**PUBLIC TRANSPORT OPTIMIZATION**

**PHASE 4**

*In this technology project you will continue building your project by developing the platform as per project requirement. Use web development technologies wherever needed. After performing the relevant activities create a document around it and share the same for assessment.*

**About ThingSpeak:**

In the endeavor to optimize public transport, the integration of real-time transit information on ThingSpeak is paramount. This process involves the amalgamation of hardware components, including GPS modules and passenger sensors installed on each bus. These devices work together to acquire and process critical data, such as location coordinates and passenger count. Subsequently, the meticulously processed data is transmitted to ThingSpeak channels via Wi-Fi or cellular networks.

A dedicated ThingSpeak channel is established to receive and display this real-time information. The channel is configured with a unique API key, ensuring secure communication between the microcontroller and the ThingSpeak platform. Through this seamless interaction, passengers and transit authorities gain access to a dynamic display of bus locations, passenger count, etc.

**Code to display Real time Transit Information on Thinkspeak:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Transit Information</title>

    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

    <style>

        body {

            font-family: Arial, sans-serif;

            margin: 20px;

        }

        #transit-info {

            border: 1px solid #ccc;

            padding: 20px;

        }

    </style>

</head>

<body>

    <h1>Real-Time Transit Information</h1>

    <div id="transit-info"></div>

    <script>

        const apiKey = 'HSVQUBXHFT54BK1V';

        function updateTransitInfo() {

            $.getJSON(`https://api.thingspeak.com/channels/ 2320111/feeds.json?api\_key=${apiKey}&results=1`, function(data) {

                const transitInfoDiv = document.getElementById('transit-info');

                transitInfoDiv.innerHTML = ''; // Clear previous content

                if (data && data.feeds && data.feeds.length > 0) {

                    const latestEntry = data.feeds[0];

                    const transitData = {

latitude: latestEntry.field1,

                        longitude: latestEntry.field2,

                        speed: latestEntry.field3,

                        directions: latestEntry.field4,

arrival time: latestEntry.field5,

passenger count: latestEntry.field6

                    };

                    const div = document.createElement('div');

                    div.innerHTML = `<strong>${transitData.route}</strong> | Location: ${transitData.location} | Passengers: ${transitData.passengers}`;

                    transitInfoDiv.appendChild(div);

                }

            });

        }

        // Update transit information every 5 seconds (adjust the interval as needed)

        setInterval(updateTransitInfo, 5000);

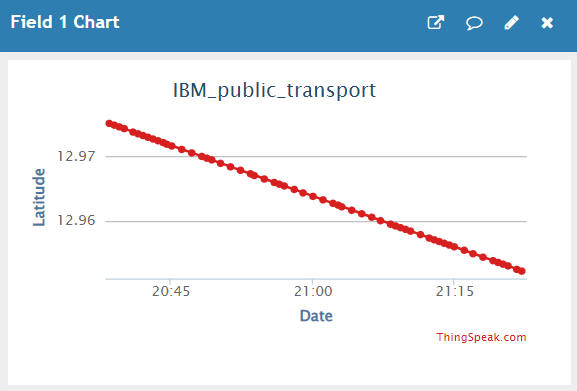
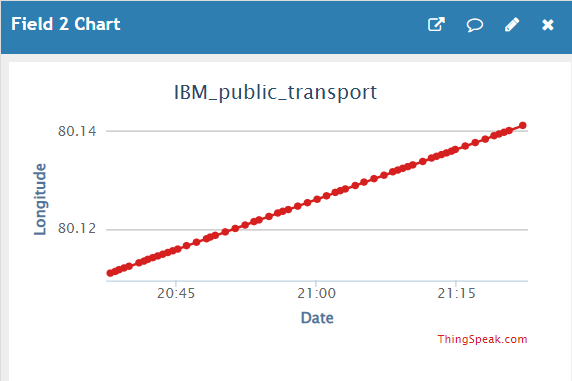
        // Initial update

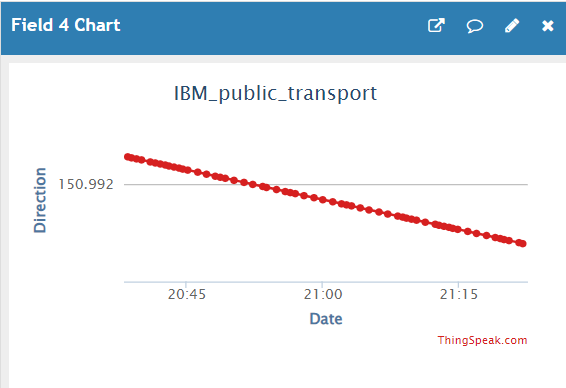
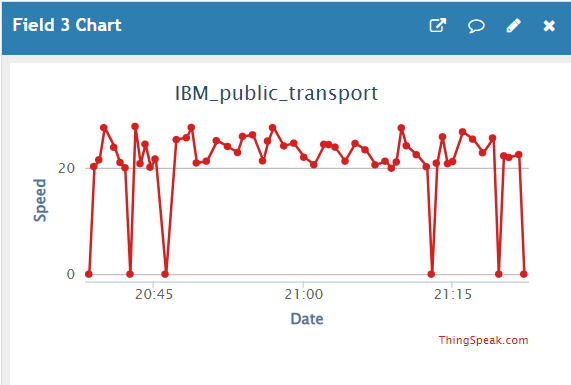
        updateTransitInfo();

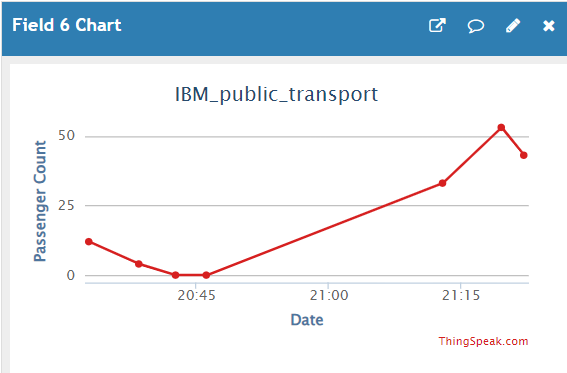
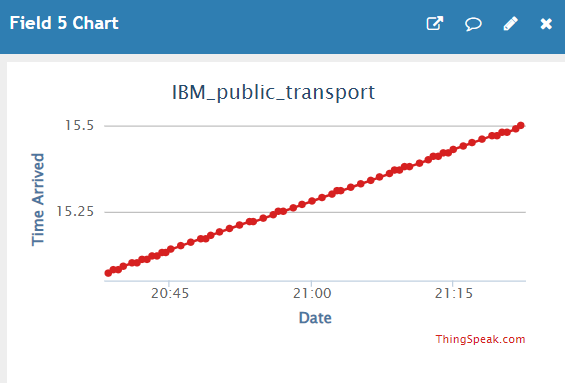
    </script>

</body>

</html>

Output:





Code for Interfacing Data with ThinkSpeak:

import geopy.distance

import requests

import time

import math

import random

import numpy as np

from datetime import datetime, timedelta

# Replace with your ThingSpeak API key and the data you want to send

api\_key = 'HSVQUBXHFT54BK1V'

# Kundrathur coordinates

kundrathur\_coords = (12.9790, 80.1064)

# Chromepet coordinates

chromepet\_coords = (12.9518, 80.1415)

# Bus stops along the path

bus\_stops = [

    {"coords": (12.9790, 80.1064), "name": "Stop 1", "passenger\_change": 10},

    {"coords": (12.9720, 80.1100), "name": "Stop 2", "passenger\_change": -5},

    {"coords": (12.9680, 80.1165), "name": "Stop 3", "passenger\_change": 15},

    {"coords": (12.9650, 80.1200), "name": "Stop 4", "passenger\_change": -8},

    {"coords": (12.9620, 80.1250), "name": "Stop 5", "passenger\_change": 12},

    {"coords": (12.9600, 80.1280), "name": "Stop 6", "passenger\_change": -7},

    {"coords": (12.9580, 80.1320), "name": "Stop 7", "passenger\_change": 20},

    {"coords": (12.9550, 80.1350), "name": "Stop 8", "passenger\_change": -10},

    {"coords": (12.9520, 80.1390), "name": "Stop 9", "passenger\_change": 8},

    {"coords": (12.9490, 80.1420), "name": "Stop 10", "passenger\_change": -12},

]

# Generate coordinates along the path

def generate\_path(start\_coords, end\_coords, num\_points):

    path = []

    for i in range(num\_points + 1):

        fraction = i / num\_points

        lat = start\_coords[0] + fraction \* (end\_coords[0] - start\_coords[0])

        lon = start\_coords[1] + fraction \* (end\_coords[1] - start\_coords[1])

        path.append((lat, lon))

    return path

# Generate a path with 100 points

selected\_stops = random.sample(path\_coords, len(bus\_stops))

# Calculate distances between consecutive points to simulate speed

speed\_values = [20]  # Initial speed

for i in range(1, len(path\_coords)):

    distance\_km = geopy.distance.distance(path\_coords[i - 1], path\_coords[i]).km

    time\_hours = distance\_km / speed\_values[-1]  # Assume constant speed

    speed\_values.append(distance\_km / time\_hours)

# Calculate bearing and map to 8 directions

def calculate\_bearing(point1, point2):

    lat1, lon1 = point1

    lat2, lon2 = point2

    delta\_lon = lon2 - lon1

    x = math.cos(math.radians(lat2)) \* math.sin(math.radians(delta\_lon))

    y = math.cos(math.radians(lat1)) \* math.sin(math.radians(lat2)) - \

        math.sin(math.radians(lat1)) \* math.cos(math.radians(lat2)) \* math.cos(math.radians(delta\_lon))

    bearing = math.atan2(x, y)

    bearing = math.degrees(bearing)

    bearing = (bearing + 360) % 360  # Ensure bearing is between 0 and 360 degrees

    return bearing

# Calculate bearings between consecutive points

bearings = [calculate\_bearing(path\_coords[i - 1], path\_coords[i]) for i in range(1, len(path\_coords))]

# Map bearings to 8 directions

directions = []

for bearing in bearings:

    angle = bearing + 22.5  # Offset by half the angle to get more accurate direction

    if angle < 0:

        angle += 360

    direction\_index = int(angle / 45) % 8

    direction = ["N", "NE", "E", "SE", "S", "SW", "W", "NW"][direction\_index]

    directions.append(angle)

# Extract latitude and longitude values

latitude\_values, longitude\_values = zip(\*path\_coords)

# Simulate passenger count and stops

passenger\_count = 0

stop\_interval = 5  # Time interval between bus stops in minutes

update\_interval = 30  # Update interval in seconds

# Generate timestamps

start\_timestamp = datetime.now()

timestamps = [start\_timestamp + timedelta(seconds=i \* update\_interval) for i in range(len(speed\_values))]

# Display the first few values

print("Latitude values:", latitude\_values[:5])

print("Longitude values:", longitude\_values[:5])

print("Speed values:", speed\_values[:5])

print("Directions:", directions[:5])

for i in range(len(speed\_values)):

    data = {

        'field1': str(latitude\_values[i]),

        'field2': str(longitude\_values[i]),

        'field3': str(max(0, speed\_values[i] + 10 - random.uniform(2, 10))),  # Speed is non-negative

        'field4': directions[i],

        'field5': timestamps[i].strftime("%H.%M"),  # Format timestamp as string

    }

    print(timestamps[i].strftime("%H.%M"))

    # Simulate stops

    for stop,passenger in zip(selected\_stops, bus\_stops):

        if stop == (latitude\_values[i], longitude\_values[i]):

            print("bus stop")

            passenger\_count = max(0, passenger\_count + passenger["passenger\_change"])  # Ensure passenger count is non-negative

            data['field6'] = str(passenger\_count)

            data['field3'] = '0'  # Speed is 0 at stops

            print(f"Bus stopped at {passenger['name']}. Passenger count: {passenger\_count}")

    # Update ThingSpeak with a delay

    time.sleep(update\_interval)

    # ThingSpeak update URL

    url = f'https://api.thingspeak.com/update?api\_key={api\_key}'

    response = requests.post(url, data=data)

    if response.status\_code == 200:

        print("Data uploaded successfully to ThingSpeak")

    else:

        print("Failed to upload data")

**MOBILE APPLICATION:**

The application MIT App Inventor plays a crucial role in the seamless operation of this system. It serves as the vital link between passengers and the server, offering essential information about theLatitude, longitude, speed, distance, arrival time and Passenger count . The application is crafted using the Blynk platform, an innovative IoT platform renowned for its ability to swiftly create impressive applications for both Android and iOS smartphones.

The architecture of the Blynk platform encompasses Blynk libraries, the Blynk server, and Blynk apps. Within the Android application it supplies passengers with estimated arrival times of the buses, reports the speed of the bus in transit, and presents the nearest available bus based on calculations of proximity between the passenger and the bus's location.

1. ThingSpeak Configuration:

* Data, such as real-time transit information, was collected and stored on ThingSpeak channels.
* Each data point included relevant fields such as route information, location, and passenger count.

1. MIT App Inventor Setup:

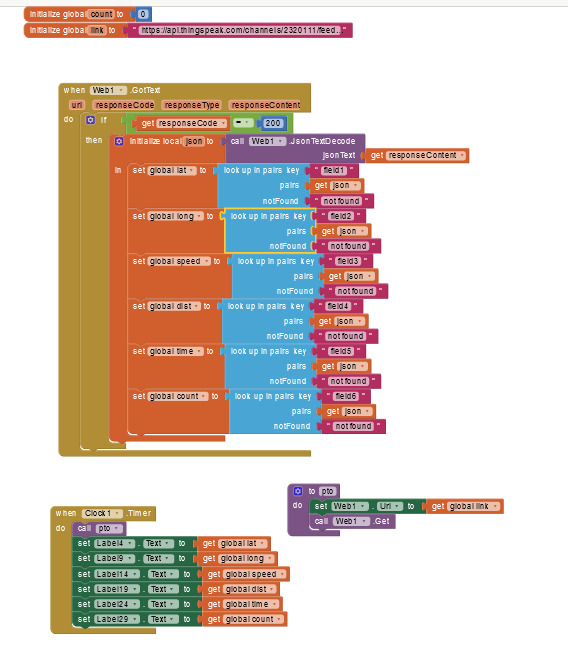
* MIT App Inventor's visual interface was utilized for app development.
* Components such as labels, text boxes, and web components were employed to design the app's user interface.

1. Data Retrieval:

* The MIT App Inventor app was configured to retrieve data from ThingSpeak channels using the Web component.
* API calls were made to ThingSpeak to fetch the latest data entries.

1. Real-Time Display:

* The retrieved data, including route information, location, and passenger count, was dynamically displayed on the MIT App Inventor app's interface.



**Team members:**

Muhil J (2021504530)

Shakti A P (2021504541)

Shamitha G (2021504543)

Shreejan A (2021504544)

Sushmitha A (2021504551)