DEVELOPMENT PART-1
IoT ENABLED PUBLIC TRANSPORT OPTIMIZATION SYSTEM

PASSENGER COUNTING: Arduino-Based Directional Entry/Exit Notification System for Passenger Counting in Public Transport.

- The Arduino-Based Directional Entry/Exit Notification System offers a groundbreaking solution to streamline passenger counting in public transport, enhancing both the accuracy and effectiveness of tracking passenger flow.

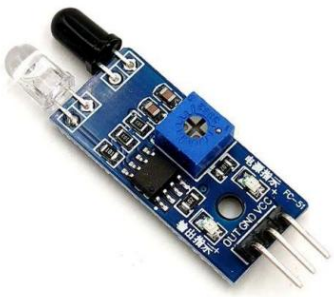
- This innovative system leverages the power of Arduino micro-controllers and infrared (IR) sensors to create a real-time and comprehensive method for monitoring passenger movements. By accurately detecting entry and exit events.

- As passengers embark and disembark from buses, trains, and other modes of public transportation, the system's IR sensors work in unison to record these actions. This data is then meticulously processed and analyzed, offering transit operators a detailed view of passenger flow dynamics, including peak travel times, station or stop-specific trends, and even occupancy levels in real time. Such information empowers transport management authorities to respond swiftly to changing demand, ensure the safety of passengers, and provide a seamless commuting experience.
- The directional entry/exit notification system paves the way for intelligent transit systems, paving the path towards improved operational efficiency and sustainability. Furthermore, it enhances the

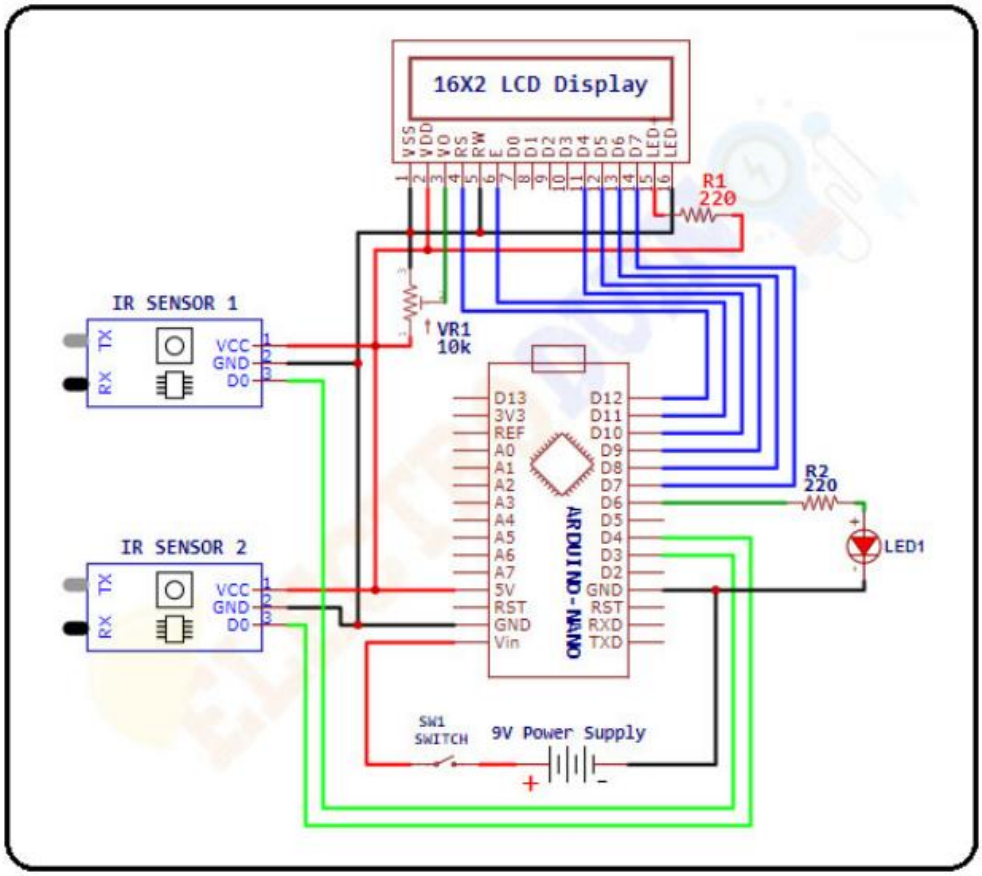
ability to assess the effectiveness of passenger counting measures for fare collection and ensures an equitable and fair billing process.

COMPONENTS :

- ARDUINO UNO ATmega328P
- INFRARED (IR) SESNORS (2X)
- LED
- BUZZER
- RESISTOR
- LCD DISPLAY (16 X 2)
- BREADBORD
- JUMPER CABLES
- POWER SOURCE
- PC WITH ARUINO IDE



SYSTEM DESIGN :



CODE.py:

To make the arduino uno board suppotive for python code the "Firamata " protocol and libraries pyfirmata must be used.

```
pip install pyfirmata
from pyfirmata import Arduino, util

# Define the serial port of your Arduino (check your Arduino IDE)
port = COM08

# Initialize the Arduino board
board = Arduino(port)

# Define the pin numbers
sensor_1_pin = 2
sensor_2_pin = 3
led_pin = 13  # The built-in LED pin on most Arduino boards

# Set up the pins
sensor_1 = board.get_pin('d:{}:i'.format(sensor_1_pin))
```

```
sensor_2 = board.get_pin('d:{}:i'.format(sensor_2_pin))
led = board.get_pin('d:{}:o'.format(led_pin))

# Initialize variables
peopleInside = 0

it = util.Iterator(board)
it.start()

try:
    while True:
        sensor1State = sensor_1.read()
        sensor2State = sensor_2.read()

        if sensor1State == 1 and sensor2State == 0:
            # Object entered the room
            led.write(1)  # Turn on the LED
            board.pass_time(10)  # Wait for 10 seconds
            led.write(0)  # Turn off the LED
            peopleInside += 1  # Increment people count

        elif sensor1State == 0 and sensor2State == 1:
            # Object exited the room
            # You can add code to control the buzzer here
            peopleInside -= 1  # Decrement people count

        print("People Inside:", peopleInside)

except KeyboardInterrupt:
    board.exit()
```

Now for transferring the data collected be the passenger counter to the MQTT cloud the following code can be used,

```
import paho.mqtt.client as mqtt
import time
import serial

# HiveMQ Cloud MQTT broker settings
```

```
broker_address =
 "ed342ccf55c1484eb534c8c92861048b.s2.eu.hivemq.cloud"
port = 8883  # Use 8883 for TLS encrypted connection
username = "pto"
password = “PTO355265@"
client_id = "aiyengar"  # Specify your desired Client ID here

# Initialize the serial connection to Arduino
ser = serial.Serial('/dev/ttyACM0', 9600)  # Replace with the correct
 port

# Callback when the client connects to the broker
def on_connect(client, userdata, flags, rc):
    print("Connected with result code " + str(rc)

# Callback when a message is received
def on_message(client, userdata, msg):
    print("Received message on topic " + msg.topic + ": " +
 str(msg.payload))

# Create an MQTT client instance
client = mqtt.Client()

# Set the callbacks
client.on_connect = on_connect
client.on_message = on_message

# Set the username and password for authentication
client.username_pw_set(username, password)

# Enable TLS (Transport Layer Security) for secure communication
client.tls_set()

# Connect to the MQTT broker
client.connect(broker_address, port, 60)

# Start the MQTT client loop (this will keep the script running)
client.loop_start()

# Keep the script running
while True:
```

```
    try:
        arduino_data = ser.readline().strip().decode('utf-8')
        if arduino_data.startswith("People Inside: "):
            people_count = arduino_data.replace("People Inside: ", "")
            client.publish("ESP32", people_count, qos=1)  # qos=1 for
At least once
    except Exception as e:
        print("Error publishing message:", str(e))
    time.sleep(1)
```

This code connects to the HiveMQ Cloud MQTT broker, reads the data sent by your Arduino code via the serial connection, and publishes it to the MQTT .

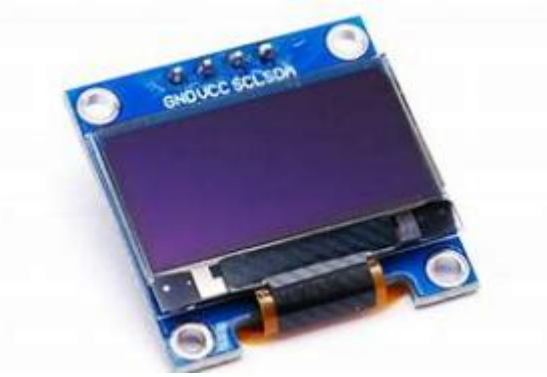
LOCATION TRACKING :

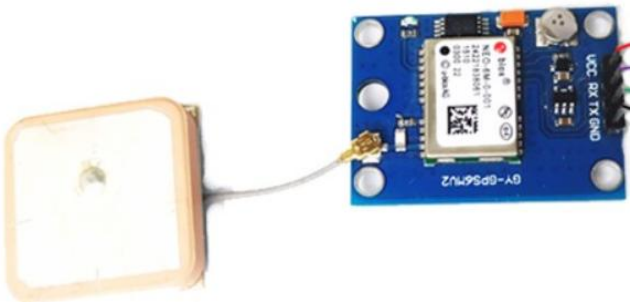
ESP32 GPS Tracker- IoT based location Tracking System

- An "ESP32 GPS Tracker" is an Internet of Things (IoT) device designed to provide real-time location tracking and monitoring capabilities. This system leverages the ESP32 microcontroller, a versatile and widely-used IoT development platform, along with a GPS (Global Positioning System) module to accurately determine and transmit the device's geographical coordinates. It is an example of a powerful and practical application of IoT technology.
- The primary purpose of an ESP32 GPS Tracker is to collect location data and relay it to a central server or cloud-based platform, where the data can be visualized and analyzed in real-time.
- The ESP32 GPS Tracker integrates GPS technology with the ESP32's wireless capabilities, allowing it to communicate over Wi-Fi, Bluetooth, or other wireless protocols. It can also include additional sensors for collecting environmental or health data. Furthermore, this system often sends data to a cloud-based platform where the information is processed and made accessible through web or mobile applications

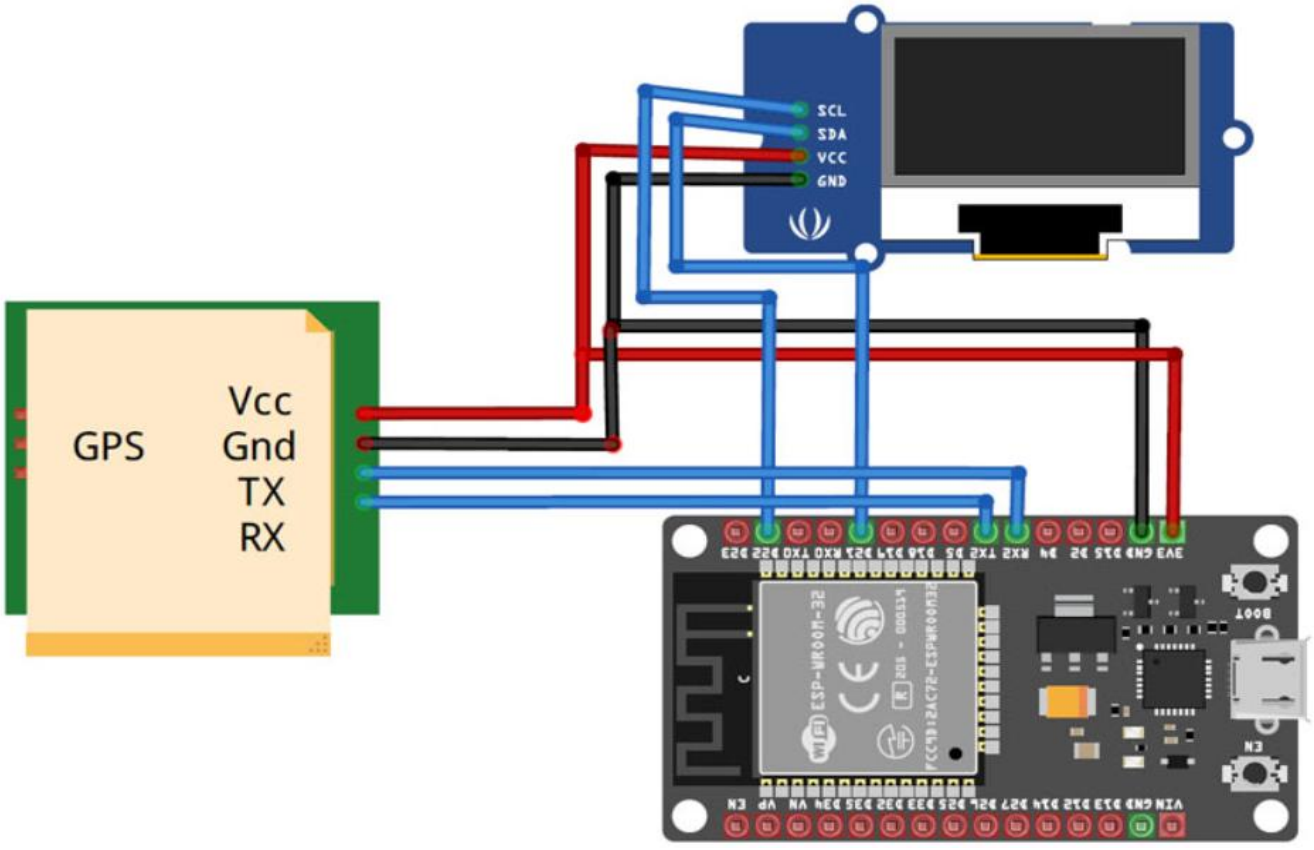
COMPONENTS:

- ESP32
- GPS Module
- OLED Display Module
- Jumper Wires
- Breadboard





SYSTEM DESIGN:



CODE.PY:

```
import time
import machine
import ubinascii
import ujson
import urequests
from machine import I2C, Pin, UART
import ssd1306
from umqtt.simple import MQTTClient

# WiFi and Blynk credentials
ssid = "Galaxy-M20"
passkey = "ac312129"
auth = "loPrSaL0eQFY9clcQ518R1SmYsRVC0eV"

# MQTT broker settings
mqtt_server =
 "ed342ccf55c1484eb534c8c92861048b.s2.eu.hivemq.cloud"
mqtt_port = 8883
mqtt_username = "aiyengar"
mqtt_password = "Mh12hn4226!!!"

# Initialize UART for GPS
uart = UART(1, baudrate=9600, tx=16, rx=17)

# Initialize OLED display
i2c = I2C(scl=Pin(22), sda=Pin(21))
oled = ssd1306.SSD1306_I2C(128, 64, i2c)

# Initialize MQTT client
client_id = ubinascii.hexlify(machine.unique_id())
client = MQTTClient(client_id, mqtt_server, port=mqtt_port,
 user=mqtt_username, password=mqtt_password)

# Connect to Wi-Fi
import network
sta_if = network.WLAN(network.STA_IF)
```

```
sta_if.active(True)
sta_if.connect(ssid, passkey)
while not sta_if.isconnected():
    pass

def display_gps(latitude, longitude):
    oled.fill(0)
    oled.text("Latitude: {:.6f}".format(latitude), 0, 0)
    oled.text("Longitude: {:.6f}".format(longitude), 0, 16)
    oled.show()

client.connect()

while True:
    gps_data = uart.readline()
    if gps_data and b'GGA' in gps_data:
        gps_data = gps_data.decode()
        lat_idx = gps_data.find(',')
        lng_idx = gps_data.find(',', lat_idx + 1)

        if lat_idx >= 0 and lng_idx >= 0:
            latitude = float(gps_data[lat_idx + 1:lat_idx + 10])
            longitude = float(gps_data[lng_idx + 1:lng_idx + 10])

            display_gps(latitude, longitude)

            location_data = "Latitude: {:.6f}, Longitude:
 {:.6f}".format(latitude, longitude)
            client.publish("ESP32", location_data)

    time.sleep(1)
```

****