PUBLIC TRANSPORT OPTIMIZATION

Phase 5 Submission

Phase 5 : Final Submission of project

Project Definition

The project aims to enhance public transportation services by integrating IoT sensors into public transportation vehicles. These sensors will be used to monitor ridership, track vehicle locations, and predict arrival times. The ultimate goal is to provide real-time transit information to the public through a user-friendly web-based platform. This project involves defining specific objectives, designing the IoT sensor system, developing the real-time transit information platform, and integrating them using IoT technology and Python.

Design Thinking Project Objectives

To address the challenges faced by public transportation systems, we have defined the following objectives:

1. **Real-time Transit Information**: Provide passengers with real-time information about public transportation services, including vehicle locations and arrival times.
2. **Arrival Time Prediction**: Develop algorithms to predict accurate arrival times based on real-time data and historical patterns.
3. **Ridership Monitoring**: Implement passenger counting mechanisms to monitor and report ridership on public transportation vehicles.
4. **Enhanced Public Transportation Services**: Utilize IoT technology to optimize public transportation schedules, reduce costs, and improve overall service quality.

IoT Sensor Design

To achieve the objectives, we will design an IoT sensor system that includes the following components:

1. **GPS Sensors**: These sensors will continuously track the real-time location of public transportation vehicles.
2. **Passenger Counters**: Deploy passenger counting sensors, such as infrared sensors or cameras, to accurately monitor ridership at each stop.
3. **Additional Sensors**: Consider adding supplementary sensors to collect environmental data, such as temperature, humidity, and air quality, to improve passenger experience and address environmental concerns.
4. **Microcontroller**: Connect the sensors to a microcontroller that will collect, process, and manage the data locally within the vehicles.

5. **Connectivity**: Ensure reliable connectivity through cellular networks, Wi-Fi, or other wireless technologies for real-time data transmission.

Real-Time Transit Information Platform

To provide passengers with real-time transit information, we will design a web-based platform with the following features:

1. **Real-Time Vehicle Location**: Display the live locations of public transportation vehicles on a user-friendly map interface.
2. **Predicted Arrival Times**: Develop machine learning algorithms to predict arrival times accurately based on real-time data and historical patterns.
3. **Ridership Information**: Present current ridership levels on each vehicle to help passengers make informed travel decisions.
4. **Service Updates**: Provide relevant information on service disruptions, delays, and other announcements impacting passengers.

Integration Approach

To seamlessly integrate the IoT sensor system with the real-time transit information platform, we will use IoT communication technologies such as MQTT or AMQP. These technologies enable efficient and lightweight data exchange between devices and cloud-based applications.

Solution Overview:

The proposed solution involves the integration of IoT sensors into public transportation vehicles, real-time data processing, and a user-friendly public platform. The key components of this solution are:

IoT Sensor Deployment:

- **GPS Sensors**: Install GPS sensors on vehicles for real-time location tracking.
- **Passenger Counters**: Implement passenger counting sensors, like infrared sensors or cameras, to monitor ridership.
- **Environmental Sensors**: Consider adding sensors for temperature, humidity, and air quality for passenger comfort and environmental monitoring.
- **Microcontrollers**: Connect sensors to microcontrollers within the vehicles to collect, process, and manage data locally.

- **Connectivity**: Ensure reliable connectivity, using cellular networks, Wi-Fi, or other wireless technologies for real-time data transmission.

Real-Time Transit Information Platform:

- **Web-Based Interface**: Develop a user-friendly webbased platform accessible to the public.
- **Real-Time Vehicle Location**: Display live vehicle locations on a map for passengers to track their ride.
- **Predicted Arrival Times**: Utilize machine learning algorithms to predict accurate arrival times based on real-time data and historical patterns.
- **Ridership Information**: Present current ridership levels on each vehicle to assist passengers in choosing less crowded rides.
- **Service Updates**: Provide relevant information on service disruptions, delays, and other announcements.

Integration of IoT Sensors and Platform:

- **IoT Communication Technologies**: Employ MQTT or AMQP to facilitate seamless integration between the IoT sensor system and the real-time transit information platform.

Benefits:

1. Improved Efficiency:

- Real-time data collected from IoT sensors enables better route planning and resource allocation.
- Operators can adjust routes and schedules dynamically, reducing delays and overcrowding.
- Optimized operations lead to improved on-time performance and reliability.

2. Enhanced Passenger Experience:

- Passengers gain access to accurate arrival time predictions, reducing wait times and uncertainty.
- Information on crowded vehicles allows passengers to make informed decisions and choose less congested routes.

- Overall, the passenger experience is significantly enhanced, leading to increased rider satisfaction.

3. **Cost Reduction:**

    i. Optimization of routes and schedules based on real-time data helps reduce fuel consumption and operational costs.

    ii. Efficient resource allocation minimizes the need for excess vehicles and staff during low-demand periods.

4. **Environmental Impact:**

    - By optimizing routes, reducing congestion, and minimizing idle times, this solution contributes to reduced carbon emissions.
    - Public transportation becomes a more environmentally sustainable choice, aligning with global efforts to combat climate change.

Simulation using Python:

Microcontroller Naming:

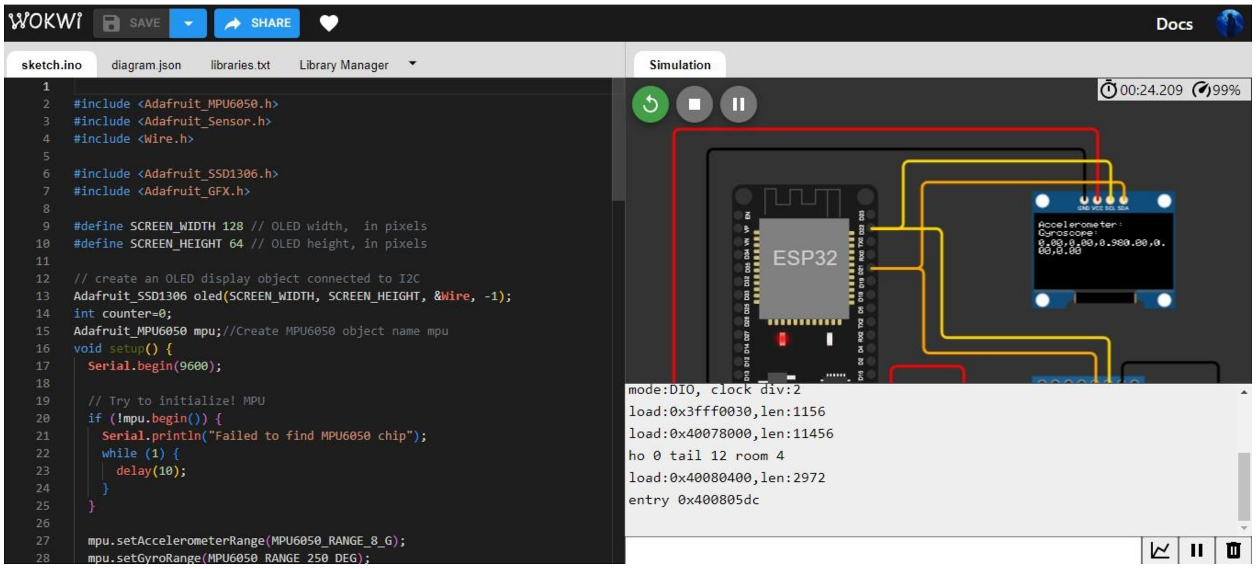
ESP32:

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 realtime 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 realtime 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:**

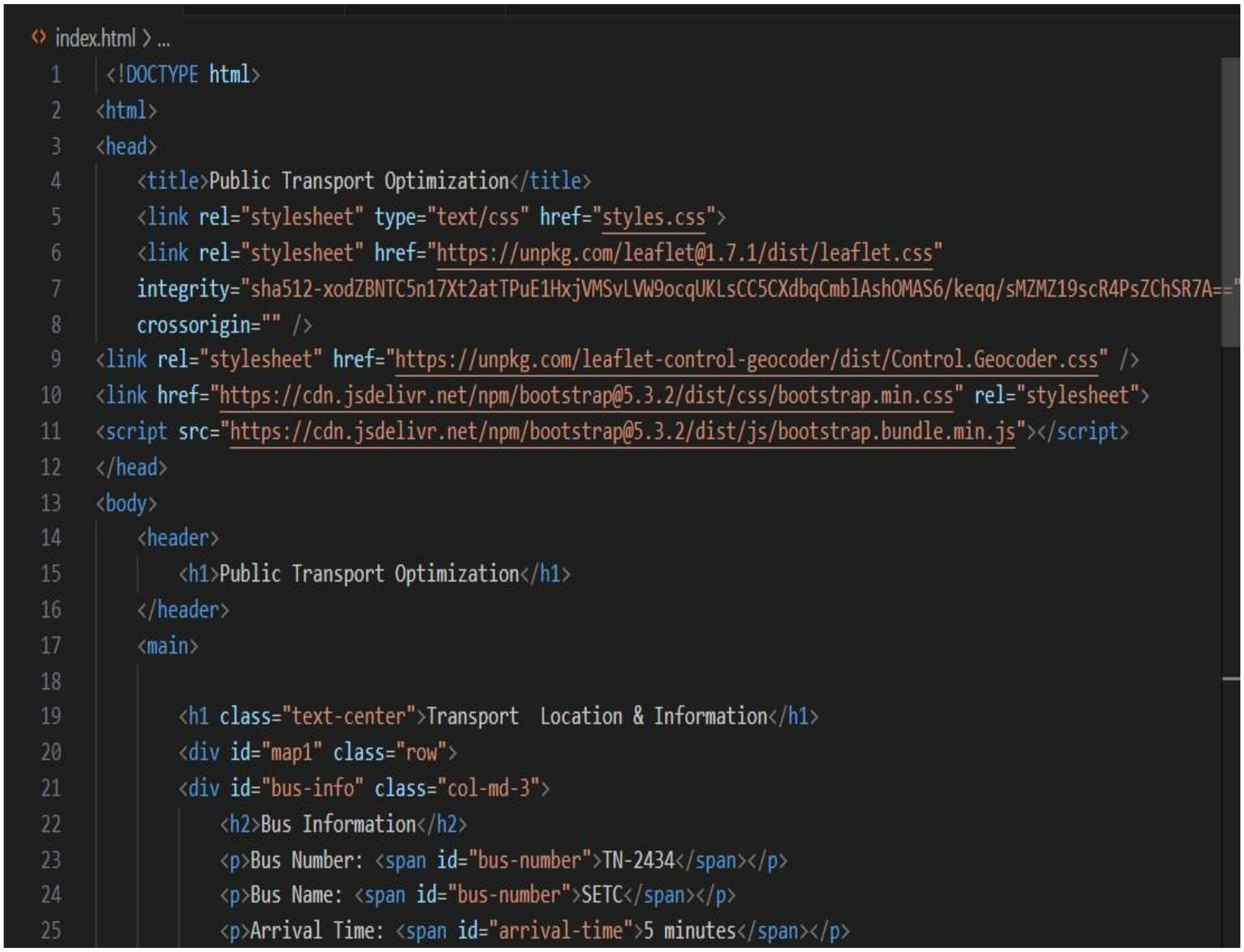
- Alternative fuel vehicles: Integrate eco-friendly buses, trams, and trains into the fleet to reduce carbon emissions.
- Renewable energy sources: Power transportation hubs with renewable energy to reduce the carbon footprint.

10. **Data Security and Privacy Measures:**

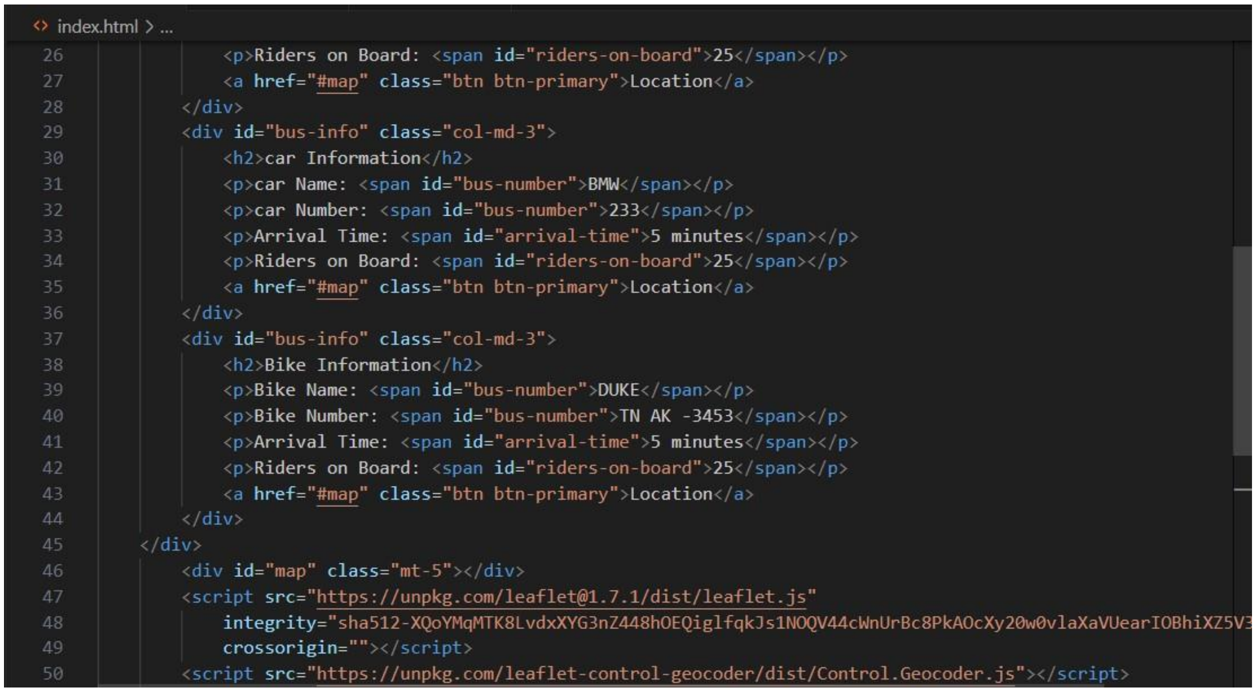
- Implement robust security protocols to protect sensitive passenger data and the integrity of the system.

Developing Web Platform:

1.index.html

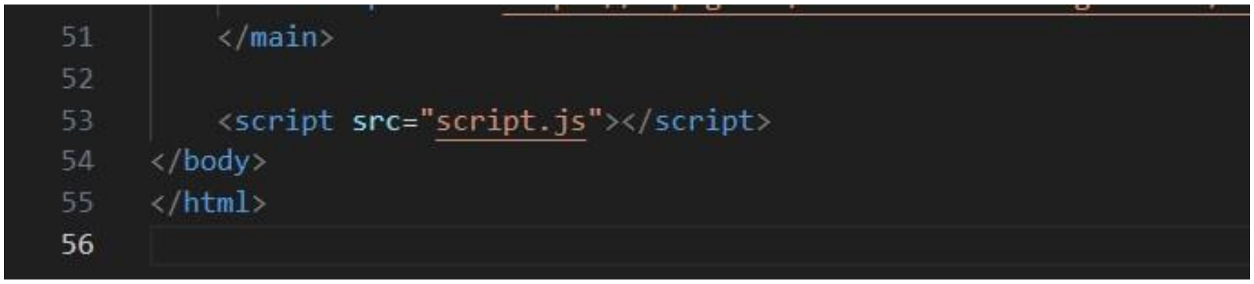


```
index.html > ...
 <!DOCTYPE html>
<html>
<head>
    <title>Public Transport Optimization</title>
    <link rel="stylesheet" type="text/css" href="styles.css">
    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css"
    integrity="sha512-xodZBNTC5n17Xt2atTPuE1HxjVMSvLVW9ocqUKLsCC5CXdbqCmblAshOMAS6/keqq/sMZMZ19scR4PsZChSR7A=="
    crossorigin="" />
<link rel="stylesheet" href="https://unpkg.com/leaflet-control-geocoder/dist/Control.Geocoder.css" />
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet">
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js"></script>
</head>
<body>
    <header>
        <h1>Public Transport Optimization</h1>
    </header>
    <main>

        <h1 class="text-center">Transport  Location & Information</h1>
        <div id="map1" class="row">
        <div id="bus-info" class="col-md-3">
            <h2>Bus Information</h2>
            <p>Bus Number: <span id="bus-number">TN-2434</span></p>
            <p>Bus Name: <span id="bus-number">SETC</span></p>
            <p>Arrival Time: <span id="arrival-time">5 minutes</span></p>
```



index.html > ...

```html
                <p>Riders on Board: <span id="riders-on-board">25</span></p>
                <a href="#map" class="btn btn-primary">Location</a>
            </div>
            <div id="bus-info" class="col-md-3">
                <h2>car Information</h2>
                <p>car Name: <span id="bus-number">BMW</span></p>
                <p>car Number: <span id="bus-number">233</span></p>
                <p>Arrival Time: <span id="arrival-time">5 minutes</span></p>
                <p>Riders on Board: <span id="riders-on-board">25</span></p>
                <a href="#map" class="btn btn-primary">Location</a>
            </div>
            <div id="bus-info" class="col-md-3">
                <h2>Bike Information</h2>
                <p>Bike Name: <span id="bus-number">DUKE</span></p>
                <p>Bike Number: <span id="bus-number">TN AK -3453</span></p>
                <p>Arrival Time: <span id="arrival-time">5 minutes</span></p>
                <p>Riders on Board: <span id="riders-on-board">25</span></p>
                <a href="#map" class="btn btn-primary">Location</a>
            </div>
        </div>
            <div id="map" class="mt-5"></div>
            <script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js"
                integrity="sha512-XQoYMqMTK8LvdxXYG3nZ448hOEQiglfqkJs1NOQV44cWnUrBc8PkAOcXy20w0vlaXaVUearIOBhiXZ5V3
                crossorigin=""></script>
            <script src="https://unpkg.com/leaflet-control-geocoder/dist/Control.Geocoder.js"></script>
```



```html
    </main>

    <script src="script.js"></script>
</body>
</html>
```

1. **<!DOCTYPE html>**: This declaration specifies that the document follows HTML5 standards.

2. **<html>**: The root element of an HTML document.

3. **<head>**: This section contains metadata and links to external resources. In this case, it links to an external CSS file and sets the document's character encoding.

4. **<meta charset="UTF-8">**: Specifies the character encoding of the document as UTF-8, which is a widely used encoding for handling text in different languages.

5. **<title>Public Transport Ptimization Platform</title>**: Sets the title of the web page, which appears in the browser's title bar or tab.

6. **<link rel="stylesheet" type="text/css" href="styles.css">**: Links an external CSS file named "styles.css" to style the web page.

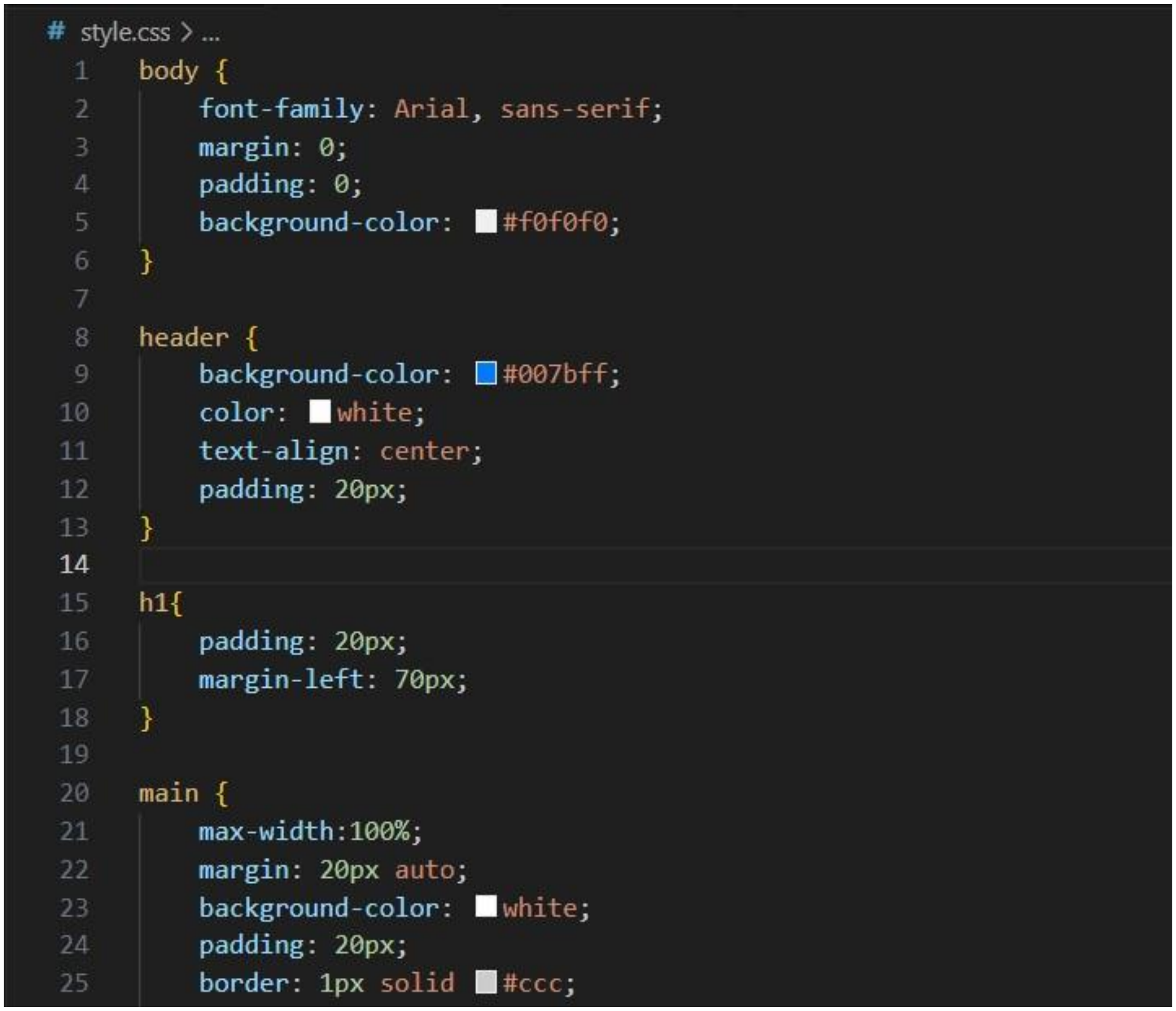
7. **<body>**: The main content of the web page is contained within the **<body>** element.

8. **<header>**: The header section typically contains the title or logo of the website. In this case, it displays the title "Environmental Monitoring Platform."
9. **<nav>**: This section contains navigation links. It's structured as an unordered list **<ul>** with list items **<li>**, each of which contains an anchor **<a>** element for navigation.
10. **<p align="center">**: This paragraph element aligns its text to the center. However, the **align** attribute is deprecated in HTML5, and it's recommended to use CSS for text alignment.
11. The **<p>** element provides information about environmental monitoring, its purpose, and what it encompasses. It's a description of the environmental monitoring concept.
12. **<section id="updates">**: This section is labeled "Latest Updates" and contains an unordered list **<ul>** of updates. Each update is presented as a list item **<li>** with a date and a description.
13. **<main>**: The main content of the web page, which contains various sections related to real-time data, data visualization, and information about the platform.
14. **<section class="sensor-data">**: This section is dedicated to displaying real-time sensor data. It contains two data items, "Temperature" and "Humidity," each with a heading **<h3>** and a placeholder paragraph **<p>** for data to be loaded via JavaScript.
15. **<section class="data-visualization">**: This section is for data visualization. It currently includes a canvas element **<canvas>** with the ID "chart" where data visualization elements like charts or graphs can be added. Additionally, it includes a script to include the Chart.js library for creating charts and another canvas element with the ID "lineChart."
16. **<section id="about">**: This section provides information about the platform, its mission, and what it offers.
17. **<section id="contact">**: This section offers contact information, including an email address where users can reach out with questions or feedback.
18. **<section id="disclaimer">**: This section includes a disclaimer regarding the informational nature of the platform and advises users to consult with experts for critical decisions related to the environment.

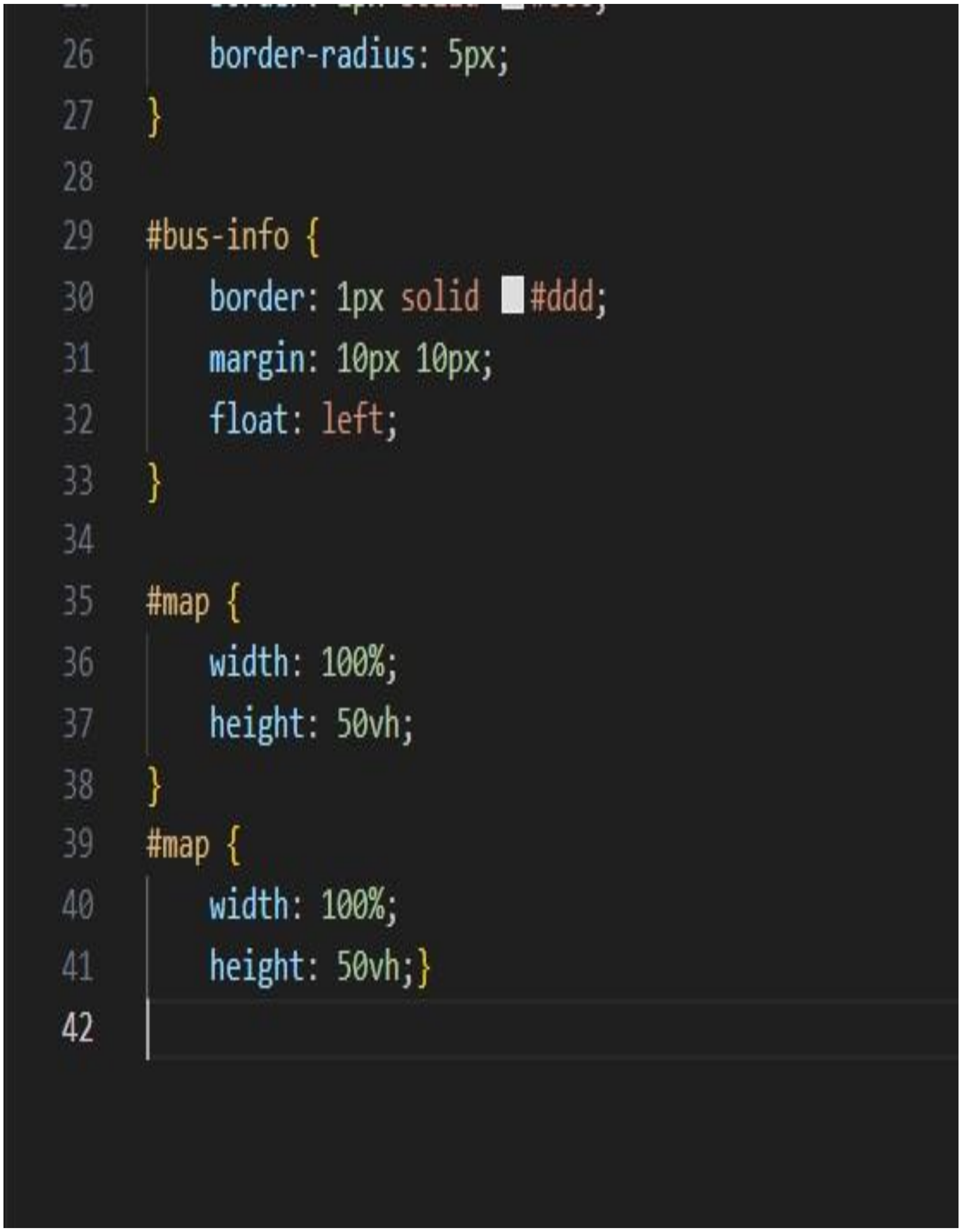
19. **<footer>**: The footer section displays a copyright notice for the year 2023, indicating ownership of the content.

20. **<script src="node.js"></script>**: This script element links to an external JavaScript file named "node.js." This file is used for real-time updates and functionality related to the platform.

2.style.css

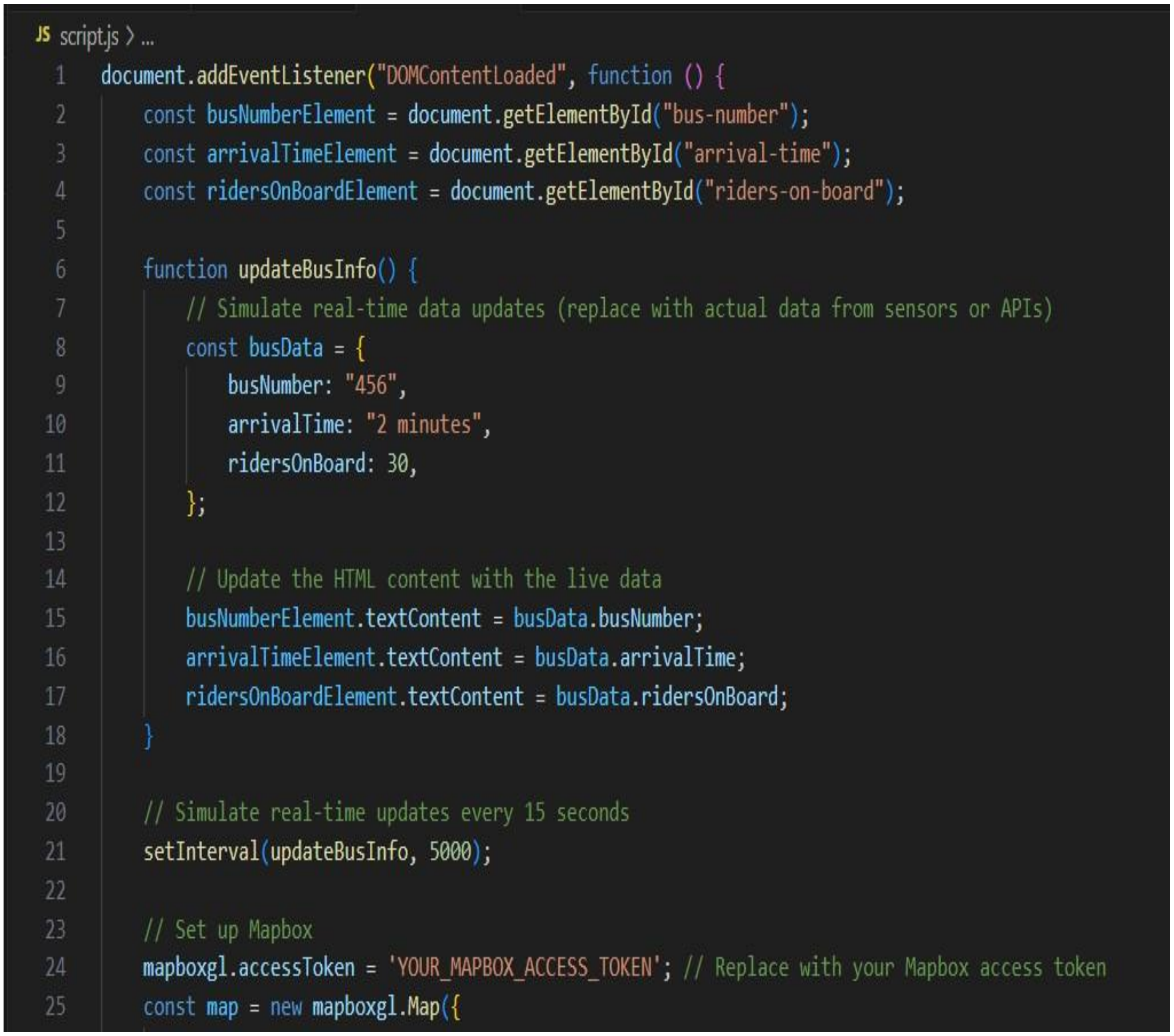


```css
# style.css > ...
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-color: #f0f0f0;
}

header {
    background-color: #007bff;
    color: white;
    text-align: center;
    padding: 20px;
}

h1{
    padding: 20px;
    margin-left: 70px;
}

main {
    max-width:100%;
    margin: 20px auto;
    background-color: white;
    padding: 20px;
    border: 1px solid #ccc;
```

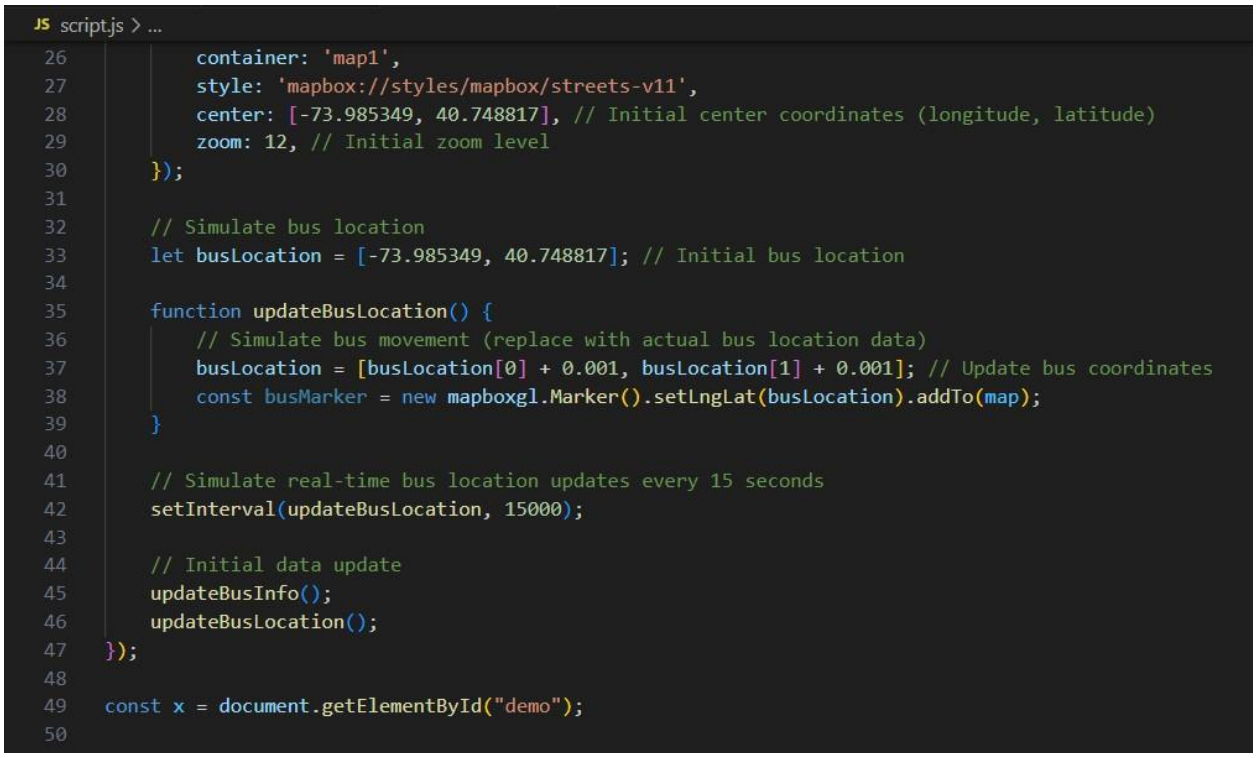


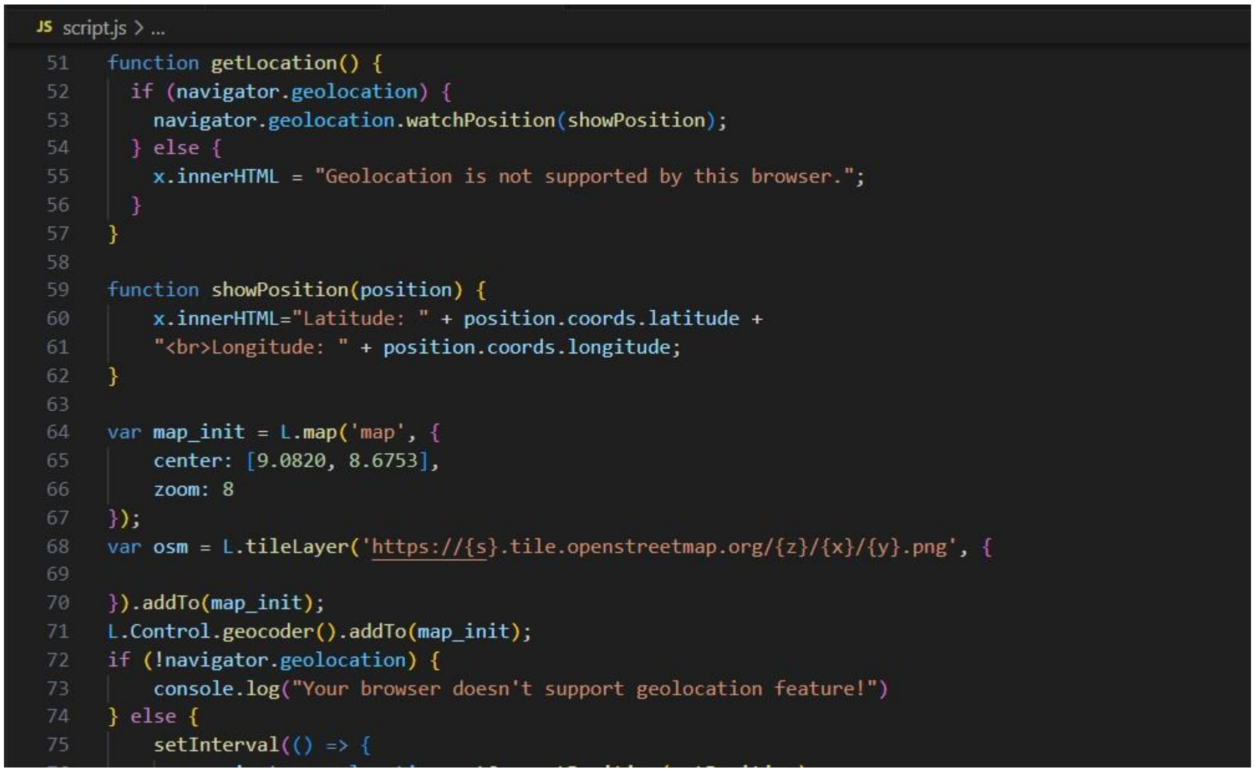
```css
    border-radius: 5px;
}

#bus-info {
    border: 1px solid #ddd;
    margin: 10px 10px;
    float: left;
}

#map {
    width: 100%;
    height: 50vh;
}
#map {
    width: 100%;
    height: 50vh;}

```

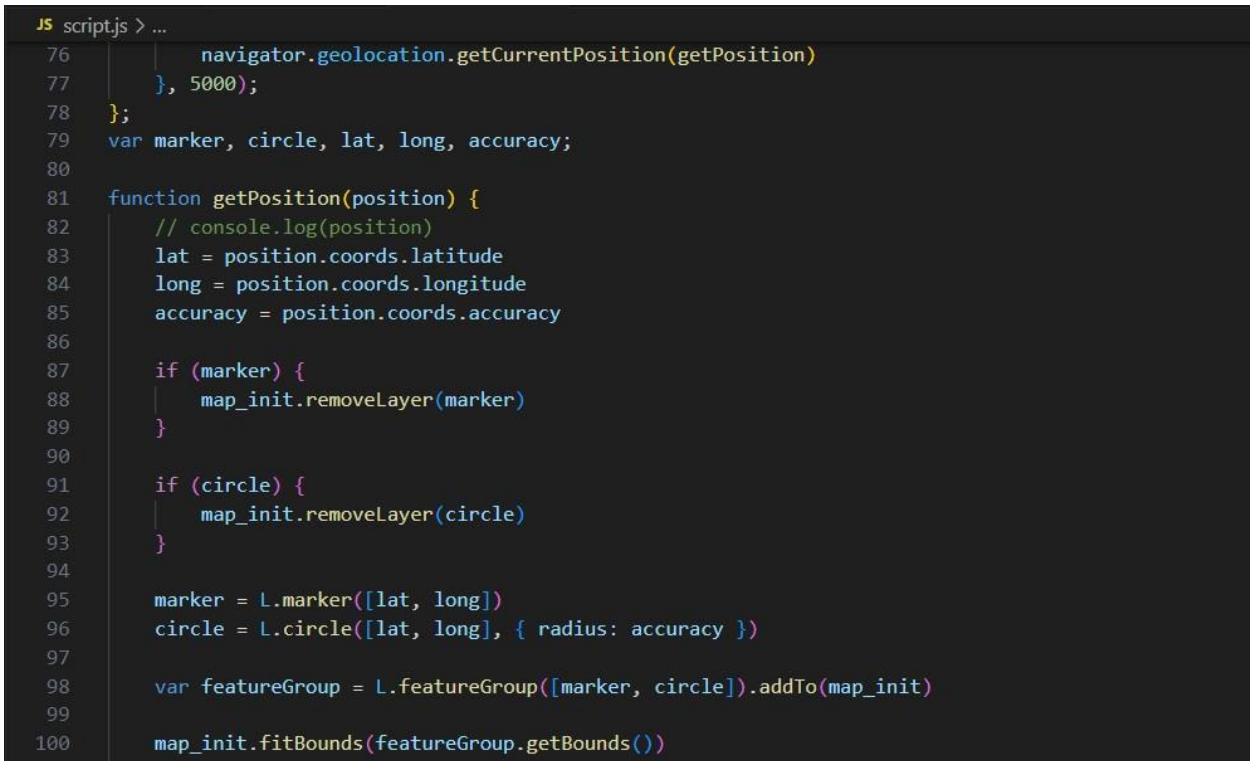
3.script.js



```
JS script.js > ...
document.addEventListener("DOMContentLoaded", function () {
    const busNumberElement = document.getElementById("bus-number");
    const arrivalTimeElement = document.getElementById("arrival-time");
    const ridersOnBoardElement = document.getElementById("riders-on-board");

    function updateBusInfo() {
        // Simulate real-time data updates (replace with actual data from sensors or APIs)
        const busData = {
            busNumber: "456",
            arrivalTime: "2 minutes",
            ridersOnBoard: 30,
        };

        // Update the HTML content with the live data
        busNumberElement.textContent = busData.busNumber;
        arrivalTimeElement.textContent = busData.arrivalTime;
        ridersOnBoardElement.textContent = busData.ridersOnBoard;
    }

    // Simulate real-time updates every 15 seconds
    setInterval(updateBusInfo, 5000);

    // Set up Mapbox
    mapboxgl.accessToken = 'YOUR_MAPBOX_ACCESS_TOKEN'; // Replace with your Mapbox access token
    const map = new mapboxgl.Map({
```

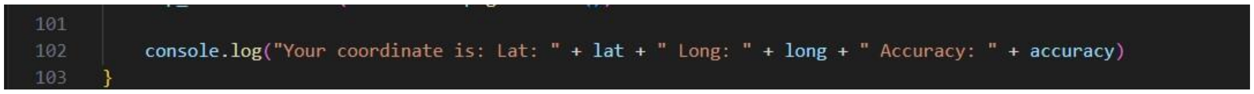

JS script.js > ...
```javascript
        container: 'map1',
        style: 'mapbox://styles/mapbox/streets-v11',
        center: [-73.985349, 40.748817], // Initial center coordinates (longitude, latitude)
        zoom: 12, // Initial zoom level
    });

    // Simulate bus location
    let busLocation = [-73.985349, 40.748817]; // Initial bus location

    function updateBusLocation() {
        // Simulate bus movement (replace with actual bus location data)
        busLocation = [busLocation[0] + 0.001, busLocation[1] + 0.001]; // Update bus coordinates
        const busMarker = new mapboxgl.Marker().setLngLat(busLocation).addTo(map);
    }

    // Simulate real-time bus location updates every 15 seconds
    setInterval(updateBusLocation, 15000);

    // Initial data update
    updateBusInfo();
    updateBusLocation();
});

const x = document.getElementById("demo");

```


JS script.js > ...
```javascript
function getLocation() {
  if (navigator.geolocation) {
    navigator.geolocation.watchPosition(showPosition);
  } else {
    x.innerHTML = "Geolocation is not supported by this browser.";
  }
}

function showPosition(position) {
    x.innerHTML="Latitude: " + position.coords.latitude +
    "<br>Longitude: " + position.coords.longitude;
}

var map_init = L.map('map', {
    center: [9.0820, 8.6753],
    zoom: 8
});
var osm = L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {

}).addTo(map_init);
L.Control.geocoder().addTo(map_init);
if (!navigator.geolocation) {
    console.log("Your browser doesn't support geolocation feature!")
} else {
    setInterval(() => {
```



JS script.js > ...

```javascript
        navigator.geolocation.getCurrentPosition(getPosition)
    }, 5000);
};
var marker, circle, lat, long, accuracy;

function getPosition(position) {
    // console.log(position)
    lat = position.coords.latitude
    long = position.coords.longitude
    accuracy = position.coords.accuracy

    if (marker) {
        map_init.removeLayer(marker)
    }

    if (circle) {
        map_init.removeLayer(circle)
    }

    marker = L.marker([lat, long])
    circle = L.circle([lat, long], { radius: accuracy })

    var featureGroup = L.featureGroup([marker, circle]).addTo(map_init)

    map_init.fitBounds(featureGroup.getBounds())
```



```javascript

    console.log("Your coordinate is: Lat: " + lat + " Long: " + long + " Accuracy: " + accuracy)
}
```

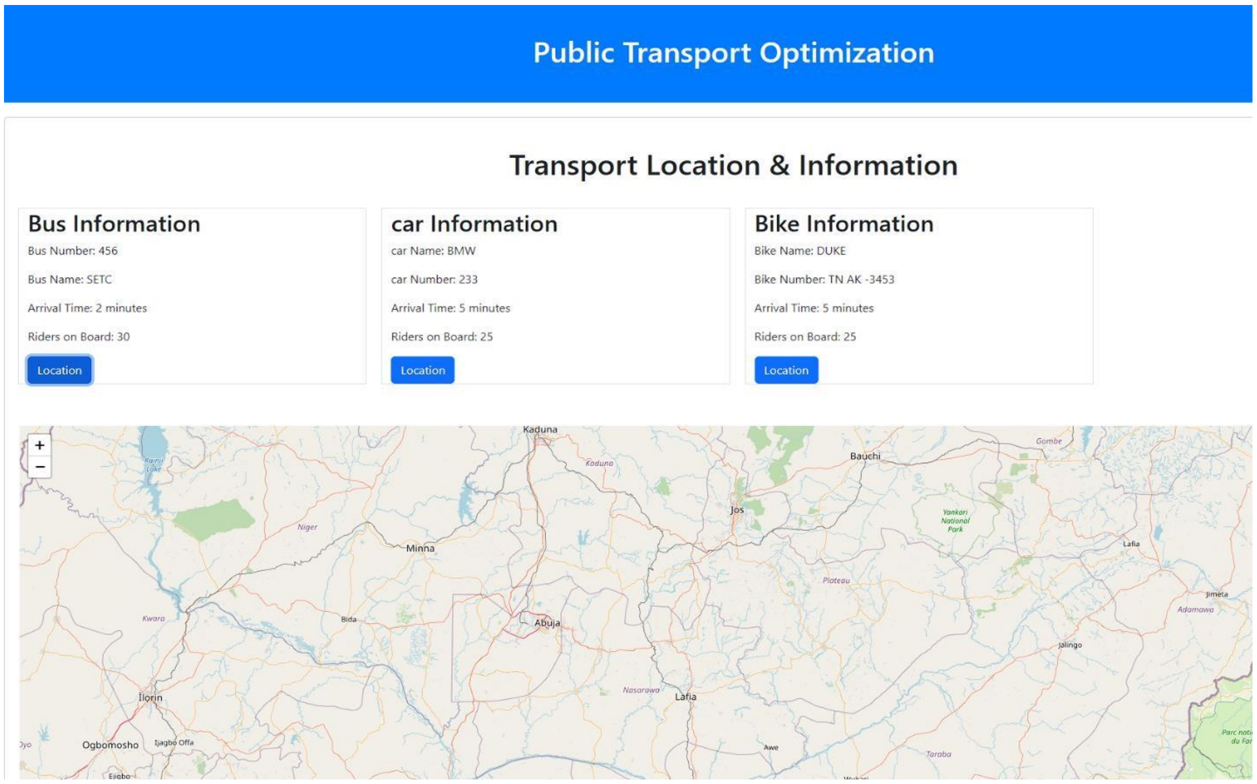
The provided JavaScript code contains functions to update sensor data, create a random chart for data visualization, and create a line chart using the Chart.js library. Let's break down the code and provide an explanation:

1. **updateSensorData Function**:
    - This function simulates the update of temperature and humidity values. In a real application, you would replace the random data with actual sensor data.
    - It generates random values for temperature and humidity within specified ranges.
    - It updates the HTML elements with the new values, specifically the elements with IDs "temperature" and "humidity."
2. **createRandomChart Function**:
    - This function creates a random chart for data visualization. Please note that in a real application, you would use a real charting library and actual data.

- It utilizes the Chart.js library to create a line chart.
- The chart includes two datasets (Temperature and Humidity) with random data points.
- The chart's configuration includes labels, colors, and other properties.
- The chart is drawn on the canvas element with the ID "chart."

3. **Updating Data and Chart on an Interval**:
   - **setInterval** is used to repeatedly update sensor data and create a new random chart every 5 seconds (5000 milliseconds). You can adjust the interval to suit your needs.
   - The **updateSensorData** function and **createRandomChart** function are called within the interval to provide updated data and charts.
4. **Calling Update Functions on Page Load**:
   - To ensure that the sensor data and initial chart are displayed when the page loads, the **updateSensorData** and **createRandomChart** functions are called immediately after defining them.
5. **createLineChart Function**:
   - This function is responsible for creating a line chart using the Chart.js library. It is distinct from the **createRandomChart** function.
   - It defines the sample data for the line chart (temperature and humidity values for different months). In a real application, you would replace this with your actual data.
   - The function sets various chart properties, including labels, colors, and scale configurations.
6. **Calling createLineChart on Page Load**:
   - To ensure that the line chart is displayed when the document is ready, the **createLineChart** function is called when the "DOMContentLoaded" event is triggered.

OUTPUT



Public Transport Optimization

Transport Location & Information

Bus Information

Bus Number: 456

Bus Name: SETC

Arrival Time: 2 minutes

Riders on Board: 30

car Information

car Name: BMW

car Number: 233

Arrival Time: 5 minutes

Riders on Board: 25

Bike Information

Bike Name: DUKE

Bike Number: TN AK -3453

Arrival Time: 5 minutes

Riders on Board: 25

Conclusion

The proposed solution involves deploying IoT sensors in public transportation vehicles to continuously collect data, including vehicle location, ridership, and environmental information. This data will be transmitted in real-time to the web-based platform, where it will be processed and displayed to passengers through an intuitive and accessible interface.

This integrated system will significantly enhance public transportation services by providing real-time information, making travel more convenient, efficient, and reliable for passengers. It also has the potential to optimize public transportation schedules, reduce costs, and contribute to environmental sustainability.

In subsequent project phases, we will delve into detailed planning, implementation, testing, and deployment to ensure the successful realization of these improvements in the public transit system.