QUARANTINE MONITORING SYSTEM USING GEOFENCING

(IOT PROJECT REPORT)

TITLE **PAGE NO.**

MEMBERS:

KADAMBARI BHUJBAL (19BEE0141)
M. SAI SRIKAR (19BEE0121)
SHANT RAKSHIT (19BEE0154)
RAKSHIT RAJ (19BEE0159)

ABSTRACT

The aim of this project is to create a wearable device to allow the monitoring of the location of patients in quarantine, to make sure that they do not leave. The device created can be strapped onto the patients and will be able to track their location. The concept used to track is geo fencing.

- The hardware part of the project includes Node MCU to which a GPS module is connected and the values of latitude, longitude and altitude are getting printed in the serial monitor.
- Now the values of the latitude, longitude are used to calculate the distance from a pre declared latitude and longitude called the home latitude and home longitude.
- The variable called "Distance_from_home" is calculated and published in the ADAFRUIT IO server as feeds. These feeds are shown in a dashboard.
- Now deciding on the radius of fencing, a trigger can be set in the servers to give us notification.
- But since we all use mobile phones, it is practical to have a notification on our phone.
- That is achieved using IFTTT applets that can be used to monitor a feed from ADAFRUIT IO and give an alert on a pre declared condition.

INTRODUCTION

Coronavirus becomes officially a global pandemic due to the speed spreading off in various countries. An increasing number of infected with this disease causes the Inability problem to fully care in hospitals and afflict many doctors and nurses inside the hospitals. This project proposes a smart system that monitors the patients holding the Coronavirus remotely, in order to protect the lives of the health services members (like physicians and nurses, police,) from infection.

There are a lot of cases of people not following the quarantine protocols and are still leaving the hospitals and their homes. We are trying to do this project to try and create a viable solution to this problem by making a GPS tracking device which will enable hospital authorities to ensure that patients are not following protocols for any reason whatsoever. With this project we hope to give relief to public, hospital authorities and the unaffected population all over the world and some confidence of being safe.

We aim to achieve this by implementing the concept of geo fencing. A geofence is a virtual perimeter for a real-world geographic area. A geo-fence could be dynamically generated—as in a radius around a point location, or a geo-fence can be a predefined set of boundaries. The use of a geofence is called geofencing.

The person under quarantine is required to wear a wearable device or install an app on their phone which relays the location of the person at specific time intervals to the monitoring station.

The monitoring station sets up a virtual geofence around the home of the person under quarantine and if the person under quarantine moves out of the area of the geofencing then an alert is sent out to the monitoring station which can then track and return the violator to quarantine immediately thus preventing the further transmission and spread of the virus.

COMPONENTS USED

NODEMCU:

NodeMCU is an open-source Lua based firmware and development board specially targeted for IoT based Applications. It includes firmware that runs on the ESP8266 Wi-Fi SoC from Espressif Systems, and hardware which is based on the ESP-12 module.

The NodeMCU ESP8266 development board comes with the ESP-12E module containing ESP8266 chip having Tensilica Xtensa 32-bit LX106 RISC microprocessor. This microprocessor supports RTOS and operates at 80MHz to 160 MHz adjustable clock frequency. NodeMCU has 128 KB RAM and 4MB of Flash memory to store data and programs. Its high processing power with in-built Wi-Fi / Bluetooth and Deep Sleep Operating features make it ideal for IoT projects. NodeMCU can be powered using Micro USB jack and VIN pin (External Supply Pin). It supports UART, SPI, and I2C interface.

GPS MODULE:

The GPS QUESTAR TTL is a compact all-in-one GPS module solution intended for a broad range of Original Equipment Manufacturer (OEM) products, where fast and easy system integration and minimal development risk is required. The receiver continuously tracks all satellites in view and provides accurate satellite positioning data. The GPS QUESTAR TTL is optimized for applications requiring good performance, low cost, and maximum flexibility; suitable for a wide range of OEM configurations including handhelds, sensors, asset tracking, PDA-centric personal navigation system, and vehicle navigation products. Its 56 parallel channels and 4100 search bins provide fast satellite signal acquisition and short start-up time. Acquisition sensitivity of -140dBm and tracking sensitivity of -162dBm offers good navigation performance even in urban canyons having limited sky view. Satellite-based augmentation systems, such as WAAS and EGNOS, are supported to yield improved accuracy. USB-level serial interface is provided on the interface connector. Supply voltage of 3.8V~5.0V is supported.

ADAFRUIT:

Adafruit.io is a cloud service - that means it does not need to be managed by the user. We can connect to it over the Internet. It's meant primarily for storing and then retrieving data but it can do a lot more than just that.

- It displays your data in real-time, online
- The projects can be connected to the internet through adafruit. Control your project remotely and save all the data for analysis.
- Connect your project to other internet-enabled devices
- Another important aspect of Adafruit is it is available for free.

Adafruit IO's MQTT API exposes feed data using special topics. You can publish a new value for a feed to its topic, or you can subscribe to a feed's topic to be notified when the feed has a new value. Any one of the following topic forms is valid for a feed:

- (username)/feeds/ (feed name or key)
- (username)/f/ (feed name or key)

Where (username) is your Adafruit IO username (the same as specified when connecting to the MQTT server) and (feed name or key) is the feed's name or key. The smaller '/f/' path is provided as a convenience for small embedded clients that need to save memory.

IFTTT:

IFTTT derives its name from the programming conditional statement “if this, then that.” What the company provides is a software platform that connects apps, devices and services from different developers in order to trigger one or more automations involving those apps, devices and services.

The automations are accomplished via applets — which are sort of like macros that connect multiple apps to run automated tasks. You can turn on or off an applet using IFTTT’s website or mobile apps (and/or the mobile apps’ IFTTT widgets). You can also create your own applets or make variations of existing ones via IFTTT’s user-friendly, straightforward interface.

ARDUINO IDE:

The Arduino Integrated Development Environment (IDE) is a cross-platform application (for Windows, macOS, Linux) that is written in functions from C and C++. It is used to write and upload programs to Arduino compatible boards, but also, with the help of third-party cores, other vendor development boards.

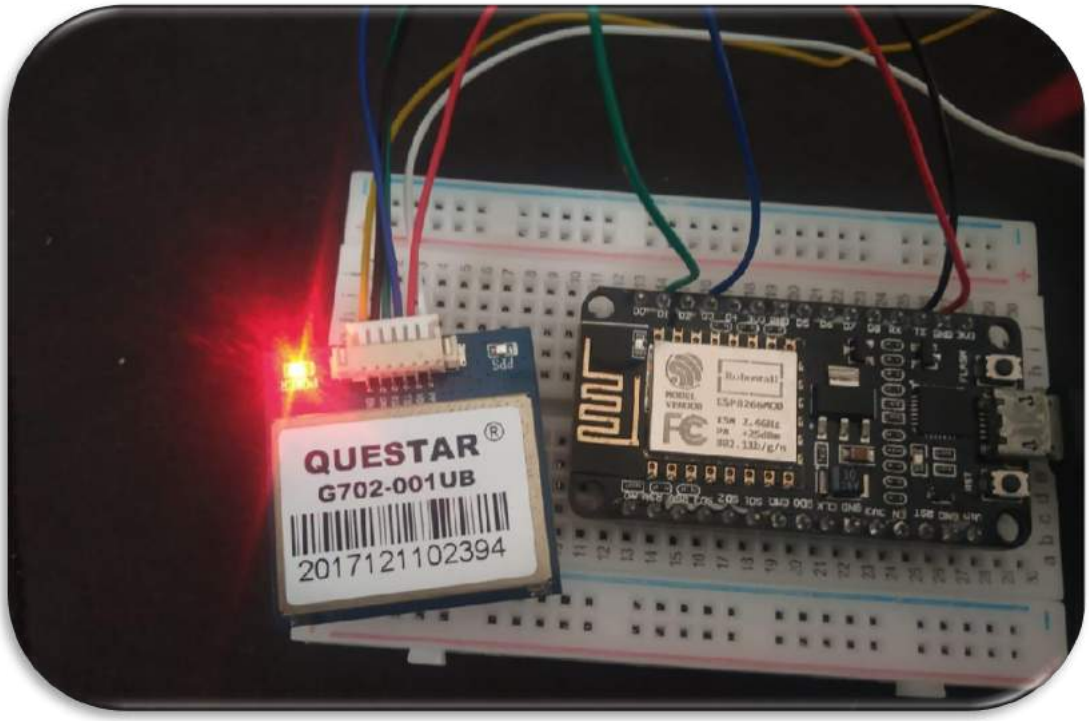
The source code for the IDE is released under the GNU General Public License, version 2. The Arduino IDE supports the languages C and C++ using special rules of code structuring. The Arduino IDE supplies a software library from the Wiring project, which provides many common input and output procedures. User-written code only requires two basic functions, for starting the sketch and the main program loop, that are compiled and linked with a program stub main() into an executable cyclic executive program with the GNU toolchain, also included with the IDE distribution. The Arduino IDE employs the program avrdude to convert the executable code into a text file in hexadecimal encoding that is loaded into the Arduino board by a loader program in the board's firmware.

IMPLEMENTATION PROCEDURE:

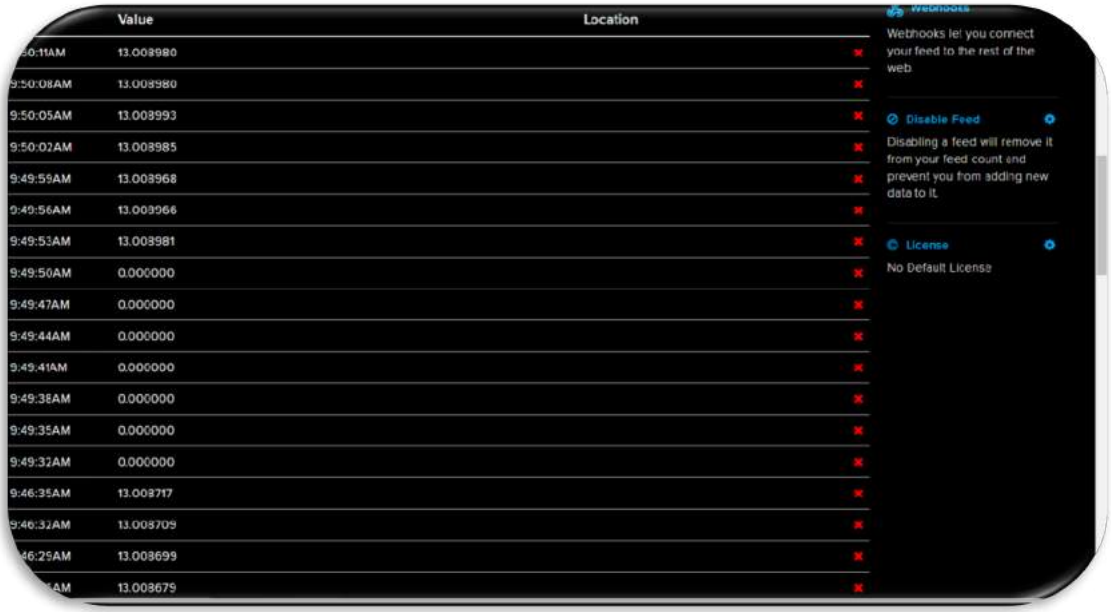
1. The circuit consists of a NodeMCU, GPS module and a buzzer.
2. The GPS module is connected serially to the NodeMCU. We use a NodeMCU here because of its wifi module.
3. Once the circuit is done, upload the code using Arduino IDE.
4. Once the code start running, we will be able to receive live data from the GPS module in the adafruit server.
5. There are three feeds in the server, one gpslat, which records the latitude of the patient, gpslng, which records the longitude of the patient and gpslatlng, which records the distance from the home centre initialized in the code.
6. We also have a dashboard on the server to display the data recorded from the gpslatlng feed.
7. The dashboard records and displays the location of the patient on a world map in satellite view.
8. It also records the time interval between the recorded location in a graph, so we know the latest location immediately.
9. We use IFTTT platform to make an applet that is used to send notification to the concerned authority.
10. The IFTTT platform checks for the distance recorded in the feed and sends a notification when it exceeds a set distance, which can be changed according to the administrator.
11. Alternatively, we could also set a trigger on Adafruit server, which will send an email to a concerned authority.

RESULTS AND CONCLUSIONS

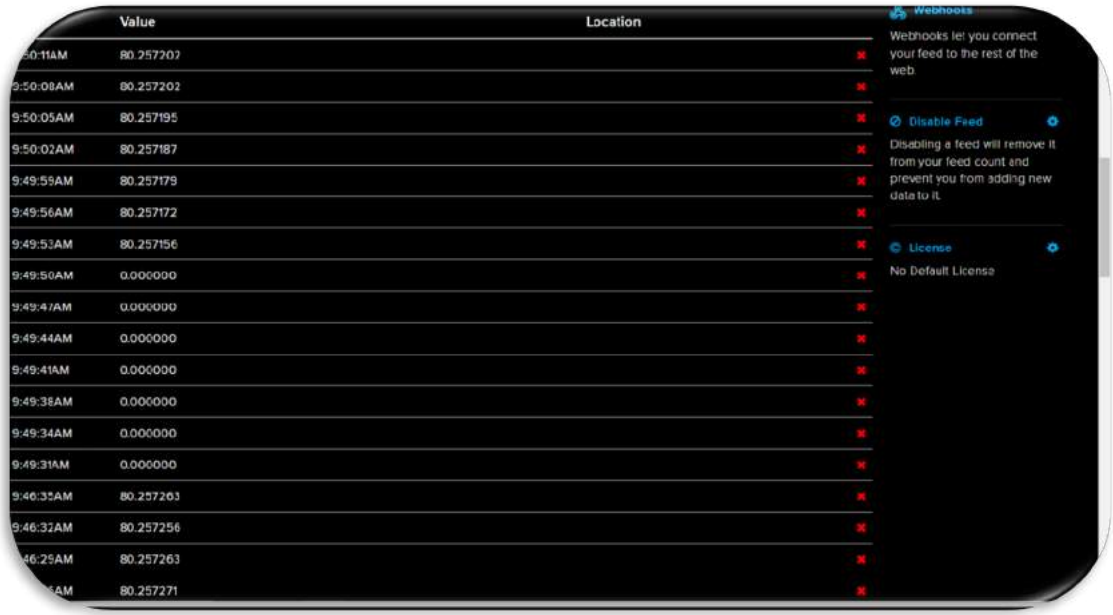
Circuit:



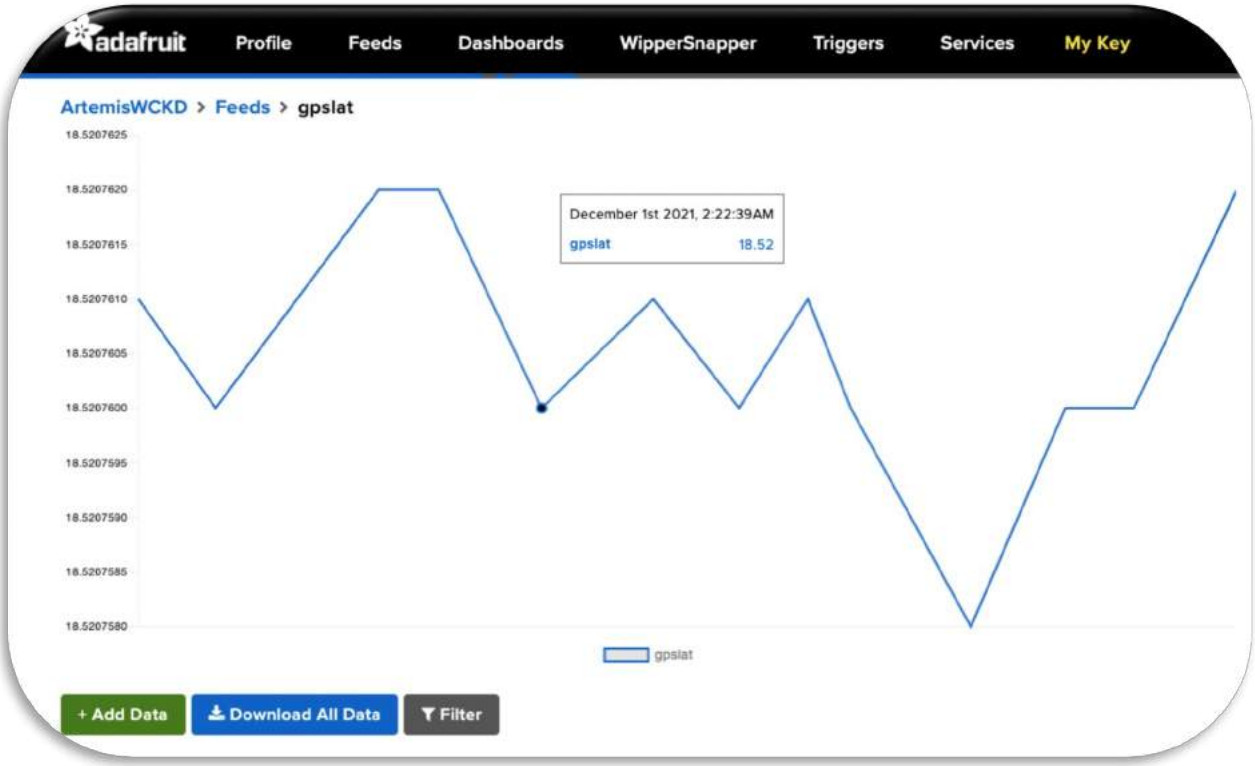
Latitude values:



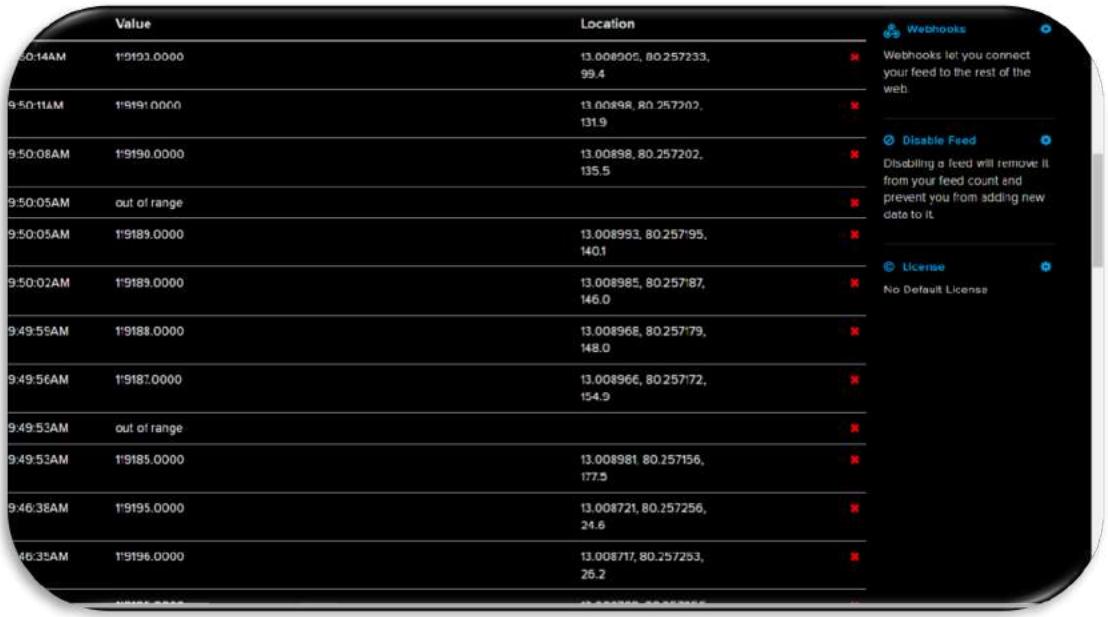
Longitude values:



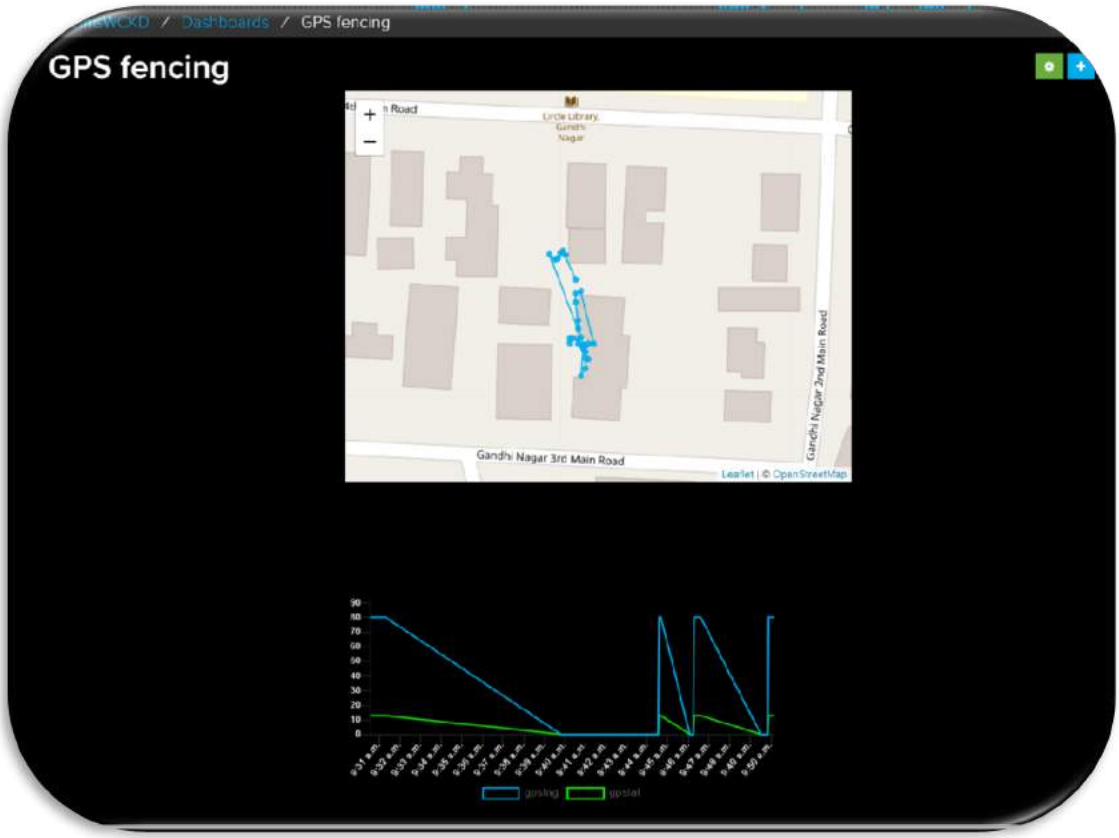
The distance from the centre is graphed:



The distance is recorded in the feeds:

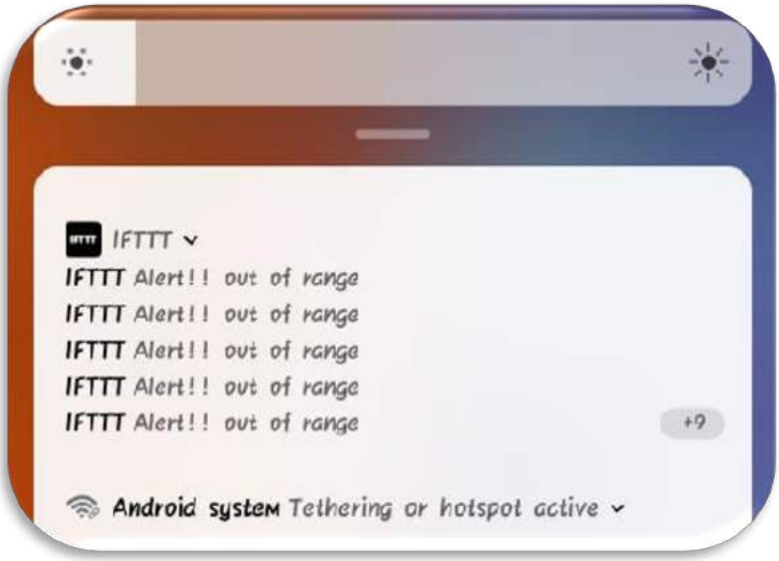


The location is marked on the map:



IFTTT activity log:

Notification on phone:



In this project we have successfully designed and implemented a quarantine monitoring system using node mcu and adafruit server based on the principle of Geo Fencing.

We have built this project in order to successfully impose the quarantine system in order to curb the spread of the virus and this can also be implemented in other useful sectors.

RECOMMENDATIONS:

This project can be improved by incorporating purpose built Printed Circuit Boards (PCBs) which will make it truly wearable.

Also the usage of highly powerful GPS modules can make the tracking very accurate and can also be used to increase the range of the tracker. So even if the patients manage to circumvent the hospital security and leave, they can still be tracked down.

The concept and implementation used here can be utilized in areas such as, correctional facilities, mental wards, hospitals, rehabilitation centres.

The data can be furthered secured by using private servers, to keep the data safe.

The Speed of the notification system can be brought down by using LTE or 5G technologies for lower latency and faster transmission of data.

The cost of production of each individual unit can be brought down by the Economies of Scale.

APPENDIX:

Code:

```
#include "Adafruit_MQTT.h"                    // Adafruit MQTT library

#include "Adafruit_MQTT_Client.h"             // Adafruit MQTT library


#include "ESP8266WiFi.h"                      // ESP8266 library

//#include <Adafruit_ssd1306syp.h>            // Adafruit Oled library for Serial


#include <TinyGPS++.h>                        // Tiny GPS Plus Library

#include <SoftwareSerial.h>                   // Software Serial Library so we can use Pins
for communication with the GPS module


//#define SDA_PIN 4                           // uses GPIO pins 4(SDA) and 5(SCL) of the
ESP8266 Adafruit Feather

//#define SCL_PIN 5                           // also known as pins D1(SCL) and D2(SDA)
of the NodeMCU ESP-12

//Adafruit_ssd1306syp Serial(SDA_PIN,SCL_PIN);          // Set OLED Serial pins


//static const int RXPin =D2, TXPin =D1;      // Ublox 6m GPS module to pins 12 and
13

static const uint32_t GPSBaud = 9600;        // Ublox GPS default Baud Rate is 9600


TinyGPSPlus gps;                              // Create an Instance of the TinyGPS++ object
called gps

SoftwareSerial ss(D3,D1);                     // The serial connection to the GPS device
```

```
const double HOME_LAT = 12.988740;//12.968675;                    // Enter Your Latitude and Longitude here
const double HOME_LNG = 80.252823;//79.158234;                    // to track how far away the "person" is away from Home

/************************ WiFi Access Point *********************************/

#define WLAN_SSID       [redacted]      // Enter Your router SSID
#define WLAN_PASS       [redacted]                  // Enter Your router Password

/************************ Adafruit.io Setup *********************************/

#define AIO_SERVER      "io.adafruit.com"
#define AIO_SERVERPORT  1883                   // use 8883 for SSL
#define AIO_USERNAME    "ArtemisWCKD"                     // Enter Your Adafruit IO Username
#define AIO_KEY         "b457b476121543c59a88dfe200f233a2" // Enter Your Adafruit IO Key

/************ Global State (you don't need to change this!) ******************/

WiFiClient client;                          // Create an ESP8266 WiFiClient class to connect to the MQTT server.

const char MQTT_SERVER[] PROGMEM    = AIO_SERVER;           // Store the MQTT server, username, and password in flash memory.
```

```
const char MQTT_USERNAME[] PROGMEM = AIO_USERNAME;

const char MQTT_PASSWORD[] PROGMEM = AIO_KEY;


// Setup the MQTT client class by passing in the WiFi client and MQTT server and login details.

Adafruit_MQTT_Client mqtt(&client, MQTT_SERVER, AIO_SERVERPORT, MQTT_USERNAME, MQTT_PASSWORD);


/****************************** Feeds ***************************************/


// Setup a feed called 'gpslat' for publishing.

// Notice MQTT paths for AIO follow the form: <username>/feeds/<feedname> // This feed is not needed, only setup if you want to see it

const char gpslat_FEED[] PROGMEM = AIO_USERNAME "/feeds/gpslat";

Adafruit_MQTT_Publish gpslat = Adafruit_MQTT_Publish(&mqtt, gpslat_FEED);


// Setup a feed called 'gpslng' for publishing.

// Notice MQTT paths for AIO follow the form: <username>/feeds/<feedname> // This feed is not needed, only setup if you want to see it

const char gpslng_FEED[] PROGMEM = AIO_USERNAME "/feeds/gpslng";

Adafruit_MQTT_Publish gpslng = Adafruit_MQTT_Publish(&mqtt, gpslng_FEED);


// Setup a feed called 'gps' for publishing.

// Notice MQTT paths for AIO follow the form: <username>/feeds/<feedname>

const char gps_FEED[] PROGMEM = AIO_USERNAME "/feeds/gpslatlng/csv";        // CSV = commas seperated values
```

```
Adafruit_MQTT_Publish gpslatlng = Adafruit_MQTT_Publish(&mqtt, gps_FEED);

/*************************************************/

void setup()
{
  Serial.begin(115200);                    // Setup Serial Comm for Serial Monitor @115200
baud
  WiFi.mode(WIFI_STA);                     // Setup ESP8266 as a wifi station
  WiFi.disconnect();                       // Disconnect if needed
  delay(100);                              // short delay

 /* Serial.initialize(); // init OLED Serial
  //Serial.clear();                        // Clear OLED Serial
  Serial.setTextSize(1);                   // Set OLED text size to small
  Serial.setTextColor(WHITE);              // Set OLED color to White
  //Serial.setCursor(0,0);                 // Set cursor to 0,0
  Serial.println(" Adafruit IO GPS");
  Serial.println("      Tracker");
  Serial.print("....................");
  //Serial.update();                       // Update Serial
  delay(1000);   */                        // Pause X seconds

  ss.begin(9600);                          // Set Software Serial Comm Speed to 9600
```

```
  Serial.print("Connecting to WiFi");
  //Serial.update();

  WiFi.begin(WLAN_SSID, WLAN_PASS);                // Start a WiFi connection and enter SSID and Password
     while (WiFi.status() != WL_CONNECTED)
       {                                  // While waiting on wifi connection, Serial "..."
         delay(500);
         Serial.print(".");
         //Serial.update();
       }
         Serial.println("Connected");
         //Serial.update();
}                                         // End Setup

void loop() {

  smartDelay(500);                        // Update GPS data TinyGPS needs to be fed often
  MQTT_connect();                          // Run Procedure to connect to Adafruit IO MQTT

  float Distance_To_Home;                    // variable to store Distance to Home
  float GPSlat = (gps.location.lat());       // variable to store latitude
  float GPSlng = (gps.location.lng());         // variable to store longitude
  float GPSalt = (gps.altitude.feet());      // variable to store altitude
```

```
 Distance_To_Home = (unsigned long)TinyGPSPlus::distanceBetween(gps.location.lat(),gps.location.lng(),HOME_LAT, HOME_LNG); //Query Tiny GPS to Calculate Distance to Home


 //Serial.clear();
 //Serial.setCursor(0,0);
 Serial.println(F(" GPS Tracking"));
 Serial.println(" --------------------");
 //Serial.update();


 Serial.print("GPS Lat: ");
 Serial.println(gps.location.lat(), 6);              // Serial latitude to 6 decimal points
 Serial.print("GPS Lon: ");
 Serial.println(gps.location.lng(), 6);              // Serial longitude to 6 decimal points
 Serial.print("Distance: ");
 Serial.println(Distance_To_Home);                   // Distance to Home measured in Meters
 //Serial.update();


 // ********************** Combine Data to send to Adafruit IO ********************************

 // Here we need to combine Speed, Latitude, Longitude, Altitude into a string variable buffer to send to Adafruit


        char gpsbuffer[30];                          // Combine Latitude, Longitude, Altitude into a buffer of size X
```

```
        char *p = gpsbuffer;                    // Create a buffer to store GPS information to upload
to Adafruit IO


        dtostrf(Distance_To_Home,  3, 4, p);            // Convert Distance to Home to a String
Variable and add it to the buffer
        p += strlen(p);
        p[0] = ','; p++;


        dtostrf(GPSlat, 3, 6, p);                 // Convert GPSlat(latitude) to a String variable and
add it to the buffer
        p += strlen(p);
        p[0] = ','; p++;


        dtostrf(GPSlng, 3, 6, p);                  // Convert GPSlng(longitude) to a String variable
and add it to the buffer
        p += strlen(p);
        p[0] = ','; p++;


        dtostrf(GPSalt, 2, 1, p);                 // Convert GPSalt(altimeter) to a String variableand
add it to the buffer
        p += strlen(p);


        p[0] = 0;                               // null terminate, end of buffer array


        if ((GPSlng != 0) && (GPSlat  != 0))          // If GPS longitude or latitude do not equal
zero then Publish
```

```
        {
        Serial.println("Sending GPS Data ");
        //Serial.update();
        gpslatlng.publish(gpsbuffer);            // publish Combined Data to Adafruit IO
        Serial.println(gpsbuffer);
        }

      gpslng.publish(GPSlng,6);                  // Publish the GPS longitude to Adafruit IO

      if (! gpslat.publish(GPSlat,6))            // Publish the GPS latitude to Adafruit IO
        {
          Serial.println(F("Failed"));          // If it failed to publish, print Failed
        } else
          {
           //Serial.println(gpsbuffer);
           Serial.println(F("Data Sent!"));
           }
    //Serial.update();
    delay(1000);

if (millis() > 5000 && gps.charsProcessed() < 10)
  Serial.println(F("No GPS data received: check wiring"));

// Wait a bit before scanning again
```

```
  Serial.print("Pausing...");
  //Serial.update();
  smartDelay(500);                              // Feed TinyGPS constantly
  delay(1000);
  if (Distance_To_Home>200)
   pinMode(D2,HIGH);
}


// ********** Smart delay - used to feed TinyGPS **********

static void smartDelay(unsigned long ms)
{
  unsigned long start = millis();
  do
  {
    while (ss.available())
      gps.encode(ss.read());
  } while (millis() - start < ms);
}


// ********** MQTT Connect - connects with Adafruit IO **********
void MQTT_connect() {
```

```
  int8_t ret;
  if (mqtt.connected()) {  return; }                    // Stop and return to Main Loop if already connected to Adafruit IO
  Serial.print("Connecting to MQTT... ");
  //Serial.update();

  uint8_t retries = 3;
  while ((ret = mqtt.connect()) != 0) {                 // Connect to Adafruit, Adafruit will return 0 if connected
       Serial.println(mqtt.connectErrorString(ret)); // Serial Adafruits response
       Serial.println("Retrying MQTT...");
       mqtt.disconnect();
       //Serial.update();
       delay(5000);                                  // wait X seconds
       retries--;
       if (retries == 0) {                           // basically die and wait for WatchDogTimer to reset me
         while (1);
       }
  }
  Serial.println("MQTT Connected!");
  //Serial.update();
  delay(1000);
}
```

REFERENCES

[1] Axel Küpper, Ulrich Bareth, and Behrend Freese, "Geofencing and Background Tracking – The Next Features in LBSs", Jahrestagung der Gesellschaft für Informatik , 4.-7.10.2011, Berlin

[2] Prof. Swati Shinde, Tanveer Shaikh, Anilkumar Vandha, Harshil Sheth, "Location-based Dynamic Advertisements Structure for Public Transit Systems", International Journal of Engineering Research & Technology (IJERT),ISSN: 2278-0181, IJERTV4IS030322, Vol. 4 Issue 03, March-2015

[3] Yoshitaka NAKAMURA, Masashi SEKIYA, Kazuaki HONDA, and Osamu TAKAHASHI, "An effective power saving method in Geo-fencing service using temperature sensors", School of Systems Information Science, Future University Hakodate, Japan ‡IDY Corporation, Japan

[4] Diksha Kewat, Vaishnavi Tonpe, Kiran Baxani, Dr. Sanjay Sharma, "Geofencing for disaster information system", International Journal of Advance Research, Ideas and Innovations in Technology, ISSN: 2454-132X

[5] Mohammad Nasajpour, Seyedamin Pouriyeh, Reza M. Pariziy, Mohsen Dorodchiz, Maria Valero, Hamid R. Arabnia, "Internet of Things for Current COVID-19 and Future Pandemics: An Exploratory Study", Research Gate