PHASE - 5

PUBLIC TRANSPORT OPTIMIZATION

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.

2. **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

3. **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.

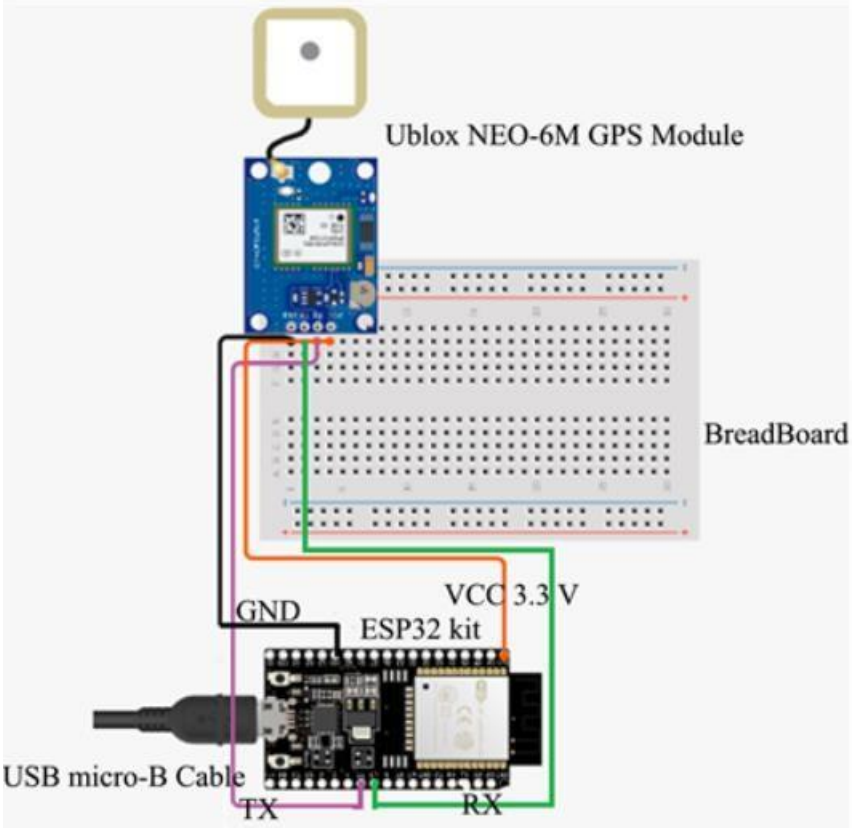
ARDUINO CODE FOR ESP32 MICROCONTROLLER:

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.

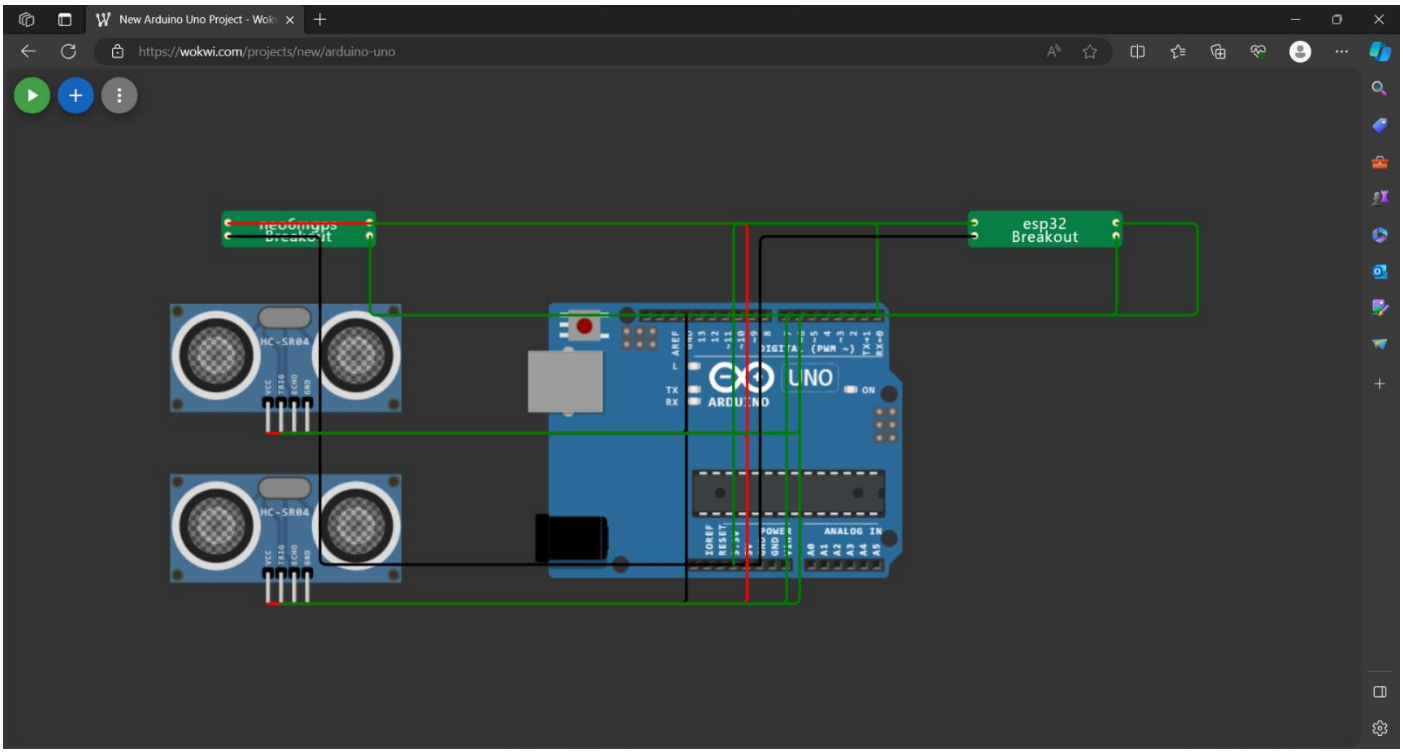
```
#define BLYNK_TEMPLATE_ID "TMPL26V4fGv5q"
#define BLYNK_TEMPLATE_NAME "Test"
#define BLYNK_AUTH_TOKEN "XEHxNF_Ur1Nt2p7wB5B20dNI1ZUwj34P"

#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++;
```

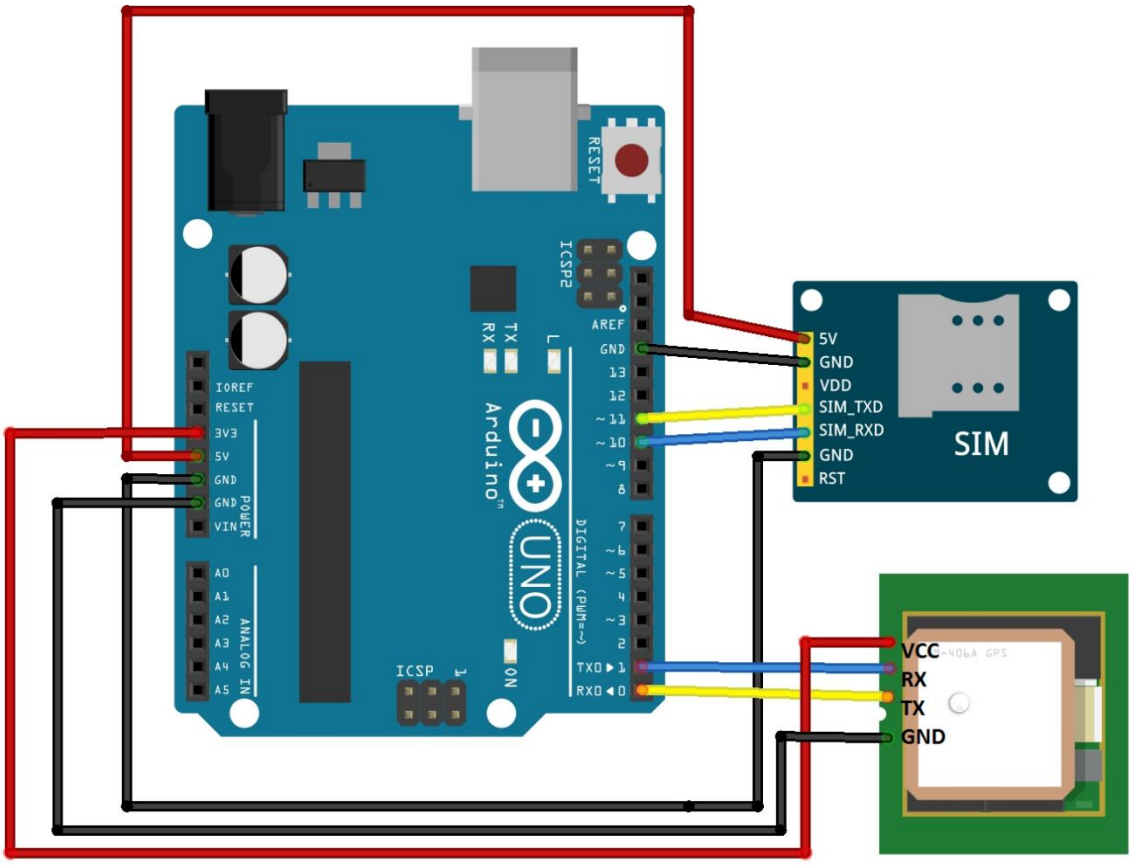
```
  Serial.println("Number of passengers inside the bus:");
  Serial.println(count);
  }
  if (dis_new2<100){
    if(count<=0){
      count=0;
    }
    else{
      count--;
    }
  Serial.println("Number of passengers inside the bus:");
  Serial.println(count);

  }
}

 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();
}
```



PASSENGER COUNT



GPS DATA SINGAL



ARRIVAL TIME CALCULATION:

Python script:

```
class
    ArrivalTimeCalcul
    ator: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) * 60return 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
```

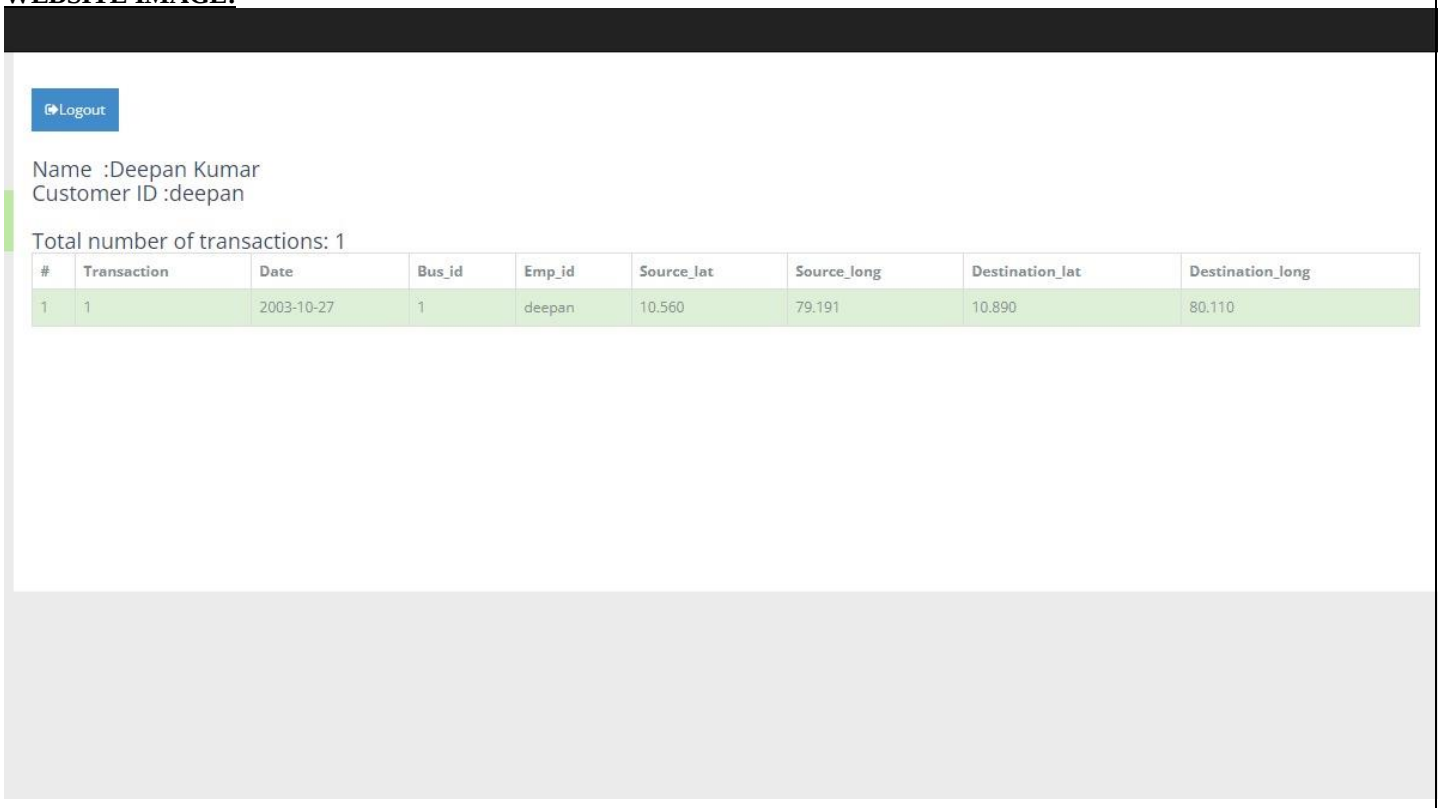
```
            return estimated_arrival_time

    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.")
```

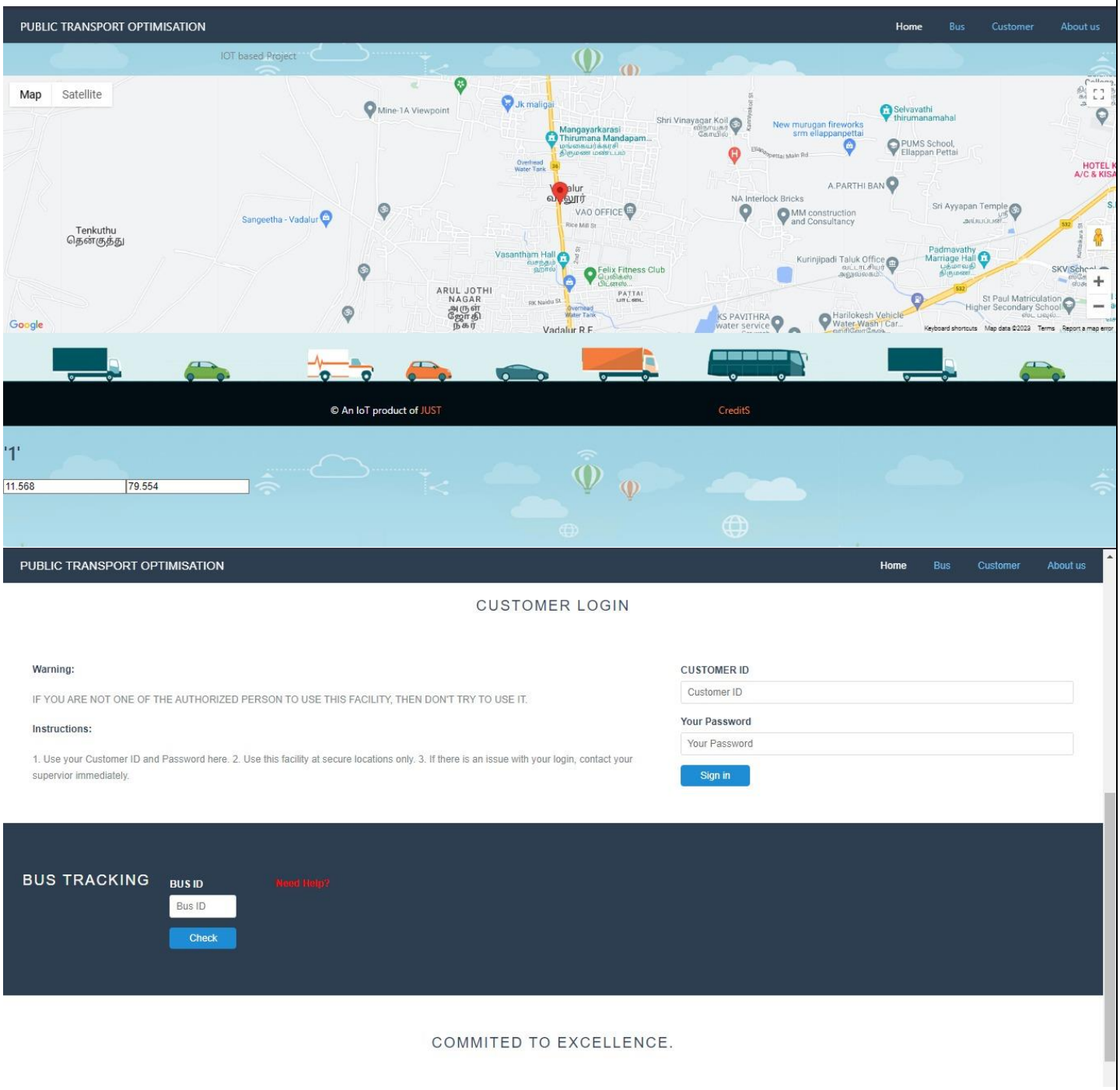
Python script on the IoT sensors to send real-time location and ridership data to the transit information platform:

```
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()
```

WEBSITE IMAGE:





MESSAGE SEND CODE

```
#include <TinyGPS.h>
#include <SoftwareSerial.h>
SoftwareSerial Gsm(7, 8);
TinyGPS gps;  //Creates a new instance of the TinyGPS object
//This is a comment
void setup()
{
  Serial.begin(9600);
  Gsm.begin(9600);
```

```
}
void loop()
{

 bool newData = false;
 unsigned long chars;
 unsigned short sentences, failed;
 for (unsigned long start = millis(); millis() - start < 1000;)
 {
  while (Serial.available())
  {
   char c = Serial.read();
   Serial.print(c);
   if (gps.encode(c))
    newData = true;

  }

 }

 if (newData)      //If newData is true
 {
  float flat, flon;
  String output="";
  unsigned long age;
  gps.f_get_position(&flat, &flon, &age);
  /*Gsm.print("AT+CMGF=1\r");
  delay(400);
  Gsm.print("AT+CMGS=\"+8801623709870\"\r");
  delay(300);
  Gsm.print("http://maps.google.com/maps?q=loc:");
 // Gsm.print("Latitude = ");
  Gsm.print(flat == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flat, 6);
  //Gsm.print(" Longitude = ");
  Gsm.print(",");
  Gsm.print(flon == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flon, 6);
  delay(200);
  Gsm.println((char)26); // End AT command with a ^Z, ASCII code 26
  delay(200);
  Gsm.println();
  delay(20000);*/
```

```
  flat == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flat, 6;
  flon == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : flon, 6;
  Serial.println("Lat:    ");
  Serial.println(flat);
  Serial.println("Lng:   ");
  Serial.println(flon);
  output="cwc19.mhhabib.info/add.php?lat="+String(flat)+"&lng="+String(flon);
  Serial.println(output);
  Serial.println("HTTP get method :");
  Serial.print("AT\\r\\n");
  Gsm.println("AT");  /* Check Communication */
  delay(500);
  ShowSerialData(); /* Print response on the serial monitor */
  delay(500);
  /* Configure bearer profile 1 */
  Serial.print("AT+SAPBR=3,1,\"CONTYPE\",\"GPRS\"\\r\\n");
  Gsm.println("AT+SAPBR=3,1,\"CONTYPE\",\"GPRS\"");  /* Connection type
GPRS */
  delay(500);
  ShowSerialData();
  delay(500);
  Serial.print("AT+SAPBR=3,1,\"APN\",\"www\"\\r\\n");
  Gsm.println("AT+SAPBR=3,1,\"APN\",\"www\"");  /* APN of the provider */
  delay(500);
  ShowSerialData();
  delay(500);
  Serial.print("AT+SAPBR=1,1\\r\\n");
  Gsm.println("AT+SAPBR=1,1"); /* Open GPRS context */
  delay(500);
  ShowSerialData();
  delay(500);
  Serial.print("AT+SAPBR=2,1\\r\\n");
  Gsm.println("AT+SAPBR=2,1"); /* Query the GPRS context */
  delay(500);
  ShowSerialData();
  delay(500);
  Serial.print("AT+HTTPINIT\\r\\n");
  Gsm.println("AT+HTTPINIT"); /* Initialize HTTP service */
delay(500);
  ShowSerialData();
  delay(500);
```

```
   Serial.print("AT+HTTPPARA=\"CID\",1\\r\\n");
   Gsm.println("AT+HTTPPARA=\"CID\",1");  /* Set parameters for HTTP
session */
   delay(500);
   ShowSerialData();
   delay(500);

Serial.println("AT+HTTPPARA=\"URL\",\"http://cwc19.mhhabib.info/add.php\"\\r
\\n");
   Gsm.print("AT+HTTPPARA=\"URL\",\"cwc19.mhhabib.info/add.php?");
   Gsm.print("&l=");
   Gsm.print(flat,6);
   Gsm.print("&n=");
   Gsm.print(flon,6);
   Gsm.print("\"\r\n");
   delay(500);
   ShowSerialData();
   delay(500);
   Serial.print("AT+HTTPACTION=0\\r\\n");
   Gsm.println("AT+HTTPACTION=0");  /* Start GET session */
   delay(500);
   ShowSerialData();
   delay(500);
   Serial.print("AT+HTTPREAD\\r\\n");
   Gsm.println("AT+HTTPREAD");  /* Read data from HTTP server */
   delay(500);
   ShowSerialData();
   delay(500);
   Serial.print("AT+HTTPTERM\\r\\n");
   Gsm.println("AT+HTTPTERM");  /* Terminate HTTP service */
   delay(500);
   ShowSerialData();
   delay(500);
   Serial.print("AT+SAPBR=0,1\\r\\n");
   Gsm.println("AT+SAPBR=0,1"); /* Close GPRS context */
   delay(500);
   ShowSerialData();
   delay(500);
   }
}
void ShowSerialData()
```

```
{
  while(Gsm.available()!=0)  /* If data is available on serial port */
  Serial.write(char (Gsm.read())); /* Print character received on to the serial
monitor */
}
```

SERVER CODE

```
const express = require('express');
const mysql = require('mysql2/promise');
const app = express();
const port = 3000;

// Create a MySQL connection pool

const pool = mysql.createPool({
  host: 'localhost',
  user: 'root',
  password: 'Deepan@1234',
  database: 'bus',
  waitForConnections: true,
  connectionLimit: 10,
  queueLimit: 0,
});

// Middleware to parse JSON in the request body
app.use(express.json());

app.post('/update-track', async (req, res) => {
  try {
    // Extract data from the request body
    const { id, lat, long } = req.body;

    // Ensure that id, lat, and long are provided and valid
    if (!id || isNaN(id) || !lat || isNaN(lat) || !long || isNaN(long)) {
      return res.status(400).json({ error: 'Invalid input data.' });
    }
    // Create a connection from the pool
    const connection = await pool.getConnection();

    // Update the "track" table
```

```
    const sql = 'UPDATE track SET lat = ?, long = ? WHERE id = ?';
    const [results] = await connection.execute(sql, [lat, long, id]);

    // Release the connection back to the pool
    connection.release();

    if (results.affectedRows === 1) {
      res.json({ message: 'Track updated successfully.' });
    } else {
      res.status(404).json({ error: 'Track with the specified ID not found.' });
    }
  } catch (error) {
    console.error('Error:', error);
    res.status(500).json({ error: 'Internal server error.' });
  }
});

app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

STEPS TO COMPLETE THIS PROJECT

Running Your Website (HTML, CSS, PHP, JavaScript):

1. Web Server Setup:
   - Install a web server on your local machine, such as Apache, Nginx, XAMPP, or WampServer.

2. Website Directory:
   - Create a directory where you will place your website files.

3. File Placement:
   - Move your website files (HTML, CSS, PHP, JavaScript) into the directory created in step 2.

4. Configuration (PHP, if applicable):
   - Review your PHP code and make sure database connections and other configurations are set for your local environment.

5. Start the Web Server:
   - Start your web server to make your website accessible. The default web root directory might be htdocs or www in XAMPP or WampServer.

6. Access Your Website:
   - Open a web browser and enter http://localhost or http://127.0.0.1 in the address bar to access your website.

7. Testing:
   - Test your website's functionality to ensure it's working as expected.

Running Your Node.js Server:

1. Server Directory:
   - Navigate to the directory where your Node.js server code is located.

2. Dependencies Installation:
   - In the terminal, run the following commands to install the required Node.js dependencies:

```
npm install
```

3. Start the Node.js Server:
   - Run the Node.js server using the following command:

```
node server.js
```

4. Access the Server:
   - Your Node.js server should now be running. You can access it at http://localhost:PORT, where PORT is the port specified in your server code.

Running Your Arduino Code:

1. Arduino IDE:
   - Open the Arduino IDE on your local machine.

2. Open Code:
   - Navigate to File > Open and select the Arduino code file from your message folder.

3. Connect Arduino:
   - Connect your Arduino device to your computer using a USB cable.

4. Configure Board and Port:
   - In the Arduino IDE, go to the Tools menu and select the appropriate Arduino board and port.

5. Upload Code:
   - Click the "Upload" button in the Arduino IDE to upload the code to your Arduino device.

6. Monitor Serial Output (if applicable):
   - If your Arduino code sends data to the serial monitor, open the Arduino IDE's serial monitor to view the output.

7. Testing:
   - Test your Arduino code to ensure it functions as expected.

By following these steps, you can run your website, Node.js server, and Arduino code on your localhost for testing and development purposes. Make sure you have the required software and dependencies installed for each component of your project.