PROJECT 4 : NOISE POLLUTION MONITORING SYSTEM

PHASE 1 : Problem definition :

The project involves deploying IOT sensors to measure noise pollution in public areas and providing real-time noise level data accessible to the public through a platform or mobile app.

Key Objectives :

- **Data Collection:** To collect and record noise pollution data from multiple sensors deployed in different locations, providing a comprehensive view of noise levels within an area.
- **Environmental Protection:** Monitor noise levels in sensitive natural areas to protect ecosystems , wildlife and human health from noise pollution.
- **Threshold Alarms**: Set predefined noise level thresholds and trigger alarms or notifications when noise levels exceed these limit**.**
- **Noise Mapping**: Create noise maps to visualize noise distribution and identify high-noise areas.
- **Mitigation Planning**: Develop strategies and policies for noise pollution reduction and mitigation.
- **Noise Control Measures**: Evaluate the effectiveness of noise control measures and technologies.

Design Thinking :

- **Understand Users:**
  Identify who will use the system and understand their needs.
- **Define the Problem:**
  Clearly state the issues related to noise pollution to be addressed.

- **Generate Ideas:**

Brainstorm creative solutions, considering both tech and non-tech options.

- **Build a Prototype:**
  Create a basic version of the monitoring system for testing.
- **Test in Real Environment:**
  Implement the prototype in a real-world setting for feedback.
- **Refine and Iterate:**
  Improve the design based on user feedback and testing.
- **Develop Final Version:**
  Create the complete noise pollution monitoring system.
- **Evaluate Impact:**
  Assess how well the system addresses noise pollution.
- **Ensure Sustainability**:
  Plan for ongoing support and maintenance.
- **Communicate Benefits:**
  Clearly communicate the system's advantages to the public and authorities.

PHASE 2 : INNOVATION :

After thorough research and analysis, we arrived at an innovative solution to solve the above problem as detailed in phase 1 of our project. We will be using the ESP8266 micro controller as well as Arduino UNO microcontroller as both these suit the best for our project.

Sensors:

Microphone sensor (LM 393): Employing IOT sensors is crucial for accurate data collection and analysis. Consider using sound level meters or microphones (LM393 sensor) as primary sensors, coupled with a microcontroller such as Arduino for signal processing and data transmission. These devices can capture real-time audio data, measuring noise levels in decibels.

Connectivity :

Ensure connectivity through a robust network infrastructure, integrating sensors with IOT devices. Utilize Wi-Fi, cellular, or LPWAN technologies for data transmission to a central server for real-time monitoring and analysis.

Cloud :

Beeceptor is used for rapid API prototyping, endpoint validation, and simulation of diverse data scenarios. While beneficial for testing, it complements, rather than replaces, dedicated cloud platforms that provide robust infrastructure for data storage, analysis, and real-time processing.

Protocol :

Implementation using MQTT (Message Queuing Telemetry Transport) protocol for efficient data transmission. This lightweight and reliable protocol ensure seamless connectivity, making it ideal for monitoring and managing noise levels effectively.

Public platform :

Design an easy-to-use interface for seamless navigation. Encourage user participation by enabling data contributions and insights sharing on social media. Educate the public about noise pollution's effects through informative resources. Support innovation with API access for developers to create applications using the data. Ensure broad accessibility through web browsers and mobile apps for widespread public awareness.

Phase -3

PROBLEM

In this project, we will create an noise pollution monitoring system to detect sound in decibels and display sound and level of the sound in decibel on an OLED. Additionally, we will link our sound pollution monitor with Blynk application and users can monitor the sound decibels on the Blynk app.

DEPLOYED IOT DEVICES:

1. **ESP32 Microcontroller:** The ESP32 is a powerful microcontroller widely used for IoT applications due to its built-in Wi-Fi and Bluetooth capabilities.
2. **Sound Sensor (Microphone):** The code implies the use of a sound sensor to measure ambient noise levels. It likely provides analog voltage readings (connected to pin A0) that are then converted into decibel levels.
3. **Buzzer:** A simple buzzer or piezo element is employed to produce sound alerts when the noise levels exceed a certain threshold. The buzzer is connected to pin 19.
4. **OLED Display (Adafruit SSD1306):** The Adafruit SSD1306 is an OLED display module used for visual output. It's connected via I2C communication (using the Wire library), and the code displays the calculated decibel values and descriptors on this display.

SOURCE CODE

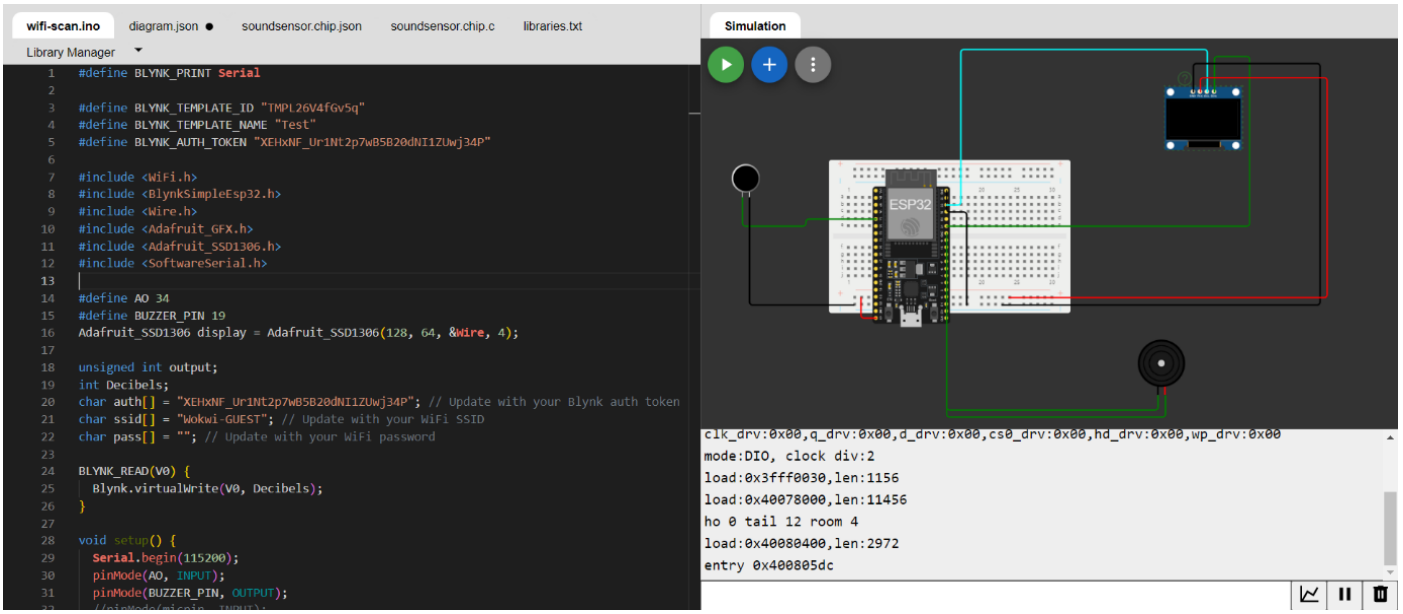
This code is used for monitoring sound levels and displaying the dB value on the OLED screen, accompanied by different displays based on the sound level and generating a sound alarm when it exceeds a certain threshold.

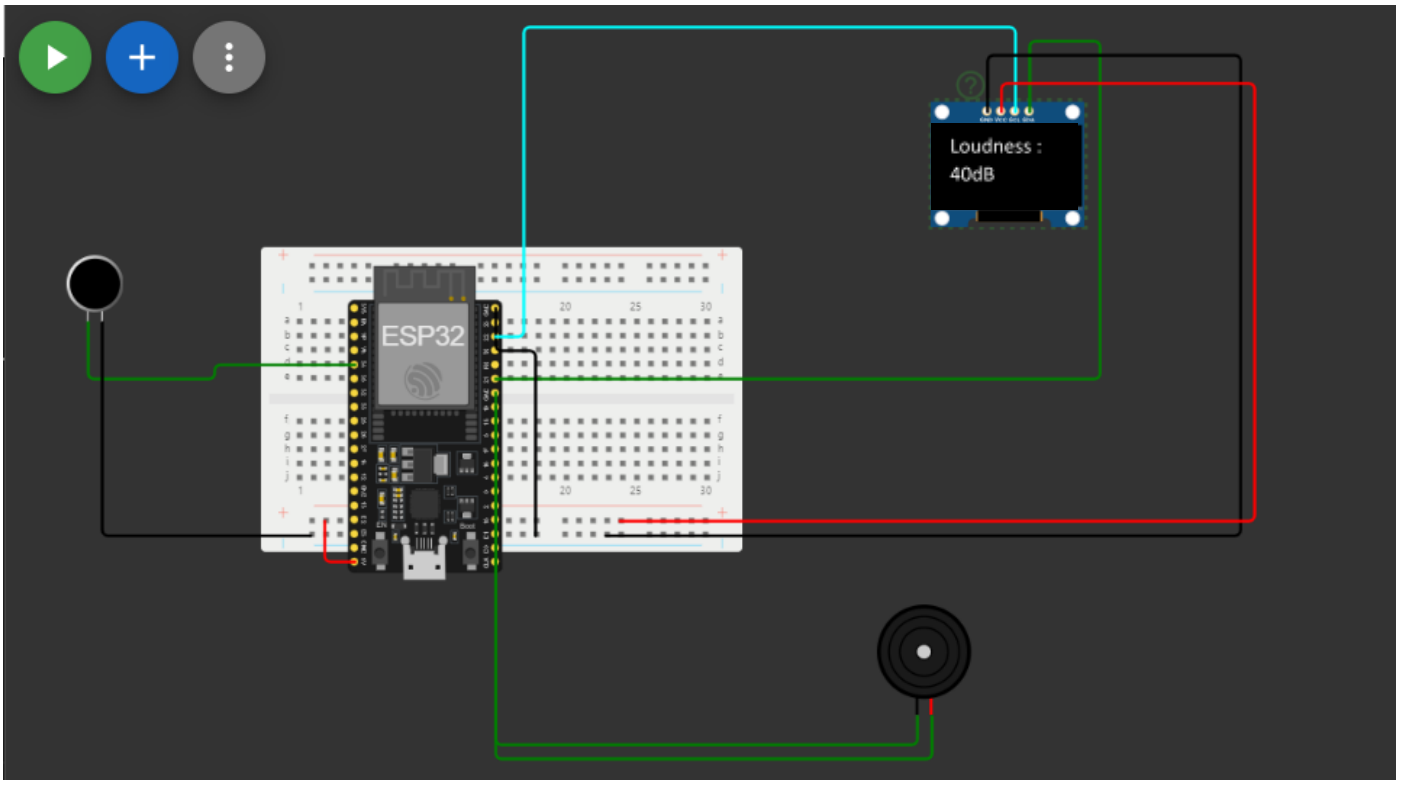
```
#define BLYNK_PRINT Serial

#define BLYNK_TEMPLATE_ID "TMPL26V4fGv5q"
#define BLYNK_TEMPLATE_NAME "Test"
#define BLYNK_AUTH_TOKEN "XEHxNF_Ur1Nt2p7wB5B20dNI1ZUwj34P"

#include <WiFi.h>
#include <BlynkSimpleEsp32.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <SoftwareSerial.h>

#define AO 34
#define BUZZER_PIN 19
Adafruit_SSD1306 display = Adafruit_SSD1306(128, 64, &Wire, 4);

unsigned int output;
int Decibels;
char auth[] = "XEHxNF_Ur1Nt2p7wB5B20dNI1ZUwj34P"; // Update with your Blynk auth token
char ssid[] = "Wokwi-GUEST"; // Update with your WiFi SSID
char pass[] = ""; // Update with your WiFi password

BLYNK_READ(V0) {
  Blynk.virtualWrite(V0, Decibels);
}
```

```
void setup() {
  Serial.begin(115200);
  pinMode(AO, INPUT);
  pinMode(BUZZER_PIN, OUTPUT);
  //pinMode(micpin, INPUT);

  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    for (;;);
  }

  delay(2000);
  display.clearDisplay();
  display.setTextColor(WHITE);

  Blynk.begin(auth, ssid, pass);
}

void loop() {
  Blynk.run();
  float mic = analogRead(A0);
  float PeakToPeak = voltageToDB(mic);

  Decibels = map(PeakToPeak, 50, 500, 49.5, 90);

  display.setTextSize(2);
  display.setCursor(0, 10);
  display.print(Decibels);
```

```
display.setTextSize(2);
display.setCursor(40, 10);
display.print("db");
display.display();

if (Decibels <= 50) {
  display.setTextSize(2);
  display.setCursor(0, 30);
  display.print("LOW");
  display.display();
} else if (Decibels > 50 && Decibels < 75) {
  display.setTextSize(2);
  display.setCursor(0, 30);
  display.print("Moderate");
  display.display();
} else if (Decibels >= 75) {
  display.setTextSize(2);
  display.setCursor(0, 30);
  display.print("HIGH");
  display.display();
  tone(BUZZER_PIN, 1000); // 1000Hz tone
  delay(1000); // Buzz for 1 second
  noTone(BUZZER_PIN); // Stop buzzing
  delay(1000); // Wait for 1 second
}

// Example mapping function
float voltageToDB(float voltage) {
  // You need to know the calibration factor and reference voltage for this conversion
  float sensitivity = 30; // Sensitivity of the microphone (you should adjust this)
  float referenceVoltage = 5.0; // Reference voltage used

  // Calculate the voltage ratio
  float voltageRatio = voltage / referenceVoltage;

  // Convert to decibels (assuming a linear relationship between voltage and decibels)
  float dB = 20 * log10(voltageRatio) + sensitivity;

  return dB;
}
```

SIMULATION AND OUTPUT





Python Script Structure

```python
import requests
import time

while True:

    noise_level = read_noise_level()


    data = {
        'noise_level': noise_level,
        'location': 'Your_Location_Info'

    }


    url = 'https://blynk/api/data'
    headers = {'Content-Type': 'application/json'}

    try:
        response = requests.post(url, json=data, headers=headers)
        if response.status_code == 200:
            print(f"Data sent successfully: {data}")
        else:
            print(f"Failed to send data. Status code: {response.status_code}")
    except requests.exceptions.RequestException as e:
        print(f"Error: {e}")


    time.sleep(5)  # Adjust the time interval based on your requirements
```

PHASE - 4

About Thing Speak:

ThingSpeak is an IoT analytics platform service that allows you to aggregate, visualize, and analyse live data streams in the cloud. The platform's real-time data management tools facilitate swift decision-making based on live information streams. Its popularity in the IoT landscape stems from the simplicity it offers in channel creation and data manipulation. Whether tracking environmental variables, controlling smart devices, or conducting research, ThingSpeak provides a robust framework. The collaborative nature of ThingSpeak encourages a community-driven approach, fostering the sharing of ideas and code snippets. Overall, ThingSpeak stands as a dynamic hub for IoT enthusiasts, amplifying the potential for innovation in the interconnected world of devices and data.

Code to display Real time Transit Information on Thinkspeak:

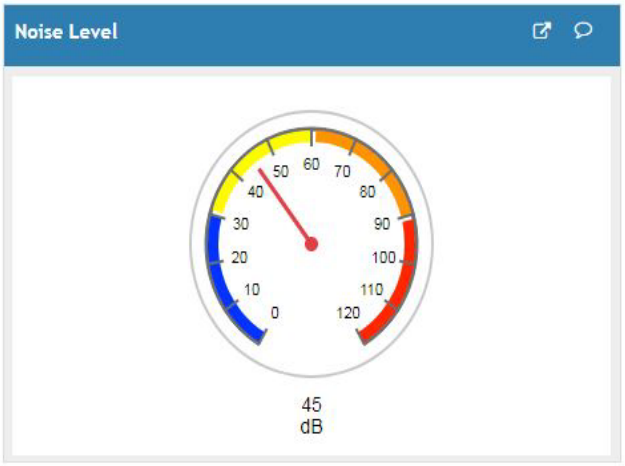
Noise_pollution.ino

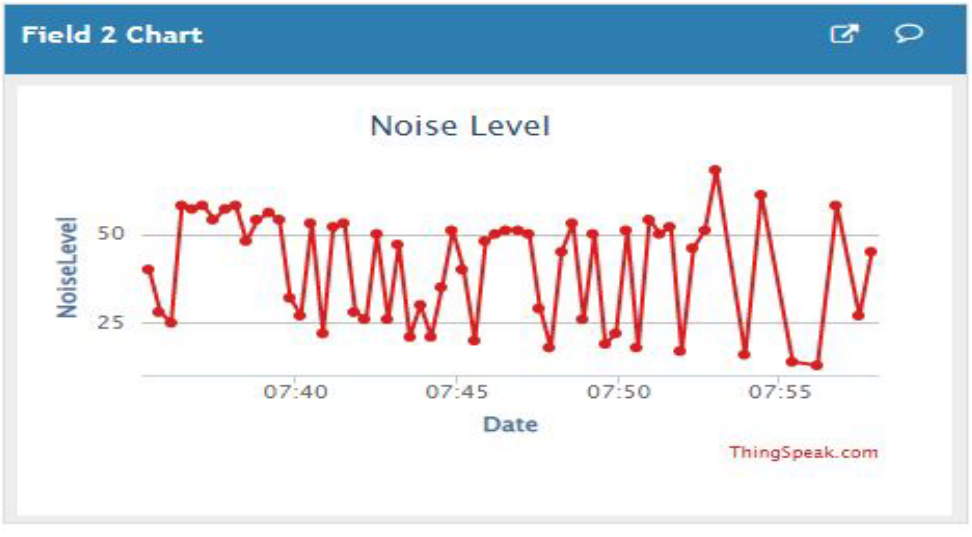
```
#include <ESP8266WiFi.h>
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128    // OLED display width, in pixels
#define SCREEN_HEIGHT 64    // OLED display height, in pixels
#define OLED_RESET -1       // Reset pin # (or -1 if sharing search Arduino reset pin)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

String apiKey = "N2WXZ15RHOJED7VC"; // Enter your Write API key from ThingSpeak
const char *ssid = "Alexahome";     // replace with your wifi ssid and wpa2 key
const char *pass = "12345678";
const char* server = "api.thingspeak.com";

const int sampleWindow = 50;                              // Sample window width in mS (50 mS = 20Hz)
unsigned int sample;_

WiFiClient client;

void setup()
{
  Serial.begin(115200);                                    //Serial comms for debugging
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);              //OLED display start
  display.display();                                      //show buffer
  display.clearDisplay();                                 //clear buffer
  display.setTextSize(1);                                 //Set text size to 1 (1-6)
  display.setTextColor(WHITE);                            //Set text color to WHITE (no choice lol)
  display.setCursor(0,0);                                 //cursor to upper left corner
  display.println("Decibelmeter");                 //write title
  display.display();                                      //show title
  delay(2000);                                            //wait 2 seconds

  WiFi.begin(ssid, pass);
```

```
36
37    while (WiFi.status() != WL_CONNECTED)
38    {
39      delay(500);
40      Serial.print(".");
41    }
42      Serial.println("");
43      Serial.println("WiFi connected");
44
45      display.clearDisplay();
46      display.setCursor(0,0);
47      display.setTextSize(1);
48      display.setTextColor(WHITE);
49      display.print("WiFi connected");
50      display.display();
51      delay(4000);
52      display.clearDisplay();
53  }
54
55  void loop()
56  {
57    unsigned long startMillis= millis();                  // Start of sample window
58    float peakToPeak = 0;                                 // peak-to-peak level
59
60    unsigned int signalMax = 0;                           //minimum value
61    unsigned int signalMin = 1024;                        //maximum value
62
63                                                          // collect data for 50 mS
64    while (millis() - startMillis < sampleWindow)
65    {
66      sample = analogRead(0);                             //get reading from microphone
67      if (sample < 1024)                                  // toss out spurious readings
68      {
69        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
        float db = map(peakToPeak,20,900,49.5,90);   //calibrate for deciBels
        display.setCursor(0,0);                      //cursor to upper left
        display.setTextSize(2);                      //set text size to 2
        display.print(db);                           //write calibrated deciBels
        display.print(" dB");                        //write units


         for(int x =5;x<114;x=x+6){                  //draw scale
           display.drawLine(x, 32, x, 27, WHITE);
         }
        display.drawRoundRect(0, 32, 120, 20, 6, WHITE);  //draw outline of bar graph
        int r = map(db,0,120,1,120);                 //set bar graph for width of screen
        display.fillRoundRect(1, 33, r, 18, 6, WHITE);    //draw bar graph with a width of r
        display.display();                           //show all that we just wrote & drew
        display.clearDisplay();                      //clear the display

        if (client.connect(server, 80)) // "184.106.153.149" or api.thingspeak.com
       {
         String postStr = apiKey;
         postStr += "&field1=";
         postStr += String(db);
         postStr += "r\n";

         client.print("POST /update HTTP/1.1\n");
         client.print("Host: api.thingspeak.com\n");
         client.print("Connection: close\n");
         client.print("X-THINGSPEAKAPIKEY: " + apiKey + "\n");
         client.print("Content-Type: application/x-www-form-urlencoded\n");
         client.print("Content-Length: ");
         client.print(postStr.length());
         client.print("\n\n");
         client.print(postStr);

       }
         client.stop();
         delay(150);
    }
```

OUTPUT:



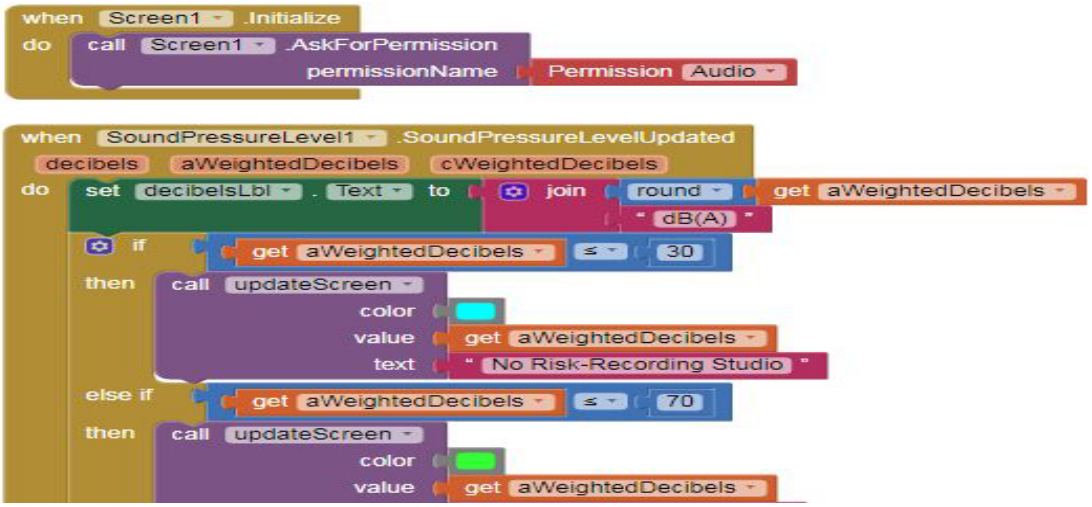


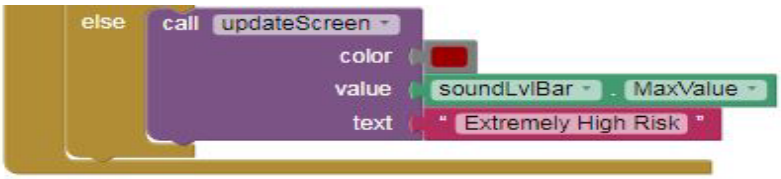
MOBILE APPLICATION:

Additionally, MIT App Inventor provides a simple and intuitive interface for testing and debugging apps in real-time. It emphasizes accessibility by enabling users to create functional apps with minimal effort. The platform supports a variety of features, including sensors, media playback, and connectivity options, allowing users to explore diverse app functionalities. MIT App Inventor serves as a valuable resource for individuals looking to kickstart their journey into mobile app development, fostering creativity and hands-on learning.

For collecting and storing real-time transit data, ThingSpeak channels were employed, capturing information like route details, location, and passenger count for each data point. In the MIT App Inventor, a visual interface facilitated app development, utilizing components like labels, text boxes, and web elements to shape the user interface. To retrieve data, the app was configured to interact with ThingSpeak channels through the Web component, making API calls for the latest entries. The app dynamically displayed the retrieved data, presenting route information, location, and passenger count in real-time on its interface.

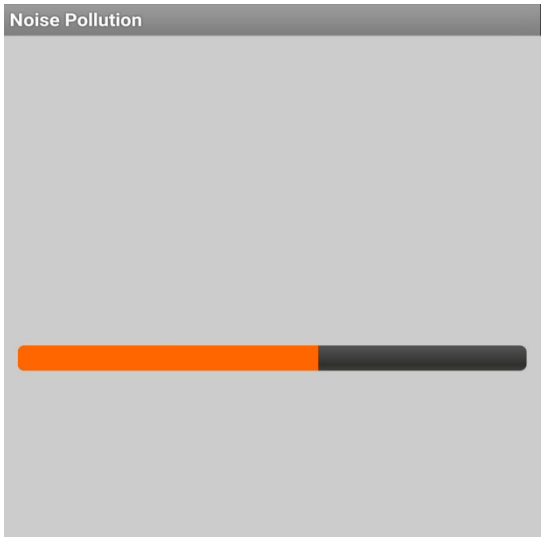
MIT APP INVENTOR:



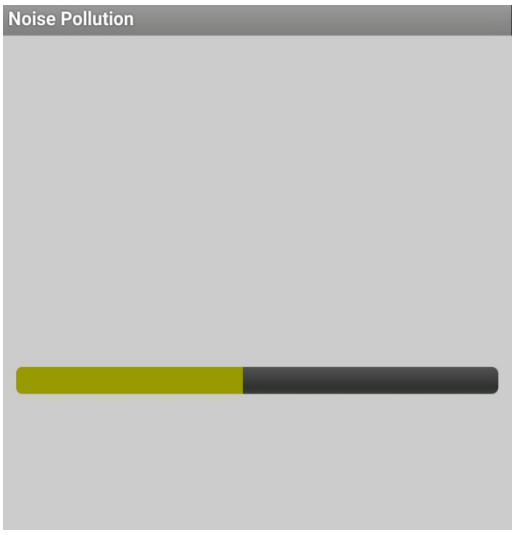


OUTPUT :

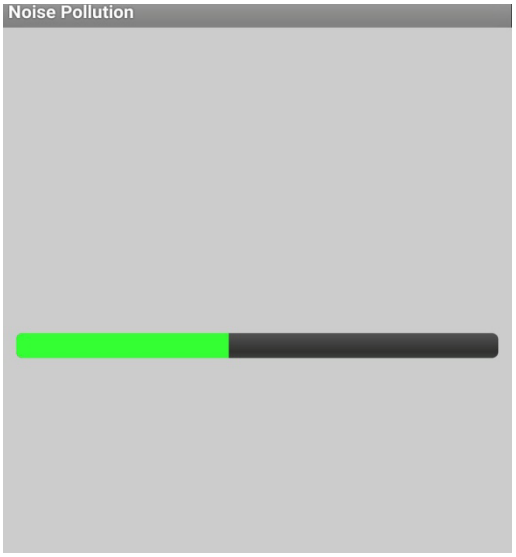
When noise level <= 120 dB



When noise level <= 85dB



When noise level <= 70dB



TEAM MEMBERS:

1. Jayanth N - (2021504523)
2. Lavanya V - (2021504525)
3. Mathana K - (2021504528)
4. Sudhanthira P - (2021504548)
5. Vidhya SS - (2021504557)