Public Transport Optimization

Phase 3:

IoT devices and then Developing a Python script on the IoT devices as per the project .

Abstract:

The IoT-Enabled Public Transportation Optimization System is an innovative project aimed at enhancing the efficiency and quality of public transportation services. This development phase, Part 1, focuses on deploying IoT sensors within public transportation vehicles to collect real-time data, primarily related to the vehicle's location and ridership. The collected data will be transmitted to a centralized transit information platform using Python scripts running on the IoT sensors.

Introduction:

Public transportation systems play a pivotal role in modern urban environments, providing an essential means of mobility for millions of people worldwide. The efficiency, accessibility, and reliability of public transport systems have a significant impact on urban congestion, environmental sustainability, and the overall quality of life in cities. Public transport optimization is a multifaceted approach aimed at improving the performance, accessibility, and sustainability of public transportation services. It involves the strategic use of technology, data, and planning to create a more efficient and passenger-friendly transit experience.

Microcontroller Naming:

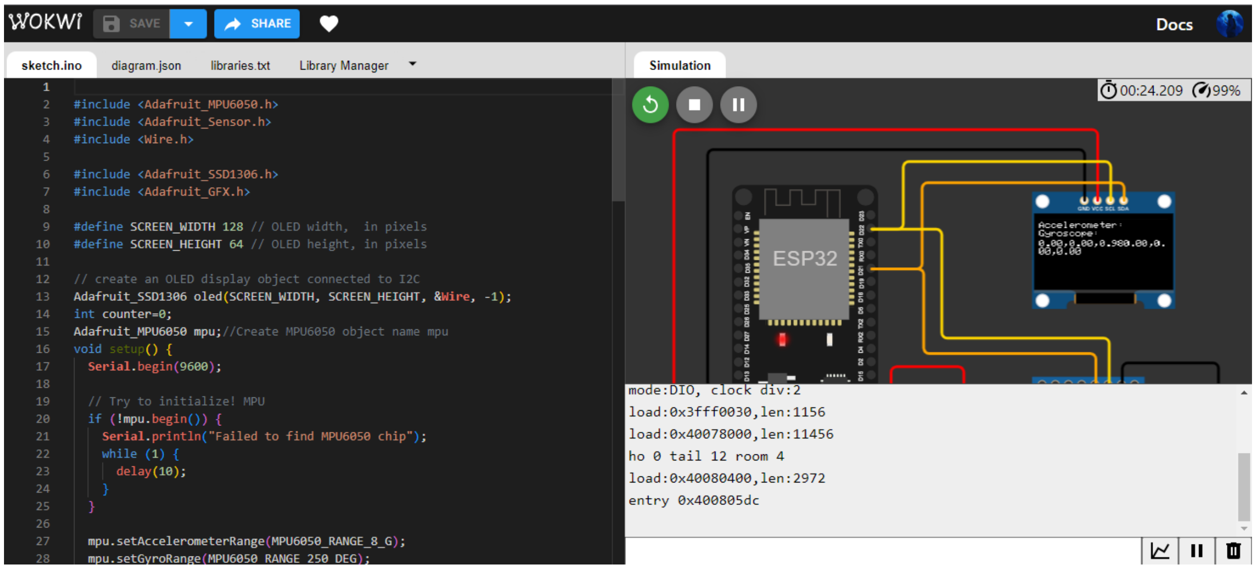
ESP32:

It is a successor to ESP8266 SoC and comes in both single-core and dual-core variations of the Tensilica's 32-bit Xtensa LX6 Microprocessor with integrated Wi-Fi and Bluetooth.

Arduino uno:

Arduino UNO is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header and a reset button.It simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

Simulation of public transport optimization:



```

#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>

#include <Adafruit_SSD1306.h>
#include <Adafruit_GFX.h>

#define SCREEN_WIDTH 128 // OLED width,  in pixels
#define SCREEN_HEIGHT 64 // OLED height, in pixels

// create an OLED display object connected to I2C
Adafruit_SSD1306 oled(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);
int counter=0;
Adafruit_MPU6050 mpu;//Create MPU6050 object name mpu
void setup() {
  Serial.begin(9600);

  // Try to initialize! MPU
  if (!mpu.begin()) {
    Serial.println("Failed to find MPU6050 chip");
    while (1) {
      delay(10);
    }
  }

  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
  mpu.setGyroRange(MPU6050_RANGE_250_DEG);
```

```
mode:DIO, clock div:2
load:0x3fff0030,len:1156
load:0x40078000,len:11456
ho 0 tail 12 room 4
load:0x40080400,len:2972
entry 0x400805dc
```

Program:

PYTHON CODE

```
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>

#include <Adafruit_SSD1306.h>
#include <Adafruit_GFX.h>

#define SCREEN_WIDTH 128 // OLED width,  in pixels
#define SCREEN_HEIGHT 64 // OLED height, in pixels

// create an OLED display object connected to I2C
Adafruit_SSD1306 oled(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);
int counter=0;
Adafruit_MPU6050 mpu;//Create MPU6050 object name mpu
void setup() {
  Serial.begin(9600);

  // Try to initialize! MPU
  if (!mpu.begin()) {
    Serial.println("Failed to find MPU6050 chip");
    while (1) {
      delay(10);
    }
  }

  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
  mpu.setGyroRange(MPU6050_RANGE_250_DEG);
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);
  //end of MPU configuration

  Serial.println("");
  delay(500);

  // initialize OLED display with I2C address 0x3C
  if (!oled.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("failed to start SSD1306 OLED"));
    while (1);
  }

  delay(1000);         // wait two seconds for initializing
  oled.clearDisplay(); // clear display

  oled.setTextSize(1);         // set text size
  oled.setTextColor(WHITE);    // set text color
  oled.setCursor(0, 2);       // set position to display (x,y)
  oled.println("Accelerometer:");
  oled.println("Gyroscope:"); // set text
  oled.display();              // display on OLED
}

void loop() {
  oled.fillRect(0, 20, 30 , 20, 0x000000);// Partial fill screen
  oled.setCursor(0, 20);

  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);
  oled.print(a.acceleration.x/10);
  oled.print(","); // set text
  oled.print(a.acceleration.y/10);
  oled.print(",");
  oled.print(a.acceleration.z/10);
  oled.print("");
  oled.display();

  oled.print(g.gyro.x/10);
  oled.print(","); // set text
  oled.print(g.gyro.y/10);
  oled.print(",");
  oled.print(g.gyro.z/10);
  oled.print("");
  oled.display();
  delay(2000);
}
```

Components:

1. **IoT Sensors:**
    - GPS trackers: These provide real-time location data for vehicles, enabling accurate tracking and route optimization.
    - Passenger counters: Track the number of passengers on each vehicle, helping to manage capacity and plan for increased demand.
    - Environmental sensors: Monitor conditions like temperature, humidity, and air quality, which can be crucial for both passenger comfort and route planning.

2. **Communication Infrastructure:**
   - Data communication networks: Reliable and fast data networks are essential for transmitting real-time data from vehicles to control centers and passenger apps.
   - Messaging protocols (e.g., MQTT, AMQP): These facilitate the efficient exchange of data between IoT devices and the central system.
3. **Central Data Management and Analysis:**
   - Centralized servers and databases: Collect, store, and process data from IoT sensors and other sources.
   - Data analytics tools: Analyze the collected data to extract actionable insights for optimizing routes, schedules, and resource allocation.
4. **Real-Time Passenger Information Systems:**
   - Passenger information displays: Inform passengers at stops and stations about real-time arrival predictions, service updates, and route information.
   - Mobile apps and websites: Provide passengers with easy access to real-time information on their smartphones and other devices.
5. **Route Planning and Scheduling Software:**
   - Software systems that use real-time data and historical information to optimize routes and schedules for efficiency and on-time performance.
6. **Fleet Management and Monitoring:**
   - Fleet management software: Helps operators track vehicle maintenance, fuel consumption, and driver behavior.
   - Telematics systems: Monitor vehicle health and performance in real-time to ensure safety and reliability.
7. **Passenger Counting and Payment Systems:**
   - Automated fare collection systems: Ensure accurate fare collection and help monitor passenger flow.

- Ticketing and payment apps: Enable contactless payment options for passengers.

8. **Traffic and Congestion Management:**
    - Traffic management systems: Coordinate with traffic signals and prioritize public transport to reduce delays.
    - Incident detection and response systems: Identify and respond to traffic incidents and emergencies.
9. **Environmental Sustainability Measures:**
    - Alternative fuel vehicles: Integrate eco-friendly buses, trams, and trains into the fleet to reduce carbon emissions.
    - Renewable energy sources: Power transportation hubs with renewable energy to reduce the carbon footprint.
10. **Data Security and Privacy Measures:**
    - Implement robust security protocols to protect sensitive passenger data and the integrity of the system.

Conclusion:

Public transport optimization is an essential endeavor for urban planners, transportation authorities, and communities looking to address urban mobility challenges, reduce environmental impact, and improve the quality of life in cities. By harnessing technology, data, and strategic planning, cities can create efficient, accessible, and sustainable public transportation systems that meet the needs of their residents and contribute to a more prosperous and sustainable urban future.