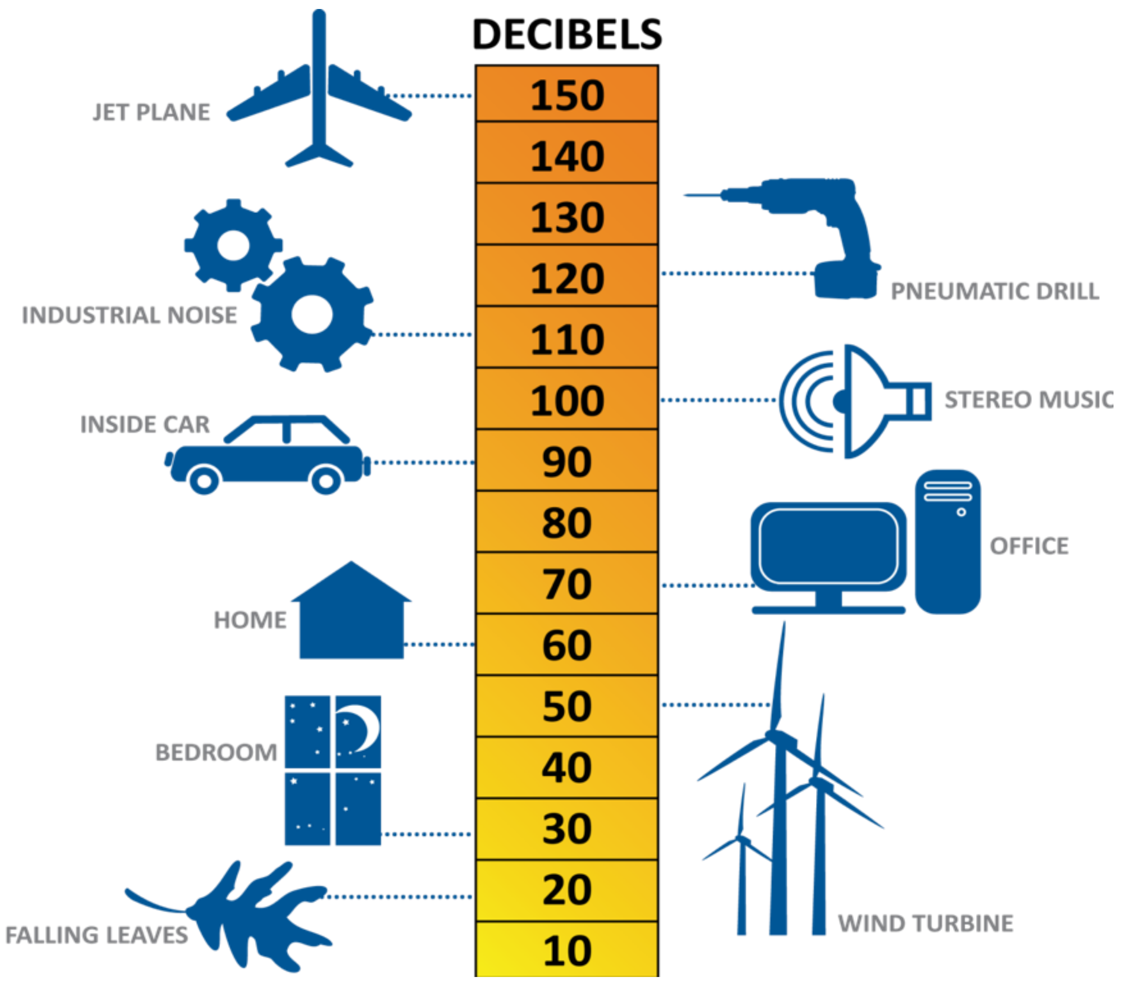
NOISE POLLUTION MONITORING USING IOT PHASE-5

INTRODUCTION

Noise pollution is one of the major environmental problems that affects the health and wellbeing of humans and animals. It can cause hearing loss, stress, hypertension, sleep disturbance, and reduced productivity. Therefore, it is important to monitor and control the noise level in different environments, such as urban areas, industrial zones, schools, hospitals, and airports.

One of the possible solutions to monitor noise pollution is to use the Internet of Things (IoT) technology. IoT is a network of interconnected devices that can collect, process, and transmit data over the internet. By using IoT devices, such as microphones, microcontrollers, and WiFi modules, we can measure the noise level in real time and send the data to a cloud server for analysis and visualization. We can also generate alerts and notifications when the noise level exceeds a certain threshold or violates the regulations.

In this paper, we present a noise pollution monitoring system using IoT that can detect and report the noise level in different locations.



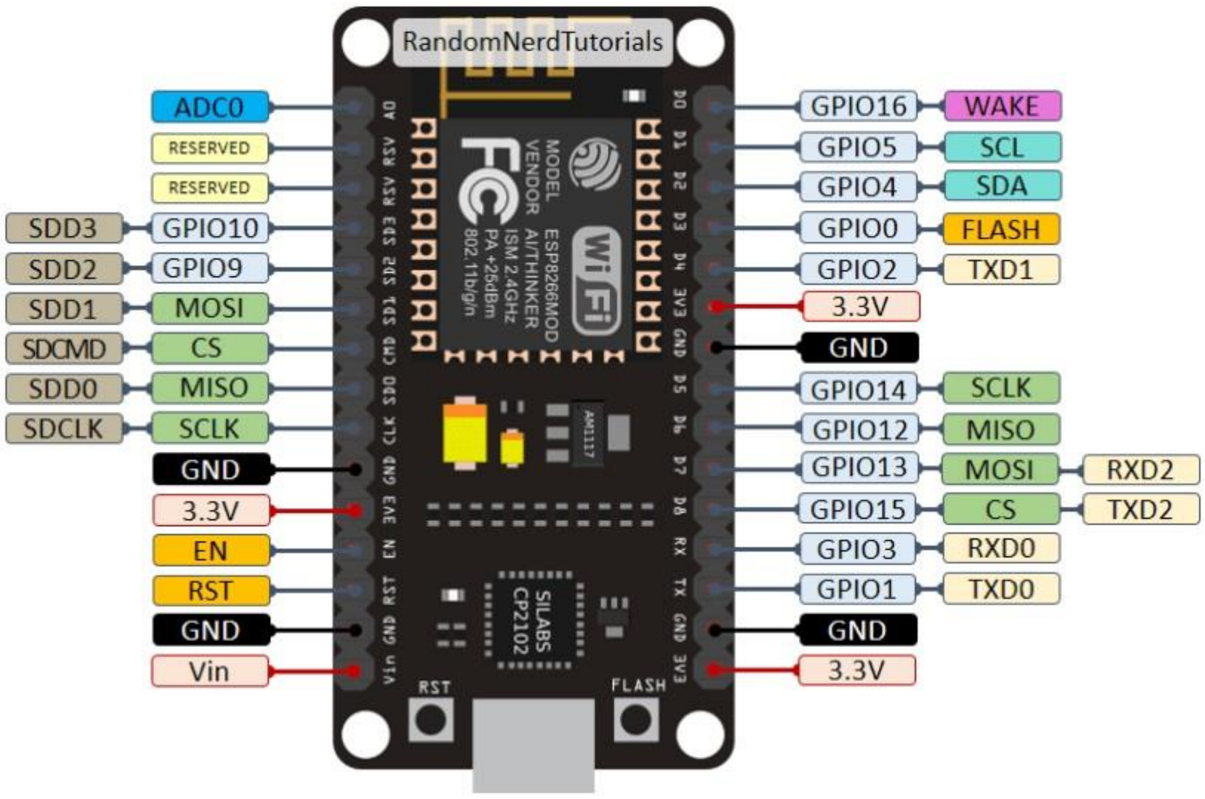
COMPONENTS REQUIRED

- ESP8266 NodeMCU Board
- Microphone sensor
- 16*2 LCD Module
- Breadboard
- Connecting wires

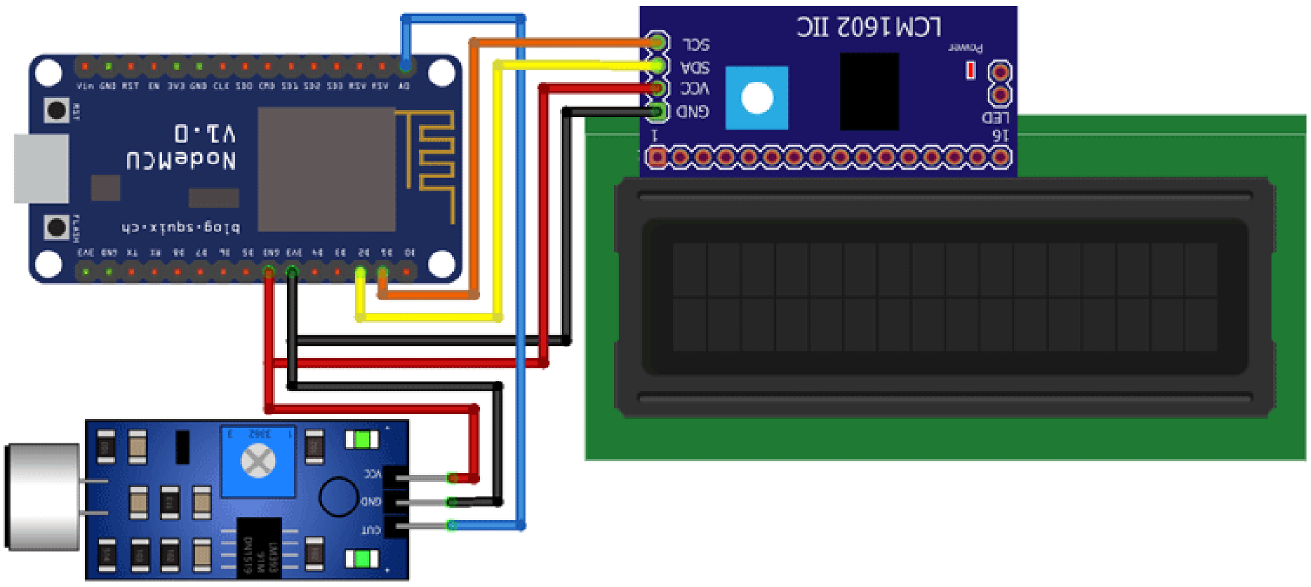
ESP8266

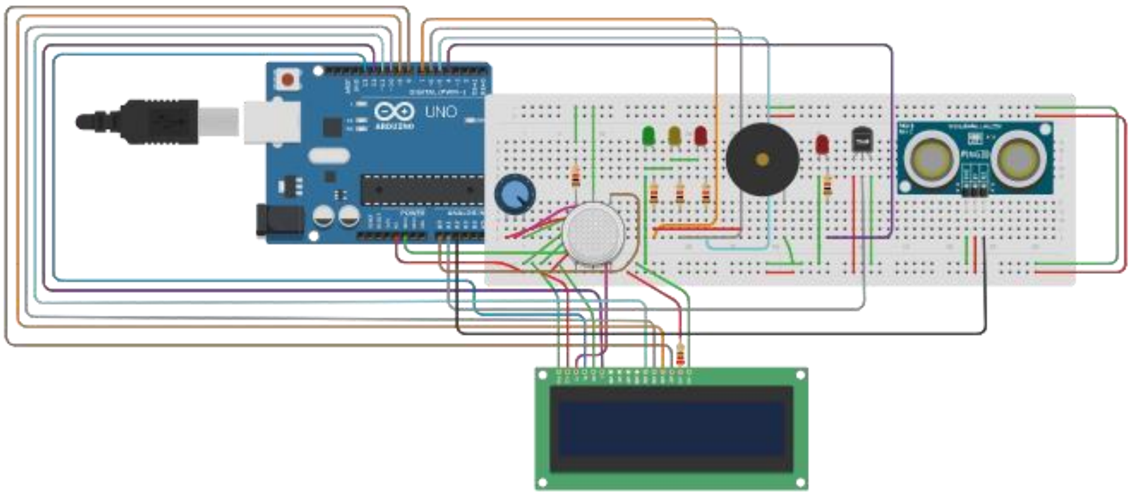
The **ESP8266** is a low-cost Wi-Fi microchip with built-in TCP/IP networking software and microcontroller capability. It is designed and manufactured by Espressif Systems in Shanghai, China. The chip contains the crucial elements of a computer, including CPU, RAM, networking (WiFi), and even a modern operating system and SDK.

PIN DIAGRAM



CIRCUIT DIAGRAM





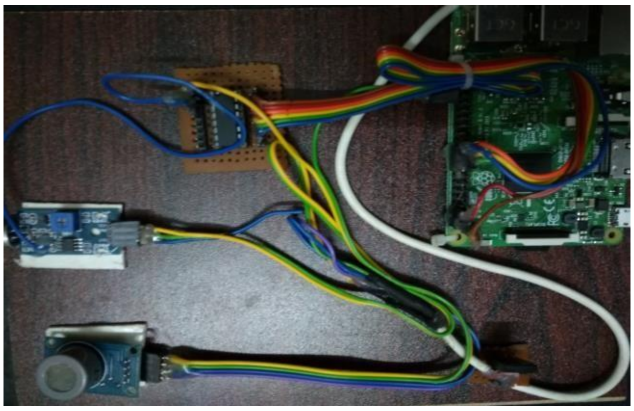
PROGRAM

```
#define BLYNK_PRINT Serial
#include <ESP8266WiFi.h>
```

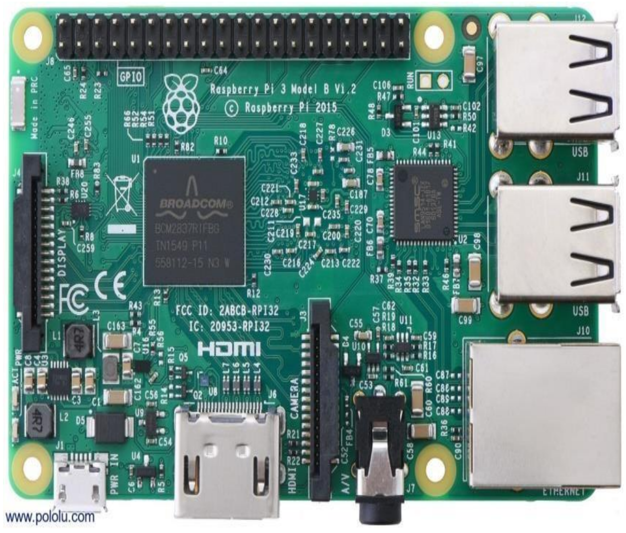
```
#include <BlynkSimpleEsp8266.h> #include <LiquidCrystal_I2C.h>
#define SENSOR_PIN A0
LiquidCrystal_I2C lcd(0x3F, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
const int sampleWindow = 50; unsigned
int sample;
int db;
char auth[] = "IEu1xT825VDt6hNfrcFgdJ6InJ1QUfsA";
char ssid[] = "realme 6";
char pass[] = "evil@zeb";
BLYNK_READ(V0)
{
  Blynk.virtualWrite(V0, db);
}
void setup() {
  pinMode (SENSOR_PIN, INPUT);
lcd.begin(16, 2);   lcd.backlight();
lcd.clear();
  Blynk.begin(auth, ssid, pass);
}
void loop() {
Blynk.run();
  unsigned long startMillis = millis();  // Start of sample window
float peakToPeak = 0;  // peak-to-peak level   unsigned int
signalMax = 0;  //minimum value   unsigned int signalMin =
1024;  //maximum value
  // collect data for 50 mS
  while (millis() - startMillis < sampleWindow)
  {
    sample = analogRead(SENSOR_PIN);  //get reading from microphone
if (sample < 1024)  // toss out spurious readings
    {
      if (sample > signalMax)
      {
        signalMax = sample;  // save just the max levels
      }
      else if (sample < signalMin)
      {
        signalMin = sample;  // save just the min levels
      }
    }
  }
  peakToPeak = signalMax - signalMin;  // max - min = peak-peak amplitude
Serial.println(peakToPeak);
  db = map(peakToPeak, 20, 900, 49.5, 90);  //calibrate for deciBels
lcd.setCursor(0, 0);   lcd.print("Loudness: ");
```

```
  lcd.print(db);
lcd.print("dB");   if
(db <= 50)
  {
    lcd.setCursor(0, 1);
    lcd.print("Level: Quite");
  }
  else if (db > 50 && db < 75)
  {
    lcd.setCursor(0, 1);
    lcd.print("Level: Moderate");
  }
  else if (db >= 75)
  {
    lcd.setCursor(0, 1);
    lcd.print("Level: High");
  }
  delay(600);
  lcd.clear(); }
```

RASPBERRY PI



Raspberry Pi 3:



The Raspberry Pi is a series of credit card- sized single board computer developed in U.K by the Raspberry Pi foundation with the intention of promoting the teaching of basic computer science. The Raspberry Pi 3 will cost the same as its predecessor, but feature much more powerful hardware. Bluetooth will be built into the board for the first time, and is powered by a Quad Core Broadcom BCM2837 64bit ARMv8 processor. The original Raspberry Pi includes video core IV GPU and has capability of 1GB RAM the system has Micro Storage Digital (SD) sockets for boot media and persistent storage. The device is powered by a 5V micro USB supply. Exactly how much current (mA) the Raspberry Pi requires is dependent on what you connect to it. All of the connectors are in the same place and have the same functionality, and the board can still be run from a 5V micro-USB power adapter [1][2].

The Raspberry Pi Model B is equipped with four USB2.0 ports. These are connected to the LAN9512 combo hub/Ethernet chip IC3, which is itself a USB device connected to the double upstream USB port on BCM2837.

We use an 8GB class 4 SD card ideally preinstalled with Raspbian (or) NOOBS. We can buy a card with NOOBS pre-installed, or you can download it for free on Raspberrypi.org.

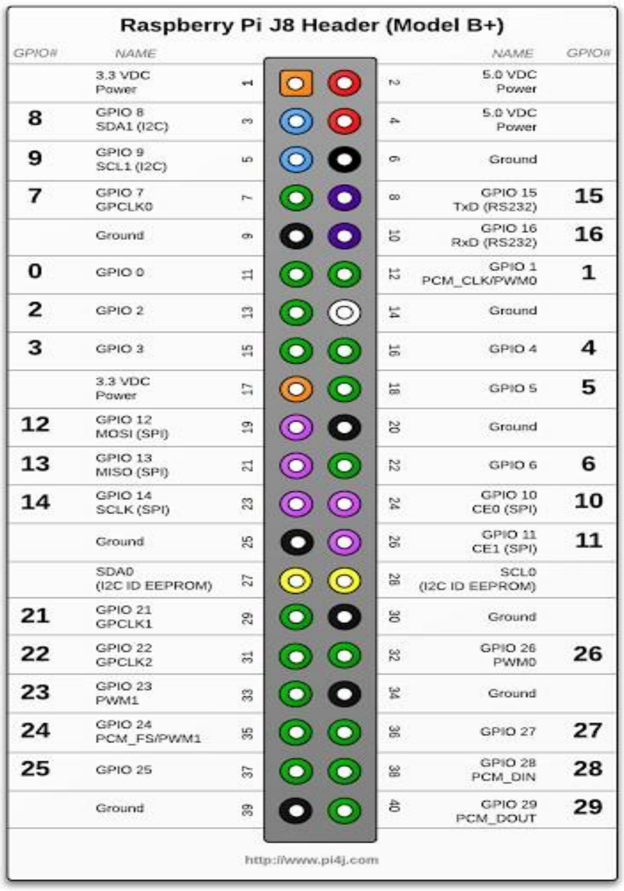
3.2 MQ7 Sensor

This is a simple-to-use Carbon Monoxide (CO) sensor, suitable for sensing CO concentrations in the air. The MQ-7 can detect CO-gas concentrations anywhere from 20 to 2000ppm. This sensor has a high sensitivity and fast response time. The sensor's output is an analog resistance. The circuit connection is very simple as it has only 3 pins. all you need to do is power the heater coil with 5V, add a load resistance, and connect the output to an ADC. It has good sensitivity to combustible gas in wide range, high sensitivity to natural gas, long life and low cost [3][4].

3.3 Sound Detector

Using sound sensor we can detect ambient sound. This board along with the microphone, has a small built-in amplifier (integrated circuit LM386). The connection scheme is very clean, composed of only 3 pins: Vcc, GND and S (signal). In the middle of the plate, there is a potentiometer for sensitivity adjustment. The board works with 5V voltage, and the signal pin should be connected preferably to an analog port, since the generated signal is variable, and thus we can see the different levels of noise picked up by the microphone.

IV GPIO PIN DIAGRAM



Raspberry Pi J8 Header (Model B+)

GPIO#	NAME			NAME	GPIO#
	3.3 VDC Power	1	2	5.0 VDC Power	
8	GPIO 8 SDA1 (I2C)	3	4	5.0 VDC Power	
9	GPIO 9 SCL1 (I2C)	5	6	Ground	
7	GPIO 7 GPCLK0	7	8	GPIO 15 TxD (RS232)	15
	Ground	9	10	GPIO 16 RxD (RS232)	16
0	GPIO 0	11	12	GPIO 1 PCM_CLK/PWM0	1
2	GPIO 2	13	14	Ground	
3	GPIO 3	15	16	GPIO 4	4
	3.3 VDC Power	17	18	GPIO 5	5
12	GPIO 12 MOSI (SPI)	19	20	Ground	
13	GPIO 13 MISO (SPI)	21	22	GPIO 6	6
14	GPIO 14 SCLK (SPI)	23	24	GPIO 10 CE0 (SPI)	10
	Ground	25	26	GPIO 11 CE1 (SPI)	11
	SDA0 (I2C ID EEPROM)	27	28	SCL0 (I2C ID EEPROM)	
21	GPIO 21 GPCLK1	29	30	Ground	
22	GPIO 22 GPCLK2	31	32	GPIO 26 PWM0	26
23	GPIO 23 PWM1	33	34	Ground	
24	GPIO 24 PCM_FS/PWM1	35	36	GPIO 27	27
25	GPIO 25	37	38	GPIO 28 PCM_DIN	28
	Ground	39	40	GPIO 29 PCM_DOUT	29

http://www.pi4j.com

Fig 8: GPIO Pin diagram

One powerful feature of the Raspberry Pi is the row of GPIO (general purpose input/output) pins along the edge of the board, next to the yellow video out socket. These pins are a physical interface between the Pi and the outside world. At the simplest level, you can think of them as switches that you can turn on or off (input) or that the Pi can turn on or off (output). In our system, 11th pin is treated as clock, 9th pin is treated as MISO (master input slave output), 10th pin is treated as MOSI(master output slave input) and 8th pin is treated as chip select.

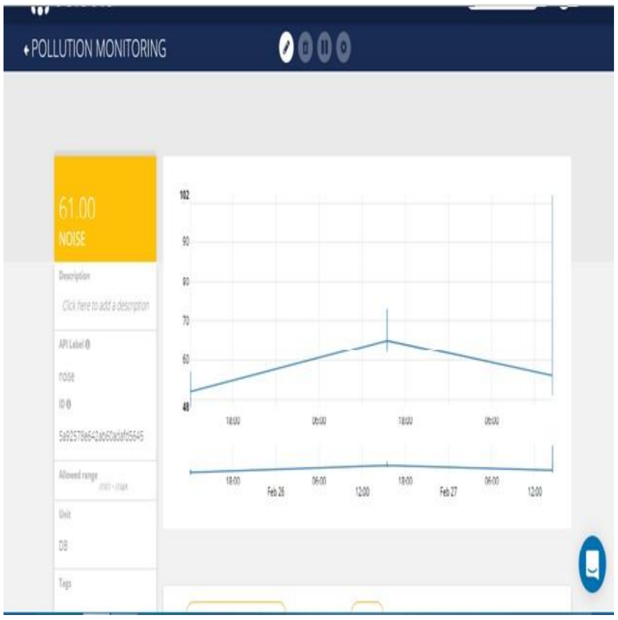
V TESTING AND OUTPUT OF THE PROJECT

Our Raspberry Pi should be equipped with raspbian os in order to run our project. We have to make sure that the raspberry pi and the system to which the raspberry pi is connected via Wi-Fi should be in the same ip address before running the code. When we switch on the raspberry pi and run the code, the reports regarding the pollution levels will be continuously updated. We can view those reports in ubidots. We have to maintain our own account in that IOT platform. The values are stored in that server. It also provides the analysis of reports. Hence, we can monitor the pollution levels.

RASPBERRY PI PROGRAM

```
import time

import sounddevice as sd

import numpy as np

import board

import digitalio

import adafruit_character_lcd.character_lcd as characterlcd


# Initialization of  LCD

lcd_rs = digitalio.DigitalInOut(board.D22)

lcd_en = digitalio.DigitalInOut(board.D17)

lcd_d7 = digitalio.DigitalInOut(board.D27)

lcd_d6 = digitalio.DigitalInOut(board.D18)

lcd_d5 = digitalio.DigitalInOut(board.D23)

lcd_d4 = digitalio.DigitalInOut(board.D24)


lcd = characterlcd.Character_LCD_Mono(lcd_rs, lcd_en, lcd_d4, lcd_d5, lcd_d6, lcd_d7, 16, 2)


# Set up the microphone

sample_rate = 44100

duration = 10  # Number of seconds to record sound


def callback(indata, frames, time, status):

    if status:

        print(status, flush=True)


def record_sound():

    with sd.InputStream(callback=callback, channels=1, samplerate=sample_rate):

        sd.sleep(duration * 1000)


def analyze_sound(data):
    # Calculate the average volume
    average_volume = np.mean(np.abs(data))

    # Classify loudness
    if average_volume < 30:
        return "Quiet"
    elif average_volume < 60:
        return "Moderate"
    else:
        return "Loud"

# Main loop
while True:
    lcd.clear()
    lcd.message = "Listening..."

    # Record and analyze sound
    sound_data, overflowed = sd.rec(int(sample_rate * duration), samplerate=sample_rate,
channels=1, dtype='int16')
    sd.wait()
    loudness = analyze_sound(sound_data)

    lcd.clear()
    lcd.message = f"Sound Level: {loudness}"

    time.sleep(10)
```

OUTPUT



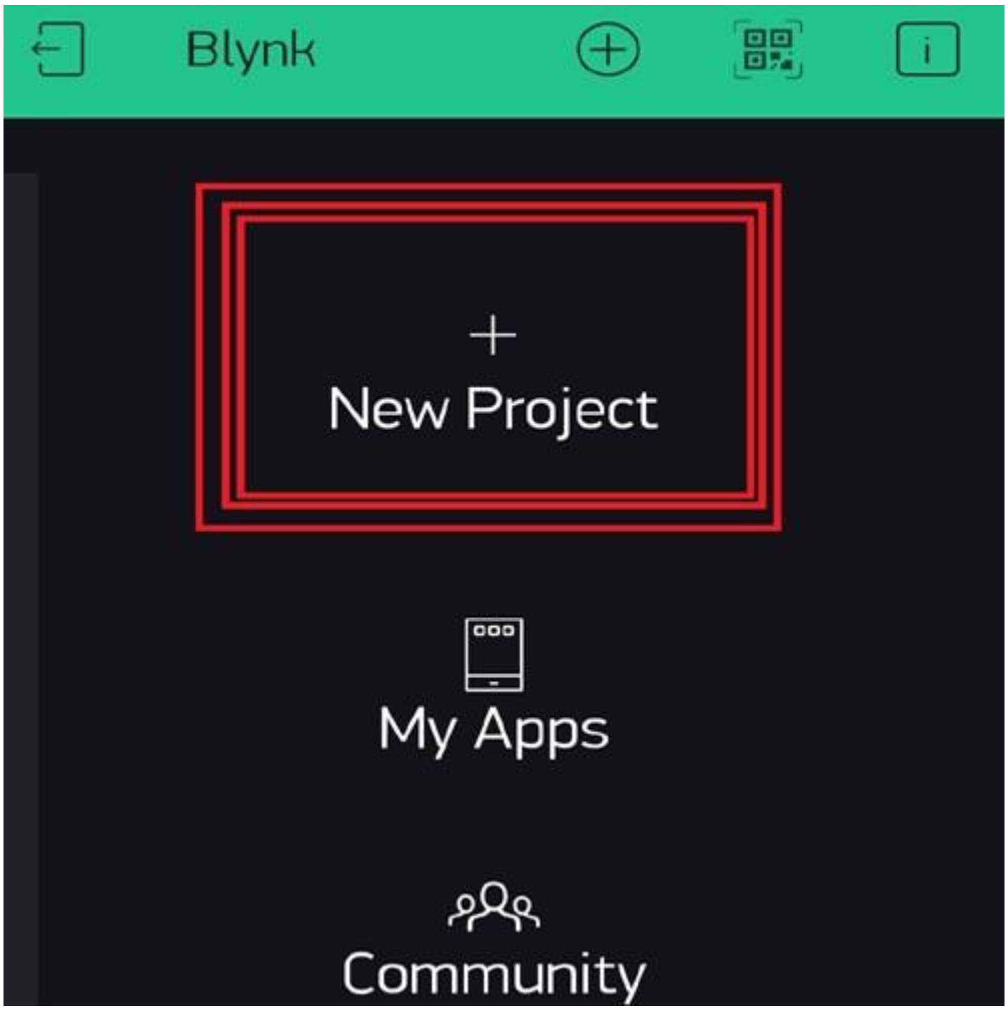
CONNECTING TO MOBILE APPLICATION (BLYNK APK)

Blynk is an **IoT software platform** that provides infrastructure for the internet of Things. **It supports** 400+ development boards, SBC's, and other modules. It allows users to build and manage connected products with no code, and to remotely control, monitor, and automate them with mobile and web applications. It also offers secure cloud, data analytics, user and access management, alerts, and Over-The-Air firmware updates[1].

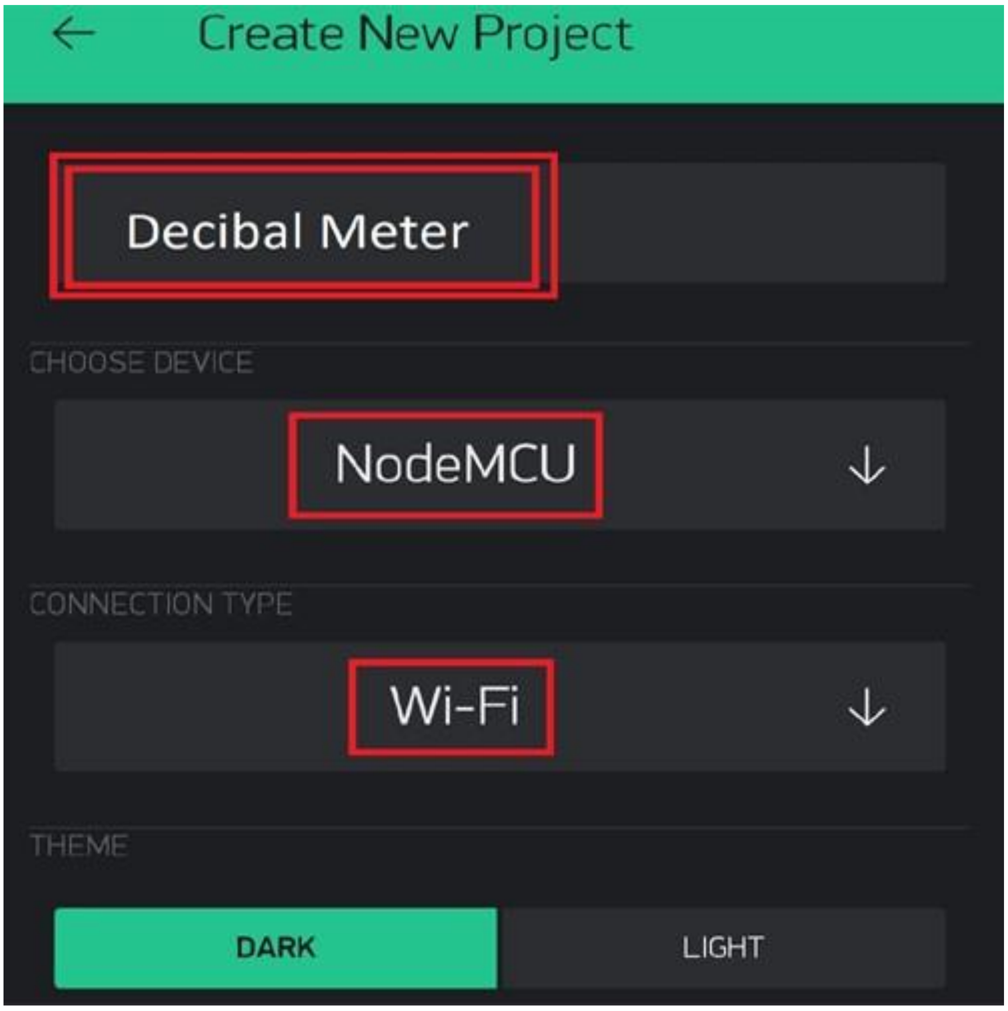
To create a device using Blynk, you can manually create a device using Blynk.Console for initial prototyping (works for any hardware) or use Static Tokens for cellular, Ethernet, and other non-WiFi connection methods. Activating devices with manually generated AuthTokens is recommended for prototyping stages or when you build a device for yourself.

- First we need to install the Blynk app from PlayStore and create an account.

- Click on the create button and create your new project.



- Give your project a name and choose the board as NodeMCU and connection type as Wi-Fi.



Create New Project

Decibal Meter

CHOOSE DEVICE

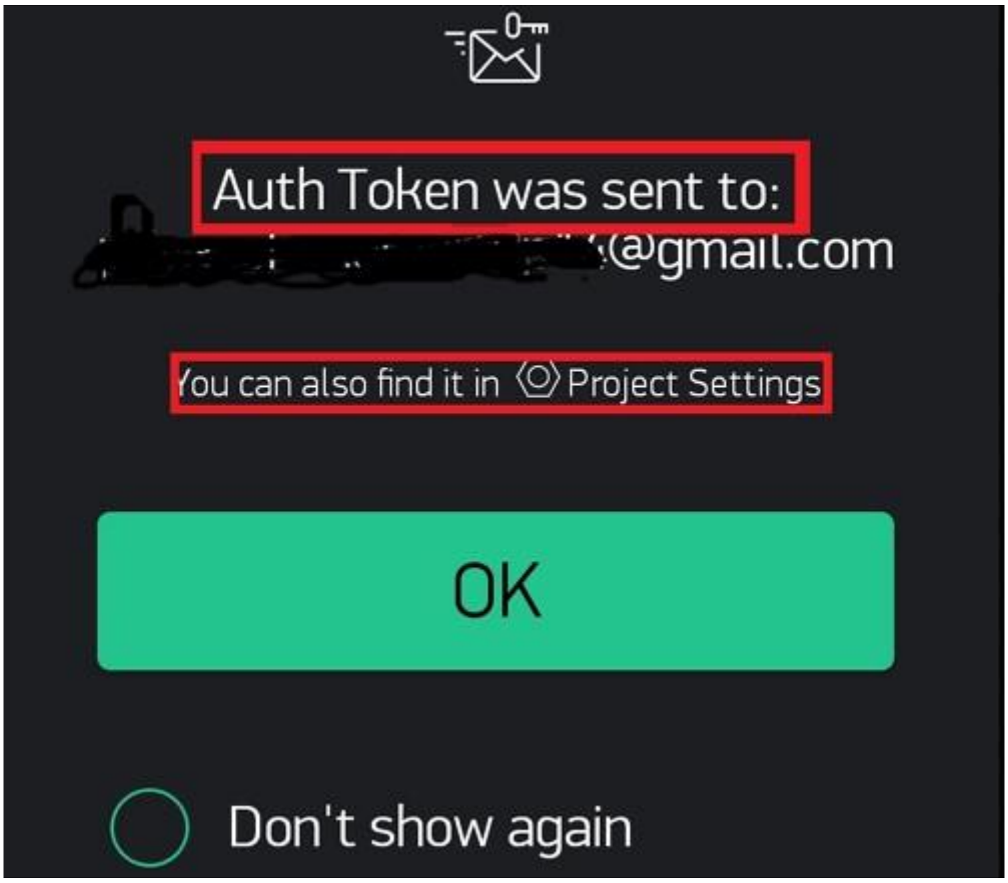
NodeMCU

CONNECTION TYPE

Wi-Fi

THEME

- An auth token will be sent to your registered email id. Keep it safe as it will be used later on while programming.

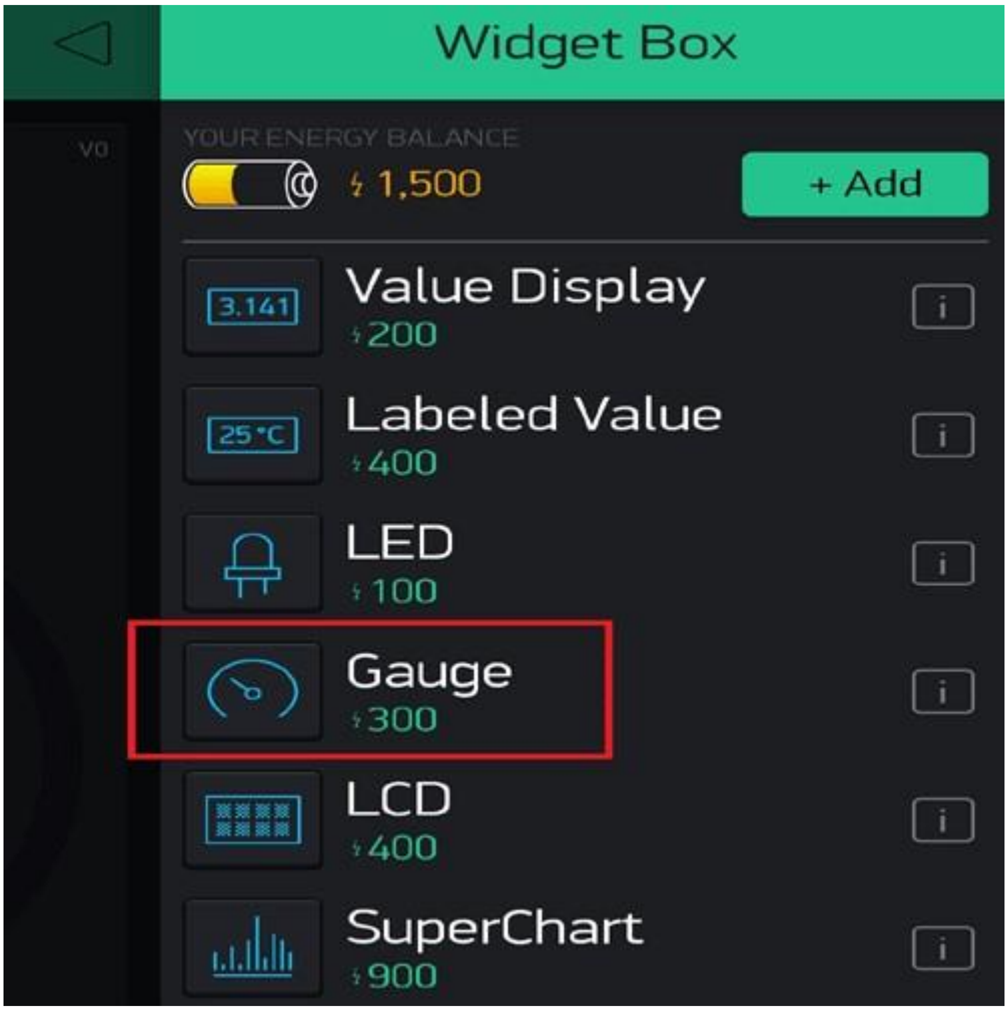


Auth Token was sent to:
[redacted]@gmail.com

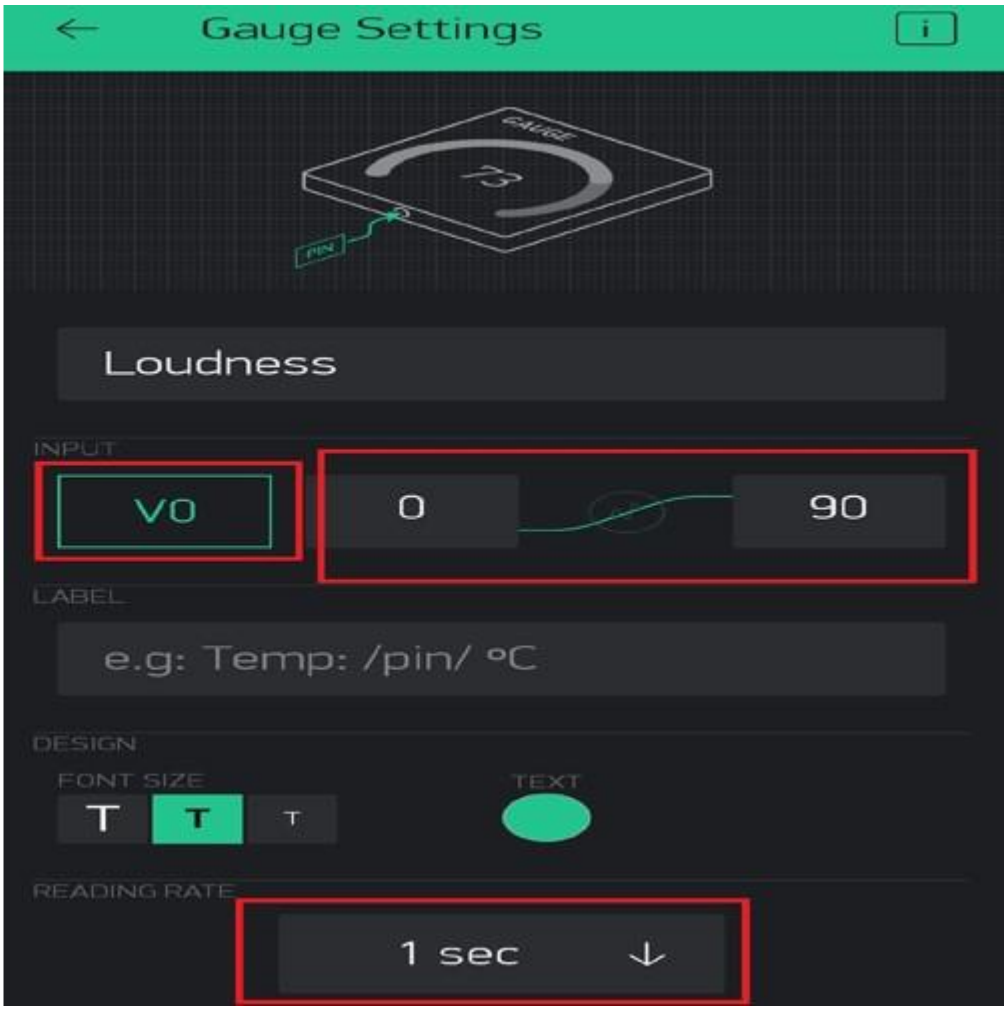
You can also find it in Project Settings

Auth Token is a unique alphanumeric string to identify your project in the Blynk's server and assigning the correct data through it.

- Now drag and drop a gauge from the list and configure it.



- Connect the gauge to virtual pin V0 and set the values to 0 and 90 respectively, also set the reading rate to 1 sec.


Gauge Settings

Loudness

INPUT

V0 0 90

LABEL

e.g: Temp: /pin/ ºC

DESIGN

FONT SIZE

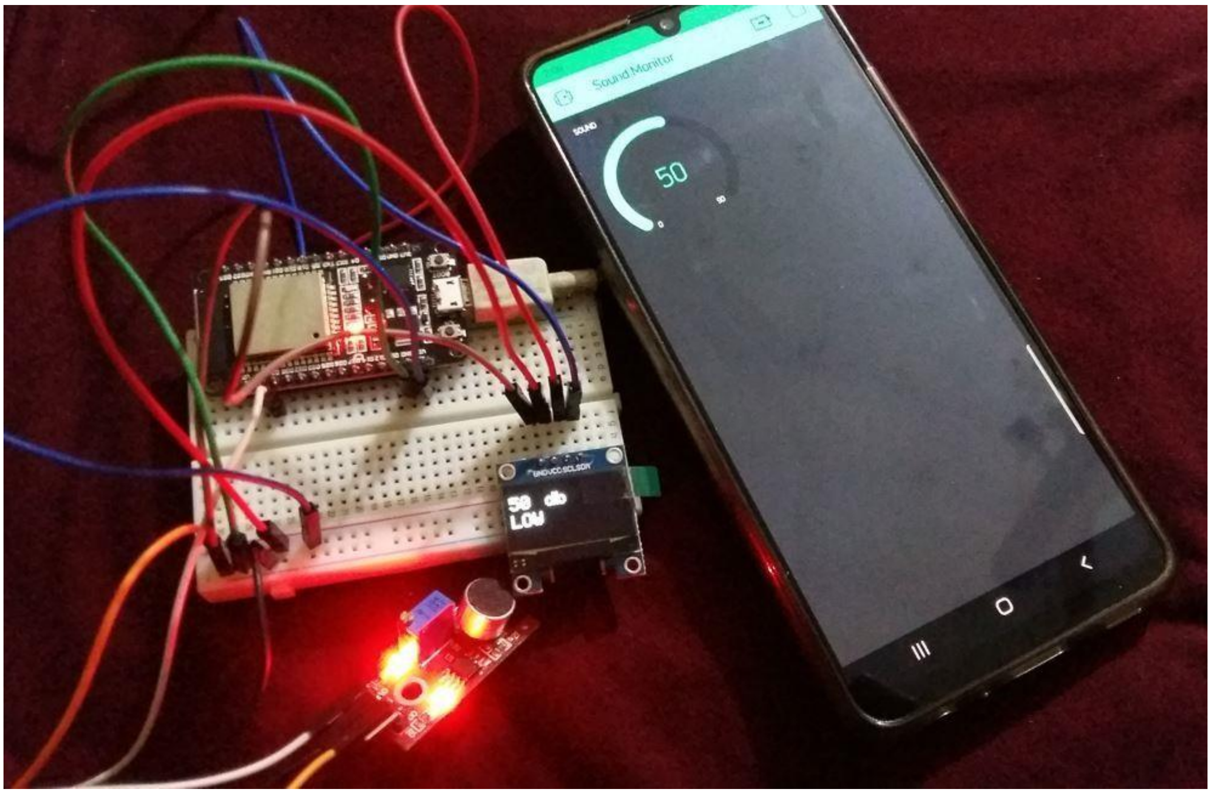
TEXT

READING RATE

1 sec

And now you’re done with setting up Blynk. Let us jump to the coding part.

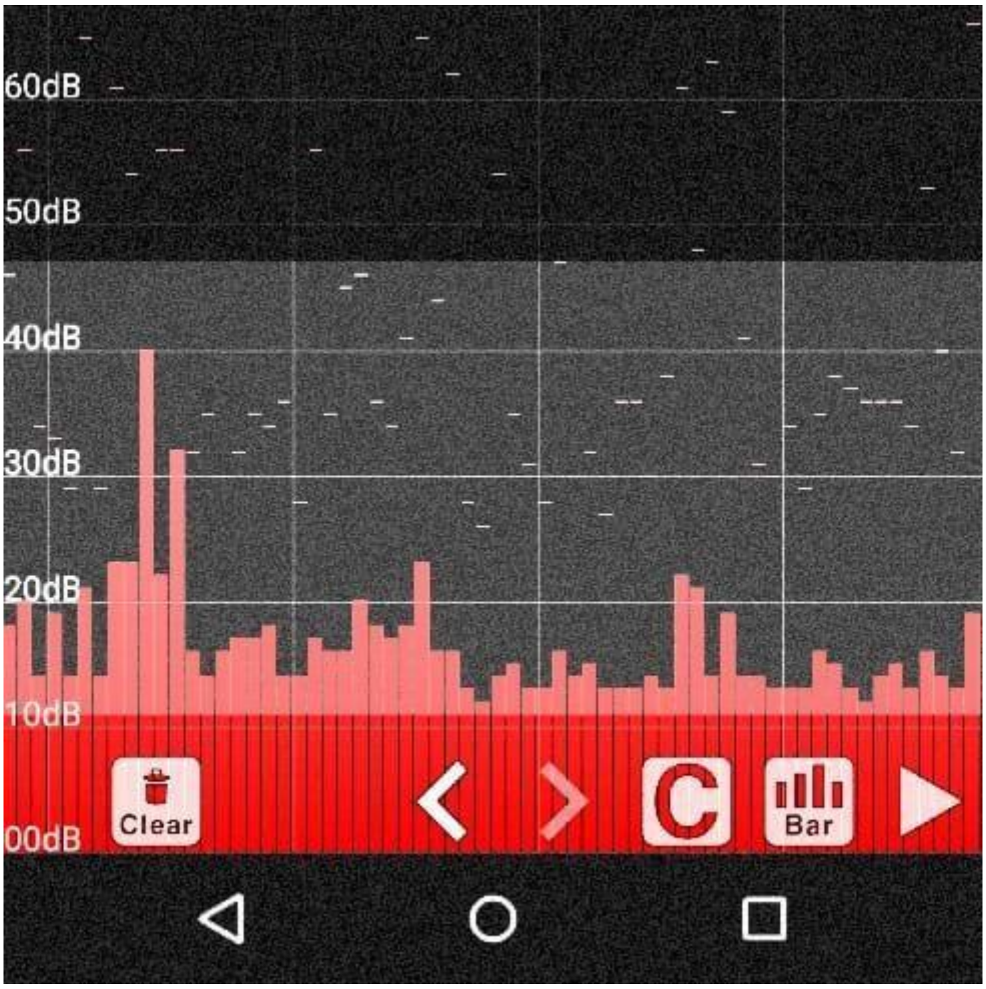
MOBILE APP OUTPUT

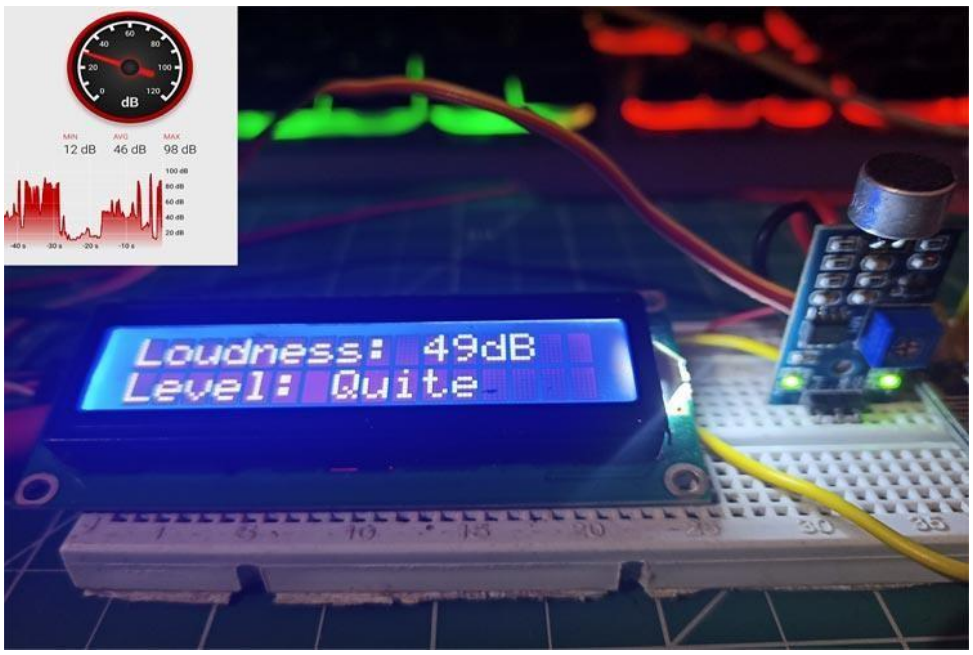


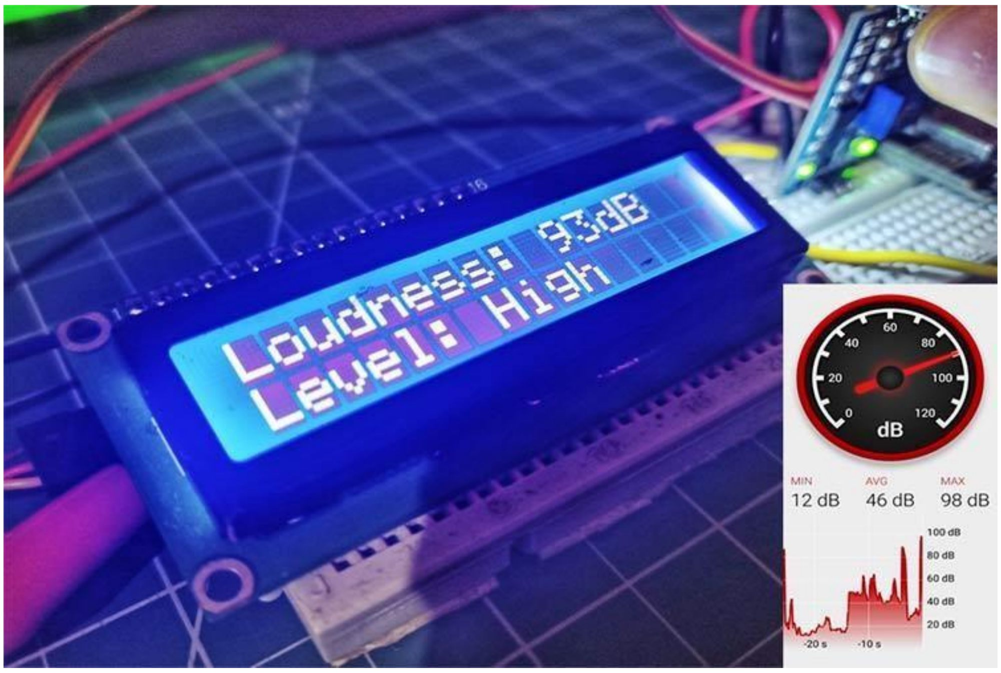
HTML PROGRAM

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Noise Pollution Monitoring using IoT</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            background-color: #f0f0f0;
        }
        h1 {
            color: #0074e4;
        }
        #noise-level {
            font-size: 36px;
            margin-top: 20px;
            color: #333;
        }
        #check-button {
            background-color: #0074e4;
            color: #fff;
            padding: 10px 20px;
            border: none;
            cursor: pointer;
            border-radius: 5px;
        }
        #container {
            background-color: #fff;
            padding: 20px;
            border-radius: 10px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.2);
            width: 300px;
            margin: 0 auto;
        }
        #hearing-range {
            max-width: 100%;
        }
    </style>
</head>
<body>
    <div id="container">
        <h1>Noise Pollution Monitoring using IoT</h1>
        <p>Current Noise Level (dB): <span id="noise-level">---</span></p>
        <button id="check-button">Check Current Noise Level</button>
        <img id="hearing-range" src="hearing-range.png" alt="Preferred Hearing Range">
    </div>

    <script>
        // Simulated function to get the current noise level from IoT
        function getNoiseLevel() {
            return Math.floor(Math.random() * 100); // Simulated noise level (0-100 dB)
        }

        document.getElementById("check-button").addEventListener("click", function() {
            const noiseLevel = getNoiseLevel();
            document.getElementById("noise-level").textContent = noiseLevel + " dB";
        });
    </script>
</body>
</html>
```

SAMPLE OUTPUT






Loudness: 93dB
Level: High
dB
MIN
12 dB
AVG
46 dB
MAX
98 dB