Phase 4-Public Transport Optimization (Part_2)

Begin by creating a web application aimed at optimizing public transport routes. The foundation involves structuring an HTML document, defining transit information, and creating a visually appealing layout using CSS. Enhance user experience by designing an intuitive interface. A well-designed interface ensures users find the application easy to navigate. Incorporate JavaScript to add interactivity. Implement event listeners to capture form submissions and extract user input effectively. This step involves the real -time transit information.

Step 1: Set Up the HTML Structure

Create an HTML file and set up the basic structure including the necessary HTML elements like **<html>**, **<head>**, and **<body>**. Include the form elements for Sensor1Distance,Sensor 2 Distance, People Entered, People Left, People Inside , and a ** ** to display the transit information.

Program

HTML

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Public Transport Optimization</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <header>
        <h1>Public Transport Optimization</h1>
```

```
    </header>
      <section class="main-content">
      <h2> Real-Time Transit Information</h2>
     <form id="sensorForm">
    <label for="sensor1">Sensor 1 Distance:</label>
    <span id="sensor1Distance">----</span> cm<br>

    <label for="sensor2">Sensor 2 Distance:</label>
    <span id="sensor2Distance">----</span> cm<br>

    <label for="enterCount">People Entered:</label>
    <span id="enterCount">----</span><br>

    <label for="leaveCount">People Left:</label>
    <span id="leaveCount">----</span><br>

    <label for="insideCount">People Inside:</label>
    <span id="insideCount">---</span><br>
</form>
</section>
</body>
</html>
```

Step 2: Add CSS Styling (styles.css)

Create a separate CSS file (styles.css) to style the HTML elements. Add styles to format the header, form, input fields for a visually appealing layout.

CSS

```
<style>
body {
   font-family: Arial, sans-serif;
   margin: 0;
   padding: 0;
}

header {
   background-color: #333;
   color: #fff;
   padding: 10px 0;
   text-align: center;
}

.main-content {
   text-align: center;
   padding: 50px;
}

form {
   margin-bottom: 20px;
}



#Real-Time Transit Information {
   font-size: 18px;
   font-weight: bold;
}
</style>
```

Step 3: Link CSS File

Inside the **<head>** section of your HTML file, link the styles.css file using the **<link>** tag to apply the defined styles to your HTML elements.

Step 4: Write JavaScript Code (script.js)

Create a JavaScript file (script.js) or add the script directly inside **<script>** tags within the **<body>** section of your HTML file. In the script, add event listeners to handle form submission and create the **Real-Time Transit Information** function to process the input values and generate the transit information.

JavaScript

```
<script>

document.getElementById("sensorForm").addEventListener("submit", function(event) {

  event.preventDefault(); // Prevent the form from submitting and reloading the page


  // Make a GET request to Arduino to fetch sensor data

  fetch("/sensorData")

    .then(response => response.json())

    .then(data => {

      // Update the UI with sensor data

      document.getElementById("sensor1Distance").textContent = data.sensor1 + " cm";

      document.getElementById("sensor2Distance").textContent = data.sensor2 + " cm";

      document.getElementById("enterCount").textContent = data.enterCount;

      document.getElementById("leaveCount").textContent = data.leaveCount;

      document.getElementById("insideCount").textContent = data.insideCount;
```

```
    })
    .catch(error => {
      console.error("Error:", error);
    });
});
</script>
```

Step 5: Display Real Time Transit Information

This code will display the real time transit information using IOT sensors. You can use `text Content` or `inner HTML` to update the content of the element with the calculated route.

Step 6: Test and Debug

Test the functionality of the form by entering the Transit Information. Use browser developer tools (console.log) to debug the JavaScript code if you encounter any issues.

Step 7: Refine and Optimize

Refine your code as needed, optimize algorithms for better performance, and enhance the user interface for a better user experience.

Conclusion

The code aims to create a web interface for real-time transit information using sensors, likely for tracking passenger counts and distances in public transport vehicles. and This code could be part of a larger public transport optimization system, where real-time data is collected and analyzed to optimize routes, schedules, and passenger management for public transport vehicles.