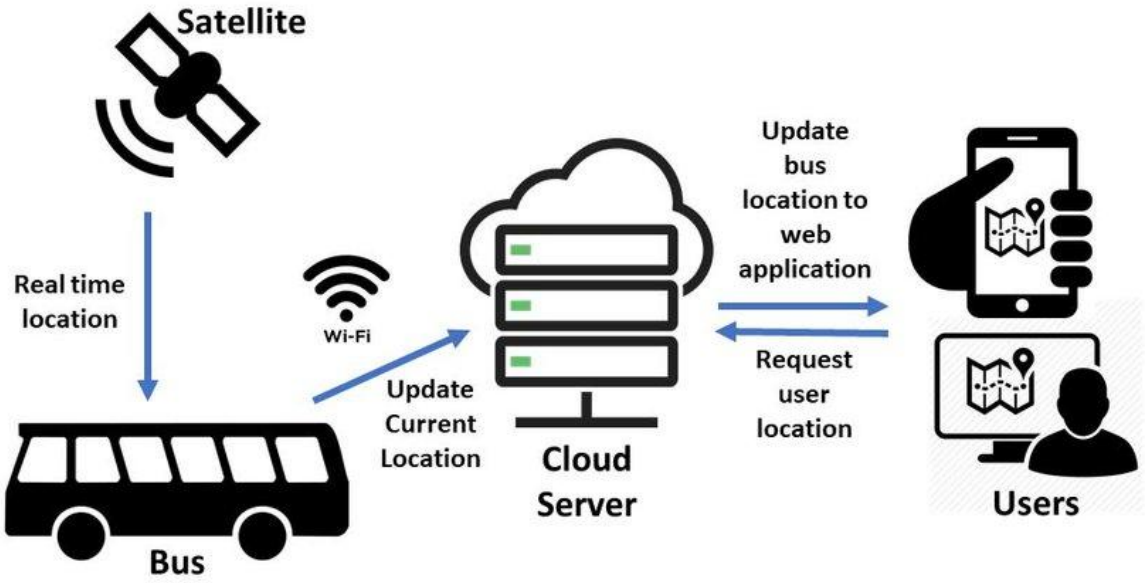
PUBLIC TRANSPORT OPTIMIZATION

PUBLIC BUS MONITORING

M.PRISHA | M.KEERTHANA | S.GOPIKA | R.DEVI

Project Description

The project involves integrating IoT sensors into public transportation vehicles to monitor ridership, track locations, and predict arrival times. The goal is to provide real-time transit information to the public through a public platform, enhancing the efficiency and quality of public transportation services. This project includes defining objectives, designing the IoT sensor system, developing the real-time transit information platform, and integrating them using IoT technology and Python.



Hardware Components

- ➢ GPS Module
- ➢ GSM Module
- ➢ RFID Reader
- ➢ Camera
- ➢ Microcontroller
- ➢ Power Supply Components
- ➢ Enclosures and Mounting Hardware
- ➢ Sensors and Peripherals
- ➢ Networking Hardware
- ➢ Storage Media

➢ Antennas
➢ Cabling and Wiring

Setup of Hardware

The hardware connections are first configured in order to build this project. Later on, the software component will be covered.

Step 1: Design IoT Sensor System Identify and procure

the necessary components and sensors:

➢ GPS Module: Select a reliable GPS module to track real-time bus locations.
➢ GSM Module: Choose a GSM module for data transmission and SMS alerts.
➢ RFID Reader: Install RFID readers on buses for student identification.
➢ Camera: Select cameras capable of capturing photos and videos.
➢ Microcontroller: Decide on a microcontroller platform (e.g., Arduino, Raspberry Pi) for data collection and processing.

Step 2: Develop Real-Time Transit Information Platform

Choose the technology stack:

➢ Use IoT technology for data connectivity and communication.
➢ Develop the platform using Python for data processing and storage. ➢ Design the platform architecture for real-time data processing and storage.

Step 3: Data Collection

➢ Install GPS modules on buses to continuously collect GPS coordinates and timestamps.
➢ Deploy RFID readers on buses to scan RFID cards during boarding and disembarking.
➢ Attach cameras inside buses to capture time stamped photos periodically.

Step 4: Data Processing

➢ Utilise the selected microcontroller to process data from sensors.
➢ Implement algorithms to:
➢ Check for route compliance by comparing GPS data to predefined routes.
➢ Verify public disembarkation by cross-referencing RFID data.
➢ Securely store processed data in local or cloud storage.

Step 5: Alerts and Communication Program the system to:

➢ Trigger SMS alerts for deviations, unauthorised stops, RFID registration issues, and unusual events.
➢ Integrate with Telegram for real-time monitoring by transportation authorities.

Step 6: Data Storage

➢ Set up secure data storage solutions:
➢ Choose between local storage on buses and cloud storage.
➢ Ensure data encryption to protect public information during transmission and storage.

Step 7: User Interface

➢ Develop a user-friendly web-based dashboard for real-time bus tracking, accessible by the public.

Step 8: Power Supply and Security

➢ Ensure reliable power sources for the IoT components, considering bus batteries and external power supplies.
➢ Implement strong encryption protocols to secure student data during both transmission and storage. Step 9: Compliance and Maintenance

➢ Comply with regulations and best practices regarding the handling of public data to protect privacy.
➢ Establish a regular maintenance schedule for the entire system, including sensor and software maintenance.
➢ Provide effective training to relevant personnel for system management and maintenance.

Software components

➢ Python 3.7 IDLE
➢ Database Management System(SQL)
➢ Web Development Tools(Django,Flask)
➢ SMS Gateway Services or APIs
➢ Telegram Bot API

Software Development

Step 1: Software Architecture and Design

➢ Create a high-level architecture for the software, identifying key components, modules, and their interactions.
➢ Design the database schema to store collected data securely.
➢ Develop a software design document detailing how each component will be implemented.

Step 2: Select Development Tools and Frameworks

➢ Choose the development tools, frameworks, and libraries based on the project requirements. For example: ➢ Use Python for backend development.
➢ Select a web framework like Django or Flask for the web-based dashboard.
➢ Choose appropriate libraries for IoT communication, data storage, and geofencing.

Step 3: Real-Time Transit Information Platform

➢ Develop the real-time transit information platform, which processes and integrates data from GPS, RFID, and cameras.
➢ Implement algorithms for real-time data synchronisation and validation.
➢ Ensure that the platform can predict arrival times based on GPS data.

Step 4: User Interface Development

➢ Create the web-based dashboard for real-time bus tracking
➢ Design the user interface to be intuitive and user-friendly.
➢ Implement features such as route visualisation, live tracking, and alerts.

Step 5: Data Storage and Security

➢ Develop data storage components to securely store collected data, either locally or in the cloud.
➢ Implement encryption protocols to protect student data during transmission and storage.
➢ Ensure data access controls and user authentication for security.

Step 6: Communication and Alerts

➢ Integrate communication protocols (e.g., SMS and Telegram) for sending alerts to parents and school authorities.
➢ Set up automated alerts for deviations, unauthorised stops, and unusual bus events.
➢ Establish real-time monitoring through the Telegram integration.

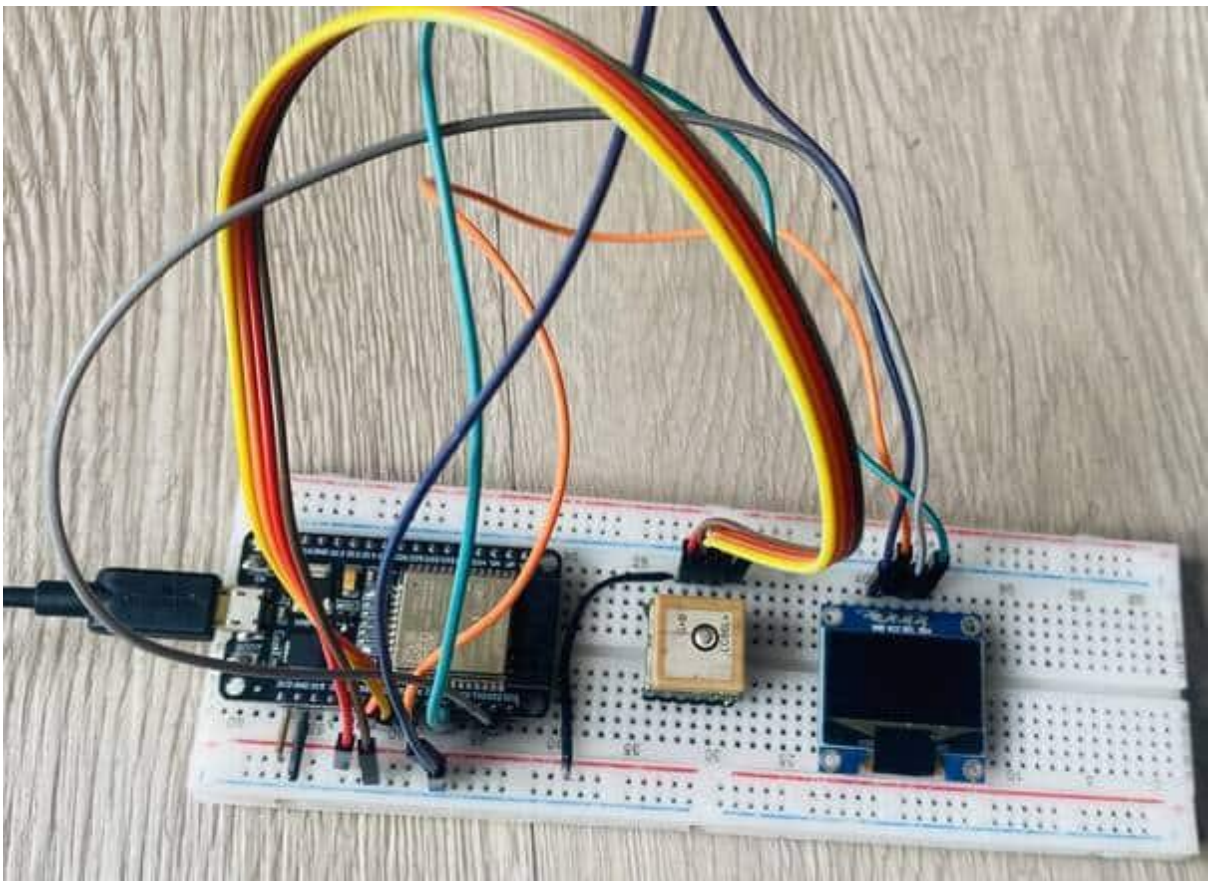
Demonstration:

Connecting the circuits for a bus monitoring GPS IoT project involves integrating various electronic components to gather, process, and transmit data. Below is a step-by-step demonstration procedure for connecting the circuits of a basic bus monitoring project using GPS and IoT

Components:

1. Arduino (or compatible microcontroller): Used for data processing and communication
2. GPS Module: Collects bus location data.
3. Temperature and Humidity Sensor (e.g., DHT11 or DHT22): Monitors environmental conditions.
4. SIM800L GSM/GPRS Module: Enables wireless communication.
5. Power Supply: Ensure the components receive adequate power.
6. Jumper Wires: Used for connecting components.

Procedure:



Power Supply:

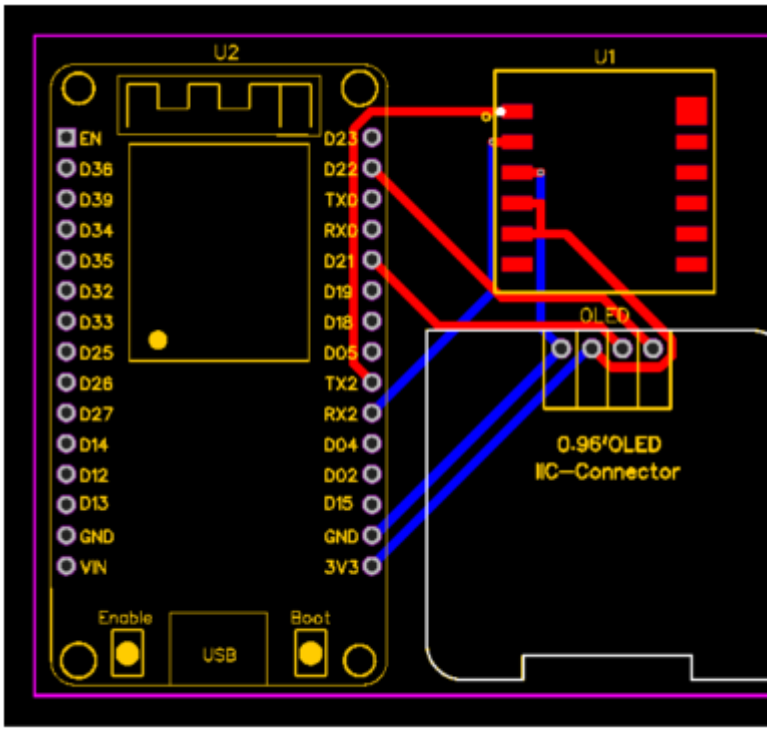
Begin by connecting the power supply to your Arduino or microcontroller board. Ensure the voltage and current ratings are suitable for the components you are using.

Connect the GPS Module:

- Connect the GPS module to the Arduino using jumper wires. Typically, this involves connecting the module's RX pin to the Arduino's TX pin and the module's TX pin to the Arduino's RX pin. Connect the module's VCC and GND pins to the appropriate Arduino power and ground pins.
- Connect the Temperature and Humidity Sensor:
  - Connect the DHT11 or DHT22 sensor to the Arduino. Connect the sensor's data pin to one of the digital pins on the Arduino, its VCC pin to the 5V power supply, and its GND pin to the ground.

Connect the GSM/GPRS Module:

  - Connect the SIM800L module to the Arduino. This typically involves connecting the module's RX to a digital pin on the Arduino (for sending commands), its TX to a digital pin (for receiving responses), and its VCC and GND to the appropriate power and ground pins.



Connect and Power the Arduino: Connect your Arduino to your computer using a USB cable for programming and power..

Arduino Programming:

- Write and upload the Arduino code that reads data from the GPS module and temperature and humidity sensor, and sends this data via the GSM/GPRS module. The code should include libraries for each component and implement data parsing and transmission functions.

Verify Circuit Connections:

- Double-check all connections to make sure they are secure and correctly wired.

Run the Code:

- Once the code is uploaded to the Arduino, open the serial monitor to see if data is being received from the GPS and sensors and successfully transmitted via the GSM module. Troubleshoot and refine the code as needed.

PROGRAM:

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#include <TinyGPS++.h>
#include <SoftwareSerial.h>
SoftwareSerial ss(10, 11); // RX, TX for GPS module
// Create a BME280 sensor object
Adafruit_BME280 bme;
TinyGPSPlus gps;
void setup() {
  // Start communication with GPS module
  ss.begin(9600);
  if (!bme.begin(0x76)) {
    Serial.println("Could not find a valid BME280 sensor, check wiring!");
    while (1);
  }

  Serial.begin(9600);
}

void loop() {
  // Read data from the BME280 sensor
```

```
  float temperature = bme.readTemperature();
  float humidity = bme.readHumidity();// Read GPS data
  while (ss.available() > 0) {
    if (gps.encode(ss.read())) {
      if (gps.location.isValid()) {
        float latitude = gps.location.lat();
        float longitude = gps.location.lng();  // Print sensor data and GPS
coordinates
        Serial.print("Temperature: ");
        Serial.print(temperature);
        Serial.println(" °C");
        Serial.print("Humidity: ");
        Serial.print(humidity);
        Serial.println(" %");
        Serial.print("Latitude: ");
        Serial.println(latitude, 6);

        Serial.print("Longitude: ");
        Serial.println(longitude, 6);  }
    }
  }

  delay(5000); // Read data every 5 seconds
}
```

Assemble the Components:

- Securely place all components within the bus, taking care to protect them from environmental factors.

Test in Real Environment:

- Place the bus with the connected components in a real environment to test data collection and communication in actual bus operations.

Monitor Data:

- Use the central server or cloud platform to monitor and analyze the data coming from the bus. Ensure it's being stored and processed correctly.

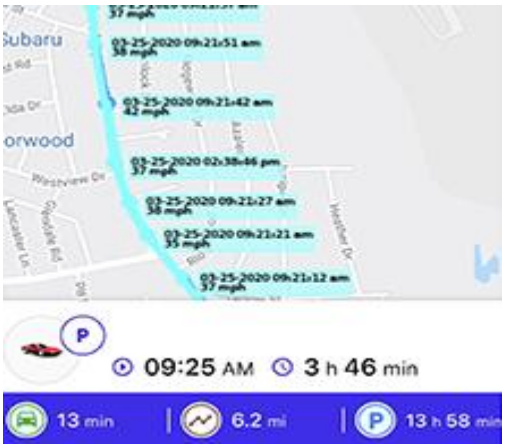
Implement Data Visualization:

- Create a web or mobile app for passengers and operators to access real-time bus data.

Maintenance and Updates:

- Regularly maintain and update the system, including sensor calibration, software updates, and hardware checks.

Output:


Subaru
orwood
03-25-2020 09:21:51 am
58 mph
03-25-2020 09:21:42 am
42 mph
03-25-2020 02:38:46 pm
37 mph
03-25-2020 09:21:27 am
38 mph
03-25-2020 09:21:21 am
35 mph
03-25-2020 09:21:12 am
37 mph
09:25 AM
3 h 46 min
13 min
6.2 mi
13 h 58 min