**Public Transportation Optimization**

Developing a Python script for Public Transport Optimization in IoT devices involves integrating various technologies and concepts.Creating a comprehensive Python script for Public Transport Optimization on IoT devices requires careful consideration of different components. Below is an program for specifying Python script focusing on real-time tracking and basic route optimization using IoT devices. For this program, we'll use MQTT (Message Queuing Telemetry Transport) protocol for communication between IoT devices.

**Python program for developing Iot devices:**

import random

import time

import json

import paho.mqtt.client as mqtt

from geopy.distance import geodesic

# MQTT Broker Settings

MQTT\_BROKER = "mqtt.eclipse.org"

MQTT\_PORT = 1883

MQTT\_TOPIC = "public\_transport"

# Example Bus Routes and Stops (Lat, Long)

BUS\_ROUTES = {

"Route1": [(40.7128, -74.0060), (40.7306, -74.0034), (40.7489, -73.9680)],

"Route2": [(40.7589, -73.9851), (40.7527, -73.9772), (40.7291, -73.9965)],

}

# Function to simulate IoT device sending location data

def simulate\_public\_transport(bus\_id, route):

client = mqtt.Client(bus\_id)

client.connect(MQTT\_BROKER, MQTT\_PORT, 60)

while True:

current\_location = random.choice(route)

client.publish(MQTT\_TOPIC, json.dumps({"bus\_id": bus\_id, "location": current\_location}))

time.sleep(10) # Simulate sending location data every 10 seconds

# Function to calculate distance between two points

def calculate\_distance(coord1, coord2):

return geodesic(coord1, coord2).kilometers

# MQTT Message Callback

def on\_message(client, userdata, message):

payload = json.loads(message.payload.decode())

bus\_id = payload["bus\_id"]

location = payload["location"]

# Simple route optimization: Choose the nearest stop

nearest\_stop = min(BUS\_ROUTES[bus\_id], key=lambda stop: calculate\_distance(stop, location))

print(f"Bus {bus\_id} is currently at {location}. Nearest stop: {nearest\_stop}")

# MQTT Setup

client = mqtt.Client("CentralServer")

client.connect(MQTT\_BROKER, MQTT\_PORT, 60)

client.subscribe(MQTT\_TOPIC)

client.on\_message = on\_message

# Start the MQTT loop

client.loop\_start()

# Simulate IoT devices (buses) sending location data

for route\_id, stops in BUS\_ROUTES.items():

bus\_id = f"Bus\_{route\_id}"

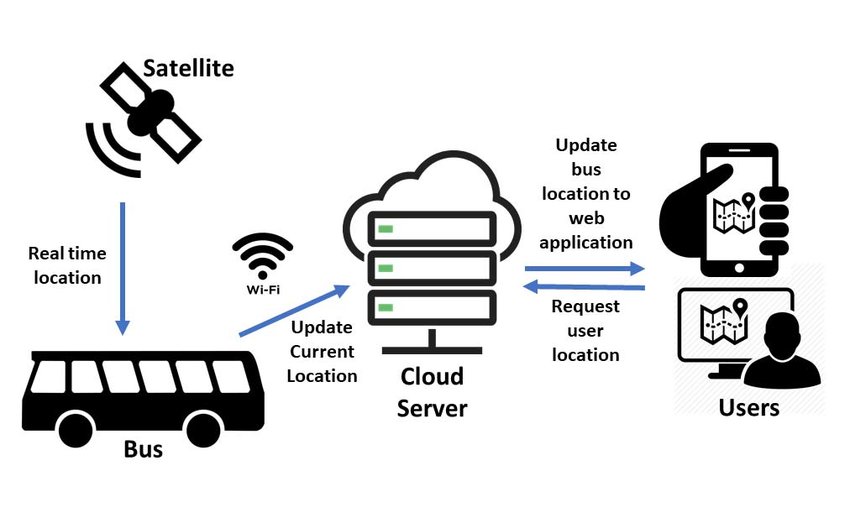
simulate\_public\_transport(bus\_id, stops)

For this Program,

1.MQTT Communication: The script uses the MQTT protocol for communication between the central server and IoT devices (simulated buses).

2.Simulated Buses: The simulate\_public\_transport function simulates buses sending their location data to the MQTT broker. You can extend this logic to integrate real IoT devices.

3.Route Optimization: The script calculates the nearest stop for each bus based on its current location. In a real scenario, you would use more sophisticated algorithms for route optimization.



**Keep in mind that IoT devices and systems can vary greatly, so adjust the script as needed for your specific hardware and requirements.**

1. Define the Problem:

Identify the specific challenges faced in public transport, such as route optimization, real-time tracking, predictive maintenance, or passenger analytics.

2. Hardware Setup:

Set up your IoT devices (sensors, GPS modules, microcontrollers) on public transport vehicles for data collection.

3. Data Collection:

Collect data from IoT devices, including GPS coordinates, speed, passenger count, and environmental conditions.

Use appropriate communication protocols (MQTT, HTTP, etc.) to transmit data to a central server or cloud platform.

4. Data Processing:

Process incoming data streams to extract relevant information.

Use Python libraries like pandas for data manipulation and analysis.

5. Real-time Tracking:

Implement real-time tracking of vehicles using GPS data.

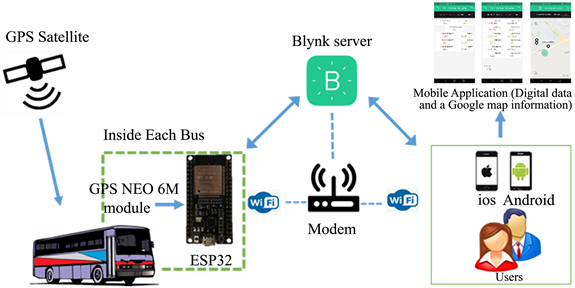
Visualize the live location of vehicles on a map using libraries like folium or web-based frameworks like Leaflet.js.

6. Route Optimization:

Utilize algorithms like Dijkstra's or A\* for route optimization.

Consider traffic conditions, stops, and passenger demand to optimize routes dynamically.

Implement logic to re-route vehicles in case of unexpected events or traffic congestion.



7. Predictive Maintenance:

Implement predictive maintenance algorithms using IoT sensor data to anticipate vehicle failures.

Send alerts or schedule maintenance tasks automatically when anomalies are detected.

8. Passenger Analytics:

Analyze passenger data to optimize schedules based on demand.

Use machine learning models for predicting passenger loads and adjust routes accordingly.

Implement features for passengers such as mobile apps to check bus schedules, delays, and available seats.

9. User Interface (Optional):

Develop a web-based or mobile interface for users and administrators.

Provide features like route planning, real-time tracking, and service updates.

10. Security and Scalability:

Implement security measures to protect data transmission and storage.

Design the system to be scalable, considering potential expansion and a large number of devices.

11. Testing and Optimization:

Test the system thoroughly under different scenarios to ensure reliability and accuracy.

Optimize algorithms and system performance based on real-world testing results.

12. Deployment:

Deploy the Python script on IoT devices, ensuring they have the necessary libraries and dependencies.

Monitor the system in real-time and have mechanisms in place for error handling and recovery.

13. Documentation and Maintenance:

Document the system architecture, algorithms used, and instructions for maintenance and troubleshooting.

Provide regular maintenance and updates to ensure the system's efficiency and security.

**Project Submitted By,**

**Name : P.Hari Prasath**

**NM I’d : au713921106017**

**Topic : Public Tranaport Optimization**

**Email I’d : hariprasath0520@gmail.com**

**College code : 7139**