PUBLIC TRANSPORT OPTIMIZATION

PHASE-3

**ABSTRACT:**

The project aims to provide transit operators and authorities with invaluable insights into real-time vehicle locations, passenger boarding and alighting counts, and historical ridership patterns. Additionally, the system facilitates proactive decision-making for efficient resource allocation, maintenance scheduling, and enhanced passenger experiences.

This project embodies an integrated approach to improving public transit by employing IoT sensors and communication protocols to bridge the gap between data collection, analysis, and real-time operations. The resulting system empowers transit authorities to make data-driven decisions, optimize services, and deliver a higher quality of service to commuters. With an emphasis on scalability, security, and real-time monitoring, this Real-Time Location and Ridership Tracking System presents a promising solution for modernizing public transportation networks.

**Hardware Components:**

To send real-time location and ridership data to a transit information platform in an IoT system, you will need a combination of hardware and software components. Below is a list of the required components:

**IoT Sensors and Devices:**

GPS Module: To collect real-time location data, you'll need a GPS module or GNSS (Global Navigation Satellite System) receiver.

Ridership Sensors: Use sensors like RFID, barcode scanners, or passenger counters to collect ridership data.

Microcontroller/Processor: Use microcontrollers (e.g., Arduino, Raspberry Pi, or specialized IoT boards) to interface with sensors, process data, and communicate with the platform.

Connectivity Module: Depending on the location and platform requirements, you may need Wi-Fi, cellular (3G/4G/5G), LoRa, or other communication modules to transmit data.

**IoT Software and Firmware:**

Embedded Software/Firmware: Develop software or firmware for the IoT devices to read sensor data, process it, and prepare it for transmission.

MQTT Client: Use MQTT (Message Queuing Telemetry Transport) as a lightweight and efficient protocol to publish data to the transit information platform.Location Services Library: If you're working with GPS data, you might need a library to access and parse GPS coordinates, like the geopy library in Python.

**Software and Services:**

Python for Raspberry Pi Programming

MQTT Broker (e.g., Mosquitto) running on a server or cloud service

Transit Information Platform (Web service for data reception and analysis)

Geopy library for GPS data parsing

RFID Python libraries for RFID reader integration.

Project Steps:

**Setup Raspberry Pi:**

Set up your Raspberry Pi with Raspbian or any suitable operating system.

Install Python and necessary libraries.

Connect GPS Module:

Connect the GPS module to the Raspberry Pi using the appropriate pins.

Write Python code to read real-time GPS location data.

Integrate RFID Readers:

Connect RFID readers to the Raspberry Pi.

Write Python code to interface with the RFID readers and identify passengers using RFID cards/tags.

Data Processing:

Combine GPS and RFID data to create a data packet.

Prepare the data for MQTT transmission.

MQTT Setup:

Set up and configure an MQTT broker on a server or cloud service.

Transmit Data:

Write Python code to send the data to the MQTT broker.

Transit Information Platform:

Create a web service or application to receive data from the MQTT broker.

Store data in a database for analysis.

Real-Time Monitoring:

Implement a monitoring dashboard or notification system for real-time tracking and alerts.

Data Analysis:

Use the collected data for ridership analytics and location tracking.

User Interface:

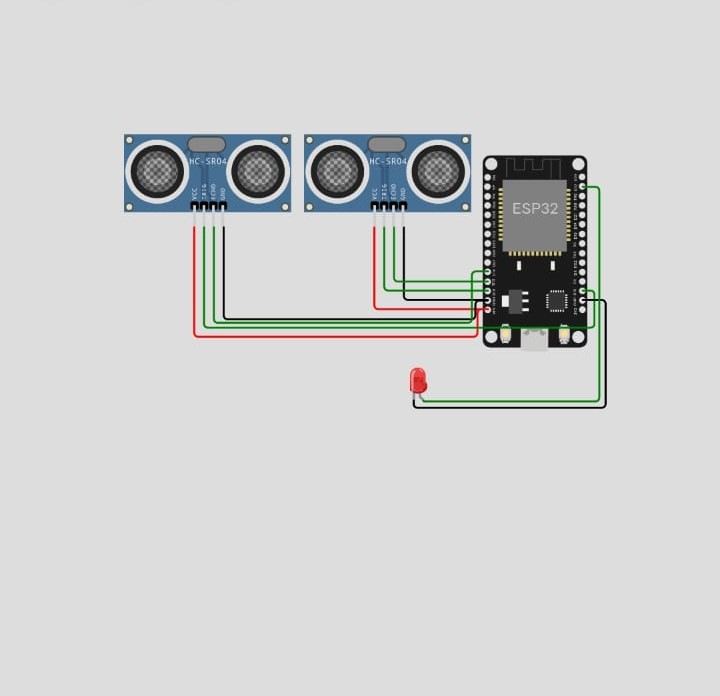
Develop a user interface for transit operators to access real-time and historical data.

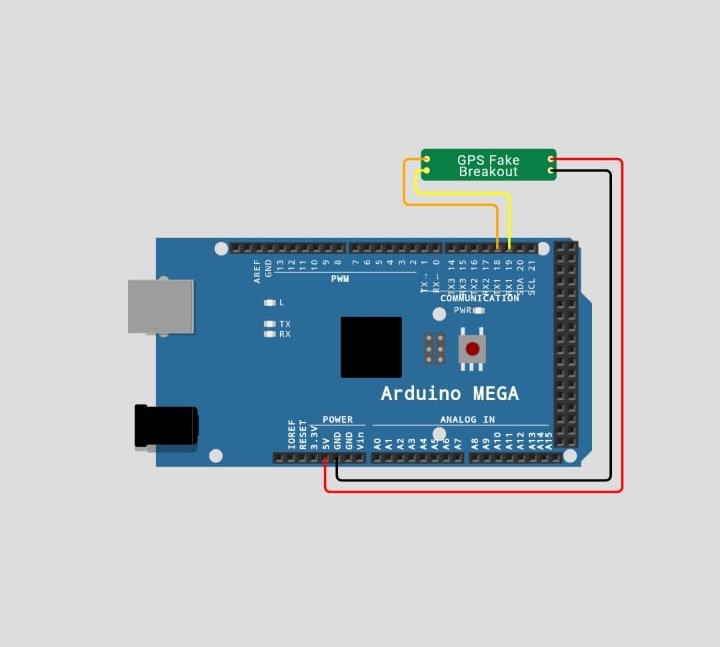
Security and Authentication:

Implement security measures to protect the data during transmission and storage.

Power Management:

Ensure that the Raspberry Pi and sensors have a reliable power supply or implement energy-efficient strategies.





**FLOWCHART:**

**+--------------------------+**

**| Transit Information |**

**| Platform (Server/ |**

**| Cloud) |**

**+--------------------------+**

**|**

**| MQTT Protocol**

**|**

**+--------------------------+**

**| MQTT Broker |**

**| (e.g., Mosquitto) |**

**+--------------------------+**

**|**

**| Wi-Fi/Cellular**

**|**

**+--------------------------+**

**| Raspberry Pi (or |**

**| Other Microcontroller)|**

**| |**

**| +------------------+ |**

**| | | |**

**| | Data | |**

**| | Processing | |**

**| | & Transmission| |**

**| | | |**

**| +------------------+ |**

**| | |**

**| | |**

**| +------------------+ |**

**| | | |**

**| | GPS Module | |**

**| | | |**

**| +------------------+ |**

**| | |**

**| | |**

**| +------------------+ |**

**| | | |**

**| | RFID Readers | |**

**| | | |**

**| +------------------+ |**

**+--------------------------+**

**SOURCE CODE:**

import requests

import time

import json

# Replace with the actual API endpoint of the transit information platform

API\_ENDPOINT = "https://transit-platform-api.com/data-endpoint"

# Function to read GPS data (simulated)

def get\_gps\_data():

# Replace this with code to read GPS data from your sensor

latitude = 123.456

longitude = 78.910

return latitude, longitude

# Function to read ridership data (simulated)

def get\_ridership\_data():

# Replace this with code to read ridership data from your sensor

riders = 50

return riders

# Main function to send data to the transit information platform

def send\_data\_to\_platform():

while True:

# Get GPS data

latitude, longitude = get\_gps\_data()

# Get ridership data

riders = get\_ridership\_data()

# Create a data payload

data = {

"latitude": latitude,

"longitude": longitude,

"riders": riders,

}

# Send data to the transit platform

response = requests.post(API\_ENDPOINT, json=data)

if response.status\_code == 200:

print("Data sent successfully")

else:

print(f"Failed to send data. Status code: {response.status\_code}")

# Wait for a specified interval (e.g., 60 seconds)

time.sleep(60)

if \_name\_ == "\_main\_":

send\_data\_to\_platform()