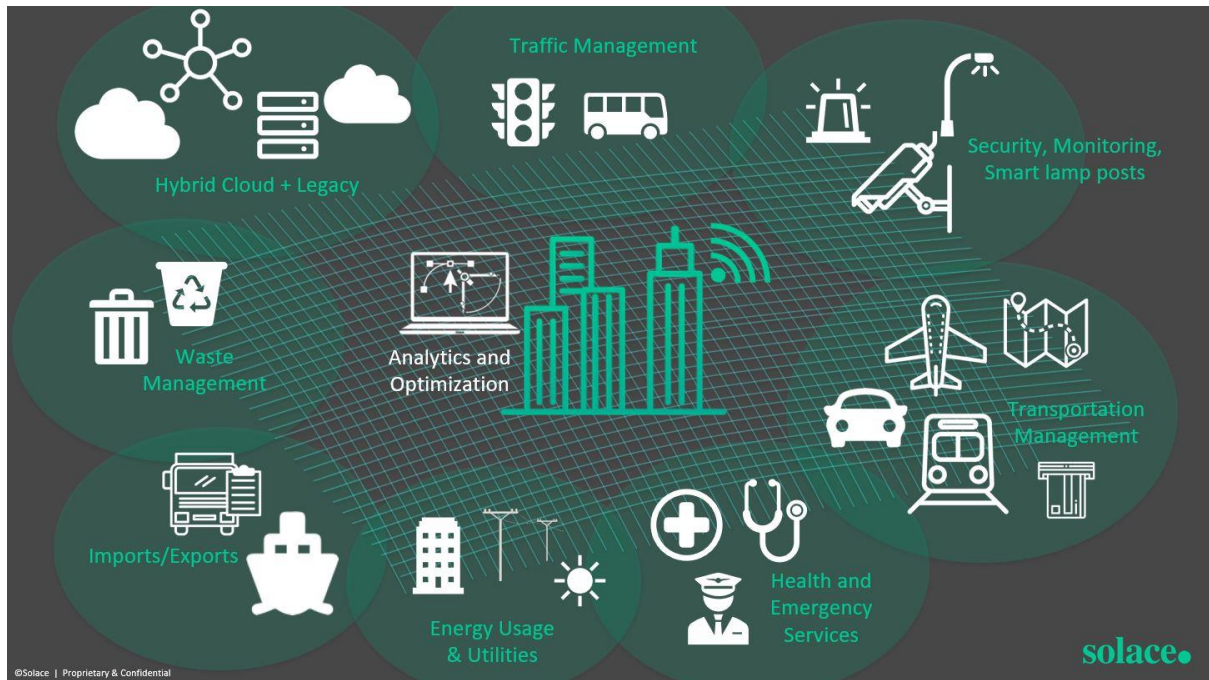


# IOT PHASE 5

## PUBLIC TRANSPORT OPTIMIZATION



Public transport optimization refers to the process of improving the efficiency, effectiveness, and overall quality of public transportation systems. This involves using various strategies and technologies to make public transportation more convenient, reliable, and environmentally sustainable.

### **IOT Projects in Public Transport**

Explore a range of fascinating IOT projects that have been implemented in public transport systems, including smart ticketing, real-time passenger information systems, and automated fleet management.

## **The Importance of Predicting Arrival Time**

Learn why accurately predicting arrival times is crucial for passengers, operators, and city planners. Discover how IOT-powered algorithms and data analysis techniques can help optimize scheduling and reduce delays.

## **Data Collection and Analysis Methods**

### **➤ Sensor Integration**

See how IOT sensors collect real-time data on vehicle locations, passenger counts, and traffic conditions to feed into predictive models.

### **➤ Big Data Analytics**

Explore the use of advanced analytics techniques to process and analyze massive datasets, unlocking valuable insights for optimizing public transport operations.

### **➤ Machine Learning**

Discover how machine learning algorithms can be trained on historical data to recognize patterns and predict arrival times with greater accuracy.

## **Development of Prediction Models**

### **➤ Supervised Learning**

Understand the process of training prediction models using labeled data, enabling them to make accurate forecasts based on past patterns.

#### ➤ **Neural Networks**

Explore how artificial neural networks can leverage the power of parallel processing and hidden layers to create more sophisticated prediction models.

#### ➤ **Time Series Analysis**

Learn how statistical techniques can analyze historical data to identify temporal patterns and seasonality, facilitating precise arrival time predictions.

### **Integration of Prediction Models**

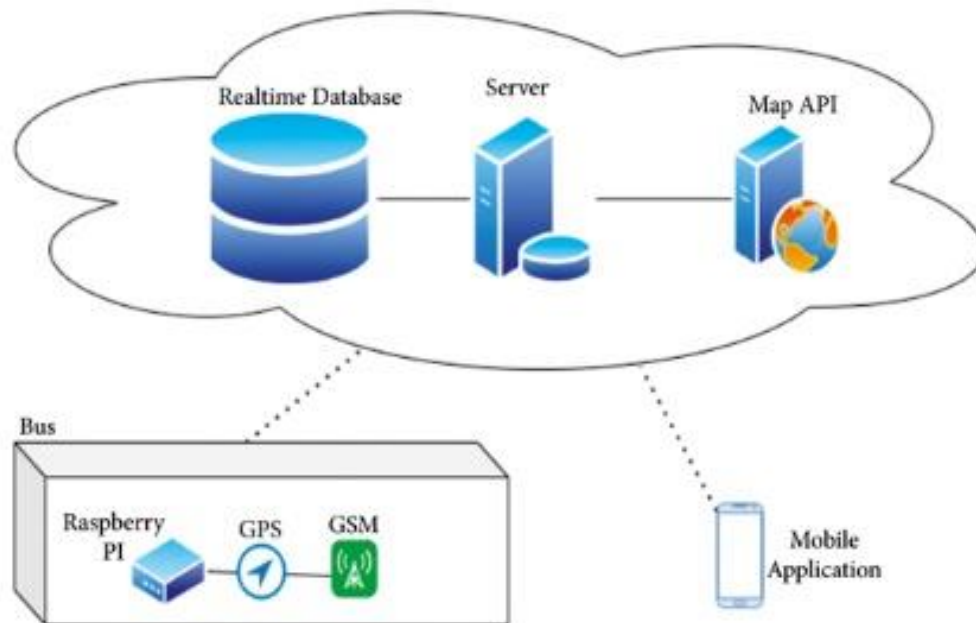
#### ➤ **Real-Time Data Sync**

Discover how prediction models integrate with real-time data feeds from sensors and other sources, enabling dynamic adjustments to arrival time estimates.

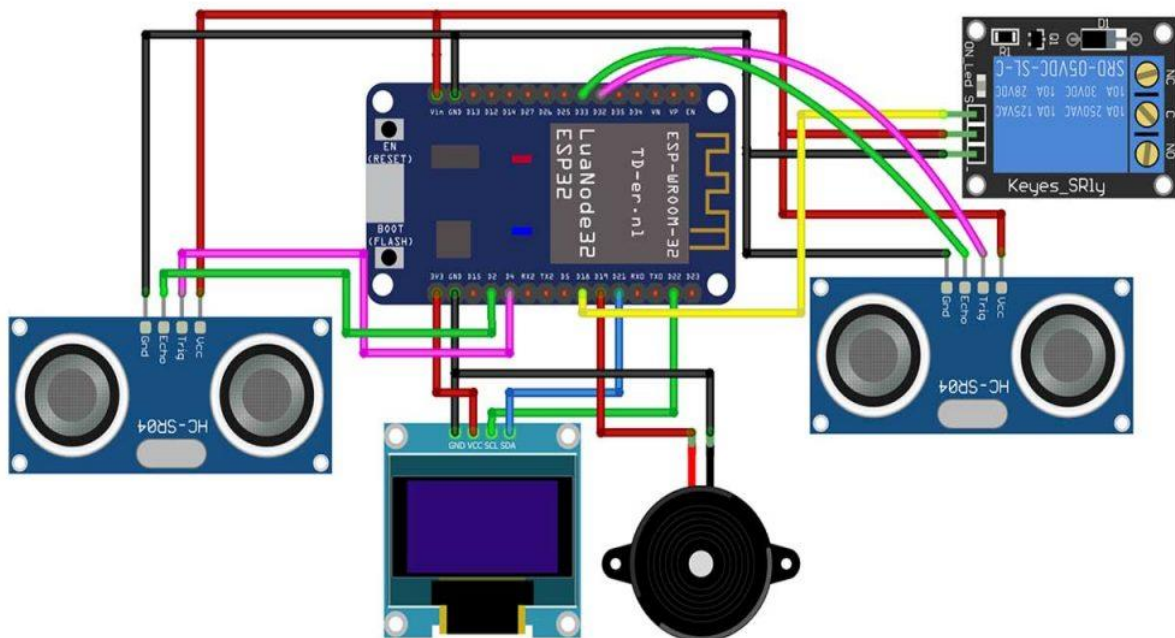
#### ➤ **Passenger Information Systems**

See how arrival time predictions are seamlessly communicated to passengers through digital displays, mobile apps, and public announcements.

## Location Provider Module



## Passenger Count



# Program for Passenger Count

```
import time
```

```
import random
```

```
class Bus:
```

```
    def __init__(self, bus_id):
```

```
        self.bus_id = bus_id
```

```
        self.location = 0
```

```
    def move(self):
```

```
        self.location += random.randint(1, 5)
```

```
    def get_location(self):
```

```
        return self.location
```

```
class Passenger:
```

```
    def __init__(self, passenger_id, destination):
```

```
        self.passenger_id = passenger_id
```

```
        self.destination = destination
```

```
    def request_bus_info(self):
```

```
        # Simulate requesting bus information
```

```
        print(f"Passenger {self.passenger_id}
```

```
        is looking for a bus to {self.destination}.")
```

```
class BusStop:
```

```
    def __init__(self, name, location):
```

```

        self.name = name

        self.location = location

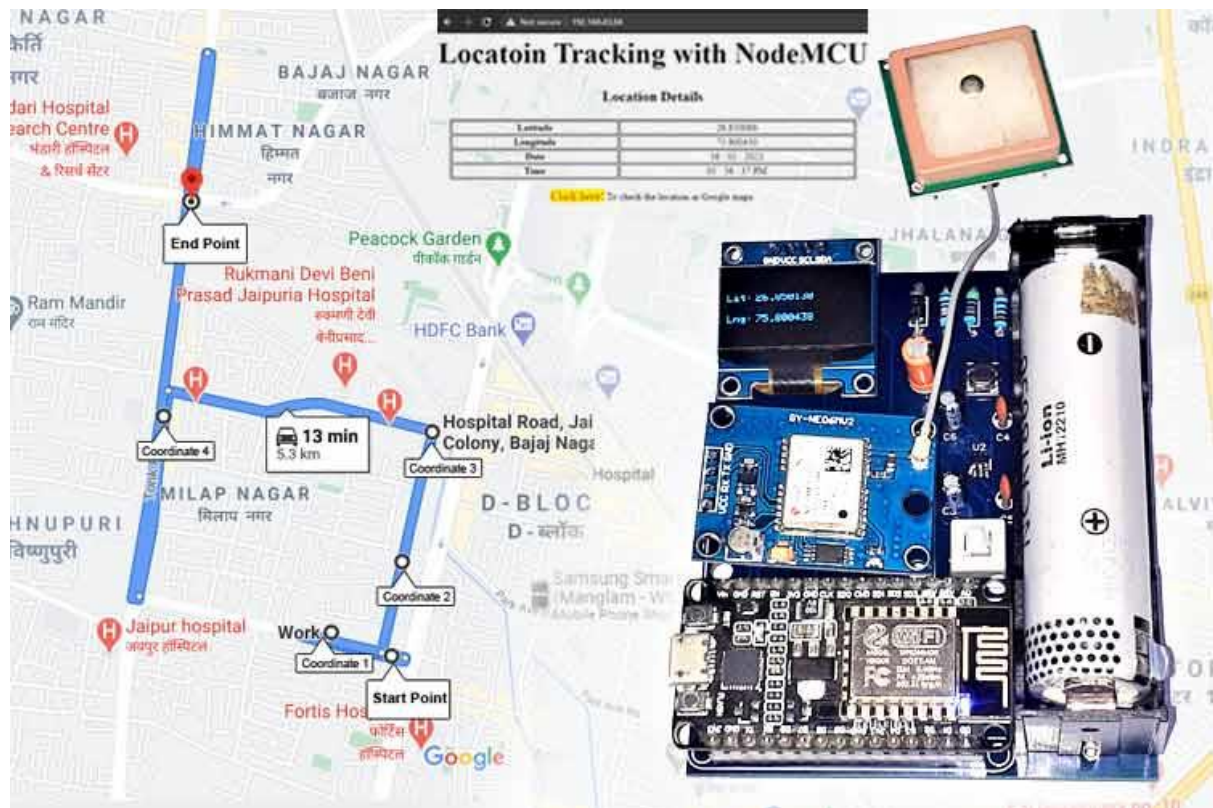
    def is_near(self, bus):
        return abs(bus.get_location() - self.location) <= 2

def main():
    # Create buses and bus stops
    buses = [Bus(1), Bus(2)]
    bus_stops = [BusStop("Stop A", 10), BusStop("Stop
    B", 20)]
    passengers = [Passenger(1, "Stop B"), Passenger(2,
    "Stop A")]
    while True:
        for bus in buses:
            bus.move()
            for passenger in passengers:
                if passenger.request_bus_info():
                    for stop in bus_stops:
                        if stop.name == passenger.destination:
                            if stop.is_near(bus):
                                print(f"Bus {bus.bus_id} is approaching
                                {stop.name} for Passenger
                                {passenger.passenger_id}.")
                                time.sleep(1)
                                # Simulate time passing

if __name__ == "__main__":
    main()

```

## IoT Based GPS Location Tracker using NodeMCU and GPS Module – Save GPS coordinates and view on Google Maps



In this DIY tutorial, we are going to build an IoT-based GPS Location tracker using NEO-6M GPS Module. This Location Tracker Board consists of a NodeMCU, OLED Display Module, NEO-6M GPS Module, and 3.7 to 6V booster circuit. Here, we have also created a simple local webserver to display the location details. This webserver contains a link to directly check the location in Google Maps. We have used PCBWay (<https://www.pcbway.com/>) to provide the PCB boards for this project. In the following sections of the article, we have covered in detail the complete procedure to design, order, and assemble the PCB boards for the IoT-based Location Tracker Board. Previously, we have also built Lora Based GPS Tracker (<https://circuitdigest.com/microcontroller-projects/lora-based-gps-tracker-using-arduino-and-lora-shield>) and Vehicle Tracking and Accident Alert System (<https://circuitdigest.com/microcontrollerprojects/msp430-vehicle-tracking-and-accident-detection-using-vibration-sensor>).

### Components Required for GPS Location Tracker

- NodeMCU ESP8266

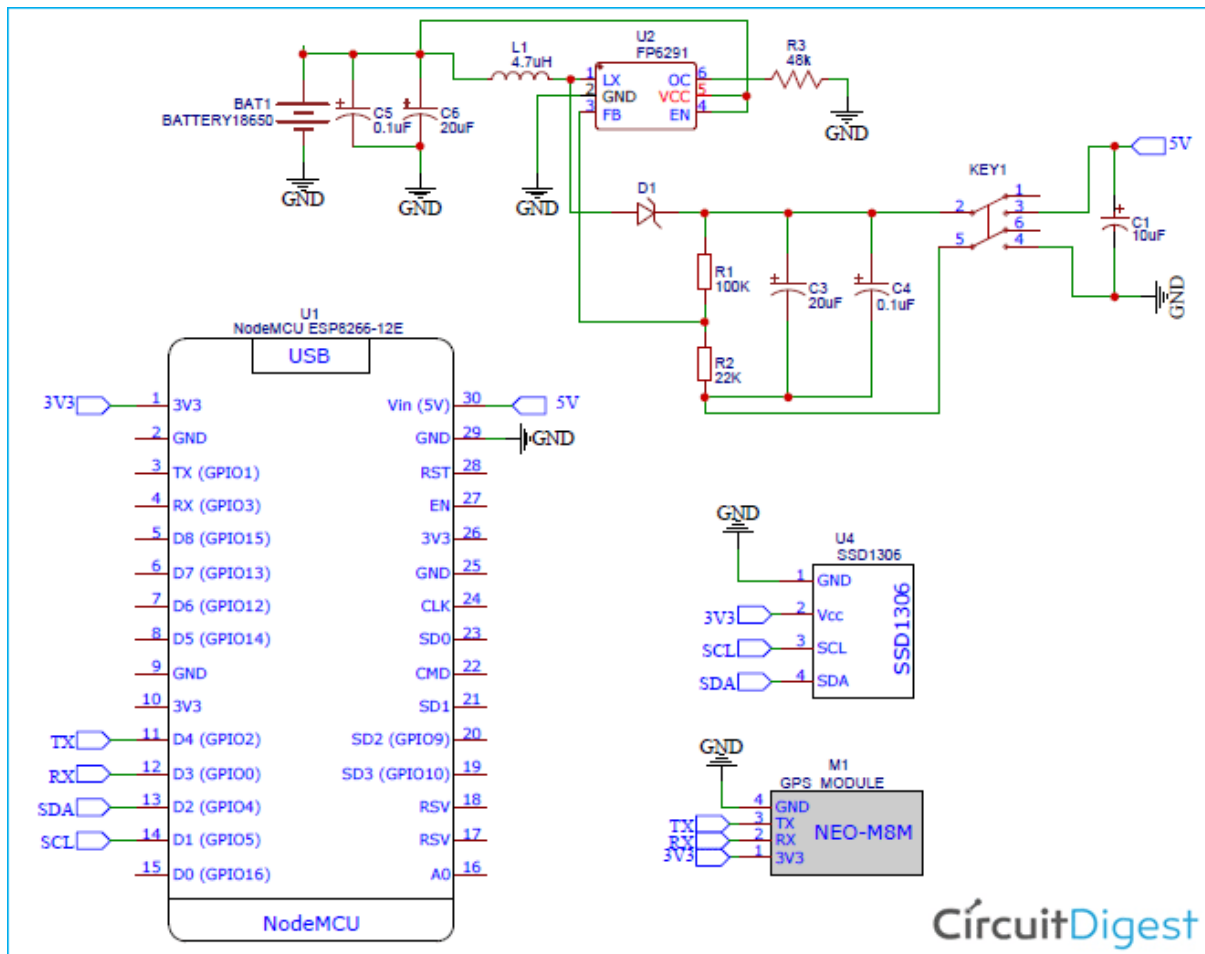
- NEO-6M GPS Module
- OLED Display Module
- FP6291 Boost Converter IC
- 3× Resistor (10k, 100k, 48k)
- 6× Capacitor (2×0.1μf, 1×10μf, 2×20μf)
- 1× Inductor (4.7μH)
- 1× Diode (1N5388BRLG)
- 18650 Lithium Cell
- 18650 Cell Holder
- 6-Pin Push Button Switch

## **IoT Based Location Tracker Circuit Diagram**

The complete circuit diagram for NodeMCU GPS Tracker Board is shown below. The schematic was drawn using EasyEDA. This HAT consists of a NodeMCU with NEO-6M GPS Module, OLED Display Module, and Booster circuit. The booster circuit is designed around a dedicated FP6291 Boost Converter IC to boost the battery voltage from 3.7v to 6V. This location tracking board can be used to track Cars/Bikes/almost anything.

FP6291 IC is a 1 MHz DC-DC Step-Up Booster IC, mainly used in the application, for example, getting stable 5V from 3V battery. You only need few extra components to design a booster circuit with this IC. Here, in this circuit, the Boost Converter circuit gets the input supply through battery terminals (+ and -). This input voltage is then processed by FP6291 IC to give a stable 6V DC supply to the V pin of NodeMCU. The output voltage from this IC can be configured using the potential divider circuit. The formula to calculate the output voltage is:

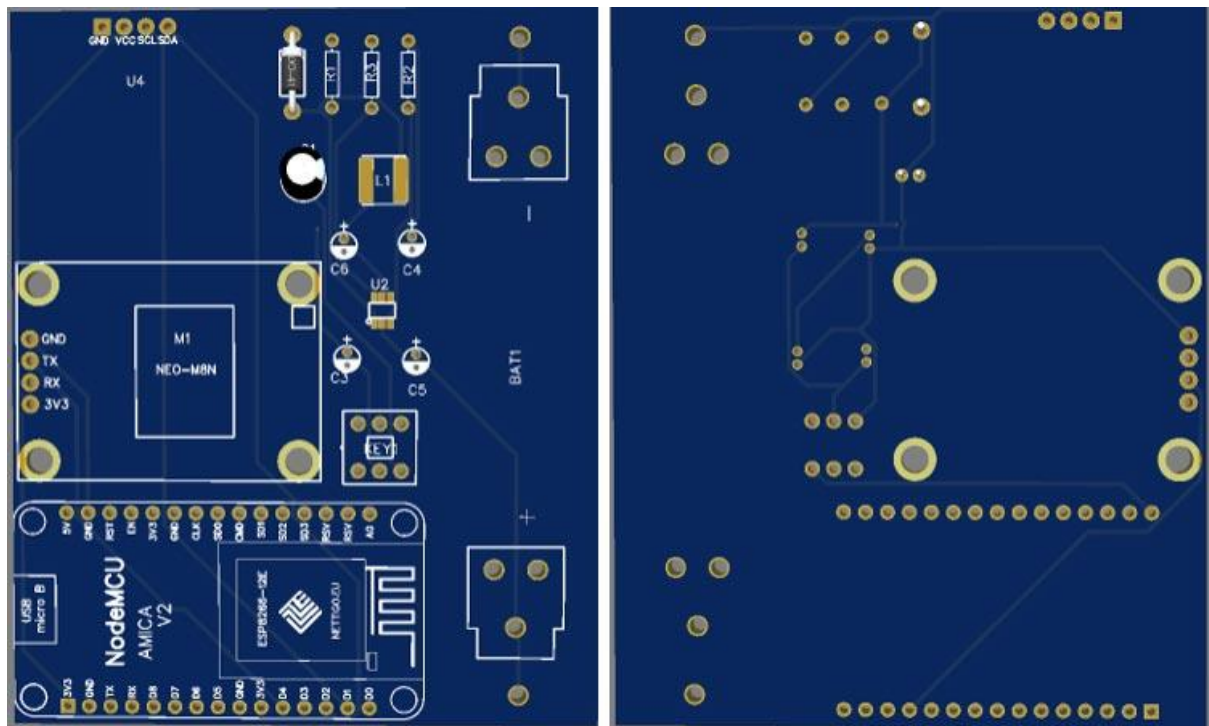




$$V_{OUT} = 0.6(1 + R / R)$$

## Fabricating PCB for NodeMCU GPS Tracker Board

Now that we understand how the schematics works, we can proceed with building the PCB for our project. You can design the PCB using any PCB software of your choice. We have used EasyEDA to fabricate PCB for this project. We have previously used EasyEDA many times and found it very convenient to use compared to other PCB fabricators. Click on the link to check all the PCB projects (<https://circuitdigest.com/diy-pcb-projects>). They also offer a component sourcing service where they have a large stock of electronic components, and users can order their required components along with the PCB order. While designing the circuits and PCBs, you can also make your circuit, and PCB designs public so that other users can copy or edit them and can take benefit from your work. We have also made this NodeMCU GPS Tracker PCB design file and GERBER file public, check the link given below:



Now, that our Design is ready, it is time to get them fabricated using the Gerber file. To get the PCB done is quite easy; simply follow the steps given below.

**Step 1:** Go to <https://www.pcbway.com/> (<https://www.pcbway.com/>), sign up if this is your first time. Then, in the PCB Prototype tab, enter the dimensions of your PCB, the number of layers, and the number of PCB you require



Home PCB Instant Quote PCB Assembly Advanced PCB Product Center Why Us?

**PCB Prototype**

**Instant Quote**  
Full feature prototype PCB custom service at low cost.

**Dimensions** 65.42 x 56.9 mm

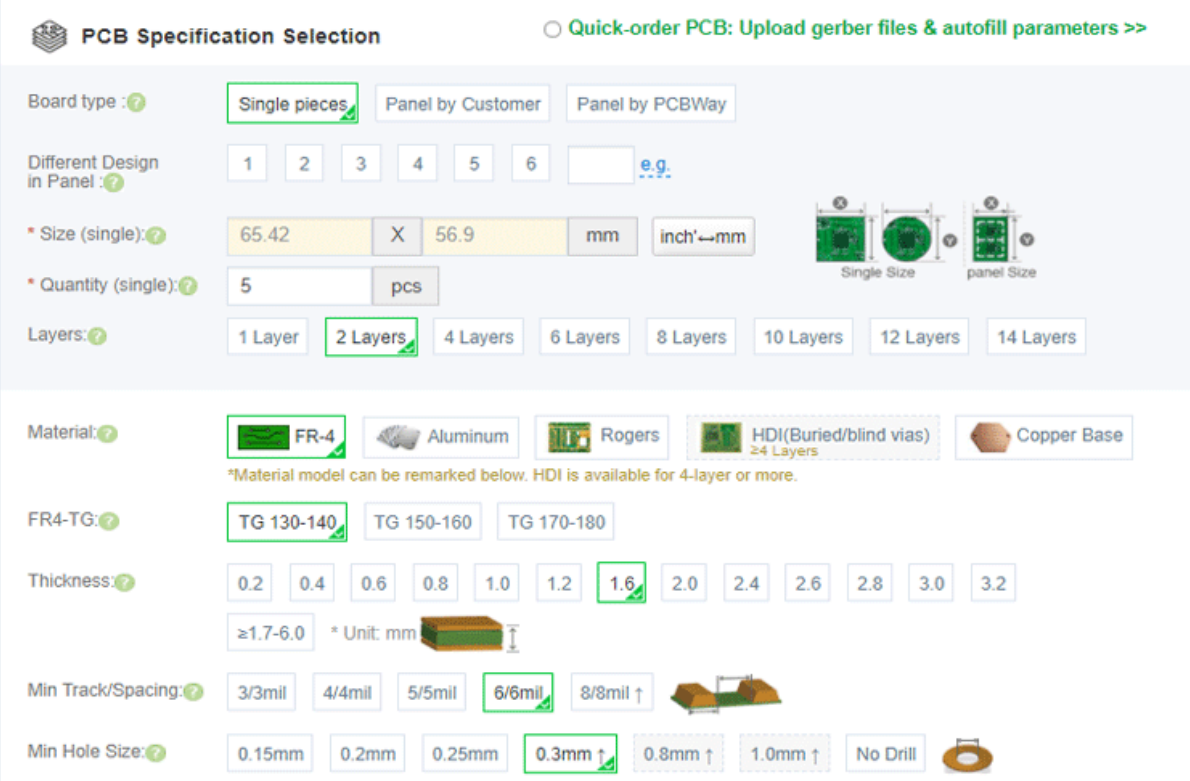
**Quantity** 5

**Layers** 2 Layers

**Thickness** 1.6mm

**Quote Now** Get \$5.00 - Free Prototype Order

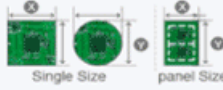
**Step 2:** Proceed by clicking on the 'Quote Now' button. You will be taken to a page where to set a few additional parameters if required like the material used track spacing, etc. but usually, the default values work fine.



**PCB Specification Selection** Quick-order PCB: Upload gerber files & autofill parameters >>

Board type: ☒ Single pieces ☐ Panel by Customer ☐ Panel by PCBWay

Different Design in Panel: 1 2 3 4 5 6  e.g.


\* Size (single): 65.42 X 56.9 mm  inch'↔mm 

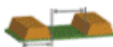
\* Quantity (single): 5 pcs


Layers: 1 Layer ☒ 2 Layers ☐ 4 Layers ☐ 6 Layers ☐ 8 Layers ☐ 10 Layers ☐ 12 Layers ☐ 14 Layers

Material: ☒ FR-4 ☐ Aluminum ☐ Rogers ☐ HDI(Buried/blind vias) ☐ Copper Base  
\*Material model can be remarked below. HDI is available for 4-layer or more.

FR4-TG: ☒ TG 130-140 ☐ TG 150-160 ☐ TG 170-180

Thickness: 0.2 0.4 0.6 0.8 1.0 1.2 ☒ 1.6 2.0 2.4 2.6 2.8 3.0 3.2  
≥1.7-6.0 \* Unit: mm 

Min Track/Spacing: 3/3mil 4/4mil 5/5mil ☒ 6/6mil ☐ 8/8mil 

Min Hole Size: 0.15mm 0.2mm 0.25mm ☒ 0.3mm ☐ 0.8mm ☐ 1.0mm ☐ No Drill 

**Step 3:** The final step is to upload the Gerber file and proceed with the payment. To make sure the process is smooth, PCBWAY verifies if your Gerber file is valid before proceeding with the payment. This way, you can be sure that your PCB is fabricationfriendly and will reach you as committed.



Upload File

 **Attach Your Gerber Files** PCBWay will never disclose your PCB files to any third party!

PCB Order Number: W304109AS1J3 (65.42 mm X 56.9 mm)Layers: 2, Thickness: 1.6, Finished Copper: 1, Solder Mask: Green Silkscreen: White

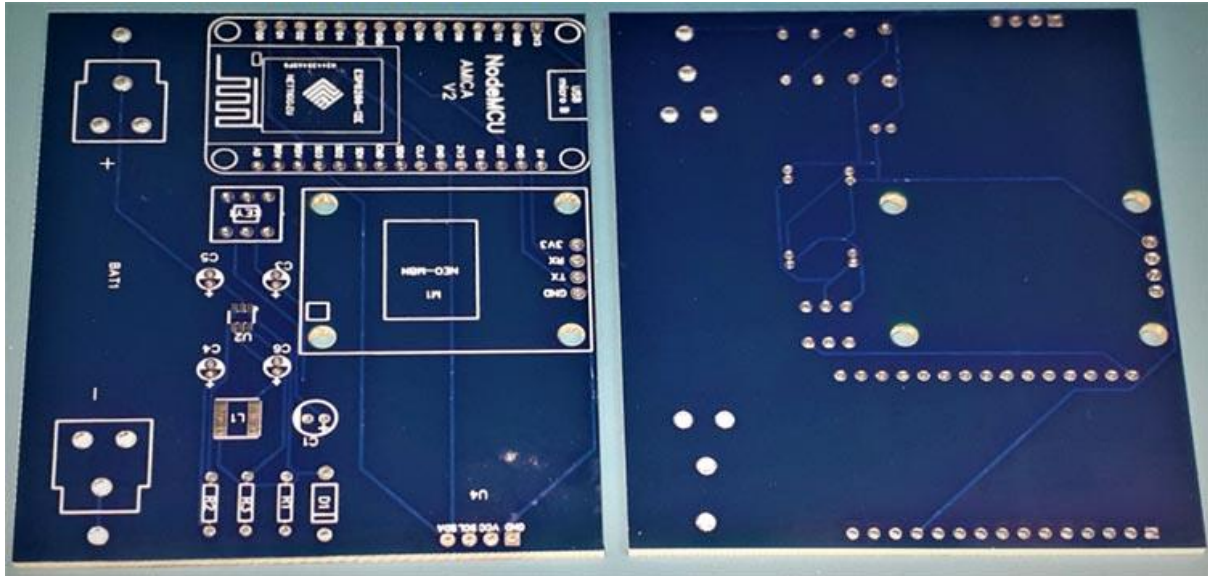
☒ gerber files(RS-274-X) ☐ DXP (.pcb/.pcbdoc) ☐ Others (eagle.brd/.cam etc)

.rar or .zip Maximum 10M(.pcb .pcbdoc .cam .brd and gerber files)

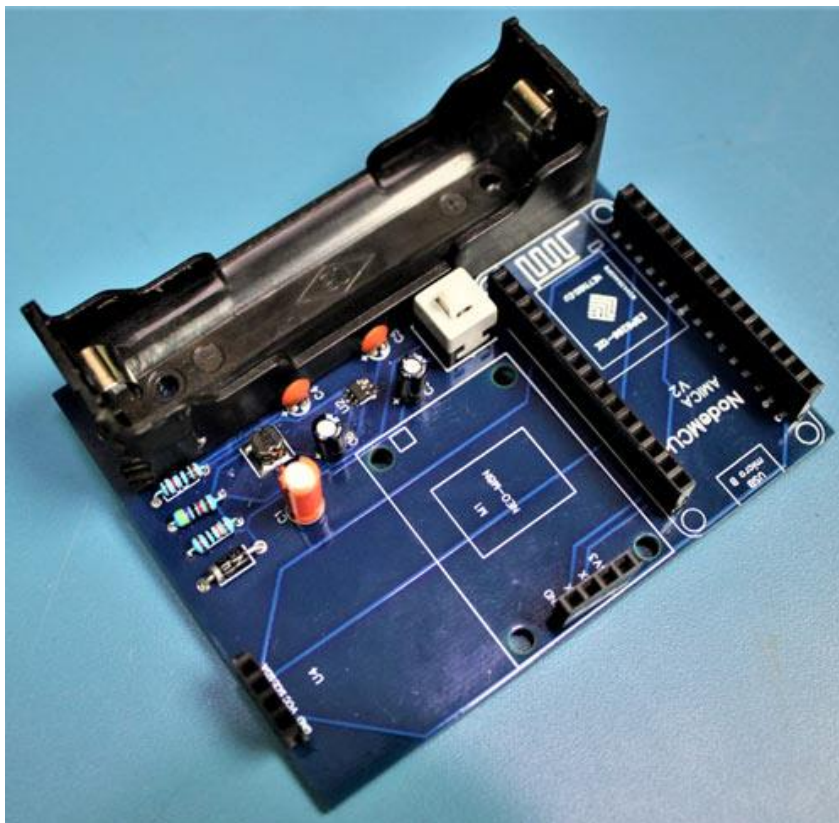
Complete Review in 5-10mins, Uploading in failure? You can skip and email your PCB files to service29@pcbway.com

## Assembling the NodeMCU Location Tracker PCB

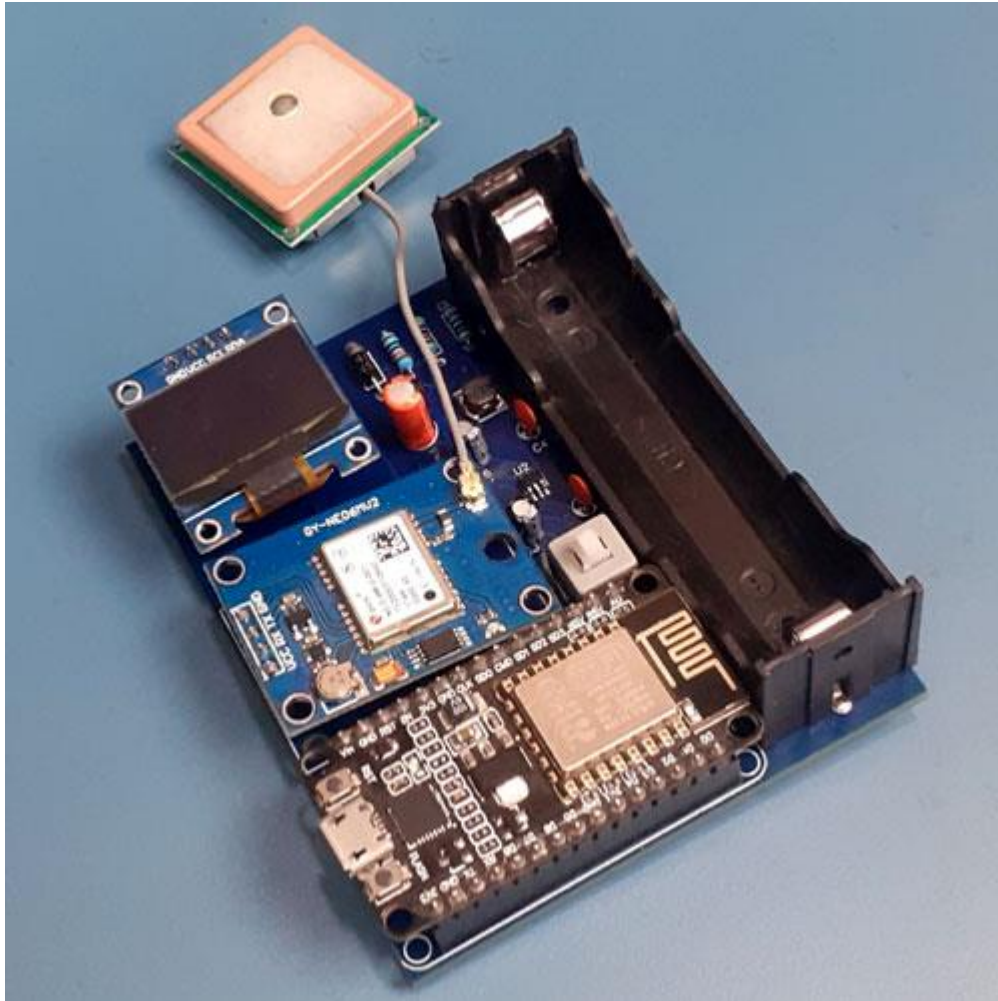
After a few days, we received our PCB in a neat package and the PCB quality was good as always. The top layer and the bottom layer of the board are shown below.



After making sure that the tracks and footprints were correct, I proceeded with assembling the PCB. The completely soldered board looked like as shown in the image below:







## Programming NodeMCU for Getting GPS Data

The complete program can be found at the end of the page, the explanation of the same is as follows. Make sure you have installed the libraries for the GPS module and OLED display before proceeding with the code.

Like always, we begin the program by adding the required libraries. Here, the **Adafruit\_SSD1306.h** library is used for OLED display, and the **TinyGPS++** and **SoftwareSerial** library is used for GPS communication.

```
#include <TinyGPS++.h>
#include <SoftwareSerial.h>
#include <ESP8266WiFi.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
```

Then, define the OLED width and height. In this project, we're using a 128×64 SPI OLED display.

You can change the **SCREEN\_WIDTH** and **SCREEN\_HEIGHT** variables according to your display.

```
#define SCREEN_WIDTH 128  
#define SCREEN_HEIGHT 64
```

Then, create an Adafruit display instance with the width and height defined earlier with the I2C communication protocol.

```
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);
```

Inside the **setup()** function, begin the serial monitor and also initialize the software serial as "ss" for communication with the GPS module and the OLED display with the **begin()** function.

```
void setup()  
{  
  Serial.begin(115200);  
  ss.begin(9600);  
  WiFi.begin(ssid, password);  
  while (WiFi.status() != WL_CONNECTED)  
  {  
    delay(500);  
  }  
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3D for 128x64  
    Serial.println(F("SSD1306 allocation failed"));  
  }  
}
```

Now, inside the **loop()** function, we will use the **TinyGPS** library to extract the latitude, longitude, date, and time data from the string and stored the values in the variables "**lat\_str**", "**lng\_str**", "**date\_str**" and "**time\_str**" respectively. TinyGPS library has an inbuilt function to get the required data from the NMEA string. The code for the same is as follows:

```

if (gps.encode(ss.read()))
{
    if (gps.location.isValid())
    {
        latitude = gps.location.lat();
        lat_str = String(latitude , 6);
        longitude = gps.location.lng();
        lng_str = String(longitude , 6);
    }

    if (gps.date.isValid())
    {
        date_str = "";
        date = gps.date.day();
        month = gps.date.month();
        year = gps.date.year();
        if (date < 10)
            date_str = '0';
        date_str += String(date);
        date_str += " / ";
    }
}

```

```

..... •
..... •

```

Once a client is connected, the webserver needs to send the response to the client. Here, all the HTML code is embedded into a variable called “s”. Then this variable is printed using the ***client.print(s)*** to send all embedded HTML code to the webpage. The latitude, longitude, date, and time values are updated once the data is received from the GPS module.

```

String s = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n <!DOCTYPE html> <html> <head> <title>Location Tracking with NodeMCU</title> <style>";
    s += "a:link {background-color: YELLOW;text-decoration: none;}";
    s += "table, th, td {border: 1px solid black;} </style> </head> <body> <h1 style=";
    s += "font-size:300%;";
    s += " ALIGN=CENTER> Locatoin Tracking with NodeMCU</h1>";
    s += "<p ALIGN=CENTER font-size:150%;\"";

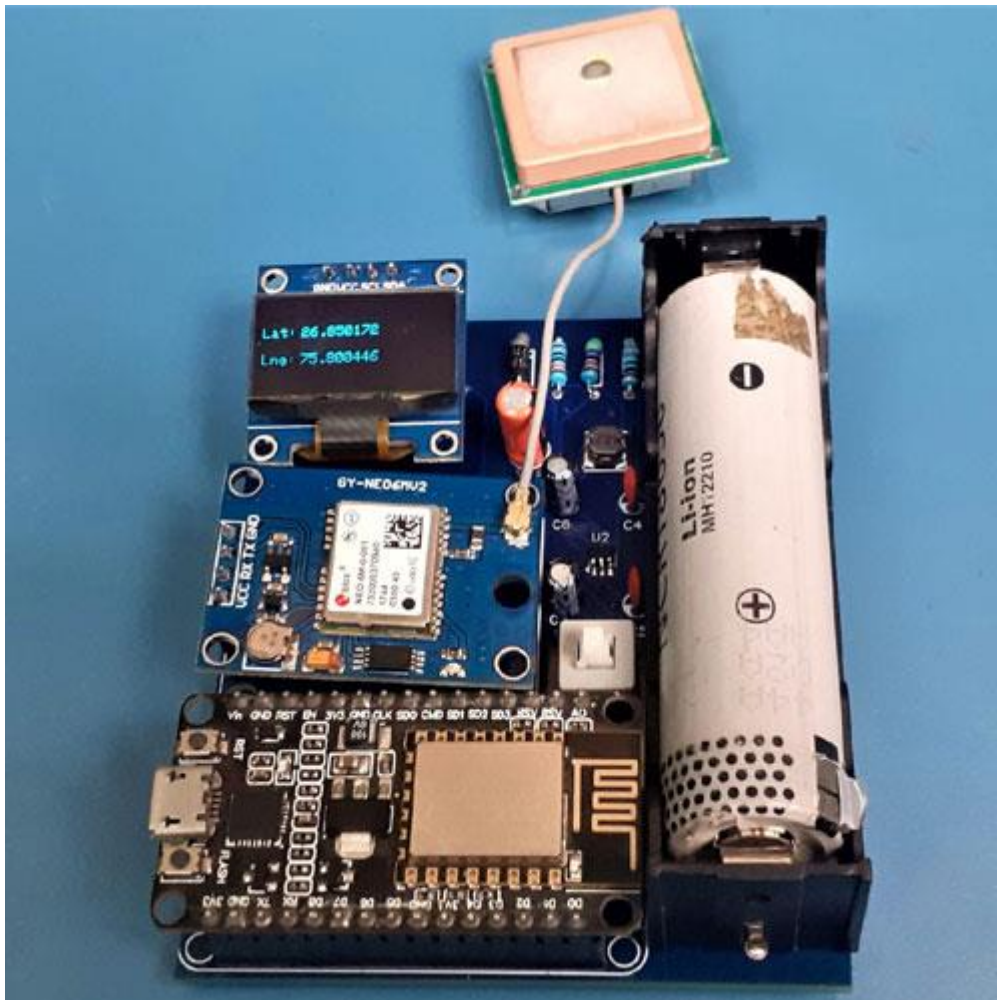
```

```
s += "> <b>Location Details</b></p> <table ALIGN=CENTER style=";
s += "width:50%";
s += "> <tr> <th>Latitude</th>";
s += "<td ALIGN=CENTER >";
s += lat_str;
```

..... •

..... •

## NodeMCU GPS Tracker Working



Once the hardware and program are ready, upload the code to the NodeMCU Board and power the setup. If the LED on the GPS module is blinking (It may take a couple of minutes), this means that the module is looking for a satellite connection to get co-ordinates, if not take the GPS module to the open area. In the serial monitor, you will be able to see the local IP address of your NodeMCU.



Now, navigate to the browser and type in the NodeMCU IP address and press ENTER. You will be able to see the Location, Date, and Time displayed as below:

[←](#) [→](#) [↻](#) [⚠](#) Not secure | 192.168.43.84

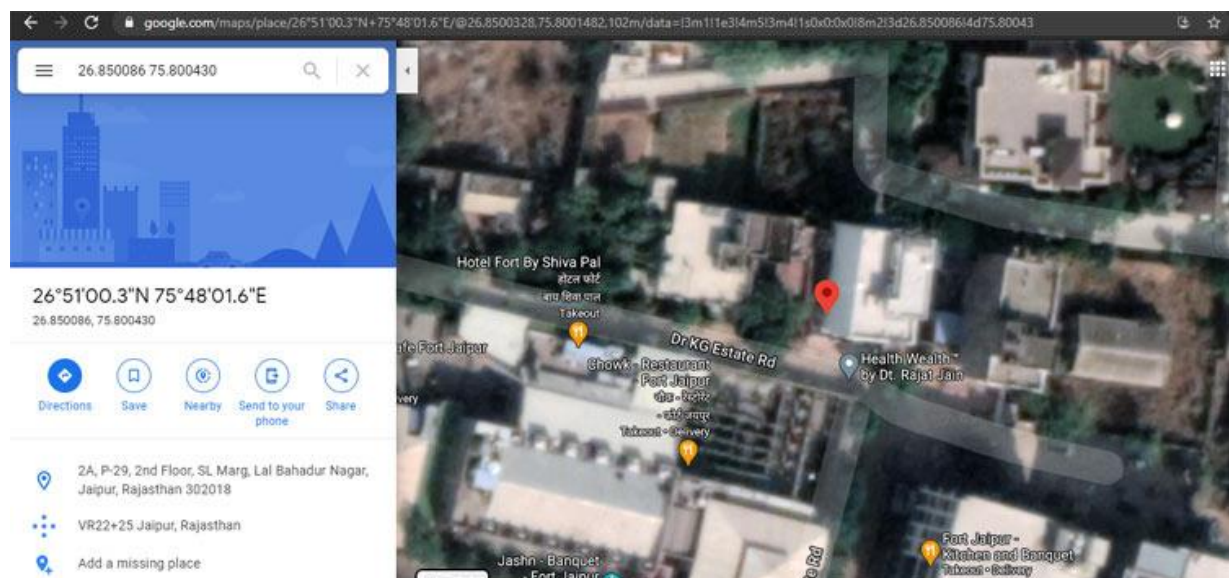
# Locatoin Tracking with NodeMCU

## Location Details

Latitude	26.850086
Longitude	75.800430
Date	16 / 02 / 2021
Time	05 : 56 : 37 PM

[Click here!](#) To check the location in Google maps.

When you click on “Click here”, it will open the location in Google Maps like below:



The complete working can also be found in the video given at the bottom of this page. Hope you understood the tutorial and enjoyed building something useful with it. If you have any doubts, you can leave them in the comment section below.

## Program:

```
#include <TinyGPS++.h>
```

```
#include <SoftwareSerial.h>

#include <ESP8266WiFi.h>

#include <Wire.h>

#include <Adafruit_GFX.h>

#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128 // OLED display width, in pixels

#define SCREEN_HEIGHT 64 // OLED display height, in pixels

// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

TinyGPSPlus gps; // The TinyGPS++ object

SoftwareSerial ss(2, 0); // The serial connection to the GPS device

const char* ssid = "Galaxy-M20";

const char* password = "ac312124";

float latitude , longitude;

int year , month , date, hour , minute , second;

String date_str , time_str , lat_str , lng_str;

int pm;

//int numreadings = 10;

String latarray[20];

String lngarray[20];

//String current_latarray[10], current_lngarray[10], previous_latarray[10],
previous_lngarray[10] ;

unsigned int i = 0;
```

```
const unsigned long Interval = 13000;

unsigned long previousTime = 0;

WiFiServer server(80);

void setup()

{

    Serial.begin(115200);

    ss.begin(9600);

    delay(2000); // Pause for 2 seconds

    Serial.println();

    Serial.print("Connecting to ");

    Serial.println(ssid);

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED)

    {

        delay(500);

        Serial.print(".");

    }

    Serial.println("");

    Serial.println("WiFi connected");

    server.begin();

    Serial.println("Server started");

    // Print the IP address
```

```
Serial.println(WiFi.localIP());

if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3D for 128x64

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

  for(;;);

}

}

void loop()

{

  unsigned long currentTime = millis();

  display.clearDisplay();

  display.setTextSize(1);

  display.setTextColor(WHITE);

  while (ss.available() > 0)

    if (gps.encode(ss.read()))

    {

      if (currentTime - previousTime >= Interval) {

        if (gps.location.isValid())

        {

          //Serial.print("Getting Data");

          latitude = gps.location.lat();

          lat_str = String(latitude , 6);

          longitude = gps.location.lng();
```

```
    lng_str = String(longitude , 6);

    latarray[i] = lat_str;

    lngarray[i]= lng_str;

    i++;

    Serial.print(i);

    if (i>=20)

    // {

        i=0; //reset to beginning of array, so you don't try to save readings
outside of the bounds of the array

    // }

    Serial.println("Latitude:");

    Serial.println(latarray[i]);

    Serial.println("Longitude:");

    Serial.println(lngarray[i]);

    Serial.println();

    display.setCursor(0, 20);

    display.println("Lat:");

    display.setCursor(27, 20);

    display.println(lat_str);

    display.setCursor(0, 40);

    display.println("Lng:");

    display.setCursor(27, 40);

    display.println(lng_str);
```

```
        display.display();

    }

    previousTime = currentTime;

}

//}

if (gps.date.isValid())

{

    //Serial.print("Getting Time");

    date_str = "";

    date = gps.date.day();

    month = gps.date.month();

    year = gps.date.year();

    if (date < 10)

        date_str = '0';

    date_str += String(date);

    date_str += " / ";

    if (month < 10)

        date_str += '0';

    date_str += String(month);

    date_str += " / ";

    if (year < 10)

        date_str += '0';
```

```
        date_str += String(year);

    }

    if (gps.time.isValid())

    {

        time_str = "";

        hour = gps.time.hour();

        minute = gps.time.minute();

        second = gps.time.second();

        minute = (minute + 30);

        if (minute > 59)

        {

            minute = minute - 60;

            hour = hour + 1;

        }

        hour = (hour + 5) ;

        if (hour > 23)

            hour = hour - 24;

        if (hour >= 12)

            pm = 1;

        else

            pm = 0;

        hour = hour % 12;
```

```
        if (hour < 10)

            time_str = '0';

            time_str += String(hour);

            time_str += " : ";

            if (minute < 10)

                time_str += '0';

                time_str += String(minute);

                time_str += " : ";

                if (second < 10)

                    time_str += '0';

                    time_str += String(second);

                if (pm == 1)

                    time_str += " PM ";

                else

                    time_str += " AM ";

            }

        }

// Check if a client has connected

WiFiClient client = server.available();

if (!client)

{

    return;
```



```

}

// Prepare the response

String s = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n <!DOCTYPE html>
<html> <head> <title>Location Tracking with NodeMCU</title> <style>";

s += "a:link {background-color: YELLOW;text-decoration: none;}";

s += "table, th, td {border: 1px solid black;} </style> </head> <body>
<h1 style="";

s += "font-size:300%;";

s += " ALIGN=CENTER> Locatoin Tracking with NodeMCU</h1>";

s += "<p ALIGN=CENTER style=\"\"font-size:150%;\"\"";

s += "> <b>Location Details</b></p> <table ALIGN=CENTER style="";

s += "width:50%";

s += "> <tr> <th>Latitude</th>";

s += "<td ALIGN=CENTER >";

s += lat_str;

s += "</td> </tr> <tr> <th>Longitude</th> <td ALIGN=CENTER >";

s += lng_str;

s += "</td> </tr> <tr> <th>Date</th> <td ALIGN=CENTER >";

s += date_str;

s += "</td></tr> <tr> <th>Time</th> <td ALIGN=CENTER >";

s += time_str;

s += "</td> </tr> </table> ";

if (gps.location.isValid())

```

```

{

    // s += "<p align=center><a style=\""color:RED;font-size:125%\""
href="http://maps.google.com/maps?&z=15&mrt=yp&t=k&q=";

    s += "<p align=center><a style=\""color:RED;font-size:125%\""
href="https://www.google.com/maps/dir/";

    //https://www.google.com/maps/dir/26.8172985,75.8286322/26.8181889,75.830
...;

    for (int j=0; j<20; j++)

    {

        s += latarray[j];

        s += ",";

        s += lngarray[j];

        if (j<10)

            s += "/";

        }

        s += "" target=""_top"">Click here!</a> To check the location in Google
maps.</p>";

    }

    s += "</body> </html> \n";

    client.print(s);

    delay(200);

}

```