**Public transportation vehicles to monitor ridership, track locations, and predict arrival times.**

**PHASE 1:SUBMISSION DOCUMENT**

**ABSTRACT:**

Monitoring ridership, tracking locations, and predicting arrival times in public transportation vehicles have become essential for enhancing the efficiency and user experience of urban transit systems**.**

**PROJECT OBJECTIVES:**

* Real-time parking space monitoring
* Mobile app integration
* Efficent parking guidance

**REAL-TIME PARKING SPACE MONITORING:**

Real-time parking space monitoring involves the use of technology to track and manage the availability of parking spaces in real-time. Here's an overview of how this is achieved.sensors and cameras,datacollection,datatransmission,dataprocessing, information display,payment integration,alerts and notification, analytics and optimization.

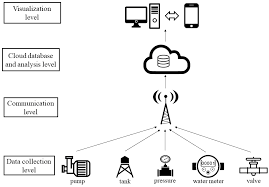
**MOBILE APP INTEGRATION:**

Mobile app integration is the process of connecting a mobile application with other systems, services, or platforms to enhance its functionality, features, and user experience. Here are some common aspects of mobile app integration.API integration,payment gateway integration.

**EFFICIENT PARKING GUIDANCE:**

Efficient parking guidance systems are designed to help drivers find available parking spaces quickly and reduce congestion in parking facilities. Here are key elements and technologies used to achieve efficient parking guidance.

**DESIGING THE IOT SENSOR SYSTEM**:



Designing an IoT sensor system involves careful planning and consideration of various components and aspects to ensure it meets the intended objectives. Here's an outline of the steps involved:

**1**. **\*Define Objectives\*:**

- Determine the purpose of the IoT sensor system. What data do you need to collect? What problems will it address?

**2. \*Select Sensors\*:**

- Choose appropriate sensors based on your objectives. Consider factors like sensor type (e.g., temperature, humidity, motion), accuracy, and communication protocols (e.g., Wi-Fi, Bluetooth, LoRa, Zigbee).

**3. \*Power Source\*:**

- Decide on the power source. It can be battery-powered, solar-powered, or connected to a reliable power grid.

**Real-time transit information platform:**

Developing a real-time transit information platform requires careful planning, integration of various technologies, and a user-centered approach. Here are the steps involved in creating such a platform:

**1. \*Define Objectives\*:**

- Determine the primary goals of your transit information platform. What are you trying to achieve? Improved user experience, reduced wait times, increased efficiency?

**2. \*Data Sources\*:**

- Identify the data sources needed, such as GPS data from vehicles, traffic data, schedules, and occupancy information. Establish partnerships with transit agencies to access their data.

**3. \*Data Integration**\*:

- Create data pipelines to collect, integrate, and process real-time data from various sources. Use ETL (Extract, Transform, Load) processes to ensure data quality.

**Integrating APPROCH:**

To use a Raspberry Pi to collect data from sensors and update a mobile app, you'll need to set up a system that involves sensor data acquisition, data processing, and communication between the Raspberry Pi and the mobile app. Here's a step-by-step guide:

**1. \*Select Sensors**\*:

- Choose the appropriate sensors for your project. Depending on your requirements, this could include temperature sensors, motion sensors, humidity sensors, etc.

**2. \*Connect Sensors to Raspberry Pi\*:**

- Connect the selected sensors to the GPIO pins of the Raspberry Pi. Make sure to follow the wiring and power requirements for each sensor.

**3. \*Install Necessary Libraries\*:**

- Install the required libraries and drivers for the sensors on the Raspberry Pi. This may involve using Python libraries, C libraries, or other programming languages depending on the sensor.

**\*Algorithm for Smart Parking:\***

STEP1:\*Initialization\*:

- Initialize the system, including the parking lot layout, sensor data, and available parking spaces.

STEP2: \*Sensor Data Collection\*:

- Continuously collect data from sensors installed at each parking space to detect occupancy (occupied or vacant).

STEP3:\*User Request\*:

- When a user requests parking, either through a mobile app or on-site device:

- Identify the user's location and desired parking duration.

- Check the availability of nearby parking spaces.

STEP4: \*Search for Available Space\*:

- Iterate through the parking spaces to find the nearest available parking spot that meets the user's criteria.

- Consider proximity to the user's current location.

- Ensure the parking space is vacant.

- Check if it meets the user's desired duration.

STEP5:\*Reserve Space\*:

- Reserve the selected parking space for the user, marking it as temporarily occupied. This prevents others from occupying it during the reservation period.

STEP6: \*User Confirmation\*:

- Notify the user of the reserved parking space, including its location, availability, and any relevant details.

STEP7: \*Parking Guidance\*:

- Provide turn-by-turn directions to the reserved parking spot if the user is on-site.

STEP8: \*User Arrival\*:

- Confirm the user's arrival at the parking spot using location data, a QR code scan, or other methods.

STEP9: \*Occupancy Update\*:

- Mark the parking space as occupied in the system.

STEP10:\*Billing and Payment\*:

- Calculate the parking fee based on the duration and rate. Charge the user through a payment gateway integrated into the system.

STEP11: \*Real-time Updates\*:

- Continuously update the parking availability information on mobile apps and digital displays to inform other users.

STEP12: \*End of Parking\*:

- Monitor the user's parking duration, and when it expires, mark the parking space as vacant again.

STEP13:\*Feedback and Reviews\*:

- Allow users to provide feedback or reviews about their parking experience, contributing to system improvement.

STEP14:\*System Optimization\*:

- Use historical data to optimize the allocation of parking spaces, predict high-demand periods, and improve the overall efficiency of the system.

STEP15:\*Maintenance and Alerts\*:

- Implement alerts for system maintenance, sensor malfunction, or other issues that require attention.

**PROGRAM :**

# Define the parking lot layout (for demonstration purposes)

total\_spaces = 20

parking\_spaces = [False] \* total\_spaces # False represents a vacant space

def display\_parking\_lot():

print("Parking Lot Status:")

for i, space in enumerate(parking\_spaces, 1):

status = "Occupied" if space else "Vacant"

print(f"Space {i}: {status}")

def find\_available\_space():

for i, space in enumerate(parking\_spaces):

if not space:

return i

return -1

def reserve\_parking\_space(space\_number):

if parking\_spaces[space\_number]:

print("Space is already occupied.")

return False

else:

parking\_spaces[space\_number] = True

print(f"Space {space\_number + 1} reserved.")

return True

def release\_parking\_space(space\_number):

parking\_spaces[space\_number] = False

print(f"Space {space\_number + 1} released.")

# Simulation

while True:

display\_parking\_lot()

action = input("Select an action (1: Reserve, 2: Release, 0: Exit): ")

if action == '0':

break

elif action == '1':

space\_number = find\_available\_space()

if space\_number != -1:

reserve\_parking\_space(space\_number)

else:

print("No available parking spaces.")

elif action == '2':

space\_number = int(input("Enter the space number to release: ")) - 1

if 0 <= space\_number < total\_spaces:

release\_parking\_space(space\_number)

else:

print("Invalid space number.")

print("Smart Parking System Exiting.")

**CONCLUSION:**

In conclusion, smart parking systems offer a promising solution to the challenges of urban congestion and parking management. These systems leverage technology to optimize parking space allocation, improve the user experience, and contribute to more efficient and sustainable urban mobility.

GIT UP LINK:

https://github.com/prabhakararaj20/prabhakararaj.git