa Initialization OK!");

display.setCursor(0,10);

display.clearDisplay();

display.print("LoRa Initializing OK!");

display.display();

delay(2000);

}

void startBME(){

I2Cone.begin(SDA, SCL, 100000);

bool status1 = bme.begin(0x76, &I2Cone);

if (!status1) {

Serial.println("Could not find a valid BME280\_1 sensor, check wiring!");

while (1);

}

}

void getReadings(){

temperature = bme.readTemperature();

humidity = bme.readHumidity();

pressure = bme.readPressure() / 100.0F;

}

void sendReadings() {

LoRaMessage = String(readingID) + "/" + String(temperature) + "&" + String(humidity) + "#" + String(pressure);

//Send LoRa packet to receiver

LoRa.beginPacket();

LoRa.print(LoRaMessage);

LoRa.endPacket();

display.clearDisplay();

display.setCursor(0,0);

display.setTextSize(1);

display.print("LoRa packet sent!");

display.setCursor(0,20);

display.print("Temperature:");

display.setCursor(72,20);

display.print(temperature);

display.setCursor(0,30);

display.print("Humidity:");

display.setCursor(54,30);

display.print(humidity);

display.setCursor(0,40);

display.print("Pressure:");

display.setCursor(54,40);

display.print(pressure);

display.setCursor(0,50);

display.print("Reading ID:");

display.setCursor(66,50);

display.print(readingID);

display.display();

Serial.print("Sending packet: ");

Serial.println(readingID);

readingID++;

}

void setup() {

//initialize Serial Monitor

Serial.begin(115200);

startOLED();

startBME();

startLoRA();

}

void loop() {

getReadings();

sendReadings();

delay(10000);

}

[View raw code](https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/raw/master/Projects/ESP32/ESP3_LoRa/LoRa_Sender_BME280/LoRa_Sender_BME280.ino)

How the Code Works

Start by including the necessary libraries for LoRa, OLED display and BME280 sensor.

//Libraries for LoRa

#include <SPI.h>

#include <LoRa.h>

//Libraries for OLED Display

#include <Wire.h>

#include <Adafruit\_GFX.h>

#include <Adafruit\_SSD1306.h>

//Libraries for BME280

#include <Adafruit\_Sensor.h>

#include <Adafruit\_BME280.h>

Define the pins used by the LoRa transceiver module. We’re using the [TTGO LoRa32 SX1276 OLED board V1.0](https://makeradvisor.com/esp32-sx1276-lora-ssd1306-oled/) and these are the pins used by the LoRa chip:

//define the pins used by the LoRa transceiver module

#define SCK 5

#define MISO 19

#define MOSI 27

#define SS 18

#define RST 14

#define DIO0 26

**Note:**if you’re using another LoRa board, check the pins used by the LoRa transceiver chip.

Select the LoRa frequency:

#define BAND 866E6

Define the OLED pins.

#define OLED\_SDA 4

#define OLED\_SCL 15

#define OLED\_RST 16

Define the OLED size.

#define SCREEN\_WIDTH 128 // OLED display width, in pixels

#define SCREEN\_HEIGHT 64 // OLED display height, in pixels

Define the pins used by the BME280 sensor.

//BME280 definition

#define SDA 21

#define SCL 13

Create an I2C instance for the BME280 sensor and a bme object.

TwoWire I2Cone = TwoWire(1);

Adafruit\_BME280 bme;

Create some variables to hold the LoRa message, temperature, humidity, pressure and reading ID.

int readingID = 0;

int counter = 0;

String LoRaMessage = "";

float temperature = 0;

float humidity = 0;

float pressure = 0;

Create a display object for the OLED display.

Adafruit\_SSD1306 display(SCREEN\_WIDTH, SCREEN\_HEIGHT, &Wire, OLED\_RST);

*setup()*

In the setup(), we call several functions that were created previously in the code to initialize the OLED display, the BME280 and the LoRa transceiver module.

void setup() {

Serial.begin(115200);

startOLED();

startBME();

startLoRA();

}

*loop()*

In the loop(), we call the getReadings() and sendReadings() functions that were also previously created. These functions are responsible for getting readings from the BME280 sensor, and to send those readings via LoRa, respectively.

void loop() {

getReadings();

sendReadings();

delay(10000);

}

**getReadings()**

Getting sensor readings is as simple as using the readTemperature(), readHumidity(), and readPressure() methods on the bme object:

void getReadings(){

temperature = bme.readTemperature();

humidity = bme.readHumidity();

pressure = bme.readPressure() / 100.0F;

}

**sendReadings()**

To send the readings via LoRa, we concatenate all the readings on a single variable, LoRaMessage:

void sendReadings() {

LoRaMessage = String(readingID) + "/" + String(temperature) + "&" + String(humidity) + "#" + String(pressure);

Note that each reading is separated with a special character, so the receiver can easily identify each value.

Then, send the packet using the following:

LoRa.beginPacket();

LoRa.print(LoRaMessage);

LoRa.endPacket();

Each time we send a LoRa packet, we increase the readingID variable so that we have an idea on how many packets were sent. You can delete this variable if you want.

readingID++;

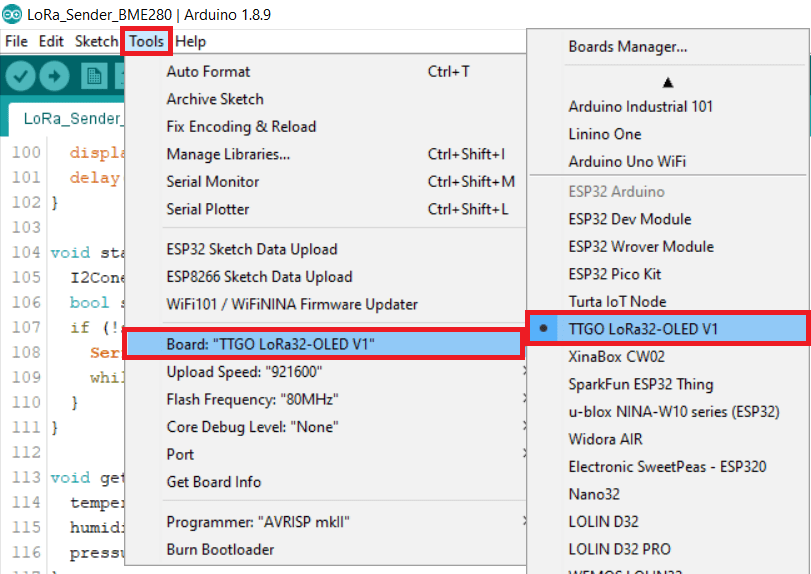
The loop() is repeated every 10000 milliseconds (10 seconds). So, new sensor readings are sent every 10 seconds. You can change this delay time if you want.

delay(10000);

Testing the LoRa Sender

Upload the code to your ESP32 LoRa Sender Board.

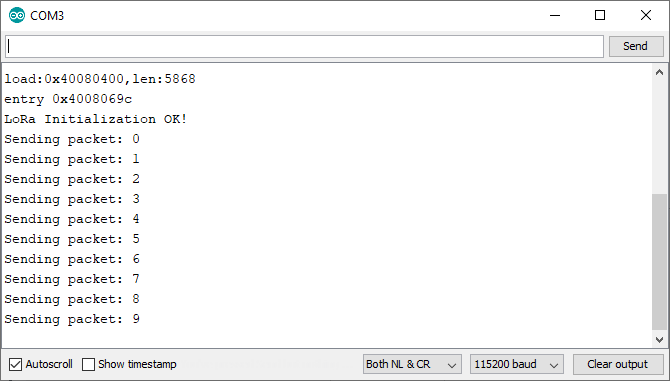
Go to **Tools**> **Port** and select the COM port it is connected to. Then, go to **Tools**> **Board**and select the board you’re using. In our case, it’s the TTGO LoRa32-OLED V1.



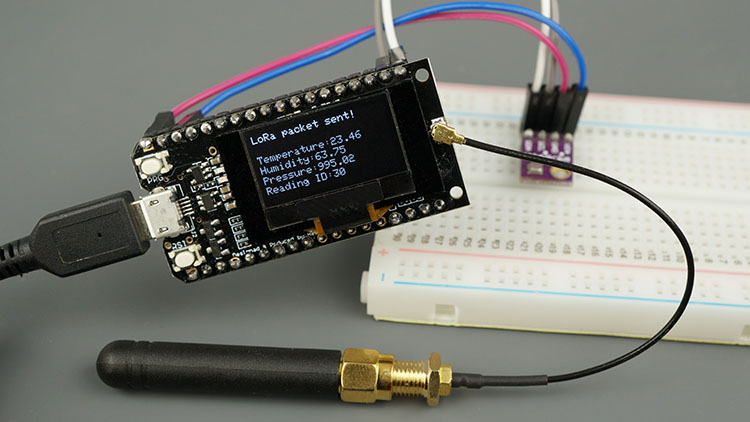
Finally, press the upload button.

Arduino IDE Upload button

Open the Serial Monitor at a baud rate of 115200. You should get something as shown below.



The OLED of your board should be displaying the latest sensor readings.

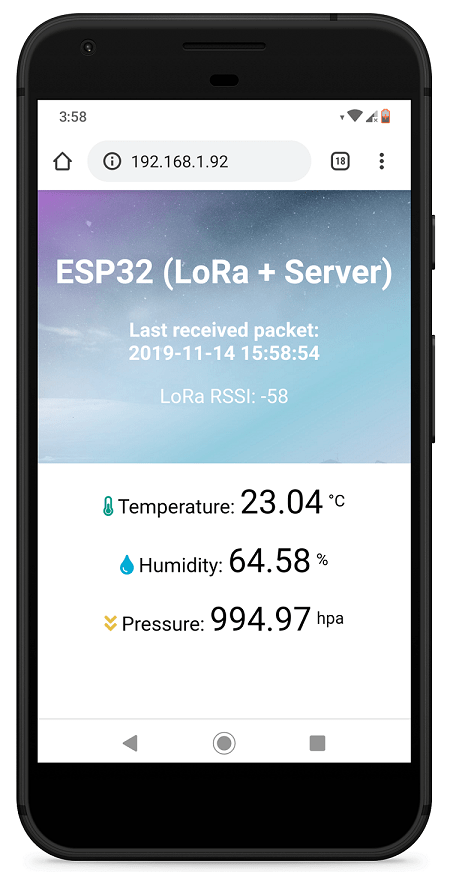


Your LoRa Sender is ready. Now, let’s move on to the LoRa Receiver.

LoRa Receiver

The LoRa Receiver gets incoming LoRa packets and displays the received readings on an asynchronous web server. Besides the sensor readings, we also display the last time those readings were received and the RSSI (received signal strength indicator).

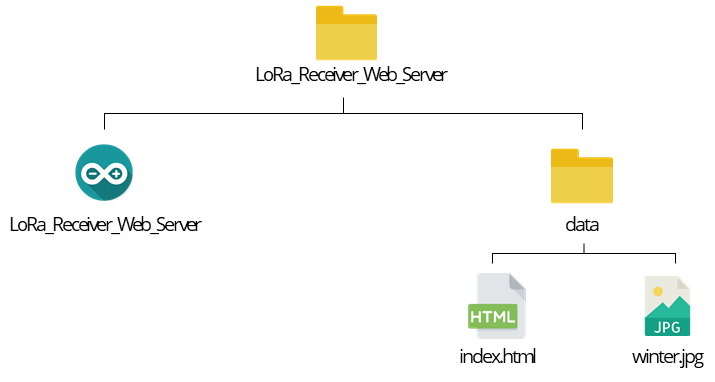
The following figure shows the web server we’ll build.



As you can see, it contains a background image and styles to make the web page more appealing. There are several ways to [display images on an ESP32 web server](https://randomnerdtutorials.com/display-images-esp32-esp8266-web-server/). We’ll store the image on the ESP32 filesystem (SPIFFS). We’ll also store the HTML file on SPIFFS.

Organizing your Files

To build the web server you need three different files: the Arduino sketch, the HTML file and the image. The HTML file and the image should be saved inside a folder called **data** inside the Arduino sketch folder, as shown below.



Creating the HTML File

Create an *index.html* file with the following content or [**download all the project files here**](https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/raw/master/Projects/ESP32/ESP3_LoRa/ESP3_LoRa.zip):

<!DOCTYPE HTML><html>

<head>

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="icon" href="data:,">

<title>ESP32 (LoRa + Server)</title>

<link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.2/css/all.css" integrity="sha384-fnmOCqbTlWIlj8LyTjo7mOUStjsKC4pOpQbqyi7RrhN7udi9RwhKkMHpvLbHG9Sr" crossorigin="anonymous">

<style>

body {

margin: 0;

font-family: Arial, Helvetica, sans-serif;

text-align: center;

}

header {

margin: 0;

padding-top: 5vh;

padding-bottom: 5vh;

overflow: hidden;

background-image: url(winter);

background-size: cover;

color: white;

}

h2 {

font-size: 2.0rem;

}

p { font-size: 1.2rem; }

.units { font-size: 1.2rem; }

.readings { font-size: 2.0rem; }

</style>

</head>

<body>

<header>

<h2>ESP32 (LoRa + Server)</h2>

<p><strong>Last received packet:<br/><span id="timestamp">%TIMESTAMP%</span></strong></p>

<p>LoRa RSSI: <span id="rssi">%RSSI%</span></p>

</header>

<main>

<p>

<i class="fas fa-thermometer-half" style="color:#059e8a;"></i> Temperature: <span id="temperature" class="readings">%TEMPERATURE%</span>

<sup>&deg;C</sup>

</p>

<p>

<i class="fas fa-tint" style="color:#00add6;"></i> Humidity: <span id="humidity" class="readings">%HUMIDITY%</span>

<sup>&#37;</sup>

</p>

<p>

<i class="fas fa-angle-double-down" style="color:#e8c14d;"></i> Pressure: <span id="pressure" class="readings">%PRESSURE%</span>

<sup>hpa</sup>

</p>

</main>

<script>

setInterval(updateValues, 10000, "temperature");

setInterval(updateValues, 10000, "humidity");

setInterval(updateValues, 10000, "pressure");

setInterval(updateValues, 10000, "rssi");

setInterval(updateValues, 10000, "timestamp");

function updateValues(value) {

var xhttp = new XMLHttpRequest();

xhttp.onreadystatechange = function() {

if (this.readyState == 4 && this.status == 200) {

document.getElementById(value).innerHTML = this.responseText;

}

};

xhttp.open("GET", "/" + value, true);

xhttp.send();

}

</script>

</body>

</html>

[View raw code](https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/raw/master/Projects/ESP32/ESP3_LoRa/LoRa_Receiver_Web_Server/data/index.html)

We’ve also included the CSS styles on the HTML file as well as some JavaScript that is responsible for updating the sensor readings automatically.

Something important to notice are the placeholders. The placeholders go between **%** signs: %TIMESTAMP%, %TEMPERATURE%, %HUMIDITY%, %PRESSURE% and %RSSI%.

These placeholders will then be replaced with the actual values by the Arduino code.

The styles are added between the <style> and </style> tags.

<style>

body {

margin: 0;

font-family: Arial, Helvetica, sans-serif;

text-align: center;

}

header {

margin: 0;

padding-top: 10vh;

padding-bottom: 5vh;

overflow: hidden;

width: 100%;

background-image: url(winter.jpg);

background-size: cover;

color: white;

}

h2 {

font-size: 2.0rem;

}

p { font-size: 1.2rem; }

.units { font-size: 1.2rem; }

.readings { font-size: 2.0rem; }

</style>

If you want a different image for your background, you just need to modify the following line to include your image’s name. In our case, it is called *winter.jpg*.

background-image: url(winter.jpg);

The JavaScript goes between the <scritpt> and </script> tags.

<script>

setInterval(updateValues("temperature"), 5000);

setInterval(updateValues("humidity"), 5000);

setInterval(updateValues("pressure"), 5000);

setInterval(updateValues("rssi"), 5000);

setInterval(updateValues("timeAndDate"), 5000);

function updateValues(value) {

console.log(value);

var xhttp = new XMLHttpRequest();

xhttp.onreadystatechange = function() {

if (this.readyState == 4 && this.status == 200) {

document.getElementById(value).innerHTML = this.responseText;

}

};

xhttp.open("GET", "/" + value, true);

xhttp.send();

}

</script>

We won’t explain in detail how the HTML and CSS works, but a good place to learn is the [W3Schools website](https://www.w3schools.com/css/default.asp).

LoRa Receiver Arduino Sketch

Copy the following code to your Arduino IDE or [**download all the project files here**](https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/raw/master/Projects/ESP32/ESP3_LoRa/ESP3_LoRa.zip). Then, you need to type your network credentials (SSID and password) to make it work.

/\*\*\*\*\*\*\*\*\*

Rui Santos

Complete project details at https://RandomNerdTutorials.com/esp32-lora-sensor-web-server/

Permission is hereby granted, free of charge, to any person obtaining a copy

of this software and associated documentation files.

The above copyright notice and this permission notice shall be included in all

copies or substantial portions of the Software.

\*\*\*\*\*\*\*\*\*/

// Import Wi-Fi library

#include <WiFi.h>

#include "ESPAsyncWebServer.h"

#include <SPIFFS.h>

//Libraries for LoRa

#include <SPI.h>

#include <LoRa.h>

//Libraries for OLED Display

#include <Wire.h>

#include <Adafruit\_GFX.h>

#include <Adafruit\_SSD1306.h>

// Libraries to get time from NTP Server

#include <NTPClient.h>

#include <WiFiUdp.h>

//define the pins used by the LoRa transceiver module

#define SCK 5

#define MISO 19

#define MOSI 27

#define SS 18

#define RST 14

#define DIO0 26

//433E6 for Asia

//866E6 for Europe

//915E6 for North America

#define BAND 866E6

//OLED pins

#define OLED\_SDA 4

#define OLED\_SCL 15

#define OLED\_RST 16

#define SCREEN\_WIDTH 128 // OLED display width, in pixels

#define SCREEN\_HEIGHT 64 // OLED display height, in pixels

// Replace with your network credentials

const char\* ssid = "REPLACE\_WITH\_YOUR\_SSID";

const char\* password = "REPLACE\_WITH\_YOUR\_PASSWORD";

// Define NTP Client to get time

WiFiUDP ntpUDP;

NTPClient timeClient(ntpUDP);

// Variables to save date and time

String formattedDate;

String day;

String hour;

String timestamp;

// Initialize variables to get and save LoRa data

int rssi;

String loRaMessage;

String temperature;

String humidity;

String pressure;

String readingID;

// Create AsyncWebServer object on port 80

AsyncWebServer server(80);

Adafruit\_SSD1306 display(SCREEN\_WIDTH, SCREEN\_HEIGHT, &Wire, OLED\_RST);

// Replaces placeholder with DHT values

String processor(const String& var){

//Serial.println(var);

if(var == "TEMPERATURE"){

return temperature;

}

else if(var == "HUMIDITY"){

return humidity;

}

else if(var == "PRESSURE"){

return pressure;

}

else if(var == "TIMESTAMP"){

return timestamp;

}

else if (var == "RRSI"){

return String(rssi);

}

return String();

}

//Initialize OLED display

void startOLED(){

//reset OLED display via software

pinMode(OLED\_RST, OUTPUT);

digitalWrite(OLED\_RST, LOW);

delay(20);

digitalWrite(OLED\_RST, HIGH);

//initialize OLED

Wire.begin(OLED\_SDA, OLED\_SCL);

if(!display.begin(SSD1306\_SWITCHCAPVCC, 0x3c, false, false)) { // Address 0x3C for 128x32

Serial.println(F("SSD1306 allocation failed"));

for(;;); // Don't proceed, loop forever

}

display.clearDisplay();

display.setTextColor(WHITE);

display.setTextSize(1);

display.setCursor(0,0);

display.print("LORA SENDER");

}

//Initialize LoRa module

void startLoRA(){

int counter;

//SPI LoRa pins

SPI.begin(SCK, MISO, MOSI, SS);

//setup LoRa transceiver module

LoRa.setPins(SS, RST, DIO0);

while (!LoRa.begin(BAND) && counter < 10) {

Serial.print(".");

counter++;

delay(500);

}

if (counter == 10) {

// Increment readingID on every new reading

Serial.println("Starting LoRa failed!");

}

Serial.println("LoRa Initialization OK!");

display.setCursor(0,10);

display.clearDisplay();

display.print("LoRa Initializing OK!");

display.display();

delay(2000);

}

void connectWiFi(){

// Connect to Wi-Fi network with SSID and password

Serial.print("Connecting to ");

Serial.println(ssid);

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

// Print local IP address and start web server

Serial.println("");

Serial.println("WiFi connected.");

Serial.println("IP address: ");

Serial.println(WiFi.localIP());

display.setCursor(0,20);

display.print("Access web server at: ");

display.setCursor(0,30);

display.print(WiFi.localIP());

display.display();

}

// Read LoRa packet and get the sensor readings

void getLoRaData() {

Serial.print("Lora packet received: ");

// Read packet

while (LoRa.available()) {

String LoRaData = LoRa.readString();

// LoRaData format: readingID/temperature&soilMoisture#batterylevel

// String example: 1/27.43&654#95.34

Serial.print(LoRaData);

// Get readingID, temperature and soil moisture

int pos1 = LoRaData.indexOf('/');

int pos2 = LoRaData.indexOf('&');

int pos3 = LoRaData.indexOf('#');

readingID = LoRaData.substring(0, pos1);

temperature = LoRaData.substring(pos1 +1, pos2);

humidity = LoRaData.substring(pos2+1, pos3);

pressure = LoRaData.substring(pos3+1, LoRaData.length());

}

// Get RSSI

rssi = LoRa.packetRssi();

Serial.print(" with RSSI ");

Serial.println(rssi);

}

// Function to get date and time from NTPClient

void getTimeStamp() {

while(!timeClient.update()) {

timeClient.forceUpdate();

}

// The formattedDate comes with the following format:

// 2018-05-28T16:00:13Z

// We need to extract date and time

formattedDate = timeClient.getFormattedDate();

Serial.println(formattedDate);

// Extract date

int splitT = formattedDate.indexOf("T");

day = formattedDate.substring(0, splitT);

Serial.println(day);

// Extract time

hour = formattedDate.substring(splitT+1, formattedDate.length()-1);

Serial.println(hour);

timestamp = day + " " + hour;

}

void setup() {

// Initialize Serial Monitor

Serial.begin(115200);

startOLED();

startLoRA();

connectWiFi();

if(!SPIFFS.begin()){

Serial.println("An Error has occurred while mounting SPIFFS");

return;

}

// Route for root / web page

server.on("/", HTTP\_GET, [](AsyncWebServerRequest \*request){

request->send(SPIFFS, "/index.html", String(), false, processor);

});

server.on("/temperature", HTTP\_GET, [](AsyncWebServerRequest \*request){

request->send\_P(200, "text/plain", temperature.c\_str());

});

server.on("/humidity", HTTP\_GET, [](AsyncWebServerRequest \*request){

request->send\_P(200, "text/plain", humidity.c\_str());

});

server.on("/pressure", HTTP\_GET, [](AsyncWebServerRequest \*request){

request->send\_P(200, "text/plain", pressure.c\_str());

});

server.on("/timestamp", HTTP\_GET, [](AsyncWebServerRequest \*request){

request->send\_P(200, "text/plain", timestamp.c\_str());

});

server.on("/rssi", HTTP\_GET, [](AsyncWebServerRequest \*request){

request->send\_P(200, "text/plain", String(rssi).c\_str());

});

server.on("/winter", HTTP\_GET, [](AsyncWebServerRequest \*request){

request->send(SPIFFS, "/winter.jpg", "image/jpg");

});

// Start server

server.begin();

// Initialize a NTPClient to get time

timeClient.begin();

// Set offset time in seconds to adjust for your timezone, for example:

// GMT +1 = 3600

// GMT +8 = 28800

// GMT -1 = -3600

// GMT 0 = 0

timeClient.setTimeOffset(0);

}

void loop() {

// Check if there are LoRa packets available

int packetSize = LoRa.parsePacket();

if (packetSize) {

getLoRaData();

getTimeStamp();

}

}

[View raw code](https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/raw/master/Projects/ESP32/ESP3_LoRa/LoRa_Receiver_Web_Server/LoRa_Receiver_Web_Server.ino)

How the Code Works

You start by including the necessary libraries. You need libraries to:

* build the asynchronous web server;
* access the ESP32 filesystem (SPIFFS);
* communicate with the LoRa chip;
* control the OLED display;
* get date and time from an NTP server.

// Import Wi-Fi library

#include <WiFi.h>

#include "ESPAsyncWebServer.h"

#include <SPIFFS.h>

//Libraries for LoRa

#include <SPI.h>

#include <LoRa.h>

//Libraries for OLED Display

#include <Wire.h>

#include <Adafruit\_GFX.h>

#include <Adafruit\_SSD1306.h>

// Libraries to get time from NTP Server

#include <NTPClient.h>

#include <WiFiUdp.h>

Define the pins used by the LoRa transceiver module.

#define SCK 5

#define MISO 19

#define MOSI 27

#define SS 18

#define RST 14

#define DIO0 26

**Note:**if you’re using another LoRa board, check the pins used by the LoRa transceiver chip.

Define the LoRa frequency:

//433E6 for Asia

//866E6 for Europe

//915E6 for North America

#define BAND 866E6

Set up the OLED pins:

#define OLED\_SDA 4

#define OLED\_SCL 15

#define OLED\_RST 16

#define SCREEN\_WIDTH 128 // OLED display width, in pixels

#define SCREEN\_HEIGHT 64 // OLED display height, in pixels

Enter your network credentials in the following variables so that the ESP32 can connect to your local network.

const char\* ssid = "REPLACE\_WITH\_YOUR\_SSID";

const char\* password = "REPLACE\_WITH\_YOUR\_PASSWORD";

Define an NTP Client to get date and time:

WiFiUDP ntpUDP;

NTPClient timeClient(ntpUDP);

Create variables to save date and time:

String formattedDate;

String day;

String hour;

String timestamp;

More variables to store the sensor readings received via LoRa radio.

int rssi;

String loRaMessage;

String temperature;

String humidity;

String pressure;

String readingID;

Create an AsyncWebServer object called server on port 80.

AsyncWebServer server(80);

Create an object called display for the OLED display:

AsyncWebServer server(80);

*processor()*

The processor() function is what will attribute values to the placeholders we’ve created on the HTML file.

It accepts as argument the placeholder and should return a String that will replace that placeholder.

For example, if it finds the TEMPERATURE placeholder, it will return the temperature String variable.

// Replaces placeholder with DHT values

String processor(const String& var){

//Serial.println(var);

if(var == "TEMPERATURE"){

return temperature;

}

else if(var == "HUMIDITY"){

return humidity;

}

else if(var == "PRESSURE"){

return pressure;

}

else if(var == "TIMESTAMP"){

return timestamp;

}

else if (var == "RRSI"){

return String(rssi);

}

return String();

}

*setup()*

In the setup(), you initialize the OLED display, the LoRa communication, and connect to Wi-Fi.

void setup() {

// Initialize Serial Monitor

Serial.begin(115200);

startOLED();

startLoRA();

connectWiFi();

You also initialize SPIFFS:

if(!SPIFFS.begin()){

Serial.println("An Error has occurred while mounting SPIFFS");

return;

}

**Async Web Server**

The ESPAsyncWebServer library allows us to configure the routes where the server will be listening for incoming HTTP requests.

For example, when a request is received on the route URL, we send the *index.html* file that is saved in the ESP32 SPIFFS:

server.on("/", HTTP\_GET, [](AsyncWebServerRequest \*request){

request->send(SPIFFS, "/index.html", String(), false, processor);

});

As mentioned previously, we added a bit of Javascript to the HTML file that is responsible for updating the web page every 10 seconds. When that happens, it makes a request on the **/temperature**, **/humidity**, **/pressure**, **/timestamp**, **/rssi** URLs.

So, we need to handle what happens when we receive those requests. We simply need to send the temperature, humidity, pressure, timestamp and rssi variables. The variables should be sent in char format, that’s why we use the .c\_str() method.

server.on("/temperature", HTTP\_GET, [](AsyncWebServerRequest \*request){

request->send\_P(200, "text/plain", temperature.c\_str());

});

server.on("/humidity", HTTP\_GET, [](AsyncWebServerRequest \*request){

request->send\_P(200, "text/plain", humidity.c\_str());

});

server.on("/pressure", HTTP\_GET, [](AsyncWebServerRequest \*request){

request->send\_P(200, "text/plain", pressure.c\_str());

});

server.on("/timestamp", HTTP\_GET, [](AsyncWebServerRequest \*request){

request->send\_P(200, "text/plain", timestamp.c\_str());

});

server.on("/rssi", HTTP\_GET, [](AsyncWebServerRequest \*request){

request->send\_P(200, "text/plain", String(rssi).c\_str());

});

Because we included an image in the web page, we’ll get a request “asking” for the image. So, we need to send the image that is saved on the ESP32 SPIFFS.

server.on("/winter", HTTP\_GET, [](AsyncWebServerRequest \*request){

request->send(SPIFFS, "/winter.jpg", "image/jpg");

});

Finally, start the web server.

server.begin();

*NTPClient*

Still in the setup(), create an NTP client to get the time from the internet.

timeClient.begin();

The time is returned in GMT format, so if you need to adjust for your timezone, you can use the following:

// Set offset time in seconds to adjust for your timezone, for example:

// GMT +1 = 3600

// GMT +8 = 28800

// GMT -1 = -3600

// GMT 0 = 0

timeClient.setTimeOffset(0);

loop()

In the loop(), we listen for incoming LoRa packets:

int packetSize = LoRa.parsePacket();

If a new LoRa packet is available, we call the getLoRaData() and getTimeStamp() functions.

if (packetSize) {

getLoRaData();

getTimeStamp();

}

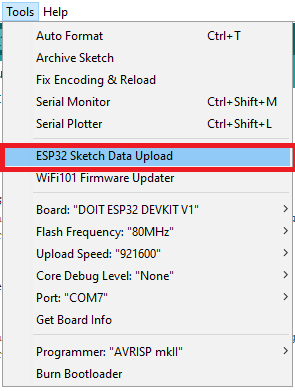
The getLoRaData() function receives the LoRa message and splits it to get the different readings.

The getTimeStamp() function gets the time and date from the internet at the moment we receive the packet.

Uploading Code and Files

After inserting your network credentials, save your sketch. Then, in your Arduino IDE go to **Sketch** > **Show Sketch Folder**, and create a folder called **data**. Inside that folder, you should have the HTML file and the image file.

After making sure you have all the needed files in the right directories, go to **Tools** and select **ESP32 Data Sketch Upload**.



After a few seconds, the files should be successfully uploaded to SPIFFS.

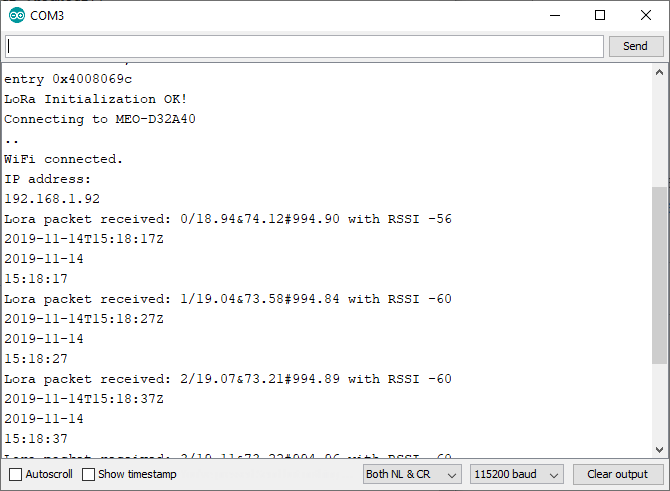
**Note:**if you don’t see the “ESP32 Sketch Data Upload” option that means you don’t have the ESP32 filesystem uploader plugin installed ([how to install the ESP32 filesystem uploader plugin](https://randomnerdtutorials.com/install-esp32-filesystem-uploader-arduino-ide/)).

Now, upload the sketch to your board.

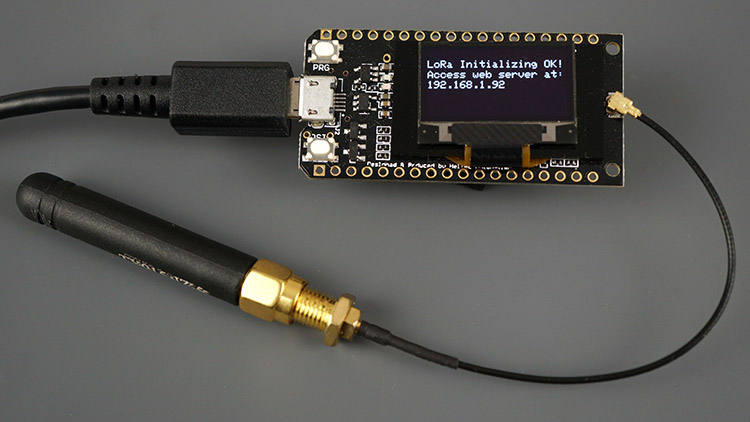
Arduino IDE Upload button

Open the Serial Monitor at a baud rate of 115200.

You should get the ESP32 IP address, and you should start receiving LoRa packets from the sender.

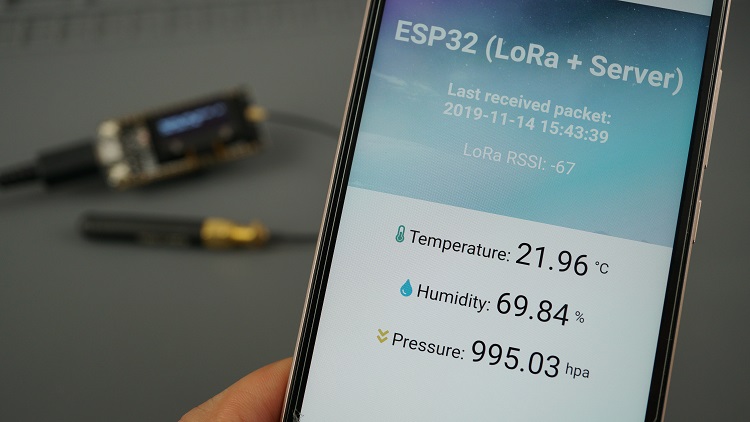


You should also get the IP address displayed on the OLED.



Demonstration

Open a browser and type your ESP32 IP address. You should see the web server with the latest sensor readings.

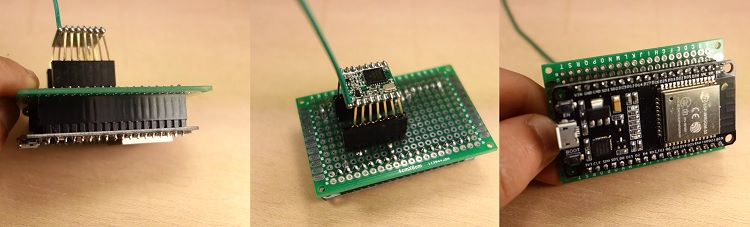


With these boards we were able to get a stable LoRa communication up to 180 meters (590 ft) in open field. These means that we can have the sender and receiver 180 meters apart and we’re still able to get and check the readings on the web server.



Getting a stable communication at a distance of 180 meters with such low cost boards and without any further customization is really impressive.

However, in a previous project using an [RFM95 SX1276 LoRa transceiver chip](https://makeradvisor.com/tools/rfm95-lora-transceiver-module/) with an home made antenna, we got better results: more than 250 meters with many obstacles in between.



The communication range will really depend on your environment, the LoRa board you’re using and many other variables.

Wrapping Up

You can take this project further and build an off-the-grid monitoring system by adding solar panels and deep sleep to your LoRa sender. The following articles might help you do that:

You may also want to access your sensor readings from anywhere or plot them on a chart:

We hope you’ve found this project interesting. If you’d like to see more projects using LoRa radio, let us know in the comments’ section.

Thanks for reading.