**PHASE -3**

**PUBLIC TRANSPORT OPTIMIZATION**

*In this technology project you will begin building your project by deploying IoT devices and then developing a Python script on the IoT devices as per the project requirement. After performing the relevant activities create a document around it and share the same for assessment.*

**Deployed IOT devices:**

1. **GPS Module**

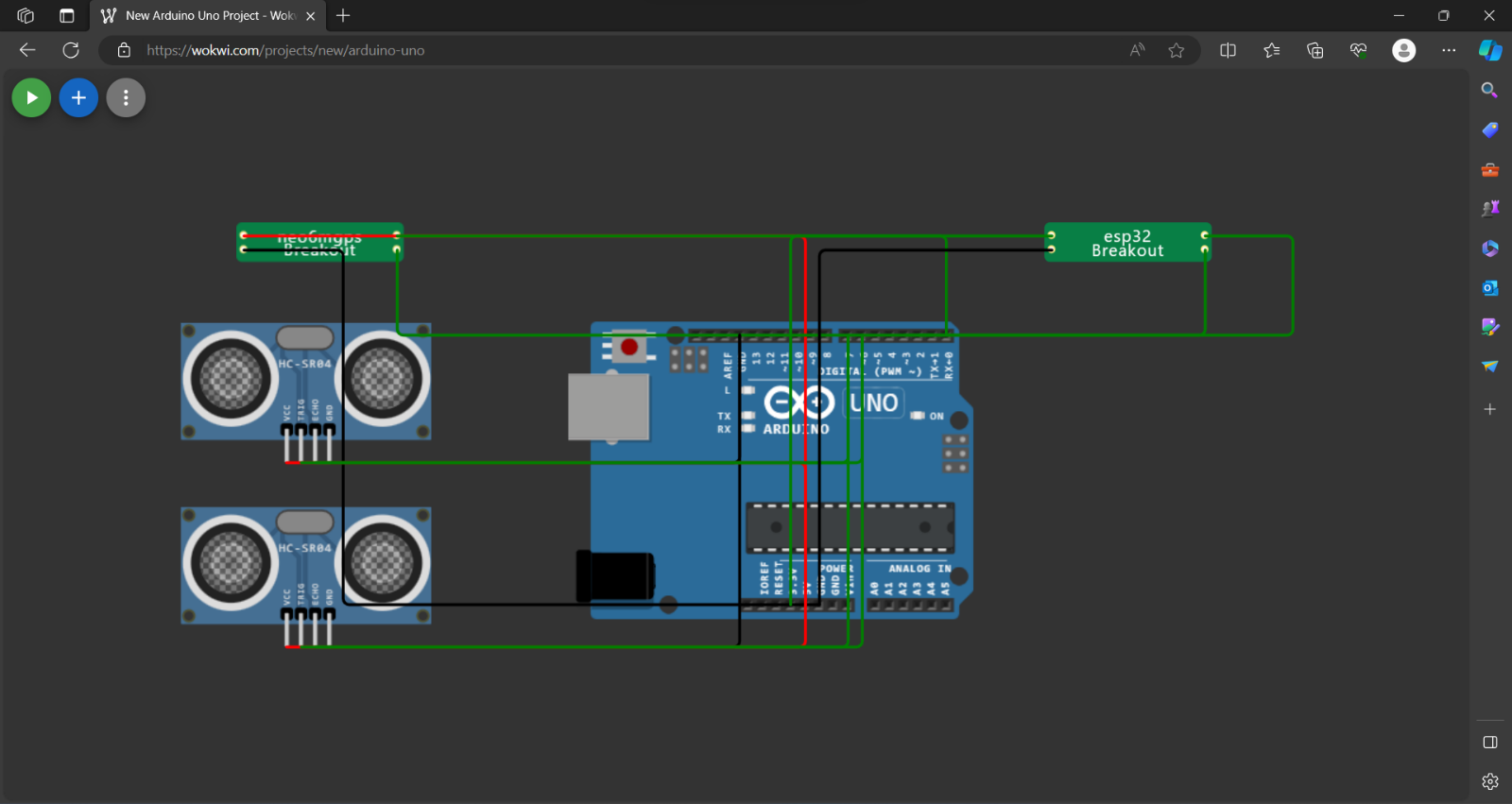
GPS (Global Position System) used for positioning and tracking buses based on satellite communication. GPS satellites cover the entire earth at all times. To get accurate GPS location data, there should be a minimum of three satellites. The NEO-6M GPS module used in the proposed system is small and works on very low power, making it ideal for tracking applications. The GPS module operates at 3.3 V, as a result, powered by connecting the GPS module to the 3.3 V pin of the ESP32.

1. **ESP32 Microcontroller**

The ESP32 is a microcontroller with a Wi-Fi module, an open-source IoT platform that is characterized by low-cost and low-power system-on-a-chip (SOC). An ESP32 has a dual-core structure and internal modules such as Wi-Fi, Bluetooth, and many Peripheral Interfaces such as IR, SPI, CAN, Ethernet, and temperature sensors

1. **Ultrasonic Sensor**

Ultrasonic sensors are commonly used with Arduino to measure distance by sending and receiving ultrasonic waves. Connect the VCC pin of the ultrasonic sensor to the 5V pin on the Arduino. Connect the GND pin of the ultrasonic sensor to the GND pin on the Arduino. Connect the TRIG pin of the ultrasonic sensor to a digital pin (e.g., Pin 7) on the Arduino. Connect the ECHO pin of the ultrasonic sensor to another digital pin (e.g., Pin 6) on the Arduino.



**ARDUINO CODE FOR ESP32 MICROCONTROLLER:**

This code enables an ESP32 to interact with Blynk, read distances from two ultrasonic sensors, and update the Blynk application with information on people entering and leaving a defined area. It also controls an LED based on these conditions. The distances are measured using the Haversine formula, which is common for calculating distances between latitude and longitude coordinates.

#define BLYNK\_TEMPLATE\_ID "TMPL26V4fGv5q"

#define BLYNK\_TEMPLATE\_NAME "Test"

#define BLYNK\_AUTH\_TOKEN "XEHxNF\_Ur1Nt2p7wB5B20dNI1ZUwj34P"

#include <WiFi.h>

#include <WiFiClient.h>

#include <BlynkSimpleEsp32.h>

int duration1 = 0;

int distance1 = 0;

int duration2 = 0;

int distance2 = 0;

int dis1 = 0;

int dis2 = 0;

int dis\_new1 = 0;

int dis\_new2 = 0;

int entered = 0;

int left = 0;

int inside = 0;

#define LED 2

#define PIN\_TRIG1 15

#define PIN\_ECHO1 14

#define PIN\_TRIG2 13

#define PIN\_ECHO2 12

BlynkTimer timer;

char auth[] = BLYNK\_AUTH\_TOKEN;

char ssid[] = "Wokwi-GUEST";   // your network SSID (name)

char pass[] = "";

#define BLYNK\_PRINT **Serial**

long get\_distance1() {

  // Start a new measurement:

  digitalWrite(PIN\_TRIG1, HIGH);

  delayMicroseconds(10);

  digitalWrite(PIN\_TRIG1, LOW);

  // Read the result:

  duration1 = pulseIn(PIN\_ECHO1, HIGH);

  distance1 = duration1 / 58;

  return distance1;

}

long get\_distance2() {

  // Start a new measurement:

  digitalWrite(PIN\_TRIG2, HIGH);

  delayMicroseconds(10);

  digitalWrite(PIN\_TRIG2, LOW);

  // Read the result:

  duration2 = pulseIn(PIN\_ECHO2, HIGH);

  distance2 = duration2 / 58;

  return distance2;

}

int count = 0;

void myTimer() {

  dis\_new1 = get\_distance1();

  dis\_new2 = get\_distance2();

  if (dis\_new1<100){

    count++;

**Serial**.println("Number of passengers inside the bus:");

**Serial**.println(count);

  }

  if (dis\_new2<100){

    if(count<=0){

      count=0;

    }

    else{

      count--;

    }

**Serial**.println("Number of passengers inside the bus:");

**Serial**.println(count);

  }

}

 void setup() {

**Serial**.begin(115200);

  pinMode(LED, OUTPUT);

  pinMode(PIN\_TRIG1, OUTPUT);

  pinMode(PIN\_ECHO1, INPUT);

  pinMode(PIN\_TRIG2, OUTPUT);

  pinMode(PIN\_ECHO2, INPUT);

  Blynk.begin(auth, ssid, pass, "blynk.cloud", 8080);

  timer.setInterval(1000L, myTimer);

}

void loop() {

  Blynk.run();

  timer.run();

}

**DISTANCE CALCULATION:**

The Haversine formula was adopted to calculate the distance that will appear in the Android app. It calculates the distance between the passenger and the bus location using the latitude and longitude of the bus and the passenger who is at home, work, or at the bus stop. The following equations can be used to calculate the distance.

a=sin^2(x3/2) +cos(x1). cos(x2). sin^2(y3/2)

c= 2. atan2(sqrt(a). sqrt(1-a))

d= R\*c

where,

x=latitude;

y=longitude;

x3= latitude2 – latidue1;

y3= longitude2 – longitude1;

R = Earth radius (6400 km);

d = distance between two locations;

**Python Script:**

import math

def haversine (lat1, lon1, lat2, lon2):

    # Convert latitude and longitude from degrees to radians

    lat1 = math. radians(lat1)

    lon1 = math.radians(lon1)

    lat2 = math.radians(lat2)

    lon2 = math.radians(lon2)

    # Differences in latitude and longitude

    dlat = lat2 - lat1

    dlon = lon2 - lon1

    # Haversine formula

    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))

    # Radius of the Earth (in kilometers)

    R = 6371

    # Calculate the distance

    distance = R \* c

    return distance

# Example Usage

latitude1 = 40.7128  # Latitude of point 1 (in degrees)

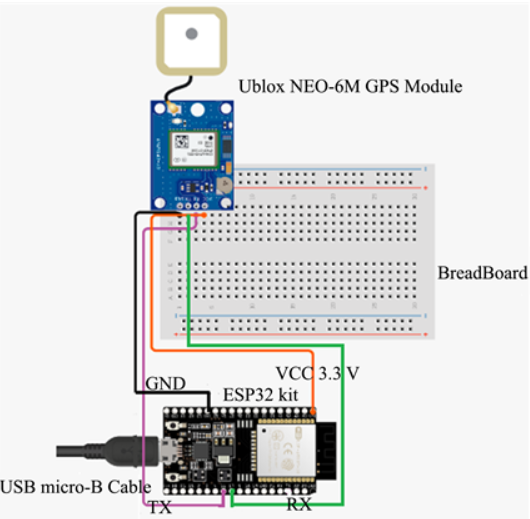
longitude1 = -74.0060  # Longitude of point 1 (in degrees)

latitude2 = 34.0522  # Latitude of point 2 (in degrees)

longitude2 = -118.2437  # Longitude of point 2 (in degrees)

distance = haversine(latitude1, longitude1, latitude2, longitude2)

print(f"The distance between the two locations is approximately {distance} kilometers.")



**ARRIVAL TIME CALCULATION:**

**Python script:**

class ArrivalTimeCalculator:

    def \_\_init\_\_(self):

        self.prev\_coordinate = (0, 0)  # Initialize previous coordinate to (0, 0)

    def calculate\_arrival\_time(self, distance, average\_speed):

        # Calculate arrival time in minutes

        arrival\_time = (distance / average\_speed) \* 60

        return arrival\_time

    def process\_new\_coordinates(self, new\_coordinate, distance, average\_speed):

        if self.prev\_coordinate == new\_coordinate:

            # Start timer and calculate delay time

            delay\_time = self.start\_timer\_and\_get\_delay()

            # Calculate final arrival time

            final\_arrival\_time = self.calculate\_arrival\_time(distance, average\_speed) + delay\_time

            self.prev\_coordinate = new\_coordinate  # Update prev coordinate

            return final\_arrival\_time

        else:

            # Process new coordinates and calculate estimated arrival time

            estimated\_arrival\_time = self.calculate\_arrival\_time(distance, average\_speed)

            # Update prev coordinate

            self.prev\_coordinate = new\_coordinate

            return estimated\_arrival\_time

    def start\_timer\_and\_get\_delay(self):

        # Placeholder function for starting timer and getting delay time

        # You should implement a timer function based on your specific environment

        delay\_time = 5  # Placeholder value, replace with actual delay time calculation

        return delay\_time

# Example Usage

arrival\_time\_calculator = ArrivalTimeCalculator()

# Example coordinates

new\_coordinate1 = (12.34, 56.78)  # Example new coordinate (latitude, longitude)

new\_coordinate2 = (12.34, 56.78)  # Example new coordinate (latitude, longitude)

distance = 10  # Example distance in kilometers

average\_speed = 40  # Example average speed in km/h

final\_arrival\_time1 = arrival\_time\_calculator.process\_new\_coordinates(new\_coordinate1, distance, average\_speed)

print(f"The final arrival time for first coordinate is approximately {final\_arrival\_time1} minutes.")

final\_arrival\_time2 = arrival\_time\_calculator.process\_new\_coordinates(new\_coordinate2, distance, average\_speed)

print(f"The final arrival time for second coordinate is approximately {final\_arrival\_time2} minutes.")

**Python script on the IoT sensors to send real-time location and ridership data to the transit information platform:**

import paho.mqtt.client as mqtt

import json

import time

from your\_sensor\_module import get\_real\_sensor\_data

# MQTT broker information

broker\_address = "mqtt.eclipse.org"

broker\_port = 1883

topic = "transit\_data"

def generate\_real\_data():

    location\_data = gps\_module.get\_location\_data()

    ridership\_data = ultrasonic\_sensor\_module.get\_ridership\_data()

    return {

        "location": location\_data,

        "ridership": ridership\_data

    }

# MQTT client setup

client = mqtt.Client()

client.connect(broker\_address, broker\_port, 60)

try:

    while True:

        # Generate real sensor data

        data = generate\_real\_data()

        # Convert data to JSON

        payload = json.dumps(data)

        # Publish data to the topic

        client.publish(topic, payload)

        # Print for verification

        print("Published:", payload)

        # Adjust the frequency based on your requirements

        time.sleep(10)  # Wait for 10 seconds before sending the next data

except KeyboardInterrupt:

    print("Script terminated by user.")

finally:

    # Disconnect from the broker

    client.disconnect()

**TEAM MEMBERS:**

Muhil J 2021504530

Shakti A P 2021504541

Shamitha G 2021504543

Shreejan A 2021504544

Sushmitha A 2021504551