PUBLIC TRANSPORT OPTIMIZATION

PHASE: 03

Code: (USING PYTHON)

```
 import time
from adafruit_bme280 import Adafruit_BME280
import board
import busio
from adafruit_ads1x15.analog_in import AnalogIn
from adafruit_tsl2591 import TSL2591
from adafruit_pn532 import PN532
from digitalio import DigitalInOut, Direction
import adafruit_dht
from pyblynk import Blynk
from tinygps import TinyGPS

# BME280 Environmental Sensor
i2c = busio.I2C(board.SCL, board.SDA)
bme = Adafruit_BME280(i2c)

# ADS1115 ADC
ads = AnalogIn(board.A0)

# TSL2591 Light Sensor
```

```
tsl = TSL2591()

# PN532 NFC Module
pn532 = PN532(board.SDA, board.SCL)

# DHT Sensor
dht_sensor = adafruit_dht.DHT22(board.D2)

# GPS Pins
gps_tx_pin = board.TX
gps_rx_pin = board.RX
gps_serial = TinyGPS()

# Blynk Setup
BLYNK_AUTH = "Your_Blynk_Auth_Token"
blynk = Blynk(BLYNK_AUTH)

# Motion Sensor Pin
motion_sensor_pin = DigitalInOut(board.D3)
motion_sensor_pin.direction = Direction.INPUT

@blynk.VIRTUAL_WRITE(1)
def bme280_data(pin):
    temperature = bme.temperature
```

```
    humidity = bme.humidity
    pressure = bme.pressure / 100.0
    blynk.virtual_write(1, temperature)
    blynk.virtual_write(2, humidity)
    blynk.virtual_write(3, pressure)

@blynk.VIRTUAL_WRITE(4)
def ads1115_data(pin):
    voltage = ads.value * (5.0 / 32767.0)
    blynk.virtual_write(4, voltage)

@blynk.VIRTUAL_WRITE(5)
def tsl2591_data(pin):
    luminosity = tsl.lux
    desired_light_level = int(map_range(luminosity, 0, 5000, 0, 255))
    blynk.virtual_write(5, desired_light_level)

@blynk.VIRTUAL_WRITE(6)
def nfc_data(pin):
    uid = pn532.read_passive_target()
    if uid:
        card_uid = ''.join([format(byte, '02X') for byte in uid])
        process_card(card_uid)
        time.sleep(1)
```

```
def process_card(card_uid):
    blynk.virtual_write(6, card_uid)

@blynk.VIRTUAL_WRITE(7)
def dht_data(pin):
    temperature = dht_sensor.temperature
    humidity = dht_sensor.humidity
@blynk.VIRTUAL_WRITE(9)
def gps_data(pin):
    sentence = gps_serial.read()
    if sentence:
        if sentence.startswith(b'$GPGGA'):
            lat, lon, _ = gps_serial.parse_gga(sentence)
            blynk.virtual_write(9, lat)
            blynk.virtual_write(10, lon)

@blynk.VIRTUAL_WRITE(11)
def motion_sensor_data(pin):
    motion_value = motion_sensor_pin.value
    if motion_value == True:
        blynk.notify("Motion detected!")

def map_range(x, in_min, in_max, out_min, out_max):
```

```
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) +
out_min


while True:
    blynk.run()
    time.sleep(1)
```

ARDUINO PROGRAM:

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#include <Adafruit_ADS1015.h>
#include <Adafruit_PN532.h>
#include <TinyGPS++.h>
#include <DHT.h>
#include <WiFi.h>
#include <BlynkSimpleEsp32.h>

char auth[] = "Your_Blynk_Auth_Token";
char ssid[] = "Your_WiFi_SSID";
char pass[] = "Your_WiFi_Password";

// BME280 Environmental Sensor
Adafruit_BME280 bme;
```

```
// ADS1115 ADC
Adafruit_ADS1115 ads;
const int fuelSensorPin = A0;

// Motion Sensor
int motionSensorPin = 14;

// GPS
TinyGPSPlus gps;
#define GPS_TX_PIN 17
#define GPS_RX_PIN 16

// DHT Sensor
#define DHTPIN 2
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);

// NFC Module
Adafruit_PN532 nfc(SDA_PIN, SCL_PIN);

// Panic Button
#define PANIC_BUTTON_PIN 2

// Congestion Sensor
```

```
#define SENSOR_PIN 15
#define LED_PIN 13

BlynkTimer timer;

void setup() {
  Serial.begin(115200);

  Blynk.begin(auth, ssid, pass);

  if (!bme.begin(0x76)) {
    Serial.println("Could not find a valid BME280 sensor, check wiring!");
    while (1);
  }

  if (!ads.begin()) {
    Serial.println("Failed to initialize ADS1115");
    while (1);
  }

  pinMode(fuelSensorPin, INPUT);
  pinMode(motionSensorPin, INPUT);
  pinMode(PANIC_BUTTON_PIN, INPUT_PULLUP);
```

```
  pinMode(SENSOR_PIN, INPUT);
  pinMode(LED_PIN, OUTPUT);

  Serial1.begin(9600, SERIAL_8N1, GPS_RX_PIN, GPS_TX_PIN);
  nfc.begin();
  uint32_t versiondata = nfc.getFirmwareVersion();
  if (!versiondata) {
    Serial.print("Didn't find PN53x board");
    while (1);
  }
  nfc.SAMConfig();

  dht.begin();
  timer.setInterval(1000L, checkMotionSensor);
}

void loop() {
  Blynk.run();
  timer.run();
while (Serial1.available() > 0) {
    if (gps.encode(Serial1.read())) {
      displayInfo();
    }
  }
```

```
if (nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, uid,
&uidLength)) {
   String cardUID = "";
   for (uint8_t i = 0; i < uidLength; i++) {
     cardUID += String(uid[i], HEX);
   }
   processCard(cardUID);
 }

 Blynk.run();
 int sensorValue = digitalRead(SENSOR_PIN);
 if (sensorValue == HIGH) {
   digitalWrite(LED_PIN, HIGH);
 } else {
   digitalWrite(LED_PIN, LOW);
 }

 float temperature = dht.readTemperature();
 float humidity = dht.readHumidity();
 uint16_t luminosity = tsl.getLuminosity(TSL2591_VISIBLE);
 int desiredLightLevel = map(luminosity, 0, 5000, 0, 255);

 Blynk.virtualWrite(V1, temperature);
 Blynk.virtualWrite(V2, humidity);
```

```
  Blynk.virtualWrite(V3, pressure);
  Blynk.virtualWrite(V4, motionValue);
  Blynk.virtualWrite(V5, voltage);
  Blynk.virtualWrite(V6, cardUID);
  Blynk.virtualWrite(V7, desiredLightLevel);

  if (temperature >= 25.0) {
    Blynk.virtualWrite(V8, HIGH);
  } else if (temperature <= 20.0) {
    Blynk.virtualWrite(V8, LOW);
  } else {
    Blynk.virtualWrite(V8, LOW);
  }
}

void displayInfo() {
  if (gps.location.isValid()) {
    Serial.print("Latitude: ");
    Serial.println(gps.location.lat(), 6);
    Serial.print("Longitude: ");
    Serial.println(gps.location.lng(), 6);
    Blynk.virtualWrite(V1, gps.location.lat());
    Blynk.virtualWrite(V2, gps.location.lng());
  } else {
```

```
    Serial.println("Invalid GPS location");
  }
}

void processCard(String cardUID) {
  Blynk.virtualWrite(V9, cardUID);
}

void checkMotionSensor() {
  int motionValue = digitalRead(motionSensorPin);
  if (motionValue == HIGH) {
    Blynk.notify("Motion detected in public transport!");
  }
}
```

FUNCTIONS USED IN THE CODE:

1) Importing libraries:
    - Time: This library is used for handling time-related operations and introduces delays.
    - board: It provides an abstraction for hardware pins and is used to define the pins used for various peripherals.
    - busio: This library is for handling I2C and UART communication.
    - adafruit_ssd1306: It is a library for controlling SSD1306 OLED displays.

- digitalio: This library is used for digital input/output (I/O) control.
- adafruit_gps: This library is specifically for interfacing with Adafruit GPS modules.

2)Pin definitions and Hardware Initialization:

- The code defines the pins used for I2C communication (SDA_PIN and SCL_PIN) and the reset pin for the OLED display (oled_reset).
- It initializes the I2C bus with the specified pins.
- It initializes the OLED display and assigns it to the display variable.
- It sets up UART communication for the GPS module on specified TX and RX pins (board. TX and board. RX).

3) Setup() Function:

- This function is responsible for the initial setup of the OLED display. It clears the display, displays a "Waiting for GPS..." message, and updates the display to show this message.

4) Loop() Function:

- This is the main loop of the program.
- Inside the loop, it calls gps. update() to update the GPS data.
- It checks if a GPS fix is available using gps.has_fix.
- If a GPS fix is available, it clears the OLED display and then displays latitude, longitude, and altitude information on the OLED screen.
- The display is updated, then the code sleeps for 1 second, and this loop continues executing.

5) Main Loop: This conditional statement checks if the script is being run as the main program.
   If it is the main program, it calls the setup() function to initialize the display and then enters the main loop by calling the loop() function