**PUBLIC TRANSPORTATION OPTIMIZATION**

**PHASE-5**

**PROJECT OVERVIEW:**

Public transportation optimization using IoT, Raspberry Pi, and sensors involves a comprehensive project that leverages these technologies to enhance the efficiency, safety, and sustainability of public transit systems.

**OBJECTIVES:**

**1. Passenger Information Systems:** Implement displays at bus stops and stations that provide real-time information about when the next bus or tram will arrive, based on the data from the IoT sensors.

**2. Traffic Management:** IoT can be used to communicate with traffic lights and road infrastructure to provide priority to public transportation vehicles, reducing travel time and improving reliability.

:**3.Maintenance Predictions:** Equip vehicles with sensors that monitor their condition, allowing for predictive maintenance to reduce breakdowns and service interruptions.

**4.Smart Ticketing:** Implement contactless payment systems using IoT, making it convenient for passengers to pay fares and reducing the need for physical tickets.

**SENSORS AND RASPBERRY PI:**

**1. Gather Your Components:**

Prepare the required sensors (e.g., GPS, environmental, passenger load) and a Raspberry Pi (e.g., Raspberry Pi 4).

**2. Power Supply:**

Ensure a stable power supply for the Raspberry Pi. You can use a dedicated power source, such as a USB adapter or a power-over-ethernet (PoE) solution.

**3. Raspberry Pi Setup:**

Install an operating system (e.g., Raspberry Pi OS) on the Raspberry Pi's microSD card.Alternatively, you can use SSH for remote Connect the Raspberry Pi to a monitor, keyboard, and mouse for initial configuration. setup.

**4. Connect Sensors:**

Attach the sensors to the Raspberry Pi. Depending on the sensor type, you might need to connect them through GPIO pins, USB ports, or other interfaces.

**5. Install Sensor Drivers:**

If the sensors require specific drivers or software libraries, install them on the Raspberry Pi. This might involve running commands in the terminal or downloading software.

**6. Sensor Calibration:**

Calibrate the sensors as needed to ensure accurate data collection. Follow the manufacturer's instructions for calibration procedures.

**7. Data Communication:**

Set up data communication for the sensors. This could involve configuring serial communication,

I2C, or USB interfaces, depending on the sensor type.

**8. Data Collection and Processing:**

Develop or install software on the Raspberry Pi to collect data from the sensors. This software should process the sensor data, potentially store it locally, and prepare it for transmission.

**9. Data Transmission:**

Decide on the data transmission method (e.g., Wi-Fi, cellular, LoRa) and set up the Raspberry Pi to send data to a central server or cloud platform. You may need to configure network settings and data protocols.

**10. Power Management:**

- Implement power management strategies to ensure efficient power usage, especially if running on battery power.

**11. Testing:**

- Test the sensor-Raspberry Pi setup to verify that it's collecting and transmitting data correctly. Check for data accuracy and ensure the software runs smoothly.

**12. Data Security:**

- Implement security measures to protect the data during transmission and storage. This may involve encryption and access controls.

**13. Data Storage and Analysis:**

- Set up a database or cloud platform to receive, store, and analyze the sensor data.

**14. Remote Access:**

- Configure remote access to the Raspberry Pi for monitoring and maintenance, if needed.

**15. Integration:**

- If this setup is part of a larger system, ensure that the Raspberry Pi integrates seamlessly with other components.

**16. Ongoing Maintenance:**

- Plan for regular maintenance, software updates, and sensor calibration to ensure the system's continued reliability.

**EXAMPLE OUTPUT:**

1. Temperature Sensor (DS18B20):

If you're using a DS18B20 temperature sensor, the output might look like this:

Temperature: 25.5°C

2. PIR Motion Sensor:

For a PIR (Passive Infrared) motion sensor, the output might be a simple detection message:

Motion Detected!

**MOBILE APP DEVELOPMENT :**

For this example, we'll use the Kivy framework to create a basic mobile app in Python:

1. Install kivy:

First, make sure you have Python installed. Then, you can install Kivy using pip:

pip install kivy

2. Create Your App:

Create a Python file for your app, e.g., public\_transport\_app.py

3. Write Python Code:

Use Kivy to build your app's interface and functionality. Here's a basic example:

python

from kivy.app import App

from kivy.uix.label import Label

class PublicTransportApp(App):

def build(self):

return Label(text='Public Transport Optimization App')

if \_name\_ == '\_main\_':

PublicTransportApp().run

In this example, a simple Kivy app displays a label with the text "Public Transport Optimization App."

4. Design and Add Functionality:

Expand your app by designing user interfaces and adding interactive elements using Kivy's layout and widget components.

Implement features like route planning, real-time tracking, and data retrieval from external sources relevant to public transport optimization.

5. Integration with Data Sources:

To optimize public transport, you may need to integrate with APIs or data sources providing information about routes, schedules, real-time vehicle data, and traffic updates.

6. Real-Time Tracking:

Implement real-time tracking of public transport vehicles using location data from GPS or other sources.

5. Integration with Data Sources:

To optimize public transport, you may need to integrate with APIs or data sources providing information about routes, schedules, real-time vehicle data, and traffic updates.

6. Real-Time Tracking:

Implement real-time tracking of public transport vehicles using location data from GPS or other sources.

7. Notifications and Alerts:

Develop functionality to provide users with real-time notifications and alerts about route changes, delays, or service disruptions.

8. Testing and Optimization:

Thoroughly test your app, including functionality, usability, and performance. Optimize the app for a smooth user experience.

9. Deployment:

When you're ready to deploy the app, you can package it for Android and iOS using Kivy's capabilities and other deployment tools for the respective platforms.

**WEB DEVELOPMENT FOR PUBLIC TRANSPORTATION OPTIMIZATION:**



**1. HTML (index.html):**

Create an HTML file to structure your web page:

<!DOCTYPE html>

<html>

<head>

<title>My Simple Website</title>

<link rel="stylesheet" type="text/css" href="styles.css">

</head>

<body>

<header>

<h1>Welcome to My Website</h1>

</header>

<nav>

<ul>

<li><a href="#home">Home</a></li>

<li><a href="#about">About</a></li>

<li><a href="#contact">Contact</a></li>

</ul>

</nav>

<main>

<section id="home">

<h2>Home</h2>

<p>Welcome to our simple website.</p>

</section>

<section id="about">

<h2>About Us</h2>

<p>We are a small team passionate about web development.</p>

</section>

<section id="contact">

<h2>Contact Us</h2>

<p>You can reach us at example@email.com.</p>

</section>

</main>

<footer>

<p>&copy; 2023 My Simple Website</p>

</footer>

<script src="script.js"></script>

</body>

</html>

**2. CSS (styles.css):**

Create a CSS file to style your web page:

/\* Reset default margin and padding \*/

body, h1, h2, p {

margin: 0;

padding: 0;

}

/\* Style header \*/

header {

background-color: #333;

color: #fff;

text-align: center;

padding: 10px;

}

/\* Style navigation menu \*/

nav {

background-color: #444;

padding: 10px;

}

nav ul {

list-style: none;

}

nav ul li {

display: inline;

margin-right: 10px;

}

nav ul li a {

text-decoration: none;

color: #fff;

}

/\* Style main content \*/

main {

padding: 20px;

}

/\* Style footer \*/

footer {

background-color: #333;

color: #fff;

text-align: center;

padding: 10px;

}

**3. JavaScript (script.js):**

You can add interactive features to your web page using JavaScript. Here's a simple example that changes the header text when the page loads:

// Change the header text

document.addEventListener('DOMContentLoaded', function() {

var header = document.querySelector('header h1');

header.textContent = "Welcome to My Simple Website";

});

**PYTHON SERVER CODE:**



Setting up a Python server involves creating a web server using Python. You can use various libraries and frameworks to achieve this, but one of the simplest methods is using Python's built-in http.server module. Here are the steps to set up a basic Python server:

**1. Open a Command Line Terminal:**

You can use the command prompt (Windows), Terminal (macOS and Linux), or any other terminal application you prefer.

**2. Navigate to the Directory:**

Use the cd command to navigate to the directory where your web files are located. For example:

cd /path/to/your/website

Start the Python Server:

Run the following command to start the Python server:

For Python 2.x:

python -m SimpleHTTPServer

python -m http.server

This will start a web server on port 8000 by default.

**4. Access Your Website:**

Open a web browser and navigate to the following address:

<http://localhost:8000/>

You should be able to access your website files from this address. If you want to use a different port, specify it when running the Python server. For example:

python -m http.server 8080

This will start the server on port 8080, and you can access your website at <http://localhost:8080/>.

* **Improved Efficiency:** By collecting real-time data on vehicle locations, passenger load, and environmental conditions, transportation authorities can optimize routes, schedules, and resource allocation. This leads to reduced wait times, shorter travel durations, and less congestion.
* **Enhanced Passenger Experience:** Passengers benefit from accurate real-time information about public transport, such as arrival times, crowdedness, and service updates. This improves overall passenger satisfaction.
* **Energy Efficiency:** Optimizing routes and load distribution can lead to reduced fuel consumption and greenhouse gas emissions, contributing to a greener and more sustainable transportation system.
* **Cost Savings:** Efficiency improvements can reduce operational costs for public transportation agencies. Lower fuel consumption and maintenance costs contribute to budget savings.
* **Safety:** Sensors and IoT technology can be used for safety enhancements, including collision avoidance, monitoring for vehicle defects, and ensuring passenger safety during boarding and transit.
* **Traffic Management:** Traffic flow can be managed more effectively by integrating public transport data with traffic monitoring systems. This can lead to less congestion on roadways.
* **Data-Driven Decision-Making:** Public transportation optimization projects generate valuable data that can be used for future planning and decision-making. This data can inform infrastructure investments and service expansions.
* **Environmental Impact:** Reduced fuel consumption and emissions not only save costs but also contribute to a cleaner and healthier environment.
* **Emergency Response:**Real-time tracking and communication can assist in emergency response situations, helping authorities respond quickly and efficiently to incidents.

**CONCLUSION:**



A public transportation optimization project utilizing IoT, Raspberry Pi, and sensors offers a wide range of benefits. It enhances the efficiency and reliability of public transportation, leading to reduced waiting times, improved passenger experiences, and cost savings for transportation authorities. Additionally, it contributes to energy efficiency and a reduction in greenhouse gas emissions, promoting environmental sustainability. The project's impact extends to improved traffic management, safety enhancements, and the collection of valuable data for informed decision-making.

**SUBMITTED BY;**

**SANTHOSH KUMAR G**

**(311421106050)**