```
/*VISITOR AUDIT SYSTEM */

/* LORA TRANSMITTER CODE */

  // Includes the libraries
#include <Wire.h>.
#include <SPI.h>
#include <LoRa.h>

 // Defines Tirg and Echo pins of the Ultrasonic Sensor
const int trigPin = 3;
const int echoPin = 4;
const int ledPin = 13;  // the pin that the LED is attached to

 // Variables for the duration and the distance
long duration, distance; // Duration used to calculate distance
int sensorCounter = 0;   // counter for the number of button presses
int lastsensorDistance = 0;
int setCounter = 20;
int incomingByte;

void setup() {

 Serial.begin (9600);
 pinMode(trigPin, OUTPUT);
 pinMode(echoPin, INPUT);
 pinMode(ledPin, OUTPUT);
 }

void loop() {

 if (Serial.available() > 0) {           // see if there's incoming serial data:
  incomingByte = Serial.read();           // read the oldest byte in the serial buffer:

  if (incomingByte == 'R') {              // if it's a capital R, reset the counter
      Serial.println("Reset");
      sensorCounter = 20;

    }
  }

 /* The following trigPin/echoPin cycle is used to determine the
 distance of the nearest object by bouncing soundwaves off of it. */
 digitalWrite(trigPin, LOW);
 delayMicroseconds(2);
 digitalWrite(trigPin, HIGH);
 delayMicroseconds(10);

 digitalWrite(trigPin, LOW);
 duration = pulseIn(echoPin, HIGH);

 //Calculate the distance (in cm) based on the speed of sound.
 distance = duration/58.2;

 if (distance <= 20 && lastsensorDistance >= 40)
{
  LoRa.beginPacket();
  sensorCounter++;
  LoRa.print(number_);
  LoRa.print(of_);
  LoRa.print(counts:);
  LoRa.print(" ");
  Serial.print("number of counts:  ");
  LoRa.print(sensor);
  LoRa.print(Counter);
  Serial.println(sensorCounter);
  LoRa.print(distance);
  Serial.println(distance);
  LoRa.endPacket();
   }

 else {
   //Serial.println("off");   not needed.
  }
  lastsensorDistance = distance;
  delay(500);

// turns on the LED when counter is at setCounter
  if (sensorCounter >= setCounter) {
    digitalWrite(ledPin, HIGH);
  }
 else {
   digitalWrite(ledPin, LOW);
  }
}
```

```
/* WIFI LORA RECEIVER CODE */


#include <WiFi.h>
#include <FirebaseESP32.h>
#include<stdio.h>
#include<string.h>
#include <SPI.h>
#include <LoRa.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
#define BAND 433E6
#define FIREBASE_HOST "minidata-8543a.firebaseio.com"
//Do not include https:// in FIREBASE_HOST
#define FIREBASE_HOST "minidata-8543a.firebaseio.com"
 //Do not include https:// in FIREBASE_HOST
#define FIREBASE_AUTH "sVXQrYDBOUZhcsUOdpQkbDPrj1eYZyub7kf1fGzW"
#define WIFI_SSID "NO>>>BODY"
#define WIFI_PASSWORD "aa9b8822y1zz"

String LoRaData="";
char c[50], *strings[10], *ptr = NULL;
//Define FirebaseESP32 data object
FirebaseData firebaseData;
FirebaseJson json;
//void printResult(FirebaseData &data);
void setup()  {
Serial.begin(115200);
Serial.println("LoRa Receiver Test");
 //SPI LoRa pins
SPI.begin(SCK, MISO, MOSI, SS);
//setup LoRa transceiver module
LoRa.setPins(SS, RST, DIO0);
if (!LoRa.begin(BAND)) {
Serial.println("Starting LoRa failed!");
while (1);
}
Serial.println("LoRa Initializing OK!");
WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
Serial.print("Connecting to Wi-Fi");
```

```
 while (WiFi.status() != WL_CONNECTED)
 {
   Serial.print(".");
  delay(30);
 }
Serial.println();
Serial.print("Connected with IP: ");
Serial.println(WiFi.localIP());
Serial.println();


 Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
  Firebase.reconnectWiFi(true);
 //Set database read timeout to 1 minute (max 15 minutes)
Firebase.setReadTimeout(firebaseData, 1000*60);
//tiny, small, medium, large and unlimited.
 //Size and its write timeout e.g. tiny (1s), small (10s), medium (30s) and large (60s).
   Firebase.setwriteSizeLimit(firebaseData, "tiny");

 String path = "/Visitor monitoring";;
 int i=0;
 delay(100);

  if (Firebase.setInt(firebaseData, path + "/Angle", i))
   {
    Serial.println("PASSED");
    Serial.println("PATH: " + firebaseData.dataPath());
    Serial.println("------------------------------------");
    Serial.println();
 }
 else
  {
 Serial.println("FAILED");
  Serial.println("REASON: " + firebaseData.errorReason());
  Serial.println("------------------------------------");
  Serial.println();      }
  if (Firebase.setInt(firebaseData, path + "/Distance", i))
  {
    Serial.println("PASSED");
    Serial.println("PATH: " + firebaseData.dataPath());
    Serial.println("------------------------------------");
   Serial.println();
 }
 else
 {
  Serial.println("FAILED");
  Serial.println("REASON: " + firebaseData.errorReason());
  Serial.println("------------------------------------");
  Serial.println();
}
 delay(1000);
 }

 void loop()  {
FirebaseJson updateData;
 //try to parse packet
 int packetSize = LoRa.parsePacket();

  if (packetSize) {
   //received a packet
  Serial.print("Received packet ");

while (LoRa.available())
{
  LoRaData = (LoRa.readString());
  LoRaData.toCharArray(c,50);
  Serial.println(LoRaData);
      byte index = 0;
      ptr = strtok(c, " ");
  // takes a list of delimiters

     while(ptr != NULL)
      {
          strings[index] = ptr;
          index++;
          ptr = strtok(NULL, " ");
  // takes a list of delimiters
      }
       int n1;
     float n2;
     n1=atoi(strings[0]);
     n2= atof(strings[1]);
      String str=String(n1);
     String str1=String(n2);
      Serial.println(n1);
      Serial.println(n2);
       updateData.set("Angle",str);
  updateData.set("Distance",str1);
```

```
   Serial.print(LoRaData);

 if (Firebase.updateNode(firebaseData, "/Visitor monitoring", updateData))
{
      Serial.println(firebaseData.dataPath());
      Serial.println(firebaseData.dataType());
      Serial.println(firebaseData.jsonString());
     }
else {
    Serial.println(firebaseData.errorReason());
  }
}
      delayMicroseconds(2);
 }
}
```