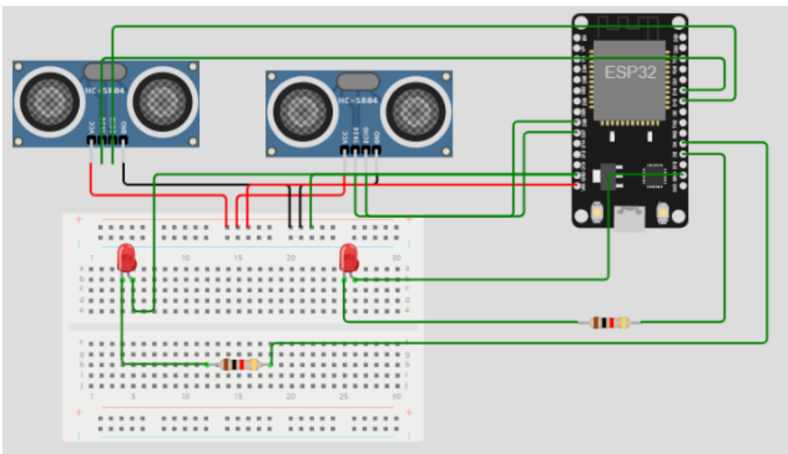
PUBLIC TRANSPORT OPTIMIZATION

Phase-3: Deploy IoT sensors (e.g., GPS, passenger counters) in public transportation vehicles to gather data using Python Script.

Introduction:

In this document we explain the components used in Public Transport System and how IoT sensors sense the data in Public Transport Vehicles for every second using **Python Codes**. The **ESP32 Microcontroller** used to control the flow of operations and send the data directly to the Cloud Server (e.g., MQTT Broker, Firebase, Blynk Server…). The two **HC-SR04-Ultrasonic Sensors** used to detect the passengers enter/exit operation at the door way of the vehicle. The **NEO-M6 GPS Module (**or other GPS modules**)** use to predict the real time location (using Latitude and Longitude positions). Using the real time location details for every second used to calculate the arrival of time of each station. The **SSD1306 Display** used to show the details.

Passenger Counter Circuit:

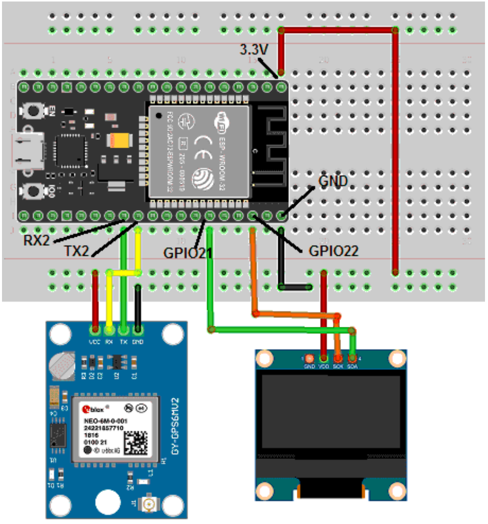


Python Code:

```
import time
from machine import Pin

trigger1 = Pin(19, Pin.OUT)
echo1 = Pin(18, Pin.IN)
trigger2 = Pin(27, Pin.OUT)
echo2 = Pin(26, Pin.IN)

distance_detection = 50  #depens on distance/length in cm

#Initial Enter/Exit Passengers
enter = 0
exit = 0
Total=0

# Function to measure distance
def measure_distance1():
	pulse_duration1=0
	pulse_start1=0
	pulse_end1=0
	distance1 = 50

	# Send a 10us pulse on the trigger pin
	trigger1.value(1)
	time.sleep_us(10)
	trigger1.value(0)

	# Wait for the echo pin to go high
	while echo1.value() == 0:
		pulse_start1 = time.ticks_us()

	# Wait for the echo pin to go low
	while echo1.value() == 1:
		pulse_end1 = time.ticks_us()

	# Calculate the pulse duration and convert to distance (in centimeters)
	pulse_duration1 = time.ticks_diff(pulse_end1, pulse_start1)
	distance1 = (pulse_duration1 / 2) / (29.1)   # Speed of sound in air is 
approximately 343 meters per second

	return distance1
```

```
def measure_distance2():
    pulse_duration2=0
    distance2 = 50
    # Send a 10us pulse on the trigger pin
    trigger2.value(1)
    time.sleep_us(10)
    trigger2.value(0)
    # Wait for the echo pin to go high
    while echo2.value() == 0:
        pulse_start2 = time.ticks_us()
    # Wait for the echo pin to go low
    while echo2.value() == 1:
        pulse_end2 = time.ticks_us()
    # Calculate the pulse duration and convert to distance (in centimeters)
    pulse_duration2 = time.ticks_diff(pulse_end2, pulse_start2)
    distance2 = (pulse_duration2 / 2) / 29.1  # Speed of sound in air is
approximately 343 m/s
    return distance2

while True:
    distance1 = measure_distance1()
    print("Distance1_US1: {:.2f} cm".format(distance1))
    distance2 = measure_distance2()
    print("Distance2_US2: {:.2f} cm".format(distance2))
    if (distance1 < distance_detection):
        print("Passenger Enter!")
        Total+=1
        Pin(4,Pin.OUT).on()
        time.sleep(5)
        Pin(4,Pin.OUT).off()
        time.sleep(5)
    elif (distance2 < distance_detection):
        print("Passenger Exit!")
        Total-=1
        Pin(2,Pin.OUT).on()
        time.sleep(5)
        Pin(2,Pin.OUT).off()
        time.sleep(5)
    else:
        Pin(4,Pin.OUT).off()
        time.sleep(3)
        Pin(2,Pin.OUT).off()
        time.sleep(3)
    print("Total Passengers: {: }".format(Total))
    time.sleep(5) # Wait for a second before taking another readings
```

GPS Circuit:




```
import time
import machine
from micropyGPS import MicropyGPS
from machine import Pin, I2C
import ssd1306
import _thread
import time
i2c = I2C(-1, scl=Pin(22), sda=Pin(21))
oled_width = 128
oled_height = 64
oled = ssd1306.SSD1306_I2C(oled_width, oled_height, i2c)
oled.text('Carte', 0, 0)  # Afficher les deux mots ''
oled.text('ESP32', 0, 10)
oled.show()
def main():
	uart = machine.UART(1, rx=16, tx=17, baudrate=9600, bits=8,
parity=None, stop=1, timeout=5000, rxbuf=1024)
	gps = MicropyGPS()
	latitudes= []
	longitudes= []
	timestamps= []
	speeds= []
	i=0
	j=0
	def haversine(lat1, lon1, lat2, lon2):
		# Radius of the Earth in kilometres
		R = 6371
		# Convert latitude and longitude from degrees to radian
		lat1 = math.radians(lat1)
```

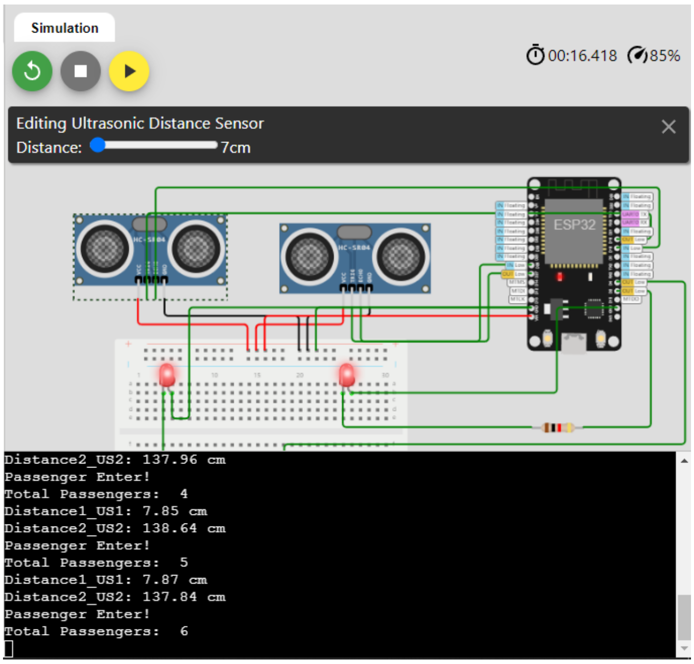
```
            lon1 = math.radians(lon1)
            lat2 = math.radians(lat2)
            lon2 = math.radians(lon2)

            # Haversine formula
            dlat = lat2 - lat1
            dlon = lon2 - lon1
            a = math.sin(dlat/2)**2 + math.cos(lat1) * math.cos(lat2) *
math.sin(dlon/2)**2
            c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
            distance = R * c

            return distance

      def calculate_speed(latitudes, longitudes, timestamps):
            speeds = []
            j=i-1
            lat1 = latitudes[i]
            lon1 = longitude[i]
            lat2 = latitudes[j]
            lon2 = longitudes[j]
            time_interval = timestamps[i] - timestamps[j]
            distance = haversine(lat1, lon1, lat2, lon2)
            speed = distance / time_interval
            speeds.append(speed)
            return speeds
      while True:
            buf = uart.readline()
            if uart.any():
                  for char in buf:
                        gps.update(chr(char))
                  print('UTC Timestamp:', gps.timestamp)
                  timestamps.append(gps.timestamp)
                  print('Date:', gps.date_string('long'))
                  print('Latitude:', gps.latitude)
                  latitudes.append(gps.latitude)
                  print('Longitude:', gps.longitude_string())
                  longitudes.append(gps.longitude)
                  if i=>1:
                        print("Speed:",calculate_speed(latitudes, longitudes,
      timestamps))
                  print('Horizontal Dilution of Precision:', gps.hdop)
                  print('Altitude:', gps.altitude)
                  print('Satellites:', gps.satellites_in_use)
                  print()
```

```
                i+=1
                oled.text("{:02d}:{:02d}:{:02.0f}".format(gps.timestamp[0], gps.timestamp[1], gps.timestamp[2]),  0, y)
                y += dy
                oled.text("Lat:{}{:3d}'{:02.4f}".format(gps.latitude[2], gps.latitude[0], gps.latitude[1]),  0, y)
                y += dy
                oled.text("Lon:{}{:3d}'{:02.4f}".format(gps.longitude[2], gps.longitude[0], gps.longitude[1]),  0, y)
                y += dy
                oled.text("Alt:{:0.0f}ft".format(gps.altitude*1000 / (12*25.4)), 0,y)
                y += dy
                oled.text("HDP:{:0.2f}".format(gps.hdop),  0, y)
                oled.show()
def startGPSthread():
    _thread.start_new_thread(main, ())

if __name__ == "__main__":
    print('...running main, GPS testing')
    main()
```

#NOTE: we must install package (library) micropyGPS from python packages (i.e., from GitHub)

Simulation Output:

Passenger Entrance:



Passenger Exit: